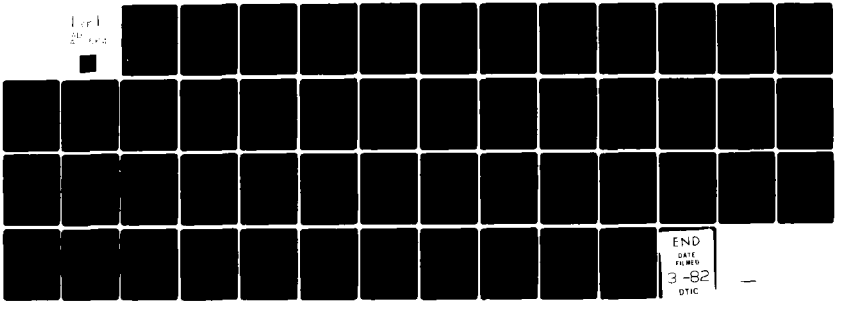
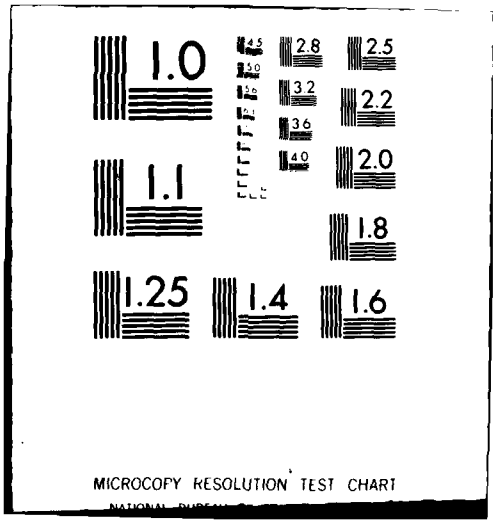
AD-A111 564 FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA W--ETC F/G 17/2
(KSOS) SYSTEM SECURITY PLAN. REVISION 2. KERNELIZED SECURE OPER--ETC(U)
1980 MDA903-77-C-0333

UNCLASSIFIED NL



END
DATE FILMED
3-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART

(2)

KSOS
SYSTEM SECURITY PLAN.

Contract MDA 903-77-C-0333
CDRL 0002AN

Prepared for:

Defense Supply Service - Washington
Room 1D245, The Pentagon
Washington, DC 20310

Approved for public release: distribution unlimited.

Ford Aerospace &
Communications Corporation
Western Development
Laboratories Division

3939 Fabian Way
Palo Alto, California 94303

KSOS System Security Plan

Ford Aerospace and Communications Corporation

Western Development Laboratories Division

1. INTRODUCTION

The purpose of this document is to describe how the security requirements of the KSOS project will be met. Security considerations in the design of KSOS-based systems and in the operation and maintenance of KSOS systems are to be documented in forthcoming reports.

1.1 Applicable Documents

The following documents, of exact issue shown, form a part of this specification to the extent specified herein. In the event of a conflict between the referenced documents and the contents of this specification shall be considered a superseding requirement. In the text, references to these documents are in the form [Name date], e,g, [Biba 75].

1.1.1 Directives, Manuals and Standards

a. DoD 5200.1-R Information Security Program Regulation

b. DoD 5200.28 Security Requirements for Data Processing (ADP)

c. DoD 5200.28-M ADP Security Manual

d. MIL-STD-483 Configuration Management

e. MIL-STD-490 Specification Practices

f. MIL-STD-1521A Technical Reviews and Audits

1.1.2 Reports

a. [A-Specs 78] "KSOS System Specification (Type A)", WDL-TR7808 Revision 1, Ford Aerospace and Communications Corporation, Palo Alto, CA (July 1978).

b. [Bell 73] Bell, D.E. and LaPadula, L.J., "Secure Computer Systems", ESD-TR-73-278, Volume I-III, MITRE Corporation, Bedford, MA (November 1973 - June 1974).

c. [Biba 75] Biba, K.J., "Integrity Considerations for Secure Computer Systems", MTR-3153, MITRE Corporation, Bedford, MA (June 1975).

d. [Boyer 79a] Boyer, R. S., and Moore, J. S., "A Computational Logic, ACM Monograph Series, Academic Press. New York, (October 1979).

e. [Boyer 79b] Boyer, R. S., and Moore, J. S., "A Theorm Prover for Recursive Functions: A User's Manual", Report CSL-91, SRI International, Menlo Park, CA, (June 1979).

f. [Emulator 78] "KSOS Computer Program Development Specification (Type B5): UNIX Emulator", WDL-TR7933, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

g. [Feiertag 77] Feiertag R. J., "A Technique for Proving Specifications are Multilevel Secure", Report CSL-109, SRI Internaltional, Menlo Park, CA (June 1980).

h. [Floyd 67] Floyd, R. W., "Assigning Meaning to Programs", Mathematical Aspects of Computer Science, vol 19, pp. 19-32 (1967)

i. [Hoare 72a] Hoare, C. A. R., "Proof of Correctness of Data Representations", ACTA Informatatica 1, pp. 271-281 (1972)

j. [Hoare 73b] Hoare, C. A. R., "Proof of a Structured Program: the Sieve of Eratosthenes", Computer Journal, 15 pp 321-325, (November 1972).

k. [Kernel 78] "KSOS Computer Program Development Specification (Type B5): Security Kernel", WDL-TR7932, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

l. [Lampson 73] Lampson, B., "A Note on the Confinement Problem", CACM, Volume 16, Number 10, pp 613 - 615 (October 73).

m. [Levitt 79] Levitt, K. N., Robinson, L., and Silverberg, A., "The HDM Handbook, Volume III: A Detailed Example in the Use of HDM", SRI Internaltional, Menlo Park, CA (June 1979).

n. [Lipner 75] Lipner, S.B., "A Comment on the Confinement Problem", Proc. Fifth Symposium on Operating Systems Principles, ACM SIGOPS Review, Volume 9, Number 5, pp 192 - 196 (19-21 November 1975).

o. [NKSR 78] "KSOS Computer Program Development Specification (Type B5): Non-Kernel Security-Related Software", WDL-TR7934, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

p. [Parnas 72] Parnas, D.L., "A Technique for Software Module Specification with Examples", CACM, Volume 15, Number 5, pp 330 - 336.LI [Robinson 79] Robinson, L., Levitt, K. N., and Silverberg, A., "The HDM Handbook, Volume I: The Foundations of HDM", SRI Internaltional, Menlo Park, CA (June 1979).

q. [Roubine 77] Roubine, O., Robinson, L., "Special Rel[illegible] Manual", 3rd ed., Technical Report CSG-45, SRI International, Menlo [illegible]k, CA (January 1977).

r. [Silverberg 79] Silverberg, A., Robinson, L., and Levitt, K. N., "The HDM Handbook, Volume II: The Langauges and Tools of HDM", SRI Internaltional, Menlo Park, CA (June 1979).

s. [Verif 78] "KSOS Verification Plan", WDL-TR7809, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

t. [Walter et al. 74] Walter, K.G. et al., "Primitive Models for Computer Security", ESD-TR-74-117, Case Western Reserve University, Cleveland, OH (January 1974).

1.2 Security Approach

This section describes how the security requirements of KSOS are to be met in the development of the system. The basic purpose of the KSOS project is to provide a mini-computer system which can be approved to allow simultaneous processing of information of differing security levels. In KSOS, this goal is met through the use of mathematical formalism throughout the development process.

It is generally accepted that security cannot be added-on to a system. Rather, to have confidence in the security of a system, mathematical formalism must be carefully applied throughout the development cycle. Only through the ability of formal techniques to generalize the behavior of systems over wide ranges of input stimuli, can one assure that there are no circumstances in which the system permits information compromise.

From its inception, the KSOS project has attempted to follow the basic principles of the Hierarchical Development Methodology (HDM) [Robinson 79], initially developed by researchers at SRI International. HDM is a conceptual framework in which mathematical rigor can be applied at each distinct phase of the system development cycle. Through the application of the principles of HDM, it is possible to create a mathematically sound logical chain of implication that the actual system realized by the "bits in the computer" complies with some mathematical abstraction of the desired property.

The KSOS project has enlarged and expanded upon HDM in order to better comply with project requirements and Government software development procedures. The KSOS project has also funded development of software tools envisaged as parts of an HDM development environment. These tools include tools for proving properties about a system design expressed in SPECIAL, for rigorous testing of the resulting system, and for formal proofs of correctness of the system software.

Much of the mathematical underpinnings of the KSOS system's security assurance procedures are discussed in the KSOS Verification Plan [Verif]. This document will not duplicate this material. Rather, it will expand upon how the principles discussed in the Verification Plan are utilized in the assurance that KSOS correctly obeys the DoD security policies. It will also discuss how "classical" security techniques are used to maintain the integrity of the resulting product.

1.3 Organization

In the KSOS project there are two organizationally distinct efforts which contribute to the overall system security assurance. The first is the Verification Task. The Verification Task is a separate effort distinct from the Design and Implementation Tasks. Its task leader reports directly to the Program Manager. The Verification Task includes the following:

a. Production of the formal specifications for the trusted parts of the system.

b. Proving that the formal specifications comply with the security model.

c. Selection of the implementation language for trusted software.

d. Proving that selected portions of the deliverable code comply with the formal specifications.

e. Consulting and advising project personnel and management on the use of HDM.

The Verification Task is, in part, staffed by personnel from SRI International, a subcontractor to FACC on the KSOS project. Some of the tools used in support of the HDM as applied to KSOS were developed by SRI, both on the KSOS contract and on other Government contracts.

The second component of the KSOS project which contributes to the system's security assurance is the Testing Task. Like the Verification Task, the Testing Task is an organizationally distinct component, reporting directly to the Program Manager. The Testing Task is closely aligned to the formal verification efforts. Conceptually, the proof that a program complies with its specifications can be viewed as a form of symbolic testing. Thus, testing and verification share a similar foundation. KSOS has gone beyond the state-of-the-art in testing. The resulting KSOS system will be tested against its (proven) formal specifications. In this way, the "bits in the computer" which realize the KSOS system will be demonstrated to comply with what the formal specifications say that the system should do.

There are other aspects of the KSOS program which contribute to the overall system security. The integrity of the developed system is insured by the following steps:

a. all personnel engaged in the design and implementation of trusted software components are cleared (at least) to the SECRET level.

b. the system will be delivered from copies maintained at the SECRET level.

c. cleared personnel will review all changes to the master copies of the system

d. FACC will make available to the Government the tools used to prove that the formal specifications and code are correct to allow the Government the opportunity to re-verify the system in an environment of whatever classification is desired.

1.4 System Security Engineering Program

Because the security of the resulting system is the most important design goal, there need not be a separate and distinct Security Engineering Program. Rather, the desire for improved system security permeates every design and implementation decision.

