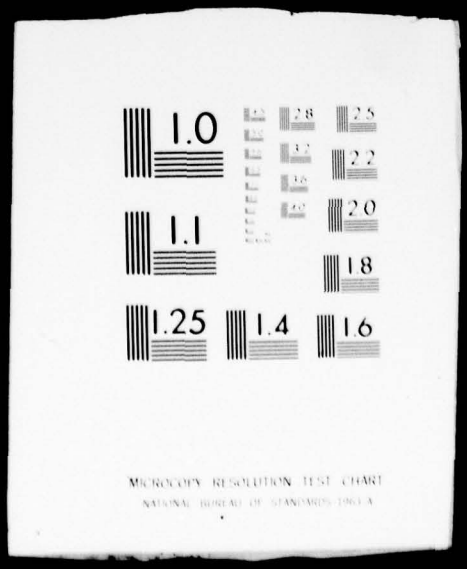
AD-A074 053 INTERNATIONAL COMPUTING CO BETHESDA MD F/G 9/2
VTS PROCESSING DISPLAY SUBSYSTEM DESIGN.(U)
JAN 79 C C HENSON, F T MICKEY, R S GRAHAM DOT-CG-81-78-1833
UNCLASSIFIED USCG-D-54-79 NL

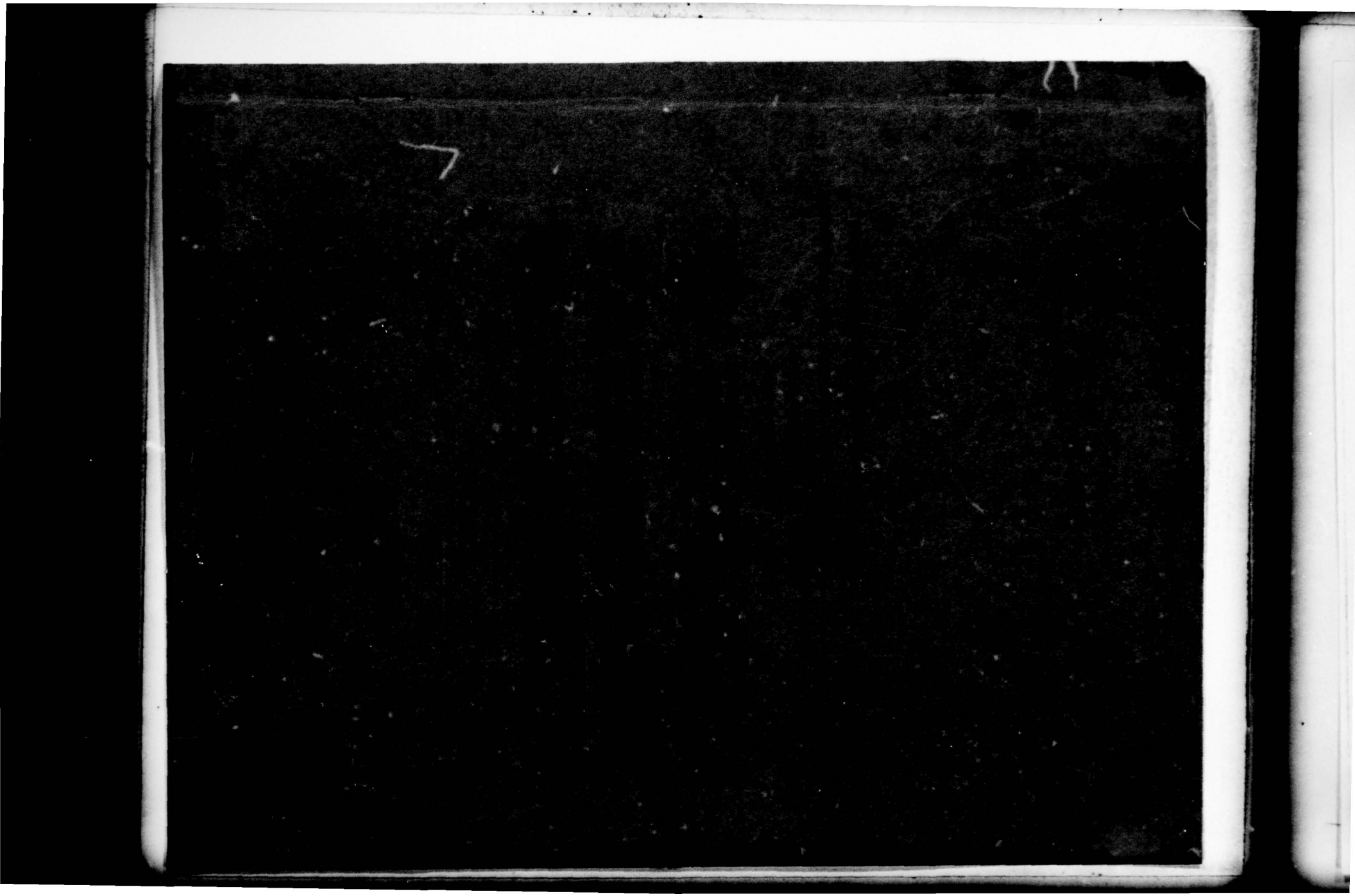
1 OF 4
AD A074053

1 OF 4

AD A074053



AD A074053



(18) USCG, CGR/DC

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
CG-D-54-79 / 13/79		

4. Title and Subtitle (6)
VTS Processing Display Subsystem Design

5. Report Date (11)
January, 1979

6. Performing Organization Code

8. Performing Organization Report No.

7. Author(s) (10)
C. C. Henson, F. T. Mickey, R. S. Graham, B. A. McIntosh

9. Performing Organization Name and Address
International Computing Company
4330 East West Highway
Bethesda, Maryland 20014

10. Work Unit No. (TRAIS) (13)

11. Contract or Grant No.
DOT-CG-81-78-1833

13. Type of Report and Period Covered (9)
Final Report,
April 1978 - January 1979

12. Sponsoring Agency Name and Address (12) 409 p.
U. S. Department of Transportation
United States Coast Guard
Office of Research and Development
Washington, D.C. 20590

14. Sponsoring Agency Code

15. Supplementary Notes
The contract under which this report was submitted was under the Technical Supervision of the Coast Guard Research and Development Center, Groton, Connecticut, 06340. R&D Center report number 13/79 has been assigned.

16. Abstract

A distributed function, multiple bus coupled minicomputer system architecture will be used for the Vessel Traffic Services (VTS) Processing/Display Subsystem. Four critical issues are examined including: Reliability; Programming Language Selection; Interprocessor Bus Design; and Multicomputer Operating System Design. System availability requirements can be met with partial redundancy with software controlled reconfiguration and recovery based on both hardware and software error and failure detection. PASCAL is the best programming language available for VTS. A relatively low cost interprocessor bus capable of transmitting at 1.5 megabits per second can be built using off-the-shelf LSI components. The VTS operating system will feature explicit message and answer communications between processes for interprocess communication and synchronization. Error detection and reporting and system reconfiguration and recovery will be controlled by the operating system. The top level of the software design is structured as a group of cooperating sequential processes which use the operating system facility for interprocess communication and synchronization, providing flexibility in assigning processes to processors. Hardware configurations are developed for three possible combinations of functional and traffic handling capabilities. Hardware requirements are included for processors, buses, discs, tapes, graphics units, CRTs, printers, and keyboards.

17. Key Words	18. Distribution Statement
Vessel Traffic Services (VTS) Distributed Processing Interprocessor Bus Operating Systems Reliability	Document is available to the U. S. Public through the National Technical Information Service, Springfield, Virginia, 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	302	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

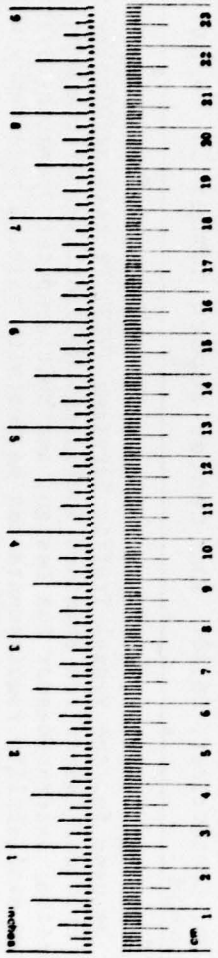
i 393469

METRIC CONVERSION FACTORS

Approximate Conversions to Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
in	inches	*2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (weight)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
Tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.96	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (exact)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

*1 in = 2.54 (exactly). For other exact conversions and more detailed tables, see NBS Misc. Publ. 286, Units of Weights and Measures, Price $2.25, SD Catalog No. C13.10:286.



Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares (10,000 m^2)	2.5	acres	
		MASS (weight)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (exact)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A

CONTENTS

Section		Page
EXECUTIVE SUMMARY		viii
1	INTRODUCTION	1-1
2	OVERVIEW OF VTS PROCESSING/DISPLAY SUBSYSTEM REQUIREMENTS	2-1
2.1	Purpose	2-1
2.2	Applicable Documents	2-2
2.3	VTC Functions and Organization	2-3
2.4	VTS System Modes, Classes and Sensor Levels	2-5
2.5	VTS System Design Goals	2-8
2.6	VTS System Data Base	2-10
2.7	Processing Capabilities	2-12
2.8	VTS System Architecture	2-13
3	CRITICAL ISSUES	3-1
3.1	Reliability	3-2
3.2	Language Selection	3-3
3.3	Bus Design	3-4
4	RELIABILITY	4-1
4.1	Software Reliability	4-1
4.2	Hardware Reliability	4-2
4.3	Reliability Calculations	4-3
4.4	Considering the Assumptions	4-7
4.4.1	Fault Detection	4-7
4.4.2	Reconfiguration	4-7
4.4.3	Independence	4-8
4.5	Designing a Reliable System	4-9

CONTENTS

Section		Page
4.5.1	Fault Detection	4-9
4.5.2	Error Reporting	4-10
4.5.3	Reconfiguration	4-10
4.5.4	Independence	4-11
4.5.5	Maintenance Policy	4-11
4.6	Another Look at System Reliability	4-13
4.6.1	Availability	4-13
4.6.2	Failsoft	4-13
5	LANGUAGE SELECTION	5-1
5.1	Criteria for the Language Selection	5-2
5.1.1	Suitability for All VTS Programming	5-3
5.1.2	Control Structures	5-3
5.1.3	Data Structuring	5-6
5.1.4	East of Use	5-8
5.1.5	Efficiency	5-9
5.1.6	Error Detection	5-9
5.1.7	Readability	5-11
5.1.8	Availability	5-12
5.2	Candidate Languages	5-13
5.2.1	ALGOL	5-14
5.2.2	BASIC	5-14
5.2.3	COBOL	5-15
5.2.4	FORTRAN	5-15
5.2.5	PASCAL	5-16
5.2.6	PL/1	5-17
5.2.7	Department of Defense Common High Order Language	5-18
5.3	Conclusion	5-21

CONTENTS

Section		Page
6	VTS BUS AND INTERFACES	6-1
6.1	General	6-1
6.2	Alternative Bus Structures	6-2
6.2.1	Bus Control Transfer	6-2
6.2.2	Bus Data Path	6-7
6.3	Bus Requirements	6-10
6.3.1	System Considerations	6-10
6.3.2	Bus Transmission Medium and Interface Connection	6-11
6.3.3	Bus Interface	6-12
6.3.4	Hardware/Software Interaction	6-14
6.4	Target Bus Design	6-16
6.4.1	Normal Operation	6-18
6.4.2	Bus Transmission Protocol	6-19
6.4.3	Error Detection and Resolution	6-22
6.4.4	Bus Characteristics	6-23
6.4.5	Interface Reliability and Flexibility	6-23
7	THE VTS OPERATING SYSTEM	7-1
7.1	Design Goals	7-2
7.2	Process Managemenet	7-5
7.2.1	Process States	7-5
7.2.2	Scheduler	7-7
7.2.3	Process Management Functions	7-8
7.3	Interprocess Communication	7-10
7.3.1	Interprocess Messages and Answers	7-11
7.3.2	Software for Interprocess Communications	7-12
7.4	Resource Allocation	7-14

CONTENTS

Section		Page
7.4.1	Deadlocks	7-14
7.4.2	User Controlled Rosources	7-19
7.5	File Management	7-20
7.6	I/O Control	7-22
7.7	Error Detection, Reconfiguration and Recovery	7-24
7.8	Other Operating System Functions	7-26
8	PRELIMINARY SOFTWARE DESIGN	8-1
8.1	Types of Processes	8-2
8.2	Overview of Software Design	8-4
8.2.1	Display Station Processes	8-4
8.2.2	Processes in the Main Processor	8-6
8.3	Display Station Processes	8-7
8.3.1	Transact Watchstander Request	8-7
8.3.2	Manage Alert Display	8-9
8.3.3	Manage Map	8-9
8.4	Processes in the Main Processor	8-10
8.4.1	Access Files	8-10
8.4.2	Demand Functions	8-14
8.4.3	Position Processing	8-17
8.4.4	Detect Hazards	8-19
8.4.5	Post Alerts	8-29
8.4.6	Logging Management	8-29
8.4.7	Manage Environmental Sensors	8-30
8.4.8	Playback Simulation Scenario	8-30
8.5	Data Bases	8-31
8.5.1	Data Base Design Concepts	8-31
8.5.2	Access Methods	8-32

CONTENTS

Section		Page
9	HARDWARE DESIGN	9-1
9.1	Processing and Memory Requirements	9-2
9.1.1	Class C, Level 4 System	9-2
9.1.2	Class B, Level 4 System	9-6
9.1.3	Class A, Level 1 System	9-8
9.2	Disc Sizing and Utilization	9-10
9.2.1	Disc Access Frequencies	9-11
9.3	Hardware Specifications	9-17
9.3.1	Processors	9-17
9.3.2	Disc Units	9-19
9.3.3	Magnetic Tape Transports	9-19
9.3.4	Line Printers	9-19
9.3.5	System Consoles	9-20
9.3.6	Display Station Peripherals	9-20
9.3.7	Configuration of Hardware Components	9-23
10	ANALYSIS AND EVALUATION OF THE DESIGN	10-1
10.1	Average Service Times	10-2
10.2	Function Response Times	10-27
10.2.1	Enter Vessel	10-27
10.2.2	Enter Passage	10-34
10.2.3	Search on a Key	10-37
APPENDIX A	SAMPLE PROCESSING POWER EVALUATION ROUTINE	A-1
APPENDIX B	OPERATING SYSTEM FUNCTIONS	B-1
APPENDIX C	INTERPROCESS COMMUNICATION SYSTEM	C-1
APPENDIX D	COMMUNICATION ERROR DETECTION AND RECOVERY	D-1

EXECUTIVE SUMMARY

Introduction

Vessel Traffic Services (VTS) Processing/Display Subsystem will assist in gathering, retrieving and correlating information as a service to the maritime community. The VTS Processing/Display Subsystem, as a part of a VTS system, will have as its objective a reduction in the probability of environmentally damaging accidents in harbor areas.

Information such as the current location, course and speed of vessels in a harbor area will provide a basis for traffic management and will be used to anticipate hazardous conditions such as potential collisions and grounding.

Computerized systems coupled with automated sensors and communications equipment will be required to enable the U. S. Coast Guard to provide these services. Requirements for computers and related hardware will vary dramatically from port to port, however.

To provide the flexibility and modularity need for the VTS Processing/Display Subsystem, a flexible and modular system architecture is required. A functionally distributed, bus coupled, multiple minicomputer architecture has been selected from a number of alternative architectures*.

Figure 1 shows the selected architecture configured to handle the maximum anticipated functional and throughput requirements. The configuration can be reduced easily to handle less vessel traffic or to provide a reduced functional capability which will be adequate for a number of ports.

*Henson, C. C., Cleaver, R. A., Kaisler, S. H., "Preliminary Design Study for VTS Processing/Display Subsystem", June, 1978.

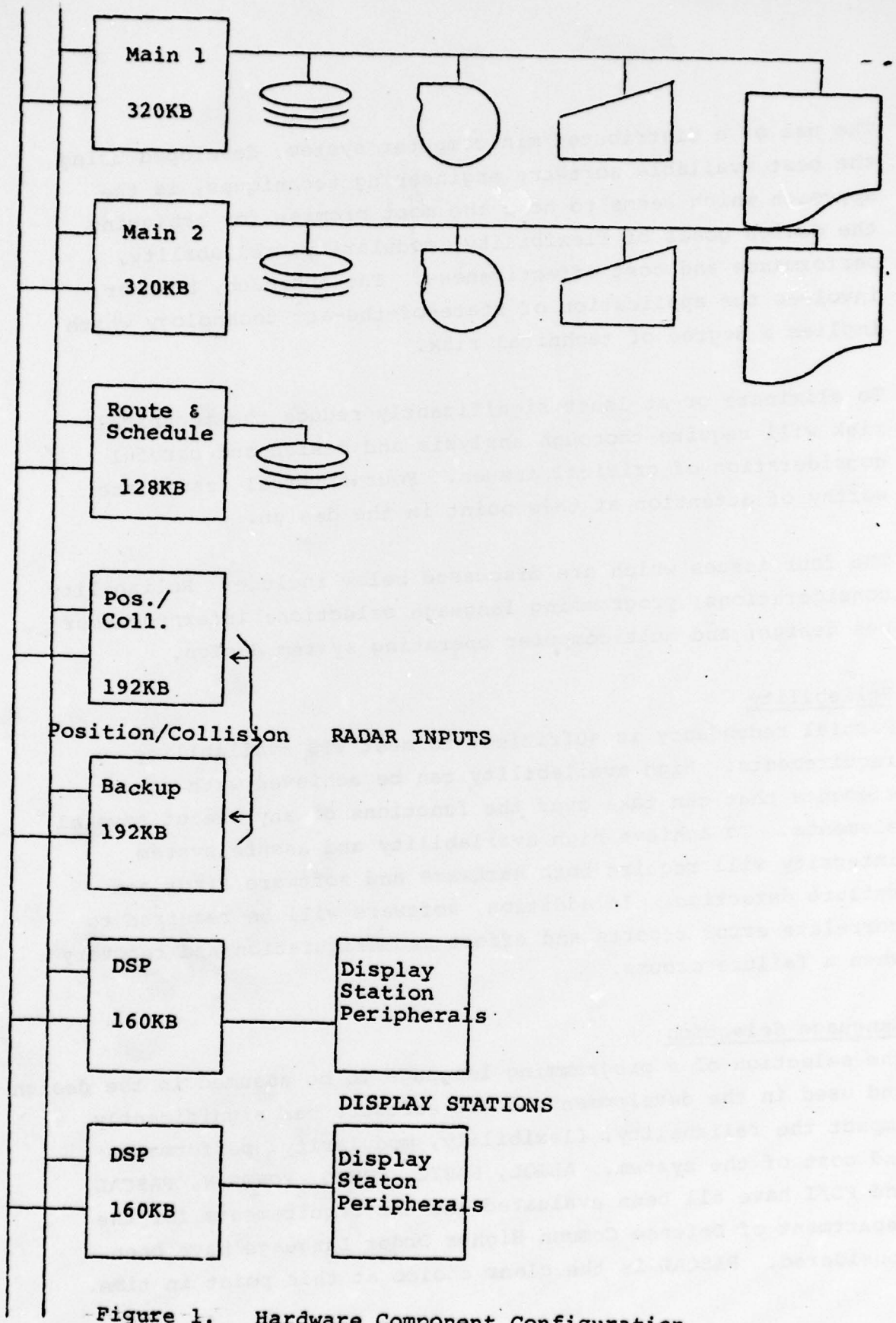


Figure 1. Hardware Component Configuration for Class C, Level 4 System

The use of a distributed minicomputer system, developed using the best available software engineering techniques, is the approach which seems to hold the most promise for achieving the design goals of flexibility, modularity, reliability, performance and cost effectiveness. The approach, however, involves the application of state-of-the-art technology which implies a degree of technical risk.

To eliminate or at least significantly reduce the technical risk will require thorough analysis and design and careful consideration of critical issues. Four critical issues are worthy of attention at this point in the design.

The four issues which are discussed below include: Reliability considerations; programming language selection; interprocessor bus design; and multicomputer operating system design.

Reliability

Partial redundancy is sufficient to meet VTS availability requirements. High availability can be achieved with spare elements that can take over the functions of any one of several elements. To achieve high availability and assure system integrity will require both hardware and software error and failure detection. In addition, software will be required to correlate error reports and effect reconfiguration and recovery when a failure occurs.

Language Selection

The selection of a programming language to be assumed in the design and used in the development of VTS software can significantly impact the reliability, flexibility, modularity, performance and cost of the system. ALGOL, BASIC, COBOL, FORTRAN, PASCAL, and PL/I have all been evaluated and the requirements for the Department of Defense Common Higher Order Language have been considered. PASCAL is the clear choice at this point in time.

Interprocessor Bus Design

A relatively low cost interprocessor bus can be built using LSI technology. Using microprocessor and communications circuits which are available off-the-shelf, a preliminary design has been developed for a bus capable of serial transmission at 1.5 million bits per second. Distributed polling can be used to pass control of the bus from processor to processor.

Other bus designs could be used for VTS. Specifications for the shared busses are included with other hardware specifications below.

Multicomputer Operating Systems

The VTS Operating System (VTS/OS) must provide the facilities normally found in a real time operating system including: process management; interprocess communications; resource allocation; file management; and I/O control. VTS/OS will be somewhat unique, however, since it must operate in a multi-computer environment and must provide extensive error detection and reconfiguration control needed for VTS.

A preliminary design for VTS/OS has been completed. VTS/OS will include a flexible interprocess communication facility that will allow applications software to be structured as a group of cooperating sequential processes that communicate information and coordinate their activities by explicit messages and answers.

Error and failure detection mechanisms will be included throughout the operating system. Errors and failures which are detected will be reported to a process in each processor called a Local Error Reporting Center (LERC). The LERCs will transmit error reports (normally over both buses) to a process called the Main Error Reporting Center (MERC) which will be active in only one processor. The MERC will correlate error reports and control reconfiguration and recovery when required.

Software Design

The top level of the software design is shown in Figure 2. At the top level will be a group of processes which communicate with each other and synchronize their operations by using the VTS/OS interprocess communication facility.

Processes will be assigned to two functional types of processor in a typical VTS configuration. Display station processors will interface with Watchstanders and Watch Supervisors. Three major processes will operate in each display station processor. One process will interface with the watchstander and will be called Transact Watchstander Request. Two other processes will manage the map display and the alert display.

A variety of processes will operate in the main processor or its backup. Processes which access files or perform demand functions will primarily operate in response to requests from processes in the display station processors. Other processes will receive and process inputs from environmental and position sensors. Hazard detection processes will operate on a time cycle to periodically check for potential hazards.

Hardware Design

The preliminary top level software design, with estimates of the resource requirements of the processes, can be used to develop the hardware configurations and device characteristics needed for VTS Processing/Display Subsystems.

The configurations will vary from port to port as a function of the anticipated vessel traffic and the specific functional capabilities required. The required number of hardware building blocks such as processors, discs, or graphics units will vary. The requirements for the individual building blocks should be the same for all sites. A checklist of hardware requirements is included in Figure 3.

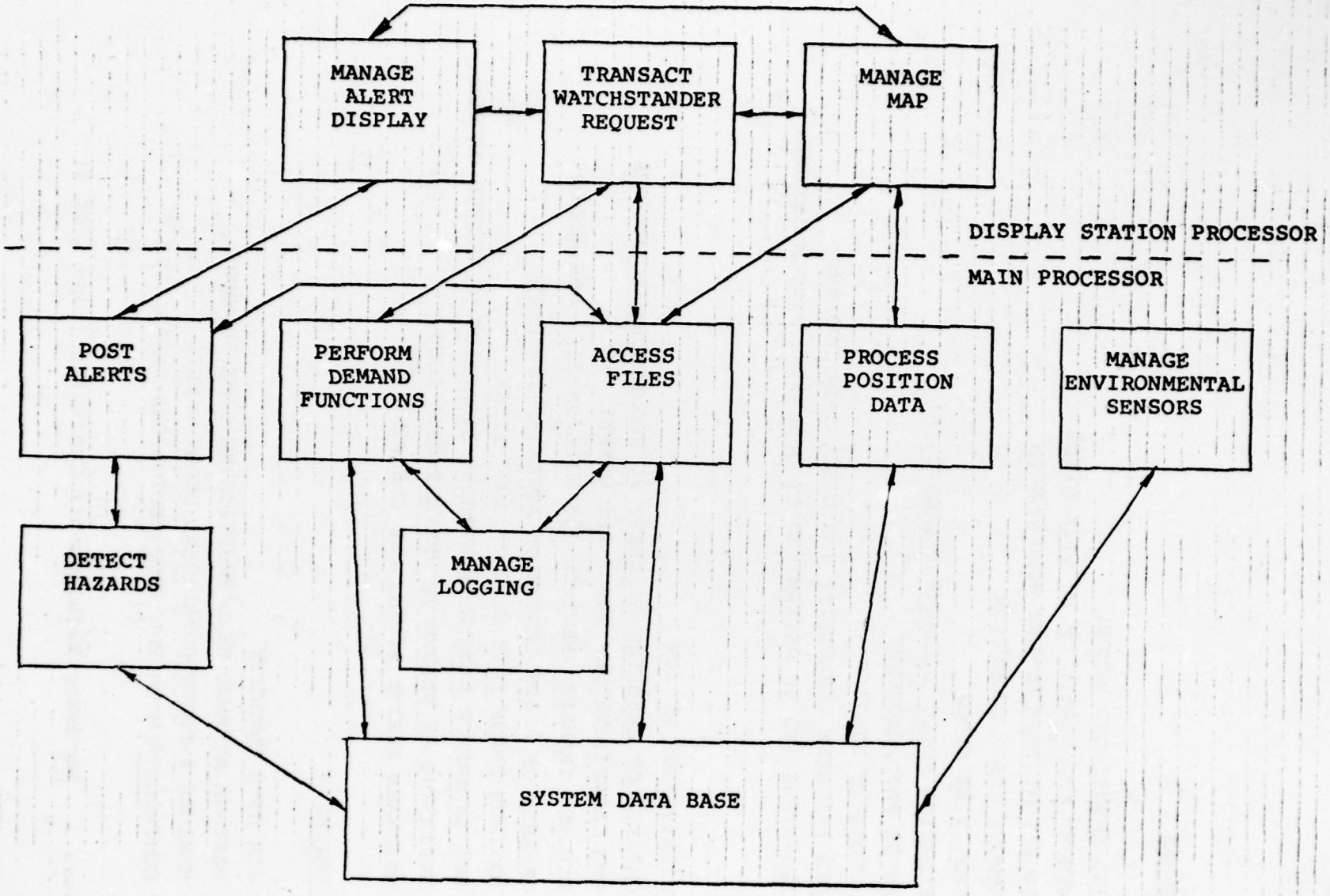


Figure 2. Major Groups of Processes

PROCESSORS

- 800 NANOSECOND MEMORY
- MEMORY EXPANDABLE UP TO 512K BYTES
- 16 BIT MINIMUM MEMORY WORD LENGTH
- MEMORY ERROR DETECTION
- REAL TIME CLOCK
- FLOATING POINT HARDWARE (CLASS C, B)
- POWER FAIL/RECOVERY
- AUTOMATIC LOAD
- DMA TRANSFER CAPABILITY FOR DISC, TAPE AND BUS
- REGISTERS (TO BE DETERMINED BY OPERATING SYSTEM DESIGN)

BUS

- PASSIVE BUS MEDIUM
- CAPABLE OF HANDLING AT LEAST 20 PROCESSORS ON ONE BUS
- DMA CAPABILITY FOR INPUT AND OUTPUT
- MINIMUM TRANSFER RATE OF 1.5 MILLION BITS PER SECOND
- CAPABLE OF A BUS LENGTH OF AT LEAST 200 FEET
- MAXIMUM ERROR RATE OF 1 IN 10^{10} BITS
- BUS INTERFACE REMOVABLE WITHOUT DISABLING PROCESSOR OR BUS
- BUFFERING AS NEEDED TO PREVENT DATA LOSS
- HARDWARE PARITY AND/OR CRC CHECKING

DISC DRIVES

- 80M BYTE CAPACITY
- MAXIMUM AVERAGE HEAD POSITIONING TIME 30 MSEC
- MAXIMUM AVERAGE ROTATIONAL DELAY 9 MSEC
- CONTROLLER WITH MULTIPLE SECTOR DMA CAPABILITY

Figure 3. Hardware Requirements Checklist (Page 1 of 3)

MAGNETIC TAPE UNITS

- IBM-COMPATIBLE
- 9 TRACK
- RECORDING DENSITY 800 OR 1600 BPI
- 45 IPS MINIMUM RECORDING SPEED
- READ-AFTER-WRITE
- DMA TRANSFER CAPABILITY
- 10.5 INCH REEL CAPACITY

MAP DISPLAY (GRAPHICS) UNITS

- 211 SQUARE INCH VIEWABLE SCREEN AREA
- 1024 BY 1024 POINT RESOLUTION
- FLICKER FREE DISPLAY OF 10,000 VECTORS
- BLINK FEATURE
- COLOR CAPABILITY (OPTION)
- X, Y POSITIONING DEVICE
- ZOOM

ALPHANUMERIC DISPLAY/KEYBOARD

- 24 LINE BY 80 COLUMN DISPLAY SCREEN
- 96 ASCII CHARACTER SET - UPPER AND LOWER CASE
- PAGING - STORAGE OF 5 PAGES MINIMUM
- SCROLLING
- DISPLAY RATE 1200 BPS to 9600 BPS
- KEYBOARD SECTIONS - TYPEWRITER, NUMERIC, EDITING, CURSOR

SPECIAL FUNCTION KEYBOARD

- KEYS FOR INITIATING TRAFFIC COORDINATOR FUNCTIONS
- 24 to 32 KEYS
- CUSTOM DESIGNED
- EIA RS-232C INTERFACE

Figure 3. Hardware Requirements Checklist (Page 2 of 3)

ALERT DISPLAY

- 22 LINES BY 40 COLUMNS OF TEXT
- 96 ASCII CHARACTER SET
- EIA RS-232C INTERFACE
- DISPLAY RATE 1200 to 9600 BPS

DISPLAY STATION PRINTER

- NON-IMPACT
- 30 CPS

LINE PRINTERS

- 600 LINES/MINUTE
- 96 ASCII CHARACTERS - UPPER & LOWER CASE
- 132 COLUMNS
- TRACTOR FEED
- 6 LINES PER INCH

SYSTEM CONSOLES (KEYBOARD & HARDCOPY PRINTER)

- 10 TO 30 CPS
- 72 COLUMNS
- EIA RS-232C OR CURRENT LOOP INTERFACE
- TYPEWRITER (QWERTY TYPE) KEYBOARD
- 64 - 96 ASCII CHARACTERS

Figure 3. Hardware Requirements Checklist (Page 3 of 3)

1 INTRODUCTION

This is the second in a series of three reports prepared by International Computing Company (ICC) under Contract No. DOT-CG-81-78-1833, for the United States Coast Guard. This second report presents the results of the design study for the Vessel Traffic Services (VTS) Processing Display Subsystem.

An initial design for this system was described in the first report, which covered the preliminary design study[1]. This second report carries the design to a greater level of detail and discusses some of the design issues which must be resolved in the design of a distributed minicomputer system.

This second report discusses software requirements and software design to the level of detail necessary to develop most of the hardware requirements. It is not intended that this report cover the software exhaustively since additional software design will be included in subsequent phases of this effort.

This second report will be followed by a third report which will provide additional detail concerning the software requirements and software design.

This second report provides first an overview of the requirements for the VTS Processing/Display Subsystem. We then deal with some of the critical system design issues which include:

- Reliability issues;
- Selection of a Programming Language for use in the Development of VTS Software;

[1] Henson, C. C., Cleaver, R. A., Kaisler, S. H., "Preliminary Design Study for VTS Processing/Display Subsystem", June, 1978.

- Specification of the computer interconnection device and a suggested design for a serial bus;

- Preliminary design of an operating system for use in VTS.

Following the discussion of these critical design issues, the current hardware and software design is presented. An analysis of system response times and a discussion of growth potential and possible limiting factors concludes the report.

2 OVERVIEW OF VTS PROCESSING/DISPLAY SUBSYSTEM REQUIREMENTS

2.1 PURPOSE

The fundamental purpose of the VTS system is to reduce the number of accidents which occur in harbors by providing information with which vessel traffic may be managed, and by providing usable and reliable information on potential vessel hazards in a timely fashion.

Information on the current location, course and speed of vessels in a harbor area can be used to anticipate hazardous conditions such as potential collisions or grounding. Such information can also predict conditions such as excessive congestion which can lead to hazardous conditions.

The VTS Processing/Display Subsystem will assist in gathering, retrieving, and correlating available information as a service to the maritime community. Computerized systems coupled with automated sensors and communications equipment will be required to enable the U. S. Coast Guard to provide these services.

2.2 APPLICABLE DOCUMENTS

A functional description has been developed by the U. S. Coast Guard[1]. In this section, a brief overview of the system will be presented. For additional information, the reader is referred to the Functional Description and to the Preliminary Design Study report[2] which documents the preceeding work on the design of the VTS Processing/Display Subsystem.

The preliminary design study dealt with the selection of a system architecture for VTS. Following the overview of the VTS functions, requirements and design goals, a brief description is given of the architecture selected for VTS. The VTS architecture will be discussed in much greater detail in subsequent chapters.

---

[1] Appendix 8, Request for Proposal No. 81-77-1833, U. S. Coast Guard Academy, New London, Connecticut.

[2] Henson, C. C., Cleaver, R. A., Kaisler, S. H., "Preliminary Design Study for VTS Processing/Display Subsystem", June 1978.

2.3 VTC FUNCTIONS AND ORGANIZATION

A Vessel Traffic Center (VTC) has some or all of the following five major functions:

- Detect and report potentially hazardous situations, such as imminent collisions, groundings, and excessive traffic congestion;
- Schedule and/or route traffic;
- Provide safety related information to the maritime community, such as other vessel traffic routes, and buoys off station;
- Maintain a traffic history for the harbor to assist with traffic research and enable measurement of VTS effectivenenss;
- Maintain a log of all VTC activities for measurement of VTS effectiveness, and provide a detailed accounting of the activities of the VTC and participating vessels on occasions when accidents occur.

The VTS is basically a data communications and processing network. It consists of:

- A voice communications network enabling communications between watchstanders (i.e., system operators) and vessel pilots;
- A network of sensors used to obtain information on vessel position and course, as well as waterway environmental parameters such as weather, tide and current data;

- Data communications links between the sensors and the VTC;

- An information storage, processing, retrieval, and display system;

- A set of vessel traffic monitoring and management procedures for real-time analysis and management of traffic flow.

2.4 VTS SYSTEM MODES, CLASSES AND SENSOR LEVELS

The various harbors in which VTS systems could be installed offer a wide range of requirements. The system design must provide the flexibility needed to adapt easily to the requirements of particular ports. Some of the parameters which affect the complexity of the harbor environment are:

- The number of vessels which traverse the harbor daily;
- The waterway geography;
- The types and numbers of sensors available;
- The class of the system;
- The current mode of operation.

Five levels of VTS systems have been defined based on the type of sensor which provides vessel position and course information. These sensor levels are:

- Level 1 - Location reports by voice on radio;
- Level 2 - Manual or automatic point sensors (e.g., magnetic or acoustic sensors which indicate a vessel's passage without identifying it);
- Level 3 - Manual area sensors, which may detect position and course information automatically, but require manual entry of this information into the processing/display subsystem;

. Level 4 - Automatic tracking by radar and input of position and course information;

. Level 5 - Automatic tracking using active shipboard electronics such as radar transponders, or Loran-C retransmitters.

The VTS processing/display subsystem is also described by its class and mode. The three possible modes are:

. Mode A - Informing, for which the following types of information are provided;

- Traffic information
  - . Identity of nearby vessels and buoys
  - . Predictions of future encounters
  - . Prediction of traffic congestion

- Navigational Information
  - . Unusual weather conditions
  - . Tide and current conditions
  - . Status of aids-to-navigation
  - . Hazards to navigation
  - . Maritime events of particular concern

. Mode B - Hazard Detection and Reporting, which provides, in addition to Mode A services, detection of the following types of hazards, which may be accomplished automatically, depending on the availability of appropriate sensors;

- Grounding
- Congestion

- Lane Stray
- Route Stray
- Dangerous Encounter
- Collision
- Excessive Vessel Speed
- Navaid Adrift/Missing
- Anchor Drift

. Mode C - Routing, which provides, in addition to the Mode A and B services, congestion free vessel routing through the waterway.

The class of a system refers to its maximum operating capability. The class of a system is designated using letters which specify the highest operating modes. A Class B system may operate in either Mode A or B. A Class B system, however, cannot operate in Mode C because the capability would not exist in a Class B system.

Most combinations of Class and Level are possible, depending on the harbor parameters and the relative need for hazard detection. A Class A system, however, can not support level 4 or 5 sensors.

2.5 VTS SYSTEM DESIGN GOALS

A number of design goals have been established for the VTS system to allow the designed system to be used to provide the class and level of service required at any given port or waterway. The hardware and software must have sufficient capability and flexibility to meet all the varying needs imposed by the different classes and levels and the wide range of system loads which will be encountered at different ports.

Some specific design goals or requirements are:

- Modularity - which allows adaptability of the system to a wide variety of harbors, expandability (and contractability) to handle increased (decreased) numbers of vessels, higher (lower) sensor levels, and greater (less) coverage area. Modularity also improves maintainability by making trouble shooting easier. Modularity also allows the standardization of hardware and software which facilitates training of hardware maintenance crews and watchstanders;

- Distributed Processing - using multiple processors to share the processing load and provide a high degree of reliability (i.e., 99.9% availability is required);

- Off-the-shelf hardware, to the extent that it is consistent with the other goals;

- The VTS processing/display subsystem must accommodate a maximum of:

    - one watch supervisor station

- ten traffic coordinator stations

- three external communicator stations

- one spare station which can be dedicated to training when all other stations are operational

2.6 VTS SYSTEM DATA BASE

The VTS System Data Base will consist of five major logical files. These files will vary in size depending on the individual requirements of a particular VTS System. In the following brief descriptions the number of records required for the maximum configuration is given.

. Vessel File - contains background information regarding vessels which traverse the VTS coverage areas frequently. This eliminates collecting data repetitively each time a vessel makes a passage through the VTS coverage area. In the maximum configuration the vessel file will have a capacity of 10,000 vessels.

. Passage File - contains information for each vessel while it traverses the VTS coverage area. This information includes the vessel's identification code, cargo, points of entry into and expected exit from the VTS coverage area, as well as intended route. Passage status, reflecting the state of the vessel in its passage, may be:

- imminent - vessel about to enter the coverage area (i.e., begin its passage)

- underway - passage currently in progress

- docked or anchored - passage ended within the coverage area

The passage file has a capacity of 2,000 passages in the maximum configuration.

- Waterway File - contains information about normal and special routes through the harbor, map cell data (e.g., depths), waypoints (which facilitate position reporting in a Level 1 system), docks, piers and Navaids.

- Environment File - contains weather, water current and tide information collected from manual or automatic environment sensors in the VTS coverage area. Its capacity is 20 automatic weather sensors, 20 automatic current/tide sensors, and 40 inputs from manual sensors of either type.

- Notices File - contains the text of official notices pertaining to some or all of the VTS coverage area. Its capacity is 100 notices.

For additional details on the VTS data base, see Section 8.4

2.7 PROCESSING CAPABILITIES

The capabilities of the VTS Processing Display Subsystem for processing input data fall into four categories:

- Background Processing
- File Operations
- Demand Processing
- Support Functions

Background processing consists of hazard detection, position update and system logging functions. File operations processing includes various data base functions for building the data files, modifying them, and deleting selected information. Demand processing allows the analysis and display of additional information in response to requests from the watchstander. Support functions consist of utility and exception functions designed to aid system usefulness and enhance system management and reliability. For further details on processing requirements see Section 8.

2.8 VTS SYSTEM ARCHITECTURE

During the first phase of the VTS design effort a variety of system architectures were considered for use in the VTS Processing/Display Subsystem. Systems including small minis were considered as were systems based on large minis. Dedicated serial lines and shared buses were considered. These physical structures were combined with logical structures which included a distributed function structure and a modified form of distributed load structure.

The alternative architectures were evaluated based on simplicity, modularity, flexibility, feasibility, maintainability, expandability, reliability, performance, and cost. The selected architecture configured for a Class C, Level 4 system capable of monitoring 900 identified vessels, is shown in Figure 2-1.

The system uses large minicomputers interconnected by a shared bus. Logically the system is a distributed function multiple processor. It is this system architecture which is the basis for the current design.

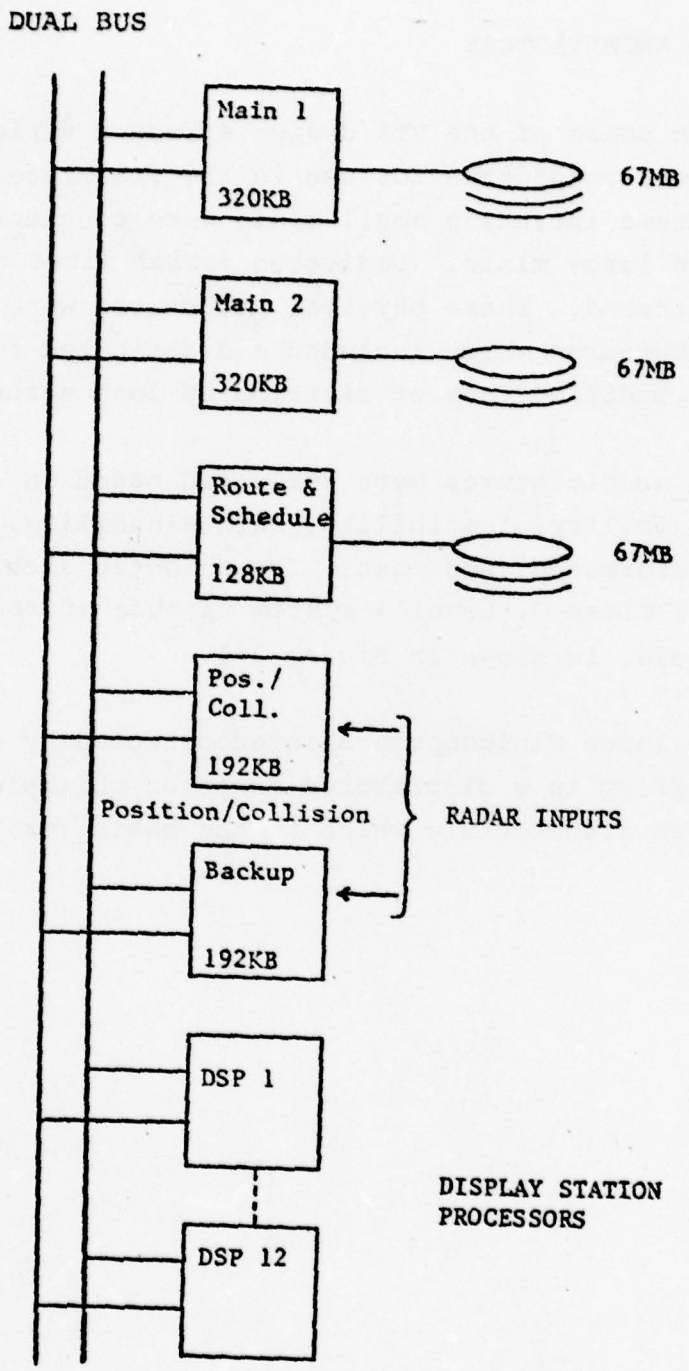


Figure 2-1. System Architecture for 900 Ships, Class C, Level 4

3 CRITICAL ISSUES

A number of approaches could be considered for the design and implementation of a system such as the VTS Processing/Display Subsystem. The goals of flexibility, modularity, reliability, performance and cost effectiveness can be achieved in varying degrees by a number of system architectures and by more than one approach to design.

While no known approach can be assured of producing an "optimum" system, the use of a distributed minicomputer system, developed using the best available software engineering techniques, seems to hold the most promise for acheiving the goals which have been set forth for VTS.

The VTS system, as currently envisioned, will be based on technology at the limits of the current state-of-the-art. While the technology proposed for VTS is not new, there remain a number of technical issues relating to both distributed processing and software engineering which have not been completely resolved.

Some of the issues which have not been resolved will have little impact on the potential success of VTS. For example, there is considerable debate over the general applicability of distributed processing. The general applicability of distributed processing has little relevance, however, since the VTS Processing/Display Subsystem is a dedicated system which has been shown to lend itself to distribution over a number of processors[1].

Several issues, however, could significantly impact VTS. We have identified four issues which are worthy of attention at this point in the VTS design. These four areas of concern will be introduced briefly in this section and discussed in more detail in succeeding sections of this report.

---

[1] Henson, C. C., Cleaver, R. A., Kaisler, S. H., "Preliminary Design Study for VTS Processing/Display Subsystem", June, 1978.

3.1 RELIABILITY

Reliability of both hardware and software is critical to the success of any system. Varying degrees of reliability may be needed depending upon the criticality of the function the system is performing. Real time systems, such as VTS, demand highly reliable software and hardware.

Relatively simple reliability calculations can show dramatic increases in the theoretical reliability of a distributed system with small increases in the quantity of hardware required (see Section 4). For such calculations to be realistic, the system must be designed to rapidly detect a failed device and reconfigure to use a spare device.

Procedures for fault detection, fault isolation and system reconfiguration are not well established, however. Although many actual or proposed distributed systems have cited increased reliability as an important benefit of the distributed approach, little has been published concerning how to actually achieve the increased reliability. In the distributed processing literature, discussions of how one can detect and isolate failures and reconfigure the systems are conspicuously absent.

In Section 4 of this report we will examine reliability in more detail and attempt to develop some basic principles and techniques to be used in the design of the VTS Processing/Display Subsystem.

3.2 LANGUAGE SELECTION

The programming language which is assumed during the design of a system and used in the development of the system software can have a profound affect on the reliability, flexibility, modularity, performance and cost of the system. Unfortunately the selection of a language for use in VTS design and development is not a simple task.

Many languages are available which could be considered. Traditional languages such as FORTRAN or COBOL are widely available and are much better than assembly language for certain types of programming. Both COBOL and FORTRAN, however, are limited in the types of programming they can support and neither supports the type of structured programming which is desired for VTS.

Other languages are available for some computers and may be useful for VTS design and development. Languages such as PASCAL have been designed to encourage structured programming and are also designed to be relatively efficient.

A review of available languages and the recommendation of one of these languages for VTS use is included in Section 5.

3.3 BUS DESIGN

A considerable amount of literature is available dealing with computer interconnection devices. These devices range from very simple to extremely complex. Several devices are commercially available at the present time and indications are that additional devices will become available in the near future.

In the section dealing with bus design (Section 6), we will review the functional requirements for the VTS interconnection bus. Since some vendors will not have off-the-shelf devices which meet the requirements, it may be necessary for the successful vendor to design and construct a bus for use in VTS. To ensure that such a device can be developed at a reasonable cost, we will also present a preliminary design for a low cost interconnection device which can be adapted by the vendors for use with their computer hardware.

3.4 OPERATING SYSTEM

The design of an efficient, flexible and powerful operating system is particularly important for the VTS Processing/Display Subsystem. Such an operating system must provide the services normally supplied in a single processor environment. In addition, the operating system must support the interprocessor communications necessary for distributed operation.

Section 7 of this report presents a preliminary design for the portions of the operating system which are unique to a distributed processing system.

4 RELIABILITY

Reliability of the hardware and software that make up a system plays an important role in determining the ultimate success of the system. It is well known that systems fail and no amount of design or care in construction of a system is sufficient to eliminate all failures. What must be done, however, is to reduce the impact of failures to an acceptable level.

4.1 SOFTWARE RELIABILITY

Software reliability is crucial to a system's success. Traditional methods for assuring that the software was reliable were based on the use of extensive testing of the software. While testing remains an important factor Dijkstra[1] and others have shown that testing alone is not sufficient to demonstrate that a program is correct since the time required to test all possible inputs and outputs would exceed the lifetime of the system.

More recent thinking has centered on correctness proofs which attempt to show that the logic of a program is correct and on testing techniques which reduce the probability of error to a small, measurable level.

Techniques for assuring the reliability of the VTS System software will be examined in more detail in the later phases of the VTS design effort. For now let us turn our attention to the problem of hardware reliability.

---

[1] Dijkstra, E. W., "Notes on Structured Programming" in Structured Programming, Dahl, O. J., Dijkstra, E.W., Hoare, C. A. R., Academic Press, New York, 1972.

4.2 HARDWARE RELIABILITY

Hardware components fail. Some fail frequently, others less frequently. Some devices are subject to wear so that the probability of failure increases with use. The probability of failure for many devices, however, seems to be essentially independent of the past history. Whatever the mechanism of failure may be, there is some probability that any given device will fail during a specified time period.

The fact that devices will fail, may or may not be a problem for a particular system. If the frequency of failure is sufficiently small and the systems can be returned to service rapidly, the reliability of the system might be adequate. The intended use of the system will determine what is an acceptable frequency of failure and how long the system can be out of service.

A frequently used measure of the reliability of a system is called availability. Availability is defined as the mean up time of the system divided by the mean up time plus the mean down time. That is:

$$A = \frac{U}{U+D}$$

Where: A = Availability
U = Mean Up Time
D = Mean Down Time

As we will show later (Section 4.6), availability is not a completely satisfactory measure of the reliability of a system. It is, however, the most commonly used measure. For the VTS Processing/Display Subsystem, the U. S. Coast Guard has determined that an availability of .999 is required.

4.3 RELIABILITY CALCULATIONS

Mathematical methods have been developed for estimating the reliability of systems made up of a number of devices. Based on probability theory, the mathematical treatment assumes that the probability functions associated with failures and repairs of individual devices are known and that failure and repair rates are independent random variables. From the known probability functions for individual devices the probability functions for combinations of devices can be calculated.

For most devices associated with a computer system, it is reasonable to assume that failures and repairs are exponentially distributed. Expressed mathematically, the probability of a failure in time t is:

$$P_s(t) = 1-e^{-\lambda t}$$

and the probability of a repair in time t is:

$$P_r(t) = 1-e^{-\mu t}.$$

Exponential distributions imply that a unit's past history has no influence on its future behavior. These characterizations of the failure and repair processes are imperfect, but it is well known that they yeild a good approximation of the behavior of many systems. λ is the inverse of the familiar mean time between failures (MTBF) and μ is the inverse of mean time to repair (MTTR).

