ARCHITECTURE OF A REAL TIME OPERATING SYSTEM*

J. L. Pruitt
W. W. Case
M&S Computing, Inc.
Huntsville, Alabama

Architecture is receiving increasing recognition as a major design factor for operating systems development which contributes to the clarity, and modifiability of the completed system. The MOSS Operating System uses an architecture based on hierarchical levels of system functions overlayed dynamically by asynchronous cooperating processes carrying out the system activities.

Since efficient operation in a real time environment requires that the number of processes and process switches be kept to a minimum, the MOSS system uses processes only where a truly asynchronous activity is identified. The layers of the MOSS Operating System do not represent a hierarchical structure of virtual machine processes, but rather a hierarchy of functions used to create the processes.

This paper describes the layering concepts and process concepts defining the system architecture. It also presents an overview of the specific functions and processes of the MOSS Operating System.

Key words and phrases: operating system design, real time operating system, layered operating system, software architecture, and process communication.

CR Categories: 3.80, 3.83, 4.35.

1. INTRODUCTION

The Modular Operating System for SUMC (MOSS) is a general purpose real time operating system for the RCA developed Space Ultrareliable Modular Computer (SUMC). MOSS is currently being implemented at M&S Computing, Inc., in Huntsville, Alabama, and is scheduled to be fully operational in the first quarter of 1977.

MOSS attempts to merge recent developments in operating system design to create a real time operating system which is more modifiable than ad hoc systems developed with previous design policies. Since MOSS is to be used as an experimental base to test various real time processing concepts, modifiability has been a major design consideration. The architecture has been selected to provide a basic framework into which relatively independent subsystems fit and interact in a well-defined manner. These subsystems can be modified or exchanged in response to changing user requirements.

Since MOSS is a real time operating system, performance has been considered throughout the system definition and design. Continual tradeoffs have been made between the conflicting performance and modifiability objectives. A major architectural decision was the separation of the static and dynamic structures of the system. The static structure, consisting of a hierarchy of functions, defines the basic framework of the subsystems and their interaction. The dynamic structure, consisting of cooperating processes representing user applications and asynchronous system activities, is superposed on the static framework. Thus, the frequent process switching and associated overhead of deeply layered processes has been avoided. Inefficiencies due to frequent handshaking between the static layers of functions

*This work was supported by NASA Contract No. NAS8-31222, managed by the Marshall Space Flight Center.

have been tolerated to aid modifiability.

The remainder of this paper is devoted to a description of the actual MOSS architecture and the concepts on which it is based. A brief description of the SUMC hardware is also included to aid reader understanding of the MOSS architecture.

2. SUMC HARDWARE

The SUMC is a microcoded processing unit organized around the basic System 360 instruction set. The major features of the hardware include:

- A multiple level interrupt structure with a separate set of registers in scratchpad memory for each interrupt level.
- Address translation hardware and microcode to support virtual memory.
- Memory protection microcode which provides access protection (read, write, execute) on a page basis.
- Memory units which provide a data lock capability to prevent simultaneous multiple access to locked data [6].

3. MOSS LAYER CONCEPTS

The MOSS architecture is based on hierarchical layers of partitions corresponding to levels of abstractions [2,3,4]. A layer defines a level in the hierarchical structure of the system and contains one or more partitions whose interaction is limited and well understood. A partition is a group of functions which are related in their effects and share common resources (data structures and/or hardware features).

Within each partition there are internal and external functions. External functions may be invoked directly by functions of another partition. The external functions of a partition provide the primitive operations of the partition to other partitions in the system. Internal functions are used only within a partition and cannot be referenced from other partitions. The internal functions are derived from the decomposition of the partition into modules which support "information hiding [5]".

3.1 Partition Interaction Rules

Interaction between partitions on the individual layers must conform to the following rules:

1. Each partition owns certain resources which other partitions are not permitted to access. The resources owned by a partition consist of the system functions and data structures used to support the partition.
2. Each layer in the hierarchy is not aware of the existence of any layers above it and cannot invoke any partition of a higher layer.

