PROCESSOR/PROCESS MANAGEMENT
TASKS, EVENTS AND COMMUNICATION CONTROL,
SOFTWARE SUBSYSTEM DESCRIPTION
EXTENDED OPERATING SYSTEM, 3A PROCESSOR

CONTENTS PAGE

1. GENERAL 2
INTRODUCTION 2
REFERENCES 2
PROCESS/TASK PROGRAMS 2
2. SYSTEM CALLS 3
SYSTEM CALL OVERVIEW 3
3. PROCESS/TASK CREATION AND MANAGEMENT 3
A. Process/Task Definition 3
B. Task Creation 4
C. Task Management 5
D. Task Termination 5
4. EVENTS AND EVENT MANAGEMENT . . 5
A. Event Purpose 5
B. Event State 6
C. System Events 6
D. Event Routines 7
E. Event Macros 7
5. DISPATCHING 8
A. Task Dispatching (PIDENT DISPAT) . 8
B. Local Event Dispatcher (PIDENT EVTDIS) 8
6. INTERPROCESS COMMUNICATION . . . 8
A. Interprocess/Task Overview . . . 8
B. Intertask Communication and Miscellaneous Administration 10
7. GLOSSARY 10
Figures
1. EOS Kernel Hierarchy 11
2. EOS Functional Structure 12
3. Task States 13
4. Task Descriptor Layout 14
5. Event Flag and Mask Layout 15
6. EOS Call Structure 16
Tables
A. Abbreviations and Acronyms 17
B. Assembly Unit Identification 18

NOTICE
Not for use or disclosure outside the
Bell System except under written agreement

Printed in U.S.A.

1. GENERAL

INTRODUCTION

1.01 Processor/process management consists of several functions such as the creation and termination of tasks, allocating the processor to tasks which are ready to execute, coordinating task and intertask communication (event management). This section describes the process/task creation, dispatching, event management, intertask communication, and the supervisor call (SVC) structure as applied to the 3A Central Control (3A CC) and the Extended Operating System (EOS).

1.02 The specific reasons for reissuing this section are:

- Add text for state transitions of event flags
- Add text to include use of event flags
- Include the SWAP_EVENT macro
- Change the event flag states and mask example in Fig. 5
- Add program name changes to text and Table B.

Revision arrows are used to emphasize the more significant changes. Equipment Test Lists are not affected.

REFERENCES

1.03 The following Bell System Practices may aid in understanding this section:

SECTION	TITLE
254-300-120	3A Central Control, Theory of Operation, Common Systems
254-340-001	Extended Operating System, Overview, Software Subsystem Description, Extended Operating System, 3A Processor
254-340-014	Memory Protection and Organization, Software Subsystem Description, Extended Operating System, 3A Processor
254-340-031	Processor/Process Management, Interrupt Handling and Timer Management, Software Subsystem Description, Extended Operating System, 3A Processor
254-340-100	Introduction to 3A Language, 3A Processor Common Systems
254-340-102	Basic and Extended 3A Processor Instruction Set, 3A Processor Common Systems
254-340-106	Extended Operating System Macros and Glossary.

1.04 A list of abbreviations and acronyms used in this section is provided in Table A. For a more complete list of associated abbreviations and acronyms for the Extended Operating System 3A Processor, refer to Section 254-340-106. Assembly Unit Identification is provided in Table B.

PROCESS/TASK PROGRAMS

1.05 A system of programs and subprograms is accessed to provide the mechanism for process/task creation, event management, and interprocess communications. The major programs accessed to provide these functions are:

(a) The Message Manager (MSGMGR PR-4C153) program identification (PIDENT) forwards messages and replies and provides miscellaneous message administration. The MSGMGR PIDENT also implements SVC 2 and SVC 3.

(b) The Miscellaneous Process Control (PROCON PR-4C155) PIDENT provides miscellaneous process/task control and implements SVC 5.