If we further assume that failures and repairs are independent, that failures are detected immediately and that switching to a redundant unit takes negligible time the equations below can be derived[2].

Let U = mean up time of a system component (MTBF), and D = mean down time of a system component (MTTR). If m of n identical units must be up for the system to be up, and D is small compared to U, the mean up time for the system Us is given by:

$$Us = \frac{U\left(\frac{U}{D}\right)^{n-m}}{\frac{n!}{(m-1)!\ (n-m)!}} \qquad (2)$$

and the mean down time for the system Ds is given by:

$$Ds = \frac{D}{n-m+1} \qquad (3)$$

If the system contains two elements having different mean up and down times and both must be up for the system to be up:

$$Us = \frac{U_1\ U_2}{U_1+U_2} \qquad (4)$$

$$Ds = \frac{U_1\ D_2 + U_2\ D_1 + D_1\ D_2}{U_1 + U_2} \qquad (5)$$

2 Einhorn, S. J., "Reliability Prediction for Repairable Redundant Systems", Proceedings of the IEEE, February, 1963. Reprinted in Distributed Processing Tutorial, Liebowitz, B. H., and Carson, J. H., IEEE Catalogue No. IEEE EH 0127-1, 1977.

If the system contains two elements but the system will be up if either element is up (redundancy):

$$Us = \frac{U_1 U_2 + U_1 D_2 + U_2 D_1}{D_1 + D_2} \qquad (6)$$

$$Ds = \frac{D_1 D_2}{D_1 + D_2} \qquad (7)$$

To understand these formulas as they apply to a distributed system consider the following example. Suppose we need 10 identical processors to perform the functions of a system. Further assume that U for each processor is 2000 hours and that D for each processor is 1 hour.

If we provide 10 processors, then all processors must be operational for the system to be up. For n identical units, it can be shown from equations 4 and 5 that:

$$Us = \frac{U}{n} \quad \text{and} \quad Ds = \frac{(U + D)^n - U^n}{n\, U^{n-1}}$$

For our 10 processors then:

$$U = \frac{2000}{10} = 200 \text{ hours}$$

$$D = \frac{(2000 + 1)^{10} - (2000)^{10}}{10\ (2000)^9} = 1.002 \text{ hours}$$

and from equation 1:

$$A = \frac{200}{200 + 1.002} = .995$$

The system then would be out of service for slightly more than an hour about once every 8 days.

If this duration and frequency of failure is unacceptable we can add one more processor allowing it to replace any one of the others that is failed. Then equations 2 and 3 would apply with n = 11 and m = 10.

$$Us = \frac{2000\left(\frac{2000}{1}\right)^{11-10}}{\frac{11!}{9!\ (11-10)!}} = 36364 \text{ hours}$$

$$D = \frac{1}{11-10+1} = .5 \text{ hours}$$

$$A = \frac{36364}{36364 + .5} = .999986$$

The system then would be expected to be out of service for half an hour about once every four years. If we added another processor the mean up time would exceed 1000 years.

It should be easy to see then that if our mathematical models are realistic, we can achieve any desired availability by increasing the redundancy provided.

4.4 CONSIDERING THE ASSUMPTIONS

As we have just shown, from the mathematical treatment of reliability, we can conclude that any desired availability can be provided if we provide sufficient redundancy. This will indeed be the case if the real world system does not violate the assumptions behind the mathematics in a significant way.

To ensure that the designed system is reliable we should review the assumptions and identify areas of concern that may cause difficulties in meeting the availability requirements.

4.4.1 Fault Detection

Our reliability model assumes that a failure can be detected instantaneously. If a device fails we have assumed that we can determine that a failure has occurred, that we can determine which device failed, and that we can replace it with a redundant unit if one exists.

For some classes of failures this is not difficult. If a disc stops functioning, the fact that it has failed will be known as soon as we attempt to use it.

Other classes of failure can be much more difficult. For example, one processor requests another processor to perform a calculation. The result returned fails to pass a reasonableness check. From this we can know that some failure has occurred. Without additional knowledge, however, we do not know which of the two processors is at fault if indeed the problem was not caused by a software error.

4.4.2 Reconfiguration

Reconfiguration is assumed to occur immediately on the detection of a failure if a redundant element is available. The mechanism that effects the switchover is assumed to be perfect, i.e., not itself subject to failure.

In a real system, a perfect switchover mechanism cannot be achieved. We can, however, make the switchover mechanism more reliable than other portions of the system and provide a means for humans to override the actions of the switch over mechanism when it fails.

4.4.3 Independence

The mathematical treatment assumes that the failures of individual devices and their repairs are independent of failures and repairs of other devices. This assumption can only be validated by carefully considering the interconnection of devices and eliminating potential problems.

The system must be designed in such a way that a failure in one processor does not induce a failure in another. It must also be designed so that the act of repairing a device does not cause a failure in another.

Independence of repair times can also be a problem if adequate repair crews and facilities are not available. This is not a major problem, however, unless there is significant queuing associated with repair crews.

4.5 DESIGNING A RELIABLE SYSTEM

Having shown that there are some potential difficulties with the assumptions on which our mathematics is based we will turn our attention to designing a system that will minimize the problems and provide a system which is reliable.

4.5.1 Fault Detection

Fault detection procedures, as we indicated previously, should allow us to quickly determine that a fault has occurred and to determine which device is at fault. The first step in accomplishing this is to provide good online diagnostic software coupled with appropriate diagnostic aids in the hardware.

Memory, for example, should use parity detection or error correcting coding (ECC) to detect and/or work around failures. Magnetic tape units should use parity and read after write to verify operations. All I/O devices should provide status information to the associated processor indicating any errors or abnormal conditions detected by the device.

Processors can fail in less obvious ways than many of the peripheral devices. Obvious failures such as a halt or a non-terminating loop by the processor can be detected fairly easily, particularly if watchdog timers are employed. Less obvious failures are more difficult to detect. A processor might, for example, continue to operate but generate erroneous data.

To attempt to detect this less obvious type of failure, the VTS system will include reasonableness checks and periodic self diagnosis.

A distributed system presents some further complications with regard to fault detection. As we indicated earlier, two processors may disagree on the state of their own and the others health. To overcome this sort of disagreement and identity the faulty element, a third processor if it exists will be used to correlate any additional information and arbitrate the dispute.

4.5.2 Error Reporting

All error conditions which are detected by any part of the system will be reported to a designated station for use by both operations and maintenance personnel. Error messages should be produced for all failures including both recovered and unrecovered I/O errors, failures to pass reasonableness tests, and any other abnormalties which may be detected.

Every attempt should be made to include sufficient data in the error message to allow the person reviewing the message to make an independent judgement, if needed, to resolve disputes.

4.5.3 Reconfiguration

The task of reconfiguration control will be assigned to a single processor. All processors will have the software necessary to perform the reconfiguration function but only one will be given the authority at any give time. The processor so designated will also be designated as the main error reporting center (MERC) which will receive and correlate error reports from other processors (see Section 7).

Before a reconfiguration action is taken, the processor which has the authority to reconfigure will perform a thorough self diagnostic to attempt to detect any failures of his own. If the self diagnostic is successful, the reconfiguration action will be reported to operations personnel and undertaken by the reconfiguration processor.

If the self diagnostic fails or if the reconfiguration processor cannot determine which element has failed, the situation will be reported to operations personnel who will make the reconfiguration decision.

4.5.4 Independence

Independence of failures and repairs must be considered in the design. This will be particularly important in the design of the interconnection bus (Section 6).

It must be possible to repair a failed bus interface without inducing failures in the other bus interfaces or the associated processors. It should be possible to replace a bus interface without disabling the associated processor. If this is not possible it may be necessary to provide additional redundancy to allow for the "failure" induced in the processor when the bus interface is replaced.

4.5.5 Maintenance Policy

Maintenance policy plays a significant role in determining the availability that can be expected of a system and must be considered in the design phase. If we return to the equation for availability (1) we can readily see that a reduction in mean down time can significantly alter the calculated availability.

Mean down time of a system or of a system component is to some extent determined by the characteristics of the system or component. Some devices are inherently difficult to trouble shoot and repair, resulting in a longer repair time.

The figure for mean down time, however, is more dramatically affected by maintenance policy. If maintenance personnel are on site, and adequate spare parts are available so that most repairs involve only a card replacement, mean down time can normally be kept quite low. If, however, maintenance personnel must be brought to the site after a failure and maintenance personnel must actually trouble shoot the individual device and replace capacitors, integrated circuits, and the like, the mean down time would be an order of magnitude higher.

For the VTS Processing/Display Subsystem there will normally be sufficient hardware to justify on site maintenance personnel. For determining the availability requirements we will assume that mean down times can be kept low by appropriate maintenance policies.

4.6 ANOTHER LOOK AT SYSTEM RELIABILITY

As we indicated earlier, availability is the most commonly used measure of reliability. Availability, however, may not be an adequate measure of the reliability of a system.

In this section we will examine reliability again and consider some additional factors that may determine how well a given system may meet the needs for reliability.

4.6.1 Availability

Availability is a useful measure of the reliability of a system. Availability, however, is a dimensionless quantity which can hide information which may be extremely important to system design and ultimate performance. Consider the following example:

System	Uptime	Downtime
A	3 hours	10 seconds
B	3 years	26 hours

Both systems have an availability of .999. Both might be unacceptable for the VTS Processing/Display Subsystem. Although system A is only down for a very short time, it fails 8 times a day. Such a system would not engender confidence in the watchstanders who used it.

System B on the other hand has an acceptable uptime but is down for more than a full day when it fails. For a critical real time system such as VTS, such a lengthy down time might be unacceptable even though it occurs infrequently.

4.6.2 Failsoft

System availability requirements often apply to a system capable of supporting all system functions at required response times. Often a number of options may be available for operation in a degraded or failsoft mode when a failure occurs.

In the VTS Processing/Display Subsystem a significant portion of the overall processing load results from hazard detection process that are executed frequently. The usefulness of the system is only slightly reduced by decreasing the frequency of execution of these hazard detection processes.

The VTS system design, then, should include the flexibility to use failsoft modes of operation if unusual combinations of failures occur. For any single failure, however, redundancy should be provided to assure that full functionality is maintained.

5 LANGUAGE SELECTION

Language has a profound affect on the way we think as well as the way we communicate. This is true for programming languages as well as for natural languages. The programming language that is selected for VTS Processing/Display Subsystem therefore will significantly affect both the design and implementation.

The selection of a programming language is not a straightforward task, however. Many languages are available but none can be called ideal for VTS. We will, therefore, evaluate a number of languages and select a language for use in the design and development of VTS. Based on our evaluation we will recommend that PASCAL be used to develop the VTS software.

5.1 CRITERIA FOR THE LANGUAGE SELECTION

There are literally hundreds of languages which have been implemented on at least one computer. Many of these languages have disappeared from use. New languages constantly appear.

FORTRAN is the oldest of the languages which are still widely used. When FORTRAN was designed in the middle fifties, little was known about what might make a good language. The designers of FORTRAN sought to produce a compiler that would generate reasonably efficient code. The design of the language itself was thought to be almost trivial.*

Since FORTRAN was designed, a great deal has been learned about what programming ought to be. The concept of structured programming first showed the importance of the control structures provided by a programming language. More recent experience has shown that data structuring may be equally important.

For the VTS development, the type of control and data structures provided will be important factors in the selection of the language. A number of other factors will also be considered. The factors to be considered include:

- Suitability for all programming required for VTS;
- Control Structures supported;
- Data Structuring provided;
- Ease of use;
- Efficiency;
- Error Detection
- Readability;
- Availability

---

*Backus, J., "The History of FORTRAN I, II and III," SIGPLAN Notices, Vol. 13, No. 8, August 1978.

These factors will be described in more detail in the subsections which follow.

5.1.1 Suitability for All VTS Programming

The development effort for the VTS Processing/Display Subsystem will include real-time applications software and a real-time multicomputer operating system. A variety of systems and applications software is required. Complex numeric and non-numeric programming will be included.

The programming language selected for VTS must be able to support all these forms of programming. It is particularly important that the language be suitable for programming the operating system. Many operating systems are written in assembly language, with applications written in a higher order language. For VTS, however, such an approach would result in a significant increase in cost since the operating system represents a major portion of the total development effort.

5.1.2 Control Structures

The control structures provided in a language have been shown to be important in the development of reliable software.* The control structures used in a program can improve both our ability to write it correctly and our ability to demonstrate its correctness.

The language selected for VTS should encourage or require the use of good control structures such as WHILE...DO, IF...THEN... ELSE, and CASE. It should discourage or even prevent the use of control structures which are considered undesirable such as GOTO and the computed IF.

---

*Gamon, J. D. and Horning, J. J., "The Impact of Language Design on the Production of Reliable Software," ACM Sigplan Notices, Vol. 10, No. 6 (June 1975), pp. 10-22.

To better understand the value of good control structures, consider the following examples.

Example 1

Associate a value with each day of the week. If the day is Monday, Tuesday or Wednesday, set the value to 1. Set the value to 2 if the day is Thursday or Friday. Set the value to 3 if the day is Saturday or 4 if the day is Sunday.

This operation can be expressed using nested IF...THEN...ELSE structures.

```
IF (day = mon) or (day = tue) or (day = wed)
   THEN x:=1
   ELSE IF (day = thur) or (day = fri)
      THEN x:=2
      ELSE IF day = sat
         THEN x:=3
         ELSE x:=4;
```

The structure shown is acceptable but the program is simpler and significantly more readable if we use a CASE statement.

```
CASE day OF
   mon, tue, wed:  x:=1;
   thur, fri     :  x:=2;
   sat           :  x:=3;
   sun           :  x:=4;
```

The example above shows how a program can be improved by selecting a control structure that represents the program's intent more effectively. A second example can perhaps make the point even more dramatically.

Example 2

Read a sequence of inputs x and y and compute the sum of their square roots until either x or y is negative.

This sequence of operations can be written using the FORTRAN computed IF and GOTO statements.

```
1 READ(x)
  READ(y)
  IF (x) 4,2,2
2 IF (y) 4,3,3
3 z = SQRT (x) + SQRT(y)
  GOTO 1
4 ...
```

The resulting program segment can be deciphered but in no way can it be called readable. A further problem is that we cannot make statements about the validity of this program segment, except that it appears to be correct. The existence of labels which can be referenced by a GOTO elsewhere in the program makes it possible to enter this segment at statement 2 or 3 which would result in improper operation.

These problems are resolved if we use a WHILE...DO construction.

```
READ(x);
READ(y);
WHILE (x> = 0) AND (y> = 0)
   DO z = SQRT(x) + SQRT(y);
        READ(x);
        READ(y);
   END
```

This program segment is readable and cannot be entered improperly.

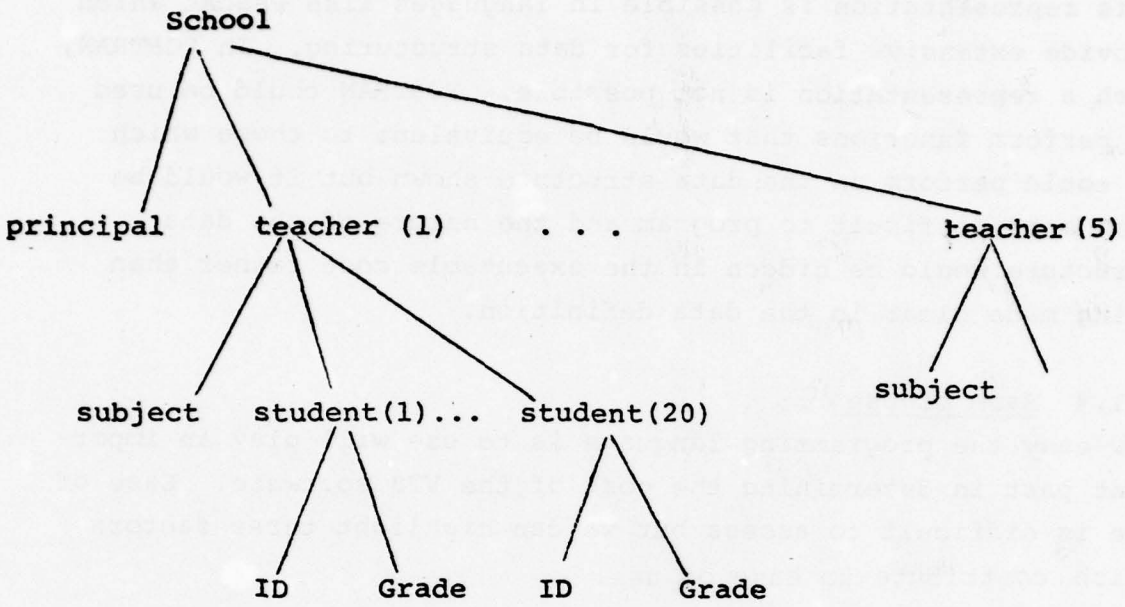
There is not a total agreement among computer scientists on which control structures are necessary and most desirable. It should be clear, however, that we should give preference to a language that provides a good set of control structures.

5.1.3 Data Structuring

Current research suggests that data structuring facilities may be as important in a programming language as the control structures.

Some languages, FORTRAN for example, provide only rudimentary facilities for data structuring such as simple integer or floating point variables and arrays. COBOL and PL/1 allow records and more extensive types of data to be defined.

More recent languages have extended the data structuring facilities to allow hierarchical structures to be built up and new data types to be defined in a reasonably straightforward manner. Consider the following example in which we have defined a school consisting of a principal and five teachers. A subject and a group of 20 students is associated with each teacher. A student ID and a grade is associated with each student. A diagram of this structure is shown below.



The data structure required can be represented as follows:

```
TYPE  School = RECORD

    Principal:  (Mr_Jones, Ms_Smith);
    Teacher list:  ARRAY (1..5) OF

        RECORD
            Subject:  (biology, English, math);
            Studentlist:  ARRAY (1..20) OF
                          RECORD
                              ID:  integer;
                              GRADE:  (A, B, C, D, F);


        END
```

This representation is possible in languages like PASCAL which provide extensive facilities for data structuring. In FORTRAN, such a representation is not possible. FORTRAN could be used to perform functions that would be equivalent to those which we could perform on the data structure shown but it would be much more difficult to program and the nature of the data structure would be hidden in the executable code rather than being made clear in the data definition.

5.1.4 Ease of Use

How easy the programming language is to use will play an important part in determining the cost of the VTS software. Ease of use is difficult to assess but we can highlight three factors which contribute to ease of use.

Simplicity of the language contributes to ease of use. A large number of constructs can make the language more effective, or it can reduce effectiveness by increasing the complexity. Unnecessary complexity should be avoided. The language selected for VTS should not have multiple means for expressing a given idea nor should it include facilities which are not needed for VTS.

What we will call uniformity is also important. Similar ideas should be expressed in the same way under all conditions. Special cases should be avoided. It should never be necessary for a moderately experienced programmer to check the reference manual to be sure how a situation should be handled.

The language should also conform as much as possible to the rules of natural and mathematical languages. If the language is unnatural, errors are almost guaranteed. PL11, for example, has no operator precedence and ignores parentheses. The statement

```
X: = 3 * (4 + 5)
```

assigns 17 to X rather than 27 and the statement

```
Y: = 1 + 2 * 3
```

assigns 9 to Y rather than 7. Neither of these yeilds the result one would normally expect. Errors would be avoided only with great difficulty.

5.1.5 Efficiency

Efficiency of the run time code is always a concern in a real time system. Assembly language has often been used for real time systems because of a concern for efficiency.

Since truly time critical parts of a system are few, the concern for efficiency is often misplaced. Efficiency cannot be ignored, however.

Efficiency is often a function of the implementation of the compiler rather than being inherent in the language itself. Efficiency will not, therefore, play a significant part in our evaluation since we are evaluating languages and not particular compilers.

5.1.6 Error Detection

Even with the best structuring, the first draft of any program is not likely to be error-free. Since a computer is more adept at checking details than the programmer, extensive error detection by the compiler is highly desirable. Compile time checks are usually superior to runtime checks since they do not affect execution speed. In addition, checking which is delayed until runtime will not discover an error until the particular section of code with the error is executed. Thus, error detection becomes data or time dependent and errors may persist for a longer time.

All compile time detection is dependent on redundant information, although redundancy can also produce errors since the user must enter the information in two different ways. One of the most effective uses of redundancy is mandatory data type declarations. Typeless languages (e.g., BCPL) or languages with automatic type conversion (e.g., PL/1) may allow subtle errors to go undetected.

Example 1: No mandatory type declarations

```
DECLARE TONE INTEGER:
    .
    .
    .
    .
T0NE = x + y;
```

Here the user has made a mistake in the assignment statement. TONE has been spelled with a zero instead of an O.

Since type declarations are not required, a new variable T0NE will be created in addition to the previously declared TONE.

Example 2: Automatic type conversion

```
DECLARE A, B, X BOOLEAN;
        .
        .
        .
      X: = A + B;
        .
        .
        .
```

What the user meant was X: = A and B; but with automatic type conversion A and B are treated as integers and added together, not logically ANDed. If the sum of the test data A and B is a valid boolean true or false, the user may never see the mistake.

5.1.7 Readability

The language should make communication between those who develop the system and those who maintain it clear, convenient and precise, since the two groups are seldom the same. A large system seldom stabilizes; it continues to grow and needs correction. If it is not understood, revision can destroy it. Thus, readability must be emphasized over writability.

Many of the features which improve the control and data structuring of a language also improve its readability. The features which add to ease of use (Section 5.1.3) normally improve readability as well. Simplicity, uniformity and naturalness are all important in readability.

Mandatory data declarations are also helpful since they require an increased level of internal documentation.

It is interesting to note that a number of languages have been designed specifically with readability in mind. The designers of ALGOL*, for example, attempted to provide a language that would be suitable for publication. ALGOL has been generally accepted in this country for the publication of algorithms in spite of the fact that it has never been widely accepted in the United States as a programming language.

*Perlis, A. J., "The American Side of the Development of ALGOL", SIGPLAN Notices, Volume 13, Number 8, August, 1978.

COBOL is perhaps unique, in that the designers intended that it be read by management*. English was used extensively and noise words were added to improve readability. It is not clear that COBOL was read extensively by management but it is highly readable.

5.1.8 Availability

There are many languages in use today on one or more computer systems and some extremely interesting language developments are now underway. For an actual implementation such as VTS, we must limit ourselves to languages which are currently available for the class of processors to be used or languages which will be available when the VTS implementation begins.

It is also important that the language be well known and accepted which will assure that programmers are available who know the language. We should also choose a language which will continue to be available throughout the lifetime of the VTS system.

The language selected for VTS must have a compiler which will execute on and generate code for the target machine.

*Sammet, J. E., "The Early History of COBOL", SIGPLAN Notices, Volume 13, Number 8, August, 1978.

5.2 CANDIDATE LANGUAGES

Hundreds of programming languages have been developed over the years. Many of these languages have been developed for specific applications. APT, for example, was designed for numerical control applications. APT has been used for numerical control for more than 20 years.

Even more specialized languages have been developed recently. BUGSYS, for example, is designed for use in preparing animated movies. ELMOL, on the other hand, is designed for use with a Moog Synthesizer. Such languages, of course, have little relevance for the VTS Processing/Display Subsystem.

A variety of languages are relatively general purpose in nature, and sufficiently available to be considered for VTS. A detailed analysis of all the languages and dialects is beyond the scope of this selection effort. The following list has therefore been distilled from the much larger list which could be compiled:

- ALGOL
- BASIC
- COBOL
- FORTRAN
- PASCAL
- PL/1

In addition, we will consider the Department of Defense Common Higher Order Language which is now under development. Although the DOD language is not yet available, there is much to be learned from the analysis done by DOD and while debate continues about how good the resulting language will be, it is clear that with DOD backing, the new language will have a major impact.

5.2.1 ALGOL

ALGOL was originally designed in the late fifties in an attempt to develop a universal language for use in numeric programming*. The original designers sought to provide a language that would be translatable by a number of computers while being suitable for publication.

For a variety of reasons, ALGOL was accepted as the language for the publication of algorithms but did not achieve great popularity in the United States as a language for actual programming. It did, however, become widely used in Europe.

ALGOL is suitable for a portion of the VTS operating system but would have to be supplemented with assembly language. Control structures are good. Data structuring facilities are limited to integer, real and Boolean variables and arrays. All data must be declared. ALGOL gets high marks for ease of use and readability.

ALGOL compilers have been developed for a variety of computers. Its availability is limited, however, for the class of mini-computers which will be used for VTS.

Overall, ALGOL can be considered as a possible but hardly ideal candidate for VTS.

5.2.2 BASIC

BASIC or Beginner's All-purpose Symbolic Instruction Code was developed in 1963 - 1964 as a language to be used by non-technical students in a time sharing environment†. In recent days it has become the primary language of the hobbyist.

---

*Perlis, A. J., "The American Side of the Development of ALGOL," SIGPLAN Notices, Volume 13, Number 8, August, 1978.

†Kurtz, T. E., "BASIC", SIGPLAN Notices, Volume 13, Number 8, August, 1978.

Although BASIC is available on nearly all computer systems and is certainly easy to use, it must be considered as totally inadequate for VTS. Control structures are very limited, data structuring facilities are almost non-existent and it simply cannot be used for operating system software making it unacceptable for VTS.

5.2.3 COBOL

COBOL was designed and developed in 1959 - 1961 and has become the primary language for business oriented programming. It is unsuited for operating system programming. It's control and data structures are good. It is highly readable but the verbosity which is required makes it somewhat difficult to use.

Most minicomputer manufacturer's have COBOL compilers available making it a possible but poor choice for VTS.

5.2.4 FORTRAN

FORTRAN was designed and developed in 1954 - 1957 and is the oldest of the languages which we will consider. The design of FORTRAN was started at a time when computers were extremely difficult to use. It met a great need which then existed and its use spread rapidly.

By today's standards FORTRAN is antiquated. Because of the massive amount of software which exists in FORTRAN, however, the language continues to be used extensively. Until recently there has been little evidence to support the notion that other languages were better than FORTRAN except in an aesthetic sense. Recently, however, the concepts of structured programming and software engineering have been accepted and the shortcomings of FORTRAN have been recognized. FORTRAN's shortcomings can now be translated into dollars for software development and maintenance.

The major deficiencies in FORTRAN are its control structures and its data structuring facilities. The control structures can be improved by using a preprocessor which translates more suitable control structures (IF...THEN...ELSE, WHILE...DO, etc.) into standard FORTRAN control structures (computed IF, DO, GOTO, etc.).

The shortage of data structuring facilities has not been overcome by preprocessors. Significant extensions to FORTRAN will be necessary if it is to overcome its shortcomings. As Barron has noted, however, "Why go to all this trouble when PASCAL exists already?"*

For VTS purposes, FORTRAN would be a poor choice. Only a small portion of the operating system could be written in FORTRAN. Ease of use is fair at best, particularly since a preprocessor would be essential and readability is poor. Perhaps the only real reasons for considering FORTRAN, even briefly, is that it is available on nearly all computer systems and is known by most programmers.

5.2.5 PASCAL

In 1971, Wirth introduced a new language known as PASCAL**. PASCAL features excellent control structures and extensive data structuring facilities which make it suitable for almost all types of programming.

---

* Barron, D. W., SIGPLAN Notices, Volume 13, Number 2, February, 1978.

** Wirth, N., "The Programming Language PASCAL", ACTA INFORMATICA, 1, pp. 35 - 63, 1971.

PASCAL is available for a variety of minicomputer systems and is supported by at least one minicomputer manufacturer. Readability and ease of use are both good.

Error detection facilities are also quite good. PASCAL, unlike the other languages discussed, attempts to do extensive error checking at compile time. Many compilers perform little more than basic syntactic checks at compile time. PASCAL uses the information required with data declarations to perform more extensive checks of the program's logic.

Documentation is available from suppliers of PASCAL compilers. Documentation and a defacto standard is also available in the form of a report prepared by the developer*.

The one shortcoming of PASCAL compilers which are currently available is that they normally generate an intermediate code which is then interpreted rather than generating directly executable code. This approach was taken to make it easy to transport the compiler from one machine to another. This problem can be overcome by developing a relatively simple translator that will convert the intermediate code to executable code.

Overall, PASCAL appears to be a good choice of language for use in the development of the VTS Processing/Display Subsystem.

5.2.6 PL/1

PL/1 was developed in 1964 - 1966 in an attempt to combine the best features of FORTRAN, COBOL, ALGOL and other known languages into a single language which would serve both commercial and scientific users. The result was a powerful but complex language that can support both scientific and commercial programming.

---

* Jensen, K., and Wirth, N., PASCAL User Manual and Report, Springer-Verlag, New York, 1974.

PL/1 can also be used for the major portion of operating system software. An example of the use of PL/1 in systems programming is the Multics Project*. Of roughly 1500 system modules, all but about 250 were written in PL/1. The Multics Project, using PL/1, demonstrated that a higher-level language is indeed viable for systems software and that maintainability is significantly improved over assembly language.

Control structures in PL/1 are good and data structuring facilities are extensive although unlike PASCAL, data structures are not extensible. PL/1 is moderately easy to read but moderately hard to use because of its complexity.

The major problem with PL/1 for VTS purposes is a lack of availability for minicomputer systems. Some limited dialects of PL/1 have been developed for minis and even micros (e.g., PL11 for the PDP/11 and PL/M for micros) but are essentially specialized machine dependent languages which are not appropriate for VTS. PL/1 also suffers from a lack of uniformity and a level of complexity which is not needed for VTS.

PL/1, then, must be considered as inappropriate for VTS.

5.2.7 Department of Defense Common High Order Language

In January, 1975 the United States Department of Defense (DOD) initiated a specification effort leading to the design and development of a common high order language for use in developing software for imbedded systems, i.e., systems in which the computer is an integral part of a larger overall system. A primary objective of the effort is to reduce the cost and/or improve the quality of the software which is currently costing DOD about three billion dollars a year.

*Clingen, C. T., Corbato, F. J., and Saltzer, J. H., "Multics - The First Seven Years", reprinted in Software Systems Principles, Freeman, P., Science Research Associates, Chicago, 1975.

The VTS Processing/Display Subsystem is essentially an imbedded system, similar in many respects to the types of systems for which the DOD language is intended. A review of the specifications* indicates that if the language fulfills the intent of the specification, it would be an outstanding choice for the language to be used for VTS.

Current design and development schedules call for a language available for experimental use in 1979 and available for general use in 1980**. Since it is not yet clear which computers will have the language available, it is unrealistic to assume that the DOD language can be used as a basis for the VTS procurement.

Although realistically the DOD language cannot be selected, a review of such an important language effort can allow us to benefit from the extensive study carried out by DOD.

The DOD requirements were developed in a series of specifications starting with the STRAWMAN, followed by the WOODENMAN, then the TINMAN and finally the IRONMAN specification. Near the end of the specification process, DOD performed an extensive analysis of existing languages. Each of the languages considered was carefully evaluated against each point of the specification to determine whether it met the specification or could be modified to meet the specification.

Language receiving formal evaluations included FORTRAN, COBOL, PL/1, HAL/S, TACPOL, CMS-2, CS-4, SPL/1, JOVIAL J3B, JOVIAL J73, ALGOL 60, ALGOL 68, CORAL 66, PASCAL, SIMULA 67, LIS, LTR, RTL/2,

---

* Department of Defense Requirements for High Order Computer Programming Languages Revised "IRONMAN", July, 1977.

** Whitaker, W. A., Lt.Col., "The U. S. Department of Defense Common High Order Language Effort", SIGPLAN Notices, Volume 13, Number 2, February, 1978.

EUCLID, PDL2, PEARL, MORAL, and EL-1. A number of other languages were examined for specific features, while others, such as APL, were immediately excluded since they were obviously inappropriate.

The evaluators determined that none of the languages could be adopted as the common language. They determined, however, that it was appropriate to select a carefully chosen base language from which to start the design of a language that met the specifications.

Without exception the DOD evaluators rejected FORTRAN, COBOL, TACPOL, CMS-2, JOVIAL J73, JOVIAL J3B, SIMULA 67, ALGOL 60 and CORAL 66 as inappropriate for the base language. PASCAL, PL/1 and ALGOL 68 were selected as appropriate base languages from which to start the design.

Four contractors were selected to start the design with two continuing after the initial phase. It is interesting to note that all four contractors selected PASCAL as a base language. It is also interesting to note that of the three languages which DOD recommended for the base language, only PASCAL is really available for use on minicomputers.

As we indicated earlier, the DOD language and supporting software is scheduled to be available for use sometime in 1980. Clearly this new language will have a significant impact on the software of the eighties. Unless the new language is a total failure, the fact that it will be a common language will represent a significant improvement over the current multiplicity of languages. Whether it will indeed be an improvement over PASCAL is open to question, however. As Dijkstra has said in commenting on the DOD effort, "Of ALGOL 60, C.A.R. Hoare once remarked that it was a significant improvement over almost all of its successors. What can we do to prevent PASCAL from sharing that fate?"*

*Dijkstra, E. W. DOD-I: The Summing Up, SIGPLAN Notices, Volume, 13, Number 7, July, 1978.

5.3 CONCLUSION

Clearly our analysis has shown that PASCAL should be chosen as the language to be used for VTS software. The other languages considered are unsuited for operating system software or are unavailable at the current time for use with the class of processor selected for VTS.

The extensive evaluation of existing languages by DOD lends credence to this conclusion.

6 VTS BUS AND INTERFACES

6.1 GENERAL

In order for the distributed network architecture of the VTS system to be practically and economically feasible, an inexpensive and reliable inter-processor bus must be provided to allow communication among the member processors of the network. The range of alternative bus structures which must be considered is limited by the operational parameters of the system architecture selected by the U. S. Coast Guard as a result of the Phase I study.

The network as defined consists of a number of minicomputers communicating as co-equal members, and thus precludes consideration of a master-slave bus control philosophy. A shared link bus interconnection was selected to support network communications, and therefore this discussion will not include alternative interconnection geometries.

The general class of contention-based bus control philosophies, such as Slotted Aloha and Listen-While-Talk, will not be considered for several reasons. First, these techniques are generally more suitable for low message-volume, many member networks over a relatively large geographical distance, while the VTS network will be a collocated system of processors with a high volume of message traffic. Additionally, current contention-based bus allocation philosophies employ cable television technology, requiring hardware for radio-frequency modulation/demodulation and for contention detection. This additional hardware, aside from raising the cost of each bus interface, will affect the reliability of the interface by introducing complex elements whose failure will result in failure of the bus interface as a unit.

6.2 ALTERNATIVE BUS STRUCTURES

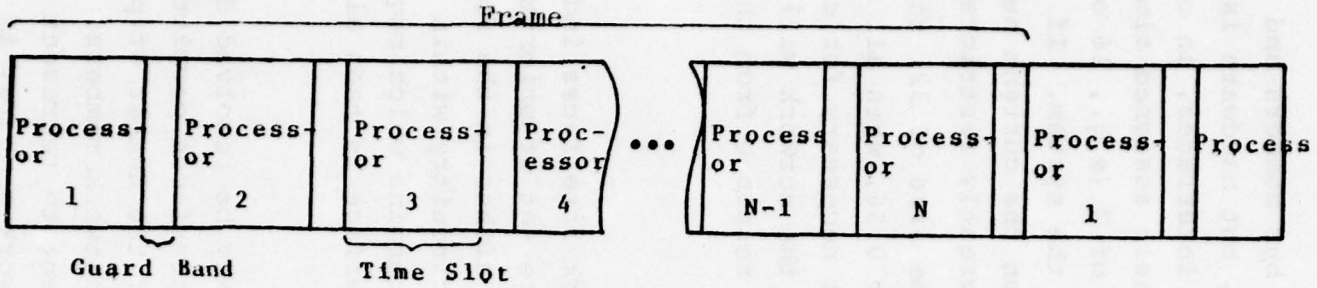
Two major structural aspects of the VTS system bus have impacts on the speed and reliability of data transfer within the network. Section 6.2.1 describes available alternatives for bus control transfer, and Section 6.2.2 presents the factors involved in evaluating serial versus parallel data bus paths. The technique employed to transfer control of the bus from one interface to another may be considered independently from the nature of the data path, although the affects of both areas must be combined to arrive at speed and reliability estimates for the complete system.

6.2.1 Bus Control Transfer

In any non-contention, shared-link bus, the interface of each member processor must receive control of the bus before it can transmit message data. The control transfer technique employed will affect the bus data rate by introducing overhead, and will affect the reliability of the bus based on the amount of circuitry required to implement the technique, any additional data paths required to perform the control transfer, and the ease with which control transfer failure may be detected and corrected. Four control transfer techniques are commonly available in shared-link bus systems: assigned time slot transmission (TDMA); priority arbitration; polling; and distributed polling, or control token transfer.

6.2.1.1 Time-Division Multiple-Access(TDMA)

In this approach, each member of the network is assigned a time slot of fixed length, during which messages may be transmitted. A fixed number of time slots constitute a frame, the length of which serves as the limit of the number of members in the system; each member may transmit once per frame. Figure 6.1 shows an example of TDMA bus control. TDMA bus control factors are the same for serial and parallel bus configurations.



Frame Size = Maximum number of processors = N

$$\text{Maximum Utilization} = \frac{\text{Number of active processors}}{N}$$

(assumes negligible guard band between Time Slots)

Figure 6.1 TDMA Bus Control

This technique requires no arbitration among bus members and no direct communication to transfer bus control, but hardware is required to synchronize timing among the bus interfaces, in order to prevent interfaces from drifting out of their assigned time slots. The frame length, usually a multiple of 2 (e.g., 16 or 32), is an absolute limit to the expandability of the system. If the frame length is made substantially larger than the current number of network members, then bus utilization is greatly restricted. In a network with eighteen members and a frame size of 32. the maximum possible bus utilization is 18/32, or 0.56, when all members are transmitting. Since communication is not necessary for control transfer, the failure of an interface within the network will not be detected by the network until a requested response from that network member is determined to be overdue.

6.2.1.2 Priority Arbitration

Priority arbitration requires that all network interfaces indicate their readiness to transmit data on a separate set of priority communication lines. Arbitration circuitry, either in the form of a centralized bus controller or distributed circuitry within each interface, must decode these requests and determine which requesting interface has the highest priority; that interface is then allowed to transmit.

The number of priority lines or codes which may be resolved determines a fixed upper limit on the number of interfaces permitted in the network. There is no overhead penalty if the number of priority lines or codes is larger than the number of network members; however, the number of priority lines must be sufficient to represent all possible priority codes, or all possible network members, if one line is used for each interface. The arbitration circuitry and additional lines increase the complexity of the interfaces, and in the case of centralized arbitration, represent a potential single point of failure which can incapacitate the bus.

The requirement of a predetermined priority among network members can pose a problem in guaranteeing that the lowest priority members can obtain adequate bus access. In situations where bus message traffic is extremely heavy, or where high-priority members are transmitting many large blocks of data, control transfer to low-priority members may be unacceptably delayed.

6.2.1.3 Centralized Polling

Polling may be applied equally well to serial and parallel data buses, and may either be conducted on the data communication lines, in the form of a polling message, or may be conducted on separate polling lines. A centralized polling unit transmits the address of the polled interface, along with an indication that bus control is available. The polled interface then responds with a negative acknowledgement, (NACK) if it has no message to transmit, or it responds with positive acknowledgement. This positive acknowledgement may be a separate code or signal (ACK), or may be signalled by transmission of the waiting message. Priority is usually not implemented in polled systems; each interface is interrogated in turn, and once all interfaces have been polled, the cycle begins again with the first interface.

Where separate polling lines are used, the number of lines required is determined by the largest polling address, such that five polling address lines can serve 2^5 or 32 network members. Where polling messages are transmitted on the bus data lines, the interface must be able to recognize the polling message and determine whether that interface is being polled. Additional circuitry may be required for this recognition and poll address comparison. Use of polling messages adds overhead to bus data transmission, but the polling messages are not lengthy, and the effect of this overhead on the bus data transmission rate decreases as bus message traffic increases. The central polling unit represents a potential single point failure which would incapacitate the bus.

6.2.1.4 Distributed Polling

Distributed polling resembles centralized polling, but the central polling unit is eliminated; the interface having control of the bus is responsible for passing control to the next interface in the polling sequence. Polling messages or dedicated polling lines may be used to transfer control, following the procedure described in Section 6.2.1.3, but an additional possibility results from the distribution of control transfer responsibility. Each interface may have control lines connected only to adjacent interfaces in the polling sequence. To transfer control, an interface transmits a logic pulse, or control token, to the next interface. The interface receiving control returns an acknowledgement signal, and is then prepared either to transmit message data on the bus or to pass control to the next interface in the polling sequence.

The control token alternative requires special wiring from each interface to its two neighbors in the polling chain. Thus, if the control token hardware on one interface becomes inoperative, control transfer along the polling chain becomes impossible. In addition, to provide the capabilitg of removing one interface from the system without affecting control transfer, active circuitry must be added to the control section of the bus to pass control to the adjacent interface; the presence of these active components decreases the inherent reliability of the data transfer medium (see Section 6.2.2).

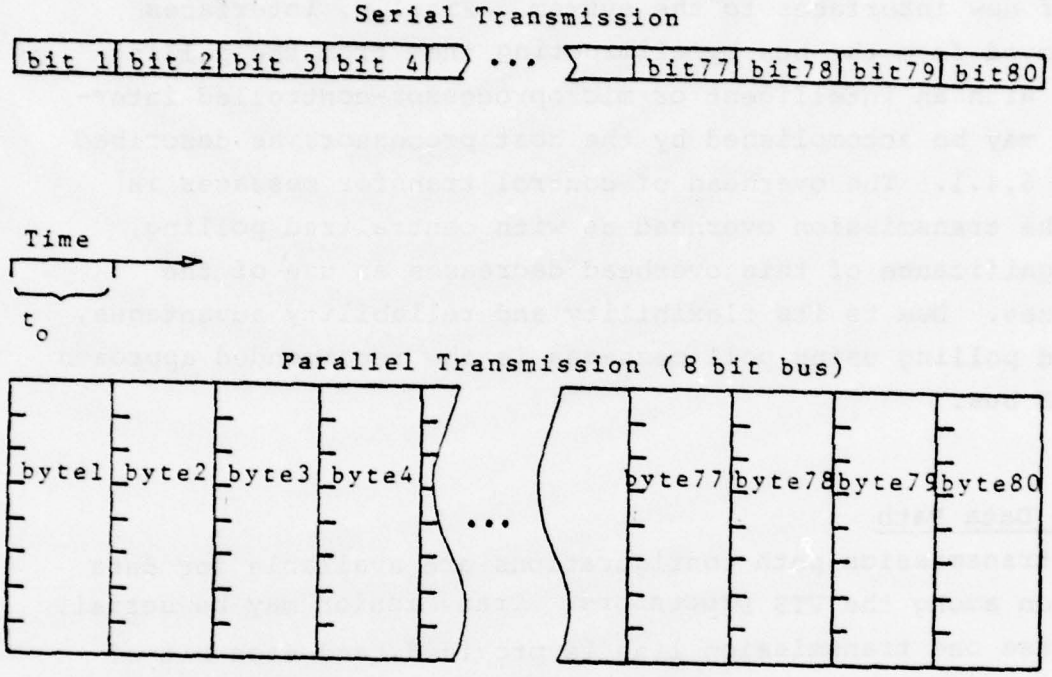
Distributed polling using poll messages offers significant advantages over other control transfer techniques. First, the same transmission lines may be used for both data transmission and control transfer, decreasing the number of wires in the bus and the number of active components in each interface, and increasing the reliability of the bus. Second, since the transfer of control is specified in a message, a byte of the message may

be reserved to designate the interface to which control is passed; this allows up to 256 interfaces to be connected to the bus without increasing the complexity of the interface, and allows easy addition of new interfaces to the system. Finally, interfaces may be removed from the bus by eliminating them from the polling sequence. With an intelligent or microprocessor-controlled interface, this may be accomplished by the host processor, as described in Section 6.4.1. The overhead of control transfer messages is added to the transmission overhead as with centralized polling, but the significance of this overhead decreases as use of the bus increases. Due to its flexibility and reliability advantages, distributed polling using poll messages is the recommended approach for the VTS bus.

6.2.2 Bus Data Path

Two basic transmission path configurations are available for data transmission among the VTS processors. Transmission may be serial, in which case one transmission line is provided, and each bit of data is presented on that line for a specific time period, followed by the next bit in the message sequence. Parallel transmission uses multiple transmission lines, and allows several bits of information to be transferred at the same time. Figure 6.2 shows a serial bus format and a parallel bus format with an eight-bit parallel data path.

Using Figure 6.2 as an example, the parallel bus structure possesses a speed advantage of eight times the serial bus transmission rate, assuming that bits are presented for the same length of time on both bus types. However, for the parallel bus to continue operating, all eight data lines must be operational, versus one line in the serial case. The active components involved in the bus data transfer (bus drivers, bus receivers, etc.) are replicated eight times per interface in the parallel case, so that the Mean Time Between



t_o = 1 bit time

Figure 6.2 Serial and Parallel Data Buses

Failures (MTBF) of this critical bus section is one-eighth that of the same section in the serial bus. The reliability advantage of the serial bus is thus traded for the higher transmission rate of the parallel bus; however, in the VTS system configuration selected, the Phase I analysis arrived at a transmission rate of approximately 500,000 bits per second in the Class C, level 4 example. Since single-circuit communications devices now exist which are capable of serial transmission at more than 2,000,000 bits per second, the transmission speed required for the VTS system may be achieved with a serial bus structure, providing higher reliability and lower cost than in the parallel bus case.

The bus transmission medium connecting the VTS bus interfaces may be passive or active. A passive bus would consist of a direct wire connection among all interfaces, having resistors to terminate the bus, preventing reflections of the transmitted data at each end of the bus from interfering with data transmission. An active bus would consist of shorter segments of bus wiring connecting devices such as transistors or logic gates, for the purpose of regenerating bus signals or isolating sections of the bus not involved in the current data transfer. The wiring of the bus has a large effect on reliability, since the inherent reliability of cable and resistors is many times greater than that of active components. To preclude the possibility of a failure of the transmission medium, a passive bus transmission medium should be selected.

6.3 BUS REQUIREMENTS

The VTS system inter-processor bus requirements are presented in this section with the goal of placing as few mandatory requirements on the bus components as are necessary to ensure proper operation and sufficient capacity to meet VTS system operational criteria. In order to prevent exclusion of feasible bus architectures due to overly restrictive requirements, those areas of bus philosophy or design having multiple solutions will be described in terms of desirable features and design goals, to allow vendors with differing solutions to the data transmission problem to present their solutions for evaluation by the U. S. Coast Guard. Section 6.4 presents a bus description which incorporates the maximum number of the desirable features presented below, as a basis for discussing interactions between bus hardware and the operating system.

Section 6.3.1 details the overall bus requirements which must be met by the communications network. Section 6.3.2 presents the requirements for the bus transmission medium and the section of the bus interface which must connect to the transmission medium. Section 6.3.3 describes the computer interface and requirements for processor-interface signals and communications.

6.3.1 System Considerations

Two primary system requirements must be met by the VTS system bus. The Phase I analysis of Candidate Architecture II indicates that, in the maximum capacity configuration, data transmission of approximately 500,000 bits per second is necessary for normal system operation. A serial bus transmission rate of 1,500,000 bits per second, or a parallel bus rate corresponding to 250,000 eight-bit bytes per second, would provide the capacity to handle peak load situations without degrading response times for individual functions within the system.

For purposes of expansion and allowing spare processors to be included in the VTS system, a minimum of 20 processors must be capable of

communicating over the inter-processor bus. Bus structures allowing more than 20 member processors provide more than the minimum flexibility and expandability, and are thus more desirable. However, for those buses where this additional capacity results in additional overhead or lower data throughput, a penalty should be assessed.

6.3.2 Bus Transmission Medium and Interface Connection

The bus transmission medium should be passive to provide for the required system reliability, and should be capable of sustaining communications at a low error rate over a distance of at least 200 feet. An error rate of one error in 10^{10} bits shall be the highest acceptable rate. The VTS environment will be relatively noisy, due to the presence of radio and radar equipment; the bus transmission medium and connectors must provide effective shielding to prevent noise from interfering with VTS system bus transmissions.

Reliability considerations require that the bus be duplicated; that is, there will be two separate busses each capable of handling the complete processing load. However, rather than having one bus carry the entire communications load while the spare bus is idle, in normal operation both busses will be active and share the communications load. This will allow exceptional communications peaks to be handled without service degradation, and will provide effective monitoring of the status of both busses to rapidly detect failure of either bus. Increased fault tolerance may be realized by the use of three busses, however, because this increases complexity significantly, it is preferred that the specified system availability be achieved by improving the reliability of the bus itself rather than increasing the number of busses.

The section of the bus interface which directly connects to the bus transmission medium must allow removal of one or more interface connections from the bus without degrading bus communications among the remaining computers; this presumes that a method exists for

gracefully excluding the interface(s) from the control transfer mechanism of the bus. This allows maintenance/replacement of an interface or computer without disabling the remaining elements of the system.

Where possible, the bus interface design should minimize the effect of a single interface failure on bus communications. The bus/interface connection should be subjected to a failure mode analysis, indicating those failure modes within the interface which could disable the bus, and evaluating their probability of occurrence.

6.3.3 Bus Interface

The basic criterion for comparing alternative bus interfaces is simplicity of design. Aside from its relation to interface cost, simplicity of interface design has a great impact on interface reliability. Where two interface designs have equal capabilities, and one design incorporates substantially fewer components, that interface will be more reliable, assuming equivalent reliability figures for the interface components. Adherence to this assumption is not realistic for calculations of reliability, as large-scale integrated circuits have lower MTBF figures than medium- or small-scale integrated circuits; consequently, proposed interfaces should be accompanied by figures for component and total interface reliability.

The bus interface will interrupt the connected computer in the event of significant occurrences involving bus data transfers. To the greatest degree possible, interruption of the processor should be minimized to prevent unnecessary software overhead in the operating system. The only condition requiring an interrupt is the termination of a message transmission or reception. Status indications will be available to the processor when this interrupt is presented, by vectored interrupt or in a status register indicating whether the terminated message was transmitted or received, and any error conditions which may have caused premature or incorrect termination of the message. Additional interrupts indicating failure of bus hardware would be desirable to aid in rapidly detecting the loss

of communications capability. Transfer of bus control among interfaces should be transparent to the connected processors; the generation of interrupts forcing processors to participate in control transfer would greatly increase the percentage of operating system overhead within the processors. Additionally, should the transfer of control among interfaces be disrupted, certain classes of bus interfaces are capable of detecting this condition and generating processor interrupts, aiding in the rapid re-establishment of communications on the failed bus. However, since the bus is duplicated, detection of control transfer interruption is not required for reliable operation.