The model of security which KSOS enforces is what has been termed a "flow model". That is, the system enforces restraints on the flow of information between entities in the system. The system enforces five distinct rules on information flow:

a. the simple security rule -- information may be read only from sources whose security level is at or below the security level of the reader

b. the security *-rule -- information can be written only to sources whose security level is at or above the security level of the writer

c. the simple integrity rule -- information can only be read from sources whose integrity level is at or above the integrity level of the reader

d. the integrity *-rule -- information can only be written to destinations whose integrity level is at or below that of the writer

e. the discretionary access rule -- information flow can only occur when the owner of the object being read or written has allowed it

In the above discussion, a security level consists of a hierarchically ordered security classification category, e.g. TOP SECRET, and a (possibly null) set of security compartments such as NO FOREIGN DISSEMINATION, or specialized need-to-know compartments. An integrity level consists of only a hierarchically ordered integrity classification category, since integrity compartments function identically to security compartments. In KSOS, three integrity classification categories have been used:

USER < OPERATOR < ADMINISTRATOR

In the following discussions the term "level" means the combination of a security level and an integrity level. When discussing comparisons between levels, what is meant is that the classification categories (both security and integrity) satisfy the relationship, and the the two compartment sets satisfy the relation interpreted as set inclusion.

The KSOS discretionary access model is patterned after that of UNIX. Each entity has an owner and a set of nine boolean conditions which express the permissions to access that object. The permissions are for reading, writing, and search/execute access by the owner, members of the same group as the owner and for all others.

It has been argued that models like this can be used for essentially all reasonable, deterministic protection requirements. Whether or not this is true, the model does clearly incorporate the existing DoD policies, and can be used to satisfy future requirements of similar character.

The threats against which KSOS must protect its data are also documented in the System Specification. Basically, KSOS must be able to cope with an arbitrary program executed in an unprivileged mode. The system must prevent information compromise regardless of what the program does. In addition KSOS is

required to identify so-called covert channels, paths by which programs may pass data in violation of the security rules through indirect means. For such channels, the bandwidth is to be estimated and also an estimate is to be made of the performance penalties incurred as a result of reducing this bandwidth if desired. Specific KSOS-based systems can use this information in discussing potential threats and scenarios. Because most of the threats are system and installation specific, it is not possible to examine them in detail in this document or in the planned Clandestine Vulnerabilities Analysis.

Providing provable security in KSOS has been a paramount consideration since the inception of the project. The initial design of the system completed during Phase I of the contract included formal specifications written in SPECIAL for the KSOS Kernel. Subsequent versions of the system design have included revised and expanded Kernel formal specifications and formal specifications for the Non-Kernel Security-Related Software (NKSR).

The implementation language for the trusted parts of the system was chosen in part from considerations of eventual proofs of correspondence between the formal specifications and the code written in the selected language. Many popular languages are not amenable to such proofs. The eventual selection, Modula [Wirth], offers excellent potential for success in the completion of code-to-specification correspondence proofs. The tools to support such proofs are under development and refinement, and have been used successfully in complete examples. Modula encourages modularization and information hiding, two features which improve the robustness and maintainability of the system.

The testing effort for KSOS also is designed to enhance the confidence in the system's security properties. In addition to the "classical" testing techniques widely accepted in the Government software development community, KSOS adds a significant dimension in performing tests which are derived from the formal specifications. A distinct test is provided for each logical path through the formal specifications. The ensemble of test cases so generated provides an important link between the formal specifications and the realized system.

For KSOS, the real impact on system security comes from formal verification of specifications and code. It is these formal methods which are the heart of the KSOS System Security Engineering Program. As shown in Figure 1, the KSOS system verification consists of two distinct proofs, and the testing of the realized system against it formal specifications.

The first type of proof is a proof that the formal specifications comply with an abstract model of the desired DoD security policy. This proof is actually a large number of proofs that the security level of objects read by a process is less than or equal to the security level of the process, and that the security level of objects written is greater than or equal to the level of the process. The current version of the KSOS formal specifications for the Kernel interface cause about 1600 of these simple inequalities to be generated. Because they are simple and because there are so many of them, mechanical proof techniques are used. These mechanical techniques include the generation of candidate theorems from the specifications, the proof of trivial cases within the theorem generator, and the use of a powerful theorem proving program developed by Boyer and Moore [Boyer 79b] for proof of the remainder. The process is largely automated, involving minimal user intervention. As shown in Figure 2,

Figure 1

the process begins with a set of formal specifications written in SPECIAL.

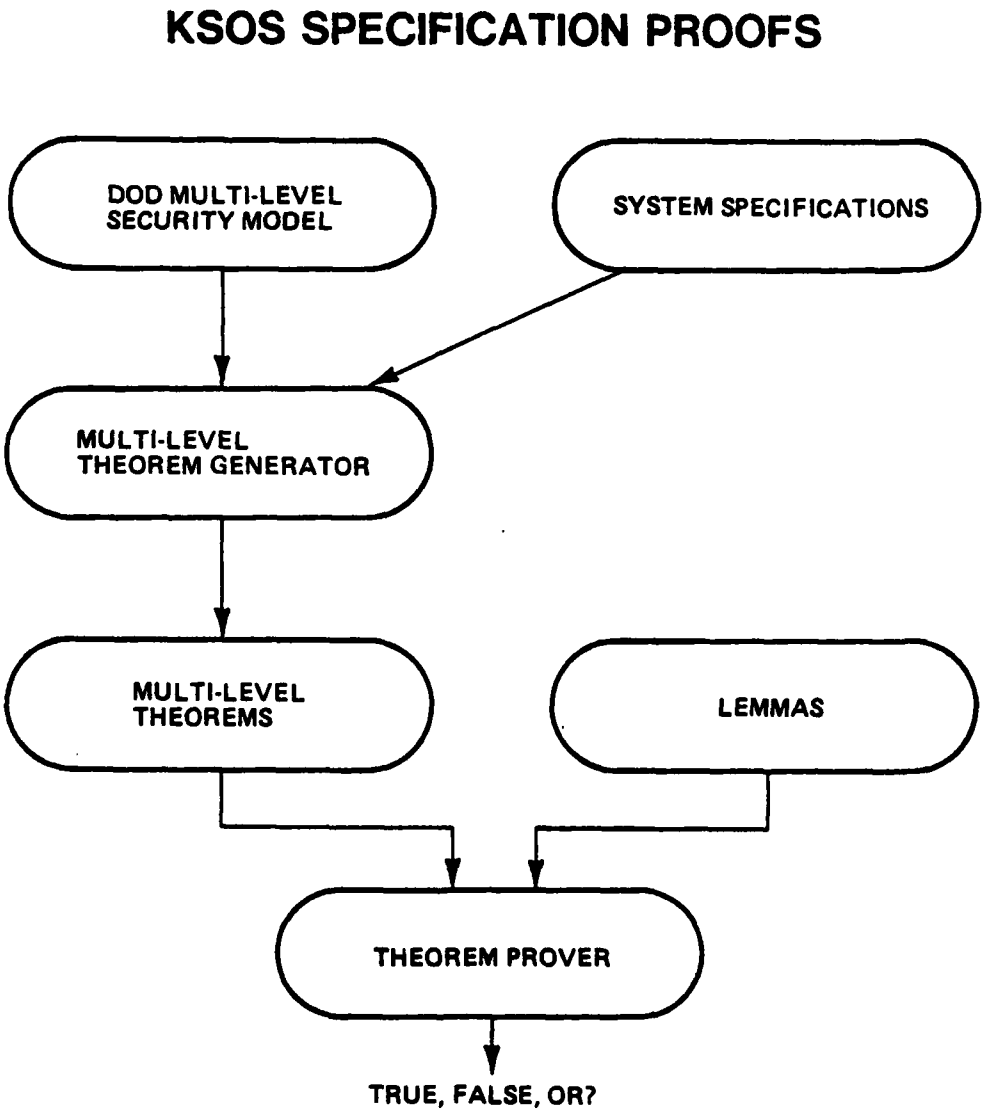


Figure 2

These specifications will already have been processed through one of the two

SPECIAL processors used in KSOS (one developed by SRI for the TOPS-20 environment, and one developed under FACC IR&D for the UNIX environment). The specifications are then processed by the multi-level security theorem generator developed at SRI. This program has embedded in it the rules of the multi-level security model. For a system specification to be multi-level secure, all information flows in it must be in accordance with the multi-level security model. As indicated above, the compliance of the system with the model manifests itself as a large number of formulae which are inequalities on the security level of objects read and written by the process. In practice many of these candidate theorems are trivially true. In the interests of speed, the trivial theorems are "proved" as part of the multi-level security theorem generator. The remainder of the theorems are submitted to the much more powerful (and therefore slower) Boyer-Moore Theorem Prover.

The method has been shown to correctly detect (by failing to prove one or more candidate theorems) both direct security violations, and the more subtle forms of violations occurring from indirect effects.

It is planned to enhance the proof of formal specifications by proving a more general class of properties of the specifications. This class are other flow-oriented invariants on the state variables (in SPECIAL, primitive VFUN's). As these plans are still being formulated, the exact restrictions on the class of invariants which can be stated and proved.

The second class of proof is a demonstration that the code of the system conforms with its formal specifications. These proofs require a more detailed level of specification than is present in the Kernel interface specifications contained in the B5 Specifications for the Kernel. This more detailed layer of specifications provides the input and output assertions for a Floyd-Hoare proof of the code. In HDM these two layers are tied together by means of mapping functions and abstract programs which relate data objects and control between the two layers. The environment for doing these code proofs is similar to that for doing the specification proofs, and is shown in Figure 3. Again, one input is the formal specifications for the system written in SPECIAL. For these code proofs, an additional level of specifications may be required. These are detailed specifications for the code, and hierarchy specifications which tie the low level specifications to the top-level specifications. This is how the input/output specifications for a program are related to the top-level specifications for the system. A second input to the code proof process is the Modula code to be proved. This is processed through a Verification Condition Generator (VCG) which produces the theorems to be proved. The function of the VCG is to produce theorems that show that the program transforms the system state which complies with its input assertions to a state which complies with the output assertions. These proofs take the form of showing that all paths from the entry to exit comply with the desired assertions on output state, given an assertion on the input state.

