

An Overview of the Kernelized Secure Operating System (KSOS)

Tom Perrine
John Codd
Brian Hardy

Logicon - Operating Systems Division

1 INTRODUCTION

This paper will present the Kernelized Secure Operating System (KSOS) as it exists today, with emphasis on its security policy, architecture, and its ability to support secure applications, specifically the ACCAT GUARD multi-level-secure application. A discussion of plans for its future development and qualitative performance information are also included.

1.1 Description of KSOS

KSOS is a multi-level-secure (MLS) computer operating system consisting of a security kernel, Non-Kernel Security-Related (NKSR) utility programs, and an optional UNIX application support environment. A KSOS software development environment will also be provided.

KSOS was designed to be a provably secure replacement for the UNIX operating system, Version 6. The system runs on an unmodified Digital Equipment Corporation PDP-11/70. The KSOS system enforces a formally specified security policy, encompassing mandatory access, integrity, and discretionary access models. Application programs that have been developed for the non-secure UNIX operating system may be ported to the highly-secure KSOS environment with minimal effort.

The KSOS system has been informally evaluated by the DoD Computer Security Center (CSC) and has been characterized as "an excellent base for developing into an AI system." [1] This is especially significant, as KSOS was designed many years before the DoD CSC Trusted Computing System Evaluation Criteria [2] was published.

1.2 KSOS Functional Architecture

The KSOS system is made up of the following functional areas: the Kernel, the Non-Kernel-Security-Related (NKSR) software, and the Kernel Interface Package (KIP). The security kernel provides the basic operating system functions of the system and enforces the security policy. The NKSR programs provide additional operating system functions and utility operations. The KIP is a UNIX-compatible run-time environment for running UNIX application programs. In the future, KSOS will include a fourth area: the KSOS development environment.

This work was sponsored by Naval Electronics System Command (NAVELEX) contract number N00039-83-0144. UNIX is a trademark of AT&T Bell Laboratories. DEC, PDP, and VAX are trademarks of Digital Equipment Corporation.

1.3 History of the KSOS project

KSOS was originally intended to be the first production-quality multi-level-secure operating system and to provide a secure UNIX replacement.[3] It was based on the results of the UCLA Data Secure UNIX [4] and MITRE security kernel experiments. [5][6]

The KSOS project began in 1977 at Ford Aerospace and Communications Corporation (FACC). Since 1981, Logicon has continued to support and develop KSOS.

While KSOS was being developed to prove the concept of a buildable security kernel, the Advanced Command and Control Architectural Testbed (ACCAT) GUARD application was developed to prove the concept of a multi-level-secure "guard" program to provide a verifiably secure access to multi-level all-source databases distributed on ARPANET-like networks. [7] Initially, a UNIX prototype of ACCAT GUARD was developed in anticipation of a secure UNIX replacement, to serve as the base operating system for GUARD.

The KSOS Kernel Interface Package (KIP) was developed to allow the migration of the UNIX-based GUARD prototype to KSOS with minimal software changes. This package has since been used to port other UNIX software to KSOS, and has demonstrated significant performance improvement over the previous UNIX compatibility supported by the original KSOS UNIX Emulator.

ACCAT GUARD, using the KIP, is currently undergoing accreditation review as a multi-level-secure system. It has shown that KSOS is robust and capable of supporting a rigorous Security Test and Evaluation (ST&E).

KSOS also served as a testbed for many (then advanced) features of secure systems. KSOS has helped to prove many of the concepts embodied in the DoD Trusted Computer Evaluation Criteria.

2 KSOS SECURITY POLICY

The KSOS security policy encompasses three orthogonal policy models, one for mandatory security, one for integrity and one for discretionary access protection. These models are defined in terms of objects (data containers) and subjects (processes acting on behalf of a user) and the rules under which subjects may access objects. If any of the three models would deny access to an object, the access is denied, i.e. all of the assertions of all of the models must be maintained at all times. All accesses to objects are mediated by the Kernel, ensuring mandatory access controls. It is not possible for a subject to access an object without Kernel intervention.

Every object in the system is marked with trusted labels, which consist of a security level, an integrity level and discretionary access permissions. Every object is labeled (marked) by the Kernel when it is created. These labels are used by the implementations of the security policy, within the Kernel, to permit or prevent accesses, as specified below.