The bus interface must be capable of Direct Memory Access (DMA) to message buffers in the host processor for transmission/reception of data. The capability of using DMA to load control information into the bus interface would be desirable to decrease the processor overhead in preparing for message transmission/reception, but is not of great importance.

The nature of the VTS system requires that messages of widely varying lengths be transmitted on the bus. The bus interface must be capable of variable length data transfer, and should require a minimum of processor interaction in handling lengthy messages. If the bus interface generates a non-error interrupt when its allocated buffer has been filled and additional message data is still being received, sufficient data buffering must be provided within the interface to allow time for the processor to allocate additional buffer space in its memory for the remaining data. A desirable feature of the bus interface would be to allow multiple buffers to be reserved for use in simultaneous reception from multiple transmitting interfaces or tasks; however, benefits resulting in decreased operating system overhead will be realized from any interface construction allowing association of an allocated message buffer with a specific expected message.

6.3.4 Hardware/Software Interaction

A device driver module is required in the VTS operating system to provide the proper signals and data to the bus interface. The complexity of this device driver and the amount of operating system overhead incurred, is dependent upon the amount of intelligence in the bus interface. A more intelligent bus interface can perform more of the interfacing functions and thus relieve the device driver and the operating system of this overhead. In evaluating a proposed bus interface therefore the merits and demerits must be weighed in terms of the host processor overhead incurred, and the impact on information throughput capability of the bus.

In most cases, the use of an intelligent bus interface means namely that the same function that the device driver would otherwise have to perform can be performed by the bus interface. For example, a low intelligence bus interface would have to interrupt the host processor when it receives a control transfer message. The device driver of the host's operating system would then determine whether or not the message was intended for that processor, and if it is, it would then perform the message transmission and then send a control transfer message to the next interface in the polling sequence. A more intelligent bus could perform these functions internally without having to interrupt the host processor. There is a case, however, where the placement of a process in an intelligent bus interface greatly simplifies the device driver software (and greatly reduces the accompanying operating system overhead); this is in the handling of variable length messages. As discussed in section 6.3.3, various length messages on the communications bus pose a problem with many possible solutions. Following are three solutions involving various degrees of bus interface intelligence. It is obvious that the first solution, involving the most intelligent interface affords the greatest reduction of operating system overhead while maintaining the highest bus communications capacity.

Solution (1): When a message of "n" bytes is expected, an "n"-byte memory area is reserved in the host processor. The bus interface must be able to recognize the incoming transmission when it arrives and transfer the message directly to the reserved memory area in the host. This technique requires the least complex device driver and the most intelligence in the bus interface.

Solution (2): Memory area is reserved by the host for a message and the bus interface is informed to reject transmissions from all but the expected source. This approach significantly decreases the transmission throughput of the bus in a multiprocessor environment. It requires less intelligence in the bus interface than the first solution, but does not greatly increase the device driver complexity.

Solution (3): Fixed-length memory areas are used to receive all transmission, with the bus interface interrupting the processor when this fixed length memory area has been filled and more data is expected. The device driver must then assign a new fixed-length memory area to receive this additional data, and must move the received segments of the transmission to the message buffer reserved for use by the application program. The device driver will have to be relatively complex in this case, and the number of interrupts generated will contribute to a high operating system overhead. The driver must be capable of allocating new memory areas for transmitted data before data from the bus is lost.

6.4 TARGET BUS DESIGN

Given the availability of large-scale-integration (LSI) circuits for communications and control, it is feasible to design a custom bus interface for the VTS system which meets the mandatory requirements and provides most of the desirable features presented in Section 6.3. This section presents one design for such an interface, involving the following currently available LSI components:

- A synchronous communications controller, capable of supporting burst mode transmissions at a rate of 1.5 million bits per second, and providing low-level protocol support (e.g., Signetics 2652 and SMC Microsystems 5025);

- A high-speed microcomputer to monitor and control operation of the communications controller and the processor interface (e.g., AMD 2901, Intel 3000 or Signetics 8X300);

- A DMA controller capable of supervising four concurrent transfers to and from the host processor's memory (e.g., Intel 8257). This DMA controller may not be necessary if the microcomputer is sufficiently fast to perform DMA to the host processor, as is the case with the target bus design shown in Figure 6-3.

Figure 6.3 shows a block diagram of the target bus interface, based upon the Signetics 8X300 microcomputer, where the DMA is performed by the microcomputer rather than a controller. The scratchpad Random-Access Memory (RAM) will hold addresses and byte counts for four concurrent DMA operations.

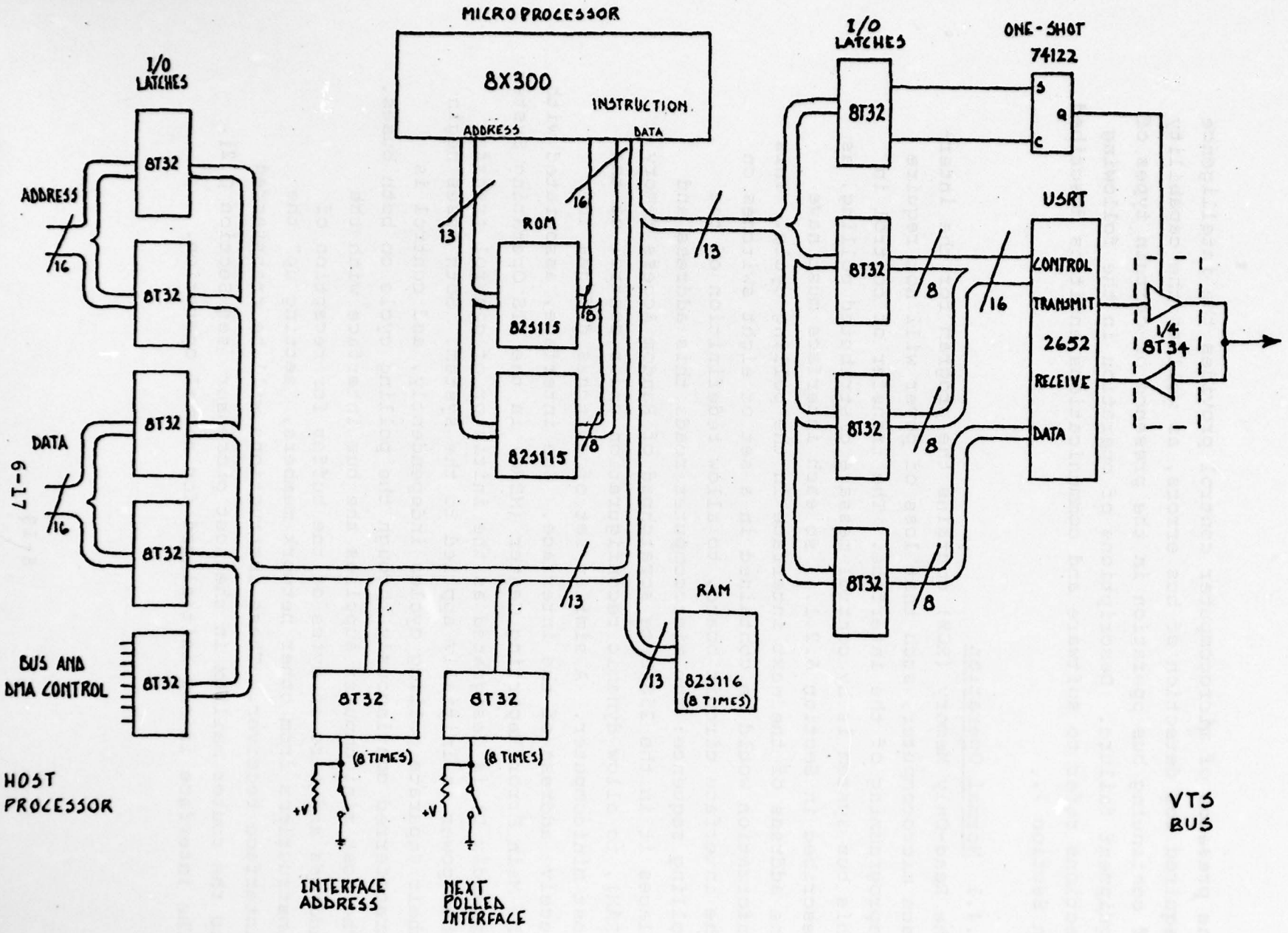


Figure 6-3. Bus Interface Block Diagram

The presence of microcomputer control provides the intelligence required for detection of bus errors, as well as the capability of continuing bus operation in the presence of certain types of equipment failure. Descriptions of operation in the following sections refer to software and communications entities described in Section 7.

6.4.1 Normal Operation

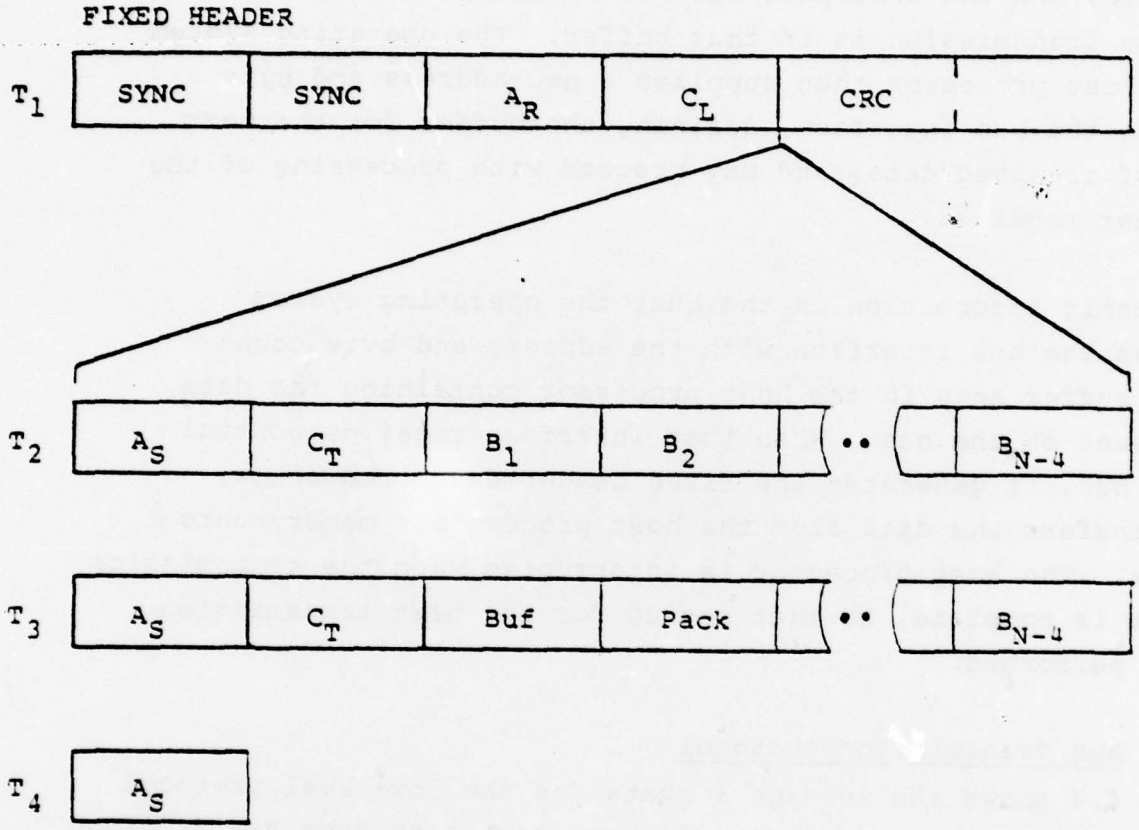
The Read-Only Memory (ROM) contains the program for the interface microcomputer, such that loss of power will not require reprogramming of the interface. The transfer of control in this bus system is by control message distributed polling, as described in Section 6.2.1.4, so each interface must have the address of the next interface in the polling cycle. This information would be contained in a set of eight switches on the interface circuit board, to allow redefinition of the polling sequence; the microcomputer reads this address and places it in the 256-byte scratchpad of Random Access Memory (RAM), to allow dynamic reconfiguration under control of the host minicomputer. A similar set of switches defines the receive address of the interface. One interface, associated with the Main Error Reporting Center (MERC) in the VTS Operating System (Appendix D) is designated as the initiator of control transfer when power is initially applied to the system. Both buses begin their separate polling cycles independently, and control is transferred continuously through the polling cycle on both buses. The host minicomputer supplies the bus interface with the address and size in bytes of the buffer for reception of instructions from other network members, "setting up" the interface receiver. These instructions will be transferred to the router mailbox in the host processor (see Section C.2). The interface is now initialized for normal operation.

When the bus interface receives a transmission, the received data is placed in the buffer area the host processor has allocated, and the host processor is interrupted when the complete transmission is in that buffer. The operating system of the host processor then supplies a new address and byte count to the bus interface, defining the buffer for the next block of received data, and may proceed with processing of the data just received.

To transmit information on the bus, the operating system supplies the bus interface with the address and byte count of the buffer area in the host processor containing the data to be sent on the bus. When that interface receives control of the bus, it generates the fixed header described below, and transfers the data from the host processor's memory onto the bus. The host processor is interrupted when the transmission process is complete, so that set-up for the next transmission can be performed.

6.4.2 Bus Transmission Protocol

Figure 6.4 shows the message formats for the low-level protocol portion of bus transmissions. Each transmission type has a unique value for its control byte C_L thus allowing the receiving interface to directly determine the transmission type from this byte. While four transmission types are identified in this figure, additional transmission types may be defined during the system design, each having its own identifying control byte value. The bus interface microcomputer monitors messages on the bus looking for its assigned receive address or a broadcast code in byte A_R. When it sees either its address or broadcast code, it then identifies the message type by its control byte value and takes different actions depending on the message type. The function of the four transmission types and some of the actions taken are as follows:



SYNC	-	Message start synchronization byte
A_R	-	Address of receiving interface (may be broadcast code)
C_L	-	Control Byte - Designates transmission type
CRC	-	Cyclic redundancy check word - 2 bytes
A_S	-	Address of sending interface
C_T	-	Count of bytes in message portion
$B_{1,2..}$	-	Message bytes
Buf	-	Buffer code (see Section C.3)
Pack	-	Packet number (see Section C.3)

Figure 6.4. Transmission Formats

Transmission type T_1: This is the control transfer message. Upon receipt of this message type, the bus interface will transmit one packet of any message which it has waiting on its queue (or the entire message if it is the short type T_2). After the packet or short message is sent the bus interface then passes control of the bus to the next interface in the polling sequence by sending a T_1 message to that interface. Note that if there is no message waiting on the queue to be sent when an interface receives control, it will pass control to the next interface immediately.

Transmission type T_2: This transmission type is for messages or instructions which are short enough to fit into the allocated general purpose message reception buffer. The interface sending this message will interrupt its host processor when the transmission is completed.

Transmission type T_3: This transmission type is for solicited messages which are too long to fit into the general purpose buffer, and must have a special buffer allocated for it prior to reception. A buffer code (Buf) accompanies this type transmission so that the receiving interface will know which reserved buffer area the message is intended for when more than one such solicited message is expected. This buffer code and the buffer address which it represents are assigned by the host processor when the message is solicited, and are placed in the bus interface's random access memory (RAM) so that the interface will know where to store the message when it arrives. This capability to place incoming messages in pre-allocated buffers relieves the

operating system Dispatcher of the burden of having to perform this buffer management and eliminates the overhead associated with having to transfer messages from a general purpose buffer to the user specified buffers.

Transmission type T_4: This transmission type is used to alert other bus interfaces (and thus their host minicomputers) that the host minicomputer of the sending bus interface appears to have failed. This is discussed further in section 6.4.3. This transmission type will always contain a broadcast code in the receive address field (A_R) which will cause all other bus interfaces to receive the message.

6.4.3 Error Detection and Resolution

The interface microcomputer monitors the host processor interface, and maintains a timer to indicate the time since the interface was last addressed by the host. If the host processor fails to address a watchdog timer port within the interface for an inordinate length of time, a type T_4 transmission is sent indicating to the other interfaces within the VTS system that this host processor is inoperative or, possibly, overloaded with processing. The MERC may take appropriate reconfiguration action based upon reception of this message.

The Sync bytes in the transmission header are detected by certain of the communications controller circuits, and may be used by the interface microcomputer to signal the presence of activity on the bus. Should an interface send a T_1 transmission to transfer bus control and fail to see bus activity following the transmission, that interface will interrupt its host processor with an indication of this error condition. The host processor may then take corrective action as noted in Appendix D.

Figure 6.3 shows the transmission and reception sections of the communications controller connected to the VTS bus through buffers controlled by a one-shot IC, which is periodically reset by the interface microcomputer. This one-shot has a time constant long enough to allow the longest transmission possible in the system to be completed before the one-shot reverts to its stable off condition and disconnects the transmission section from the bus. However, should a failure occur within the bus interface microcomputer which causes the interface to continue transmitting on the bus, after the one-shot reverts to its off state, the interface will no longer tie up the bus, and the other interfaces may re-establish control transfer on the bus.

6.4.4 Bus Characteristics

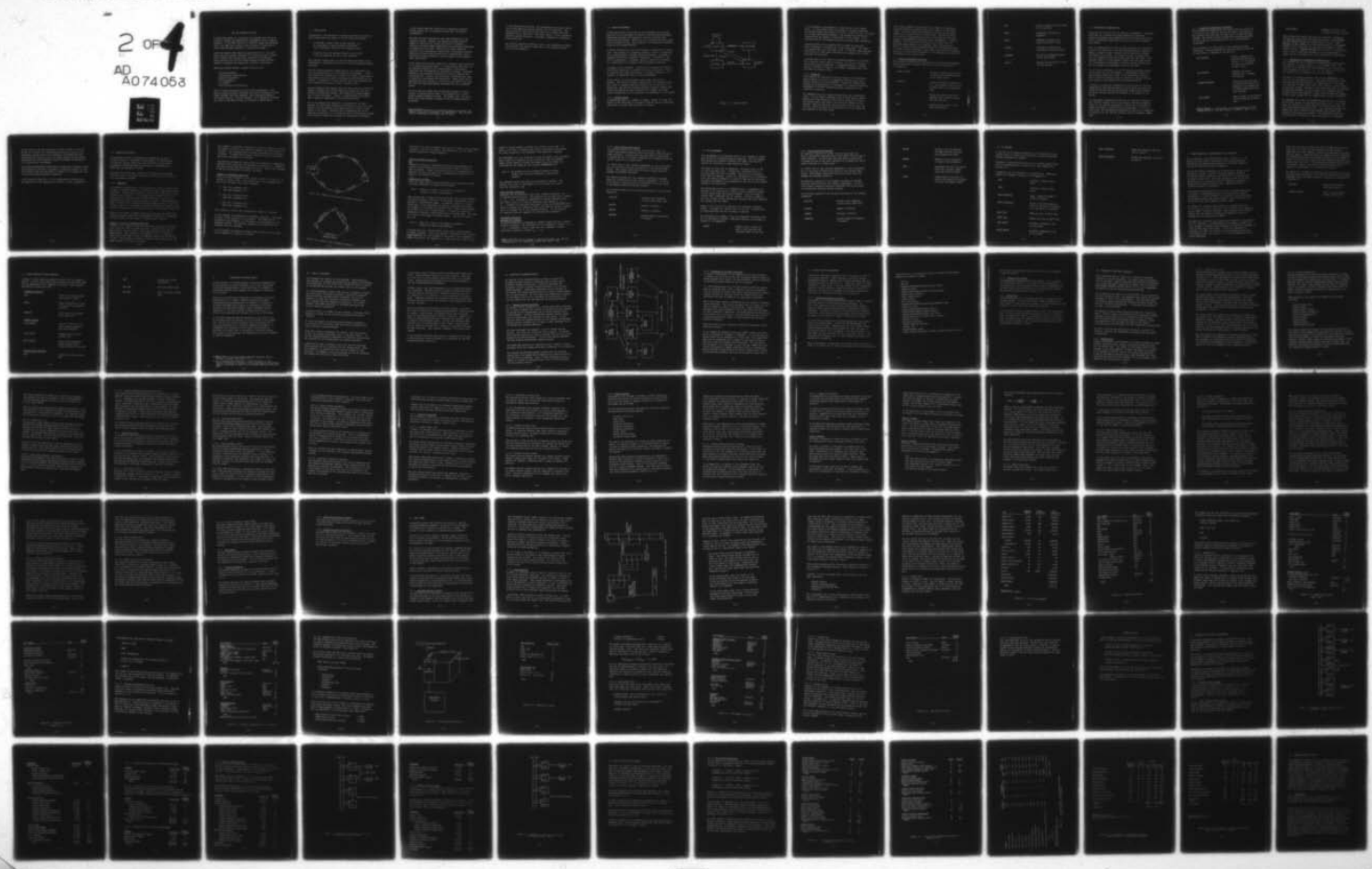
The structure of the bus interface allows the VTS bus itself to be a totally passive transmission medium. With proper termination and shielding of the bus medium and impedance matching with the bus buffers, the required bus length of 200 feet is easily realized with possible extension to greater distances. Provided that the host processor of one bus interface is directed to modify its polling transmission to exclude the next sequential interface in the polling cycle, that next interface may be physically disconnected with no effect on bus transmissions. Should an interface fail or be removed, the bus will be reconfigured under host processor control to allow continued control transfer.

6.4.5 Interface Reliability and Flexibility

The use of microcomputer interface control provides several additional benefits. The simplicity of design allows the bus interface to be constructed with relatively few integrated circuits, thereby increasing the inherent reliability of the interface. The interface is designed such that data transfers to and from the host processor memory may be performed as eight bit or sixteen bit transfers, allowing interface modification without redesign by changing the ROM interface program. Finally, the use of powerful LSI components substantially decreases the cost of the VTS bus interface.

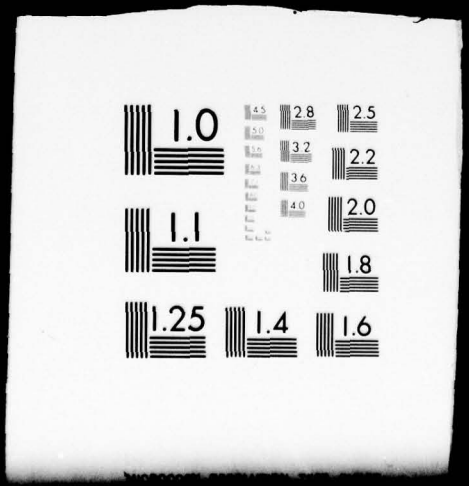
AD-A074 053 INTERNATIONAL COMPUTING CO BETHESDA MD F/G 9/2
VTS PROCESSING DISPLAY SUBSYSTEM DESIGN.(U)
JAN 79 C C HENSON, F T MICKEY, R S GRAHAM DOT-CG-81-78-1833
UNCLASSIFIED USCG-D-54-79 NL

2 OF 4
AD A074053



2 OF 4

AD
A074053



7 THE VTS OPERATING SYSTEM

An operating system is a collection of programs that act as an interface between the applications programmers and the computer hardware. It provides services to simplify coding and debugging, and at the same time, controls the allocation of resources to ensure the efficient operation of the entire system.

Operating systems have advanced significantly over the years. Early operating systems consisted of little more than a collection of macros and routines which simplified I/O handling. Today operating systems have become complex software systems capable of managing and supporting a variety of functions.

Operating systems normally include facilities for:

- Process Management;
- Interprocess Communications;
- Resource Allocation;
- File Management;
- I/O Control;
- Error Detection and Recovery.

What is called the operating system often encompasses a whole array of system software including job control languages, virtual memory systems, data base managers, and elaborate procedures for system security. For VTS, however, it is important to narrow the scope of the design effort by focusing on design goals.

7.1 DESIGN GOALS

Traditionally, the designers of operating systems have had two major goals which also apply to the VTS Operating System:

- To provide a higher level virtual machine, i.e., an environment in which programs can operate that facilitates the use of the computer system;

- To assure that the computer system is used effectively by managing the use of its resources.

More specific design goals for the VTS Operating System can be established by examining the nature of the VTS Processing/Display Subsystem.

The VTS Processing/Display Subsystem will be a dedicated real time system operating in a multi-computer environment. It must have sufficient flexibility to allow it to handle a wide range of requirements that may change significantly from one VTS site to another. Availability is also critical since the subsystem will be in operation seven days a week, 24 hours a day.

The multicomputer environment which the VTS Operating System must support differentiates the VTS Operating System from the vast majority of operating systems. Most operating systems support only a single processor. Clearly, a multicomputer environment adds a degree of complexity to the operating system.

One of the design goals should be to eliminate or at least minimize the additional complexity with which the user must cope. Ideally, the multicomputer environment would be totally transparent to the users. Although total transparency may not be completely achievable, it is essential that an application program have no need to know the allocation of processes to processors

or the current physical location of a cooperating process. Stated another way, an essential design goal is location transparency.

The need for high availability also adds complexity to the operating system. Ideally, the operating system would be capable of detecting all errors and failures instantly, isolating a failed element immediately and recovering completely in essentially zero time. While such an ideal cannot be achieved, we can establish a design goal of providing automatic detection of errors and failures and of automatically reconfiguring the system whenever a failed element can be unambiguously identified*.

Other design goals include simplicity, efficiency and flexibility in the operating system itself. If these design goals are met, the operating system will be understandable, maintainable, modifiable and unencumbered by unnecessary overhead. Increased complexity and overhead introduced by multiple computers and reconfiguration increase the difficulty in achieving the goals of simplicity, efficiency and flexibility. These goals can be substantially met, however, if we do not lose sight of the fact that the VTS Processing/Display Subsystem will be a dedicated system.

The fact that the system will be dedicated allows us to omit a variety of features which ought to be considered for a more general purpose operating system. The design, then, can focus on the facilities needed to support a VTS Processing/Display Subsystem.

*Since failure detectors are themselves subject to failure, 100 percent surety is not possible. By unambiguous, then, we mean having an extremely low probability of error.

In the sections which follow, the requirements and a basic design for the VTS Operating System (VTS/OS) will be presented. The major operating system procedures which are callable from user programs will be described. Additional detail on user accessible operating system functions is included in Appendix B, Operating System Functions.

For critical portions of VTS/OS, such as the interprocess communication system, the internal operation of the operating system will also be presented.

7.2 PROCESS MANAGEMENT

VTS/OS must provide facilities for the management of processes. Functions will be provided for creating and destroying processes, scheduling processes for execution, and for activating and deactivating processes. VTS/OS will also provide facilities which cause a process to wait, when necessary, for resources or synchronization with another process.

A precise definition of the term "process" is difficult to formulate. The concept of a sequential process, however, is essential for an understanding of modern operating systems. A process is sometimes defined as a program in execution. A process is the unit of concurrency. A process, while it exists in a processor, is a combination of a program, status information and data.

A program is not a process. A program is a sequence of instructions which are design to perform a particular function or group of functions when they are executed. A process, however, is a sequence of operations which are being performed.

Within VTS/OS, a process control block (PCB) will define a process. The PCB will contain a description of the machine state which must be restored when the process is to be executing. In particular, the PCB will contain pointers to process unique data areas, the address at which execution will resume, and other status information which is necessary for management of the process.

7.2.1 Process States

A process may exist in a number of states. Figure 7-1 shows the states in the existence of a process in VTS/OS and the transitions which move a process from one state to another.

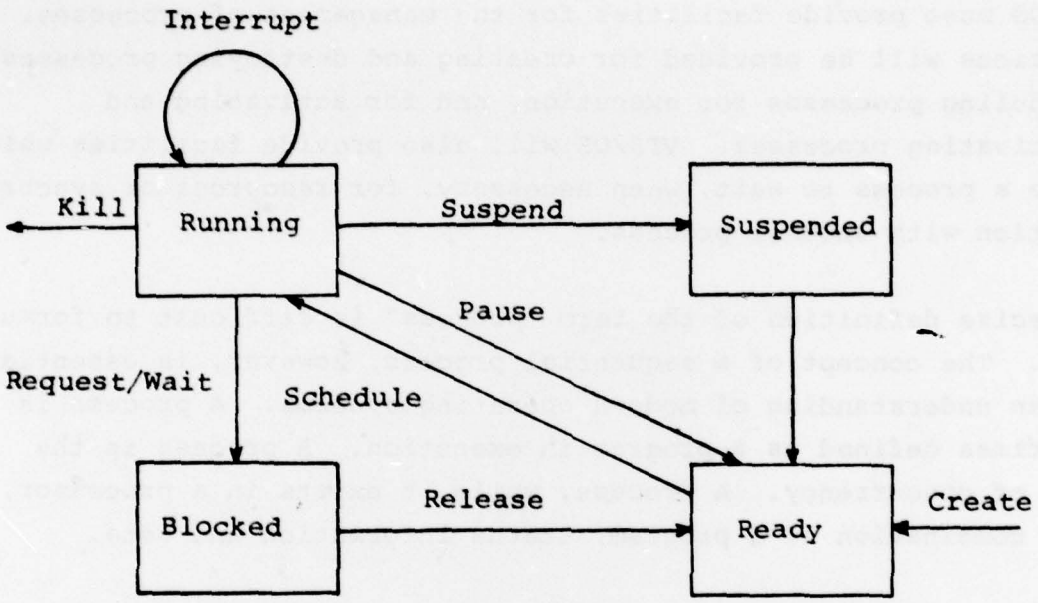


Figure 7-1. Process States

In one processor, one process at a time may be in the running state. In the running state the process is actually executing. The process leaves the running state when it must wait for a resource or a signal from another process. The process also leaves the running state and ceases to exist when it kills itself. If an interrupt occurs, the process leaves the running state temporarily.

A process which is not executing but is ready for execution when a processor is available is in the ready state. Any number of processes may be in the ready state. In VTS/OS, ready processes will exist on a ready queue from which the scheduler will select processes for execution.

A process which is not ready for execution because it is waiting for a resource (other than a processor) is said to be blocked. A process may also be suspended. A suspended process is dormant and will continue to exist in the dormant suspended state until activated by another process.

7.2.2 Scheduler

The scheduler for VTS/OS will control access to the processors. Each processor will have an independent scheduler which will control access to its own processor. Except for communication between and synchronization of processes in different processors, processors in the VTS Processing/Display Subsystem will be completely autonomous.

The scheduler will be a simple routine which will continually cycle. It will remove a process from the ready queue, perform mapping and other operations which may be required to restore the machine state, and jump to the execution address for the process. When the executing process enters a state other than running, control returns to the scheduler which will setup the next process on the ready queue.

This simple scheduler can be used with a number of scheduling strategies which are determined by the structure of the ready queue. A single round-robin scheduling strategy can be implemented by a first-in-first-out ready queue. By using priority as a basis for determining the order of processes on the ready queue, a non-preemptive scheduler can be implemented. Priority, however, must be used with extreme care to ensure that all processes are eventually run, whatever their priorities are. That is, higher priority processes should not be scheduled ahead of lower priority processes so often that lower priority processes never get a chance to be scheduled (or are scheduled only after an inordinately long waiting time).

7.2.3 Process Management Functions

The following functions will be provided for controlling processes. Calling sequences for these functions are given in Section B.1 of Appendix B.

CREATE PROCESS	Creates a process control block and enters the created process on the ready queue.
SCHEDULE	Creates a process and enters it on a timer queue so that it will be made ready at the specified time.
WAIT	Blocks the process until the specified set of event notices has been received.
POST	Sends an event notice to the specified process.

KILL	Allows a process to kill itself or its children.
PAUSE	Temporarily releases the processor.
DELAY	Suspends a process for a specified period of time.
SUSPEND	Suspends a process until activated by another process.
ACTIVATE	Returns as suspended process to the ready queue.
GET ID	Returns the process id of the calling process.

7.3 INTERPROCESS COMMUNICATION

Processes are by nature able to operate independently. Processes which make up a system are seldom completely independent. Normally, processes must communicate with each other and must be able to synchronize operations in which two or more processes participate.

Both the communication of data between processes and sychronization will be essential in the VTS Processing/Display Subsystem. Except in special cases a process will be assigned to manage each major type of data. To access the data, a process will have to communicate with the process that owns that data. This approach leads to a high degree of modularity and lends itself to a multicomputer environment in which a process needing data may not reside in the processor which has the data.

It will also be necessary in the VTS Processing/Display Subsystem to start work on, for example, a watchstander request on one processor under the control of one process and continue work on another processor under the control of another process. This requires both communication and synchronization.

A flexible interprocess communication facility can be used for communication and synchronization, both of which are essential. Indeed we believe that the interprocess communication facility is one of the most significant elements in the design of the VTS/OS.

If interprocess communication facility is efficient and easy to use, the VTS software can be very flexible and can be adapted easily to the various configurations which will be required to handle the range of loads which are anticipated. If, on the other hand, the interprocess communication facility is inadequate or difficult to use, the VTS software could be extremely cumbersome.

7.3.1 Interprocess Messages and Answers

The interprocess communication and synchronization problem can be solved by using explicit message and answer communication. This approach has been used for some time in single processor systems. For VTS/OS we have extended the technique described by Brinch Hansen* and modified it to suit a multicomputer environment.

User processes will have access to five operating system procedures. These procedures are described briefly below and in more detail in Appendix B, Section B.2.

SEND MESSAGE	Causes a message to be sent from one process to another. If the message is long only a message notice is sent.
WAIT MESSAGE	Suspends the caller process until a message is received.
TRANSFER MESSAGE	Physically moves the data if required and returns the address of the message and an answer buffer which has been assigned.
SEND ANSWER	Sends a reply to the process who initiated the conversation.

*Brinch Hansen, P., The Nucleus of a Multiprogramming System, Communications of the ACM, Volume 13, Number 4, April, 1970.

WAIT ANSWER	Suspends the caller until an answer has been returned.

These primatives provide the basis for efficient, convenient interprocess communication and synchronization. SEND MESSAGE and SEND ANSWER provide the communication facility. WAIT MESSAGE and WAIT ANSWER allow for synchronization and TRANSFER MESSAGE allows efficiency by reducing the movement of large blocks of data. TRANSFER MESSAGE also allows us to control data flow so that a particular processor is not overwhelmed by a sudden influx of messages that would overload the buffer space.

7.3.2 Software for Interprocess Communications

The software that will perform the interprocess communications will operate at several levels. At the highest level, the software will interface with the processes by way of the functions described in Section 7.3.1. At the lowest level will be the software that interfaces with the bus hardware.

The group of software modules which include SENDMESSAGE, etc., we will call the message router. The router package will determine the physical destination for communications. The router will maintain a global table which has an entry for each process in the total system. Entries in the global table are the addresses of physical processors in which each of the processes reside. Router routines also maintain a table which includes the address of each local process and its communications status information.

For messages which must be transmitted over the bus, the router routine will attach the destination processor ID. It will then enter the message on a queue which will be serviced by what we have called the dispatcher process. The dispatcher process will handle the task of breaking long messages into packets and attaching the appropriate headers. The dispatcher then calls the bus driver to physically send each packet of the message.

On the receive side the interrupt routines included in the bus driver will receive the transmission and pass it along to the dispatcher process which will reassemble the message and call the appropriate router routines to update message status tables associated with the receiving process and wakeup the process if it is waiting for the message.

The software described briefly above is designed in layers to give good modularity and make it possible to adapt the design if a bus that differs from that described in Chapter 6 is selected for use in the VTS Processing/Display Subsystem.

A more detailed description of the communication software and the operation of the communication system is given in Appendix C.

7.4 RESOURCE ALLOCATION

An operating system is responsible for managing the use of system resources. Resources include memory, I/O devices and in the most general sense also include access to data bases, procedures, and system tables which must be accessed by no more than one process at a time.

The operating system must control its resources in an orderly fashion so that the systems functions can be effectively carried out and deadlock avoided.

7.4.1 Deadlocks

All operating systems which allow processes to share resources must be able to prevent deadlocks or be able to detect and recover from deadlocks. Deadlocks are caused by unrestricted competition by the processes for resources. If processes are allowed to wait for a resource while holding another shared resource, a cycle (i.e., deadlock) may occur in which no process is able to proceed because each is waiting for a resource held by another process in the cycle. Either solution to the deadlock problem, prevention or detection and recovery, requires giving up some portion of the system's resources.

There are two types of shared resources which may cause deadlocks: serially reusable resources and consumable resources. The methods of dealing with the two types of resources are different.

Example 1: Serially Reusable Resources

Process A has a buffer filled with data and is waiting for permission to use the tape to output the data. Process B has received permission to use the tape to input data and is waiting for an empty buffer. The situation is pictured in 7-9a. A suspends itself until the tape is free; B suspends itself until the buffer is free. Each process is holding a resource while waiting for the other process to release a resource.

This example illustrates a deadlock on serially reusable resources. The number of units of the resources (buffers and tape drives) is constant. Each resource unit is either available or allocated to a process. A process may release a resource only if it has been previously allocated to the process.

The operating system can recover from this deadlock by temporarily removing B and reclaiming the resources allocated to B. Process A may then use the tape and release the buffer. After A has finished B may be allocated both the buffer and tape.

Example 2: Consumable Resources

There are three processes in the system; process A, process B, and process C (Figure 7-9b). Each process waits for a message from one of the processes then sends a message.

A: Wait for a message from C.
Then send message to B.

B: Wait for a message from A.
Then send a message to C.

C: Wait for a message from B.
Then send a message to A.

Each process is waiting for a message and unable to continue.

In this example the messages are consumable resources. The number of units of the resource is not constant. The resources are produced by the processor, not allocated to the processes by the operating system. The resources acquired by a process are not returned; they are consumed.

In this example the operating system cannot recover from the deadlock by temporarily removing a process.

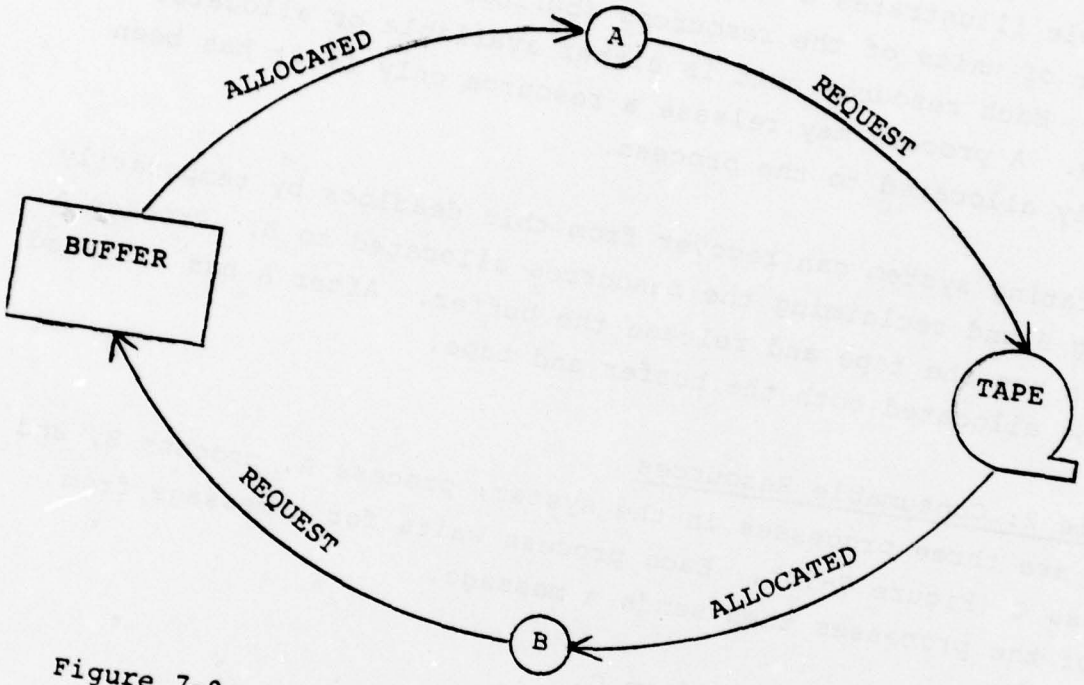


Figure 7-9a Deadlock with Serially Reusable Resources

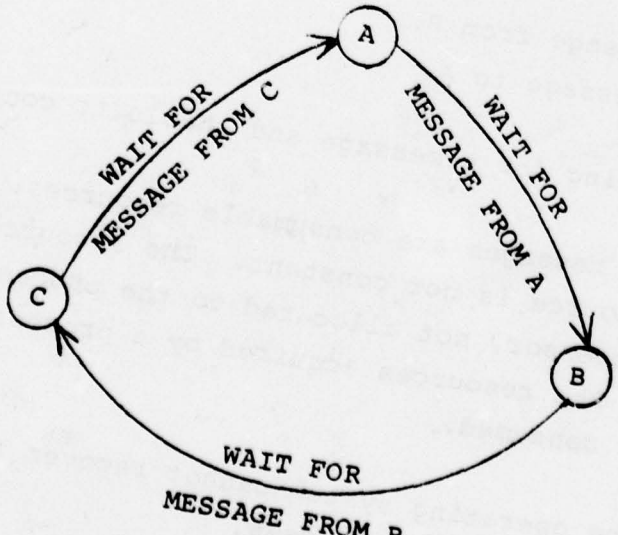


Figure 7-9b Deadlock with Consumable Resources

The VTS/OS will have to manage both serially reusable and consumable resources. We will now look at the steps which will be taken to prevent deadlocks on these resources.

Serially Reusable Resources

Bus

Deadlock on the bus is prevented by dividing bus messages into packets and allocating the bus only for the length of time to send one packet. Packet switching eliminates the possibility of deadlock over bus allocation but introduces the possibility of deadlock over buffer allocation.

Communication Buffers

There are three rules which are enforced by the operating system to prevent deadlocks on communication buffers.

Rule 1: There is a limit to the number of messages a process can send simultaneously.

Each process has a fixed size private buffer area to send messages and receive answers. The size of the buffer area depends on the process's conversational needs. In addition, there is a system buffer pool in each processor for special conversational needs. For example, if a process occasionally has to send a long message, the process will use a system buffer. It would be wasteful to permanently allocate a large buffer which is seldom used. But there is a limit to the number of system buffers a process can have at one time.

Rule 2: There is a limit to the number of messages a process can answer simultaneously.

A process must use a system buffer rather than a private buffer to answer a message. The system buffer is reserved by the process's TRANSFERMESSAGE call and released by the router's Kill Temporary buffer instruction (see Appendix C). There must be a limit on the

number of system buffers a process can reserve at one time. If there is no limit, a process could use all the buffers and prevent all other processes in the processor from answering a message.

The SENDANSWER call will eventually cause the buffer to be released. There cannot be a deadlock on bus allocation; and, the message sender cannot refuse to accept the answer since the buffer was allocated for the answer by the original SENDMESSAGE call.

Rule 3: No process can fill another process's private buffer area with long messages or unsolicited answers.

Only message notices are sent by one process to another. The message is written in the sender's private buffer area rather than the receiver's area.

Other Reusable Resources

Deadlocks can be prevented on the other reusable resources with an ordered resource policy. The resources are divided into k classes, C_1, C_2, ...C_k. A process is permitted to request resources from any class C_i only if it has no resources from classes C_i, C_{i+1} ... C_k. A request for resources from class C_k must be honored eventually since no process with resources from C_k can make any further requests until it releases all of its resources from C_k. It can be shown by induction that all requests will eventually be honored.*

Consumable Resources

Messages and Answers

To prevent deadlocks on messages or answers, there will be a conversational hierarchy. The processes will be arranged in levels, L_1, L_2 ...L_t. A process in level L_j may wait for a message or answer only from a process in levels L_{j+1} ...L_t.

*Shaw, A.C., The Logical Design of Operating Systems, pp. 227-231, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

7.4.2 User Controlled Resources

The operating system will control the resources under its control and prevent deadlocks which could result from the use of those resources. Provision has also been made to allow users to manage the use of the other resources or to manage the use of system resources at a higher level.

As a basic tool in user resource management, a set of functions will be available to define and operate on user defined semaphores. More complex mechanisms can also be setup using the SUSPEND and ACTIVATE functions.

The operating system will not attempt to prevent or overcome deadlocks associated with user resource management. To do so would require elaborate detection mechanisms that would create an unacceptable operating system overhead.

The following summarizes the functions provided for user defined semaphores.

SETUPUCB	Defines a user semaphore or Use Control Block (UCB).
RESERVE	Request a resource.
RELEASE	Releases a resource.
USECOUNT	Returns number of resources or waiters.

7.5 FILE MANAGEMENT

File management is an important factor in an information system such as the VTS Processing/Display Subsystem. The file management system will support a directory of named files and will control the allocation of mass storage to files.

Two types of files will be supported. The major files in VTS will be contiguous files. Storage for a contiguous file is allocated when the file is created. The size of a contiguous file cannot be changed. Access to a contiguous file is efficient since the physical address of a block of data can be calculated by adding the address of the beginning of the file to the relative address of the data block.

The second type of file is a segmented file. A segmented file consists of an index which is used to find the physical address of data blocks given the relative address. Only the index is created when a segmented file is created. As data is added to the file, blocks of mass storage are allocated and their addresses are entered into the index.

Access to all files and I/O devices will reference a channel number. A channel is a logical port to the file or device which is associated when the file or device is opened.

The following is a summary of the file management functions which will be provided for VTS/OS. Additional details are provided in Section B.3 of Appendix B.

CREATE	Create a file. Enters its name in the directory and allocates space to the file.

7.4.2 User Controlled Resources

The operating system will control the resources under its control and prevent deadlocks which could result from the use of those resources. Provision has also been made to allow users to manage the use of the other resources or to manage the use of system resources at a higher level.

As a basic tool in user resource management, a set of functions will be available to define and operate on user defined semaphores. More complex mechanisms can also be setup using the SUSPEND and ACTIVATE functions.

The operating system will not attempt to prevent or overcome deadlocks associated with user resource management. To do so would require elaborate detection mechanisms that would create an unacceptable operating system overhead.

The following summarizes the functions provided for user defined semaphores.

SETUPUCB	Defines a user semaphore or Use Control Block (UCB).
RESERVE	Request a resource.
RELEASE	Releases a resource.
USECOUNT	Returns number of resources or waiters.

DELETE	Deletes a file by removing the name from the directory and releasing the space.
RENAME	Rename a file by replacing its name in the directory.
OPEN	Associates the file (or device) name with a channel number and allow access to begin.
CLOSE	Stops access to the file (or device) and removes the association with the channel number.

7.6 I/O CONTROL

A full set of I/O control functions will be needed to provide access to the I/O devices and files which the VTS Processing/ Display Subsystem will support.

Reference to devices and files will use a channel number which will be associated with the device by an open command (see Section 7.5).

A summary of the I/O functions is included below. Additional details are included in Section B.4 of Appendix B.

READ	Performs a random access read.
WRITE	Performs a random access write.
READ SEQUENTIAL	Reads a specified number of words in sequence.
WRITE SEQUENTIAL	Writes a specified number of words to the next file address or to the device in sequence.
READ LINE	Reads one line of ASCII data.
WRITE LINE	Outputs one line of ASCII data.
READ RECORD	Performs a sequential read of one record.
WRITE RECORD	Performs a sequential write of one record.

READ CHARACTER	Reads one character from the file or device.
WRITE CHARACTER	Writes one character to a file or device.

7.7 ERROR DETECTION, RECONFIGURATION AND RECOVERY

As we indicated in the VTS/OS design goals (Section 7.1), the ability of the system to recognize errors and failures and reconfigure the system when necessary, is extremely important for the VTS Processing/Display Subsystem.

Errors and failure conditions can be detected at various points throughout a system. Operating systems normally recognize errors and failure conditions associated with I/O devices such as tapes or disc units. Much more extensive error and failure detection will be included in VTS/OS. Appendix D contains a description of some particular error conditions which the operating system will recognize.

Error and failure detection will be a part of normal operations. In addition, a background process will periodically perform diagnostics. In each processor, one process will be responsible for assembling error and failure reports. This process will be called the Local Error Reporting Center (LERC).

The LERCs will transmit error reports to a process which will be active in one of the processors called the Main Error Reporting Center (MERC). Whenever possible, error messages will be transmitted over both buses to increase the probability that the message will be received by the MERC.

The MERC will correlate error reports and determine if reconfiguration is indicated. If reconfiguration is needed, the MERC will first perform an extensive self diagnosis to reduce the probability that the MERC itself has failed. If the self-diagnosis is successful the MERC will control the reconfiguration and cause a new global table (see Section 7.3.2) to be transmitted to each of the processors.

Most error and failure reports on which the MERC would base its decisions would come from the operating system functions. Some types of errors, however, require the cooperation of system users to detect. Erroneous data, for example, can be transmitted by a defective processor with proper parity, and in the proper format so that the bus interface and the communications sof are cannot detect the error. The applications process which receives the message, however, may easily detect bad data such as a request for performance of a non-existent function.

The VTS/OS will include two functions which applications can use to report error conditions to the LERC and from there to the MERC. Additional detail on these functions is included in Section B.5 of Appendix B.

BAD DATA	Reports that bad data have been received.
MISSING PROCESS	Reports that a process cannot be contacted.

7.8 OTHER OPERATING SYSTEM FUNCTIONS

A number of other operating systems functions will be needed in VTS/OS. A few of these additional functions have been identified and are summarized below. Appendix B includes a more detailed description of these functions.

DEBUGGING FUNCTIONS

HALT	Halts the running program and produces a dump.
TRACE	Starts recording of a list of the last "n" processes executed.
ENDTRACE	Prints the list recorded since TRACE call.

OVERLAY CONTROL

LOAD OVERLAY	Reserves overlay area and loads requested program when area available.
RLSE OVERLAY	Releases control of the overlay area.
NEXT OVERLAY	Loads overlay requested without freeing area. Allows chaining of overlays.

TIME AND DATE FUNCTIONS

TIME	Returns the current system time.

DATE	Returns the current system date.
SET TIME	Sets the system clock.
SET DATE	Sets the current system date.

8 PRELIMINARY SOFTWARE DESIGN

Initial design of a software system involves the decomposition of the software into major elements. For a real time system the initial design results in the definition of a number of concurrent processes that together will perform the functions of the system.

There are no established techniques for performing the initial decomposition of a system. Composite design and related techniques provide methodologies for design of a program by top down techniques which lead to a hierarchical structure. As Myers* has indicated, however, these techniques are not applicable to the initial phase of design in which we, in effect, must determine what programs will make up the system.

