Bolt Beranek and Newman Inc.



12

Report No. 4458

LEVEL II

# Quarterly Technical Report: TCP for the HP3000



May 1980

Prepared for:
Defense Advanced Research Projects Agency

QUARTERLY TECHNICAL REPORT:
TCP FOR THE HP3000

May 1980

This research was supported by the Defense Advanced Research Projects Agency under the following contract: MDA904-80-C-0214

Submitted to:

Director
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

Attention: MIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. AD-A086 832 / 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) Quarterly Technical Report: TCP for the HP3000 | 5. TYPE OF REPORT & PERIOD COVERED 12/15/79 to 4/30/80 |
| | 6. PERFORMING ORG. REPORT NUMBER BBN-4458 |
| 7. AUTHOR(s) R. D. Bressler | 8. CONTRACT OR GRANT NUMBER(s) MDA904-80-C-0214 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton St., Cambridge, MA 02138 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order No. 3214 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd., Arlington, VA 22209 | 12. REPORT DATE May 1980 |
| | 13. NUMBER OF PAGES 11 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DSSW Rm. 1D 245, The Pentagon Washington, DC 20310 | 15. SECURITY CLASS. (of this report) UNCLASSIFIED |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE/DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Transmission Control Protocols (TCP), HP3000, ARPANET

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Quarterly Technical Report covers one phase of an ongoing research effort to implement TCP protocols on an HP3000 computer system.

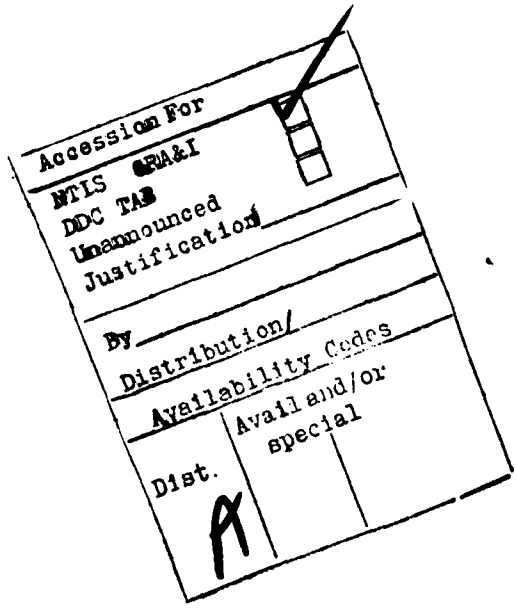
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

# Table of Contents





# 1 Introduction

This QTR covers one phase of an ongoing research effort to implement TCP protocols on an HP3000 computer system. The phase of the work covered in this report is an analysis of the HP3000 computer and its MPE operating system. The analysis places special emphasis on the elements of the operating system which will directly affect the implementation of network protocol software.

# 2 Current HP3000 Structure

The HP3000 CPU is a medium speed machine which uses a stack architecture. It executes uncomplicated instructions in one to two microseconds. Code and data are separate and thus all code is re-entrant. There are approximately 38 hardware registers which make up the state of the processor, most of which are associated with the stack (data) and the current instruction address (code).

Memory is divided into segments. A segment is a contiguous block of memory of any desired length up to 32K words. Individual segments are swapped in and out of memory as needed. Memory paging, a scheme which uses fixed size memory chunks as the basis for memory swapping, is not used in the HP3000. A segment may be designated as code or data by the operating

system.

## 3 Process State Manipulation

The processor state contains pointers to the beginning and end of the code and data segments. In a program, which may consist of more than one code segment, the fastest type of procedure call is a call within the same code segment -- a procedure frame is pushed onto the stack, but no change need be made to the code segment descriptors. A procedure call into a different code segment is more expensive because the CPU must pick up the new segment number and offset from the linkage area of the current code segment and then change the segment descriptor registers to redefine the current code segment. In addition, if the new code segment is not currently swapped into memory, the process must be suspended until the segment is available. Similarly, when the called procedure returns, the calling segment may have been swapped out, and again the process might have to be suspended.