Partitions of higher layers may invoke the external functions of a partition of a lower layer or the same layer to utilize the resources of that partition. The invocation hierarchy of layers is not restricted to references only one level below the requesting layer. Rather, layers may directly reference any lower layer in the hierarchy. The enforcement of the layer interaction rules and the knowledge of external functions available to higher layers has been maintained as a design management responsibility, not a function of the executing software.

3.2 Partition Selection Criteria

Partitions are selected to support abstractions of system capabilities. Each abstraction is supported by a set of software functions which map the characteristics of hardware capabilities and/or elementary logical structures into higher level logical structures which present a simple, consistent view of the hardware or software represented. Partitions are provided to hide the actual implementation of complex mechanisms from higher layers of the system.

Partitions are placed on layers based on the nature of their abstraction. The lowest layers contain partitions which are closest to the actual machine and provide basic services. Succeedingly higher layers are selected based on services required from lower layers and services provided for higher layers. Section 5 identifies and describes the partitions selected for MOSS.

3.3 Hardware Structure Versus Software Structure

The SUMC hardware and MOSS software are being developed concurrently, allowing hardware/software tradeoffs to be made during the design phases. The hardware interrupt structure has been matched to the software structure, thus, eliminating the restrictions on the MOSS design which would have resulted if the design had to match an existing interrupt structure

The hierarchy of the layers reflects the hierarchy of the hardware interrupt structure. Since an interrupt results in a change of execution sequence, much the same as a direct call in software, an interrupt generated during the execution of a layer function which is processed by a higher layer represents a violation of the layer interaction rules. For example, since the MOSS partition responsible for handling I/O interrupts is in layer 2, the partitions of layer 1 would not be allowed to receive I/O interrupts during their execution and could not use the I/O facilities. However, partitions in layer 3 and above would be allowed to receive I/O interrupts and to use the I/O facilities of layer 2.

4. MOSS PROCESS CONCEPTS

The static structure of MOSS is overlayed by a dynamic structure of asynchronous, cooperating processes carrying out the system activities. Processes do not have hierarchical relationships. Instead, a process is the independent, asynchronous execution of one or more of the system partitions under control of the static structure. Several processes may execute a partition simultaneously since the functions of the partitions contain the facilities which coordinate the multiple requests for their resources.

The MOSS process concept is based on the following definitions:

1. A processor is an entity which performs transformations of data. In the MOSS environment, a processor is the Central Processing Unit (CPU).

2. A program is an ordered sequence of tranformations presented for execution to the processor.

3. A process is the actual or potential execution of an ordered sequence of transformations on a set of data [7].

A process is a logical entity which exists independently from a processor, but requires a processor to advance in state. More than one process, each representing some collection of programs and resources, can execute concurrently sharing available resources, especially the processor.

There are two significant extensions to the basic process concept which have been used in MOSS. The first, the subprocess concept, depends on the distinction between external and internal interrupts. An external interrupt is an interrupt which signals an occurrence which is not directly related to the currently executing instruction in the processor -- for example, an I/O completion interrupt. An internal interrupt, on the other hand, results directly from the execution of the current instruction -- for example, a supervisor call (SVC) or a page trap interrupt. A subprocess is used to denote the processing of an internal interrupt. The internal interrupt performs a service directly supporting the active process and, as such, is treated as a continuation of the process. The subprocess is similar to a subroutine invoked via a direct transfer and uses the process state information as parameters.

The other extension of the process concept is the task process. A task is the implementation of a user program on MOSS. The task process represents the execution of the user program and all subprocesses associated with it. The subprocesses provide the synchronous operating system services invoked to support the user program including explicit supervisor calls, implicit paging requests, and implicit error handling. The state information for a task process includes user supplied resource and real time control parameters in addition to the hardware state information used for all processes.

4.1 Process Selection Criteria

Processes are selected to control the asynchronous execution of independent sequences of activities. Independent is used in the sense that the activities do not directly communicate (i.e., via direct call) and their execution is not dependent on the state information of another process. The following criteria are used to identify processes in MOSS:

1. A separate process is used to represent each user task.

2. A separate process is used to represent each MOSS function which:

   - Performs a service which does not directly support another process or task; or

   - Performs a service which directly supports another process or task, but which is executed asynchronously.