2.1 Mandatory Security Model

The KSOS mandatory security model is the Bell-LaPadula model [8], which is in turn based on Department of Defense policies for the handling and dissemination of classified material. It is described in terms of the "simple security property,"

which determines what information a user may see, and the "security *-property," which prevents a user from lowering the classification of information (downgrading). The KSOS model includes both security classification levels, such as "SECRET" or "TOP SECRET", and "need-to-know" categories, such as "NO FOREIGN" or "NO CONTRACTOR".

2.2 Integrity Model

In this model, integrity is defined as the mathematical dual of security, and the intent of the model is to protect the system's information from modification, while allowing it to be read by any process.

As the mandatory security model controls who may obtain data stored in the system, the integrity model controls who may place data into the system, and how that data may be combined with other data.

There are two integrity properties which control object accesses: the "simple integrity property" and the "integrity *-property". The simple integrity property prevents a high-integrity process from reading low integrity data, which might then be written into a high integrity object. [9] The integrity *-property prevents a low integrity process from writing into a high integrity object. These properties keep low integrity ("less trusted") information from propagating into high integrity ("more trusted") objects.

An example is the KSOS mount table. Every process in the system may need to read this database, but only the operator or system administrator may change it. This database is assigned an integrity level of OPERATOR, which prevents user processes (running at the lower integrity level USER) from writing to the database file.

2.3 Discretionary Access Model

The KSOS system also includes a discretionary access model, which is derived from the UNIX discretionary model. [10] As in UNIX, every person using the system is assigned a user identifier, and every user is a member of at least one group of users. Every object has a permission set for the objects' owner, other members of the owner's group, and other users of the system. There are three permissions; "read", "write" and "execute", indicating the ability to extract information from, send information to, and (for files) the permission to load the file into memory as an executable program. These permissions are established at the discretion of the objects' owner.

3 KSOS KERNEL

The heart of the system is the KSOS Security Kernel. The Kernel is a complete operating system which provides a secure environment for the execution of user programs. It supports multiple isolated processes, a file system and a set of "supervisor" services. The Kernel is based on a reference monitor concept, wherein every access to every object is mediated by the Kernel according to its security policy.

3.1 Kernel objects and subsystems

The Kernel supports the following objects: processes, memory segments, devices, disk extents, files, and file subtypes. These objects are created and destroyed only by the Kernel, and all accesses of the objects are controlled by the Kernel in accordance with its security policy. Each object is the responsibility of one of the major subsystems of the Kernel, described below. All Kernel objects are labeled with their security and integrity levels and discretionary access information. All Kernel objects are assigned a unique identifier called a Secure Entity Identifier (SEID, pronounced "seed") which is a binary quantity. The SEID can be thought of as the Kernel's "name" for an object. A SEID is not a capability, and having the SEID does not imply any access privileges to the object. The only way to manipulate the Kernel objects is through the use of Kernel calls, identifying the object by its SEID.

The KSOS Kernel is split into four major functional areas: Process Management, Memory Management, Input/Output, and the Reference Monitor. Each of these areas are responsible for maintaining internal Kernel databases reflecting the state of all objects under the control of the Kernel.

3.2 Processes

All KSOS processes are managed by the Kernel Process Management Subsystem. This subsystem creates and deletes processes from the system, schedules them for execution and controls all interprocess communication. The real-time clock is also implemented in this subsystem, as are pseudo-interrupts and software trap handlers. Some of these sub-subsystems are visible to a user process by means of Kernel calls, others, such as the scheduler, are acting "behind the scenes," and are invisible to the user process.

The Kernel process is the only active object (subject) in the KSOS object space. Processes are the means by which programs are executed on the machine. A process performs its work by manipulating KSOS objects, i.e. reading and writing to and from files or devices, or communicating with other processes. A process consists of a program image, process context information, a memory address space, and a processor state.

New processes come into being at the request of other processes through use of Kernel calls.

A process may execute with special privileges. Such a process is a "trusted" process and may violate the KSOS Kernel security policy or use system control functions. These processes can become privileged only through the actions of the System Administrator.

3.3 Memory Segments

The Kernel Segment Management Subsystem is responsible for allocating, deallocating, swapping and controlling access to the segments of the primary memory of the system. The details of physical memory management and swapping are typical of many operating systems, and will not be discussed here. We will concentrate on the novel features of the KSOS segmentation subsystem.