(c) The Operating System Process Dispatcher (DISPAT PR-4C150) PIDENT returns control to an interrupted system routine, process/task, or to a new process/task, as appropriate. The PIDENT is entered via a GOTO DISPAT instruction.

(d) The Process Event Dispatcher (EVTDIS PR-4C151) PIDENT gives control to a specified event routine.

(e) The Lab Operating System Tables (LOSTAB PR-4C147) PIDENT provides the various ap-

plication parameters which define the system hardware and software configuration.◀

2. SYSTEM CALLS

SYSTEM CALL OVERVIEW

2.01 System calls provide the means for EOS or application tasks to request services from the EOS kernel functions. System calls are analogous to software interrupts in that the task making such a request is, in effect, interrupting itself and relinquishing control of the processer to another task. At the conclusion of the request, control may or may not be returned directly to the requesting task, depending on how many higher priority tasks have been placed on the ready list. Supervisor calls are implemented by use of EOS macros which expand into an SVC instruction. The system service call functional structure is described in this part; and the functional processing for each type of service call is described in the process management part to which it pertains, eg, event control, intertask communication, etc.

2.02 The EOS active system call macros provide the means of implementing SVCs. System macros are divided into ten major categories with each category representing a major functional element. The active system call macros include a supervisor call instruction as part of the expansion. The ten major categories of EOS macros are:

(1) Timer control

(2) Event control

(3) Current process control

(4) Interprocess communication

(5) Storage control

(6) External process control

(7) Input/output (I/O) control

(8) Maintenance

(9) General purpose

(10) File system.

The supervisor instruction in the active system call macros is in the form "SVC n" where n is equal to a number designating a specific index into a service transfer vector which is located at memory locations 40 through 5F (Hex) in the write-protected first 4K of store. Refer to Section 254-340-106 for the system macro descriptions and to Section 254-340-014 for the description of EOS memory organization.

2.03 The service transfer vectors as defined in LOSTAB are comprised of a series of entries each specifying a transfer vector entry number, a PIDENT, and an entry point. The transfer vectors (pertaining to this section) as described in LOSTAB are structured as follows:

(a) ▶***Transfer Vector for SVC 2:*** The main entry point for SVC 2 is SENDMGWG in PIDENT MSGMGR. Logic at SENDMGWG provides interprocess communication by sending or forwarding of messages and replies.

(b) ***Transfer Vector for SVC 3:*** The main entry point for SVC 3 is MSGSVC3 in PIDENT MSGMGR. The MSGSVC3 subroutine provides interprocess communication by testing or retrieving messages.

(c) ***Transfer Vector for SVC 4:*** The main entry point for SVC 4 is MWAIT in PIDENT PRSTAT. Process and event control is provided by logic at MWAIT by placing a process in the WAIT state until a set of specified events occurs.

(d) ***Transfer Vector for SVC 5:*** The main entry point for SVC 5 is PRC in PIDENT PROCON. Miscellaneous process control functions are provided by logic at PRC.◀

The functional descriptions of processing resulting from a supervisor call are described in the following paragraphs of this section.

3. PROCESS/TASK CREATION AND MANAGEMENT

A. Process/Task Definition

3.01 The basic operating structure of the EOS consists of system or application defined processes or tasks. A process is the execution of a series of programs with the order of execution being specified by a file of commands. A task is a program or execution module with all the necessary information to provide asynchronous execution. Since EOS is a task-oriented system, the term "task" will be used

when referring to a process or task, and differentiation between the two will be made only where necessary.

3.02 The EOS manages the control and availability of the system resources, the passing of control between the application and system task, and the flow of information between loosely coupled asynchronous application tasks. The EOS management is implemented by using two levels (or modules)—the EOS process level and the EOS kernel level. The kernel level consists of the following hierarchy (Fig. 1) of main functions:

(a) Dispatcher

(b) Interrupt handler

(c) Timer

(d) Intertask communicator.

The functions have been structured such that calls from one function to another can only be made inward. A function may skip levels. For example, the intertask communicator function can call on the dispatcher without calling on the timer, and without the timer, in turn, calling on the interrupt handler. The kernel also contains maintenance functions to control the rollback and restart of a task as well as initialization functions.