These types of proofs of programs have always been excruciatingly difficult to complete. Recently, considerable attention was paid in the research press to the completion of a proof for a 327 line program, which alleges to be one of the largest programs ever proven correct. The main difficulty occurs from having to synthesize exactly the right set of lemmas and additional assertions to enable the theorem prover to prove the resulting theorems. If the lemma set is

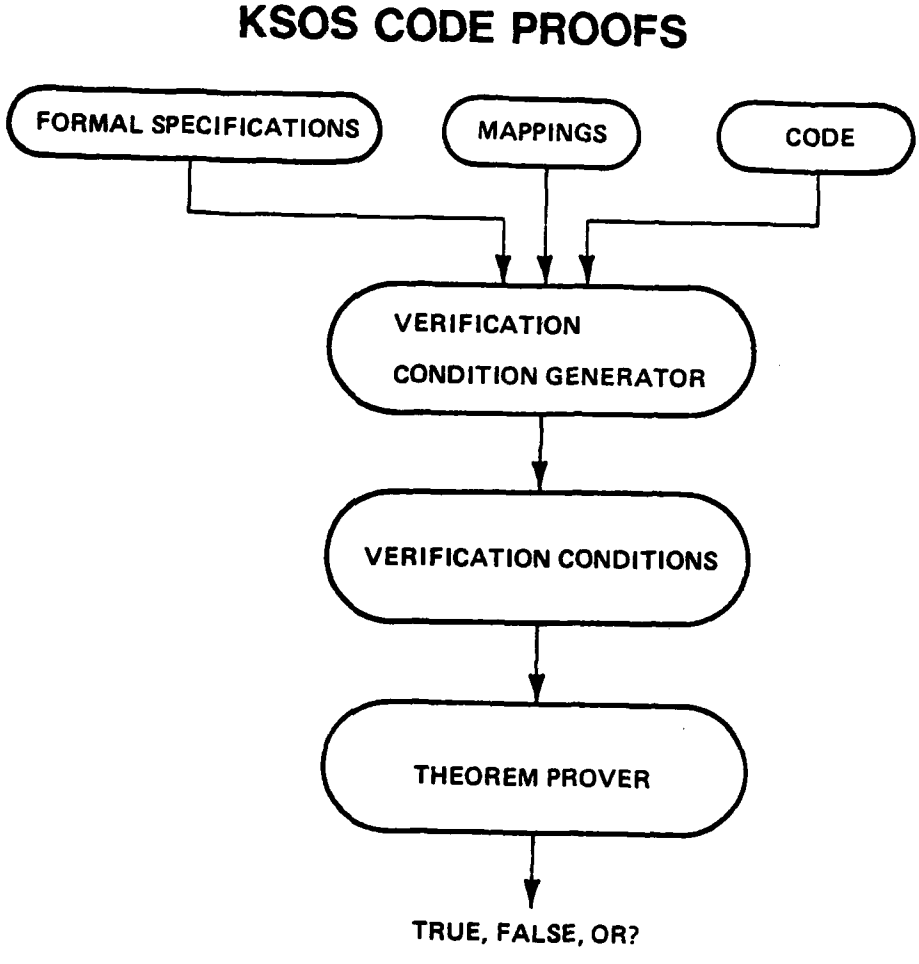


Figure 3

too weak or is incomplete, the theorems cannot be proved. If the lemma set is

too powerful, or overly rich, the theorem prover bogs down in the extraneous detail, and also fails to complete the proofs.

Because of the enormous cost associated with proofs of programs, KSOS has planned only to demonstrate the feasibility of the technique, and to provide a limited number of data points for estimating the costs of a more complete verification effort. It should also be noted that verification of parallel programs such as the KSOS Kernel is still well beyond the state-of-the-art. Although there is a great deal of active research in this area, it is premature to speculate about the potential for complete code verification of the KSOS Kernel.

This is not to say that the program proof process employed for KSOS is without benefit to the project. There are significant benefits which accrue from being prepared to undertake program proofs. The most important single factor influencing the difficulty of the program proof process is the selection of the implementation language. Most contemporary languages pose very serious additional impediments to the proof process over and above those intrinsic difficulties outlined above. The selection of the implementation language for the trusted portions of KSOS rested heavily on the desire to facilitate eventual code proofs. The Verification Plan outlined the criteria for choice of the KSOS implementation language. The eventual decision process was outlined in "Language Gram #2", which is attached to this report as Appendix A. The two most important criteria, both in terms of their impact on code proofs, and in terms of their weeding out of potential language candidates are the need for strong typing of variables and for the prevention of aliasing. These features are becoming more generally accepted as being good features for any programming language, regardless of whether or not proofs are contemplated. Thus, the need for a language amenable to eventual code proofs has lead KSOS to chose a language which incorporates features that are believed to improve overall program quality. It should be noted that the proposed new DoD standard programming language, Ada, would have been a very strong candidate for KSOS had it been available.

1.5 Program Data Flow

The system security engineering efforts discussed above are completely integrated into the design and implementation process. Members of the test and verification teams have been active in design of the system, and participate in reviews of designs and code. These reviews have lead to suggestions for alternative designs or changes in the implementation which would improve the testability of the system. Where security flaws have been uncovered by the verification efforts, the design has been changed to remedy the problem. Such integration is crucial to the success of a secure system. Unless all of the design and implementation staff is cognizant of the impact of the requirements for system security, no external System Security Engineering function can hope to provide the required security assurance.

1.6 System Security Engineering Functions

Because security is such an integral part of the entire KSOS effort, the traditional functions performed by a System Security Engineering organization are done by components throughout the project. The basic objectives to be satisfied were established in the System Specification and the Verification

Plan, which were approved by the Government at the end of the Phase I effort. These documents discuss a mathematical model of security that was derived from the work done by Bell and LaPadula [Bell 73]. The formal specifications for the system are to be proven to comply with this mathematical model. The model has been widely accepted as the basis for several different secure systems. It has also received intensive scrutiny over the time since its creation. The model is an abstraction of the relevant DoD policies. Thus, although these policies have had and can be expected to have subtle changes in emphasis or in implementation, the alteration of the model is not expected. In the design of KSOS, the potential for change is also managed by means of isolating the security decisions in a pair of functions, SMXflow and SMXdap. Should there be alterations in the environment which KSOS is operated, it is conceptually possible that these modules could be altered to accommodate the change. Such modifications could not be lightly undertaken, however.

As described earlier, the System Security Engineering functions are completely integrated with the design and implementation efforts. Personnel from the test and verification tasks are active participants in reviews of design and code. Their inputs on improvements for testability have been incorporated into the KSOS design. Where the verification efforts have uncovered security flaws, changes to the design have been made. Should the code proof efforts expose errors, these errors will be corrected in the deliverable code.

In discussing the synthesis of the KSOS security system, it must be recognized that the entire design process has been one of attempting to synthesize measures which would enhance system security. In evaluating proposed solutions to design questions, the overriding concern has always been whether the proposed solution would enhance system security. KSOS has done a great deal in the application of contemporary research results. The project has funded the development of tools to verify Modula programs, to prove that a formal specification satisfies the multi-level security model, and for the extraction of test case information from formal specifications. In a real sense, a strong secondary goal of the KSOS effort has been to move the technologies of formal specification and verification from the research laboratory into the production environment.

KSOS testing has been discussed above. The basic concept is to (at a minimum) demonstrate that the system exhibits the proper behavior in all cases described in the formal specifications. Since the formal specifications are intended to be a definitive statement of the system's behavior, such an approach is well founded. Additional tests will be performed which are derived from requirements stated in the System Specification, the Development Specifications and the Product Specifications. As much of this testing as possible will be automated, being driven from test scripts. In this way, the testing can be done more economically, and more repeatably than with more manual methods. It should be noted that these system and CPCI level tests are in addition to an extensive and organized program of unit tests and debugging tests. It is a requirement that all the code in trusted portions of the system be exercised by at least one test case. In many instances this can be accomplished by CPCI level or system level tests. However, some cases can only be exercised in unit tests. For example, abnormal condition handling ("This cannot happen") code cannot be easily exercised by calls upon the outer interfaces. To lend additional confidence to the system's security features, it is recommended that the Government sponsor a

penetration exercise. Such an exercise would allow a serious attempt to be made to cause information compromise.

The issue of changes to the system inducing security flaws is one which will be treated in more detail in the planned Secure Systems Maintenance and Support Plan. Potentially any change may cause a vulnerability to be induced. When a change is proposed to the system design, the revised formal specifications must be re-proven to comply with the multi-level security model. The proposed change would be iterated upon until this proof effort was successful. The test cases for the system would also have to be revised to incorporate the changes to the specifications. Where the proposed change does not affect the design (formal specifications), the burden of assessing the impact of the change falls upon the approval process. Approval of proposed configuration changes will include an explicit assessment of the impact upon system security that the change will have. After an approved change has been implemented, the CPCI and system level test cases will be repeated as need be to demonstrate that the change has not affected other previously correct cases.

FACC has had SRI International as a subcontractor for verification support for the life of the KSOS project. SRI has developed some of the tools used in the formal assurance of system security. FACC intends to perform the required verifications of the system's formal specifications, together with SRI personnel. The tools used in these proofs are all public domain and will be furnished to the Government. This would allow the proofs to be re-executed in a classified environment if desired.

The system generation process is discussed in the Development Specifications for the Non-Kernel Security-Related Software. The KSOS system delivered to the Government will be derived from copies maintained at the SECRET level. Information from the UNCLASSIFIED development system will migrate to the SECRET system packs only after review by cleared personnel. Because FACC has been able to obtain authorization to operate the GFE PDP-11/70 development system in classified mode, the rather cumbersome procedures discussed in the Maintenance and Support Plan involving the ARC at Moffett NAS may be replaced by a more reasonable system. The GFE system will be periodically operated in SECRET mode following the approved procedures for change of classification. (These procedures are included in this report as Appendix B.) Information will be migrated to the SECRET mode system in a controlled manner, and copies from the SECRET mode system will be reviewed by cleared personnel.

System installation and checkout will be accomplished following the principles enumerated in the Maintenance and Support Plan.

APPENDIX A

KSOS Trusted Component Implementation Language Selection "Language Gram #2"

(This report was originally included in the January, 1979 Status Report for KSOS.)

FACC-WDL-STD-KI-326

subject: Selection of a KSOS Implementation Language date:December 15, 1978

from: Thomas A. Berson

LanguageGram #2

1. LANGUAGE CHOICE FOR KSOS

The implementation of any program depends in subtle ways upon the programming language chosen for the task. The computational model provided by the language affects the way in which the implementors view their system. It provides not only the vocabulary of primitives from which the system is constructed but also the vocabulary of concepts in terms of which the implementors cast their communication with the system designers, with its users, and with one another.

Nowhere is the choice of programming language so critical as it is in the case of an operating system, which, to be useful, must itself establish and maintain an efficient and consistent computational model for Use by unknown users, in unforseen ways, over a long lifetime. In addition to these already stringent operating system requirements, the KSOS operating system is also required to provide strict enforcement of a security model and must be shown to conform in detail with a body of formal specifications. The ability to meet these added goals of security and provability depends heavily upon proper choice of programming language

Both the customer and the contractor anticipated the importance of a suitable language to the success of KSOS. Criteria for language selection were developed early and published in the KSOS Verification Plan [1]. Implementation language was identified at the PDR as an element of technical risk to the program. The identification was only too accurate; language selection has required almost continuous attention on the part of KSOS program management and technical personnel.

A decision procedure for language selection was also developed during Phase I and published as part of the Verification Plan. This procedure has served KSOS well and continues to be the operational rule-of-thumb for language selection and is therefore reproduced here verbatim from the Verification Plan, p. 40.

> "IF the Euclid compiler development effort at Toronto continues on its present course and gives adequate support THEN use it -- with some accommodation for hardware error signaling. Some subsetting of the language may be desirable, for simplicity, efficiency, and enhancement of any eventual verification.

ELSE IF Modula has been adequately extended and those extensions supported, THEN use it.

{[in March 1978] it seems that Euclid will be appropriate, although Modula provides an attractive alternative. At this point it is extremely unlikely that any more ELSE clauses are needed. However, there are other languages that may be appropriate, e.g.,}

ELSE IF ILPL is adequately supported, THEN use it.

ELSE IF Gypsy is adequately supported, THEN use it.

ELSE use UCLA Pascal.