Memory segments are an abstraction of the virtual memory visible to a process. Memory is made up of segments, each of which resides in either the Kernel, Supervisor, or User domains. (The domain structure is provided by the PDP-11 memory management hardware.) The KSOS Kernel resides in Kernel domain, NKSR programs typically reside in Supervisor domain and user application programs reside in User domain. A process program image and its data reside in a single domain, but transfers of control may span domain boundaries (under Kernel supervision).

User memory is organized into named segments. Like processes, segments are named by their SEIDs. A user process can have up to 16 segments resident in memory at any time, eight of which are allocated by the hardware architecture for instructions (I-Space) and eight of which can contain only data (D-Space). Each segment is limited in size to 8K bytes. Therefore, there is a limitation of 64K bytes of instructions and 64K bytes of data per memory management domain, per process. The maximum of 64K bytes address space can be spanned by the memory management system only if every segment is of the maximum size.

The Kernel permits a process to manage its data segments in a manner that can be used to best fit the application. For example, a process can dynamically create and destroy data segments, as well as determine which of its known segments are to be resident in main memory, and where, at any given time.

One of the more novel (and useful) features of the KSOS segmentation subsystem involves the use of shared segments. A process may create a segment, specifying that it is to be "sharable". Any other process may then "rendezvous" with the segment, under Kernel mediation. At this point, both processes have the same physical memory mapped into their virtual address spaces. This feature permits a very high bandwidth communication path for cooperating processes.

3.4 Input/Output Management

The KSOS I/O Subsystem is responsible for managing devices, disk extents, files and file subtypes. Devices, disk extents and files are increasingly abstract representations of physical input/output devices. File subtypes provide an extension of the Kernel defined object types.

3.4.1 Devices

Devices under KSOS are handled by the low-level device drivers within the Kernel I/O Management Subsystem. It is at this level that interrupts are handled, device commands and data are sent to and from the devices, and data buffers and device status registers are examined. Storage device contents are addressed by logical block number. Non-kernel programs are unable to perform I/O directly, but must make requests to the Kernel to have it perform I/O on their behalf.

Devices have minimum and maximum security levels that indicate what classifications of data may be sent to or received from it.

User terminal devices are handled in a novel fashion. There are several "virtual paths" to each terminal, each of which can be at a different security/integrity level. One of these paths is reserved for use by the system and is called the "Secure Path." This path provides a trusted communications path from the user to the Kernel, for use in invoking trusted functions. When the user uses the "secure attention" key, it is guaranteed that he is communicating with trusted software, and not a program that may have been left executing at the terminal by

another user. This secure path is used during login, logout and any time that the user must be communicating with trusted NKSR software.

3.4.2 Disk Extents

Disk extents are the next higher level of abstraction of devices, specifically mass-storage devices such as disks. An extent is a named set of contiguous blocks on a given disk device, which can be used as a private, logical storage device. As a "device" (and an object), an extent has security and integrity information governing information flows to and from the extent. The contents are selected by relative block number from the beginning of the extent. Disk extents are intended for use by programs that wish to manage their own storage space, without the imposition of any file structure by the Kernel. They might be used, for example, by a relational database, which uses its own special internal "file" format on top of an extent.

3.4.3 Files

The KSOS Kernel provides a "flat" file system. There are no directories or links and files are named only by their SEIDs. All of the security and integrity information is checked and maintained at the Kernel level. A file system resides in an extent on a disk, and may be "mounted" (made known to the system) or "unmounted". Files are allocated in 512 byte blocks, the blocks of which need not be contiguous. Both random and sequential access are supported. As files are created, they are marked by the Kernel with the security and integrity levels of the process that created it. Files may be opened, closed, read or written only by making requests to the Kernel.

3.4.4 File Subtypes

File subtypes allow a System Administrator to define a special type, or flavor, of file for special handling by the system. These are called "file subtypes," and may be thought of as a private object. They can be used in support of object-oriented programming, to implement special-use reference monitors on top of the Kernel.

File subtypes are a special object in KSOS. They have security, integrity and discretionary access information, just like other objects. They also have an owner. In practice, the discretionary access allows write access to the owner only. This becomes a "private type" of the type manager (the owner). Only the owner may open the subtype for writing.