An important side effect of a procedure call into another segment is that the new segment may specify that it is to be run in privileged mode. Calling a procedure inside such a segment would place the calling process in privileged mode until the called procedure returned. This technique is used to effect system calls in the MPE operating system because it was felt this

would be more efficient than a process context switch.

Switching from one process to another requires saving all 38 hardware registers of the current process, loading the registers for the new process, and insuring that both the code and data segments needed by the new process are present in memory. It also involves running the process scheduler in the operating system. For comparison, this is considerably slower than a procedure call between code segments when the new code segment is present in memory, though about the same if the new code segment is not in memory. Furthermore, any program function that requires switching process contexts must recognize that, in a multiprogramming operating system, waking up a process does not guarantee that the awakened process will be the next process scheduled to run.

## 4 Network Interface Hardware

The interface unit between the HP3000 computer and the ARPANET machines will be HP's Intelligent Network Processor (INP). This device consists of two boards located in the HP 3000 main cabinet. It is a microprogrammed interface unit whose microcode is down-line loaded by HP 3000 software. HP will supply the microcode to make the INP obey the X.25 LAP protocol and will supply the device driver necessary to access the INP.

The INP will be connected to a BBN C30 (MBB) computer. This machine will convert the X.25 protocols from the INP into suitable ARPANET protocols.

## 5 Operating System Software

The operating system for the HP 3000 is known as the Multiprogramming Executive System (MPE). It offers both batch and interactive job capabilities and allows multiple concurrent users of either type. It offers a file system which manages files on disk, magnetic tape and/or punched cards. Some I/O devices, such as the line printer, have spooler programs built in to the system.

User programs are run as processes within MPE. Each process has associated with it a code segment and a stack (data) segment. In privileged mode, it may run in "split-stack mode", where it is allowed to have two data segments. The most common use of split-stack mode is to access tables in the operating system during system calls (known as Intrinsics; see below).

The design of MPE is greatly influenced by the HP3000 hardware architecture. MPE's organization heavily relies on operations which incur little processor overhead while avoiding operations which incur large amounts of processor overhead. The most striking example of this is the MPE's dependence on user

processes for a large number of what would ordinarily be considered systems functions. MPE avoids the use of "systems" processes to perform these systems functions.

This design organization is a direct result of the stack architecture of the HP3000. The large number of status registers which must be saved when a new process is invoked makes process switching a very expensive operation. The time needed to perform a procedure call into a new segment of system code is typically less than the time to switch context from one process to another. Writing efficient code for this machine has thus led to organizing the system as relatively independent "utility" routines callable by the user rather than as a collection of separate processes which manage I/O devices and system utilities. These operating systems calls, called Intrinsics, are implemented as subroutine calls into system code segments. The Intrinsics use the split stack mode to separate the operating system functions from the user's program. The program segments which implement the Intrinsics run in a privileged mode which allows them to directly access system tables and I/O device tables.

One notable side-effect of this design is that system resources such as I/O devices are assigned to only a single program and are not normally shared. This approach has allowed the systems programmers to create a complex operating system without tackling the problems of interprocess communication and

resource sharing. As will be discussed later, it also has a significant effect on protocol software design.

## 6 Input/Output

Input/Output operations are typically a two step operation. The first step is initiation of the desired operation. This involves checking to insure that access to the device is allowed (software protection), and issuing I/O instructions to the device to initiate the desired action. This step usually occurs as a result of an intrinsic call to the device handler code and thus is executed on the user's stack. The second step is the operation completion handling. This may occur using either the Interrupt Control Stack (ICS) or the System Control Stack, neither of which is the user's stack. The choice of which stack to use depends on the specific device's function.

A consequence of this system design is that "system code" tends to be executed using the data stack of the first user process needing the function. If process 1 wants to do an I/O operation, it invokes a system procedure which knows how to manage that I/O device. If now process 2 wishes to invoke the same device, and if the device is capable of supporting more than one request concurrently, it invokes the same routine. To avoid multiprocessing hazards in issuing I/O commands, the system procedure first checks to see if it is the first invocation of

itself -- if not, it queues the request and exits; if it is, it proceeds to issue the I/O instructions. If the request was queued, it is assumed that the first process will detect the newly queued request and process it also. The first process is thus performing system functions for the second, and all later, processes, and will be charged run time for doing their work. In practice, we do not expect this to be significant, but in theory, the first process could run indefinitely, even if its own request has long since completed.