For the VTS Processing/Display Subsystem we must select a set of processes which can perform all the functions of the subsystem. We must also select processes which can perform their functions effectively in a multiple computer environment. To accomplish this objective we will select various kinds of processes and assign them to processors so that each process will reside in one processor and communicate with other processes by explicit messages**.

---

* Myers, G. J., Reliable Software Through Composite Design, Petrocelli/Charter, New York, 1975.

** It is theoretically possible to allow processes to move from one processor to another. Such an approach is unnecessary, however, and leads to significant unneeded complexity and overhead.

8.1 TYPES OF PROCESSES

All processes are similar in many respects. Associated with each process is a program which need not be unique to the process. Also associated are data and status information which is unique to the process and defines the current state of the process.

In a multiprogramming system, each process is capable of operating concurrently with other processes. Concurrent operation implies that the processes share a processor so only one process at a time can actually be executing, otherwise they act as though they were executing in parallel. In a multicomputer environment processes in different processors may actually operate in parallel.

Processes differ in a number of ways, however. Processes differ primarily in terms of the functions they perform and their longevity.

To start the system, an initialization process is required. Its function is to initialize data and status information and create the major system processes.

Three major types of processes are created. The first, which we will call primary processes, react to external stimuli. In the VTS Processing/Display Subsystem, the processes which provide the interface to the watchstanders are primary processes.

A second major type of process we will call service processes. These processes do not interface with the outside world. Their only purpose is to provide services that are needed by other processes. An example of service processes will be the processes that manage the retrieval of data from files.

A third major type of process normally performs monitoring functions. These processes operate in a cyclic fashion. They may be clock driven or may schedule themselves based on the system clock. Hazard detection processes will be cyclic processes in the VTS Processing/Display Subsystem.

Primary, service and cyclic processes are all essentially permanent processes. They are normally created at system initialization and continue indefinitely. Temporary processes may also be created when it is desirable to create additional concurrent paths. Temporary processes are created to perform a particular function and exist only for the period of time required to carry out that function.

Before discussing the preliminary software designs, we should also note one other way of categorizing processes. Some processes are unique. Other groups of processes which could be called families of processes perform identical functions. Members of a family of processes perform their functions for different sets of inputs or for different users. For VTS, the processes that interface with the watchstanders form a family. Each process uses an identical program and is capable of performing precisely the same functions. Each, however, supports a different watchstander.

In the following sections these types of processes will be used in describing the top level design of the software for the VTS Processing/Display Subsystem.

8.2 OVERVIEW OF SOFTWARE DESIGN

An overview of the software design is shown in Figure 8-1. Included are major groups of processes and two major inter-processes communications paths. Groups of processes are assigned to either a display station processor or to the main system processor. Additional processors will be included in the largest VTS configurations which will require further separation of the processes. We will ignore this additional complexity for the moment until we have discussed the configuration that will be adequate for the vast majority of VTS installations.

8.2.1 Display Station Processes

The display station processor will include three major processes. One of these processes will provide the primary interface between the system and the watchstander and is called Transact Watchstander Request. Transact Watchstander Request accepts inputs from the watchstander and responds to those inputs. If the function cannot be handled entirely by the Transact Watchstander Request process, messages will be sent to other processes which can perform the required operation.

Two other processes are required. One will manage the map display. The other will manage the alert display. The Manage Alert Display process will receive messages from the Post Alerts process in the main processor. It will also receive messages from the Transact Watchstander Request process.

The Manage Map process will communicate with a number of other processes which will supply up-to-date information to be displayed.

The Transact Watchstander Request process will be a primary process while the Manage Map and Manage Alert Display processes will be service processes. Each of these processes will be a member of a family of processes with a set of these processes for each display station.

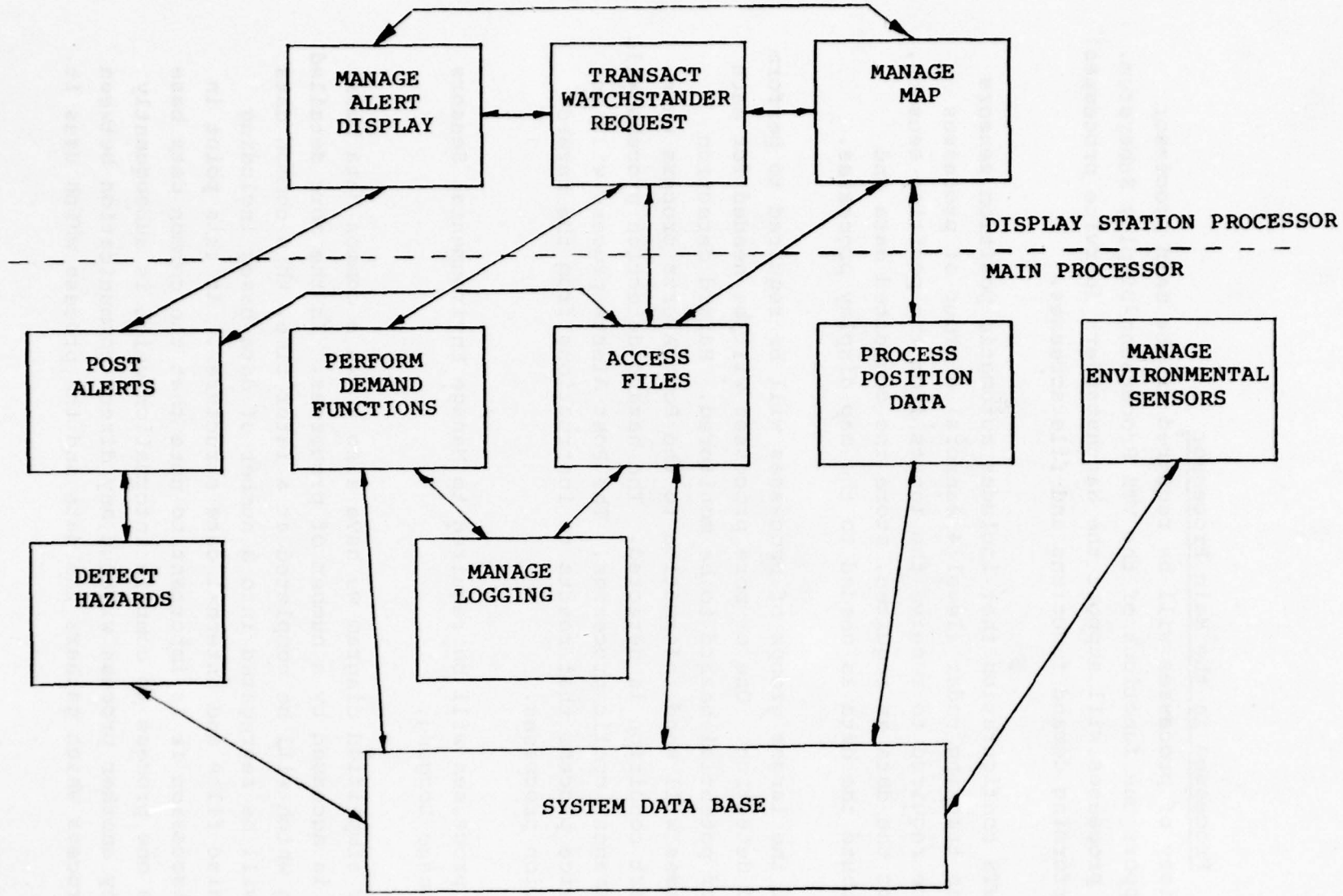


Figure 8-1. Major Groups of Processes

8.2.2 Processes in the Main Processor

A variety of processes will be required in the main processor to support the functions of the VTS Processing/Display Subsystem. These processes will support the Watchstanders (service processes) by performing demand functions and file accesses.

In a VTS configuration that includes automatic position sensors such as tracking radar (level 4 sensors) a group of processes will be required to receive the inputs from the position sensors, convert the data as required, store the converted data and distribute the data as needed to the map display processes.

One of the larger groups of processes will be required to perform hazard detection. One or more processes will be needed for each type of potential hazard to be monitored. Hazard detection processes will send information to the Post Alerts process when an alert condition is detected. The hazard detection process will be permanent cyclic processes. The Post Alerts process will be a service process that reacts to instructions from the hazard detection processes.

Other processes will be required to Manage Environmental Sensors and Manage Logging.

In our simplified diagram we have also shown a common data base which is accessed by a number of processes. In the more detailed design which will be completed at a later time, this common data base will be segregated into a number of data bases including both disc files and internal data structures. At this point in the discussion it is important to note that the common data base allows one process to compile information which is subsequently used by another process without any direct communication between the process which gathers the data and the process which uses it.

8.3 DISPLAY STATION PROCESSES

Three major application processes will be required in each of the display stations. These processes, including Transact Watchstander Request, Manage Alert Display and Manage Map were discussed briefly in Section 8.2.1. Additional detail on these processes is given below. For each process we will discuss the inputs or stimuli that will cause the process to perform its function, the outputs or responses the process generates and the nature of the processing which it will perform.

8.3.1 Transact Watchstander Request

One member of the Transact Watchstander Request family of processes will reside in each display station processor. By way of the operating system, Transact Watchstander Request receives inputs from the Watchstander's standard and function keyboards. Following the entry of a function key or function identifier, the function will be validated and the appropriate procedure will be called to handle the specified function.

The particular procedure will display appropriate forms and/or cues to the Watchstander. Entries by the Watchstander are received and verified. Whenever required, the Transact Watchstander Request process sends messages to other processes in the display station or to processes in the main processor. It then waits for the function, such as a data base inquiry, to be performed and an answer sent. It then formats and outputs the appropriate information.

When the procedure is completed, the process again waits for an input of a function key or function identifier from the Watchstander.

A partial list of the functions which the Transact Watchstander Request must support includes:

- Log On
- Log Off
- Display/Enter/Modify/Delete Vessel Record
- Identify Vessel
- Display/Enter/Modify/Delete Passage Record
- Update Vessel Position
- Enter Communication
- Change Status
- Get Local Traffic
- Display/Enter/Modify/Delete Environmental Data
- Search on a Key
- Display/Enter/Modify/Delete Notice
- Display/Enter/Modify/Delete Forecast
- Display/Enter/Modify/Delete Call Data
- Display/Enter/Modify/Delete Route Segments
- Respond to Alert
- Determine Encounters
- Compute Relative Position
- Compute CPA
- Route/Schedule Vessel
- Change System Parameters (Allowed to Watch Supervisor only).

The Transact Watchstander Request processes will be permanent primary processes.

8.3.2 Manage Alert Display

The Manage Alert Display process will manage a list of alerts which have been sent to it for display to the watchstander. It is essentially a service process which reacts to messages from the Post Alerts process (Section 8.4.5) and from the Transact Watchstander Request process (Section 8.3.1).

8.3.3 Manage Map

The Manage Map process is assigned the task of keeping the map display updated. It receives position update information on each vessel in track every six seconds. It also receives periodic updates on vessels which are being dead reckoned.

It also receives control messages which result from watchstander entry or other system processes which must result in a change in what is displayed or how it is displayed. A vessel which has been lost from track, for example, will continue to be displayed with a blinking symbol or at its last known position until the watchstander changes its status.

8.4 PROCESSES IN THE MAIN PROCESSOR

As we indicated earlier, there are a variety of processes required in the main processor. Two major types of processes will be included: Service processes such as those that perform file access and demand functions on request from the Transact Watchstander Request process (Section 8.3.1); and independent processes such as the hazard detection processes.

The processes which support the watchstander can be organized in two quite different ways. One approach would assign one or more processes to each of the watchstanders. The resulting family of processes would each perform identical functions except for serving a different watchstander. If programs were reentrant all code could be shared.

The approach we have chosen assigns processes on a functional basis. One or a family of processes will be assigned to manage accesses to each major file or to perform a major function. This functional design of processes will, we believe, reduce the complexity of the software and simplify the task of utilizing additional processors for the largest VTS configurations.

For each of the process discussed below we will describe the inputs it receives, the processing it performs and the outputs it generates.

8.4.1 Access Files

Service processes will be provided in the main processor to handle display (retrieve), enter, modify, and delete record functions for the system files. For nearly all the system files a single process is more than adequate to handle the number of accesses to be handled per second, even in the largest configurations. It may be desirable to provide two or more processes for those functions associated with the passage file in the largest, most demanding VTS configuration, however.

8.4.1.1 Access Vessel File

The Access Vessel File process will receive messages from the Transact Watchstander Request process in each display processor (Section 8.3.1). These messages will instruct the Access Vessel File process to enter a new vessel record, retrieve a vessel record for display, retrieve a vessel record for potential modification, replace a vessel record with an updated copy, or delete a vessel record. These operations will support the display, enter, modify and delete vessel functions.

The Access Vessel File process will maintain a list of vessel records which are in use by each watchstander. This will allow a watchstander to be notified in the unlikely event that another watchstander is already using the record when a request is made. A watchstander can also be notified if another watchstander has altered the record during the time he was attempting to alter it.

The Access Vessel File process will maintain the indexes to the vessel file. The vessel file will contain a fixed maximum number of records. The Access Vessel File process will be responsible for assigning unused records when a new vessel is entered. If the file is full, the process will select a vessel which has not been accessed recently, automatically delete that record and assign the record space to the new vessel.

When the process has completed the requested operation it will send an answer to the requesting process indicating that it was successful or unsuccessful and the reason if unsuccessful. If a retrieval was requested and was successful, the retrieval record will also be forwarded to the requesting process.

8.4.1.2 Access Passage File

The Access Passage File process performs essentially the same function as the Access Vessel File process except that it deals with the passage file rather than the vessel file. It receives messages from the Transact Watchstander Request process which instruct it to enter a new passage record, retrieve a passage record for display, retrieve a passage record with intent to update, replace a passage record with an updated copy or to delete a passage record.

These operations will support the passage file functions including:

- Display Passage Data
- Enter Passage
- Update Vessel Position
- Modify Passage Information
- Enter New Communication
- Delete Passage
- Change Status
- Identify Vessel
- Update Vessel Position

The Access Passage File process will manage the allocation of free space in the Passage File and assure that the most critical vessels are included in the Passage File by automatically deleting less critical vessels if need be to make space available. Normally, however, space should be available since space is made available regularly when vessels leave the coverage area. In addition, the passage file should be sized to handle more than the maximum anticipated number of vessels.

The Access Passage File process will maintain the indexes to the passage file and will maintain a list of Watchstanders who are modifying passage records so that notification can be given when needed.

When the process has completed the requested operations it will send an answer to the requesting process indicating that it was successfully or unsuccessfully completed. If a passage record was successfully retrieved it will be included with the answer.

8.4.1.3 Search on a Key

The Search on a Key process receives input messages from the Transact Watchstander Request process (Section 8.3.1). A message will include the identification of the file to be searched and the logical combination of keys to be used. For example, the message might request a search of the vessel file for all vessels over 300 feet long which have been in the VTS coverage area within the last month.

The Search on a Key process will scan the specified file and create an answer which includes those records which match the search criteria. The search operation will read large blocks of the file and perform a sequential search of each block.

8.4.1.4 Access Environmental Information

The Access Environmental Information process will receive input messages from the Transact Watchstander Request process (Section 8.3.1) and from the Manage Environmental Sensors process (Section 8.4.7). It will control the entry, updating and retrieval of both manually input and automatically collected environmental data.

8.4.1.5 Access Waterway Characteristics Files

The Access Waterway Characteristics Files process will receive messages from the Transact Watchstander Request process (Section 8.3.1). Messages will include requests to enter, retrieve, replace, or delete records of cell data, notices, route descriptions, waypoints, docks and piers, navagational aids and other data related to the characteristics of the waterway. A single process is required to handle these functions since they are infrequently utilized. (Other functions access the waterway characteristics file, but they have their own dedicated process to perform this access. This function is for operator invoked accesses only).

The Access Waterway Characteristics Files process will perform the requested operations and formulate an answer which will report success, report errors or return the requested data.

8.4.2 Demand Functions

In addition to the functions described in Section 8.4.1 which enter and retrieve information from the system, other functions will be provided which perform calculations, analysis and other processing based on information stored in the VTS Processing/ Display Subsystem.

If the information is readily available in the display station processor, the Transact Watchstander Request process (Section 8.3.1) will perform the function itself. CPA calculations and relative position determinations, for example, can normally be done by the Transact Watchstander Request process. Other functions will be performed by individual processes in the main processor.

8.4.2.1 Find Local Traffic

The Find Local Traffic process receives messages from the Transact Watchstander Request process (Section 8.3.1). A message includes a vessel and a radius. An answer is formulated which includes all vessels within the specified distance from the vessel.

The Find Local Traffic process will examine the current position of each vessel known to the system. The algorithm used for stage 1 of collision processing (Section 8.4.4.1) should be used to make a rapid check of the differences in x and y coordinates. This effectively screens out all vessels which are not within a square area surrounding the designated vessel. An actual distance calculation can then be used for vessels which are not rejected by the initial screening operation.

8.4.2.2 Determine Encounters

The Determine Encounters process receives a message from the Transact Watchstander Request process (Section 8.3.1) which includes a vessel and a lookahead time span. The location, course and speed of other vessels within the VTS coverage area will be compared against that of the specified vessel. An answer will then be formulated which includes all vessels which the specified vessel will meet, cross or overtake during the look ahead time.

8.4.2.3 Route/Schedule Vessel

The Route/Schedule Vessel process will receive an input message from the Transact Watchstander Request specifying a vessel identifier and proposed route and schedule. It will formulate an answer indicating that the proposed route or schedule is acceptable or that potential problems have been detected. If problems are detected, the subsystem will attempt to adjust the schedule or select an alternate route which will be included with the answer.

In a large VTS configuration, a substantial amount of data may have to be analyzed to perform the routing and scheduling function in spite of the fact that no elaborate optimization procedures will be required. If this function is used extensively, a family of processes will be needed to allow multiple vessels

to be scheduled nearly simultaneously. For the largest of the configurations we have analyzed, a processor and a disc are effectively dedicated to this function.

8.4.2.4 Adjust System Parameters

The Adjust System Parameters process will receive message from the Transact Watchstander Request process which is interacting with the Watch Supervisor. The messages will include a designation for the parameter(s) to be changed and the new value(s) to assign. The parameters are used to define and adjust the operation of the system. Included are parameters which define the rate at which hazard detection processes operate and parameters which define constricted areas.

To allow maximum flexibility an extensive set of options will be available to the Watch Supervisor. Since a considerable amount of software may be needed for this process and since only one individual at a time may be using these functions, it may be appropriate to overlay the programs which support this process.

When the function has been completed, an acknowledgment message (answer) will be returned to the Transact Watchstander Request process.

8.4.2.5 Setup/Edit Simulation Scenario

The Setup/Edit Simulation Scenario will receive messages from the Transact Watchstander Request process requesting that the appropriate functions be performed. The functions will allow a new scenario to be setup, real data to be recorded in the scenario and records for the simulated data base to be entered, modified or deleted.

Simulation will be used for training operations personnel and for exercising and testing the system under artificial conditions.

Answers will be returned to the Transact Watchstander Request process indicating the completion of the requested operation and including retrieved simulated data when appropriate.

8.4.3 Position Processing

Four processes or families of processes have been identified to perform the functions related to position sensors including the processing required to support tracking radar.

8.4.3.1 Convert Radar Data

A family of interrupt level processes will be provided to interface with radar tracking units. These processes will receive the data as it is transmitted by the radar units and queue the data for processing by the Convert Radar Data process.

The Convert Radar Data process will perform coordinate conversions and update the main memory tables which contain current location, course and speed of all tracker vessels. Status information included with the radar input will be checked for error, dropped contract, new contact, or merged contact flags.

The Convert Radar Data process will attempt to match new contacts with vessels already known to the system such as a vessel entering the radar area from a Level 1 area. Merged tracks will also be processed by keeping a record of merged vessels and attempting to match with new contacts.

When the Convert Radar Data process has completed conversion, storage and analysis of the data, a message will be sent to the Distribute Position Data process for distribution to display processors.

8.4.3.2 Distribute Position Data

The Distribute Position Data process will receive messages from the Convert Radar Data process and send update messages to appropriate manage map processes.

If the interprocessor bus supports broadcast messages, the Distribute Position Data process can send a single message to all processors simultaneously. If a braodcast capability is not available, the Distribute Position Data process will send update messages in sequence to each appropriate Manage Map process.

8.4.3.3 Manage Position Table

The Manage Position Table process handles access to the position tables. It receives messages from other processes requesting current position data. An answer message will be formulated including the requested data.

This process is used to provide information hiding and flexibility. Critical hazard detection processes will bypass the Manage Position Data process and use the data already, however, to reduce the overhead associated with the use of position data.

8.4.3.4 Handle Position Sensors

The Handle Position Sensors process will support acoustic and/or magnetic sensors when these are included in the system. These sensors are capable of detecting that a vessel is passing its position.

The Handle Position Sensors process will compare the location of the sensor against vessels that are expected to be passing that point. Position information will be updated for vessels that can be uniquely identified.

8.4.4 Detect Hazards

Processes will be included to attempt to detect potentially hazardous situations and provide information (alerts) which will allow the vessel or vessels involved to be notified so the hazard can be avoided.

The VTS Processing/Display Subsystem will have the capability of detecting the following hazards:

- Potential Collision
- Lane Stray
- Route Stray
- Potential Grounding
- Excessive Congestion
- Dangerous Encounters
- Anchor Drift
- Navaid Adrift/Missing
- Excessive Vessel Speed

The task of the software is to recognize these conditions and report the potential hazards to the watchstanders as alerts. In order to recognize these conditions, the software must monitor the data on the location, course, and speed of vessels and navaids.

Recognizing that processing resources will be required to perform these functions, a maximum frequency of execution is given for the monitoring software for each function in the discussion which follows. Each of the required processes will be a cyclic permanent process. The time cycle on which each will operate will be controllable by the Watch Supervisor (see Adjust System Parameters, Section 8.4.2.4).

Ideally the subsystem would generate an alarm only when a truly hazardous condition exists. A variety of normal conditions can, however, be cited which would result in an alarm. To prevent these false alarms the subsystem will also allow the watch supervisor to exempt any particular physical area(s) from any or all of these hazard checks. This capability may be used, for example, to prevent false collision warning alerts in narrow channels where vessels pass so close together, even in a normal passing situation, that an alert would normally result.

Additionally, the subsystem will allow watchstanders to exempt particular vessels from any or all of the hazard checks. This capability may be used to avoid false collision alarms for pilot boats, tugs, etc., which have intentional "collisions" as a part of their every day operation, or to prevent lane stray alarms for ferries, etc., which are not normally constrained to lanes.

8.4.4.1 Hazard Detection Monitor Process

The Hazard Detection Monitor process will control the time schedule for the hazard detection processes. It will base its functions on the time schedule set by the Watch Supervisor and send messages to trigger individual hazard detection processes. When each process completes its cycle it will send an answer to the monitor process which will record its completion.

If processes fail to complete in the scheduled time, the monitor process will generate error messages indicating that the system has degraded. If the subsystem is operating in a degraded mode the Watch Supervisor will have the option of adjusting the time schedule or the relative priorities assigned to each type of hazard detection process.

8.4.4.2 Predict Collisions

The subsystem will be required to detect potential collisions if sensors are available which can determine the location, course and speed of the vessels with a high degree of accuracy.

To detect potential collisions will require a periodic check on each pair of vessels in the area covered by Level 4 or 5 sensors. Depending on the algorithm used, this processing required can be very large since for n vessels, n(n-1)/2 checks must be made.

To minimize the processing required, three conditions, which are successively more selective have been specified. A separate process will be used to monitor each of these three conditions.

Stage 1 Process

The Stage 1 process will check each pair of results in the VTS coverage area every 30 seconds (minimum cycle time).

The Stage 1 process is initiated by a message from the Hazard Detection Monitor process. The first step in the Stage 1 processing is to formulate the collision table. The collision table will contain vessel position, course and speed data from the tracker tables. Vessel data is extracted from the tracker tables so that only vessels need be examined.

An efficiently coded loop will be used to compare the coordinates of each pair of vessels. If the difference in the X or Y coordinates is less than the selectable tolerance,

additional checking will be performed to determine if the vessel pair is exempt from collision processing. A vessel pair will be exempt if neither is identified, both are anchored or docked, one or both vessels has been marked exempt, one or both vessels is in a designated exempt area, or one or both vessels is not in track.

If the vessel pair is not exempt it will be entered into the Stage 2 list for further checking by the Stage 2 process.

Stage 2 Process

The Stage 2 process checks each vessel pair entered in its list by the Stage 1 process every 15 seconds (minimum cycle time). For each vessel pair in the list, the most recent radar input data will be used to calculate CPA and time to CPA. If CPA and time to CPA are both less than the values set by the Watch Supervisor, the vessel pair will be entered into the Stage 3 list for checking by the Stage 3 process.

Stage 3 Process

The Stage 3 process analyzes vessel pairs which have been entered into its list by the Stage 2 process. The minimum cycle time will be 6 seconds. Processing includes an estimation of collision risk based on:

- CPA
- The rate that the CPA is increasing or decreasing (dCPA)
- The time remaining until CPA is reached (tCPA)
- The approximate sizes of the vessels involved (one of four sizes for each vessel)
- The relative hazard of the cargos aboard.

The risk is estimated using these factors in the following equation:

$$\text{Risk} = C_4 \frac{(S\text{-}CPA)}{tCPA} - C_5 \frac{dCPA}{tCPA} + H$$

where C_4 and C_5 are weighting constants for their associated factors. "S" is the vessel size risk factor which has one of 10 values based on a table of risk values for the ten possible combinations of the four vessel size categories. "H" is the risk value based on the hazard of the cargo. Up to 20 cargo types will have values associated with them in the data base. "H" will equal the value of the most hazardous cargo aboard a vessel, or if both vessels are carrying hazardous cargo it is the sum of the values for the most hazardous cargo aboard each vessel. The tables of size and cargo risk values are data base constants accessible by the Watch Supervisor.

The third stage process will generate an alert message to the Post Alerts process when the risk value exceeds a threshold set by the Watch Supervisor and has exceeded it for more than an adjustable time period. The time limit required before an alert is issued is included to reduce the probability of false alerts being generated. When the risk value drops below the established threshold for longer than the time limit, the third stage process will send a cancellation message to the Post Alert process.

8.4.4.3 Detect Lane Stray

The Detect Lane Stray process will cycle once every 30 seconds (minimum cycle time). All vessels which are in

track by Level 4 or 5 sensors and are following a lane will be checked to determine if the vessel is outside the proper lane of travel or will be outside the lane in "n" minutes based on straight line dead reckoning. The constant "n" will be a data base constant accessible by the Watch Supervisor.

If the vessel is outside the assigned lane or will be, an alert message will be sent to the Post Alert process.

As with other hazard detection processes, the Detect Lane Stray process will be initiated by a message from the Hazard Detection Monitor process. When it has completed its cycle it will send an answer to the monitor process.

8.4.4.4 Detect Route Stray

The Detect Route Stray process will cycle at a rate not faster than once every 30 seconds. It will be initiated by a message from the Hazard Detection Monitor process and will generate an answer when it completes its cycle. Each identified non-exempt vessel's reported location will be compared with its intended route. A route stray alert will be issued if a vessel is in track by Level 4 or 5 sensors and the reported position is not within a predefined distance from any route segment of a vessel's prescribed route.

A route stray alert will also be generated by the Access Passage File process (Section 8.4.1.2) when an Update Vessel Position message is received indicating that a vessel within a Level 1, 2 or 3 area is not on its intended route. This additional type of route stray alert will not in anyway affect the Detect Route Stray process, however.

8.4.4.5 Predict Grounding

The Predict Grounding process will check each identified vessel for potential grounding at a rate not faster than once every 30 seconds. The potential grounding check will be based on:

- The reported draft of the vessel.
- The depth of water at MLLW in the route segments or cells through which the vessel is expected to pass.
- The present water level relative to MLLW from water level sensors, tide schedules or manual entry.

The potential grounding detection process will dead reckon each vessel ahead up to a preset amount of time (as selected by the Watch Supervisor but not longer than 10 minutes) and verify that the actual water level on its intended route over that time period (i.e., the depth at MLLW plus the height of water above or below MLLW) exceeds the draft of the vessel. If a vessel is following a route structure this dead reckoning will be based upon its reported speed and its intended route, and will consider the depth of the appropriate route segments. If, however, the vessel is not following a route structure, the dead reckoning will revert to straight line dead reckoning based on the vessel's current course and speed, and will use for the grounding determinations the depth of that portion of the waterway which the dead reckoned track will traverse.

If grounding is projected on the current course of the vessel, an alert message will be sent to the Post Alerts process.

When each cycle of the grounding detection process is completed a completion message will be sent to the Hazard Detection Monitor process. The Predict Grounding process will then wait for another message from the monitor process before repeating the cycle of grounding checks.

8.4.4.6 Detect Excessive Congestion

The Detect Excessive Congestion process will detect when more vessels are in or will be in a critical area than the area can safely handle. The checks will be repeated at a rate not faster than once every 30 seconds. Up to 40 critical areas will be defined by a center point and a radius distance (in a two dimensional harbor representation) or by waypoints and linear distances along routes from waypoints (in a one-dimensional representation). Each critical area will be assigned a maximum capacity value. The amount of actual congestion will be determined using the technique defined below which yields a congestion value. If this value will exceed the critical area's maximum capacity within a pre-established look-ahead time span (not greater than 30 minutes), a congestion alert message will be sent to the Post Alerts process.

The look-ahead time span will be a data base constant accessible by the Watch Supervisor station. The look-ahead will be accomplished by advancing all vessels which are in a route structure along their intended route at their reported (or measured) speed. Vessels outside a route structure will not be considered. The value each vessel contributes toward this total capacity will be dependent upon its size and cargo risk value. A table of capacity contribution constants for

each of four vessel size categories will be stored in the data base and be accessible by the Watch Supervisor station. The capacity contribution resulting from the various hazardous cargoes will be the same as the 20 values established for these cargoes in the potential collision hazard detection process described earlier. The actual congestion of an area will be the sum of the size and cargo capacity contribution constants from each vessel in the critical area.

As with the other hazard detection processes, the Detect Excessive Congestion process will be initiated by a message from the Hazard Detection Monitor process. When it has completed its checks it will send an answer to the Hazard Detection Monitor process.

8.4.4.7 Predict Dangerous Encounters

The Predict Dangerous Encounters process is used to detect when vessels are approaching a passing or overtaking in a channel too narrow to safely handle it, or when a vessel will be crossing the channel too close to an oncoming vessel in the channel. The location of each end of up to 20 constricted areas will be specified in the data base. A preset time interval before a vessel will enter any constricted area, the subsystem will verify that the constricted area will be free of opposing traffic of sufficient size to be hazardous. Also, whenever a vessel is about to cross the channel, the movement will be accomplished at least a preset amount of time before or after any vessel in the channel arrives at the crossing location.

Each of the preset times mentioned above will be data base constants accessible to the Watch Supervisor. Also in the

data base for each constricted area are specifications of the size combinations of vessels (one of four sizes for each vessel) which constitute a dangerous encounter. The Predict Dangerous Encounters process will cycle at a rate not faster than once every 30 seconds. The rate, determined by the Watch Supervisor defined data base value, will be controlled by the Hazard Detection Monitor Process.

8.4.4.8 Detect Anchor Drift

The Detect Anchor Drift process will check the present measured position of all identified anchored vessels within the coverage of Level 4 or 5 sensors. An anchor drift alert message will be sent to the Post Alerts process if a vessel moves outside the specified swing radius from its assigned anchorage location. This process cycles at a rate not faster than once per minute.

8.4.4.9 Detect Buoy Adrift or Missing

The measured position of each buoy designated by Watch Supervisor which is within the coverage area of Level 4 or 5 sensors will be compared to its normal position. The normal display symbol for a buoy is replaced with a special symbol when Level 4 or 5 sensors lose track of a specified buoy (buoy missing), or if a buoy in track is outside its specified watch circle (buoy adrift). It will inform the Manage Map process of the need for a special symbol. Processing to detect a buoy adrift/missing will be repeated at a rate not faster than once every minute, as determined by the message from the Hazard Detection Monitor process.

8.4.4.10 Detect Excessive Vessel Speed

The Detect Excessive Vessel Speed process will check the measured speed of vessels in all route segments and "cells" for which a maximum speed has been specified. If a vessel is proceeding at a speed "x" knots above the limit (where "x" is a data base constant accessible by the Watch Supervisor station) an alert will be issued. This check will be performed at a rate not faster than once every minute and will be controlled by messages from the Hazard Detection Monitor process.

8.4.5 Post Alerts

The Post Alerts process receives messages from the hazard detection process described above. Alert messages will be forwarded to appropriate Manage Alert Display processes in the display station processes. The Post Alerts process will be responsible for monitoring a master alert queue.

8.4.6 Logging Management

The Logging Management process will provide support for the Logging functions carried out by the subsystem. A family of process will be used with one member supporting each type of Log.

The Logging process will receive messages from processes wishing to enter data into the Log. The Logging Management process will enter the data into a buffer and call the operating system to write them to the Logging medium when the buffers are full.

8.4.7 Manage Environmental Sensors

The Manage Environmental Sensors process will receive data from environmental sensors and enter the data into the system data base.

8.4.8 Playback Simulation Scenario

An additional process will be included to control the playback of a simulation scenario. The scenario data will be read from the data bases and appropriate data will be sent to other processes to allow the playback process to operate as a source for messages normally generated by other operational processes.

8.5 DATA BASES

Information storage and retrieval are central to the VTS Processing/Display Subsystem. A portion of the information is automatically acquired by sensors such as radar. The balance of the information is entered by watchstanders.

A portion of the data requires extremely rapid access and will be stored in main memory. Moving head discs will be used to store data which is less frequently accessed or must be permanently recorded.

In a Level 4 or 5 system every six seconds, automated sensors will provide updated information on the location, course and speed of tracked vessels. This data is used by many of the hazard detection processes and will be kept in main memory. Other critical data for hazard detection such as the draft of vessels and their cargo hazard value will also be stored in main memory.

Vessel, passage, environmental and waterway characteristics data will all be stored on moving head disc files.

In the previous discussion of the system software both files stored on disc and files maintained in main memory were collectively referred to as the system data base. In the discussion which follows the term data base is used primarily in the more limited sense of disc based data.

8.5.1 Data Base Design Concepts

The data base will be designed to emphasize both simplicity and efficiency because of the real-time nature of the system and because of the response time requirements set forth in the functional specification. We do not foresee using a complete data

base management system (DBMS) because of the processing overhead associated with separating the physical and logical data files. Instead, we propose making the logical and physical files identical, and designing data access methods appropriate for each file, taking into consideration the number of keys utilized, as well as response time requirements.

Maximum retrieval efficiency is not the primary goal. The VTS data base consists of portions which are relatively static. Where efficiency is a factor, however, the files are dynamic. Records are added, deleted and modified frequently. Access mechanisms must be designed therefore to satisfactory response times for all these operations.

At this stage of the design it is important to define the information content of the files so that requirements for mass storage can be verified. The precise data formats can be resolved during final detailed design. At that point the hardware will have been selected and the precise sector size will be known.

8.5.2 Access Methods

A number of techniques could be used for accessing files for the VTS Processing/Display Subsystem. The technique we suggest for files which cannot be addressed directly is based on a multi-level index structure. This access mechanism is straightforward and its performance is easily predicted. It also is suitable for files from which records are frequently deleted. Some other access methods such as hashing are excellent for efficient retrieval but are less desirable when records must be deleted.

A multilevel index structure is shown in Figure 8-2. At each level of index, the keys will be in ascending alphanumeric order. At all levels except the lowest level, the entries will indicate

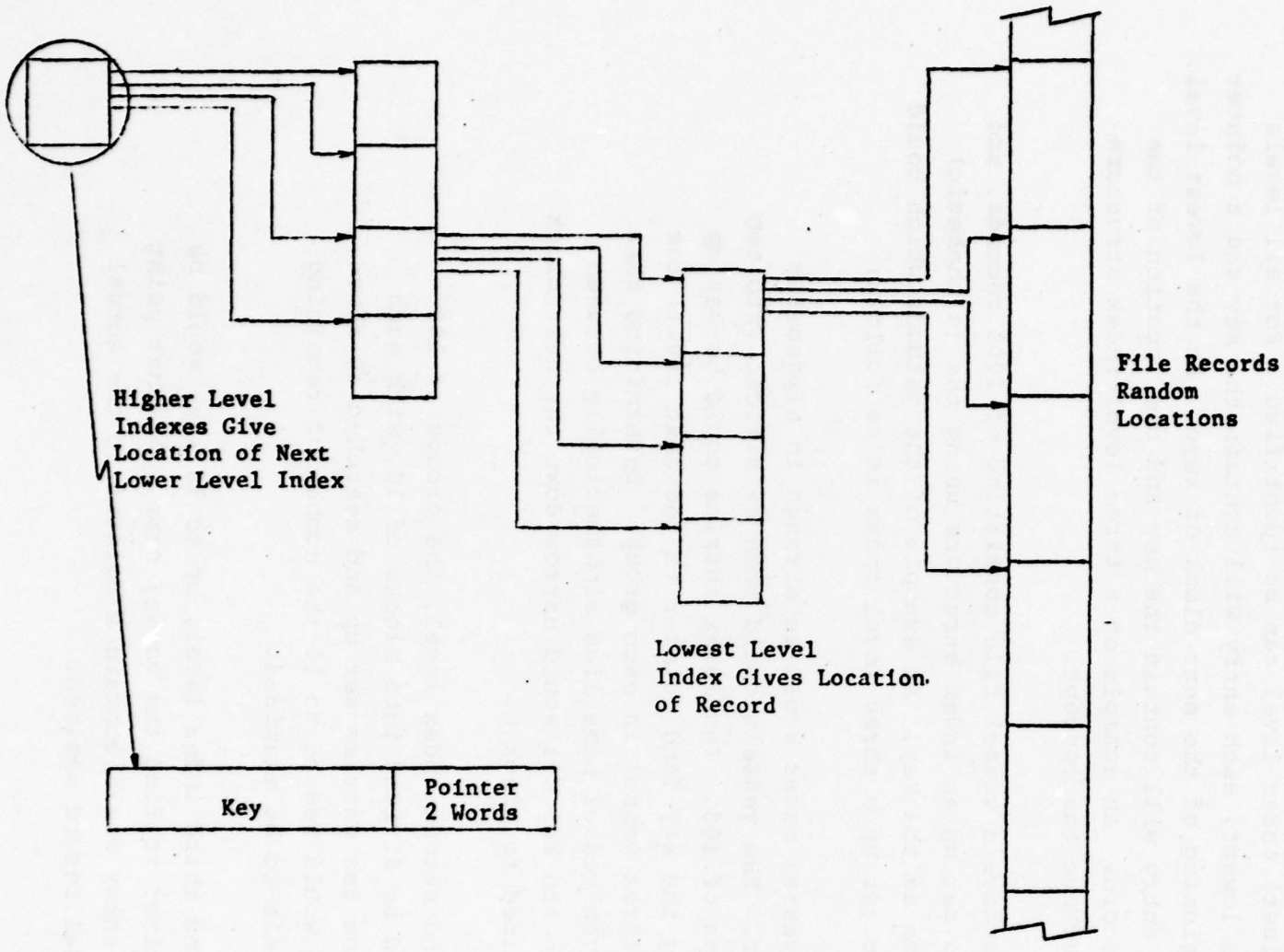


Figure 8-2. Multi-Level Index Structure

the first key in that block of keys. By comparing successive pairs of entries at the first level, the block containing the next (or last) index level can be identified. For all levels except the lowest, each entry will contain the key and a pointer to the beginning of the next block of keys. At the lowest level, the index entry will contain the key and the location of the desired record. An example of a three level index structure will illustrate the concept.

Suppose we have a vessel file consisting of 1000 records, and we want to set up an index structure using the (alphabetic) vessel name as the key. An example of the methods which could be used to set up a three level index is as follows:

- The vessel names would be arrange in alphabetic order. The vessels could then be divided into ten groups of 100. Ten index entries could be set up using the key (and pointer to the next level) for the first vessel in each group. Determining that a given vessel name lies alphabetically between key n and key n+1 would narrow down the next block examined to block n.

- At the second index level, the groups of 100 could be divided into blocks of 10, with each of the ten entries set up and searched as above. This would reduce to 10 the number of remaining vessels to be examined.

- At the third index level, up to 10 keys would be examined to find the vessel name. At that point the entry would contain a pointer to the actual vessel record desired.

Care must be taken when creating or eliminating a vessel record. When creating a vessel record, modification of all three indexes could be necessary if the entry requires addition to a block which is full. This is because we are assuming a fixed maximum for the number of entries in each block, so that the index blocks do not become disproportionate in size, thereby causing potentially drastic differences in response times for data access, as a function of the location of an entry within an index block. When deleting a vessel, similar modifications to the indexes are required if the deleted vessel causes an index block to be mptied.

For access to non-indexed files, or for searches on a key or keys which are not supported by indexes, the entire subject file will be examined to find the particular record or records of interest or to satisfy the requirements of the particular search (i.e., to produce the deisred output parameters subject to the limitations imposed by the key or keys).

These access methods provide the basis on which the analysis of disc accessing frequency was done. This analysis is given in Section 9.2.

Likewise, for the environment data, the following files have been identified:

- Weather sensors
- Current and tide sensors
- Manually-input weather data
- Forecasts

For the passage file, we have separated the identification and text of all communications into a separate file, because of the potential length of this data.

Figure 8-3 summarizes the mass storage requirements for the applications data files. This figure indicates the maximum storage needed for each file. The maximum number of records, the record size, and the total file size is given for all file categories except three, the map storage, software and miscellaneous files. For these three file categories only an estimate of required storage can be made until details of the software structure are developed.

The file sizing data given in Figure 8-3 includes only the logical storage requirements for each file. When the logical file requirements are mapped onto some physical disc device, there will be some unused space. This unused space results from taking into consideration the physical disc sector size. For example, if a logical record in a given file is 250 bytes long, and the disc sector size is 256 bytes, it is typical to assign one record per sector, leaving six bytes per sector for this file unused. Multiple records per sector may be assigned if the sector size is greater than or equal to an integral multiple of the logical record size. Thus, the number of unused bytes is a function of the disc sector and record sizes, and so we will not estimate the extra space required at this point.

8.5.3.1 Vessel File

Figure 8-4 presents vessel file sizing data. For each data element the units (if any), and the number of bytes required is given. Implicit in the number of bytes per entry for each data element is the requirement that it is large enough to accommodate the longest possible entry especially for those elements with varying lengths.

FILE	NUMBER OF RECORDS	RECORD SIZE(BYTES)	FILE SIZE(BYTES)
Vessel File	10,000	369	3,690,000
Vessel Indexes	10,000	72	720,000
Passage File	2,000	817	1,634,000
Passage Indexes	2,000	66	132,000
Communications	40,000	168	6,720,000
Route Segments	220	96	21,120
Waterway Cells			
- Primary	160,000*	16	2,560,000
- Supplementary	16,000	294	4,704,000
Waypoints	400	30	12,000
Docks and Piers	600	118	70,800
Navaids	2,000	14	28,000
Weather Sensors	20	25	500
Current and Tide Sensors	20	22	440
Manual Weather Data	40	219	8,760
Forecasts	10	210	2,100
Notices	100	10,087	1,008,700
Map Storage			10,000,000
Software Files			20,000,000
Miscellaneous			4,000,000
TOTAL			55,312,420

*Maximum total coverage

Figure 8-3. File Sizing Summary

DATA ELEMENT	UNITS	LENGTH (BYTES)
Vessel Name	Characters	25
Lloyd's Registry or Military Hull No.	Characters	8
Radio Call Sign	Characters	6
Type	Types	1
Gross Weight	Tons	4
Flag	Characters	14
Owner	Characters	36
Maximum Speed	Knots	1
Maximum Draft	Feet	1
Minimum Draft	Feet	1
Beam	Feet	2
Overall Length	Feet	2
Overall Height at Minimum Draft	Feet	1
Doctor Aboard Normally	Yes - No	1
Local Agent - Name	Characters	36
Address & Phone #	Characters	55
Time Required for Crash Stop	Seconds	2
Type of Navigational Equipment	Types	2
Number of Screws	---	1
Minimum Turning Radius	Feet	2
Date Data Last Verified	--	2
Date Vessel Last Active	--	2
Miscellaneous Comments	Characters	160
Linkage Pointer	--	4
TOTAL		369

Figure 8-4. Vessel File Sizing

The vessel file has four different keys through which background information about any vessel may be accessed or defined:

- Lloyd's Registry Number (hull number for military vessels)
- Radio call sign
- Name
- Location

The location key is the only one not stored in the file, and is primarily used for determining the identity of a vessel in passage, based on an operator supplied location.

8.5.3.2 Passage File

Figure 8-5 provides sizing information for the passage file. This file contains basic information for the passage, as well as other time dependent information for each vessel depending on its passage status (imminent, underway, anchored or docked). If the passage status is imminent, only the basic data (see Figure 8-5) is necessary. As noted above, information about communications has been defined as a separate file.

In Level 4 and 5 systems, position, course, speed and size data about each vessel in passage are not stored in the passage file. Instead, these data will be kept in a memory table, because they change rapidly, and are used extensively for hazard detection calculations. For unidentified vessels only this type of data is stored.

DATA ELEMENT	UNITS	LENGTH (BYTES)
Vessel ID Code	Characters	4
Vessel Name	Characters	25
Vessel Type	Types	1
Passage Status	States	1
Present Sector	Sectors	1
Origin or Point of Entry to VTS	Coordinates	4
	Characters	40
Destination or Point of Exit	Coordinates	4
	Characters	40
Date/Time of Entry	Coordinates	4
Scheduled Date/Time of Exit		4
Pilot - Designation	Characters	6
Name	Characters	36
Barge Makeup	Characters	40
Cargo	Types	1
Actual Draft	Feet	1
Lane Program Flag		1
Intended Route	Segments	100
Link to Vessel File		4
Link to Communications		4
TOTAL		321
Underway (Levels 1, 2, 3)		
For Each Checkpoint Passed (Up to 50):		
Checkpoint Designation	Characters	4 x 50
Date/Time at Checkpoint		4 x 50
Speed of Advance to Next Checkpoint	Knots	1
Designation of Next Checkpoint	Characters	4
Time of Arrival at Next Checkpoint	2	
TOTAL		405

Figure 8-5. Passage File Sizing
(Page 1 of 2)

DATA ELEMENT	UNITS	LENGTH (BYTES)
Anchored (All Levels)		
Anchorage Designation	Characters	14
Swing Radius of Mooring	Feet	2
Location of Anchorage	Coordinates	4
	Characters	40
Date/Time Anchorage Established		4
Date/Time Expected to Get Underway		4
Drift Status		1
TOTAL		69
Docked (All Levels)		
Dock or Pier Designation	Characters	14
Date/Time Arrived		4
Date/Time Scheduled to Depart		4
TOTAL		22
Communications		
Link Pointer		4
Date/Time of Communications		4
Summary of Communication		160
TOTAL	Characters	168

Figure 8-5. Passage File Sizing
(Page 2 of 2)

The passage file data may be accessed through five keys:

- Vessel ID code
- Name
- Pilot Designation
- Tracker ID (assigned by the tracking system in Level 4 and 5 systems)
- Location

The tracker ID and location keys are not kept in the passage file, but will be used internally to identify vessels. In addition, the location key may be used by the watchstander to identify a vessel in passage.

8.5.3.3 Waterway Characteristics File

Figure 8-6 gives the file size for the waterway file. Excluded from this figure is information regarding the geographical breakdown of the waterway into map cells. This is treated in Section 8.5.3.4.

The Waterway file is subdivided functionally into four files as mentioned above. Relatively static information is contained in each of these four files describing normal and special routes taken by vessels in passage, waypoints used to estimate vessel position, the various docks and piers in the waterway, and aids to navigation present in the waterway.

DATA ELEMENT	UNITS	LENGTH (BYTES)
Route Segments		
Segment Designation	Characters	6
Coordinates of End Points of Straight Line Portions	Coordinates	80
Length of Route Segment	Feet	4
Minimum Depth at MLLW	Feet	1
Speed Limit	Knots	1
For Normal Route Segment:* Minimum Width of Channel	Feet	2
For Special Route Segment:* Date/Time Created		4
TOTAL		94 - 96
Waypoints		
Designation of Waypoint	Characters	6
Location	Coordinates	4
Location of Waypoint in Route Segments		20
TOTAL		30
Docks and Piers		
Designation	Characters	14
Length	Feet	2
Depth at MLLW	Feet	1
Width	Feet	2
Facilities Available	Types	4
Location	Coordinates	4
Name of Owner	Characters	36
Address and Phone Number	Characters	55
TOTAL		118
Navigational Aids		
Designation	Characters	6
Location	Coordinates	4
Watch Circle Radius	Feet	2
Type of Aid	Types	1
Adrift or Missing Processing Flag		1
TOTAL		14

*Only one of these two entries is required.

Figure 8-6. Waterway Characteristics File Sizing

8.5.3.4 Waterway Cell Data Base Structure

The VTS coverage area will be subdivided into cells which will be used to describe the characteristics of the waterway. The waterway database is structured as three levels, the highest of which is resident in memory. See Figure 8-7.

At the first level the VTS area is divided into cell blocks. Each block represents a 10 x 10 matrix of cells. The memory resident descriptor is a single word containing:

- Mean lowest low water (MLLW)
- Flags describing features of the cell block including:
  - Routes/lanes
  - Docks/piers
  - Hazards
  - Navigational aids
  - Sensors
  - Waypoints

In a maximum coverage VTS, the memory table would represent a 40 x 40 matrix with each word describing a block of cells with a total size of approximately 10 nautical miles on a side.