{It is remotely feasible to use C, but only with the addition of various programming constraints, and with the explicit understanding that the kernel would subsequently be recoded in a more suitable language, before any program verification is attempted. However, such a course is neither necessary nor advantageous.}"

Examine this decision procedure in light of present [December 1978] knowledge.

— The Euclid compiler development at Toronto was unable to keep to its schedule. The language is presently unsupported.

+++ Modula is well supported. Extensions are, for the most part, easy to acccomplish.

— ILPL is not supported.

— Gypsy support is not yet adequate.

+++ UCLA Pascal is available.

It seems clear from the above that Modula should be the KSOS implementation language, and that UCLA Pascal should be identified as a contingency fallback language.

With the success of KSOS depending so heavily upon the proper choice of implementation language, and with the recent difficult experiences with Euclid fresh in mind, we undertook critical examination and comparison of Modula and of UCLA Pascal. The next portions of this paper introduce the languages, describe the criteria upon which they were examined, give criterion-by-criterion results of the examination, and finally draw the conclusion that the selection of Modula as the KSOS implementation language is well justified at this time.

2. CANDIDATE LANGUAGES

Modula and UCLA Pascal have a common ancestor: Pascal [2,3]. The chief aim of the Pascal authors was to develop a language for the teaching of programming as a systematic discipline. For this, they based Pascal upon the principles of

structuring and the expression form of Algol 60 [4]. They replaced Algol 60's declaration with one of their own devising and incorporated data structuring facilities. The language is widely used and there exist a great number of implementations. Much of "modern" programming language development is based upon Pascal. There is a Pascal Users Group, and a committee working toward ANSI Standard Pascal. Such are the trappings of success.

2.1 Modula

Modula [5] is a later product of the Pascal authors and is largely based upon that earlier language. The intention of the Modula authors was to define a language well suited to the programming of dedicated computer systems, and to this end they have provided facilities in the language for multiprogramming and for the operation of peripheral devices. These facilities are inherently machine-dependent. The language therefore provides a construct, called a module, to encapsulate them (i.e. to restrict their validity or existence to a specific and small section of the program).

Another goal of the Modula authors was that the language should be small and easy to implement.

The Modula implementation used for language evaluation was obtained from the University of York (England). The Modula author is reported to feel that this is at present the best implementation of the language. The compiler obtained runs under UNIX, and was slightly modified by us to run under PWB/UNIX.

2.2 UCLA Pascal

UCLA Pascal [6] is described as a systems programming subset of Pascal. It was developed as an implementation language for the Data Secure Unix project.

Here is a list of differences between UCLA Pascal and standard Pascal in terms of language features [7]. UCLA Pascal has no WITH, no SETS, no variant records, and no real arithmetic functions (although real variables are supported). Standard Pascal I/O (files, etc.) is not supported. Although NEW is not supported, pointers are. UCLA Pascal adds declaration of variables at fixed locations, the passing arrays of various sizes to a single procedure, and extensive bit operators.

UCLA Pascal has been extended by the Systems Development Corporation (SDC) [8] which utilizes it for maintenance and extension of the UCLA Data Security Kernel. The major SDC additions are register variables, yet more bit operators, procedures and functions as formal parameters, multidimensioned arrays, and the C ++ and -- operators.

The language is implemented as a translator from UCLA Pascal to C. (The standard C compiler is then used to complete the compilation.) The version used for evaluation was obtained from SDC. This language is known as Extended UCLA Pascal, which name will be represented in this paper by the acronym "Xlas" (eXtended ucLA paScal).

3. LANGUAGE EVALUATION CRITERIA

What should KSOS be looking for in a programming language? How shall the language selection criteria be chosen? The answers flow directly from the goals of the KSOS project. It is required that the KSOS implementation language contribute in as many ways as possible to the security of the system, to the correctness of implementation, to the ease of coding, to the understandability of the programs, and to the ease of verificaton. The chief criteria for language selection are these:

* The language must exist. Its definition must be precise and there must be a compiler for it which runs on and produces code for the KSOS target machine (PDP 11/70).

* The compiler must be supported in such a way that repairs, extensions, and modifications required for KSOS implementation and maintenance can be made quickly and effectively.

* The compiler should produce code which can run on a bare machine. (After all, KSOS will have to.)

* The object code produced by the compiler should be efficient in its use of machine resources. (I.e., small and fast!)

* The language should allow user-defined types.

* The language should be strongly typed and type-safe.

* The language should provide for encapsulation of objects. This is seen as an aid to correct programming and as an aid to verification.

* Multiprogramming must be supportable.

* The writing of machine dependent pieces of code must be supported. Ideally the machine dependent pieces can be encapsulated.

* The language should be supported by development, testing and debugging tools.

* The language, or a minor restriction of the language, should be verifiable. Ideally, a verification environment for the language would exist.

It is hard not to be misunderstood here. All of these criteria are important in selecting a language for KSOS, but we neither anticipate nor require that any single language meet them all. That would be a utopia; KSOS is being built in the less-than-perfect real world. Also, other valid criteria for language selection could be nominated. We feel that these comprise a fairly complete list of those which both are of primary importance to KSOS and have the power to illuminate the differences between Modula and Xlas.

4. LANGUAGE COMPARISON

This section of the paper contains a criterion-by-criterion comparison of York Modula and Xlas. The criteria used are those developed in the previous section.

4.1 Language Definition and Compiler Quality

4.1.1 Modula The definition of Modula is contained in one of three Modula-related articles [5,9,10] published in 1977 in a single issue of Software -- Practice and Experience. (The other two articles are "The Use of Modula" and "Design and Implementation of Modula"). The definition is complete and precise. The introduction of various language constructs is motivated and examples are used to illustrate their intended use. The syntax of the language is precisely defined using a modified Backus-Naur formalism. Syntax diagrams are included.

The University of York Modula Compiler [11,12] was introduced in Section 2.1, above. It implements the language defined in [5] except that:

i. All names must be declared before use,

ii. The VALUE statement for presetting global variables has not been implemented, and

iii. A maximum recursion depth may be declared in a procedure heading. If this depth is omitted the procedure is considered to be non-recursive.

The compiler, which is written in BCPL, is implemented as four passes. Pass 1 is a recursive descent parser with error recovery. This is straightfoward, as the Modula grammar can be deterministically parsed top-down using a single look-ahead symbol. Pass 2 performs semantic analysis. The prohibition against use of names before their delcaration allows the symbol and type tables to be implemented as stacks. This pass implements Modula's visibility rules. A reverse Polish representation of the code is output.

Pass 3 is the code generator. It operated by simulating the operations of an abstract stack machine (called the "M-code Machine" [13]). The pass generates a representation similar to PDP-11 machine code. Pass 4 is a peephole optimizer. It outputs a representation suitable for assembly with the UNIX assembler.

4.1.2 Extended UCLA Pascal The definition of Xlas is contained in a Reference Manual [8], available from SDC. This manual is self standing in its description of the Xlas subset. The definition of language features is concise. Examples are offered [14]. The argument that Xlas is a useful or necessary language is not made. The language as defined appears to have no existence in the absence of its compiler.

The compiler for Xlas was introduced in Section 2.2. It implements exactly the language defined in the Xlas Reference Manual. In fact, the Manual contains, in an Appendix, the LALR(1) grammar used by the Xlas compiler.

The compiler is referred to as a translator. It is a single pass yacc-based parser which directly generates C language productions. These productions are then compiled by the standard UNIX C compiler.

An unfortunate aspect of this arrangement is that the use of the C intermediate representation is not effectively hidden from the programmer. The translation-compilation of an Xlas program can result in error messages from the C compiler. These occur as the "result of using an unimplemented feature of Xlas" or as the result of "a syntax or semantic error in the [Xlas] module that the translator was unable to detect" [14].

4.1.3 Comparison Modula's definition is neater than that of Xlas. The definition of Modula is published in the public domain; the definition of Xlas is not. Modula is the product of a mature and widely respected language designer, the father of Pascal. Modula is a later product than Pascal. Modula is motivated as a language, Xlas is apparently not. The Modula compiler does a better job than the Xlas compiler of protecting the language user from the mysteries and mechanisms of the compilation process. We therefore conclude that Modula is superior to Xlas when judged by the critera of language definition and compiler quality.

4.2 Compiler Support

4.2.1 Modula The development and support of the University of York Modula compiler is funded by the British Government's Science Research Council (SRC). Current funding for Modula support takes the form of a three year grant, which began in March 1978, at the rate of one man-year/year. In addition to this SRC funded support York itself provides support at approximately 1/3 man-year/year.

York actively advertises and distributes its Modula compiler. Purchasers of the compiler are obliged by their purchase agreement to report compiler bugs (and locally generated fixes) to York for central maintenance. The present version of the compiler is Version 2.

We have established a local mailbox for York and communicate with them via the Arpanet's London TIP. Using this channel we have been exploring with them the desirability and costs of extensions to Modula. They have readily agreed to implement specific proposed extensions [16,17] when, in their opinion, the extensions are of general utility to the Modula community. In these cases the costs will be covered by the SRC grant.

In the course of our communications with York we became aware that the compiler was written by a Research Fellow named Jeremy Holden. Holden is an honors graduate of York's Computer Science Department -- and is, at this time, looking to change jobs. We would very much like to hire him for the purpose of maintaining, extending, and supporting his Modula compiler as well as for other language development work. This would put us in a very strong KSOS language support position -- we would be independent of a language supplier.

Holden has been approached on this matter and is willing (in fact, keen) to join us and to do the work outlined above. He will be available at the beginning of February, 1979. There is however a difficulty. Holden is a UK citizen, he has no US visa. How can we get him one in time?

4.2.2 Extended UCLA Pascal The UCLA Pascal translator and its SDC extensions were produced with public funding. There are, however, no funds at the moment for for Xlas support.

SDC, nevertheless, has expressed a willingness to provide limited support to the extent that they are willing to "look at" reported bugs without any guarantee of service.

FACC employs personnel with sufficient yacc experience that Xlas support could be undertaken in-house. With KSOS' requirements in mind it seems that this would be preferable to SDC's limited support. The resulting language could be called "FACC's Extended SDC Extended UCLA PASCAL!" (Readers are invited to suggest acronyms).

4.2.3 Comparison Modula has an independently funded support group, Xlas does not. The Modula support group is willing to make language extensions and to fix bugs. SDC is only willing to study bugs. The support of either language could be undertaken by FACC (and transferred to the KSOS maintainence contractor.) In the case of Modula this would involve overcoming the visa problem and hiring the Modula implementor. In the case of Xlas an existing or additional employee with appropriate background would be assigned to the task.

We conclude from the above that York Modula is better supported and more easily supportable than Xlas.

4.3 Support of Bare Machine

4.3.1 Modula A prime goal of the Modula language is to support the programming of dedicated applications on small machines. The very nature of this class of program is that it runs upon bare machines. Any Modula compiler which conforms with the Modula Report must provide this facility.

The run time system of York Modula is contained in a nucleus which provides all code necessary for process scheduling, context switching, and send and wait operations. The size of the nucleus is about 150 PDP-11 words. (cf. 4K words for Concurrent Pascal). This nucleus is all that is required to run a Modula program on a bare machine.

In addition to the bare machine nucleus there is an alternate nucleus which allows (sequential) Modula programs to be run under UNIX. Selection between bare machine and UNIX environments is controlled by a compiler parameter. Both nucleuses (?nuclei) are written in assembler.