When a file is created, a subtype may be specified. At this point, the subtype is entered into the Kernel's information about the file. Later, when a process attempts to open the file, it must have already successfully "opened" the subtype with the same mode, or the file open will fail. As the owner is the only process which may open the subtype for writing, only the owner may open the "subtyped" file for writing.

For example, the UNIX Directory Manager (UDM) implements the hierarchical UNIX-like file system from the more primitive Kernel file system. The UDM creates Kernel files with a subtype that only it may write. These files are then used by UDM as UNIX directories.

Later, any process that attempts to open the directory file, for writing, must have already opened the subtype for writing. As the UDM is the only process which may open the subtype for writing, no other process may open the directory file for writing. This ensures the integrity of the information in the directory file. However, any process may open the directory file for reading (subject to other access constraints, of course). Subtyped files can be thought of as a "non-discretionary, discretionary access model", where there are permissions for reading, writing and executing, but the permissions are set and maintained by the Kernel, and may not be changed at the discretion of a user program.

This feature of KSOS is very useful for implementing special-purpose databases where only a single process which "owns" the database is to be allowed to update it. It has also been used for the TCP/IP network daemon, to protect files used by the network manager, and will be used to implement multi-level-secure mailboxes for the Secure Mail Facility.

3.5 Reference Monitor

The KSOS reference monitor has been mentioned in passing in the discussions of the other Kernel functional areas. The reference monitor ensures that all accesses to the objects protected by the Kernel are permissible under the KSOS Security Model. This module is invoked by the other functional areas to determine the validity of the attempted accesses (or information flows).

4 NON-KERNEL SECURITY-RELATED (NКСR) SOFTWARE

The NKSR software that is part of the KSOS system falls into four functional areas: Secure User Services, System Operation Services, System Maintenance Services, and System Administration Services.

4.1 Secure User Services

The Secure User Services NKSR programs are responsible for initializing the KSOS system and providing a secure path from the user to all of the trusted NKSR services. Programs in this area include:

- * Initial Process

This program is responsible for initializing the security levels of the KSOS system objects. It is the first process to execute after the bootstrap process.

- * Secure Server Process

This is the command processor of the system. It manages the different virtual paths to the user's terminal and invokes other NKSR services at the request of the user.

- * Login and Logout

Login is responsible for performing user authentication functions and creating the initial user environment. Logout destroys the user's process and makes the terminal available to other users.

4.2 System Operation Services

These programs contain functions that are necessary to support a general purpose operating system. Such functions include:

- * Line Printer Daemon

This is the "daemon" process that performs line printer spooling.

- * Mount/Unmount

These facilities control the mounting and unmounting of file systems.

- * Network Daemon

This daemon process handles the TCP/IP DDN or ARPANET connections.

- * UNIX Directory Manager (UDM)

UDM implements a hierarchical, UNIX-like file system from the more primitive Kernel "flat" file system. Text string names and directories are implemented by this program.

4.3 System Maintenance Services

These programs provide the necessary functions to maintain the KSOS file systems in a usable state, such as:

- * Storage Consistency Check (STC)

STC checks the consistency of the Kernel file system, reporting any lost blocks, duplicated blocks, etc.

- * Directory Consistency Check (DCC)

DCC checks the consistency of the UNIX file system maintained by the UNIX Directory Manager, reporting directories that are in an inconsistent state, etc.

- * File System Dump/Restore

These utilities provide backup and restore of KSOS file systems.

4.4 System Administration Services

This class of programs provides the functions needed to assist the System Administrator in easily managing a multi-user, multi-level system. These functions include:

* User Registration and Removal

This program allows the System Administrator to add new users, remove users and identify the clearances of the users to the system.

* System Profile Maintenance

This program maintains the system profile database, which describes the particular KSOS installation in terms of software versions, site name, etc.

* Audit Capture Process (ACP)

The Kernel and NKSR software generate audit events for several reasons, including user login, logout, object creation, access failures, activity on possible covert channels, etc. The Audit Capture Process receives these messages and writes them to an audit file.

5 KERNEL INTERFACE PACKAGE (KIP)

The Kernel Interface Package (KIP) provides a UNIX Version 6 system-call compatible interface, except for those system calls which have been identified as security flaws of UNIX. (The functions of the latter system calls have been subsumed into the NKSR software.)