Each cell block is a 10 x 10 matrix of individual cell descriptions which are each 8 words. Each cell represents an area that is approximately 1 nautical mile on a side. Included in the cell descriptor is the following: (See Figure 8-8)

. Mean lowest low water for the cell	1 byte
. Speed limit in cell	1 byte
. Flags describing cell features	2 bytes

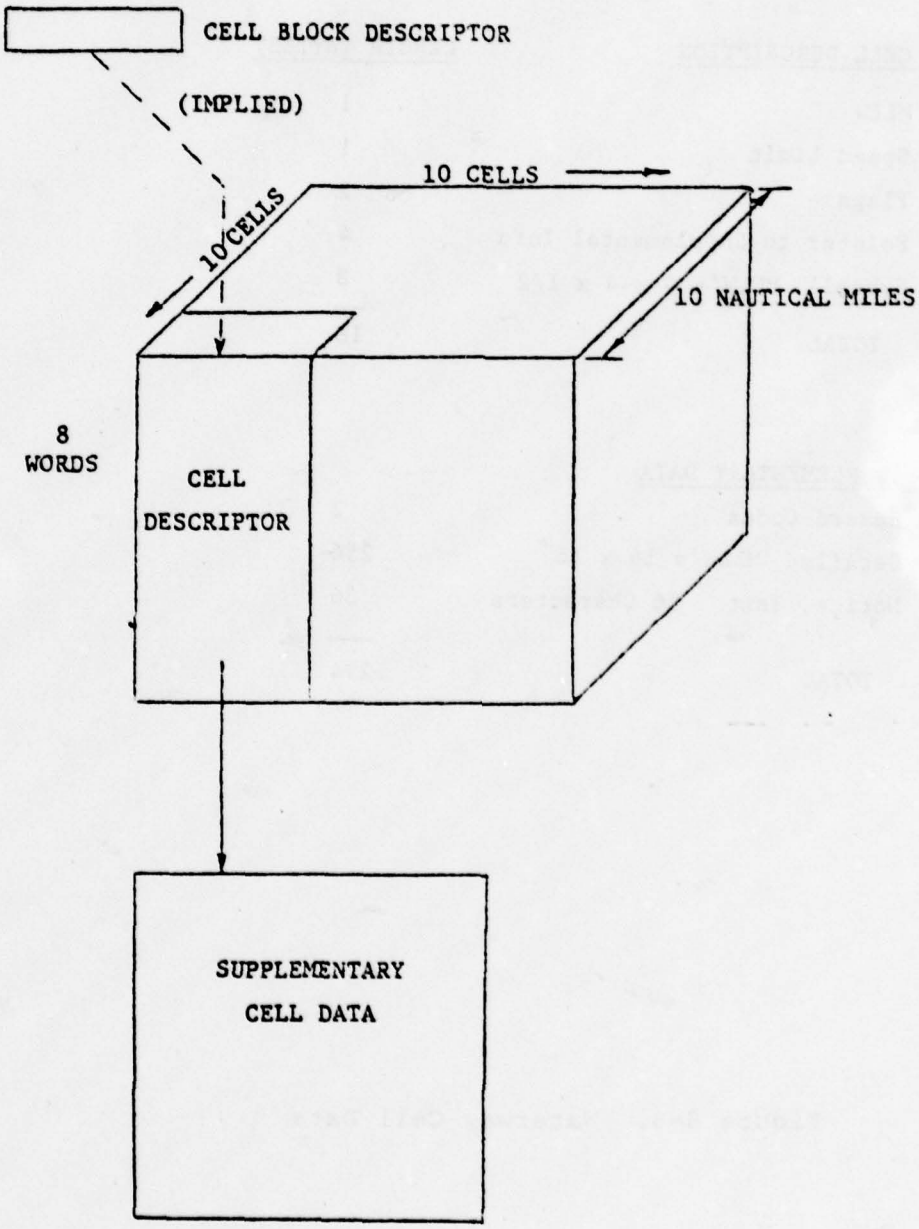


Figure 8-7. Waterway Base Description

CELL DESCRIPTION	LENGTH (BYTES)
MLLW	1
Speed Limit	1
Flags	2
Pointer to Supplemental Info	4
Subcell MLLW's 4 x 4 x 1/2	8
TOTAL	16

SUPPLEMENTARY DATA	
Hazard Codes	2
Detailed MLLW's 16 x 16	256
Notice, Text 36 Characters	36
TOTAL	294

Figure 8-8. Waterway Cell Data

- Subcell descriptor 8 bytes
- Pointer to supplementary data 4 bytes

The subcell descriptor included above contains a 4 x 4 matrix of 4 bits each. The data items are a coded representation of the MLLW within a cell. The value zero indicates a water level equal to the MLLW for the entire cell. Values 1 through 15 determine the water level by the formula:

$$MLLW_{subcell} = MLLW_{cell} + 2 \times VALUE$$

The cell descriptor contains the MLLW for each area of the waterway down to approximately 1/4 nautical mile on a side. If additional detail is needed for MLLWs or if the flag word indicates the presence of hazards or other features requiring elaboration, a detail block will exist containing the required data.

8.5.3.5 Environment File

Figure 8-9 gives the size of the environment file, which has been subdivided into four files. These files contain information, as a function of location within the waterway, about the

- weather status, such as temperature, and visibility, from automatic and manual input;
- waterway status, such as direction and speed of current, and tide level;
- weather forecast.

DATA ELEMENT	UNITS	LENGTH (BYTES)
Automatic Weather Stations		
Designation	Characters	14
Location	Coordinates	4
Temperature	Degrees	1
Visibility	Miles	1
Precipitation Rate	Scaled	2
Wind Direction	Degrees	2
Wind Speed		1
TOTAL		25
Automatic Current and Tide Sensors		
Designation	Characters	14
Location	Coordinates	4
Direction of Current	Degrees	2
Speed of Current		1
Tide Level Referred to MLLW		1
TOTAL		22
Manual Data Inputs		
Source of Data	Characters	36
Date/Time Entered		4
Location Where Valid	Sector(s)	1
	Coordinates	4
Key Words	Characters	14
Free Form Text	Characters	160
TOTAL		219
Forecasts		
Source	Characters	36
Date/Time Entered		4
Date/Time Span Valid		8
Area Covered	Sector(s)	2
Forecast	Characters	160
TOTAL		210

Figure 8-9. Environment File Sizing

potential length of this data.

8.5.3.6 Notices File

Figure 8-10 lists the information necessary for the notices file. The main portion of this file is the text of the notice. Other information identifies and describes the notice in terms of its applicability to a waterway area (i.e., sector(s) and the time period for which the notice is valid.

8.5.3.7 Map Display Storage

Ten basic maps (maximum) are required for the VTS coverage area. We assume 10,000 vectors for each basic map (x1 magnification). In addition, three levels of magnification are required, x2, x3, x5. We assume that the number of vectors required for the magnifications is equal to 10,000 multiplied by the square of the magnification factor. Thus, for all four magnification levels, 390,000 vectors are required for each map. Since at this level of detail most vectors would be represented by short vector forms requiring 2 bytes, we will allow approximately 10 million bytes of storage for map data.

8.5.3.8 Software Files

Storage will be necessary for software object files in all systems, and for source files in systems which have the program development capability. This storage must include sufficient space for applications and operating system software necessary for all processors. The source and object data will not be stored in the same file. In addition, applications and operating systems software will require separate files. Finally, the software necessary for each processor may be grouped and stored with the processor, depending on the architecture chosen.

For a first approximation of the total software storage required, we have estimated 20 million bytes, based on experience with similar systems.

DATA ELEMENT	UNITS	LENGTH (BYTES)
Type of Notice	Types	1
Light List Number if Applicable	Characters	36
Portion(s) Covered by Notice	Sectors	2
Name or Number of Navaid if Applicable	Characters	36
Date/Time Entered		4
Date/Time Span Valid		8
Text	Characters	10,000
TOTAL		10,087

Figure 8-10. Notices File Sizing

Figure 8-3. File Sizing Summary

8.5.3.9 Miscellaneous Files

The files described so far do not necessarily define precisely the total data base required. Various other data files may be required to back up volatile data at intervals in case of system failure, provide temporary storage for intermediate results of a procedure, or store system parameters. We estimate that roughly 4 million bytes could be required for these types of files.

9 HARDWARE DESIGN

In this chapter, resource requirements will be established for each of three VTS configurations. The three configurations are:

- A Class C, Level 4 System handling 900 identified vessel passages underway simultaneously.
- A Class B, Level 4 System handling 150 identified and 350 unidentified vessel passages underway simultaneously.
- A Class A, Level 1 System handling 100 vessel passages underway simultaneously.

Resource requirements will include main memory sizing, processing power required, and disc sizing and access rates.

After resource requirements have been presented, the hardware requirements for components of the VTS Processing/Display Subsystem will be presented.

9.1 PROCESSING AND MEMORY REQUIREMENTS

To determine processing and memory requirements for each of the three configurations, the processes discussed in Chapter 8 have been assigned to processors. Estimates have been made for memory and processing requirements for each process or group of processes.

The nucleus of the operating system and appropriate I/O drivers are included with each processor. Operating system overhead has been estimated to require 15 percent of each processor. Operating system memory requirements vary from processor to processor depending on the drivers included.

Processes are assigned to processors as discussed in Chapter 8 for the two smaller configurations. For the largest configuration, two additional types of processors have been added to offload a portion of the processing which would normally be assigned to the main processor.

9.1.1 Class C, Level 4 System

A Class C, Level 4 System capable of handling 900 identified vessel passages simultaneously will require a total of 17 processors assuming there are 12 display stations. This configuration is shown in Figure 9-1. Estimated memory and processing loads for each type of processor are given below.

9.1.1.1 Main Processor Requirements

Two main processors will be included to handle the loads shown below. Each processor is capable of handling the full load so one can serve as a spare which can replace either the other main or the routing and scheduling processor.

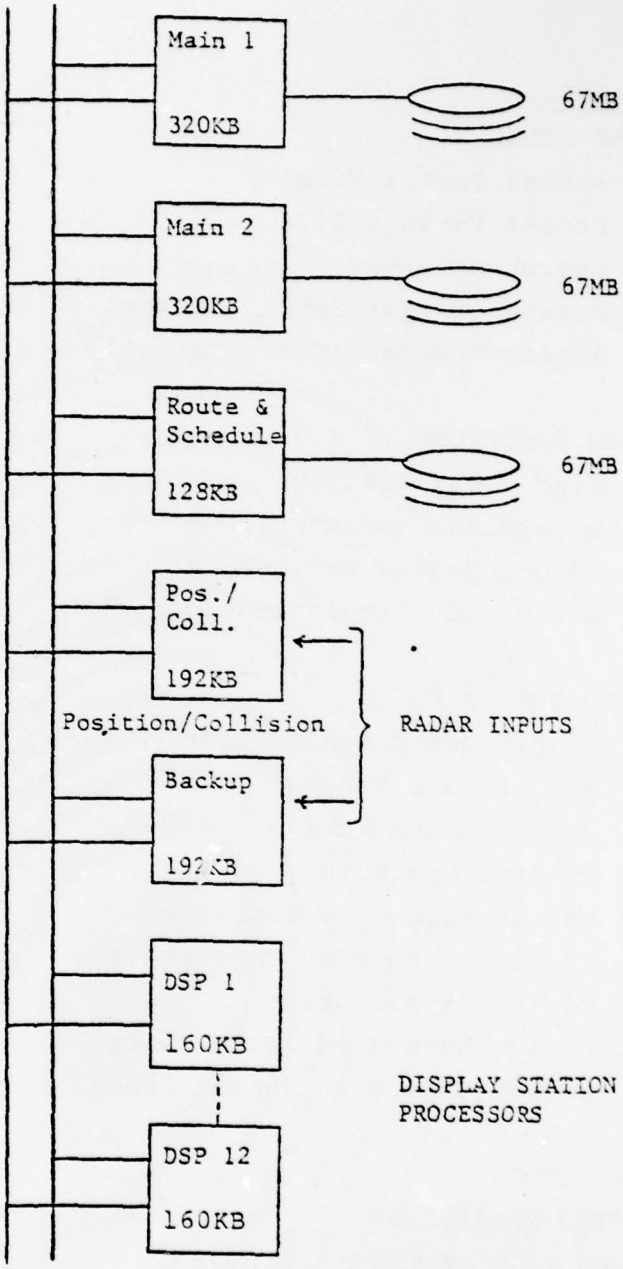


Figure 9-1. Shared Bus, Large Mini Architecture
900 Ships, Class C, Level 4

PROCESSES	CYCLES/SEC	MEMORY K BYTES
ACCESS FILES	132,780	29
. Access Vessel File		
. Access Passage File		
. Search on a Key		
. Access Environmental Information		
. Access Waterway Characteristics		
DEMAND FUNCTIONS	40,460	18.3
. Find Local Traffic		
. Determine Encounters		
. Adjust System Parameters		
. Setup/Edit Simulation Scenario		
HAZARD DETECTION		
. Hazard Detection Monitor	5,000	2
. Detect Lane Stray	30,000	12
. Detect Route Stray	30,000	12
. Predict Grounding	156,000	30
. Detect Excessive Congestion	6,000	2
. Predict Dangerous Encounters	15,000	2
. Detect Anchor Drift	4,500	2
. Detect Navaid Adrift/Missing	9,900	10
. Detect Excessive Vessel Speed	4,500	2
POST ALERTS	20,500	8.7
LOGGING MANAGEMENT	24,550	21
MANAGE ENVIRONMENTAL SENSORS	5,000	4
PLAY BACK SIMULATION SCENARIO	131,000	24
MISCELLANEOUS	10,600	57.9
OPERATING SYSTEM	187,500	42
FILE MANAGEMENT SYSTEM		30
TOTAL	813,290	308.9

9.1.1.2 Routing and Scheduling Processor Requirements

PROCESS	CYCLES/SEC	MEMORY K BYTES
Route/Schedule Vessel	375,000	22
Access Files	17,780	10
Operating System	187,500	42
File Management		30
TOTAL	580,280	104

9.1.1.3 Position/Collision Processor Requirements

Two position/collision processors will be included with one serving as a backup. Either of the two will be capable of handling all position processing and collision detection functions.

PROCESS	CYCLES/SEC	MEMORY K BYTES
Position Processing		
. Convert Radar Data	150,000	133
. Distribute Position Data	10,000	1
. Manage Position Table	5,000	1
. Handle Position Sensors	1,080	1
Hazard Detection		
. Predict Collisions Processes	491,100	13.8
Operating System	187,500	29
Total	844,680	178.8

9.1.1.4 Display Station Processor Requirements

PROCESS	CYCLES/SEC	MEMORY K BYTES
Transact Watchstander Request	15,150	11
Manage Alert Display	200	1.5
Manage Map	37,750	81
Operating System	187,500	26
Total	240,600	119.5

9.1.2 Class B, Level 4 System

A Class B, Level 4 System capable of handling 150 identified and 350 unidentified vessel passages simultaneously will require two main processors. Seven display station processors are assumed. This configuration is shown in Figure 9-2.

The display station processors will be slightly less loaded than in the Class C, Level 4 case but the differences are insignificant and will not be shown here.

The estimates shown below are for the two main processors. Either of these processors can cover the full load shown so that one can be an online spare for the other.

PROCESSES	CYCLES/SEC	MEMORY K BYTES
Access Files	117,540	29
Demand Functions	13,460	15.3
Hazard Detection		
. Hazard Detection Monitor	5,000	2
. Predict Collisions	79,400	9
. Detect Lane Stray	5,000	12
. Detect Route Stray	5,000	12
. Predict Grounding	26,000	23
. Detect Excessive Congestion	1,000	2
. Predict Dangerous Encounters	2,500	2
. Detect Anchor Drift	900	2
. Detect Navaid Adrift/Missing	9,900	4
. Detect Excessive Vessel Speed	900	2
Position Processing		
. Convert Radar Data	83,000	51
. Distribute Position Data	5,000	1
. Manage Position Table	3,400	1
. Handle Position Sensors	8,300	1
Post Alert	20,300	8.7
Logging Management	10,950	20

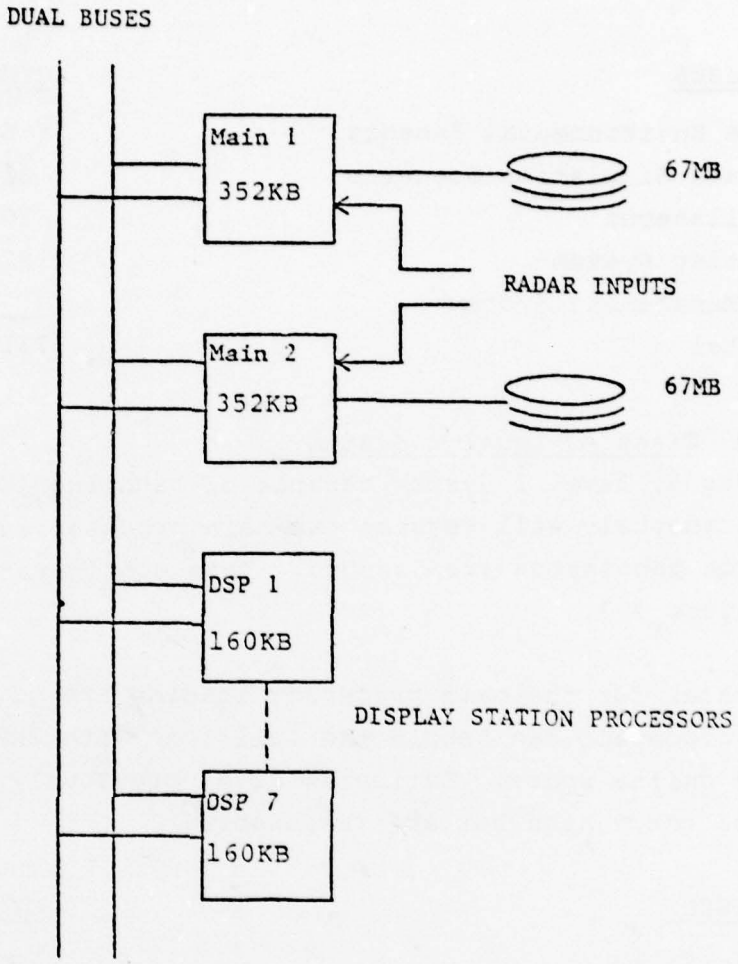


Figure 9-2. Shared Bus, Large Mini Architecture
500 Ships, Class B, Level 4

PROCESSES	CYCLES/SEC	MEMORY K BYTES
Manage Environmental Sensors	5,000	4
Playback Simulation Scenario	131,000	24
Miscellaneous	10,600	29.9
Operating System	187,500	45
File Management System		20
Total	731,650	319.9

9.1.3 Class A, Level 1 System

A Class A, Level 1 System capable of handling 100 vessel passages simultaneously will require two main processors. Five display station processors are assumed. This configuration is shown in Figure 9-3.

Estimates for the main processor loading are given below. Either main processor can handle the full load with the other serving as an online spare. Estimates given previously (Section 9.1.1.4) are slightly high but are reasonable.

PROCESSES	CYCLES/SEC	MEMORY K BYTES
Access Files	102,510	29
Demand Functions	9,660	18.3
Hazard Detection		
. Hazard Detection Monitor	5,000	2
. Detect Excessive Congestion	6,000	2
. Predict Dangerous Encounters	1,500	2
. Detect Excessive Vessel Speed	900	2
Post Alerts	20,100	8.7
Logging Management	2,080	17
Manage Environmental Sensors	1,000	2
Playback Simulation Scenario	80,000	24
Miscellaneous	10,960	16.9
Operating System	187,500	42
File Management System		18
Total	427,210	183.9

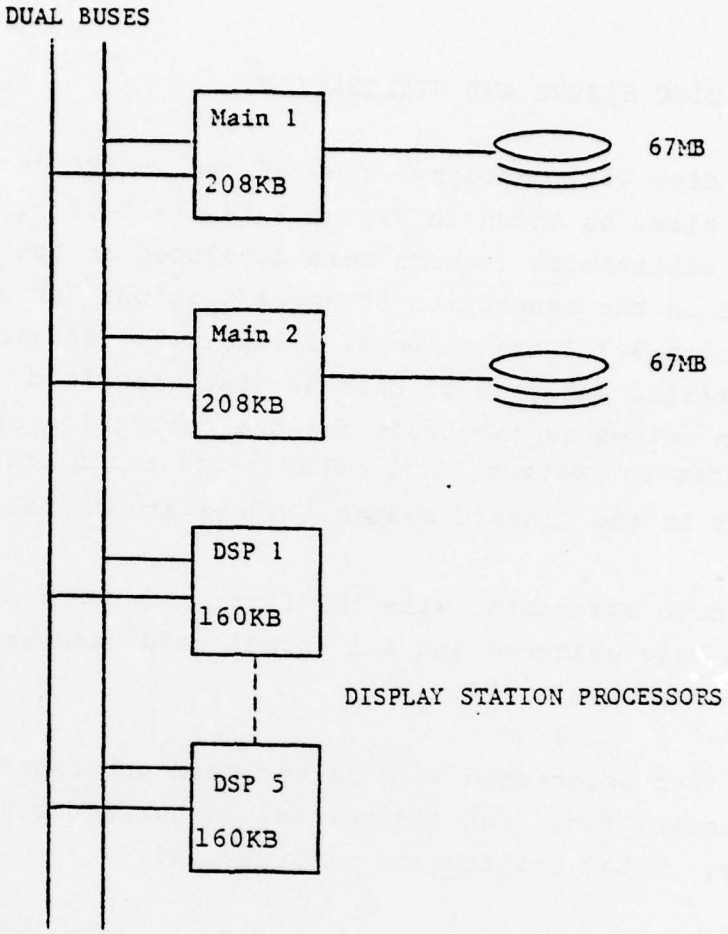


Figure 9-3. Shared Bus, Large Mini Architecture
100 Ships, Class A, Level 1

9.2 DISC SIZING AND UTILIZATION

Each disc will contain a copy of the entire data base. The data base size, as shown in Figure 8-14, is 54.5 million bytes. The disc utilization factors were developed in the Phase I report$_1$, based on the assumption of specifications for a 3330 type disc (Section 9.3.3 of the Phase I report$_1$). Section 9.2.1 presents a detailed analysis of disc accesses required for VTS functions, which serves as the basis for the derivation of utilization figures in Section 10.1. The greatest utilization factors occur in the Class C system. These are as follows:

The disc associated with the first main processor will handle data base searches and all normal reads and writes for a total utilization of .474.

The disc associated with second main processor will support the grounding functions and maintain a current copy of the data base. Total utilization will be .426.

The third disc will be associated with the routing and scheduling processor. It will also maintain a current copy of the data base. Total utilization will be .481.

In order to avoid high utilization rates (and resulting excessive response times), the disc access times should meet those assumed in Section 9.3.3 of the Phase I report$_1$.

9.2.1 Disc Access Frequencies

In this section we will develop the number of disc accesses required for the major watchstander functions in the three scenarios defined in Section 3 of the Phase I report:

- Scenario 1 - A Class C, Level 4 system with 900 passages per day for identified vessels
- Scenario 2 - A Class B, Level 4 system with 150 passages per day for identified vessels
- Scenario 3 - A Class A, Level 1 system with 100 passages per day for identified vessels

Figure 9-4 presents, for each subfunction of each major procedure, the number of disc reads and writes required. Cases where either a read or write access is not required are indicated by a dash.

For the vessel and passage files, we have assumed a three level index structure. The fractional number of writes indicated in Figure 9-4 takes into consideration the fact that modifications to the index structure will be required (although infrequently for higher levels), when creating or deleting records in the file, as discussed in Section 8.4.2.

The total number of reads and writes required for each major function are also indicated. These numbers are then rounded to the next higher integer and used in subsequent figures for each scenario. Using USCG provided data for the number of executions per passage for each major function, we may calculate the number of reads and writes (and the sum of these) required for each scenario.

ENTER VESSEL	READS	WRITES
Search Index for Existing Vessel	3	-
Enter Vessel Name in Index	-	1.2
Check Free Block	1	-
Update Free Chain Pointers	-	1
Write New Record	-	1
Search Name Index and Insert Pointer	3	1
Insert in Call Sign Index	3	1.1
Insert in Lloyd's Index	3	1.1
TOTAL	13	6.4
DELETE VESSEL		
Search Index for Vessel	3	-
Read Vessel Record	1	-
3 Search and Deletes @ 3, 1.1	9	3.3
Unlink from Active or Inactive Chain	2	2
Link to Free Chain	1	1
Update Free Chain Pointers	-	1
Write Blank Vessel Record	-	1
TOTAL	16	8.3
MODIFY VESSEL DATA		
Search Index for Vessel	3	-
Update Vessel Data	1	1
TOTAL	4	1
ENTER NEW PASSAGE		
Search Vessel Index	3	-
Read Vessel Data	1	-
Search Passage Index	3	-
Get Free Passage Block	3	3
Write New Passage Block	-	1
Insert in Name Index	3	1.1
Insert in ID Code Index	2	1.1
Insert in Pilot ID Index	2	1.1
Unlink Vessel from Inactive List	2	3
TOTAL	19	10.3
CHANGE STATUS		
Search Passage Index	3	-
Read Passage Record	1	-
Write Updated Passage Record	-	1
TOTAL	4	1

Figure 9-4. Disc Access Estimates by Function
(Page 1 of 2)

DELETE PASSAGE	READS	WRITES
Search Index for Vessel	3	-
Read Record	1	-
Search and Delete from Indexes	7	3.3
Link Vessel Record to Inactive List	2	3
Link Passage Record to Free List	1	1
Write Blank Passage Record	-	1
TOTAL	14	8.3
IDENTIFY VESSEL		
Search Passage Index	3	-
Read Passage Record	1	-
Delete Unidentified Record	3	3
Write New Passage Record	-	1
TOTAL	7	4
UPDATE VESSEL POSITION		
Search Passage Index	3	-
Read and Write Passage Record	1	1
TOTAL	4	1
ENTER NEW COMMUNICATION		
Search Passage Index	3	-
Read Passage Record	1	-
Read Previous Communications	2	-
Get Free Block (Blocking = 10)	-	.2
Write Pointer to New Block	-	.1
Write New Communication	-	1
TOTAL	6	1.3
MODIFY PASSAGE INFORMATION		
Search Passage Index	3	-
Read and Write Passage Record	1	1
TOTAL	4	1

Figure 9-4. Disc Access Estimates by Function
(Page 2 of 2)

FUNCTION	OPERATIONS PER DAY	PER FUNCTION READS	PER FUNCTION WRITES	PER FUNCTION TOTAL	PER DAY READS	PER DAY WRITES	PER DAY TOTAL
Enter New Vessel	180	13	7	20	2,340	1,260	3,600
Modify Vessel Data	18	4	1	5	72	18	90
Delete Vessel	180	16	9	25	2,880	1,620	4,500
Enter New Passage	540	19	11	30	10,260	5,940	16,200
Change Status	1,080	4	1	5	4,320	1,080	5,400
Delete Passage	540	14	9	23	7,560	4,860	12,420
Identify Vessel	810	7	4	11	5,670	3,240	8,910
Update Vessel Position	3,600 +	4	1	5	14,400	3,600	18,000
Modify Passage Information	180	4	1	5	720	180	900
Enter New Communication	4,500	6	2	8	27,000	9,000	36,000
Scheduling/Routing	2,700	45	-	45	121,500	-	121,500
Other Demand: Encounters, Local Traffic, etc.	6,000	10	-	10	60,000		60,000
Total					256,722	30,798	287,520
Peak Rate*					11.88/sec	1.43/sec	

*Four times average rate
+Four per vessel for Level 4 system

Figure 9-5. Disc Accesses - Watchstander Functions
Class C, Level 4 - 900 Vessels

	Operations Per Day	Per Function R	Per Function W	Per Day Reads	Per Day Writes	Per Day Total
Enter New Vessel	30	13	7	390	210	600
Modify Vessel Data	3	4	1	12	3	15
Delete Vessel	30	16	9	480	270	750
Enter New Passage	90	19	11	1710	990	2700
Change Status	180	4	1	720	180	900
Delete Passage	90	14	9	1260	810	2070
Identify Vessel	135	7	4	945	540	1485
Update Vessel Position	600+	4	1	2400	600	3000
Modify Passage Information	30	4	1	120	30	150
Enter New Communication	750	6	2	4500	1500	450
Other Demand	1000	10	-	10000	-	10000
TOTAL				22537	5133	26170
Peak Rate*				1.04/sec	.24/sec	

*Four times average rate
+Four per identified vessel in Level 4 system

Figure D-6. Disc Access - Watchstander Functions
Class B, Level 4 - 500 Identified Vessels

	Operations Per Day	Per Function R	W	Reads	Writes	Total
Enter New Vessel	20	13	7	260	140	400
Modify Vessel Data	2	4	1	8	2	10
Delete Vessel	20	16	9	320	180	500
Enter New Passage	60	19	11	1140	660	1800
Change Status	120	4	1	480	120	600
Delete Passage	60	14	9	840	540	1380
Identify Vessel	-	7	4	-	-	-
Update Vessel Position	800	4	1	3200	800	4000
Modify Passage Information	20	4	1	80	20	100
Enter New Communication	500	6	2	3000	1000	4000
Other Demand	667	10	-	6670	-	6670
TOTAL				15998	3462	19460
Peak Rate*				.74/sec	.16/sec	

*Four times average rate.

Figure 9-7. Disc Accesses - Watchstander Functions Class A, Level 1 - 100 Vessels

9.3 HARDWARE SPECIFICATIONS

The purpose of this section is to present the minimal hardware specifications necessary to support VTS system requirements. The software requirements, in terms of processing speed, memory and disc space, and disc access rates, are the primary quantitative requirements. Other major criteria are flexibility and reliability. The hardware must be flexible in order that it may be easily configured to economically meet the needs of different system classes, levels, and vessel traffic loads. Dynamic flexibility is also necessary to allow the system to continue operating (in a degraded mode) in case of hardware component failures. Such reliability shall be possible by specifying that every component and interconnection be backed up by at least one alternative component or path.

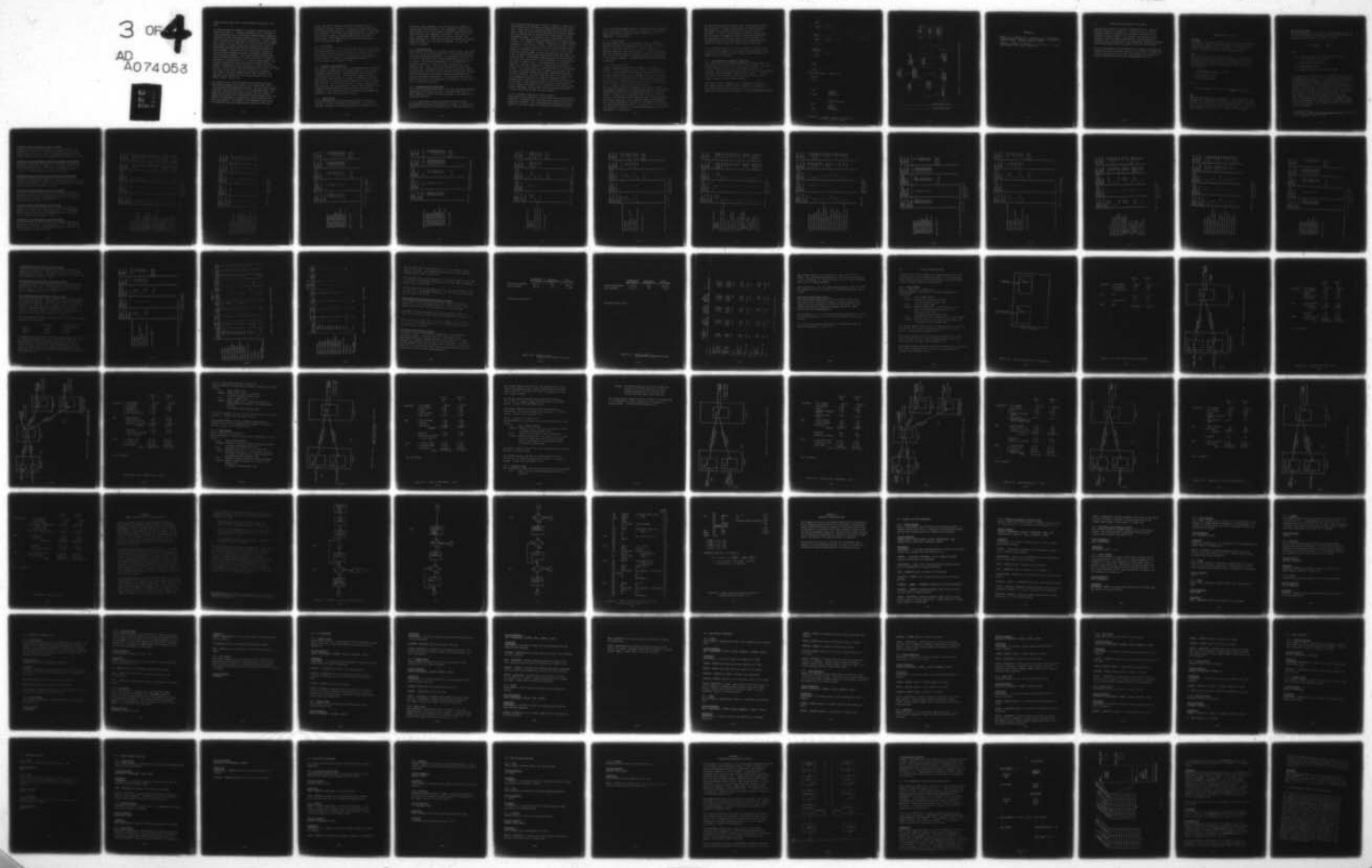
9.3.1 Processors

Each processor shall have a maximum memory cycle time of 800 nanoseconds. The required number of registers will be determined upon detailed operating system design.

Factors such as instruction cycle time and instruction repertoire are not specified, in order to avoid unduly restricting the selection of available processors. Instead, proposed processors should be tested with a representative mix of programs to determine whether the processor meets the performance requirements. Benchmark programs should be developed to reflect the instruction mixes which are anticipated, based on the Phase III system software design. If possible, the benchmark programs should be coded in the language selected for system development. A method is suggested in Appendix A for an approximate comparison of the relative power of different processors in the absence of benchmark capability.

AD-A074 053 INTERNATIONAL COMPUTING CO BETHESDA MD F/G 9/2
VTS PROCESSING DISPLAY SUBSYSTEM DESIGN.(U)
JAN 79 C C HENSON, F T MICKEY, R S GRAHAM DOT-CG-81-78-1833
UNCLASSIFIED USCG-D-54-79 NL

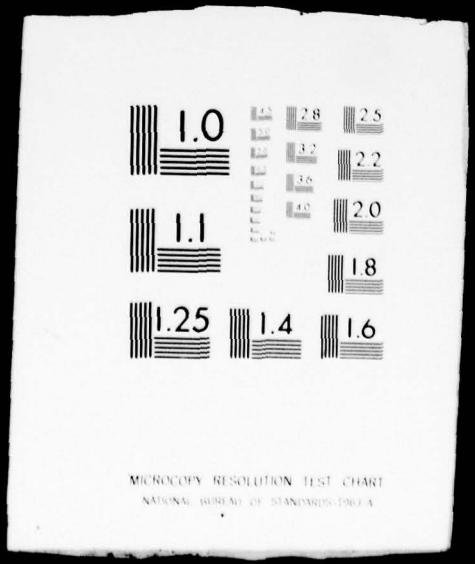
3 OF 4
AD A074053



3 OF 4

AD

A074053



Each processor shall have a minimum memory word length of 16 bits.

The memory sizes for computers in different configurations are shown in Section 9.1. A memory management unit is required to allow addressing of up to 512K bytes (maximum memory expansion potential) if the instruction address field is not sufficiently large to allow direct addressing of 512K bytes. A memory expansion chassis shall be supplied to allow easy addition of memory. There shall be a memory protection mechanism which prevents applications programs from accessing the protected memory of other programs. Included with each non-display processor in the two larger configurations shall be hardware to perform floating point arithmetic. The floating point hardware shall have a minimum word length of 32 bits, and shall be capable of performing a multiplication in less than 10 microseconds. The software supplied with the floating point hardware shall be capable of performing a sine calculation in less than 400 microseconds. Other features which each processor must have are a real-time clock, power failure/recovery hardware, an automatic load feature, and direct memory access (DMA) capability for efficient input/output with mass storage devices (discs and magnetic tapes) and the bus.

The real-time clock (RTC) provides the operating system and applications programs with access to the system time, for use in controlling and scheduling processes. The RTC should provide interrupt intervals within the 1 to 10 millisecond range. The power fail safe/recovery feature ensures that the content of memory and registers are not destroyed whenever power fails, and enables the processor to automatically resume operation when power is restored. The automatic load mechanism should allow the operating system and applications software to be loaded

either from disc or from the bus, upon initiation from the processor's operator console. For each disc, tape drive, and bus interface there should be a controller capable of high speed data transfers. The vendor shall demonstrate that no data can be lost due to delays in starting DMA transfers, or due to other causes. Every DMA controller shall have priority over the CPU in accessing memory.

9.3.2 Disc Units

Each disc shall have a minimum unformatted capacity of 80M bytes. The average rotational delay shall be less than 8.35 milliseconds. The average head positioning time may not be greater than 30 milliseconds. Each disc shall be connected to its processor by a disc controller capable of performing DMA transfers of multiple sectors (see Section 9.3.1.1).

9.3.3 Magnetic Tape Transports

All classes of systems will require two IBM-compatible transports. They shall be 9-track, with a recording density of 800 bpi NRZI or 1600 bpi phase-encoded. The minimum tape velocity during recording shall be 45 ips. They shall be capable of using 10.5 inch diameter tape reels with a capacity of 2400 feet of tape. They shall have a read-after-write capability to check for errors while writing. Each drive shall have an integral power supply. There shall be two tape controllers (one for each transport) and suitable interfaces between these controllers and selected processors. The controllers shall be capable of performing DMA transfers to and from the processor memory.

9.3.4 Line Printers

Each system shall have two high speed printers for output of reports. One shall be backup for reliability. Each shall be switchable between the two main processors, such that it may be

used with either processor. The printer shall be capable of printing 600 lines per minute. The alphanumeric character set shall consist of at least 96 ASCII characters, including both upper and lower case alphabetic characters. There must be 132 print columns with 6 lines per inch vertical spacing. The paper feed shall be of the tractor type, and should have printing width adjustable up to at least 14.8 inches. A single line buffer is required.

9.3.5 System Consoles

There shall be two hard-copy terminals with keyboards each connected to one of the main processors via a full duplex EIA RS-232C or current loop serial interface. A transmission rate of up to 300 baud is required (with a corresponding printing speed of up to 30 cps). The printing mechanism of the terminal shall be capable of printing a line of at least 72 characters. It is desirable that it have a tractor feed mechanism, rather than friction feed. The terminal shall include a typewriter (QWERTY type) keyboard. Both the printer and the keyboard shall provide 64 ASCII characters which include uppercase alphabetics with numerics and control characters. It is desirable that the full ASCII character set of 96 characters be provided with both uppercase and lowercase characters.

9.3.6 Display Station Peripherals

This section provides specifications for the individual peripheral components of the display station. The numbers and types of these components will be selected according to the needs of each particular VTC site.

9.3.6.1 Alphanumeric Display/Keyboard and Slave Displays

The alphanumeric display screen shall be capable of displaying a minimum of 24 lines of 80 characters each. Both the display and

the typewriter keyboard shall be able to input or output the 96 ASCII character set including both upper and lower case characters. There shall be a display blinking field function which may be turned on and off by function keys or by program control. Both the display and the keyboard shall input/output in block transmission mode. The keyboard shall be detachable from the display. The keyboard shall be divided into four separate sets of keys, a) a typewriter ("QWERTY") keyboard, b) a numeric keypad consisting of the digits 0 through 9, period, and minus ('enter' optional), c) a set of editing keys which shall include insert/delete for character and for line, tab, scroll up, scroll down, next page, previous page, clear screen and clear line, and d) a set of cursor function keys which shall include up, down, left, right and home. An automatic repeat key must be included with the cursor and editing keys, and is desirable for the typewriter keys and the numeric keys. The display shall have a memory capable of storing a minimum of 5 display pages (each page of memory must be large enough to contain the data of a full display screen). The display unit must be capable of performing the following functions: display next page, display previous page, scroll up one line, and scroll down one line. The display and its transmission line must be capable of displaying data at a minimum rate of 1200 bits per second. Up to 9600 bits per second is desirable. Some display stations may have a slave display, i.e., a display with keys only for controlling the display (clear screen, next page, previous page, scroll up, and scroll down). A slave display has no keys for input of data.

9.3.6.2 Traffic Coordination Function Console

This console shall consist of a set of keys for performing special functions. A minimum of 24 keys shall be provided, with up to 32 keys being desirable. The minimum keyboard configuration shall consist of the keys grouped by functions as specified in Section

3.6.1 of the VTS Processing/Display Subsystem Functional Description[2]. The function console shall be connected to the processor using an EIA RS-232C interface, with a transmission rate compatible to the interface.

9.3.6.3 Alert Display

The alert display shall be a receive-only display capable of displaying at least 40 columns by 22 lines of text. It must be capable of displaying the entire 96 character ASCII set, including upper and lower case characters. It shall be interfaced to the display station processor via an EIA RS-232C interface, and shall be capable of displaying data at a minimum rate of 1200 bits per second. A maximum rate of 9600 bits per second is desirable.

9.3.6.4 Hard Copy Printer

A small low-speed printer shall be capable of printing the contents of the alphanumeric display screen. The specifications for such a printer shall be the same as the specifications for a system console (Section 9.3.5), except that it must be able to print lines of 80 characters and that no keyboard is required. In addition, it is required that the terminal be non-impact for quiet operation. The printer may have a friction feed mechanism.

9.3.6.5 Graphic Display Screen and Processor

The unit shall include both a display screen and an internal programmable processor for controlling the displays. The graphics display screen shall have a minimum usable area of 211 square inches. It shall have a resolution which allows at least 1024 points to be displayed in both horizontal and vertical dimensions. The display units shall have a graphics command storage space and a writing speed which together will be capable of producing a flicker-free display consisting of at least 10,000 vectors and 500 characters. The use of long persistence phosphors to meet this requirement is unacceptable.

The display should have a feature which, upon command, causes specified sets of vectors and characters to blink on and off. The capability to produce displays with multiple colors is desirable. If color is not available, then the graphics unit must be capable of producing vectors of different intensities. The capability to change the scaling factor of display for selected portions of a screen image with a maximum magnification of at least 5 times the original scale is desirable.

An x, y positioning device must be provided to allow the user to indicate and transmit to the display station processor points on the screen or particular items (where an item consists of a symbol or a set of vectors).

9.3.7 Configuration of Hardware Components

The organization of the hardware components described above is illustrated in Figure 9-8 for Class C, Level 4 system. For Class B and A systems, only the main and display processors are included. Each main processor shall have the following peripherals: disc drive, tape transport, system console, and line printer. In the Class B, Level 4 system, the radar tracker data (level 4 sensors) would be input to main processor.

The display station components are illustrated in Figure 9-9. The display station printer would be optional, as desired. An auxiliary graphics processor and display is also shown as an option that would allow an operator to view two maps simultaneously.

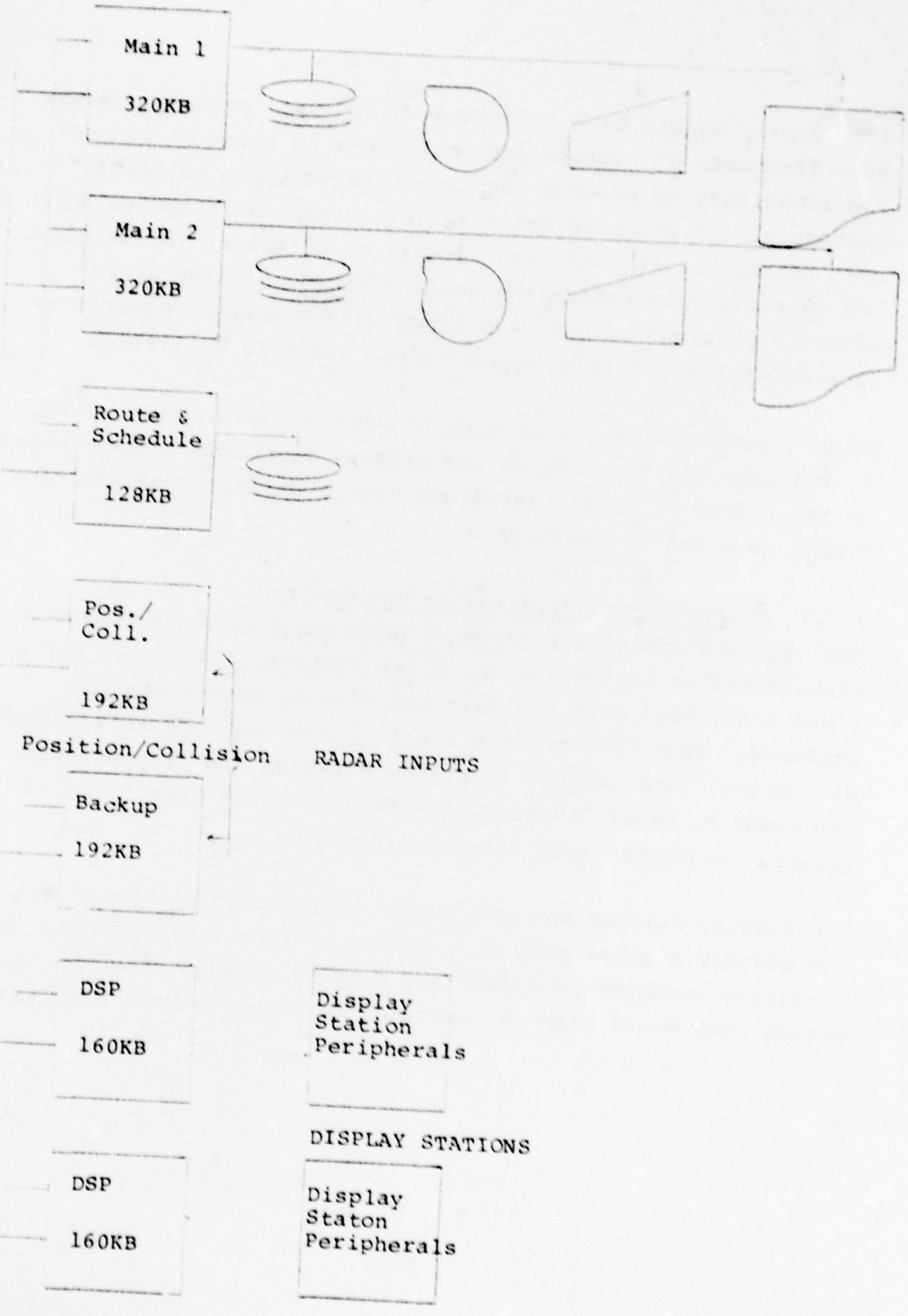


Figure 9-8 Hardware Component Configuration for Class C, Level 4 System

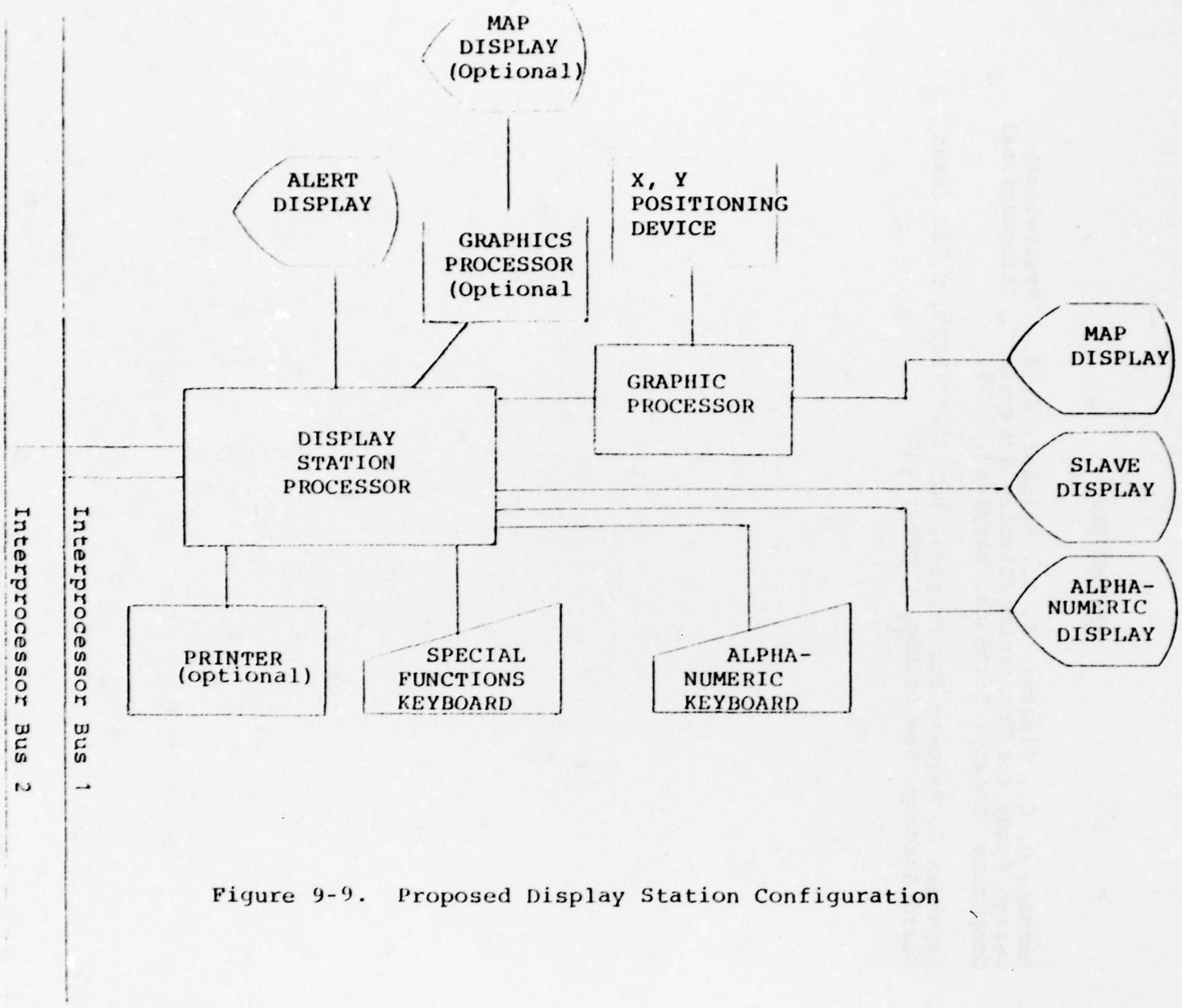


Figure 9-9. Proposed Display Station Configuration

REFERENCES

1. Henson, C. C., Cleaver, R. A., Kaisler, S. H., "Preliminary Design Study for Processing/Display Subsystem", International Computing Company, Bethesda, Maryland, 1978.