4.3.2 Extended UCLA Pascal The run-time environment required by Xlas programs is exactly that required by C programs (i.e. UNIX). The routines of the standard C library are used for function implementation. Operating system demands are made in terms of UNIX system calls.

No specific provision is made for generation or support of Xlas programs on a bare machine. However several programs written in Xlas, notably UCLA Data Secure Unix, do run on a bare machine. This indicates that the feat is possible, but not how it is accomplished.

4.3.3 Comparison Modula makes explicit provision for bare machine target environments; Xlas does not. A Modula run time nucleus exists. Its size is known, and it is small. An Xlas run time does noe apparently exist.

We conclude that Modula is clearly superior to Xlas in its support of programming for bare machines.

4.4 Efficiency of Object Code

An experiment was run in order to determine the relative efficiency of the object code produced by Modula and Extended UCLA Pascal. A program for ordered insertion in a doubly-linked list was specified and programmed in both languages. The object code was examined. Table 1 gives a summary of the experiment in terms of statement count.

Case	Statement Count		
	Entry	Procedure Body	Exit
Modula :no pointers :WITH statements	0	73	2
Modula :no pointers :no WITH stmts	0	89	2
Extd UCLA Pascal :pointers :register vars	7	47	8
Extd UCLA Pascal :no pointers :register vars	7	68	8
Extd UCLA Pascal :no pointers :no register vars	7	95	8

Table 1. Object Code Comparison.

4.4.1 Modula Modula procedure call, entry and exit are particularly efficient and are done in-line. This efficiency is welcome in view of the heavy use of procedures encouraged by successively decomposed design and by functional and data abstraction as used in KSOS.

The procedure body code produced by Modula was inspected. It was determined that the code size and efficiency could be improved by optimization of the use of registers in holding temporaries. Post-processing of the Modula object code

with an available optimizer failed to make any improvement.

Two cases were tried. It is clear from the table that the use of the WITH construct leads to better object code.

4.4.2 Extended UCLA Pascal Three cases were tried. It is clear from the table that SDC's added features were well chosen in that their use contributes significantly to reduction in procedure body size. They do not effect procedure call, entry and exit code.

4.4.3 Comparison Modula is four to eight times more efficient than Xlas in procedure call entry and exit. Xlas, with pointers and registers, produces much better procedure body code than Modula. Xlas used without these features produces larger and less efficient code than Modula

It must be noted that the use of pointers in KSOS programming must be ruled out upon verifiability grounds. Under this restriction registers are still useful for holding iteration variables.

Given the above information, and in the absence of timings and KSOS characteristic measurements of dynamic procedure call counts, we conclude that Modula and Xlas are roughly equivalent with respect to the efficiency of their object code.

4.5 User-Defined Types

The languages under study provide similar mechanisms for user type definition. They do however differ in type encapsulation and type safety, as will be detailed below.

4.6 Type Safety

4.6.1 Modula The type equivalence rule of York Modula is very regular. It is the "Structural Type Equivalence" rule defined in [18]. Informally, this is that two variables are considered to be of the same type whenever they have components of the same type structured in the same way.

Every constant, variable and expression has one and only one type. The type signatures of built-in operators are defined, and the types of their operands must be equivalent. Similarly, the types of actual parameters must be equivalent to the types of their corresponding formals.

To the best of our knowledge the York Modula Compiler enforces these rules exactly.

4.6.2 Extended UCLA Pascal The type equivalence rule of Xlas is not explicitly defined. One assumes from reading the Xlas Reference Manual that the type equivalence rule of Pascal applies.

Unfortunately, the implementation of Xlas fails to provide any reasonable type safety. The following program is processed by the Xlas translator without any complaint.

```
procedure typeerr;          { type error demonstration }
type
   day = (monday,tuesday,wednesday,thursday,friday);

var
   c: char;
   i: integer;
   b: boolean;
   s: record
      s1: char;
      s2: array [1..10] of boolean;
      s3: day;
      end;
   d: day;
begin
    i := c;     {oops..}
    c := b;     {ugh!}
    b := i;     {eeek}
    s := b;     {yeccchch}
    d := s^;    {yikes!?!}
end;
```

The C compilation phase for this program produces three error messages.

4.6.3 Comparison The Modula type rule is explicitly stated; the Xlas type rule is not. The Modula type rule is enforced; the Xlas type rule is not. We conclude that York Modula is far, far more type safe than Xlas.

4.7 Encapsulation

4.7.1 Modula

The Modula module was introduced in Section 2.1. Its use is quite general, not restricted in any way to machine-dependent portions of the program. A Modula module is a collection of constant, type, variable and procedure declarations. These are called the objects of the module. They come into existence when control enters the procedure to which the module is local. The module may be thought of as a fence around its objects. An essential property of the module construct is that one may precisely determine the fence's transparency. This is accomplished by two lists of identifiers associated with the module. The define list names all module objects which are visible (exported) outside the module. The use list names all objects declared outside the module which are to be made visible inside (imported). Modules may be nested.

Such fine-grained and explicit control over the scope of object names is in harmony with the methods of program specification, construction, and verification being used for KSOS. It is worthwhile noting here that the Euclid language includes a module construct similar to the Modula module.

4.7.2 Extended UCLA Pascal Xlas does not provide encapsulation beyond that normally associated with procedures. There is a "module" construct in the language, but its effect is that of a global break in scope (like that

associated with separate compilation) rather than that of a semipermiable fence. Xlas does not allow modules to be nested. Incidentally, it does not allow procedures to be nested either!

4.7.3 Comparison Modula modules provide selective import and export and may be nested. Xlas modules provide total break of scope and may not be nested. From the point of view of encapsulation, Modula is clearly superior to Xlas.

4.8 Multiprogramming

4.8.1 Modula Multiprogramming is built-in to Modula. Concurrency is achieved using processes, which are conceived of as executing in parallel with one another. Multiple activations of the same process are allowed. The storage requirements of a process can always be predicted. Synchronization is achieved by the use of signal variables, upon which the operations send, wait and awaited are defined. Interface modules can be used to ensure mutual exclusion when several processes share a common variable. Modula signals are more primitive than semaphores; it has been shown that they can be used to implement semaphores.

The Modula nucleus performs process scheduling and context switch. These facilities, and the process construct which they maintain, are exactly what are needed for the most primitive version of KSOS process and correspond to the lowest levels of the KSOS Kernel implementation decomposition.

4.8.2 Xlas Multiprogramming is not directly suported by Xlas. In a UNIX environment support could be trivially achieved via UNIX system calls. Providing multiprogramming support in a bare machine environment would be a more complicated matter, involving creation (from scratch) of a suitable nucleus.

4.8.3 Comparison Support of multiprogramming is an explicitly stated goal of Modula. It is not included in the goals of Xlas (nor in those of Pascal). The Modula compiler provides facilities to support multiprogramming; the Xlas compiler does not. The Modula facilities are identically those required at the lowest level of KSOS. We conclude that, in the matter of multiprogramming, Modula is the superior language.

4.9 Machine Dependent Code

4.9.1 Modula Modula supports machine dependent code via the device module construct. This provides a closed scope wherein machine dependent language extensions are made available. These language extensions fall into two categories.

i. Association of variables with fixed addresses at declaration. On the PDP-11, for example, this facility gives access to (and encapsulates) the device registers.

ii. A means to receive and process interrupts. This is provided for by device processes and by the doio statement.

Modula provides no means for linking to separately compiled (assembled) programs.

4.9.2 Extended UCLA Pascal Xlas provides a facility for the binding of variables to addresses. It does not include any facility for interrupt handling. However, it is possible to link an Xlas program to programs written in C or in assembler. Clearly, this escape mechanism allows most anything in the way of machine dependent programming to be accomplished.

4.9.3 Comparison Machine dependent facilities are built-in to Modula; they are built-on (or outside of) Xlas. Both allow the use of fixed addresses within the language. Only Modula provides language-based interrupt handling facilities. Xlas allows escape to external routines; Modula does not. Modula localizes machine dependent code; Xlas does not. We conclude that, on balance, Modula provides a fuller and safer set of machine dependent facilities.

4.10 Test and Debug Tools

4.10.1 Modula Modula provides no debugging facilities at all aside from its halt procedure. No facilities are provided for separate compilation (the concept is said to be antithetical to the intended style of programming in the language.)

4.10.2 Extended UCLA Pascal A great many utilities and debugging tools have been developed for use in conjunction with Xlas. Separate compilation is supported (with some restriction). There is a load map making utility. A preprocessor has been adopted from the C preprocessor. A system module processing package allows any number of separate Pascal and C source files to be translated and linked together to build a system.

In addition to these utilities, all the UNIX-supported C debugging aids are applicable to Xlas programs as a consequence of the language's use of C as an intermediate representation. When using these tools, however, one does not debug in terms of Xlas. Instead, one debugs in terms of C constructs and C coordinates.

4.10.3 Comparison Some aid is preferable to none. When it comes to utilities and debugging aids, Xlas is far superior to Modula.

4.11 Verifiability

The verifiability of Modula and of Pascal (N.B. Pascal, not Xlas) was studied on our behalf by SRI International. They were asked, for each language, to recommend programming restrictions which would be necessary to ensure the verifiability of the resulting program. Their report is included as Appendix A to this paper. The discussion in the next two subsections is offered as an addition to the discussion in the Appendix.

4.11.1 Modula The scope limitations and type encapsulation provided by the Modula module are a great aid to verification. So too is the strict type safety afforded by the York compiler. Additional Modula aids to verifiability are the localization of machine dependent code in device modules and the absence of any escapes into separately compiled 'foreign'-language routines.

The programming restrictions recommended in Appendix A for Modula are minor. They are easily enforced, perhaps with a preprocessor. The language

remaining after the restrictions are applied is still quite useful for systems programming.

A verification environment for Modula programs is being constructed at SRI International. This environment includes (and is therefore compatable with) the specification and design proof methodologies being used for KSOS.

4.11.2 Extended UCLA Pascal The lack of effective encapsulation and the prohibition against procedure nesting result in large implicit interfaces. In consequence, Xlas programs are difficult to verify. The absence of safe type checking and the escapes to separately compiled routines compound the problem.

The programming restrictions recommended in Appendix A for Pascal are serious. Those for Xlas would cut deeper. The problems are hard, and the adequacy of the remaining language for disciplined, constructive systems programming is in doubt.

A verification environment for Pascal exists as the XIVUS system at ISI [19]. This system has been used to verify several procedures from the UCLA Data Secure UNIX Kernel. (It is not clear exactly what these procedures were verified against. This particular kernel was implemented before its specifications were written). The alegbraic form of specification used by XIVUS is not compatible with the form used for KSOS development, although there may exist a mapping from one form to the other. The XIVUS Pascal proof rules would have to be modified to encompass the SDC alterations of the language.

4.11.3 Comparison Modula has constructs which lend themselves to verification; Xlas does not. The problems of enforcing verifiability restrictions on Modula are easy; enforcing these restrictions on Xlas is hard. Restricted Modula is a useful language; restricted Xlas is less useful. Neither language has a verification environment, both are close to having one. The verification environment of Modula will be comparable with KSOS development methodology, that of Xlas will not. We conclude that Modula lends itself to verification much more easily than does Xlas.