The KIP is a library of subroutines and functions, one for every supported UNIX system call, which are linked with the user program. These library functions invoke KSOS Kernel calls to carry out their functions. Very little data is maintained by the KIP from call to call. The KIP can be viewed as a UNIX to KSOS call translator.

The KSOS KIP allows the easy migration of existing software written for the UNIX environment to the multi-level-secure environment of KSOS. This method of providing a UNIX environment allows source-level compatibility, and better performance than the original UNIX Emulator.

6 KSOS DEVELOPMENT ENVIRONMENT

At the present time, the KSOS software development environment requires a PWB/UNIX system. All programs are prepared using the UNIX development tools.

There are plans to provide a full KSOS development environment running on KSOS. This will give the KSOS system the ability to maintain itself. Initially, a minimum set of software development tools will be installed on KSOS. As a preliminary feasibility study, the "ed" editor was ported with very few changes, making use of the KIP.

The next phase is to select the set of tools that will be ported from UNIX. Some candidates for UNIX software to be ported are the UNIX "shell", the "C" compiler, the UNIX assembler, the loader and the Source Code Control System. A screen editor will also be chosen and ported.

7 ACCAT GUARD on KSOS

The Advanced Command and Control Architectural Testbed (ACCAT) GUARD application is a multi-level-secure application developed for the Naval Electronic Systems Command (NAVELEX).

ACCAT GUARD will provide a certifiably secure interface between two computers or subnets on the Defense Data Network (DDN) which are operating at different security levels.

The GUARD system is responsible for secure exchange of information between HIGH level and LOW level network connections. The boundary between these HIGH and LOW levels is guaranteed through the KSOS multi-level security protection mechanisms in accordance with the DoD security policies.

ACCAT GUARD allows the passing of information from the LOW to the HIGH network automatically, but ensures that all information passing from the HIGH network to the LOW network is subjected to manual sanitization and manual review for downgrade. The downgrade is performed by a formally specified, trusted program, the Downgrade Trusted Process (DGTP), which is the only component of the GUARD application software which is trusted (or privileged) to perform the downgrade (by the KSOS security mechanisms).

The ACCAT GUARD system has passed Security Test and Evaluation (ST&E), and is under accreditation review by the Defense Intelligence Agency. The ST&E has shown the robustness of the Kernel and the application, by executing under a wide variety of load conditions for extended periods of time. Most importantly, no security weaknesses were discovered during the ST&E.

ACCAT GUARD was developed with several goals in mind. Its primary goal is to validate the concept of a guard as a buildable multi-level-secure application. But in addition it validates the concept of KSOS as a production-quality security kernel, and demonstrates the capability of KSOS to host an application which was originally written for execution on UNIX, using the KIP.

8 KSOS - FUTURE PLANS

KSOS development is continuing at Logicon, in support of the ACCAT GUARD system. Additional areas of research include developing KSOS into a DoD CSC A1 certifiable system, porting KSOS to alternate hardware architectures, and providing performance and functional enhancements to the existing system.

8.1 DOD CSC A1 Certification

KSOS was designed to be a secure system before the DoD CSC published the Trusted Computer System Evaluation Criteria. The security policy was not an add-on, and security was the prime design goal. The KSOS Kernel is described by a Formal Top Level Specification (FTLS), expressed in SPECIAL. An informal review of correspondence between the FTLS and the KSOS implementation has been performed, but the formal document has not yet been produced. KSOS has been characterized as an excellent base from which to build a secure system.

Over the lifetime of KSOS, several verification efforts have been performed by the MITRE corporation. The results of these efforts are available from MITRE. The most recent effort [11] involved the Kernel FTLS, which was examined using the

Hierarchical Design Methodology (HDM) [12] tools.

This effort produced 1638 theorems, 939 of which were proven trivially and 431 of which were eliminated as duplicates. This left 268 theorems, of which the theorem prover was able to prove 47, leaving 221 unproven theorems. The number of unproven theorems can be further reduced to a minimum by several methods, including adding additional assertions to the specification. Any remaining unproven theorems which are shown to indicate covert channels may be handled by limiting the bandwidth of the channel, or auditing the use of the channels.