3. A separate process is used to represent each external interrupt.

Section 6 identifies and describes each of the processes selected for MOSS.

4.2 Process Coordination

Since the processes within the MOSS environment cannot communicate directly, process coordination mechanisms are provided which allow processes to exchange data and synchronize their activities. MOSS provides three distinct mechanisms for process coordination which have been chosen to restrict the process interactions to a minimal, yet, sufficient set. These mechanisms are:

- The data lock; mutually exclusive access to shared data.
- The software interrupt (SINT); queuing requests for service.
- The block/unblock; awaiting completion of a requested service.

These functions could have been implemented with a general mechanism such as Dijkstra's semaphores, but only at the risk of increased complexity in process interactions.

MOSS processes are not hierarchically related and thus, not limited in their interactions by the static structure of the system. The MOSS process coordination mechanisms limit process interactions aiding the reliability and testability of not only this design, but also future changes.

4.2.1 Data Locks

Processes accessing shared data must be coordinated to insure the integrity of the data. This coordination is provided by a hardware assisted lock mechanism which guarantees mutually exclusive access to shared data. The hardware provides lock and unlock instructions which address a lockword in memory. The lock instruction stores a process identification code in the lockword. The unlock instruction requires a matching code to effect the unlock (zeroing the lockword). These instructions provide protection against accidentally issued unlocks.

A process requesting a data lock for a currently locked lockword is suspended until the lockword has been unlocked. Deadlocks are prevented by an ordering scheme that is a natural result of the hierarchical structure. All functions accessing a specific data structure and the data structure itself are on the same layer in the hierarchy. Thus, each function is complete with respect to the data structure, i.e., the lock and unlock are contained within the function. Downward calls can be made with a data structure locked, but upward returns always leave the data structure unlocked, thus, preventing circular lock requests among processes. There is no MOSS mechanism to insure the pairing of locks and unlocks within a function; this has been maintained as a design management responsibility.

4.2.2 Software Interrupts

The software interrupt (SINT) mechanism provides the capability for process to be queued for service in much the same manner that a hardware interrupt queues an interrupt handling process. Each process contains a SINT count as part of its state information. The value of the SINT count, ranging from zero to an arbitrary positive integer, indicates the number of the queued requests for the process. A SINT driven process is able to service a queue of requests serially and suspend itself when finished (when the SINT count becomes zero). Other processes in the system may queue the SINT driven process by incrementing its SINT count. There are two basic primitives associated with the SINT mechanism:

1. Wait allows a service process to decrement its SINT count indicating completion of the current service request. If the value of the SINT count goes to zero, the process is suspended until the SINT count is incremented by another process. A wait request applies only to the process making the request.

2. Set SINT increments the SINT count of a process, queuing the specified process for service. If the value of the SINT was zero before the set SINT, the process immediately becomes eligible for activation. A process may not set its own SINT.

Information passing to a SINT driven process is through a known request queue for the process. A SINT driven process performs a single function associated with a layer in the hierarchical structure. This layer contains the request queue and the queue servicing routines. Invoking the process requires placing the request in the queue and then performing the set SINT operation.

4.2.3 Blocking and Unblocking Processes

The block/unblock mechanism is independent of the SINT mechanism. A blocked process is not released when its SINT counter is incremented. The two operations described below are provided.

1. Block allows a process to suspend

itself unconditionally.

2. Unblock allows a process to reactivate a blocked process. The unconditional block/unblock mechanism allows a process to suspend itself pending a request to be completed by another process. While the process is suspended, it can still have its SINT counter incremented by other processes. Its queue of requests is then handled serially when the process is unblocked.

5. MOSS LAYER SPECIFICATIONS

MOSS is organized into eleven hierarchical layers, each containing one or more partitions. This section describes each layer starting with the lowest hierarchical layer and proceeding to the highest (see Figure 1). Recall that modules within a given layer may not invoke modules of higher layers.