5. LANGUAGE SELECTION

A summary of the evaluation made in Section 4 is presented as a qualitative box score in Table 2. For each criterion, the language judged superior is indicated by "+"; the language judged inferior is indicated by "-". Equal judgement is indicated by "=" for both languages. No attempt has been made to be more quantitative than this as no objective means of quantification exist.

Qualitative scoring: + superior - inferior = equivalent	Modula v	Extended UCLA Pascal v
4.1 Language Def. and Compiler Quality	+	-
4.2 Compiler Support	+	-
4.3 Bare Machine	+	-
4.4 Object Code Efficiency	=	=
4.5 User Defined Types	=	=
4.6 Type Safety	+	-
4.7 Encapsulation	+	-
4.8 Multiprogramming	+	-
4.9 Machine Dependent Code	+	-
4.10 Test and Debug Tools	-	+
4.11 Verifiability	+	-

Table 2. Criterion-by Criterion Qualitative Evaluation Box Score

Despite the qualitative nature of the scoring the outcome is quite clear. York Modula is a more desirable KSOS implementation language than Extended UCLA Pascal. The two are not even close.

This conclusion confirms the guidance offered by the by the KSOS language selection procedure. Further, it offers strong support to FACC's choice of Modula as the KSOS implementation language.

FACC's plans for Modula are these:

i. Develop in-house Modula compiler competence. Employ Jeremy Holden, if possible. Provide close support to KSOS implementators. Make KSOS-specific language extensions and alterations

ii. Encourage and guide the University of York in making language extensions which are of general use.

iii. Define and develop appropriate test and debug tools.

iv. Define verifiability restrictions. Develop a preprocessor to enforce them.

v. Cooperate with SRI International in defining, implementing, and testing a Modula verification environment.

vi. Investigate the cost/benefit of applying global optimization within the Modula compiler.

6. ACKNOWLEDGEMENTS

The information presented in this report was developed and evaluated as a joint effort by members of the FACC KSOS Project. Significant contributions were made by John Nagel, Ken Biba, Luke Dion, Jon Forest and Rance DeLong.

7. REFERENCES

[1] "KSOS Verification Plan", WDL-TR7809, Ford Aerospace and Communications Corp., Palo Alto, CA (March 1978).

[2] N. Wirth, "The Programming Language Pascal", Acta Informatica, Vol. 1, pp. 35-63 (1971).

[3] K. Jensen and N. Wirth, Pascal Users Manual and Report, 2nd ed., Springer-Verlag, New York (1974).

[4] Algol-60 Report ++

[5] N. Wirth, "Modula: a Language for Modular Multiprogramming", Software -- Practice and Experience, Vol. 7, pp. 3-35 (1977).

[6] E. Walton, "UCLA Pascal Translation System" ++

[7] G. Popek, "SDC Improvements to UCLA Pascal Translator", ARPAnet message, (10 October 1978).

[8] D. Clemans, "Pascal/Cee Translator Reference Manual", Systems Development Corp., Santa Monica, CA (?).

[9] N. Wirth, "The Use of Modula", Software — Practice and Experience, Vol. 7, pp. 37-65 (1977).

[10] N. Wirth, "Design and Implementation of Modula", Software -- Practice and Experience, Vol. 7, pp. 67-84 (1977).

[11] I.D. Cottam, "Functional Specification of the Modula Compiler", YCS-13, University of York, Dept. of Computer Science (1978).

[12] J. Holden and I.C. Wand, "An Assessment of Modula", YCS-16, University of York, Dept. of Computer Science (1978).

[13] I.C. Wand and J. Holden, "MCODE", YCS-14, University of York, Dept. of Computer Science (1978).

[14] D. Clemans, "Pascal/Cee Translator Users Guide", Systems Development Corp., Santa Monica, CA (?).

[15] D. Clemans, "Pascal/Cee Translator Utility Manual", Systems Development Corp., Santa Monica, CA (?).

[16] K.J. Biba, "Some Comments on the Use of Modula as the KSOS Kernel Implementation Language", STD-KDN-6, Ford Aerospace and Communications Corp., Palo Alto, CA (September 1978).

[17] I.C. Wand and I.D. Cottam, "Re: STD-KDN-6", letter, University of York (November 1978).

[18] J. Welsh, W.J. Sneeringer and C.A.R. Hoare, "Ambiguities and Insecurities in Pascal", Software -- Practice and Experience, Vol. 7, pp. 685-696 (1977).

[19] "1977 Annual Technical Report", ISI/SR-77-8, USC/Information Sciences Institute, Marina delRey, CA (1977).

8. APPENDIX A

Recommendations for Use of Pascal and Modula

R. J. Feiertag and P. G. Neumann
SRI International

This memo gives some recommendations on the use of the programming languages PASCAL and MODULA as implementation languages for KSOS. The recommendations are directed toward making programs written in the languages easier to verify in a formal manner. However, most of the recommendations also serve to make programs more easily understandable. The definitions for the languages examined are the original published definitions (see [1] for PASCAL and [2] for MODULA). Modifications to these languages and issues relating to particular implementations of these languages are not addressed in this memo.

Recommendations common to both languages are given first.

- Both PASCAL and MODULA permit aliasing of variable names on procedure of function calls. In KSOS, such aliasing should never be necessary. Although rules to eliminate aliasing can be quite complex in order to retain as much of the power of the language as possible, much of the power of the language is unnecessary for KSOS and so some fairly simple rules should suffice.

  1. Use "var" parameters only when absolutely necessary. The use of "const" parameters does not lead to aliasing.

  2. Do not pass different elements of the same variable to the same procedure as separate "var" arguments or global variables.

- It is necessary when using the Floyd method of verifying programs to include assertions at strategic places in a program. Neither PASCAL nor MODULA have a special construct to meet this purpose. One means to introduce such a statement with no change to the compiler is to reserve certain comments for this purpose. For example the statement

  {assert ...}

  in PASCAL or

  (*assert ...*)

  in MODULA to identify assertions to the verifier. These statements would, of course, be treated as comments by the compiler.

- In order for the implementation language to be verified with respect to specifications written in SPECIAL it is necessary to identify those cases in the program that correspond to exceptions in the specification. This can be done by means of a very simple convention.

  Reserve the first or last parameter to each visible function as the exception parameter (it must be a "var" parameter). The value of this argument on return is the exception value (0 means no exception). Assignments to this parameter followed by a return are interpreted as the raising

of an exception.

- The "with" statement also causes aliasing. Although most cases of the use of the "with" statement can be properly handled by an automated verifier, it can easily lead to misinterpretation by a human reader. The "with" statement should either be eliminated or aliasing should not be permitted, i.e., within a "with" statement the fully qualified form of reference should not be allowed.

The following are recommendations for PASCAL only.

- The use of "goto" statements and labels can lead to very complex control paths and consequently very difficult formal verification. The use of "goto" statements should either be eliminated or only forward transfer should be permitted.

- Variant records cause problems in PASCAL because they are not type-safe. All references or assignments to components of variant records should be embedded within case statements that discriminate on the tag of the variant record. And references and assignments to particular components should only be allowed in those portions of the case statement for the value of the tag corresponding to the component.

- PASCAL contains an allocate statement but no means for freeing storage that has been allocated. It is likely that most implementations would provide some kind of "freeing" mechanism. "Freeing" can lead to dangling pointers (i.e., pointers that point to nonexistent data). It is essential that freeing occur in such a way as to guarantee that dangling pointers do not occur. Any invocation of the "free" mechanism must be encapsulated together with code that places the null value in all pointers to the freed object. This encapsulation should assert that all pointers pointing to the freed object have been assigned the null value.

- Pascal contains no mechanisms for controlling or coordinating parallel programs. Such mechanisms will PASCAL contains no mechanisms for controlling or be necessary for the implementation of KSOS. It is recommended that some simple and widely used mechanisms be used. Although the verification of parallel programs is still a difficult problem a simple mechanism such as binary semaphores, wait and notify, or monitors is more likely to be easily verifiable than some more complex mechanism. In any case, more complex mechanisms should be unnecessary for KSOS.

The following recommendations apply to MODULA only.

- In MODULA, device modules can contain implementation dependent code. The verifier will not contain semantics for the device dependent code and will not be able to verify the correctness of such device modules. In order that the verifier can verify programs that call procedures within device modules, the device module procedures should contain assertions that describe their function.

- In MODULA the semantics of lexically nested interface modules is unclear. There is no need for lexically nested interface modules in KSOS and so

lexically nested interface modules should not be used.

- The semantics of the "signal" statement is also unclear. Specifically it is not clear if assertions made before the execution of a signal statement are valid after execution of the signal statement. This problem can be bypassed by requiring the signal statement to be the last executable statement of a procedure. This requirement renders the problem irrelevant.

- Modula permits use-lists to be attached to each procedure in order to declare all the global variables imported by the procedure. Rather than being optional, use-lists should be required on all procedures. This requirement makes importation of global variables explicit to the reader and makes the verifiers job a little easier.

Of course, no collection of rules can guarantee an easily verifiable program. The ease of formally verifying a program depends upon the good sense of the programmer. In general, anything that improves the clarity of the program will make it easier to verify. In a language such as PASCAL which has no encapsulation facility, it is up to the programmer to organize his procedures in a manner akin to encapsulation. This is in keeping with the methodology and with recent programming language development. Good sense on the part of the programmer will go farther toward promoting easy verification than any language design or programming style rules.

REFERENCES FOR APPENDIX A

[1] N. Wirth, "The Programming Language Pascal", Acta Informatica, Vol. 1, pp. 35-63 (1971).

[2] N. Wirth, "Modula: a Language for Modular Multiprogramming", Software Practice and Experience, Vol. 7, pp. 3-35 (1977).

APPENDIX B

Security Requirements for the KSOS Development Facility (KDF)

(This appendix is Chapter VII in the FACC/WDL Security Standard Procedure covering Automatic Data Processing Systems, which has been formally approved by the WDL Government Security.)

SECTION VII

SECURITY REQUIREMENTS

FOR THE

KSOS DEVELOPMENT FACILITY (KDF)

SECTION VII - SECURITY REQUIREMENTS FOR THE KSOS DEVELOPMENT FACILITY (KDF)

A. GENERAL

1. This section establishes the security measures for the protection of classified information generated, stored, processed, or used in the automatic data processing (ADP) system located in the KSOS Development Facility (KDF) located in Room 414 of building 1 of the WDL facility. The system consists of a government furnished Digital Equipment Corporation (DEC) PDP-11/70 computer system.

2. The specific security requirements under which this system will operate are delineated herein together with the administrative, physical, and personnel security measures implemented to protect the classified material and information used or processed in the KDF. The security requirements of Section I of this manual, which are also pertinent to this system, are applicable.

3. ADP System Security Supervisor and Custodian. An ADP system security supervisor and custodian have been appointed for the KDF (see Appendix A, Section VII). These individuals are responsible to the facility security supervisor for implementation of security procedures and practices described in this section and in Section I of this manual.