Since this verification effort, however, the Kernel has had minor changes. The current FTLS correctly reflects the state of the KSOS implementation, but needs additional work in the area of specifying more assertions, to allow more of the theorem proving to be performed automatically.

Although Logicon is not currently under contract to develop KSOS to the AI level, we expect this effort to proceed in parallel with further development of the system, i.e. all changes that are made will be designed with eventual AI certification in mind. According to the Computer Security Center, "KSOS is an excellent base for developing into an AI system." All of the areas in which KSOS is deficient have been identified, and the necessary development activities have been specified.

8.2 New hardware architectures

One of the original design goals of KSOS was to provide an easily portable system that could be moved across machine architectures with minimal re-design effort.

We are currently investigating the migration of the KSOS system to other hardware architectures, specifically, the Digital Equipment Corporation VAX. Most limits of KSOS performance are imposed by the architecture of the PDP-11. Migration to a VAX will provide many benefits, especially allowing KSOS to reside on a wide price/performance range of machines, all running functionally identical Kernel software, with only minor changes to support different central processors.

It is expected that we will be able to move KSOS to the new PDP-11/73 hardware which is a less-expensive, single-board implementation of the full PDP-11 architecture. This could be accomplished with little or no effort providing a low-cost hardware base for the current PDP-11 kernel.

Other architectures which may be examined in the future include the Motorola 68000 family, National Semiconductor NS32000 series and other high-performance microprocessor families.

8.3 Performance and Functional Enhancements

Several opportunities for performance and functional improvements have been identified within the KSOS system. In particular, there are plans to improve the Kernel Input/Output Subsystem, specifically in the areas of terminal handling, asynchronous I/O and device request handling.

The current KSOS design allows multiple processes to share a single copy of read/execute-only instruction spaces. However, the implementation of shared instruction segments is incomplete. Shared instruction space will also decrease the swapping load on the system and increase throughput dramatically.

The current KSOS KIP supports a UNIX Version 6 environment within the MLS environment of KSOS. Other system call translators will be written for other UNIX versions. For example, interface packages could be written to support the 4.2BSD or AT&T System V system call interfaces. Once the system calls are available, application software can be ported with minimal effort.

9 PERFORMANCE

This section reports the results of some informal, qualitative performance measurements which were performed recently. The figures are not intended to state absolutely the performance differences between the environments, but to give a feel for the performance of a security kernel and point out that applications that are intended to run on a kernel will benefit if written to use the native environment of the kernel. Applications can be ported directly, using the KIP, but performance will be traded for ease of migration.

These programs use the following environments: the KSOS Kernel native-mode run-time environment, the UNIX Version 6 environment as provided by the Kernel Interface Package on top of the Kernel, and the PWB/UNIX environment. All systems were run on the same PDP-11/70 at Logicon.

9.1 Tasks

The following types of programs were identified as being "interesting", because they exercise various parts of the different environments and are typical of the tasks of application programs.

- * CPU intensive

This task is to repeatedly compute the prime numbers less than 10000, using a relatively inefficient algorithm (Sieve of Eratosthenes).

- * Interprocess Communication (small messages)

This task is to pass many 10-byte messages from one process to another.

- * Interprocess Communication (large message)

This task is to repeatedly pass a single large (1000-byte) message from one process to another and back again.

- * File Input/Output Intensive

This task is to open an existing file, write 64 blocks of data, rewind the file and read the data.

- * Process Creation

This task times the various process creation mechanisms. A process is loaded into the system. This process starts off a child process, and then exits. The child does the same thing. This continues for 100 process creations.

9.2 Experimental Results

The selected test cases have been chosen to correspond to the types of operations typically performed by application software. Test programs were written in C for execution under UNIX. These UNIX-based programs were executed under KSOS/KIP environment with no changes to the software. Finally, each program was modified to execute directly with the KSOS Kernel. For each test case, the total elapsed time required to complete the test was measured. The test results were normalized with respect to the time required to complete the same functional test under the UNIX operating system. These results are summarized in the following table.