5.1 Layer 1

Layer 1 consists of the timer management, processor management, process management, exception handler, and log queue management partitions. These partitions provide basic system services which can be requested by all system partitions.

5.1.1 Timer Management Partition

This partition provides timing services for the MOSS processes. The partition controls the setting of an internal timer and the processing of the external interrupt generated when this timer expires (see Section 6.2).

5.1.2 Processor Management Partition

This partition maintains the status of the central processing unit and allocates the processor to the highest priority ready process. Time slicing of the processor is not utilized.

5.1.3 Process Management Partition

This partition controls the progress of processes in the MOSS environment. The partition creates processes, coordinates their activities, and deletes them. Process coordination is accomplished via the SINT and unconditional blocking/unblocking mechanisms.

The functions of the processor management and process management partitions produce a multiprogramming environment.

5.1.4 Exception Handler Partition

This partition analyzes all detected errors in MOSS. Hardware/firmware detected errors are reported via the SUMC interrupt structure while software detected errors are reported via normal procedure invocation.

5.1.5 Log Queue Management Partition

This partition manages the in-core buffers of the system log. All entries into the system log are made via this partition. The partition activates the log management process whenever a log buffer becomes full in order to have the full log buffer written to the system log data set.

5.2 Layer 2

Layer 2 contains the channel management partition. This partition centralizes the control of the SUMC channel hardware. It is responsible for scheduling the channels, processing I/O interrupts, and maintaining the channels' logical status.

The partition is placed at this hierarchical level to enable the memory management partition to request paging I/O operations.

5.3 Layer 3

Layer 3 contains the memory management partition. This partition supports the abstraction of virtual memory; i.e., the ability to reference an address space which is relatively independent of physical memory and whose contents may actually be in main memory or in a backing store called external paging memory (EPM). This allows main memory to be shared among programs whose individual or total memory requirements exceed the main memory size.

Under MOSS each task is assigned a linear address space of 2^{24} bytes. Each address space is divided into four segments; task private, job common, system common, and MOSS private. There are not separate copies in memory of the last three segments for each task. The hardware permits the sharing of single copies of each job's common area, the system common area, and the MOSS private area.

Main memory is allocated on a job basis and varies between a user-defined minimum and optimum amount depending on the requirements of other jobs. Paging for all tasks of a job is done in the main memory allocated to the job. This scheme contributes to the repeatability of successive executions of a job; an important

MOSS STRUCTURE OVERVIEW

HARDWARE INTERRUPT LEVEL	LAYER	PARTITIONS
User	11	1 User tasks 2 Systems tasks
SVC	10	1 SVC handler 2 External control 3 Log management
	9	1 Sampling performance monitoring
	8	1 Program management
	7	1 Logical I/O 2 Console management
	6	1 I/O resource management
	5	1 Access management
	4	1 Event management
Page Trap	3	1 Memory management
I/O	2	1 Channel management
Timer Exceptions	1	1 Timer management 2 Processor management 3 Process management 4 Exception handler 5 Log queue management

Figure 1

real time performance consideration.

When a page is referenced which is not in main memory, a page trap interrupt is generated. This internal interrupt is processed by the memory management partition which brings the required page into memory. However, the layering of MOSS prevents this interrupt from being honored if it is incurred by modules at or below the memory management partition. Therefore, such modules must always be locked in main memory.

Page swapping is performed on a page group basis, i.e., one to four pages of 1K bytes each. The SUMC virtual memory hardware supports address translation and page traps at the page group level.

5.4 Layer 4

Layer 4 contains the event management partition. This partition provides the services for communicating the occurrence of significant conditions (called events) between processes. The partition also provides the capability to suspend a process pending the occurrence of a logical combination of events.

Events do not exist in the MOSS environment until a process defines a condition to be an event. Although the event management partition maintains the status of events, it relies on other partitions to detect the conditions which constitute the event. That is, the reporting and processing of events has been centralized while event detection has been decentralized. This is consistent with the principal of dedicated ownership of resources discussed in Section 3.1.