4. The KSOS Development Facility will operate in a Dedicated security mode. The KSOS Development Facility operates in support of the DoD Kernelized Secure Operating System (KSOS) Program. The highest level of classified information processed by the system is SECRET. Approximately five (5) percent of work to be performed is classified. The hardware is unclassified.

B. ACCESS CONTROLS

1. Access to the KDF System

a. All personnel accessing the KDF system during classified operations will have the appropriate clearance, need-to-know, and briefings in accordance with Section I of this manual. All personnel having unescorted access to the KDF system during unclassified operations will have at least a CONFIDENTIAL clearance.

(1) The highest level of classification which may be processed on the system is SECRET. All personnel accessing the system or area during classified operations shall hold at least a SECRET clearance. The ADP System Security Custodian will ensure that all personnel are briefed as to additional safeguards required for No Foreign Dissemination or to other additional restrictions as required.

(2) Only terminals in the immediate area of the KDF will be enabled during classified operations. All remote terminals will be disconnected. All external communication lines from the KDF will be disconnected during classified operations.

(3) The system will be accessed only by authorized personnel:

(a) Ford Aerospace and Communications Corporation (FACC) employes assigned to the KSOS project. All such employes will be cleared at least to the SECRET level.

(b) Subcontractor personnel from SRI International assigned to the KSOS project. All SRI personnel will be under the supervision of FACC employes in accordance with established visitor control procedures. All such SRI personnel will be cleared at least to the SECRET level.

(c) Other FACC personnel not assigned to the KSOS project will be allowed in the area only under escort. Unescorted, uncleared personnel will not be allowed in the area at any time.

(d) Vendor personnel (see below) under the supervision of FACC personnel in accordance with established visitor control procedures.

(e) A list of authorized personnel will be maintained by the KDF System Security Custodian and posted by the entrance.

b. Vendor personnel will NOT be permitted access to the system during classified operations. Present vendors to FACC for the KDF system are:

Digital Equipment Corporation
Ann Arbor Terminals, Inc.
Hewlett-Packard, Inc.
Omron, Inc.
Delta Data, Inc.

c. Access to entrance combinations and unescorted entry to the KDF will be controlled and limited to employes who are appropriately cleared and assigned to the KSOS project. Unescorted privileges will be approved only after the individual has been briefed as to his/her security responsibilities. Verification of clearance and need-to-know will be the responsibility of the ADP System Security Custodian or Shift Controller on duty whenever an individual enters.

d. The KDF System will be running a version of the PWB/UNIX operating system when operating in classified mode.

(1) PWB/UNIX does not normally assign a security classification to files. To identify the security of a file, classified files will be placed only under directories whose names identify the classification of the file, for example:

/u/ejm/cva.UNC/file_1
/u/ejm/cva.SECRET/file_2

It should be considered that these are only user aids. All files on the system during classified operations should be considered to be classified at the highest level on the system ("system high"). All output will be considered to be at the system high.

(2) PWB/UNIX has a user-controllable password scheme. All users in the classified mode will have passwords. The UNIX O/S encrypts the password file. Users will be required to change their passwords every three months, upon deletion of any authorized user who had knowledge of the password, or upon suspected compromise of the password. All users given accounts on the classified mode system will be cleared system high. Incorrect passwords cause the PWB/UNI system to abort the log-in process. Passwords are not echoed during login.

(3) During file creation, each user will assign his/her file a specific file protect code to establish read, write, extend and delete permission for his/herself and other authorized users. In most cases, only the file owner and priviledged users will be able to alter a file. All other specified users will have read capability only.

e. Core memory is divided into partitions. When a user program is loaded into a partition, the operating system utilizes virtual addressing to determine the core limits available to that user. Any attempt on the part of the user to address memory outside of these limits will result in an abort.

SEC:REQ:S7:f:pt/146

f. The KDF system will be declassified prior to leaving it unattended. (See paragraph D of this section for procedures.) The KDF system will be protected against tampering by requiring a visual walkaround inspection prior to start-up in classified mode.

g. Privileged Users. PWB/UNIX includes one user, "root", who is the "owner" of the operating system and many utilities. The user root" has broad powers to allow for on-line system maintenance. The "root" password will be restricted to only personnel cleared to the system high security level. An absolute minimum number of individuals will be allowed to operate as "root". Only personnel directly engaged in system maintenance will be permitted to operate as "root". The "root" password for the classified system will be protected as SECRET information.

h. Maintenance. Maintenance of any equipment during classified operations must be authorized by the ADP System Security Custodian/Shift Controller. The system shall be tested by appropriately cleared and specifically designated personnel to assure that the security features are effective. Audit trail records of these transactions shall be maintained. Routine maintenance of any equipment processing classified data will not be permitted.

2. Physical Controls

a. The KSOS Development Facility (KDF) is located in building 1, room 414. The KDF is enclosed within an approved Restricted Area. Access into the area is via two doors equipped with cypher locks. The cypher lock combination is issued to only those authorized, cleared personnel assigned to the KDF.

Building 1 access is controlled by a guard. After hours, the building is protected by a proprietary alarm system (intrusion and fire) monitored in the Plant Protection Office, building 3. Average response time is five (5) minutes.

b. A Restricted Area will be established for all classified operations.

(1) All entrances will be posted with: "Restricted Area -- Authority DoD Manual 5220.22M."

(2) All personnel entrances will be secured with a push-button combination lock.

(3) The Restricted Area will remain in effect until the KDF System is declassified.

c. All classified operations will take place in the Restricted Area. No remote terminals or communication lines will be enabled during a classified mode. There will be no classified transmission outside of the area.

d. All equipment processing classified data will be posted with conspicuous classification signs indicating the level of classification involved.

e. A large conspicuous sign will be posted adjacent to the equipment. The sign will indicate that classified operations are in progress and the level of the classification involved.

f. During classified operations, the system will be attended by a cleared employee at all times.

g. Whenever the system is declassified but unattended, the system hardware integrity will be protected by seals and visual walkaround inspections prior to classified mode startup. In addition, the entrances to the area will remain locked.

C. The KSOS Development Facility System

1. Digital Equipment Corporation PDP-11/70 System

a. CPU
b. 128 WOS of Memory
c. 2 ea-RP06 Disk Drives and controller (176 Mbytes each)
d. RS04 Fixed Head Disk (1 Mbyte)
e. RK05J Removable Cartridge Disk (2.4 Mbyte)
f. LP11 -WA High speed printer
g. TWE 16-EA Magnetic tape
h. TME 16-FA Magnetic tape
i. TC11-GA DEC tape
j. PC11 Paper tape reader punch
k. DH11 Communications Multiplexer
l. DL11 Asynchronous Line Adapter
m. DN11 Bell 801 System Unit
n. KW11P Realtime Clock
o. FP11C Floating Point Unit
p. KMX11 Auxilliary Communications Processor

2. Terminals

a. Digital Equipment Corp. - LA36 DEC writer II
b. Hewlett Packard - HP2645A
c. Delta Data Systems - G 4000L
d. Ann Arbor Terminals - 204027E
e. Omron - 8025A (2 ea)
f. Anderson Jacobson - AJ 832

SEC:REQ:S7:f:pt/146

3. Storage Media

Medium	Highest Classification
RP06P Disk Pack	SECRET
RK05P Disk Cartridge	SECRET
DEC Tape	SECRET
Mag. Tape	SECRET

4. Keypunch

None

5. Channel Assignment. The PDP-11/70 does not utlize I/O channels. All I/O is via common memory addresses. The PWB/UNIX operating system controls all physical I/O.

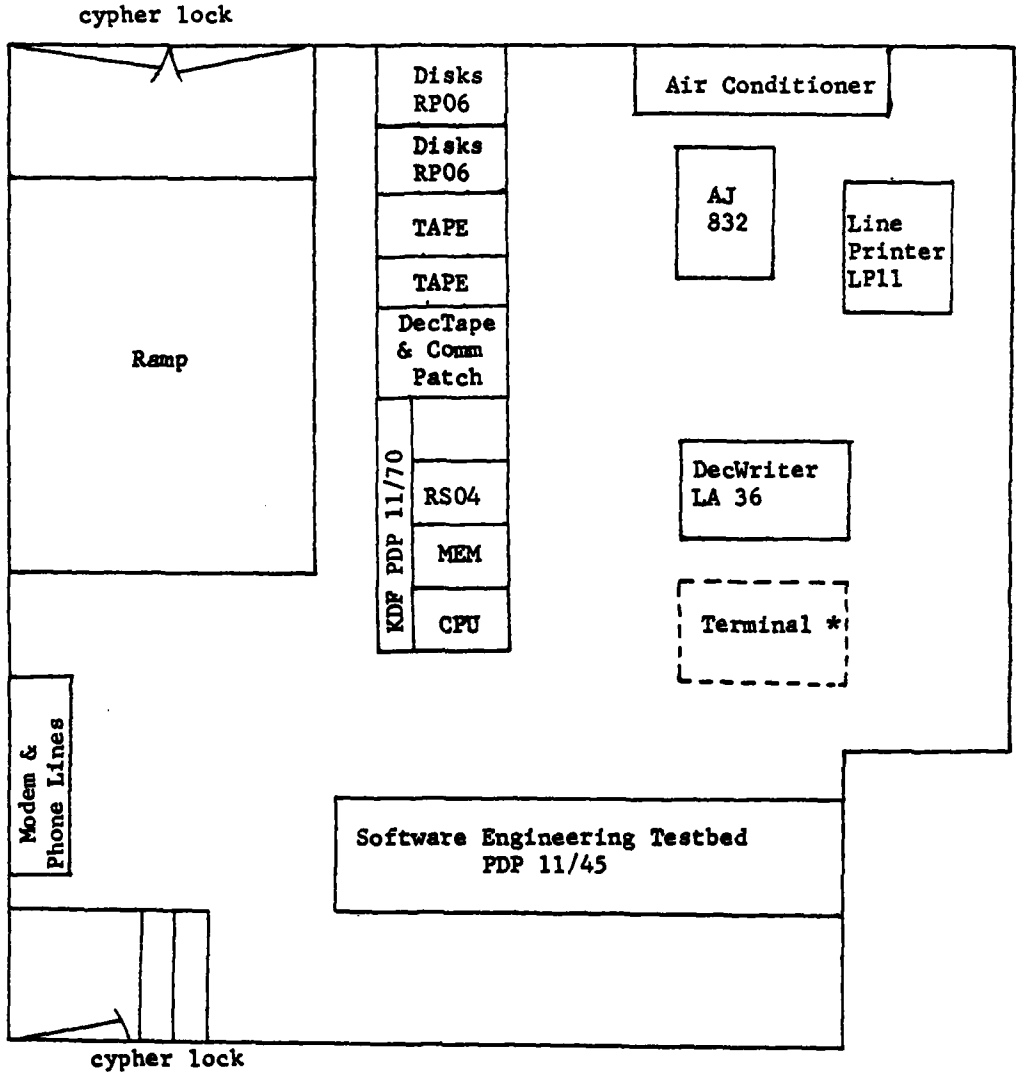
6. Equipment Additions/Deletions. Diagnostic and declassification procedures will be implemented as required, to ensure system security and protection of classified information whenever equipment is added, modified, or removed.