Performance Characteristics Ratio - KSOS vs UNIX

Test Scenario	UNIX	KSOS (KIP)	KSOS (Kernel)
CPU-bound	1.0	1.0	1.0
IPC (10 bytes/msg)	1.0	25.0	3.9
IPC (1000 bytes/msg)	1.0	65.0	1.7
IPC (5000 bytes/msg)	1.0	325.0	0.3
I/O-bound	1.0	1.6	1.4
File Creation	1.0	139.0	13.9
Process Creation	1.0	67.6	30.8

Software portability versus desired performance characteristics continues to be the topic of intense debate and trade-off analyses. This issue is especially significant when developing application software which will operate under a secure system such as KSOS. As shown in the table, rather significant gains in performance can be achieved by tailoring the application to the features provided by the kernel. In particular, the ACCAT GUARD system performance was improved by approximately a factor of three by applying this concept to a small set of carefully chosen application modules.

It is interesting to note that for CPU-bound application software, the KSOS Security Kernel does not impose any performance penalties when compared to UNIX. Also, I/O-bound software only has a marginal decrease in performance. This result was anticipated due to the differences in the I/O design philosophy between UNIX and KSOS. Furthermore, the version of the test that executed directly under the KSOS Kernel was only slightly improved over the performance of the similar test which executed under the KSOS/KIP environment. Therefore, for this case of application processing, the UNIX-based software can be migrated to KSOS with relatively small performance penalties.

In applications which required a high degree of interprocess communication, the tests clearly indicate that using the features of the kernel will provide rather significant increases in performance, particularly when rather large messages must be exchanged. It is interesting to note that as the message size increases, the interprocess communication features provided by the KSOS Kernel will permit a higher effective throughput rate than UNIX.

Finally, new object creations, particularly processes and UNIX file systems managed outside the KSOS Kernel, will require considerably more computational resources than non-security kernel-based operating systems such as UNIX. Again, this result was anticipated due to significant differences in the design between UNIX and KSOS.

10 CONCLUSION

KSOS is a demonstrated, full-featured operating system built according to the latest philosophies in computer security. It runs on commercially available hardware and holds the promise of providing secure processing on a family of hardware that spans the spectrum of computers from micro to mainframe. With KSOS, a system implementer may easily port existing operational software to a secure environment and not necessarily pay a great performance penalty. KSOS is an ongoing software project that offers a solution to the MLS problem while continuing to improve its features and performance.

REFERENCES

- [1] Letter to Commander, NAVELEX, 25 June 1984, subject: KSOS-11 Security Assessment
- [2] CSC-STD-001-83, "Department of Defense Trusted Computer System Evaluation Criteria," 15 August 1983.
- [3] E.J. McCauley and P.J. Drongowski, "KSOS: The Design of a Secure Operating System," in Proceedings, AFIPS National Computer Conference, AFIPS Press, Arlington, Va., 1979, Vol 48, pp. 345-353.
- [4] G.J. Popek, M. Kampe, C.S. Kline, A. Stoughton, M. Urban, and E.J. Walton, "UCLA Secure UNIX," in Proceedings, AFIPS National Computer Conference, AFIPS Press, Arlington, Va., Vol 48, pp. 355-364.
- [5] W.L. Schiller, "Design of a Security Kernel for the PDP-11/45," MITRE MTR-2709, MITRE Corp., Bedford, Mass., June 1973.
- [6] K. Biba, J. Woodward, and G. Nibaldi, "A Kernel Based Secure UNIX Design," MITRE ESD-TR-79-134, MITRE Corp., Bedford, Mass., May 1979
- [7] D. Baldauf, "ACCAT GUARD Overview," MITRE MTR-3861, MITRE Corp., Bedford, Mass., Nov. 1979
- [8] D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MITRE MTR-2997, MITRE Corp, Bedford, Mass., July 1975.
- [9] Biba, K.J. "Integrity Considerations for Secure Computer Systems," MITRE MTR-3153, MITRE Corp., Bedford, Mass., June 1975.
- [10] D.M. Richie and K. Thompson, "The UNIX Timesharing System," in Communications of the ACM, Vol. 17, No. 5, pp. 365-375. (May 1974)
- [11] K.E. Kirkpatrick, "KSOS Verification Part I: Analysis of the Specifications and Use of the Verification Tools," (working paper), MITRE Corp., Bedford, Mass., February 1982
- [12] B.A. Silverberg, "The HDM Handbook, Volume II: The languages and tools of HDM," SRI International, 1979.