5.5 Layer 5

Layer 5 contains the access management partition. This partition provides the services for controlling the access rights to selected resources in such a manner as to prevent deadlock situations between processes. A requesting process whose specified access request cannot be satisfied is blocked until the required resources become available.

To prevent deadlock situations, the partition requires that access requests for different classes of resources be made in a specific order and prohibits additional access requests for any class of resources to which the process currently has access [1].

5.6 Layer 6

Layer 6 contains the device management partition. This partition provides the services for allocating, scheduling, and maintaining the logical status of the SUMC I/O devices (except for the paging device controlled by the memory management partition).

The partition translates I/O requests into channel programs and queues the requests until the required device becomes available. The channel management partition is subsequently invoked to schedule the associated channel for execution of the channel program. The partition also performs I/O error recovery if required.

An actual situation involving this partition provides an example of the modifiability of MOSS: Because device characteristics and configurations are maintained by this partition and hidden from the other MOSS partitions, the other partitions could be designed and coded even though the device models and configurations were still undefined.

5.7 Layer 7

Layer 7 contains the logical I/O and console management partitions.

5.7.1 Logical I/O Partition

This partition provides the services for supporting I/O requests which are specified in terms of logical units rather than physical units. The partition performs two types of transformations; logical units are translated into their associated physical device addresses and logical I/O functions are translated into one or more physical I/O functions. The device management partition is invoked to perform any required device I/O.

5.7.2 Console Management Partition

This partition coordinates the use of the system console among the MOSS processes. The partition supports the abstraction of a virtual console for each process. This enables a process to use its own virtual console without the necessity of coordinating input or output with other users.

5.8 Layer 8

Layer 8 contains the program management partition. This partition provides services for controlling the initiation, termination, and status of jobs and tasks. A job is defined as a unit of work which consists of one or more tasks. A task is the basic unit of work processed by MOSS and is the smallest entity competing for system resources.

5.9 Layer 9

Layer 9 contains the sample performance

monitoring partition. This partition provides services for collecting system performance data at periodic time points.

The collected data indicates the current utilization of the system resources. This data is recorded and subsequently processed to provide a statistical report which indicates system bottlenecks and poorly utilized resources.

5.10 Layer 10

Layer 10 contains the SVC handler, external control, and log management partition.

5.10.1 SVC Handler Partition

This partition provides the interface between MOSS and tasks executing at the user level. All executing tasks must invoke this partition whenever MOSS services are required. The partition is invoked by an internal interrupt generated when a task executes a supervisor call (SVC) instruction

5.10.2 External Control Partition

This partition provides services which permit the control of MOSS operation from an external source (e.g., the console typewriter). The partition contains functions which allow the system operator to obtain and/or modify the current status of the system.

5.10.3 Log Management Partition

This partition maintains the system log data set. Its primary purpose is to output the system log buffers which are filled by the log queue management partition.

5.11 Layer 11

Layer 11 contains the user task and system task partitions. These partitions may not execute privileged instructions or access the MOSS private segment of the address space.

5.11.1 User Task Partition

This partition contains all user programs.

5.11.2 System Task Partition

This partition provides various system support programs. System tasks are processed by MOSS in the same manner as user tasks. System tasks include linkers, loaders, reader/interpreters, and output writers.

6. MOSS PROCESSES

As previously stated, the MOSS environment may be viewed as a set of cooperating processes where each process is a unit of work to which the processor may be allocated. MOSS provides facilities for controlling the creation, execution, and deletion of individual processes, and also provides services for controlling communication and synchronization between processes.

In this section, each major MOSS process is identified and its purpose explained.

6.1 I/O Interrupt Handler Process

This process performs the initial interpretation of the data furnished with I/O interrupts. The process remains inactive until a hardware I/O interrupt signal is accepted by the processor.

If the I/O interrupt indicates that an active I/O request has completed, the interrupt data is passed to the process which requested the I/O and that process is reactivated. Such processes normally return to higher layer components which perform device-dependent analysis of the data. If the I/O interrupt is not associated with an active I/O request, the process activates the device attention process. In addition, if the I/O interrupt indicates that a channel or device has become available, the I/O interrupt handler process attempts to restart the channel with a request queued for that channel.