7. Other Systems. Another ADP System is located in the same area. The two systems will not be connected when the KDF ADP System is operating in a classified mode.

8. Teleprocessing. Remote terminals to the KDS system are located in adjacent rooms and in other facilities. These remote terminals and communication lines will be disconnected prior to the beginning of classified mode processing. All connections are via a communications Patch Panel. All patch cords will be removed prior to classified processing. In addition, the comm lines will be disconnected from the modems located in the panel for phone lines. Disconnecting the patch cords and comm lines will physically isolate the PDP 11/70 from all other systems and remote terminals.

9. Physical Layout. See Figure 1

10. System Software Description. The primary system is PWB/UNIX. Local modifications have been made. All such modifications are controlled and are performed by cleared personnel. The purpose of the KDF is to provide a machine on which to develop and test a new operating system called KSOS. KSOS will be a multi-level secure version of UNIX.



* Other terminals listed in Paragraph C.2. which may be relocated inside controlled area for classified operations.

Figure 1. KSOS Development Facility

SEC:REQ:S7:f:pt/146

11. Operating System (O/S)

a. O/S Name and Level Number PWB/UNIX Version 3.0

b. The standard PWB/UNIX operting system is stored on the RP06 disk. To protect software and system integrity, the O/S disks used for classified operations are marked and protected as SECRET material. Backup O/S disks are company proprietary material and are used only for unclassified operations. Emergency use of backup O/S disks for classified operations will be permitted only after the disks and the system have been tested by the System Analyst. Modifications to the O/S used during classified mode will be performed only during classified mode by cleared personnel. The source code and object modules for the O/S will be protected at the system high security level. All output of the system (printer output, terminal output, magnetic tape, DEC tape, and disks) will be treated as being classified at the system high. They will be disposed of or accounted for in accordance with Section I and other pertinent security policies.

The PWB/UNIX can write into any file. Operating System documentation is vendor supplied and is unclassified. This documentation is current and adequate to support the system.

c. Logging Features. PWB/UNIX has a large number of utilities to support logging. The normal directory structure indicates the time of latest modification and of last access to every file in the system. Normally all disk storage is on-line. The classification of files is not present. The KDF will use directory names to indicate the classification of files, for example: /u/eum/CVA SECRET/file 2. **Manual logs will also be used.**

The O/S will log activity by the users. This feature will be enabled, and a printout of the log will be obtained when the system is declassified. The logs will be reviewed weekly by the KDF ADP System Security Custodian for suspicious activity. The log will be known as the account file.

d. Terminal Operation. All terminals within the KDF will be considered to be running classified information whenever any user is operting in a classified mode. Although software techniques exist to logically disconnect terminals, all remote terminals and communictions lines will be physically disconnected prior to commencing classified operations. They shall remain disconnected until the system is declassified.

e. Modification. Temporary or permanent modification of the operating system shall be tested by the System Analyst to assure that the security features are effective. Audit trail records of these transactions shall be maintained. The ADP Systems Security Custodian/Shift Controller will verify that tests are completed.

Testing of the O/S by the operating system analysts will consist of the following:

(a) The O/S disk and/or tape will be loaded into the system.

(b) All system functions will be performed to ensure that the system is operating properly; i.e., there are no anomalies in the operating system.

(c) All available diagnostic procedures will be run to detect possible anomalies in the operating system.

(d) Attempts will be made to circumvent all system security features, such as: can a file be accessed without a password or with an incorrect password?

(e) If no anomalies are detected, the O/S disk or tape may be dedicated for classified processing. The system analyst will label the media as indicated below and then release it to the appropriate DCS custodian for marking and control as a SECRET document.

This media #_____
containing an unclassified operating system has been tested and is hereby certified as being acceptable for classified operations.

_____ _____ _____
(name) (date) (signature)

D. SECURITY PROCEDURES

1. Start-up Procedures The ADP System Security Custodian, or his designated alternate, will prepare the system for processing classified material. Specifically, this will include the following.

   a. Post "Restricted Area" signs on the entrances, and ensure that the cypher locks are secure.

   b. Ensure that only authorized personnel are in the area.

   c. Place "SECRET" signs on each I/O device, and adjacent to the system.

   d. Check the system and area for unauthorized entry and tampering. Inspect cables and equipment for unauthorized additions, modifications, etc.

   e. Disconnect all remote terminals and communications lines Remove all patch cords from comm patch panel and disconnect all comm lines from Modems located in the panel for phone lines. Ensure the PDP 11/70 is disconnected from the adjacent PDP 11/45 system.

   f. Clear all peripheral devices. This consists of removing and storing all removable disks, tapes, and other removeable secondary storage media which are not dedicated for classified opertions, changing from unclassified to classified ribbons on the line printer and hard copy terminals and feeding sufficient blank paper through the printers to gurantee that the UNCLASSIFIED portion is separated (torn off).

   g. Clear all internal memory and circuitry to be used for the storage of classified information on the dedicated operating system. This is accomplished by bootloading from the System Purge tape. This program will clear the RS04 fixed disk, the main memory, and the cache by overwriting all locations. This tape will be safeguarded at the SECRET level to protect its integrity.

   h. Mount the set of disks and tapes dedicated for classified operations.

   i. Bootload the dedicated O/S from the dedicated disk(s) and verify that the system is operational.

   j. Provide a classified waste bag in the area.

   k. Make a final security check of the above and complete the KDF Security Procedures checklist.

2. Operating Procedures

a. The System Security Custodian or his designated alternate will be responsible for monitoring all activities and for ensuring tht all necessary logs are completed.

b. Only authorized personnel are allowed into the restricted area when the system is in a classified mode. All authorized users have a responsibility to assure that unauthorized personnel are denied access to classified data.

c. Each authorized user will be responsible for ensuring that personnel entering the KDF are properly cleared and have a valid need-to-know.

d. Each person entering the KDF is responsible for assuring that the door is resecured (closed, latched, and cypherlock engaged) after his/her entry and exit. Further, they are responsible for assuring that only authorized personnel enter with them.

e. During classified operations, the O/S will maintain a log which will indicate time on/off, inputs and outputs, media used, the nature of the work done, the disposition of any output, and that proper security procedures have been followed. (See sample form.) The log will be known as the account file.

f. Users will be responsible for reporting any trouble with the system to the System Security Custodian, and for filing a system trouble report.

g. Access during non-working hours will be limited to personnel specifically authorized by the ADP System Security Custodian. A specific individual will be appointed the designated alternate for the Security Custodian. This individual will be responsible for compliance with all security procedures.

h. Visitors will be allowed to access the KDF system only in accordance with established visitor control procedures. During classified operations, visitors must sign in as classified visitors and have valid visit authorization cards (WDL 2256) establishing clearance and need-to-know.

i. All classified material used or generated will be marked and safeguarded in accordance with the WDL SPP and Section I of this manual.

(1) CRT terminals will be cleared when not in use and whenever the operator leaves the terminal.

(2) Removeable magnetic media used during classified operations will be marked and controlled as classified acountable documents. Classified magnetic media will remain classified until declassified in accordance with Section I of this manual.

(3) All hard-copy output generated in classified mode will be accounted for in one of three categories:

- o Waste (placed in classified waste bags)
- o Work-in-process (accounted for in the Work-in-Process (WIP) envelopes)
- o Final output (accounted for by the document control system.)

Each user will be responsible for maintaining his/her WIP envelopes, and for assuring that classified waste is handled correctly. The Document Control Station (DCS) custodian responsible for the KDF will enter final documents into the documents control system. The user is responsible for assuring timely delivery of final documents to the DCS custodian. (Normally, all output will be considered WIP until it can be entered into the DCS.)

3. Shut-down Procedures. Upon completion of classified processing, the system will be returned to an UNCLASSIFIED (BLACK) state by the ADP System Security Custodian, or designee, in the following way.

a. Obtain printout of Account File.

b. Declassify core, cache, and fixed disk by executing the System Purge program. The System Purge tape is controlled as a classified document. This program alternates writing all ones and all zeroes throughout memory and cache 100 times. This program will also alternate writing ones, zeroes, and a seven (octal radix) on the fixed disk, so that the disk is overwritten a minimum of three times. The final overwrite will remain on the disk. A printout or dumps of memory, cache, or disk at this point would reflect either all zeros or all sevens (octal radix). Take random samples to ensure that the memory, cache, and disk have been declassified. If equipment malfunction prevents declassification, refer to paragraph E, Emergency Plan.

c. Remove all printed output from the printers. Safeguard as classified material.

d. Remove dedicated operating system disk.

e. Remove printer ribbons from printer and store in approved classified container. If ribbon is worn and will be replaced, place in a locked classified waste container for destruction.

f. Inspect all CRT/keyboards to ensure that CRT's are clear. Inspect screen surface under high intensity internal CRT illumination to detect evidence of burned-in information.

g. Remove all disk packs for storage as classified material. Cleared disks are still to be considered classified material and will be stored in locked classified containers.

h. Remove paper and magnetic tape for storage as classified material.

i. Collect and secure all classified waste containers in an approved safe.

j. Remove "SECRET" signs from all equipment and the "Restricted Area" signs from the doors.

k. Collect, review, and file user logs. Complete the KDS Security Procedures Check List.

E. Emergency Plan

1. In the event of an emergency or malfunction which prevents verification of a successful completion of the KDE declassification procedure, the system will remain in a classified mode until such time as the declassification and verification can be accomplished. The ADP System Security Custodian will be immediately notified. The custodian will take steps to either have the system repaired by cleared service personnel or have the classified media (core, cache, RS04 disk) removed and stored in approved containers pending repair of the system.

2. In case of an emergency the following will be notified:

a. KDF System Security Custodian

b. ADP System Security Supervisor

c. Government Security Unit Supervisor

d. Plant Protection Unit Supervisor

e. Additional personnel, as required, in accordance with Facility Emergency Procedures.

SEC:REQ:S7:f:pt/146

3. Evacuation. Should it be necessary to evacuate the KDF, the following actions will be accomplished (if possible) prior to evacuation.

   a. The System Security Custodian or his designated alternate will:

      (1) Lock all classified material in approved cabinets.

      (2) Execute System Clear Program.

      (3) Power down the KDF System. (The emergency power cutoff should only be used in emergency evacuations, as its use may damage the equipment.

      (4) Secure the KDF doors.

   b. In the case of prolonged or permanent evacuation, or when it is not possible to safeguard the classified material adaquately, the Facility Security Supervisor shall initiate other safeguards for relocation of or, as a last resort, destruction of classified material in accordance with the Facility Emergency Procedures.

   c. At the conclusion of the emergency, the System Security Supervisor will complete an inventory to determine if any classified material is missing. The Security Supervisor will be notified immediately upon suspicion or discovery of missing material.

4. Emergency During Non-Working Hours. In case of an emergency during non-working hours, Plant Protection Unit guards will act as required to handle the emergency. During non-working hours the KDF will normally by UNCLASSIFIED (i.e. it will have been declassified prior to the end of the working hours). If access is required during classified operations outside of normal working hours, a cleared guard will act as escort and shall make a detailed report of who entered the room, what was done, the time spent in the room, etc. The guard shall remain at the door until the area is properly secured. Plant protection will notify the personnel listed in "2" above.


ATE
LME