2. Appendix 8, Request for Proposal No. 81-77-1833, U. S. Coast Guard Academy, New London, Connecticut.

10 ANALYSIS AND EVALUATION OF THE DESIGN

The response time for a function is measured from the instant an operator completes a request for a function until the time the requested action is completed and the data is available at the operator's display station. The response time includes the CPU processing time, the disk access time for each read and write operation, the time required for bus communication between processors and the time spent waiting for access to the CPU, disk and bus.

In the first section of this chapter we will discuss the method and assumptions used to estimate each part of the function response time. In the second section we will use these estimates to calculate the average response times for several watchstander functions.

10.1 AVERAGE SERVICE TIMES

Processor

The number of CPU cycles required for each function was estimated in the Phase I Report. These estimates have been refined and are presented in Chapter 8 and 9 of this report. We assume a 1.25 million instruction cycles per second processing speed.

Disk

The average service time for a disk access, T_s, is the sum the average time to locate the cylinder, the average time to locate the data on the cylinder and the time to transfer the data. We have assumed the following characteristics which are typical of 3330 type disks:

- 30 ms average to locate a cylinder
- 16.67 ms rotation time
- 20480 bytes per track
- 512 bytes per sector

Thus,

$$T_s = 30\text{ms} + \frac{16.67\text{ms}}{2} + (\#\text{sectors}) \times 512 \frac{\text{bytes}}{\text{sector}} \times \frac{16.67 \text{ ms/rev}}{20480 \text{ bytes/rev}}$$

Bus

There are two types of bus transmissions: short messages and long messages which are broken into packets. A short message is at most 256 bits long of which 64 bits are the message header. Packets are 2048 bits long of which 80 bits are packet header. In addition we have assumed a bus data rate of 1.5 million bits per second.

Average Waiting Time

The average waiting time for access to each processor, disk, and bus have been computed using queuing theory results developed by Khintchine and Polloczek as discussed in Martin[1].

$$\bar{t}_w = \frac{\rho \bar{s}}{2(1-\rho)} \left[1 + \left(\frac{\sigma_s}{\bar{s}} \right)^2 \right]$$

where

$\bar{t}_w$ = average time waiting for service by the facility (i.e., processor, disk or bus)

ρ = utilization of the facility

$\bar{s}$ = average service time

σ_s = standard deviation of the service times

The computation for each facility is discussed in detail below and summarized in Table 10-3. To prevent any one long process from monopolizing a processor, each long process is divided into processing segments. Whenever a process is scheduled to run (having reached the top of the ready queue) only one processing segment of that process is allowed to run before the process must relinquish control of the processor back to the scheduler. The scheduler will then select the next process on the ready queue to run, and that process will run one segment. Thus, the average service time referred to in these calculations is the average time it takes to run a processing segment, not the whole process. Short processes, however, may consist of only one segment.

---

1 James Martin, Design of Real-Time Computer Systems; Prentice-Hall, Inc. Englewood Cliffs, NJ; 1967

Processor Waiting Time/Class C, Main Processor

The service times for each function in the main processors are summarized in Table 10-1. The utilization factor is .651, the average service time is 1.214 ms., the standard deviation of the service times is 1.711 ms., and the average waiting time is 3.374 ms.

Processor Waiting Time/Class C, Routing and Scheduling Processor

The service times for each function in the Routing and Scheduling processor are summarized in Table 10-2. The utilization is .464, the average service time is 7.953 ms., the standard deviation is 3.648 ms., and the average waiting time is 4.166 ms.

Processor Waiting Time/Class C, Position/Collision Processor

The service times for each function in the Position/Collision processor are summarized in Table 10-3. The utilization is .676, the average service time is 2.830 ms., the standard deviation is 5.800 ms., and the average waiting time is 15.341 ms.

Processor Waiting Time/Class C, Display Processor

The service times for each function in the Display processor are summarized in Table 10-4. The utilization is .195, the average service time is .260 ms., the standard deviation is .290 ms. and the average waiting time is .071 ms.

Processor Waiting Time/Class B, Main Processor

The service times for each function in the Main processor are summarized in Table 10-5. The utilization is .585, the average service time is 1.273 ms., the standard deviation is 1.957 ms. and the average waiting time is 3.020 ms.

Processor Waiting Time/Class B, Display Processor

The service times for each function in the Display processor are summarized in Table 10-6. The utilization is .174, the average service time is .251 ms., the standard deviation is .284 ms., and the average waiting time is .060 ms.

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Lane Stray	30	1	30	1,000	30,000
Route Stray	30	1	30	1,000	30,000
Dangerous Encounters	30	1	30	500	15,000
Congestion Analysis	30	1	30	200	6,000
Anchor Drift	15	1	15	300	4,500
Navaid Missing	33	1	33	300	9,900
Demand Calculation:					
Relative Position	1	1	1	1,500	1,500
Dangerous Encounter	1	1	1	500	500
CPA	1	1	1	300	300
Local Traffic	1	1	1	32,400	32,400
Potential Grounding	30	1	30	5,200	156,000
Excessive Vessel Speed	15	1	15	300	4,500
Data Retrieval:					
Searches	1	40	40	2,500	100,000
Environmental	1	1	1	5,000	5,000
Notices Lookup	1	1	1	10,000	10,000
Scenario Creation	.1	1	.1	600	60
Scenario Editing	.1	10	1	5,100	5,100
Scenario Playback	65	1	65	2,015	131,000
Alert Responses	1	4	4	5,000	20,000

Table 10-1. Class C, Level 4
Main Processor
Page 1 of 3

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Alert Posting	.5	1	.5	1,000	500
Vessel Passage History	10	1	10	1,000	10,000
Operation Log	10	1	10	1,000	10,000
Daily Traffic Summary	1.04	1	1.04	770	800
VTS Report Management	~0		~0	*	5,000
Inter VTS Communication	~0		~0	*	5,000
Display Distribution	2	1	2	300	600
VTS Initialization	~0		~0	*	5,000
Systems Parameter Adjust.	~0		~0	*	600
System Backup Data	15	1	15	250	3,750
Environmental Data Process.	25	1	25	200	5,000

Table 10-1. Class C, Level 4
Main Processor

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
File Management:					
Enter Vessel	.008	20	.16	2,500	400
Modify Vessel	.0008	5	.004	2,500	10
Delete Vessel	.008	25	.20	2,500	500
Enter Passage	.024	30	.72	2,500	1,800
Identify Passage	.036	11	.396	2,500	990
Update Passage	.32	5	1.60	2,500	4,000
Modify Passage	.008	5	.04	2,500	100
Enter Communication	.20	8	1.60	2,500	4,000
Delete Passage	.024	23	.552	2,500	1,380
Change Status	.048	5	.24	2,500	600
Environmental File	4	1	4	1,000	4,000
Operating System					187,500
			402.152		813,290

*These functions are executed relatively infrequently.

Table 10-1. Class C, Level 4
Main Processor

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Routing/Scheduling	.033	900	30	12,500	375,000
File Management:					
Enter Vessel	.008	20	.16	2,500	400
Modify Vessel	.0008	5	.004	2,500	10
Delete Vessel	.008	25	.2	2,500	500
Enter Passage	.024	30	.72	2,500	1,800
Identify Passage	.036	11	.396	2,500	990
Update Passage	.32	5	1.60	2,500	4,000
Modify Passage	.008	5	.04	2,500	100
Enter Communication	.20	8	1.60	2,500	4,000
Delete Passage	.024	23	.552	2,500	1,380
Change Status	.048	5	.24	2,500	600
Environmental File	4	1	4	1,000	4,000
Operating System					187,500
			39.512		580,280

Table 10-2. Class C, Level 4
Routing and Scheduling Processor

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Potential Collision:					
Stage 1	.033	600	20	24,273	485,460
Stage 2	.067	1	.067	27,000	1,800
Stage 3	.167	1	.167	22,725	3,800
Vessel Position Update	150	1	150	1,000	150,000
Position Sensor Data Process.	3	1	3	360	1,080
Tracker Data Processing	12.5	1	12.5	1,200	15,000
Operating System					187,500
			185.734		844,640

Table 10-3. Class C, Level 4
Position/Collision Processor

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Watchstander Initialization	~0		~0	*	1,500
WS Command	1.29	1	1.29	350	450
Data Entry	9.51	1	9.51	300	2,850
Alpha Numeric Display	6	1	6	2,200	13,200
Alert Display	1	1	1	200	200
Map Display	.016	1	.016	6,200	1,000
Vessel Position Display	150	1	150	245	36,750
Operating System					187,500
			167.806		243,450

*This function is executed relatively infrequently.

Table 10-4. Class C, Level 4 Display Processor

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Potential Collision:					
Stage 1	.033	100	3.3	22,923	76,480
Stage 2	.067	1	.067	15,000	900
Stage 3	.167	1	.167	12,625	2,020
Vessel Position Update	83	1	83	1,000	83,000
Position Sensor Data Process	23	1	23	360	8,300
Route Stray	5	1	5	1,000	5,000
Lane Stray	5	1	5	1,000	5,000
Dangerous Encounters	5	1	5	500	2,500
Congestion Analysis	5	1	5	200	1,000
Anchor Drift	3	1	3	300	900
Navaid Missing	33	1	33	300	9,900
Demand Calculation:					
Relative Position	1	1	1	1,500	1,500
Dangerous Encounter	1	1	1	500	500
CPA	1	1	1	300	300
Local Traffic	1	1	1	5,400	5,400
Data Retrieval:					
Searches	1	40	40	2,500	100,000
Environmental	1	1	1	5,000	5,000
Notices lookup	1	1	1	10,000	10,000
Alert Response	1	4	4	5,000	20,000
Alert Posting	.3	1	.3	1,000	300

Table 10-5. Class B, Level 4
Main Processor

Page 1 of 3

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Scenario Creation	.1	1	.1	600	60
Scenario Editing	.1	10	1	5,100	5,100
Scenario Playback	65	1	65	2,015	131,000
Potential Grounding	5	1	5	5,200	26,000
Excessive Vessel Speed	3	1	3	300	900
System Backup Data	3	1	3	250	750
System Parameter Adjustment	~0		~0	*	600
Vessel Passage History	5	1	5	1,000	5,000
Operations Log	5	1	5	1,000	5,000
Daily Traffic Management	.260	1	.260	770	200
VTS Report Management	~0		~0	*	5,000
Inter VTS Communication	~0		~0	*	5,000
Display Distribution	2	1	2	300	600
VTS Initialization	~0		~0	*	5,000
Environmental Data Process.	25	1	25	200	5,000
Tracker Data Processing	7	1	1	1,200	8,400

Table 10-5. Class B, Level 4
Main Processor

Page 2 of 3

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
File Management:					
Enter Vessel	.0012	20	.024	2,500	60
Modify Vessel	.00012	5	.0006	2,500	2
Delete Vessel	.0012	25	.03	2,500	75
Enter Passage	.004	30	.12	2,500	300
Identify Passage	.008	11	.008	2,500	220
Update Passage	.056	5	.28	2,500	700
Modify Passage	.004	5	.02	2,500	50
Enter Communication	.036	8	.288	2,500	720
Delete Passage	.004	23	.092	2,500	230
Change Status	.008	5	.04	2,500	100
Environmental File	.017	1	.017	1,000	17
Operating System					187,500
			322.1936		731,580

*These functions are executed relatively infrequently.

Table 10-5. Class B, Level 4 Main Processor

Page 3 of 3

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Watchstander Initialization	~0		~0	*	1,500
WS Command	1.04	1	1.04	350	350
Data Entry	1	1	1	300	300
Alpha Numeric Display	3	1	3	2,200	6,600
Alert Display	1	1	1	200	200
Map Display	.016	1	.016	6,200	1,000
Vessel Position Display	83	1	83	235	19,500
Operating System					187,500
			89.056		216,950

*This function executed relatively infrequently.

Table 10-6. Class B, Level 4
Display Processor

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Route Stray	.036	1	.036	10,000	360
Dangerous Encounters	3	1	3	500	1,500
Congestion Analysis	3	1	3	2,000	6,000
Excessive Vessel Speed	3	1	3	300	900
Vessel Position Update	.036	1	.036	10,000	360
Demand Calculation:					
Relative Position	.017	1	.017	15,000	255
Dangerous Encounter	.017	1	.017	500	9
CPA	.017	1	.017	300	5
Local Traffic	1	1	1	3,600	3,600
Data Retrieval:					
Searches	1	40	40	2,500	100,000
Environmental	.017	1	.017	5,000	85
Notice Lookup	.017	1	.017	10,000	170
Alert Response	1	4	4	5,000	20,000
Alert Posting	.1	1	.1	1,000	100
Scenario Creation	.1	1	.1	600	60

Table 10-7. Class A, Level 1
Main Processor

Page 1 of 3

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Scenario Editing	.1	10	1	5,100	5,100
Scenario Playback	40	1	40	2,000	80,000
System Backup Data	2	1	2	250	500
System Parameter Adjustment	~0		~0	*	600
Vessel Passage History	1	1	1	1,000	1,000
Operations Log	1	1	1	1,000	1,000
Daily Traffic Summary	.104	1	.104	770	80
Display Distribution	2	1	2	300	600
Environmental Data Process.	5	1	5	200	1,000
Inter VTS Communication	~0		~0	*	5,000
Position Sensor Data	1	1	1	360	360
VTS Initialization	~0		~0	*	5,000
VTS Report Management	~0		~0	*	5,000

Table 10-7. Class A, Level 1
Main Processor
Page 2 of 3

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
File Management:					
Enter Vessel	.0008	20	.016	2,500	40
Modify Vessel	.00008	5	.0004	2,500	1
Delete Vessel	.0008	25	.02	2,500	50
Enter Passage	.0024	30	.072	2,500	180
Identify Passage	.004	11	.044	2,500	110
Update Passage	.036	5	.180	2,500	450
Modify Passage	.0008	5	.004	2,500	10
Enter Communication	.024	8	.192	2,500	480
Delete Passage	.0024	23	.0552	2,500	138
Change Status	.0048	5	.024	2,500	60
Environmental File	.017	1	.017	1,000	17
Operating System					187,500
			108.0856		427,680

*These functions executed relatively infrequently.

Table 10-7. Class A, Level 1
Main Processor

Page 3 of 3

Processor Waiting Time/Class A, Main Processor

The service times for each function in the Main processor are summarized in Table 10-7. The utilization is .342, the average service time is 1.662 ms., the standard deviation is .826 ms., and the average waiting time is .539 ms.

Processor Waiting Time/Class A, Display Processor

The service times for each function in the Display processor are summarized in Table 10-8. The utilization is .156, the average service time is 1.155 ms., the standard deviation is .532 ms., and the average waiting time is .129 ms.

Disk Access Waiting Time/Class C, Level 4 System

There are three disks in a Class C system. The first disk handles file updates and searches. The second disk handles file updates and potential groundings. The third disk is used for routing and scheduling. It must also keep an updated copy of the files. The number of times each function is performed and the number of 1-sector, 2-sector, 10-sector, 30-sector and 40-sector reads and writes for each function are summarized in Table 10-9.

The average number of sectors transferred per second and the standard deviation are listed below:

	Sector/sec	Standard deviation
. Disk 1	4.362	4.056 sectors
. Disk 2	19.073	19.051
. Disk 3	15.658	13.607

To compute the waiting time for the disk access we must also compute the average and standard deviation of the time to locate the cylinder and the average and standard deviation of the time to locate the data on the cylinder. We have assumed that the seek time is a linear function of the number of cylinders which must be skipped and that the latency time is uniformly distributed between 0 ms. and 16.67 ms.

	Number of times function is executed per second	Number of processing segments per function	Number of processing segments per second	Number of cycles per segment	Number of cycles per second
Watchstander Initialization	~0		~0	*	1,500
WS Command	1.03	1	1.03	350	350
Data Entry	.06	1	.06	300	20
Alpha Numeric Display	2	1	2	2,200	4,400
Alert Display	1	1	1	200	200
Map Display	.016	1	.016	6,200	1,000
Vessel Position Display	.036	1	.036	210	10
Operating System					187,500
			4.142		194,980

*This function executed relatively infrequently.

Table 10-8. Class A, Level 1 Display Processor

	Iterations per second	Number of Reads	Number of Writes	Number of 1-Sector R/W	2-Sector R/W	10-Sector R/W	30-Sector R/W	40-Sector R/W	Disk #1	Disk #2	Disk #3
Enter Vessel	.008	13	7	3	17	-	-	-	X	X	X
Modify Vessel	.0008	4	1	2	3	-	-	-	X	X	X
Delete Vessel	.008	16	9	9	16	-	-	-	X	X	X
Enter Passage	.024	19	11	2	28	-	-	-	X	X	X
Identify Passage	.036	7	4	2	9	-	-	-	X	X	X
Update Passage	.32	4	1	2	3	-	-	-	X	X	X
Modify Passage	.008	4	1	2	3	-	-	-	X	X	X
Enter Communication	.20	6	2	4	4	-	-	-	X	X	X
Delete Passage	.024	14	9	2	21	-	-	-	X	X	X
Change Status	.048	4	1	2	3	-	-	-	X	X	X
Update Env. File	.017	0	1	1	-	-	-	-	X	X	X
Routing	.125	45	0	-	-	-	45	-			X
Grounding	.033	125	0	-	-	-	-	125		X	
Searches	1	4	0	-	-	4	-	-	X		
Other Demand Inc. Local Traffic	.2778	10	0	10	-	-	-	-	X		

Table 10-9. Class C, Level 4. Disk Accesses.

	Iterations per second	Number of Reads	Number of Writes	Number of 1-Sector R/W	Number of 2-Sector R/W	Number of 10-Sector R/W	Number of 30-Sector R/W	Number of 40-Sector R/W
Enter Vessel	.0012	13	7	3	17	-	-	-
Modify Vessel	.00012	4	1	2	3	-	-	-
Delete Vessel	.0012	16	9	9	16	-	-	-
Enter Passage	.004	19	11	2	28	-	-	-
Identify Passage	.008	7	4	2	9	-	-	-
Update Passage	.056	4	1	2	3	-	-	-
Modify Passage	.004	4	1	2	3	-	-	-
Enter Communication	.036	6	2	4	4	-	-	-
Delete Passage	.004	14	9	2	21	-	-	-
Change Status	.008	4	1	2	3	-	-	-
Update Env. File	.017	0	1	1	-	-	-	-
Grounding	.033	125	0	-	-	-	-	125
Searches	1	4	0	-	-	4	-	-
Other Demand Inc. Local Traffic	.046	10	0	10	-	-	-	-

Table 10-10. Class B, Level 4. Disk Accesses.

For the first disk the utilization is .474, the average service time is 40.150 ms., the standard deviation is 10.205 ms. and the average waiting time is 19.275 ms.

For the second disk the utilization is .426, the average service time is 46.28 ms., the standard deviation is 12.818 ms., and the average waiting time is 18.468 ms.

For the third disk the utilization is .481, the average service time is 44.857 ms., the standard deviation is 11.551 ms., and the average waiting time is 22.121 ms.

Disk Access Waiting Time/Class B, Level 4 System

In the Class B system there are two disks. One disk is able to perform all of the disk access functions. The second disk keeps updated copies of the files and serves as an on-line backup.

The number of times each function is executed and the number of reads and writes for a Class B system are summarized in Table 10-10.

The utilization for each disk is .456, the average service time is 47.471 ms., the standard deviation is 12.130 ms. and the average waiting time is 21.195 ms.

Bus Waiting Time/Class C, Level 4

An estimate of the interprocessor communication load was made in the Phase I report. Position input data, data base writes, watchstander requests, display replaces, alert notices and status messages will generate 296,584 bits of data per second. In addition, we have assumed 74,146 bits per second (25% of 296,584) of data from functions not included above and 74,146 bits per second for short non-packetized instructions and acknowledgements.

	Transmitted per second	Bits per Transmission*	Bits per second
Packetized messages	188.375	2096	394,834
Short messages	386.177	304	117,398

*Includes control token.

Table 10-11. Class C, Level 4
Interprocessor Communication Load

	Transmitted per second	Bits per Transmission*	Bits per second
Packetized messages	59.599	2096	124,920
Short messages	122.177	304	37,142

*Includes control token.

Table 10-12. Class B, Level 4
Interprocessor Communication Load

	ρ, Utilization Factor	$\bar{s}$, Average Service Time, ms.	σ_s, Standard Deviation of s, ms.	$\overline{tw}$, Average Waiting Time, ms.	$\overline{tq} = \overline{tw} + \bar{s}$, ms.
Class C					
Processors:					
Main	.651	1.214	1.711	3.374	4.588
Routing	.464	7.953	3.648	4.166	12.119
Pos/Coll	.676	2.830	5.800	15.341	18.171
Display	.195	0.260	0.290	0.071	0.331
Disks:					
Main #1	.474	40.150	10.205	19.275	59.425
Main #2	.426	46.280	12.818	18.468	64.748
Routing	.481	44.857	11.551	22.121	66.978
Bus:	.360	.654	.565	.368	1.022
Class B					
Processors:					
Main	.585	1.273	1.957	3.020	4.293
Display	.174	0.251	0.284	0.060	0.311
Disk:					
Main #1, #2	.456	47.471	12.130	21.195	69.666
Bus:	.114	.653	.561	.083	.736
Class A					
Processor:					
Main	.342	1.662	.826	.539	2.201
Display	.156	1.155	.532	.129	1.284
Disk:					
Main #1, #2	<.456	<47.471	<12.130	<21.195	<69.666
Bus:	<.114	< .594	< .561	< .083	< .736

Table 10-13. Average Waiting Times

Each message packet will have 1968 bits of data and 80 bits of header. The non-packetized messages will have 192 bits of data and 64 bits of overhead. In addition, there will be at least one control token (48 bits) between messages.

The utilization is .360, the average message service time is .654 ms., the standard deviation is .565 ms., and the average waiting time is .368 ms.

Bus Waiting Time/Class B, Level 4

The interprocessor communication load for position input data, data base writes, watchstander requests, display replaces, alert notices and status messages is 93,832 bits per second. We have assumed 25% additional packetized messages and 25% for short instructions and acknowledgements.

The utilization is .114, the average message service time is .653 ms., the standard deviation is .561 ms. and the average waiting time is .083 ms.

All of the waiting times estimates for the processors, disks and busses are summarized in Table 10-13.

10.2 FUNCTION RESPONSE TIMES

In this section we will compute the average response times for the Enter Vessel, Enter Passage and Search on a Key functions. These functions have been evaluated because of the large number of disk accesses required by each function.

10.2.1 Enter Vessel

Case 1: Record not in Vessel File

The watchstander's actions and the system's responses are listed below:

1. W/S: Types "ENTER VESSEL".
   System: Returns blank vessel record form.
   (1 second allowed response time)
2. W/S: Fills in vessel name.
   System: Searches Vessel File. If the record is not there, the system advances cursor to the next line of the form.
   (3 seconds allowed response time)
3. W/S: Completes the vessel record form and types "ENTER".
   System: Adds the record to all copies of the Vessel File.
   (10 seconds allowed response time)

The average response time for the first system action in this case is 2.509 ms. for a Class C system and 2.324 ms. for a Class B system. (Figure 10-14 and Table 10-15.)

The average response time for the second system action in this case is 197.695 ms. for a Class C system and 201.551 ms. for a Class B system. (Figure 10-16 and Table 10-17.)

The average response time for the third system action is 1384.693 ms. for a Class C system and 1332.110 ms. for a Class B system. (Figure 10-18 and Table 10-19.)

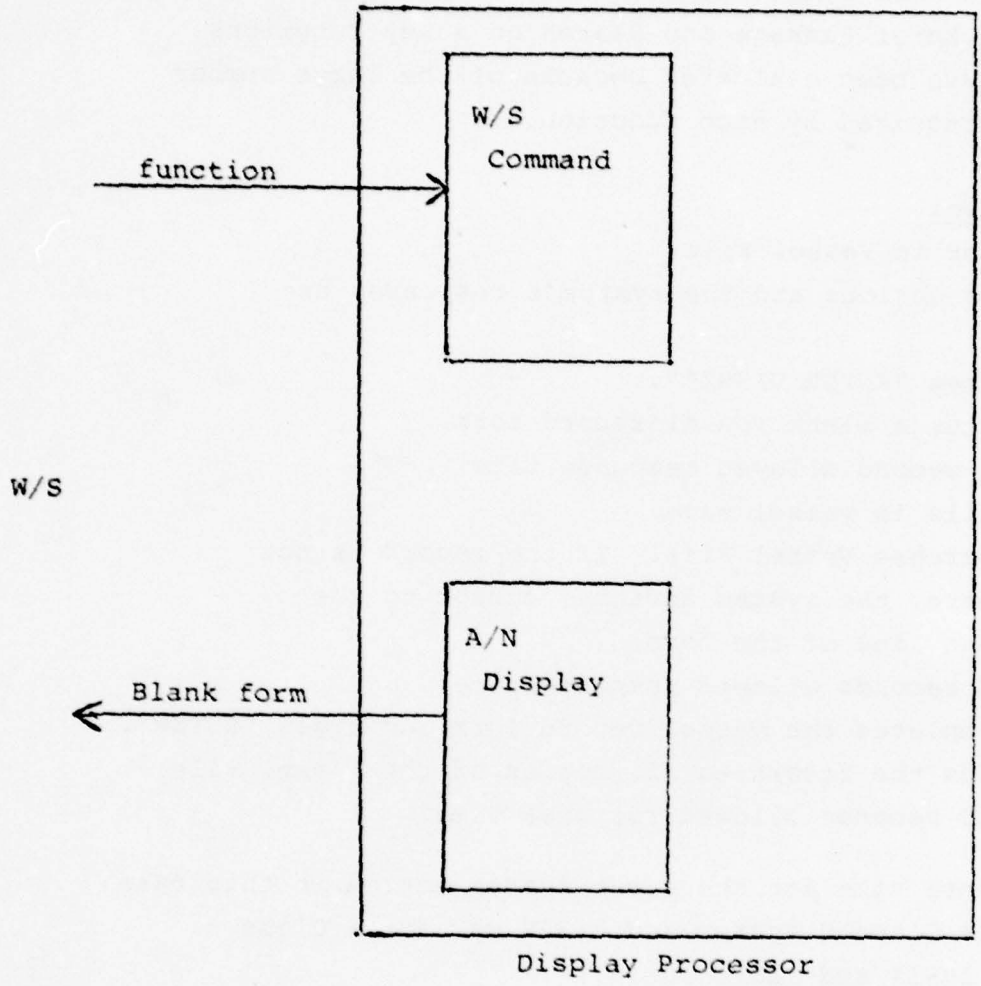


Figure 10-14. Return a Blank Form to Watchstander.

		Class C ms.	Class B ms.
Processor:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	2 Display Waits	.142	.120
Bus:	None		
		2.182	2.160
	Overhead (15%)	.327	.324
Disk:	None		
		2.509 ms.	2.324 ms.

Table 10-15. Return Blank Form to Watchstander

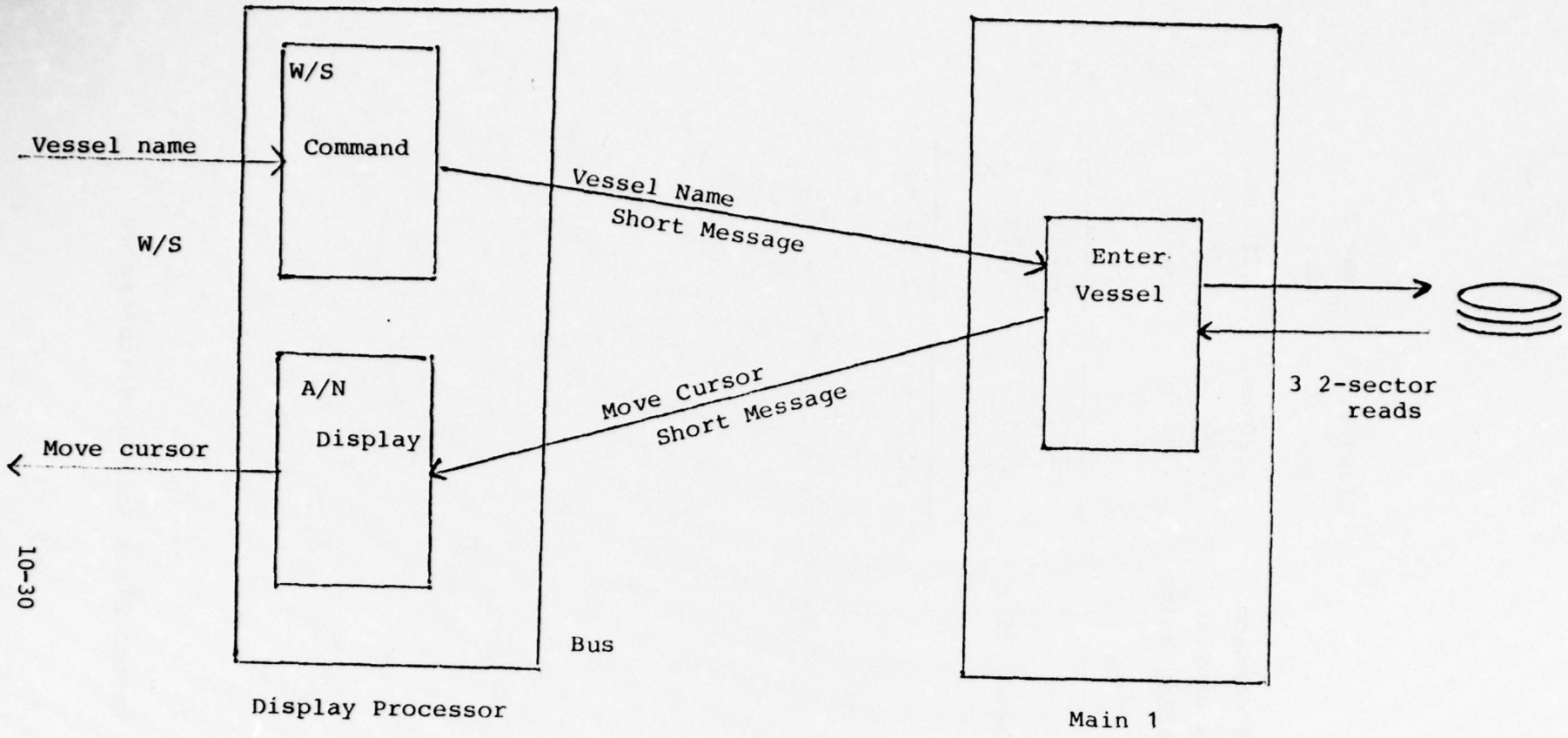


Figure 10-16. Case 1: Vessel Record not in Vessel File

		Class C ms.	Class B ms.
Processor:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	3 Reads	6.000	6.000
	2 Display Waits	.142	.120
	3 Main Waits	10.124	9.060
Bus:	2 Short Messages	.405	.405
	2 Bus Waits	.736	.166
	SUBTOTAL	19.447	17.791
	Overhead*	2.917	2.669
Disk:	3 2-Sector Reads	117.506	117.506
	3 Disk Waits	57.825	63.585
	TOTAL	197.695 ms.	201.551 ms.

*15% of Subtotal

Table 10-17. Search Vessel File. Case 1.

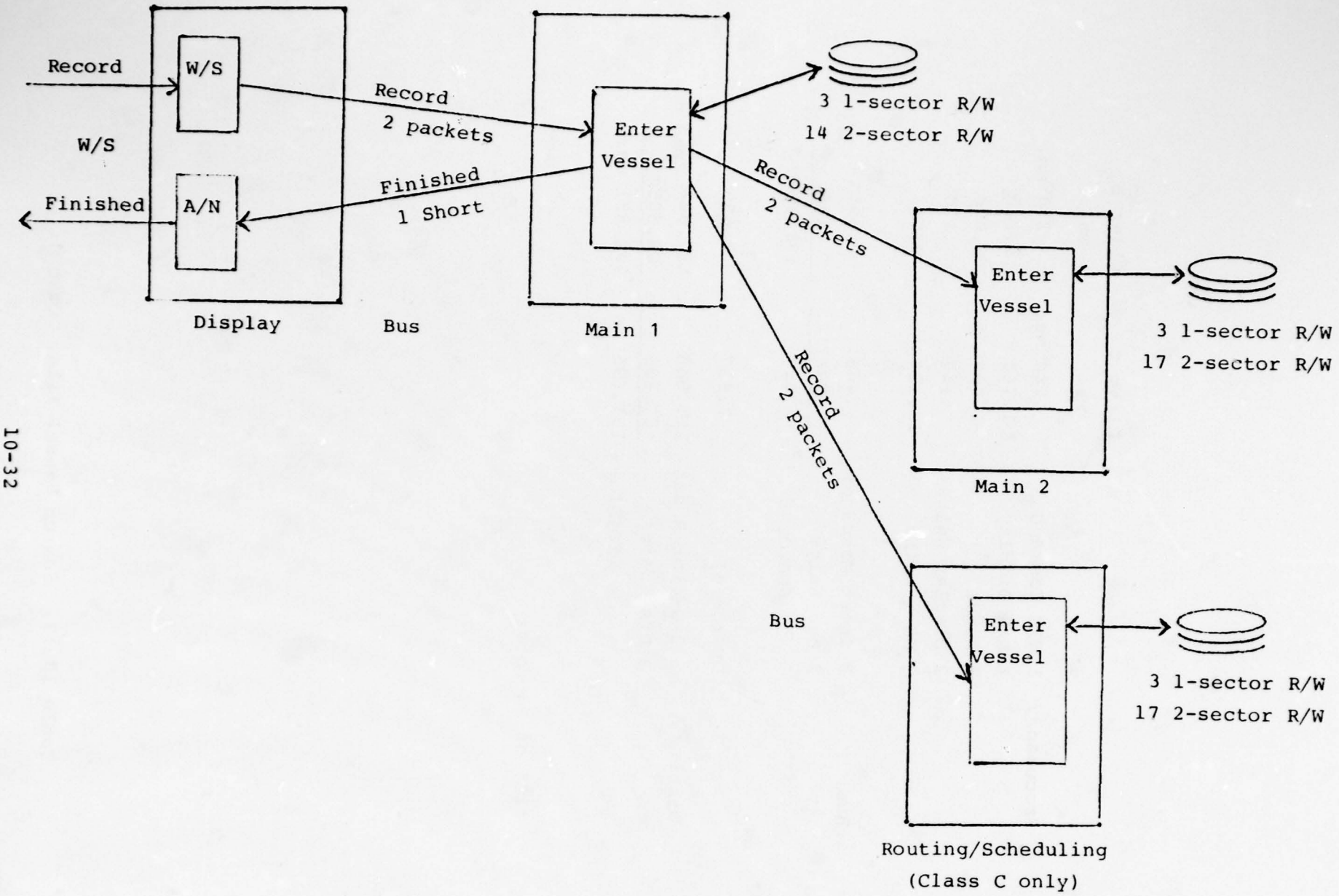


Figure 10-18. Case 1: Record not in Vessel File.
Add record to Vessel File.

		Class C ms.	Class B ms.
Processor:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	20 Read/Write	40.000	40.000
	2 Display Waits	.142	.120
	20 Main/Router Waits	83.320	
Bus:	1 Short Message	.201	60.400
	Packet (C=6, B=4)	8.384	5.589
	Bus Waits (C=7, B=5)	2.576	.415
	SUBTOTAL	136.665	108.767
	Overhead*	20.500	15.315
	Packetizing Overhead (.5 ms./packet)	3.000	2.000
Disk:	3 1-sector R/W	116.255	116.255
	17 2-sector R/W	665.873	665.873
	20 Disk Waits	442.420	423.900
	TOTAL	1384.693 ms.	1332.110 ms.

*15% of Subtotal

Table 10-19. Add to Vessel File. Case 1.

Case 2: Vessel Record already in Vessel File

The watchstander's actions and the system's responses are listed below:

1. W/S: Types "ENTER VESSEL"
   System: Returns blank vessel record form.
   (1 second allowed response time)
2. W/S: Fills in vessel name
   System: Search Vessel File. Returns the record to watchstander and switches to the Modify Vessel function.
   (3 seconds allowed response time)

The average response time for the first system action is the same as Case 1. (Figure 10-14 and Table 10-15.)

The average response time for the second system action is 226.228 ms. in a Class C system and 271.348 ms. in a Class B system. (Figure 10-20 and Table 10-21.)

10.2.2 Enter Passage

Case 1: Record is in Vessel File

The watchstander's actions and the system's responses are listed below:

1. W/S: Types "ENTER PASSAGE"
   System: Returns a list of four ways to identify a vessel.
   (1 second allowed response time)
2. W/S: Identifies the vessel in one of four ways.
   System: Searches Vessel File. If record there returns the vessel information and a blank passage form.
   (10 seconds allowed response time)
3. W/S: Completes passage information and types "ENTER".
   System: Adds the passage record to all copies of the passage file.
   (3 seconds allowed response time)

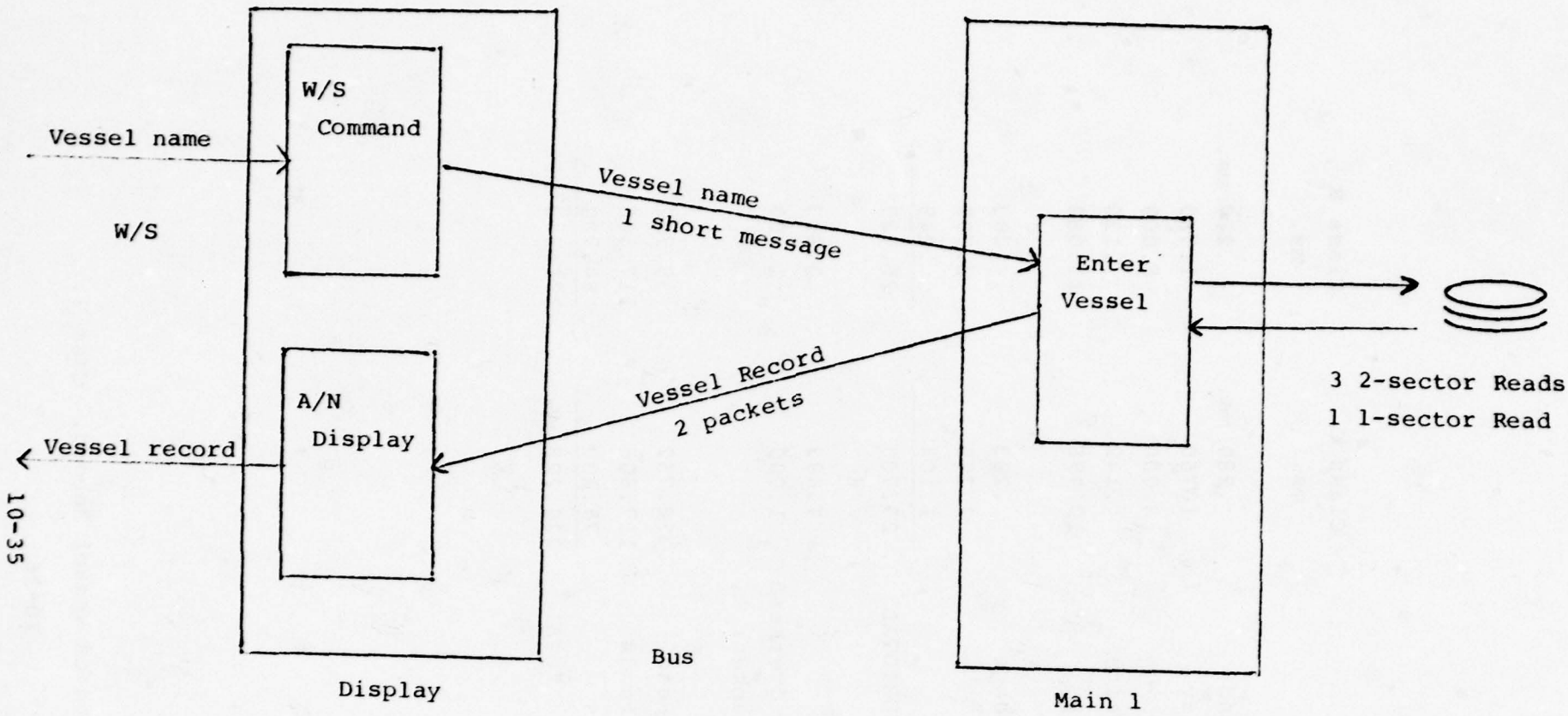


Figure 10-20. Case 2: Vessel Record Already in Vessel File. Locate Vessel Record.

		Class C ms.	Class B ms.
Processor:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	4 Reads	8.000	8.000
	2 Display Waits	.142	.120
	4 Main Waits	13.496	12.080
Bus:	1 Short message	.203	.203
	2 Packets	2.795	2.795
	3 Bus Waits	1.104	.249
	SUBTOTAL	27.780	25.487
	Overhead*	4.092	3.823
	Packetizing Overhead (.5 ms./packet)	1.000	1.000
Disk:	1 1-sector Read	38.752	38.752
	3 2-sector Reads	117.506	117.506
	4 Disk Waits	77.100	84.780
	TOTAL	226.228 ms.	271.348 ms.

*15% of Subtotal

Table 10-21. Locate Vessel Record. Case 2.

The average response time for the first system action is the same as the response time to return a blank form in the Enter Vessel function: 2.509 ms. for a Class C system and 2.324 ms. for a Class B system.

The average response time for the second system action is 263.774 ms. in a Class C system and 269.145 ms. in a Class B system. (Figure 10-22 and Table 10-23.)

The average response time for the third system action is 1795.256 ms. in a Class C system and 1730.190 ms. in a Class B system. (Figure 10-24 and Table 10-25.)

Case 2: Record not in Vessel File

The watchstander's actions and the system's responses are listed below:

1. W/S: Types "ENTER PASSAGE"
   System: Returns a list of four ways to identify a vessel. (1 second allowed response time)
2. W/S: Identifies the vessel in one of four ways.
   System: Searches Vessel File. If record is not in Vessel File, the system switches to the Enter Vessel function and displays a blank vessel record. (10 seconds response time)

The average response time for the first system action is the same as Case 1 of Enter Passage.

The average response time for the second system action is 197.694 ms. in a Class C system and 201.552 ms. in a Class B system. (Figure 10-26 and Table 10-27.)

10.2.3 Search on a Key

The watchstander's action and the system response are listed below:

W/S: Enter 1-10 data base item names and relational operators.

System: The system searches up to 10,000 records and returns up to 10,000 characters of output. (Allowed response time is 2 minutes plus 2 minutes per data base item in the key)

The average system response time for a search of 10,000 records is 2.06 minutes in a Class C system and 2.03 minutes in a Class B system. (Figure 10-28 and Table 10-29.)

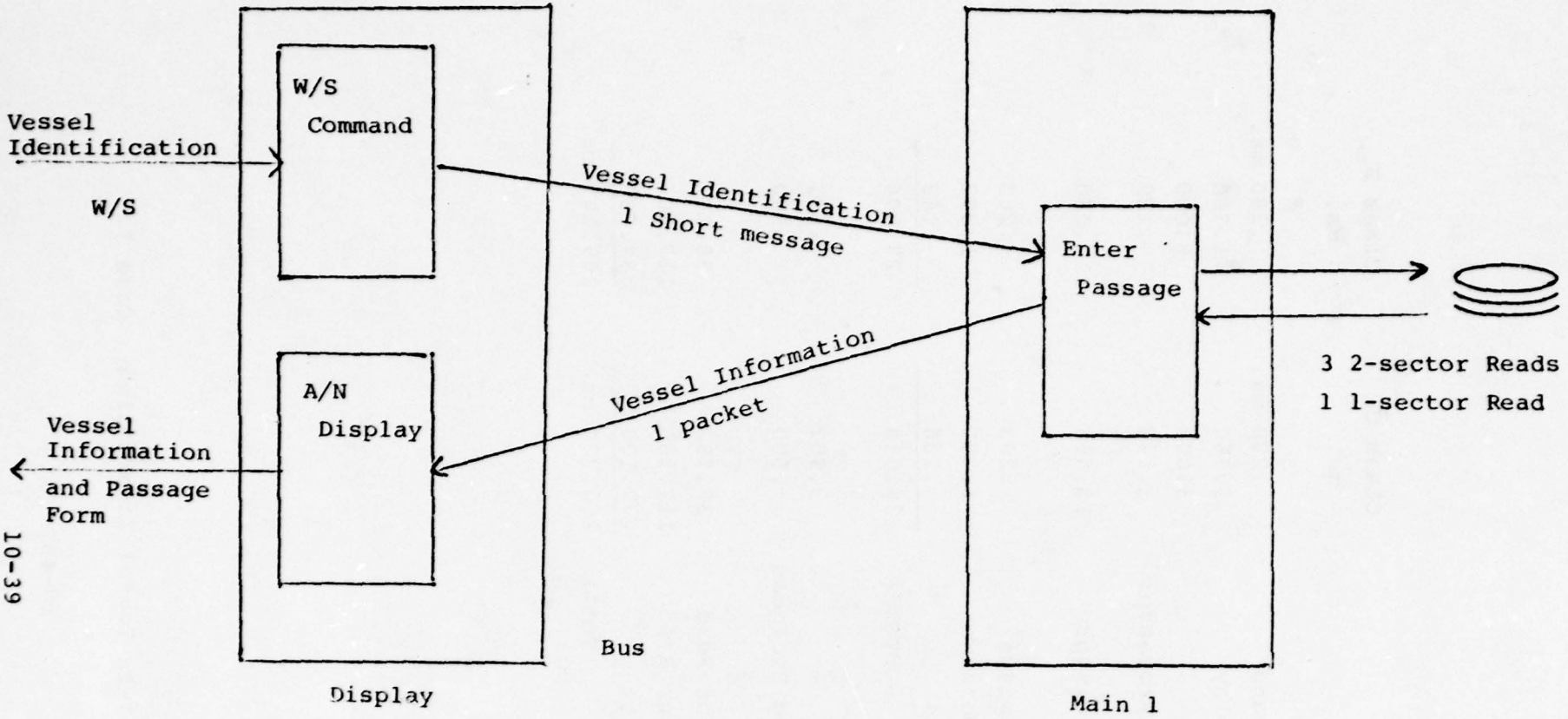


Figure 10-22. Return Part of Vessel Record File.

		Class C ms.	Class B ms.
Processor:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	4 Reads	8.000	8.000
	2 Display Processor Waits	.142	.120
	4 Main Processor Waits	13.496	12.080
Bus:	1 Short Message	.203	.203
	1 Packet Message	1.397	1.397
	2 Bus Waits	.736	.166
	SUBTOTAL	26.014	24.006
	Overhead*	3.902	3.601
	Packetizing Overhead	.500	.500
Disk:	1 one-sector Read	38.752	38.752
	3 two-sector Read	117.506	117.506
	4 Disk Waits	77.100	84.780
	TOTAL	263.774 ms.	269.145 ms.

*15% of Subtotal

Table 10-23. Return Vessel Information. Case 1.

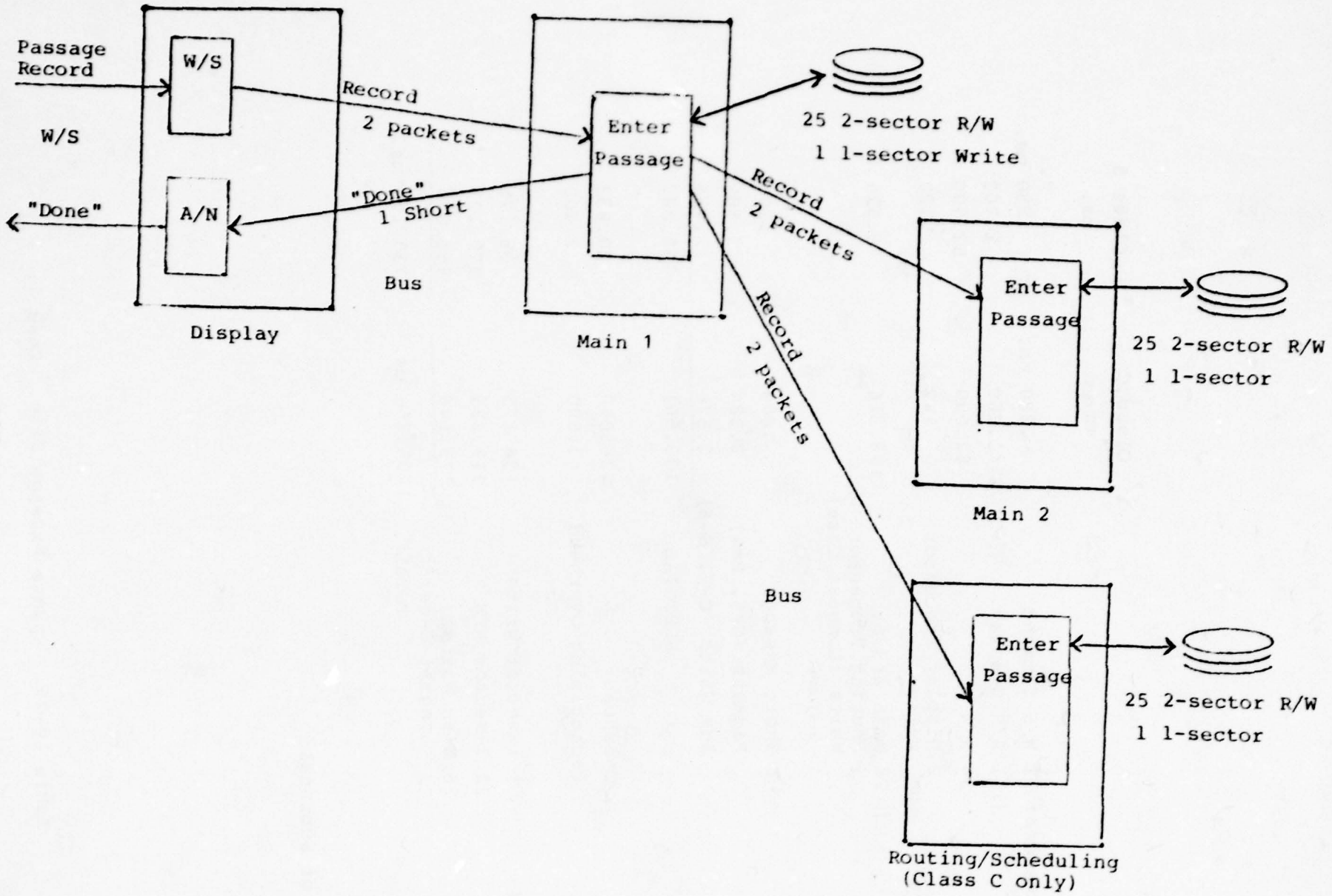


Figure 10-24. Update Passage File. Case 1.