6.2 Timer Interval Expired Process

This process performs the functions associated with the expiration of a predefined time interval. These functions include the activation of any processes which have requested their own suspension until the expiration of the time interval and the resetting of the timer for the next time interval. This process is activated by the hardware interrupt generated when the processor's interval timer expires.

6.3 Device Attention Process

This process performs detailed interpretation of the data provided by I/O interrupts which are not associated with an active I/O request. This process is activated by the I/O interrupt handler process when an "unexpected" I/O interrupt is received.

6.4 Data Bus Monitoring Process

This process examines the status of selected real time data sensors (e.g., analog inputs) at periodic time points. The data obtained is used to maintain a representation of the devices in main memory. These data values may subsequently be used to satisfy requests to read the real time data. This method expedites the processing of requests for real time data and reduces the request load on the channels associated with the monitored devices.

The process is activated by the timer interval expired process at the beginning of each monitoring interval.

6.5 Job Process

MOSS creates a job process for each job. The job process performs the functions required for initiating and terminating a job.

Each job consists of one or more tasks. The job process creates and activates the task process for the initial task of a job. In addition, the job process is activated whenever a task of the job terminates. This permits the job process to control sequential scheduling of tasks and to determine when the job has completed.

6.6 Task Process

A task process is created for every task which executes within the MOSS environment. A task process is created when a request is received by MOSS to execute a task and exists until the task is terminated.

The task process is created and activated by the job process if the task is the initial task of the job or if the user requests sequential scheduling of the tasks of his job. Otherwise, task processes are created and activated in response to a task scheduling request from another task of the same job.

6.7 Sampling Performance Monitoring Process

This process collects basic system performance data and enters it into the system log. The process is activated by the timer interval expired process at each sampling time point.

6.8 External Control Process

This process carries out the commands which are entered at the system console. The process is activated by the device attention process when an external control command is read from the system console device.

6.9 Log Management Process

This process outputs a system log buffer to the system log data set. The process is activated by a log queue management function whenever a system log buffer becomes full. This function is a process because the writing of the log buffer is asynchronous to the other processes in the MOSS environment.

7. CONCLUSION

The architecture of MOSS has proven to be successful in achieving the basic design goal of modifiability. The hierarchical relationship of the MOSS partitions has remained unchanged since the early phases of the design while the dynamic structure and partition interfaces have been augmented to satisfy additional user requirements.

The concepts of exclusive ownership of resources by partitions and information hiding within modules has permitted the internal design of the partitions to be modified without affecting the interfaces into the partition or the design of other partitions.

The concepts of strictly controlling process communication has permitted processes to be modified and added without affecting other MOSS processes. For example, the data bus monitoring process has been added since the original design.

8. ACKNOWLEDGEMENTS

As with many development efforts, the architecture of MOSS was not wholly conceived at one time. It was an evolutionary process of the ideas and effort of several people over an extended time. We especially wish to recognize the work of T. T. Schansman, G. P. Williams, L. C. Keller, J. R. Conway, and J. R. Bounds.

9. REFERENCES

[1] Coffman, E. G., Jr., Elphick, M. J. and Shoshani, A. System deadlocks. Computing Surveys 3, 2 (June, 1971), 67-78.

[2] Dijkstra, E. W. The structure of the "THE" multiprogramming system. Comm. ACM 11, 5 (May, 1968), 341-346.

[3] Liskov, B. H. A design methodology for reliable software systems. AFIPS Conference Proceedings, Vol. 41 (1972), 191-199.

[4] Liskov, B. H. The design of the Venus operating system. Comm. ACM 15, 3 (March, 1972), 144-149.

[5] Parnas, D. L. On the criteria to be used in decomposing systems into modules. Comm. ACM 15, 12 (December, 1972), 1053-1058.

[6] RCA Advanced Technology Laboratories. Outline of Functional Description of Central Processing Unit. Report No. SUMC-C-R-SP-002, Vol. 1, CEI Specification, January, 1975.

[7] Watson, R. W. Timesharing System Design Concepts, McGraw-Hill, 1970.