## 7 Interprocess Communication

Interprocess communication under the current version of MPE is a problem. One technique that may be used is that of the logical device. This is employed chiefly to accomplish multiplexing of physical devices. The facility is implemented by creating a new entry in the system's Device Information Table, and by creating a set of procedures to be the "device handler". The handler will be run in privileged mode.

Like other system device handlers, the procedures to manage the device are invoked directly by the user process, and the user's stack is used by the system code. This has the advantage of speed, since it avoids some process context switching.

There are a number of drawbacks to this technique. First, the Device Information Table entry must be maintained as if it were a real hardware device. This requires knowledge of all the MPE internal functions that might access this table. Furthermore, since these tables are system internal, they are subject to change with each new release of MPE. Use of the table requires Priviliged Mode. Bugs in the code would have a greater chance of crashing the system. The greatest drawback is that logical devices are still under development at HP, and are more than usually likely to change over time.

A new operating system feature, not yet released officially, that has been written for *MPE is an interprocess* communication method known at HP as message files. These correspond to Unix ports, and allow unrelated processes to communicate with one another. Each message file has one or more "reader" processes and one or more "writer" processes. During use, these files act as FIFO queues. If a reader finds the queue empty, it is hung until something is written into the file. If a writer finds the queue full, it will hang until a reader appears. Both reader and writer may specify a timeout on how long they will wait if hung.

Message files are implemented using the file system. Read, write, and query commands are all patterned after the file system commands. The message file code is designed so that if readers and writers stay more or less in synchronization, disk I/O will

not be needed. However, if the writers get far enough ahead of the readers, the message file will start being spooled out onto disk.

Message files are to be introduced as user level functions by HP, and, as such, their use will not change with new releases of the operating system. Code for this feature has already been implemented at HP and is available with both MPE III and the future MPE IV. They appear to be relatively easy to use and do not require knowledge of the internals of the operating system. Their chief drawbacks are that a process context switch is required between writer and reader, and that some file system overhead is incurred.

Timeouts, as seen in message files, are another new HP function that will be available. The older version of timeouts simply suspended the process for a fixed amount of time, but did not allow the process to be awakened by the completion of an I/O event during its sleep. The new version is equivalent to setting a timer whose alarm may be awaited with the same IOWAIT Intrinsic that awaits I/O completion. It allows a process to wait for either some I/O device operation completion or the passage of some maximum amount of time, whichever occurs first. Alternatively, a timeout could be used to insure that waiting for a specific event will terminate if the expected event does occur soon enough. There will be both user level and system internal

ways of accomplishing timeouts.

## 8 Existing INP Software

The code to drive the INP is part of the CS/3000 Communications Software package from HP. It contains code to send and receive packets via the INP and code to manipulate the Device Information Tables. The code also allows the user to down-line load microcode into the INP memory. It contains Intrinsics to open and close the line and to read and write packets. The microcode makes the INP be an X.25 LAP, as opposed to LAP B, interface. It also allows the INP to buffer up to eight 128-byte packets. These packets are read by CS/3000 as soon as possible to keep the INP from losing packets for lack of buffer space in the INP. This technique allows the INP to function as a full duplex device, even though the MPE operating system offers only a half duplex control mechanism in its software.

# DISTRIBUTION

ARPA
Director (3 copies)
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209
Attn: Program Manager

V. Cerf
R. Collison

DEFENSE DOCUMENTATION CENTER (12 copies)
Cameron Station
Alexandria, VA 22314

BOLT BERANEK AND NEWMAN INC.
50 Moulton Street
Cambridge, MA 02138

R. Bressler
J. Sax
W. Edmond
J. Haverty
M. Wingfield
D. Buckler
K. Hahn
F. Heart
R. Brooks
Library