			Class C ms.	Class B ms.
Processors:	1	W/S Command	.280 ms.	.280 ms.
	1	A/N Display	1.760	1.760
	26	R/W	52.000	52.000
	2	Display Processor Waits	.142	.120
	26	Main and/or Routing Processor Waits (Longest Case)	108.316	78.520
Bus:	1	Short Message	.203	.203
		Packets (C=6, B=4)	8.384	5.589
		Bus Waits (C=7, B=5)	2.576	.415
		SUBTOTAL	173.661	138.887
		Overhead*	26.049	20.833
		Packetizing Overhead	3.000	2.000
Disk:	1	1-sector Write	38.175	38.175
	25	2-sector R/W	979.225	979.225
	26	Disk Waits (Longest Case)	575.146	551.070
		TOTAL	1795.256 ms.	1730.190 ms.

*15% of Subtotal

Table 10-25. Update Passage File. Case 1.

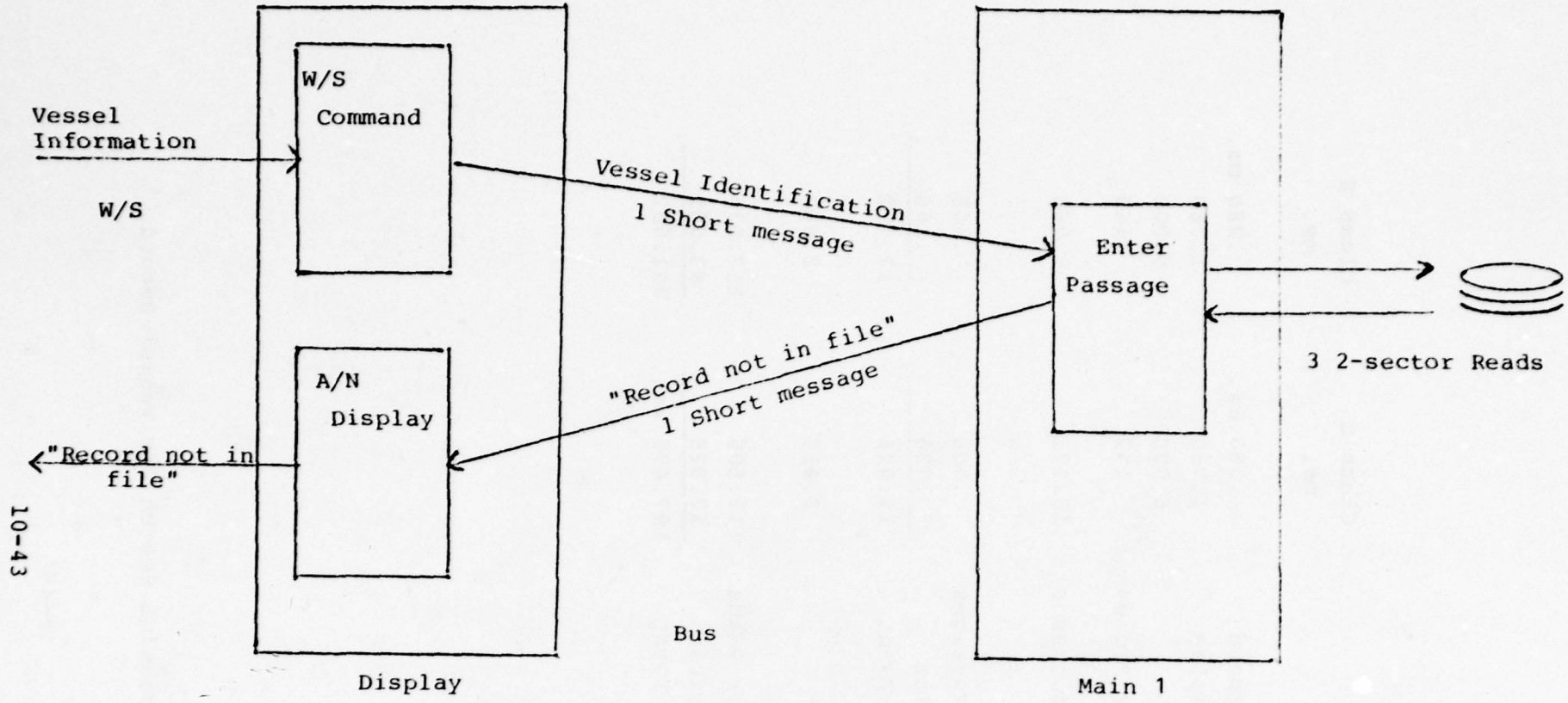


Figure 10-26. Unsuccessful Search for Vessel Record.

		Class C ms.	Class B ms.
Processors:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	3 Reads	6.000	6.000
	2 Display Processor Waits	.142	.120
	3 Main Processor Waits	10.122	9.060
Bus:	2 Short Messages	.406	.406
	2 Bus Waits	.736	.166
	SUBTOTAL	19.446	17.792
	Overhead*	2.917	2.669
Disk:	3 2-sector Reads	117.506	117.506
	3 Disk Waits	57.825	63.585
	TOTAL	197.694 ms.	201.552

*15% of Subtotal

Table 10-27. Unsuccessful Search for Vessel Record.

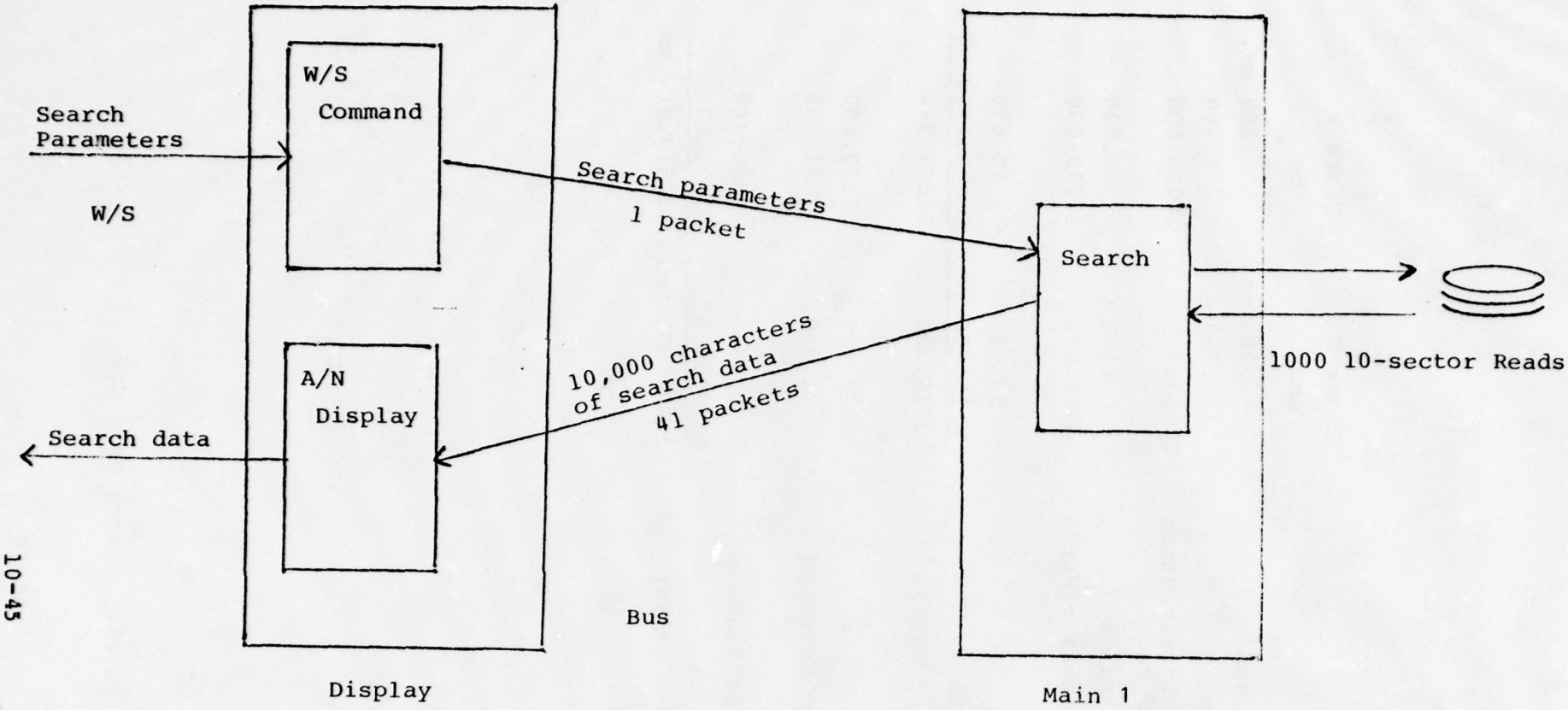


Figure 10-28. Search on a Key.

		Class C ms.	Class B ms.
Processors:	1 W/S Command	.280 ms.	.280 ms.
	1 A/N Display	1.760	1.760
	10,000 Records investigated (2 ms. / rec.)	20000.000	20000.000
	2 Display Waits	.142	.120
	10,000 Main Processor Waits	33740.000	30200.000
Bus:	42 Packets	58.688	58.688
	42 Bus Waits	15.456	3.486
	SUBTOTAL	53816.326	50264.344
	Overhead*	8072.449	7539.650
	Packetizing Overhead	21.000	21.000
Disk:	1,000 10-sector Reads	42500.000	42500.000
	1,000 Disk Waits	19275.000	21195.000
	TOTAL	123684.775 ms.	121519.984 ms.

*15% of Subtotal

Table 10-29. Search on a Key.

APPENDIX A

Sample Processing Power Evaluation Routine

A method is needed to rapidly screen candidate processors (minicomputers) to determine those which have a substantial probability of meeting the PDS (Processing/Display Subsystem) requirements. A way of accomplishing this is to run a benchmark process on each candidate processor and compare the time it takes the benchmark to run against a predetermined maximum allowable time. Processors taking longer than this time, even though they might be able to perform the VTS function, have too high of a risk that they will be inadequate, and thus will not be considered.

This appendix describes such a benchmark process. It is based upon the PDS stage 1 collision warning process, which is one of the most time critical processes in the PDS. The acceptable running time for this benchmark is 10 seconds or less. Figure A-1 is a flowchart describing the benchmark process. It should be noted that, while this routine presents the considerations required for an efficiently programmed collision warning stage 1 calculation, it is not intended as the final version of the stage 1 calculation routine. Rather, it presents an appropriate mix of machine instructions with which to guage the suitability of candidate processors.

The algorithm consists of two loop constructions, with the outer loop executed for all vessels in the vessel table, and the inner loop executed for all vessels between that vessel indexed by the outer loop and the end of the vessel table. This allows the position of each vessel to be compared to the position of every other vessel in the table, without duplication. Where N is the number of vessels, the inner loop will execute $\frac{N(N-1)}{2}$ times.

For the purposes of evaluating the execution time required by the program shown in Figure A-2, the following simplifying assumptions are made:

- The calculation of Δx and Δy always results in a negative value, requiring negation of each Δx and Δy;

- Of the Δx values calculated, one half of those values are less than the critical distance C, and require calculation of Δy values;

- All calculated Δy values are less than C, and require the subroutine call to "STAGE1A".

The timing calculation in Figure A-2 shows that, for 900 ships, the sample evaluation routine would require 8.10 seconds to execute on a processor* with 800 nanosecond memory cycle time. Other processors may be compared to the model by creating an assembly language program which conforms to the flowchart and performing the execution time calculation. This gives a rough indication of the processing power; however, capabilities unique to individual processors will generally not be comparable using this technique. Also, no indication of programming language efficiency is available without compilation and execution of a benchmark program.

*The assembly language shown is a modified form of the UNIVAC V77 assembly language. The timing shown does not, however, represent an actual V77 processor.

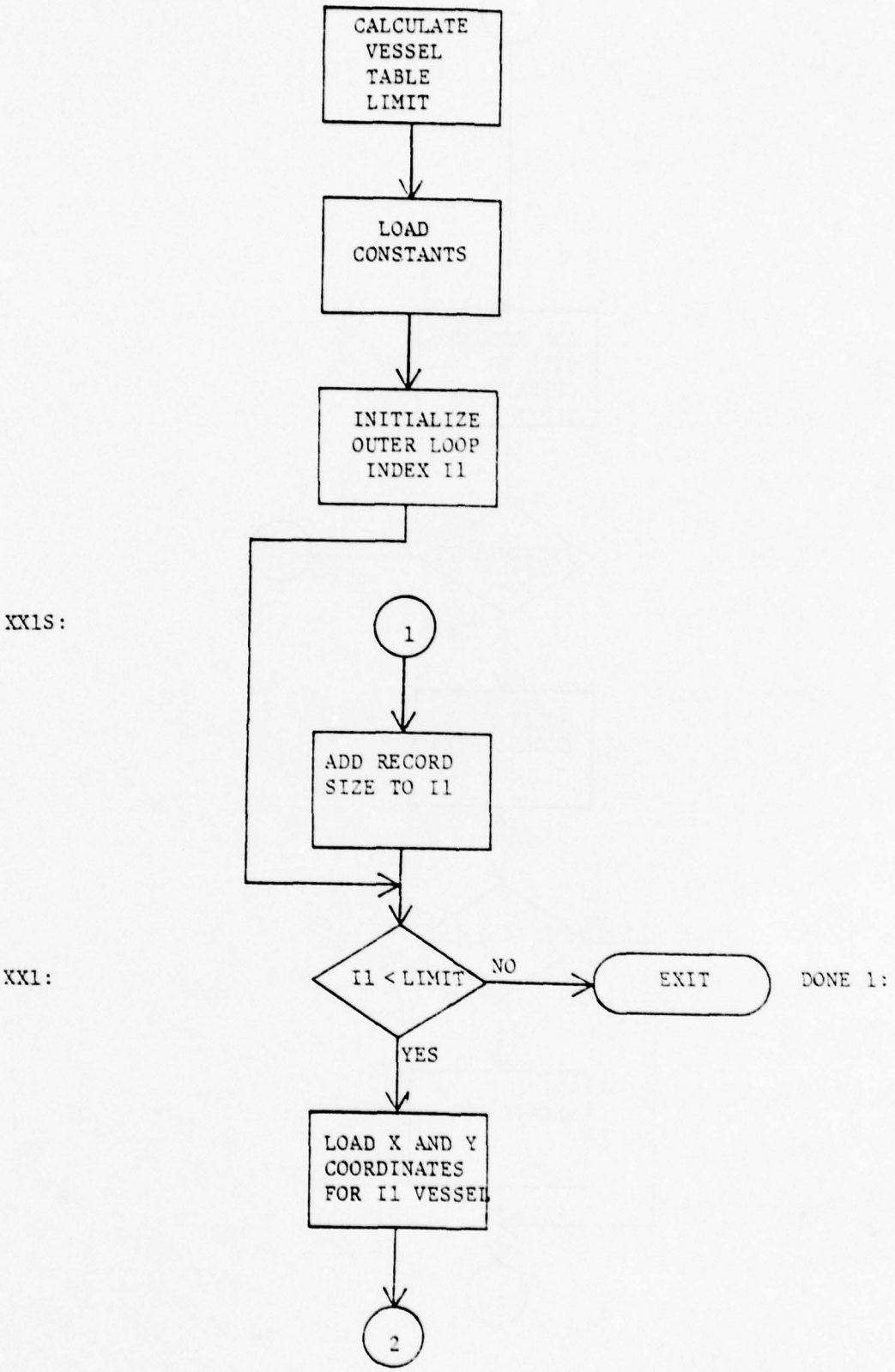
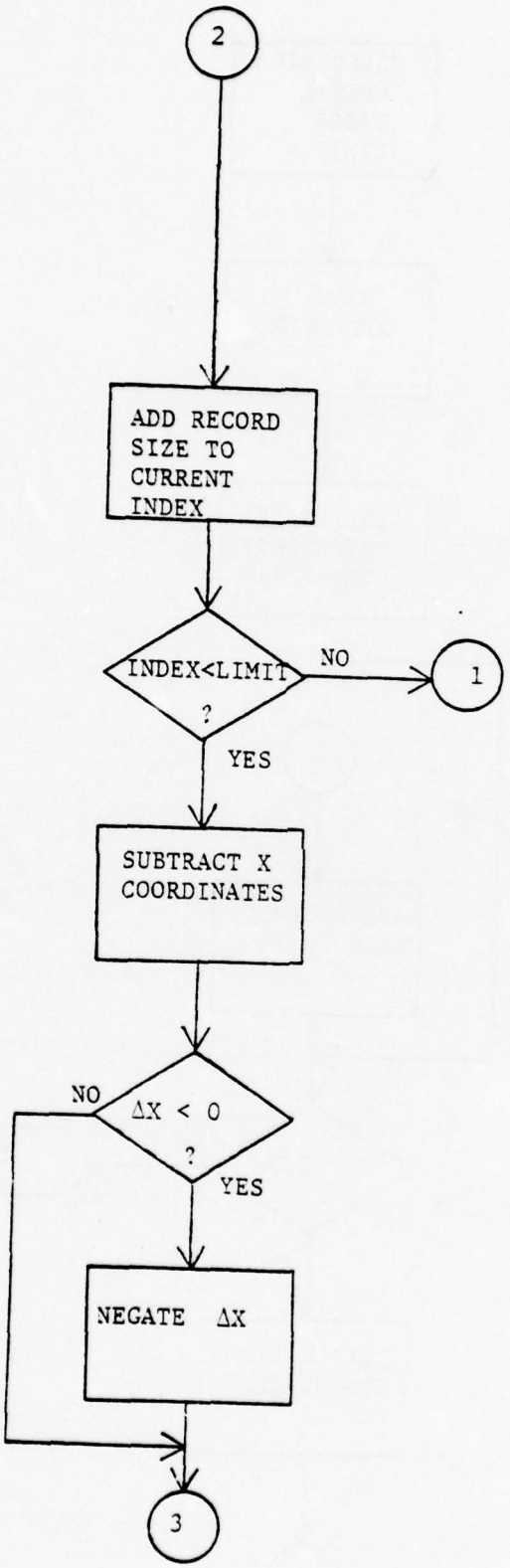
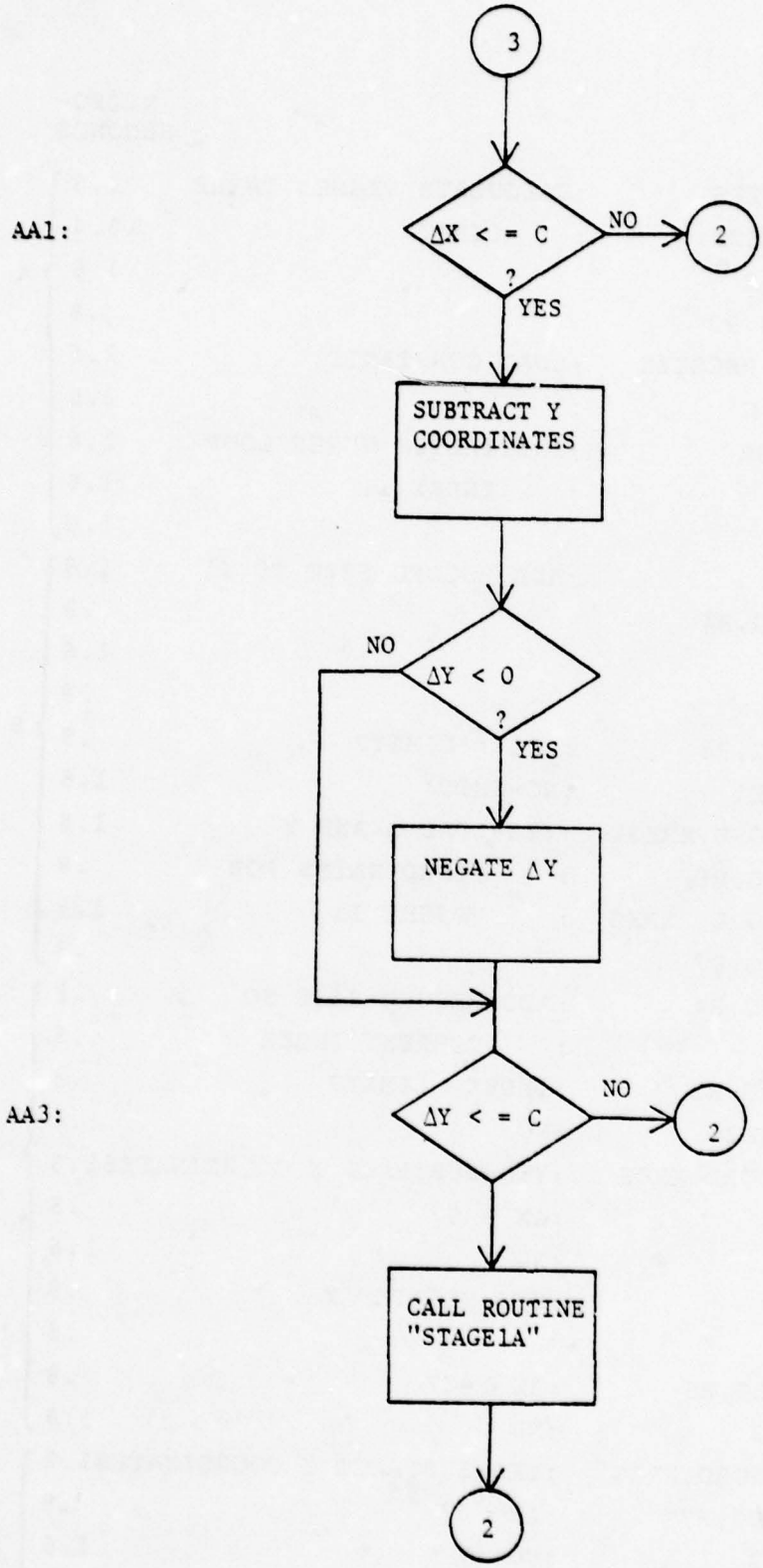


Figure A-1. Sample Evaluation Routine Flowchart

XX2S:


2
ADD RECORD SIZE TO CURRENT INDEX
INDEX<LIMIT ?
NO
1
YES
SUBTRACT X COORDINATES
NO
ΔX < 0 ?
YES
NEGATE ΔX
3


3
AA1:
ΔX < = C
?
NO
2
YES
SUBTRACT Y
COORDINATES
NO
ΔY < 0
?
YES
NEGATE ΔY
AA3:
ΔY < = C
NO
2
CALL ROUTINE
"STAGE1A"
2

```
                                                          MICRO-
                                                          SECONDS
        LDA     NRSHIPS         ;CALCULATE VESSEL TABLE    1.6 ┐
        MULI    RECSIZE         ;     LIMIT                5.4 │
        ADDI    TADDR                                      1.6 ├ A
        T       AREG,R3                                     .8 │
        LD      R4, RECSIZE     ;LOAD CONSTANTS            2.6 │
        LD      R5, C           ;                          2.6 │
        LDA     TADDR           ;INITIALIZE OUTER LOOP     1.6 │
        STA     I1              ;     INDEX I1             1.6 │
        JMP     XX1                                        1.6 ┘
XXLS:   LDX     I1              ;ADD RECORD SIZE TO I1     1.6 ┐
        ADR     XREG,R4                                     .8 │
        STX     I1                                         1.6 │
        TXA                                                 .8 ├ B
XX1:    SBR     AREG,R3         ; I1 < LIMIT?               .8 │
        JAP     DONE1           ;NO-EXIT                   1.6 │
        LDA     XCOORD,XREG     ;YES-LOAD X AND Y          1.6 │
        T       AREG,R6         ;     COORDINATES FOR       .8 │
        LDA     YCOORD, XREG    ;     VESSEL I1            1.6 │
        T       AREG,R7                                     .3 ┘
XX2S:   ADR     XREG,R4         ;ADD RECORD SIZE TO         .8 ┐
        TXA                     ;     CURRENT INDEX         .8 │
        SBR     AREG,R3         ;INDEX < LIMIT?             .8 │
        JAP     DONE2           ;NO                        1.6 │
        LDA     XCOORD,XREG     ;YES-SUBTRACT X COORDINATES1.6 ├ C
        SBR AREG,R6             ;ΔX < 0?                    .8 │
        JAP     AA1             ;NO                        1.6 │
        CPA                     ;YES-NEGATE ΔX              .8 │
        IAR                                                 .8 │
AA1:    SBR     AREG,R5         ;ΔX < =C?                   .8 │
        JAP     AA2             ;NO                        1.6 ┘
        LDA     YCOORD,XREG     ;YES-SUBTRACT Y COORDINATES1.6 ┐
        SBR     AREG,R7         ;ΔY < 0?                    .8 │
        JAP     AA3             ;NO                        1.6 │
        CPA                     ;YES-NEGATE ΔY              .8 ┘
```

Figure A-2. Sample Evaluation Routine Listing and Timing Calculation (Page 1 of 2)

```
          IAR                                                    .8 ┐
AA3:      SBR       AREG,R5       ;ΔY  = C                       .8 │
          JAP       AA2           ;NO                           1.6 │ D
          STX       I2            ;YES-CALL ROUTINE "STAGE1A"   1.6 │
          JSR       STAGE1A                                     1.6 │
          (PARAMETER LIST)                                          │
          LDX       I2                                            1.6┘
AA2:      JMP       XX2S                                          1.6 } C
DONE2:    JMP       XX1S                                          1.6 } B
DONE1:    (EXIT)
```

A TOTAL = 19.4 μ sec
B TOTAL = 13.6 μ sec
C TOTAL = 13.6 μ sec
D TOTAL = 12.8 μ sec

PROCESSING TIME FOR N= 900 SHIPS IS:

$$19.4 + 13.6N + 13.6 \times \frac{N(N-1)}{2} + \frac{12.8}{2} \times \frac{N(N-1)}{2}$$

$$= 19.4 + 12{,}240 + 5{,}501{,}880 + 2{,}589{,}120$$

$$= 8{,}103{,}259.4 \text{ or } 8.10 \text{ seconds}$$

Figure A-2. Sample Evaluation Routine Listing and Timing Calculation (Page 2 of 2)

APPENDIX B
OPERATING SYSTEM FUNCTIONS

This appendix contains a preliminary description of the functions of the operating system which will be available to applications. These functions constitute a subset of the functions which would be expected to be included in the final operating system for the VTS Processing/Display Subsystem. Modification of these functions may be required and additional functions will certainly be added before the design of the operating system is completed.

Descriptions are written in the form of a preliminary user's manual for the VTS operating system. As such it can serve as a preliminary external specification of the operating system.

B.1 PROCESS AND EVENT MANAGEMENT

B.1.1 Create Process

This is the mechanism for creating parallel execution paths. A process control block (PCG) is created for the designated process and the newly created process is scheduled for execution.

Calling Sequence:

CREATE process (EVENT NUMBER, SUSPEND, INDEPENDENCE, NAME, DATABLOCKSIZE, DATA BLOCK, PROCESSID, STATUS)

Parameters:

EVENTNUMBER: 1..16 used by created process to signal the creating process when finished (see POST Section B.1.4).

SUSPEND: (YESSUSPEND, NOSUSPEND) used to determine whether creating process should be suspended.

INDEPENDENCE: (free, child) indicates whether created process will act independently of creating process.

NAME: IDENTIFIER name of process to be scheduled.

DATABLOCK: INTEGER size of data block array sent by creating process.

DATABLOCK: ARRAY (1..DATABLOCK) datablock from creating process.

PROCESSID: INTEGER a variable parameter whose value at return will be the ID of created process.

STATUS: STATUSTYPE a variable parameter whose value at return will be status of Create process call. Status may be OK, illegal event number, or illegal name.

B.1.2 Schedule a Process at a Specific Time

Schedule allows the calling process to schedule another process at a specific time. It is similar to create process (B.1.1).

Calling Sequence

SCHEDULE (EVENTNUMBER, SUSPEND, INDEPENDENCE, NAME, TIME, DATABLOCKSIZE, DATABLOCK, CYCLIC, PROCESSID, STATUS)

Parameters

EVENTNUMBER 1..16 used by created process to signal creating process when finished.

SUSPEND: (YESSUSPEND, NOSUSPEND) used to determine whether or not to suspend calling process.

INDEPENDENCE: (free, child) indicates whether created process will be independent of calling process.

NAME: IDENTIFIER name of process to be scheduled.

TIME: TIMERECORD time that process to be scheduled.

DATABLOCKSIZE: INTEGER size of data block array sent by creating process.

DATABLOCK: ARRAY (1..DATABLOCK) data block from creating process.

CYCLIC: (NOCYCLIC, YESCYCLIC) indicates whether process is to be cyclically scheduled. If cyclic, the cycle is in time record.

PROCESSID: INTEGER a variable parameter whose value at return will be the ID of the created process.

STATUS: STATUSTYPE a variable parameter whose value at the return will be the status of the call. Returned status may be OK, illegal event number, illegal name, or illegal time.

B.1.3 Wait for a Set of Events to Occur

Wait allows a process to suspend itself until a set of events occurs. The calling process is suspended until all events specified have reported (see POST B.1.4).

Calling Sequence:

WAIT (SETOFEVENTS)

Parameters:

SETOFEVENTS: SET OF 1..16

B.1.4 Post a Result

Post allows a process to send a status report and data back to its parent process. At creation (i.e., CREATE PROCESS B.1.1 or SCHEDULE B.1.2) the parent specifies an event number which is stored in the child's PCB. When POST is called the status and data is stored in the parent's PCB with this event number. If this event completes a set of events waited for, the parent will be returned to the ready list.

Calling Sequence:

POST (Reportdata)

Parameters:

REPORTDATA: ARRAY (1.3) where the first word is the status code and the last two words are data.

B.1.5 Kill a Process

Kill allows a process to kill itself or any process which it has created by a CREATE PROCESS or SCHEDULE. All decendents of the process are killed and the PCB and any resources allocated are returned to the system.

Calling Sequence:

KILL (PROCESSID, STATUS)

Parameters:

PROCESSID: INTEGER this is the PROCESSID which was returned by the CREATE PROCESS or SCHEDULE call.

STATUS: STATUSTYPE a variable parameter whose value at the return will be the status of the call. Possible status returns include OK and illegal process id.

B.1.6 Pause

Pause allows a process to temporarily suspend itself by placing itself at the end of the ready queue. Pause should be called during long execution sequences which may monopolize the processor.

Calling Sequence:

PAUSE

B.1.7 Delay

Delay allows a process to suspend iteself for a fixed amount of time.

Calling Sequence:

DELAY(TIME)

Parameters:

TIME: TIMERECORD amount of time task is to be delayed.

B.1.8 Suspend

The suspend function allows a process to deactivate itself without killing itself. A suspended process will not continue execution until an activiate (B.1.9) is issued. When an activate is issued, the process will continue execution with the instructions which immediately follow the call to SUSPEND.

Calling Sequence:

SUSPEND

B.1.9 Activate

The activate function allows a process to activate another process which has been suspended. If the specified process is suspended, the suspension flag will be reset and the specified process placed on the ready queues. If the specified process is not suspended, the activate function will be ignored.

Calling Sequence:

ACTIVATE (PROCESSID)

Parameter:

PROCESSID: INTEGER the process id which was returned by the CREATE PROCESS, SCHEDULE or GET ID call.

B.1.10 Get Id

To GETID call returns the process id of the calling process.

Calling Sequence:

GET ID (PROCESSID)

Parameter:

PROCESSID: INTEGER a return variable which will contain the id of the calling process.

B.2 INTERPROCESS COMMUNICATION

B.2.1 Send Message

Send message allows one process to communicate with another. The sender puts the message into a buffer and reserves space for an answer. Send message puts the sender's name on the receiver's in list and wakes up the receiver if it is waiting for a message. The sending process may continue as soon as it receives the message identification number.

Calling Sequence:

SENDMESSAGE (RECEIVER, MESSAGEBUF, ANSBUF, ID)

Parameters:

RECEIVER: IDENTIFIER the logical name of the receiving process. (The actual location of the receiving process is transparent to the sender.)

MESSAGEBUF:BUFFER the buffer containing the message.

ANSBUF:BUFFER the buffer reserved for the answer.

ID: MESSAGEID a variable which at the return contains the message identification number for later reference.

B.2.2 Wait Message

Wait message suspends the calling process until a message has been received.

Calling Sequence:

WAITMESSAGE

B.2.3 Transfer Message

Transfer message physically moves the data if required and allows the receiver to get the address of the message and answer buffers. If the sender and the receiver are in the same processor this will be the address of the buffer specified by the sender in the send message call. Otherwise the address will be the address of the buffer where the message has been copied.

Calling Sequence:

TRANSFERMESSAGE (SENDER, ID, PTR1, PTR2)

Parameters:

SENDER: IDENTIFIER this is the logical name of the process which was in receiver's IN list.

ID: MESSAGE ID this is the message identification number.

PTR1: BUFFER this variable will contain the address of the message buffer.

PTR2: BUFFER this variable will contain the address of the answer buffer.

B.2.4 Send Answer

After the receiver of a message has called TRANSFER MESSAGE (B.2.3) and filled the answer buffer, SENDANSWER copies the answer into the original answer buffer if the sender and receiver are in different processors. The original message is marked as answered. If the process receiving the answer is waiting for an answer it is returned to the ready queue.

Calling Sequence:

SENDANSWER (SENDER, ID, PTR2)

Parameters:

SENDER: IDENTIFIER this is the logical name of the process which sent the message.

ID: MESSAGEID this is the message identification number.

PTR2: : BUFFER this is the address of the buffer containing the answer.

B.2.5 Wait Answer

Wait answer suspends the calling process until any one of its messages have been answered. An answer may be from a process which received a message or a system answer may be received if the process to whom the message was sent has been removed or fails to respond.

Calling Sequence:

WAITANSWER

B.3 FILE MANAGEMENT

B.3.1 Create a File

Create adds a new file to the directory and allocates mass storage space to the file. File characteristics are specified by the parameters.

Calling Sequence:

CREATE (EVENTNUMBER, FILENAME, FILETYPE, FILESIZE, STATUS)

Parameters:

EVENTNUMBER: 1..16 used by the file manager to signal the calling process when the create is completed.

FILENAME: IDENTIFIER the name of file being created.

FILETYPE: FILERECORD this is a record with variable fields containing a file type and a set of attributes for that file type.

FILESIZE: INTEGER size of a file record.

STATUS: STATUSTYPE a variable which will contain the status of the call at return. Possible statuses which may be returned include OK, file name already in directory, illegal name, insufficient space, or illegal type.

B.3.2 Delete a File

Delete removes file from directory and releases the space allocated to the file.

Calling Sequence:

DELETE (EVENTNUMBER, FILENAME, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when the delete has been completed.

FILENAME: IDENTIFIER the current name of the FILE.

STATUS: STATUSTYPE a variable which contains the status at the return which may be OK, file is open, file does not exist, or file is permanent.

B.3.3 Rename a File

Rename changes the name of a file in the directory without altering any of the other attributes.

Calling Sequence:

RENAME (EVENTNUMBER, OLDNAME, NEWNAME, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal the calling process when the rename has been completed.

OLDNAME: IDENTIFIER the current name of the file.

NEWNAME: IDENTIFIER the new file name.

STATUS: STATUSTYPE a variable which contains the status when the call is completed. Possible status returns included OK, file open, file does not exist, and illegal file name.

B.3.4 Open a File

Open activates a created file or an I/O device for use and associates the file or device with a channel. It also specifies whether the file or device is to be read only, write only, read/write and the type of buffering to be used, if any.

Calling Sequence:

OPEN (EVENTNUMBER, FILENAME, MODE, CHANNEL, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal the calling process when the OPEN has been completed.

FILENAME: IDENTIFIER the current name of the file or the identifer for the I/O device.

MODE: MODERECORD a record indicating the mode including write only, read/write and the type of buffering to be used, if any.

CHANNEL: INTEGER a variable which contains the number associated with the logical path between the user and the file or device.

STATUS: STATUSTYPE a variable which will contain the status of the OPEN call at return. Status returned may be OK, illegal file name, illegal mode or channel not available.

B.3.5 Close

Close removes a file or device from active use by closing the channel.

Calling Sequence:

CLOSE (EVENTNUMBER, CHANNEL, MODE, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal the calling process when the close has been completed.

CHANNEL: INTEGER this is the channel number which as assigned by the previous OPEN.

MODE: MODERECORD this allows channel to be closed for reading or writing only.

STATUS: STATUSTYPE this variable contains the status of the call at the return. Possible status returns include OK, illegal channel number, illegal mode, and channel not open.

B.4 INPUT/OUTPUT OPERATIONS

B.4.1 Read

Read performs a random access input from a segmented or contiguous file.

Calling Sequence:

READ (EVENTNUMBER, CHANNEL, BUFFER, NUMWORDS, FILEADDR, STATUS).

Parameters:

EVENTNUMBER: 1..16 used to signal the completion of READ.

CHANNEL: INTEGER the channel previously specified in OPEN.

BUFFER: IOBUFFER the buffer where the input will be stored.

NUMWORDS: INTEGER the number of words to be transferred.

FILEADDR: INTEGER location in file from which input is to be read.

STATUS: STATUSTYPE a variable which will have the status when the call is complete. Possible status returns include OK, channel closed, illegal channel number, illegal file name, file read protected, parity, or other read error.

B.4.2 Write

Write performs a random access output to a segmented or contiguous file.

Calling Sequence:

WRITE (EVENTNUMBER, CHANNEL, BUFFER, NUMWORDS, FILEADDR, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal the completion of the WRITE operation.

CHANNEL: INTEGER the channel previously specified in the OPEN command.

BUFFER: IOBUFFER the buffer containing the data to output.

NUMWORDS: INTEGER the number of words in the buffer.

FILEADDR: INTEGER location in the file to which output is to be written.

STATUS: STATUSTYPE a variable which will have the status when the call is complete. Possible status returns include OK, channel closed, illegal channel number, illegal file name, file write protected, parity or other write error.

B.4.3 Read Sequential

READSQ will transfer the specified number of words from a file or device to the user buffer. Files which are accessed sequentially will read data from an address based on a file pointer stored with the channel which is cleared on OPEN.

Calling Sequence:

READSQ (EVENTNUMBER, CHANNEL, BUFFER, NUMWORDS, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when input is compete.

CHANNEL: INTEGER number of a channel which has been opened for input.

BUFFER: IOBUFFER address of user buffer to receive input.

NUMWORDS: INTEGER number of words to be read.

STATUS: STATUSTYPE a variable which will contain the status when the call is complete. Possible status returns include OK, channel closed, illegal channel number, read error, and file or device read protected.

B.4.4 Write Sequential

WRITESQ will transfer the specified number of words from the user buffer to the file or device.

Calling Sequence:

WRITESQ (EVENTNUMBER, CHANNEL, BUFFER, NUMWORDS, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when write is complete.

CHANNEL: INTEGER number of channel opened for output.

BUFFER: IOBUFFER address of user buffer to be output.

NUMWORDS: INTEGER number of words to be output.

STATUS: STATUSTYPE a variable containing the status of the call when complete. Possible status returns include OK, channel closed, illegal channel number, parity error, and file or device write protected.

B.4.5 Read Line

READLINE reads on line into the user specified buffer. A line will include a sequence of characters through an end-of-line character.

Calling Sequence:

READLINE (EVENTNUMBER, CHANNEL, BUFFER, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when the read is complete.

CHANNEL: INTEGER number of channel opened for input.

BUFFER: LINEBUFFER user buffer to receive input.

STATUS: STATUSTYPE a variable which will contain the status when the read is complete. Possible status returns include OK, channel closed, illegal channel number, parity error, and file or device read protected.

B.4.6 Write Line

WRITELINE outputs one line from user specified buffer.

Calling Sequence:

WRITELINE (EVENTNUMBER, CHANNEL, BUFFER, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when the write is complete.

CHANNEL: INTEGER number of a channel which has been opened for output.

BUFFER: LINEBUFFER address of the buffer containing the line to be written.

STATUS: STATUSTYPE a variable which will contain the status when the write is complete. Possible status returns include OK, channel closed, illegal channel number, parity error, and file or device write protected.

B.4.7 Read Record

READRECORD reads on record from a file or device.

Calling Sequence:

READRECORD (EVENTNUMBER, CHANNEL, BUFFER, NUMWORDS, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when read is complete.

CHANNEL: INTEGER the number of a channel which has been opened for input.

BUFFER: IOBUFFER address of user buffer to receive input.

NUMWORDS: INTEGER maximum number of words to transfer.

STATUS: STATUSTYPE a variable which will contain the status when the call is complete. Possible status returns include OK, illegal channel number, channel closed, and read protected.

B.4.8 Write Record

WRITERECORD outputs one record to a file or device.

Calling Sequence:

WRITERECORD (EVENTNUMBER, CHANNEL, BUFFER, NUMWORDS, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal the calling process that the write is complete.

CHANNEL: INTEGER the number of a channel opened for output.

BUFFER: IOBUFFER address of record to be output.

NUMWORDS: INTEGER number of words to be output.

STATUS: STATUSTYPE a variable which will contain the status when the call is complete. Possible status returns include OK, channel closed, illegal channel number, parity error, and write protected.

B.4.9 Read Character

READCHAR inputs a single character.

Calling Sequence:

READCHAR (EVENTNUMBER, CHANNEL, A)

Parameters:

EVENTNUMBER: 1..16 used to signal calling process when read to complete.

CHANNEL: INTEGER the number of a channel opened for input.

A: CHAR variable which will contain the character to be read.

B.4.10 Write Character

WRITECHAR outputs one character.

Calling Sequence:

WRITECHAR (CHANNEL, A)

Parameters:

CHANNEL: INTEGER the number of a channel opened for output.

A: CHAR character to be output.

B.5 ERROR REPORTING

B.5.1 Received Bad Data

If a process receives an inconsistent or meaningless data it must report this to the Main Error Control Center. Bad data may be detected in either a message or an answer.

Calling Sequence:

BADDATA (PROCESSNAME, DATASENDER)

Parameters:

PROCESSNAME: IDENTIFIER the logical name of the process recognizing the bad data.

DATASENDER: IDENTIFIER the logical name of the process which sent the data.

B.5.2 Missing Process

If a process is unable to contact another process it must report this to the Main Error Reporting Center.

Calling Sequence:

MISSING (PROCESSNAME)

Parameters:

PROCESSNAME: IDENTIFIER the logical name of the process which cannot be contacted.

B.6 DEBUGGING FUNCTIONS

B.6.1 Halt

HALT halts the running program and produces a dump.

Calling Sequence:

```
HALT
```

B.6.2 Trace

TRACE keeps a record of the names of processes executed. The list is output at the next ENDTRACE call. Names will be stored in a circular list so that the last "n" names will be available.

Calling Sequence:

```
TRACE
```

B.6.3 End Trace

ENDTRACE outputs the list of the last "n" processes executed since the previous TRACE call.

Calling Sequence:

```
ENDTRACE
```

B.7 OVERLAY CONTROL FUNCTIONS

B.7.1 Load Overlay

LOAD OVERLAY reserves an overlay area and loads the requested program when the area is available.

Calling Sequence:

LOADOVERLAY (EVENTNUMBER, NAME, STATUS)

Parameters:

EVENTNUMBER: 1..16 used to signal the calling process when the requested overlay has been loaded.

NAME: IDENTIFIER the name of the overlay to be loaded.

STATUS: STATUSTYPE a variable which will contain the status when the call is complete. Possible status returns include OK, illegal overlay name, and parity or other read error.

B.7.2 Release Overlay

RSLEOVERLAY releases an overlay area. If a LOADOVERLAY is pending, the next program will be loaded.

Calling Sequence:

RLSEOVERLAY (NAME)

Parameters:

NAME: IDENTIFIER the name of the overlay program to be released.

B.7.3 Next Overlay

NEXTOVERLAY releases the current overlay and loads the specified overlay in the same area. The specified overlay is loaded even if there are outstanding requests for the overlay area. This allows one overlay to chain to the next overlay of a large program.

Calling Sequence:

NEXTOVERLAY (CURRENTNAME, NEXTNAME)

Parameters:

CURRENTNAME: IDENTIFIER name of the overlay program to be released.

NEXTNAME: IDENTIFIER name of next overlay to be loaded.

B.8 USER DEFINED SEMAPHORES

The following functions allow users to define and use counting semaphores.

B.8.1 Set Up Use Control Block

SETUPUCB defines and intializes a Use Control Block. The Use Control Block consists of a counter and a wait queue.

Calling Sequence:

SETUPUCB (COUNT, UCBID)

Parameters:

COUNT: INTEGER maximum number of units available.

UDBID: INTEGER a variable which will contain the Use Control Block identification for use in RESERVE and RELEASE calls.

B.8.2 Reserve

RESERVE decrements the count in the Use Control Block. If the count is greater than or equal to zero the calling process is signalled immediately. Otherwise the calling process is added to the wait queue in the Use Control Block.

Calling Sequence:

RESERVE (EVENTNUMBER, UCBID)

Parameters:

EVENTNUMBER: 1..16 used to signal the calling process when a unit is available.

UCBID: INTEGER the identification number returned by the SETUPUCB.

B.8.3 Release

RELEASE increments the count in the Use Control Block. If the count is negative or zero the first request in the wait queue is signalled.

Calling Sequence:

RELEASE (UCBID)

Parameters:

UCBID: INTEGER the identification number returned by the previous SETUPUCB.

B.8.4 Use Count

USECOUNT returns the value of COUNT in the Use Control Block. This value is the number of units available or the negative of the number of tasks on the wait queue.

Calling Sequence:

X: = USECOUNT (UCBID)

Parameters:

UCBID: INTEGER the use count block identification number.

Variables:

X: INTEGER X will contain the value of count.

B.9 TIME AND DATE FUNCTIONS

B.9.1 Time

TIME returns the system time to the users location.

Calling Sequence:

X: = TIME

Variables:

X: TIMETYPE X is a variable which will contain the hours, minutes, and seconds at the function return.

B.9.2 Date

DATE returns the system date to the user specified location.

Calling Sequence:

X: = DATE

Variables:

X: DATETYPE X is a variable which will contain the day, month and year at the function return.

B.9.3 Set Time

SETTIME allows the user to set the system clock.

Calling Sequence:

SETTIME (TIME, STATUS)

Parameters:

TIME: TIMETYPE the time supplied by the user.

STATUS: STATUSTYPE a variable which will contain the status of the call which may be OK or illegal time.

B.9.4 Set Date

SETDATE allows the user to set the system date.

Calling Sequence:

SETDATE (DATE, STATUS)

Parameters:

DATE: DATETYPE the date supplied by the user.

STATUS: STATUSTYPE a variable which will contain the status of the call which may be OK or illegal date.

APPENDIX C
INTERPROCESS COMMUNICATION SYSTEM

The following is an overview of the interprocess communication scheme in the VTS/OS. The architecture chosen in Phase I consists of a group of processors connected by a shared bus. Each processor is closely coupled to a unit of memory. A process within a processor can talk to any other process, whether they are both in the same unit of memory or not. Communication will normally take place via explicit data exchanges (messages and replies) rather than through shared memory even if both processes are in the same processor; that is, a process must ask another process for data rather than reading it from a common table. The communication software is arranged in layers (Figure C-1). At each level there is a protocol, a system of rules, to control the flow of data and to detect and correct errors.

The communication layers are invisible to a process. The process addresses any other process directly by its logical name. Neither the sender nor receiver know the other's location. This location transparency makes it possible to dynamically reconfigure the system if a processor fails.

The router is responsible for routing the messages and answers for all processes within its processor. The router is aware of the location of all processes in the system and adds the destination information, processor address and buffer code to all messages and answers it sends to the dispatcher.

The dispatcher and the bus interface handle the mechanical details of the transmissions. The router indicates where the information to be sent is located and the destination. The dispatcher and bus interface break the message into packets, add a header, and send each packet to the bus.

We will now examine a typical conversation between two processes and the operations performed by the operating system at each level.

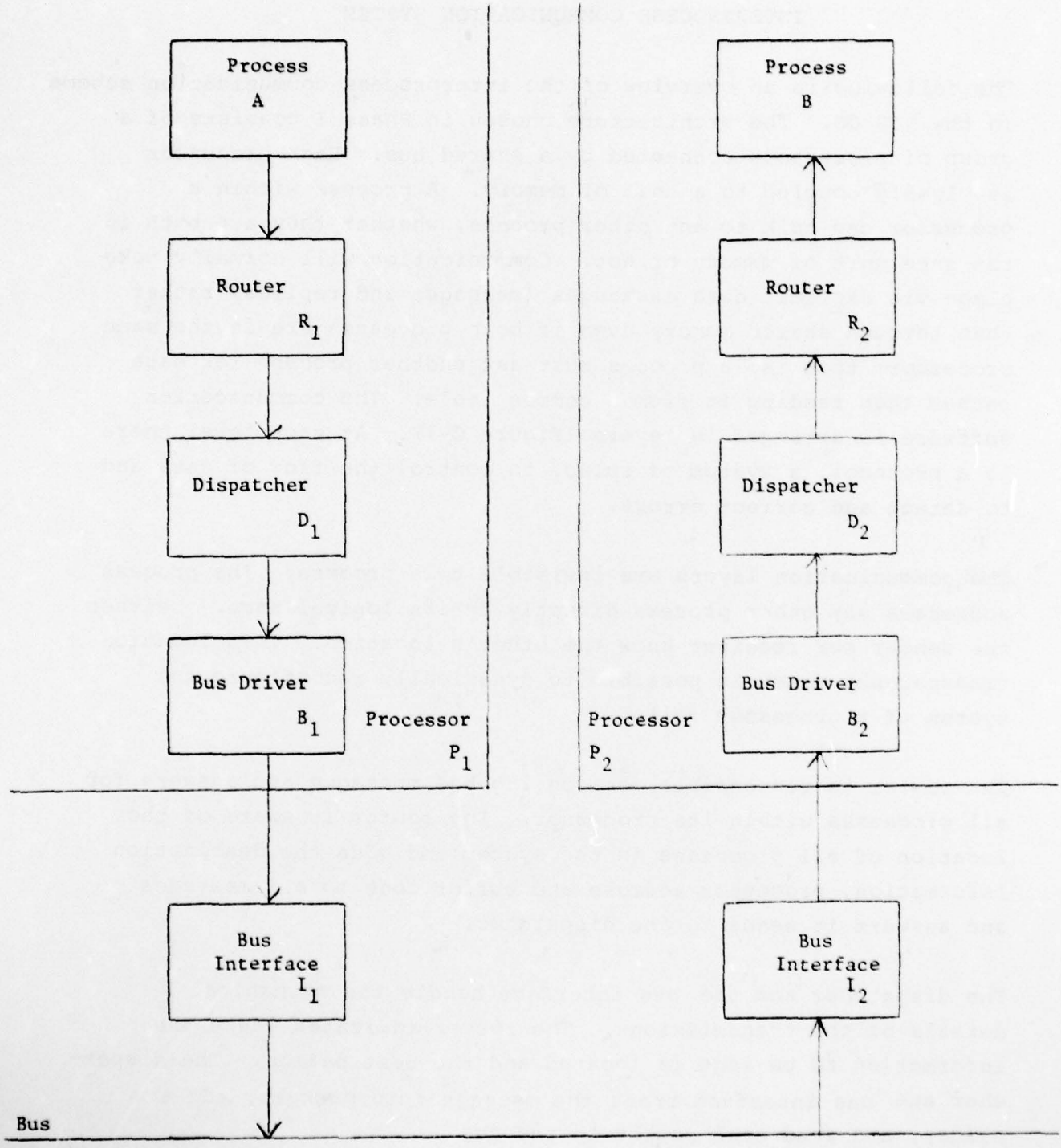

Process
A
Router
R_1
Dispatcher
D_1
Bus Driver
B_1
Processor
P_1
Bus
Interface
I_1
Process
B
Router
R_2
Dispatcher
D_2
Bus Driver
B_2
Processor
P_2
Bus
Interface
I_2
Bus

Figure C-1

A Conversational Scenario

Process A and process B are two cooperating but independent processes. A sends a message to B and continues without waiting for a reply from B. When A needs the answer from B, it suspends itself until the answer arrives. Initially process B is idle, waiting for a message from A. When B receives the message, B prepares the answer and sends it to A. After sending the answer, B is free to begin with another task. The actions of both processes are summarized in Figure C-2.

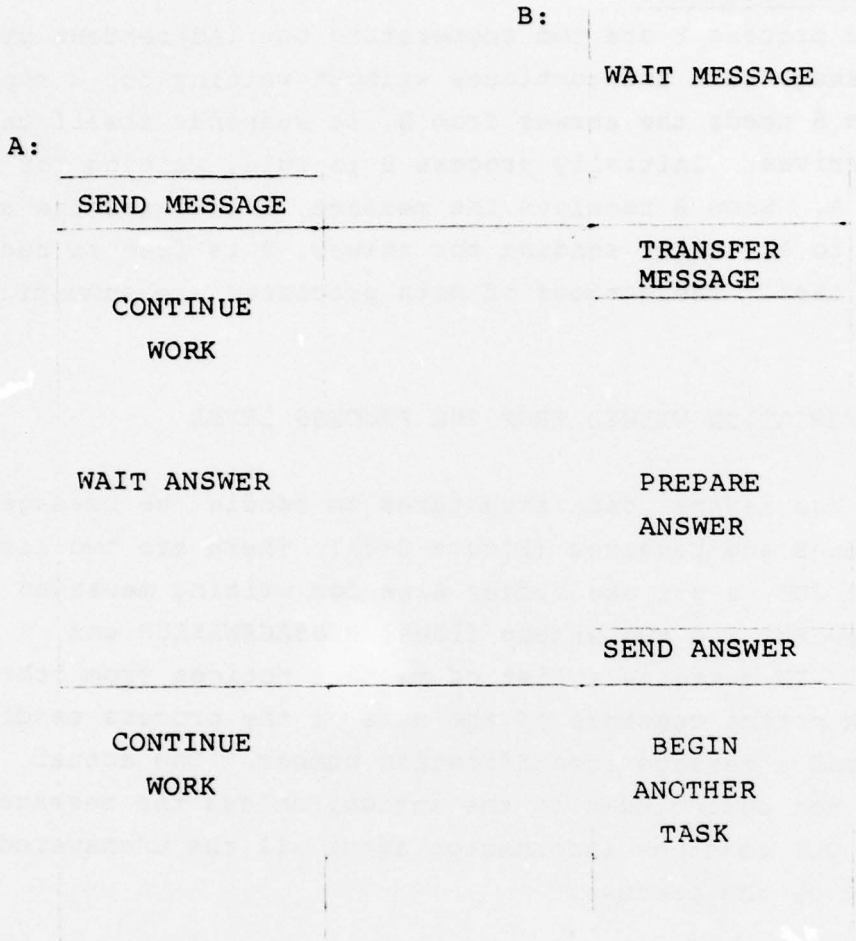
C.1 THE CONVERSATION VIEWED FROM THE PROCESS LEVEL

Each process has several data structures to handle the messages and answers it sends and receives (Figure C-3). There are two lists, called IN and OUT, a private buffer area for writing messages and receiving answers, and two wakeup flags, MESSAGEWAKEUP and ANSWERWAKEUP. IN contains a list of message notices from other processes. A notice consists of the name of the process sending the message and a message identification number. The actual message will not be included in the notice, unless the message is very short. OUT contains information about all the unanswered messages sent by the process.

We will now examine the five support functions used by A and B during their conversation: SENDMESSAGE, WAITANSWER, WAITMESSAGE, TRANSFERMESSAGE, and SENDANSWER. Where parameters passed to these support functions are pointers, the symbol "^" will precede the name of the data area indicated by the pointer.

SENDMESSAGE

When A has a message to send B, A writes the message in one of its own buffers, MESS, and reserves enough space for the answer in another one of its buffers, ANS. After A calls SENDMESSAGE (B, ^MESS, ^ANS), a message identification, ID, is returned to A, and A is free to continue. B's name, the message identification, the status of the message and pointers to MESS and ANS are placed on A's OUT list (Figure C-3). A notice with A's name and the message identification is added to B's IN list. The message will not be sent to B at this time unless the message is very short. B is informed




```
A:  SEND MESSAGE (B, ↑MESS, ↑ANS)   B:  WAIT MESSAGE
      .                                    .
      .                                    .
      .                                    .
    WAIT ANSWER                          TRANSFER MESSAGE (A, ID)
      .                                    .
      .                                    .
      .                                    .
                                         SEND ANSWER (A, ID,
                                           .
                                           .
                                           .
```

Figure C-2

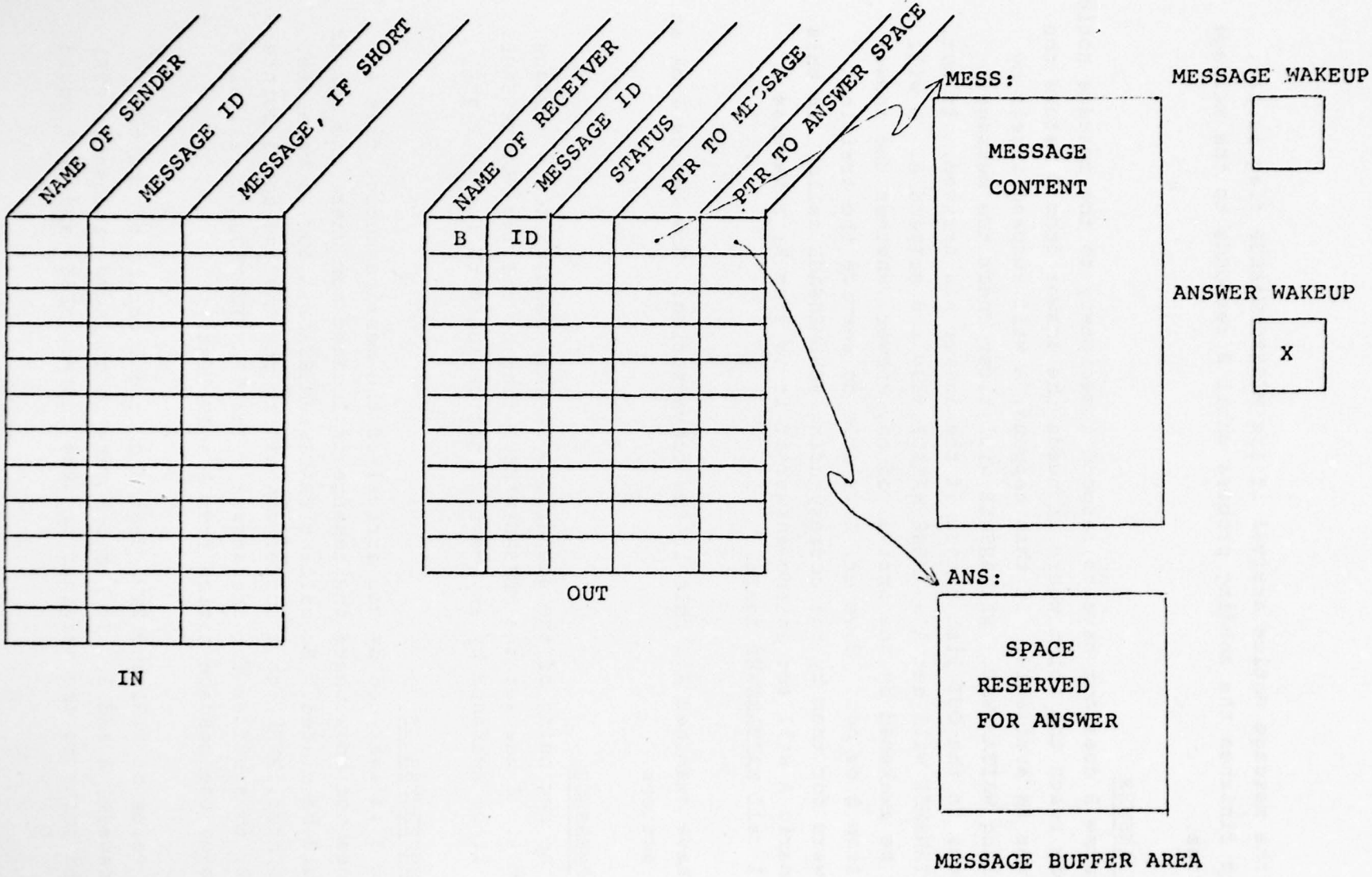

NAME OF SENDER
MESSAGE ID
MESSAGE, IF SHORT
IN
NAME OF RECEIVER
MESSAGE ID
STATUS
PTR TO MESSAGE
PTR TO ANSWER SPACE
B
ID
OUT
MESS:
MESSAGE
CONTENT
MESSAGE WAKEUP
ANSWER WAKEUP
X
ANS:
SPACE
RESERVED
FOR ANSWER
MESSAGE BUFFER AREA

Figure C-3

of the message notice arrival if its MESSAGEWAKEUP flag is set. This finishes the sending process until B responds to the message notice.

WAITANSWER

Because B does not have to respond immediately to the message notice, A may reach the point where it needs the answer from B before the answer is available. If this happens, A will suspend itself by calling WAITANSWER. WAITANSWER will first check the message status in the OUT list to see if the answer has arrived. If not, WAITANSWER will set A's ANSWERWAKEUP flag and suspend A. A will now be awakened by the arrival of any answer, whether the answer is from B or not. However, A is able to service the arriving answers for them by the corresponding SENDMESSAGE calls. In this scenario A will act on the answer if it is from B; otherwise A will call WAITANSWER again.

We have examined A's role in the conversation. Now let us look at B's actions.

WAITMESSAGE

At the beginning of the scenario B is idle waiting for a message from A. B has set the MESSAGEWAKEUP flag by calling WAITMESSAGE. B will be awakened by any message notice added to its IN list.

TRANSFERMESSAGE

When B is awakened by the arrival of the message notice from A, B does not know where the message is located nor where the answer should be placed. By calling TRANSFERMESSAGE(A,ID), B receives a pointer, PTR1, to a buffer containing the message and a pointer, PTR2, to a buffer for the answer. The TRANSFERMESSAGE call also removes the message notice from B's IN list.

The value of PTR1 and PTR2 depend on the location of the two processes, A and B. If A and B were in the same processor, PTR1 would point to the original message buffer, MESS, and PTR2 would

point to the reserved answer buffer, ANS (Figure C-4). In this scenario A and B are in different processors, so PTR1 points to a buffer with a copy of the message and PTR2 points to a buffer of the same size as ANS. Both buffers are in B's processor, P2 (Figure C-5).

SENDANSWER

After B writes the answer in the buffer pointed to by PTR2, B calls SENDANSWER (A, ID, PTR2). B is now finished with the conversation. The answer will be copied into process A's buffer, ANS, if necessary and A will be awakened if its ANSWERWAKEUP flag is set.

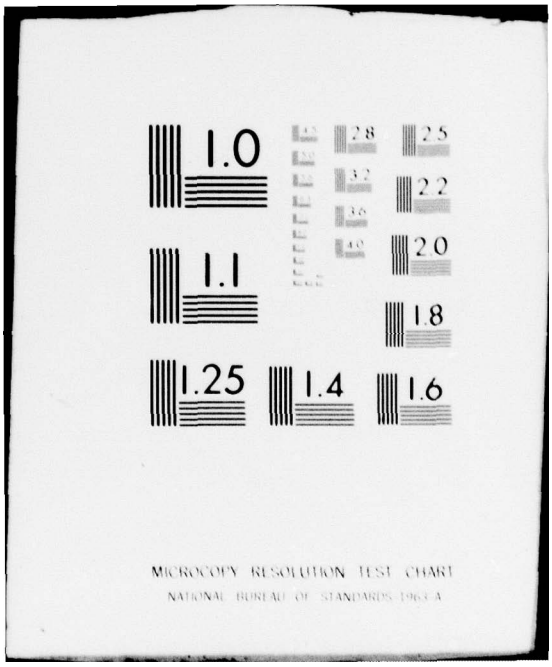
C.2 THE CONVERSATION VIEWED FROM THE ROUTER LEVEL

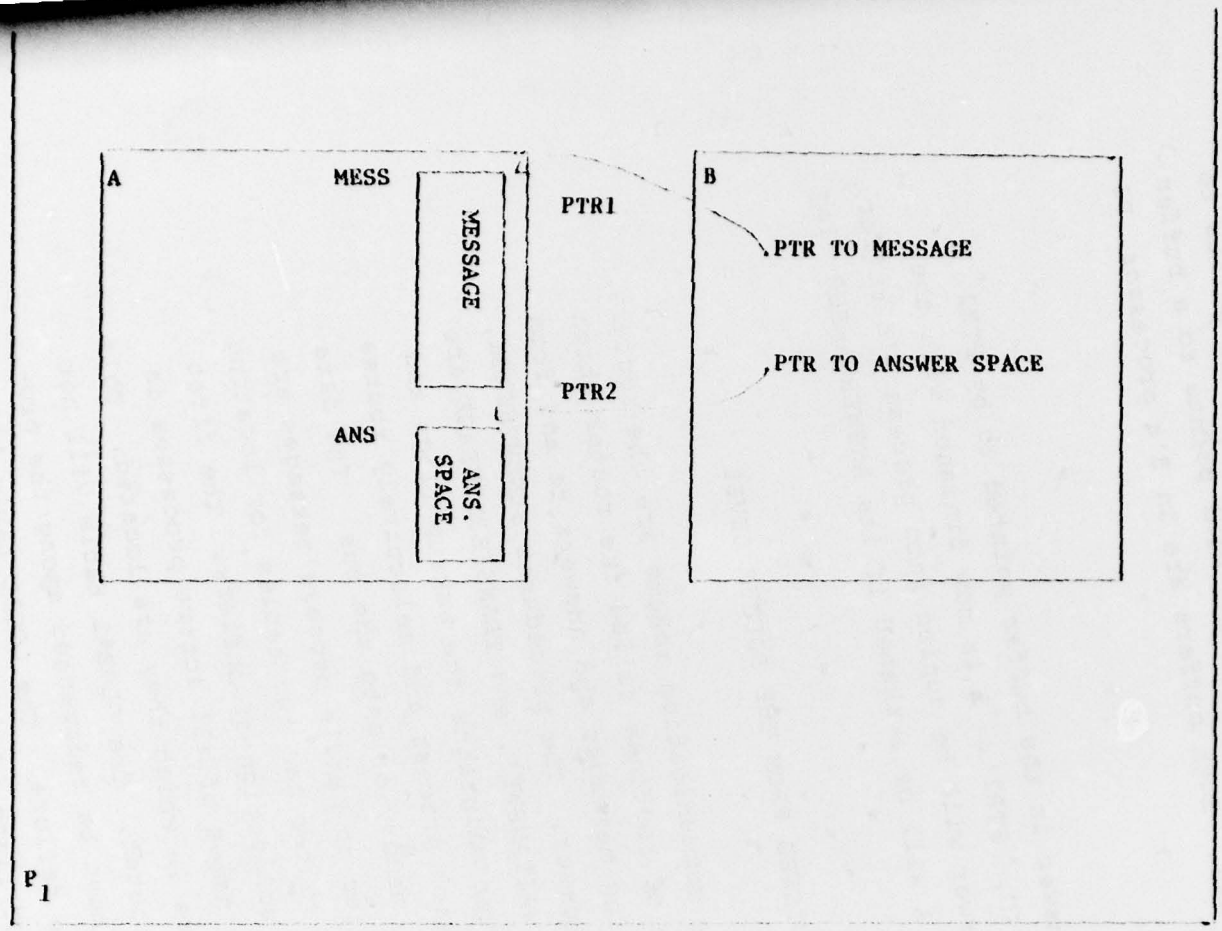
At the next level of the communication scheme are the routers. Each process has a group of routines called the router which are responsible for routing messages and answers to and from the processes in its processor. The procedures SENDMESSAGE, WAITMESSAGE, SENDANSWER, WAITANSWER, and TRANSFERMESSAGE are router routines. The router maintains the wakeup flags and the IN and OUT lists for each process and selectively routes the communication to local memory or onto the bus. The data structures used by the router to handle process messages are shown in Figure C-6. Each router has two tables for locating processes, a mailbox and a collection of buffers. The first table, GLOBAL, contains the names of all active processes in the system and the processors in which they are located. This table is the same for each router. The GLOBAL table will not change unless the processes must be rearranged among the processors because of a hardware failure. The LOCAL table lists the location of the IN and OUT lists of each process in the processor. The MAILBOX collects short instructions from other routers. The system buffers are used to hold temporary copies of process messages and answers.

AD-A074 053 INTERNATIONAL COMPUTING CO BETHESDA MD F/G 9/2
VTS PROCESSING DISPLAY SUBSYSTEM DESIGN.(U)
JAN 79 C C HENSON, F T MICKEY, R S GRAHAM DOT-CG-81-78-1833
UNCLASSIFIED USCG-D-54-79 NL

4 OF 4
AD A074053

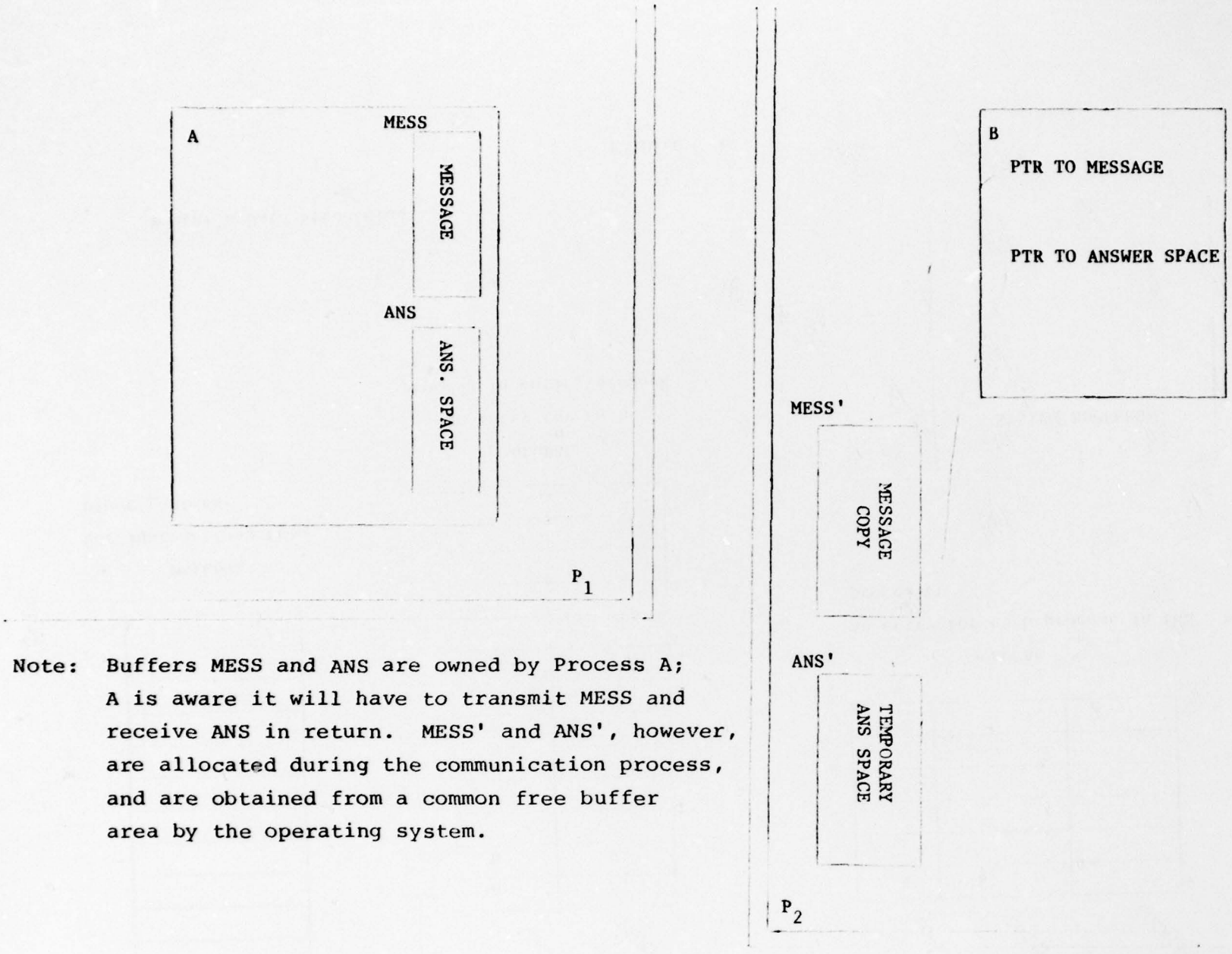
END
DATE FILMED
10-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A



Case where Process A, Process B in the same processor

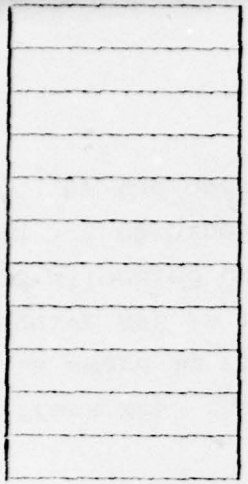
Figure C-4



Note: Buffers MESS and ANS are owned by Process A; A is aware it will have to transmit MESS and receive ANS in return. MESS' and ANS', however, are allocated during the communication process, and are obtained from a common free buffer area by the operating system.

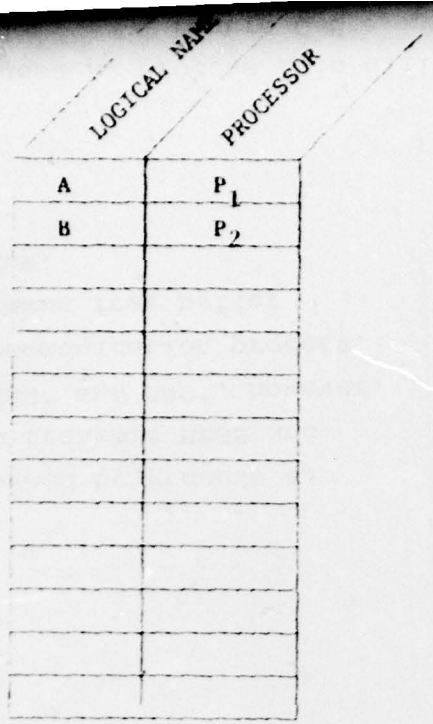
Case where Process A, Process B in different Processors.

Figure C-5



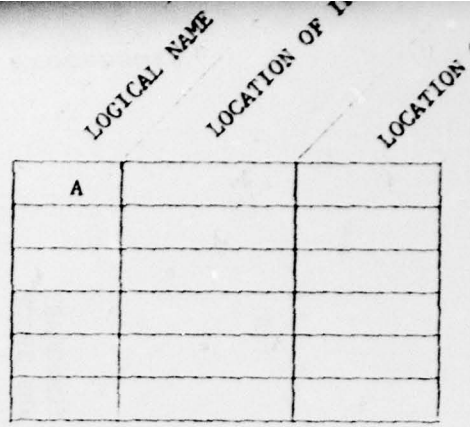
MAILBOX

For instructions from other routers



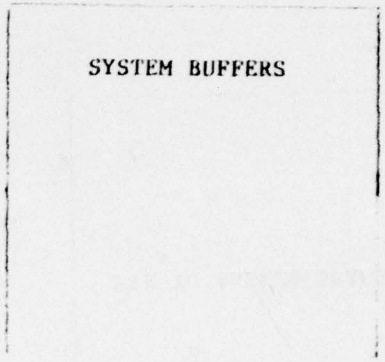
GLOBAL

An entry for each process in entire network



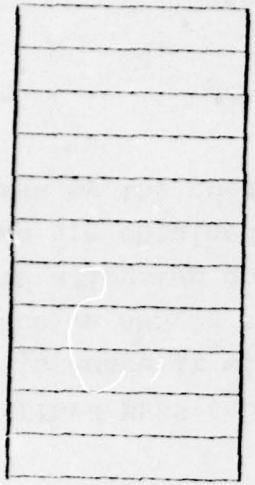
LOCAL

An entry for each process in the processor



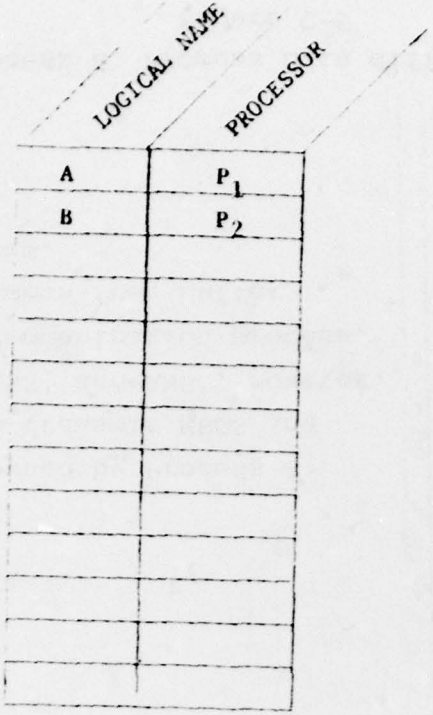
Router's Data Structures

Figure C-6



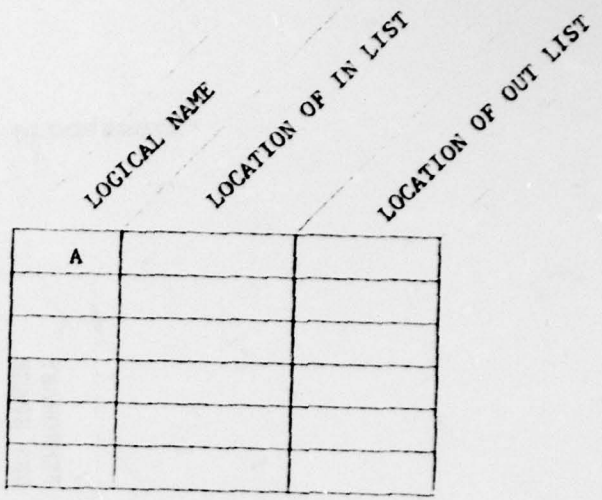
MAILBOX

For instructions from other routers



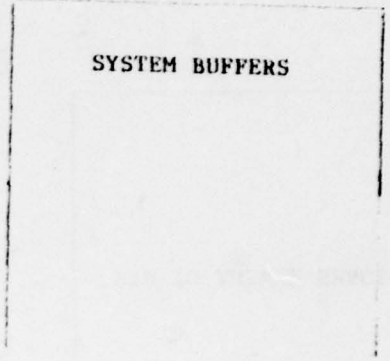
GLOBAL

An entry for each process in entire network



LOCAL

An entry for each process in the processor



Router's Data Structures

Figure C-6

We will now consider the routers' roles in the conversational scenario. R_1 is the router in A's processor. R_2 is the router in B's processor.

SENDMESSAGE

Both R_1 and R_2 perform several jobs when A calls SENDMESSAGE (B, MESS, ANS). Buffers MESS and ANS are owned by Process A, and may exist in data areas local to A. R_1 computes the message identification ID and fills in A's OUT list with B, ID, MESS, ANS and the message status. R_1 then uses GLOBAL to determine where to send the message notice. If B were in P_1, then R_1 would put the message notice on B's IN list. (The location of B's IN list would be in the LOCAL table.) In this scenario an instruction of the form:

FILL IN* B A, ID message, if short

is sent on the bus to R_2's MAILBOX.

When R_2 receives the FILL IN instruction it adds the message notice to B's IN list. If B's MESSAGEWAKEUP signal is set, R_2 will awaken B by changing B's status from suspended to active.

WAITANSWER

R_1 handles the WAITANSWER call from A. R_1 sets A's ANSWERWAKEUP signal and sends a message to the processor scheduler to suspend A. The bus is not utilized in a WAITANSWER call.

WAITMESSAGE

R_2 handles the WAITMESSAGE call from B. R_2 sets B's MESSAGEWAKEUP signal and sends a message to the processor scheduler to suspend B. The bus is not utilized in a WAITMESSAGE call.

*FILL IN is a mnemonic representing the numeric instruction type for this message.

TRANSFERMESSAGE

Process B calls TRANSFERMESSAGE (A, ID) to find the location (PTR1) of the message and the location (PTR2) of the answer buffer. This call requires R_1 and R_2 to perform several tasks. R_2 removes the message notice from B's IN list and finds A's location in the GLOBAL table. If A were in P_2, then R_2 would set PTR1 to point to MESS and PTR2 to point to ANS (Figure C-4). In this case R_2 would be able to find the location of MESS and ANS from A's OUT list.

The TRANSFERMESSAGE call is more complicated in the scenario where A is in P_1 and B is in P_2. If a copy of the message from A is in B's IN list, R_2 sets PTR1 to be equal to this location. R_2 then reserves a system buffer for the answer. PTR2 is set equal to this buffer location. However, if the message is long and therefore has not yet been transferred to P_2, R_2 reserves two buffers, MESS' for the message and ANS' for the answer, from a common buffer pool. Then R_2 sends the following instruction to R_1's mailbox:

```
Transfer      A, ID M  *
```

R_1 will send the message directly to the message buffer MESS' in P_2. After the entire message is in MESS', R_2 sets PTR1 equal to MESS' and PTR2 to ANS' (Figure C-5).

SENDANSWER

B calls SENDANSWER (A,ID, PTR2) after it has prepared the answer. If A were in the same processor then B's answer would already be in ANS. So R_2 would only need to change the message status in A's OUT list and awaken A if its ANSWERWAKEUP signal were set.

*M is a buffer code. Buffer codes are discussed in Section C.3.

In this scenario, with A and B in different processors, R_2 sends the answer on the bus directly to A's answer buffer, ANS in P_1. This is possible with a bus interface capable of associating specific bus transmissions with reserved buffer areas, as described in Section 6.4.2. After the answer is in ANS, R_1 awakens A and sends R_2 the following instruction:

Kill temporary buffers ID

R_2 then returns MESS' and ANS' to the pool of the system buffers, and the conversation is complete.

C.3 The Conversation Viewed from the Dispatcher and Interface Level

The routers send and receive all interprocessor communication through a dispatcher. Each processor has a dispatcher which, together with the interface processor, prepares all outgoing transmissions, sends outgoing transmissions to the bus and places all incoming transmissions into the proper buffers.

The dispatcher breaks long transmissions into small packets before sending them over the bus. There are several reasons for doing this. The data sent by the routers will vary from intructions of one or two bytes to process messages and answers which may be several thousand bytes long. It would be difficult for the interface processor to manage transmission of so many different sizes. Furthermore, the probability of error is high when long transmissions are sent. Long transmissions would also monopolize the bus, delaying short urgent messages. By breaking the transmission into packets the bus is not allocated for the full duration of a message but only long enough to pass a packet.

Several extra bytes of information are sent on the bus with each packet (Figure C-7). Some of the information is provided by the router when it gives the dispatcher the data to send. The router

TRANSMISSION FORMAT:

SYNC	A_R	C	A_S	COUNT	Buf	P	TRANSMISSION PACKET	CRC

SYNC - Synchronization byte to signal the beginning of a transmission

A_R - Bus address of the receiving processor

C - Control byte to distinguish control tokens from transmission

A_S - Bus address of the sending process

COUNT - Number of bytes in the transmission

Buf - Buffer, code to indicate where transmission should be delivered in the receiving processor

P - Packet sequencing information

CRC - Cyclic Redundancy Check

Transmission Format (Type T3)

Figure C-7

tells the dispatcher which processor is to receive the information (A_R). The router may also know the buffer code (Buf) which indicates where the transmission should be placed by the receiving dispatcher and interface processor. The buffer code is not a pointer; rather, it is an identifying code assigned to allow the receiving bus interface to recognize the transmission as expected and place the data in the reserved buffer for that transmission. The interface processor puts this information on each packet and adds the synchronization bytes, control bytes, sender address, packet length, packet sequencing information and a cyclic redundancy code.

Each interface processor has a Buffer Code Table to match the buffer code (Buf) on an incoming transmission with the buffer reserved for the transmission. If there is no buffer code, the transmission will be placed in the router's mailbox. After all packets are in the proper buffer, the dispatcher notifies the router.

The following examines the bus transmissions and the Buffer Code Table entries which result from the conversation between process A and process B.

SENDMESSAGE

R_1 writes the FILLIN B A, ID instruction in a buffer and asks D_1 to send a transmission from this buffer to processor P_2. The transmission will not have a buffer code associated with it, since it is short enough to fit in a general-purpose instruction buffer. R_1 also adds an entry to Buffer Code Table 1 to correlate ID (the message identification computed by R_1) with the ANS buffer (Figure C-8). This entry will be used during the SENDANSWER call to transfer B's answer directly to A's buffer. The buffer code, address of the ANS buffer, and length of that buffer will be output to the bus interface by D1; the return message with that buffer code will be placed in ANS by a DMA transfer.

D_1 sends the transmission to D_2 when the bus is available. Because the transmission does not include a buffer code, D_2 will place the transmission in R_2's mailbox.

TRANSFERMESSAGE

Before R_2 sends the transfer instruction to R_1, R_2 adds an entry to Buffer Code Table 2 associating a buffer code, M, with the MESS' buffer (Figure C-8). This entry will allow the message to be transferred directly to the MESS' buffer. R_2 then writes the TRANSFER A, ID M instruction in a buffer and asks D_2 to send the instruction to processor P_1. Buffer Code M is transmitted in the data sector of this instruction; it will appear in the buffer code field of the transmission of the message from A to B. R_1 will use M as a buffer code to have the message sent directly to MESS'.

When R_1 receives the transfer instruction, R_1 asks D_1 to send the message from the MESS buffer with buffer code M to processor P_2. If the message is long, D_1 will break it into packets. Each packet will have the buffer code, M, attached to it. D_2 puts each packet into MESS'. When all packets have arrived, D_2 will notify R_2. R_2 then removes the M entry from Buffer Code Table 2.

SENDANSWER

When B's answer is in ANS', R_2 asks D_2 to send the answer from ANS' with buffer code, ID, to processor, P_1. D_2 breaks the answer into packets and adds ID to each packet. D_1 will assemble the packets in ANS. (The entry associating ID with the ANS buffer was put into Buffer Code Table 1 by R_1 during the SENDMESSAGE call.)

After R_1 is notified that the answer is in ANS, R_1 removes the ID entry from Buffer Code Table 1. R_1 then writes a KILL ID instruction in a buffer and asks D_1 to send the instruction to processor P_2. When the instruction is in R_2's mailbox, the bus conversation is finished.

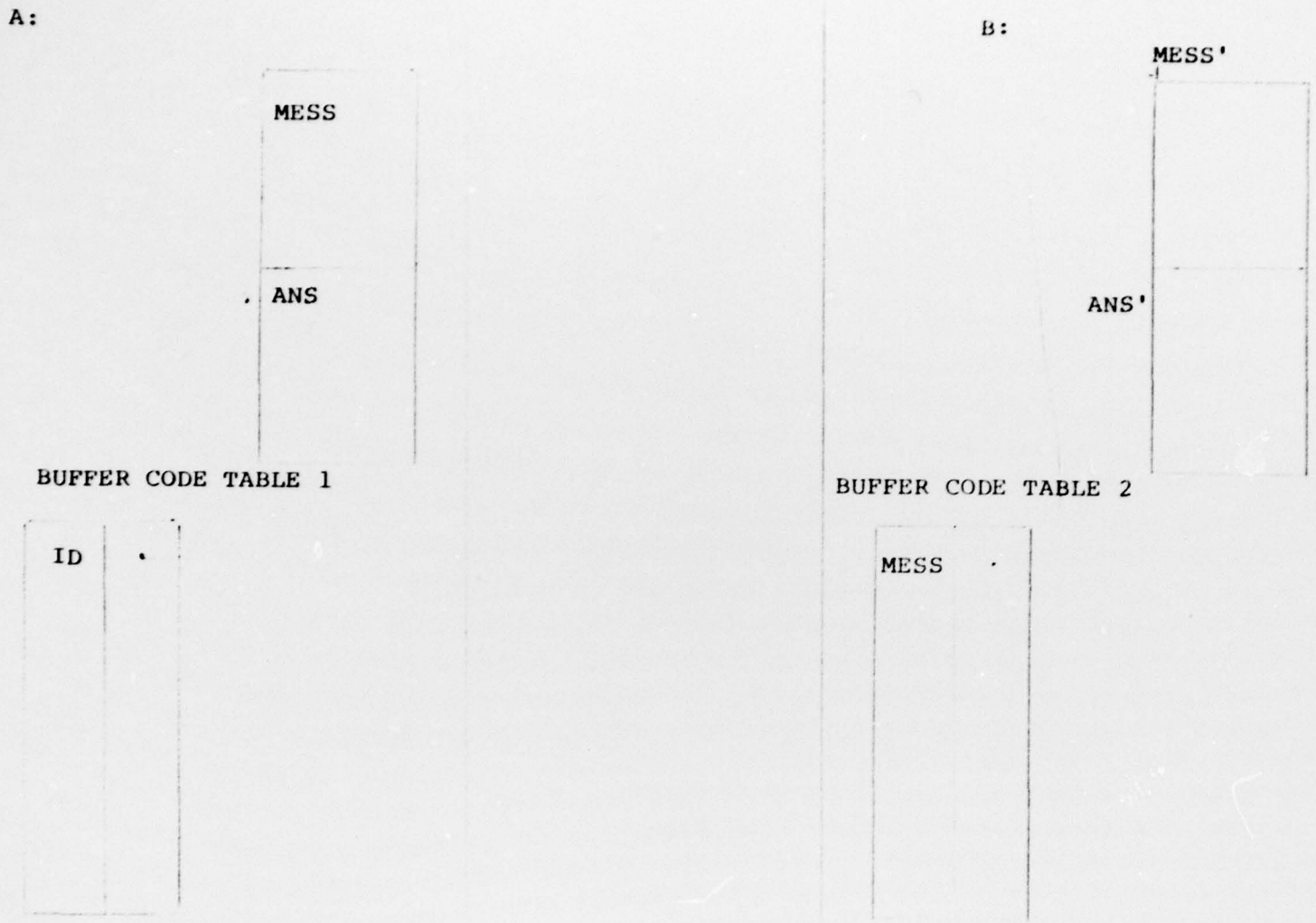

A:
MESS
ANS
BUFFER CODE TABLE 1
ID
B:
MESS'
ANS'
BUFFER CODE TABLE 2
MESS
P1
P2

Figure C-8

APPENDIX D
COMMUNICATIONS ERROR DETECTION AND RECOVERY

The following is a description of errors which can occur on the VTS bus. The intention here is to indicate the symptoms which will be detected for each type of error, to present possible causes for that error, and to define the corrective action taken by the system.

D.1 ERROR DETECTION

There will be two types of operating system processes involved in error detection. Each processor will have a Local Error Reporting Center (LERC) to record and analyze the faults detected by the processor software and interface processor. There will also be one Main Error Reporting Center (MERC) which will be responsible for coordinating the error reports from each LERC (Figure D-1)*. Most problems will be reported to the MERC on both buses.

The error reporting rules are summarized below. Note that in describing these rules bus #1 designations are used (e.g. I_k, D_k, R_k, etc.). However, all these rules apply equally to bus #2 (I'_k, D'_k, R'_k, etc.).

*There will actually be a copy of the MERC software in each main processor; but only one copy will be the active MERC.

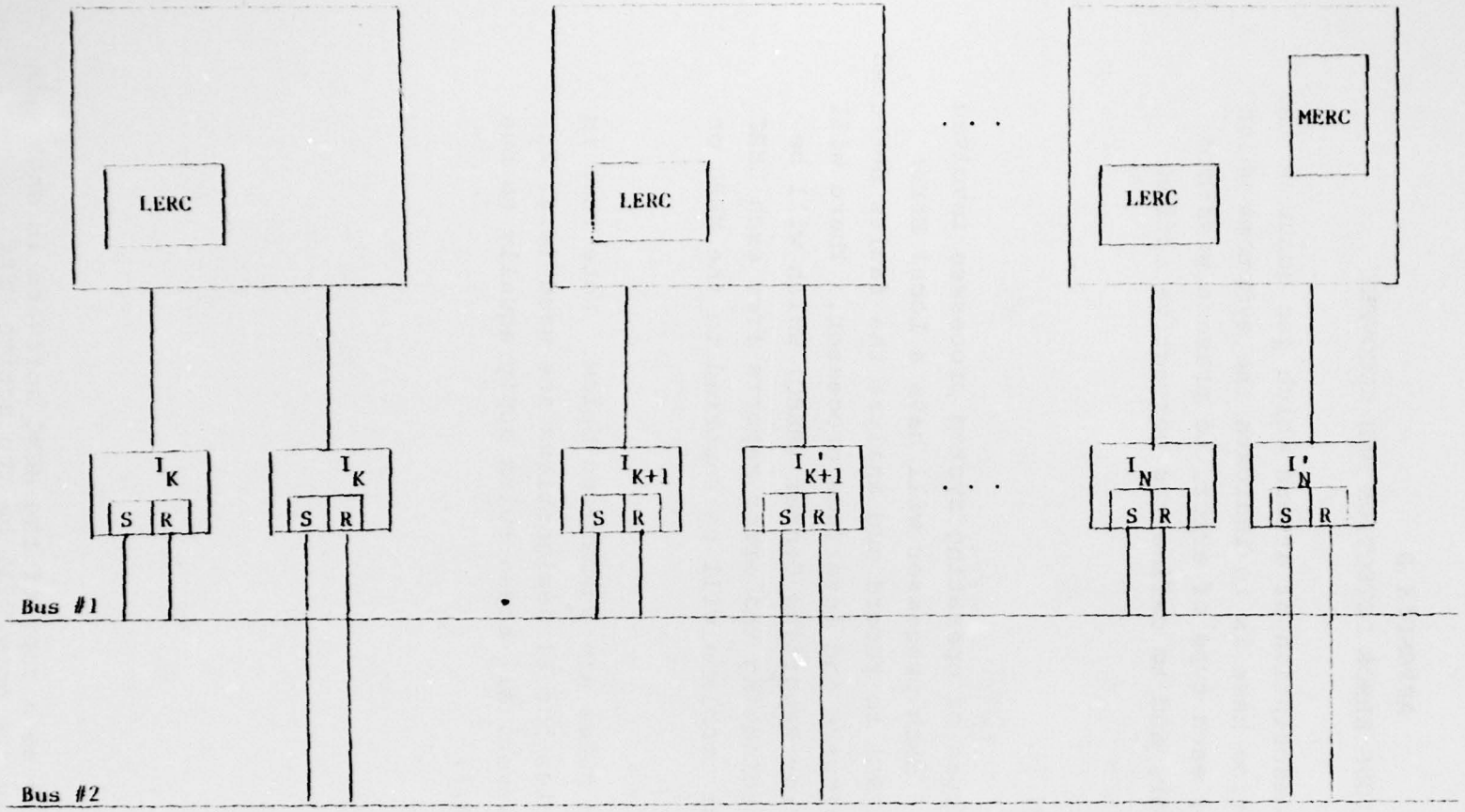


Figure D-1

Detector: Interface I_K

Symptom: No activity on the bus after sending a control message poll to I_{K+1}

Possible Causes:
1. Defective bus
2. Defective receiver on I_{K+1}
3. Defective sender on I_K

Corrective Action: I_K reports the problem to its LERC. The LERC immediately sends a report to the MERC on both busses. Then I_K tries to pass the control message to I_{K+2}. If the problem is a defective receiver on I_{K+1}, communication will continue.

Detector: Interface I_K

Symptom: No message activity on bus

Possible Causes:
1. Defective bus
2. Defective receiver on I_K
3. Defective sender or receiver on another interface

Corrective Action: I_K reports the problem to LERC which sends a message to MERC on the other bus.

Detector: Dispatcher D_K

Symptom: A missing or out of sequence packet

Possible Causes: Garbled transmission

Corrective Action: The dispatcher requests a retransmission of all packets since the last packet received in sequence. Since the packets are always sent in order, the receiver will immediately detect missing packets. This eliminates the need to acknowledge each packet of a long message.

Detector: Dispatcher D_K

Symptom: Incorrect Cyclic Redundancy Code

Possible Causes: Garbled transmission

Corrective Action: The dispatcher requests a retransmission from the sender. (Note: the sender code may be incorrect. In this case the request for retransmission will be ignored.)

Detector: Dispatcher D_K

Symptom: Large number of transmission errors detected over a period of time

Possible Causes: Intermittent failure of the bus or receiver on I_K

Corrective Action: Dispatcher reports to LERC which sends a message to MERC on both buses.

Detector: Router R_K

Symptom: Inconsistencies in LOCAL and GLOBAL tables

Possible Causes: Incomplete reconfiguration

Corrective Action: Router sends a message to MERC.

Detector: Router R_K

Symptom: Unable to deliver a message notice from A because B's IN queue is full.

Possible Causes: 1. Process B is not responding
2. The system is overloaded

Corrective Action: The router sends a special answer to A's message: 'B's IN list is full'. The router wakes A up if A is waiting for an answer. Process A may either try to send the message later or inform MERC.

Detector: Router R_K

Symptom: Attempt by A to send a message to a nonexistent or inactive process (LOCAL and GLOBAL tables are consistent).

Possible Causes: 1. Missing receiver process
2. Defective sending process

Corrective Action: The router sends a special answer to A's message: 'Non-existent process'. The router wakes A up if A is waiting for an answer.

Detector: Timeout monitor for Router R_K

Symptom: A message in A's OUT queue has not been answered

Possible Causes: 1. Original message notice never received
2. Interface I_K is defective
3. The bus is overloaded or defective

Corrective Action: Send out another message notice. If there is no response to the second message, the router reports the problem to LERC which sends a message to MERC on both buses.

Detector: Timeout Monitor for Process A

Symptom: Process B has not answered a message sent by A

Possible Causes:
1. The system is overloaded
2. Broken communication link
3. B's processor is defective
4. A's processor is defective

Corrective Action: Process A sends a message to MERC

Detector: Process A

Symptom: Receives incorrect data from B

Possible Causes:
1. Garbled transmission
2. A's processor defective
3. B's processor defective

Corrective Action: Process A sends a message to MERC and requests retransmission from B.

Detector: Timeout Monitor for Process A

Symptom: Not scheduled at correct time

Possible Cause: System is degrading

Corrective Action: Process A sends a message to MERC

D.2 RECOVERY

In the remainder of this section, four fault situations and the methods used to recover from them are discussed.

External Interference

External interference may contaminate any portion of a bus message. Some of these transmission errors may be detected and corrected by the bus interfaces. If the receiver address and the sender address on the transmission (A_R and A_S in Figure 7-7) are intact, the receiver can ask for a retransmission. The intended receiver will examine the message and request a retransmissions from the sender if there are any inconsistencies in the message count or checksum.

If the receiver address or sender address is garbled, the transmission will be lost. Either the request for retransmission will never be issued or the request will go to the wrong sender and be ignored.

A lost transmission will be detected by the dispatcher if the transmission is one packet from a long process message or answer. The next packet to arrive will be out of sequence. Any missing or delayed communication will eventually be noticed by the time-out monitor for the router.

Failure of a Bus or an Interface Processor

All message traffic must be switched to the remaining working bus at the first hint of trouble on one of the busses. This will allow the system to continue normally while the MERC analyzes the problem. When a bus or any interface attached to it fails, the MERC will begin to receive a collection of error messages on the working bus (or on both busses). With the arrival of the first message, the MERC will broadcast a command on the working bus to switch all communication to the working bus.

The MERC will try to determine which component has failed by examining the error messages it has received and by sending out diagnostic messages. For example, if all of the error messages arrived only on bus #2, then bus #1 may be dead. (Error messages are supposed to be sent on both busses.) However, it is possible that the receiver section of the interface for the processor containing the MERC is defective. The MERC will use diagnostic messages to determine which component, the bus or the receiver, has failed.

Power Failure

At the start of a power failure, each processor will save its registers. The rest of the data in the processors is already in nonvolatile memory. Only the transmission on the bus will be lost. After the power is restored, the MERC will restart communication on the bus by sending out a control message poll.

Processor Failure

The Main Error Reporting Center manages the recovery operation after a processor fails. When a processor begins to fail, the MERC will receive error reports from many sources. The MERC analyzes the reports to determine which processor has failed and distributes the functions from the failed processor to the remaining processors.

There are several major problems to overcome during the recovery. The MERC will receive error reports from many sources including the failing processor. But the failing processor may send inaccurate reports. For example, if processor 1 has failed it may send reports that processor 2, processor 3, and processor 4 have failed. At the same time processor 2, processor 3, and processor 4 may each send reports that processor 1 has failed. The MERC must be able to ignore the reports from the failing processor, processor 1. The

problem of determining which processor to disable is compounded if the failing processor contains the MERC. The processor will not only send false error reports but may attempt to disable a healthy processor. To prevent this, the MERC must examine the health of its own processor through extensive self check diagnostics before any reconfiguration of the system. Additionally, in order to verify that the active MERC and its processor are working, the active MERC will be periodically sending messages to the inactive MERC(s) over both busses. If the inactive MERC fails to receive these messages, it will assume MERC has failed and will take over and become the active MERC.