B. Task Creation

3.03 Tasks are created in two ways:

(1) During system initialization, tasks are created from LOSTAB.

(2) During a task execution, the executing task can create another task.

Both methods use the TASK macro to define and describe the task. The TASK macro identifies the task to the system and establishes the entry point for the task, its priority, the state the task is to be brought up in, and whether it is to be suspended.

3.04 The system process level (Fig. 2) tasks are controlled and executed in the same manner as application tasks. Currently the only function implemented as a process is the EOS command interpreter. All other functions run as tasks.

3.05 The EOS manages resources by moving tasks through various tasks states according to the task priorities and position in the ready list. Only one task may be in the RUNNING state at any given time. When each task is in the READY state, it is entered onto a ready list, which can be accessed through the use of system macros.

3.06 When a task is in the RUNNING state (executing) and an interrupt occurs, control of the processor is passed to the interrupt handler which determines which interrupt service routine will have control of the processor. When an interrupt occurs, the currently executing task is interrupted temporarily and placed in the INTERRUPTED state (Fig. 3) while the interrupt handler is executed. Upon completion of the interrupt processing, the interrupted task may be returned to either the RUNNING state or to the READY state. If the interrupt service routine readied a task of higher priority than the interrupted task, the interrupted task that was placed in the INTERRUPTED state will be moved to the READY state and the higher-priority task will be moved to the RUNNING state. The state of the interrupted task will be saved in the save area of the task descriptor (Fig. 4) memory location so that it may resume execution at the point where the interrupt occurred. However, if the interrupted task has the higher priority, the interrupted task will be returned to the RUNNING state to continue executing, and the newly readied task will be placed on the ready list.

3.07 When a running task makes a system call, the service request administrator services the system call and then passes control to the dispatcher. The dispatcher checks the ready list and takes action similar to that for the interrupt.

3.08 When an initialization occurs, the operating system processes a series of tasks defined to it by the TASK macros in LOSTAB. When the task is to be brought up in the READY state, the operating system will insert the task on the ready list ordered by priority level. From the ready list, the task will be dispatched and moved to the RUNNING state. Tasks created in the WAIT state may be made ready through satisfaction of the wait by a particular event or intertask communication. Tasks brought up in the SUSPENDED state can be made ready through the use of the RESUME macro.

3.09 A task may also be created by use of the TASK macro in an application program. A task created by executing a TASK macro in an application program is considered to be the "child" of the task

that activates it. The activating task is referred to as the "parent" task. The task is then activated by the ACTIVATE macro. The task may be brought up in either the READY or WAIT state. If brought up in the READY state, it will be added to the ready list. If brought up in the WAIT state, it will not be added to the ready list. When brought up in the WAIT state, it can be made ready through satisfaction of the wait conditions by an event or intertask communication.

3.10 Tasks brought up in either the WAIT or SUSPENDED state are resumed or awakened based on their system identification (ID). The ID is the mechanism for keeping track of tasks that are not on the ready list.

C. Task Management

3.11 Task management consists of several functions. These functions include the creation and termination of tasks, the allocation of the processor to tasks that are ready to execute, synchronization of tasks, and intertask communication.

3.12 Tasks created at system initialization may be activated either at system initialization or activated at some later time by an executing task. Tasks defined as asynchronous operations are capable of concurrent processing. However, only one task can be executing on the processor at a time. The processor management function of the operating system has mechanisms to determine which task is to be active (in the RUNNING state). All other tasks in the system are in one of several task states. Figure 3 illustrates these states graphically. The task states are:

(a) ***INACTIVE:*** A pseudo-state representing the condition of a nonexecuting task.

(b) ***HOLD:*** Intended to be the state into which a task will go upon activation until the needed resources are allocated to it. Currently, resource allocation is not done in this manner, and any task activated with the users specification of READY will go immediately to the READY state.

(c) ***READY:*** A task is ready when all conditions necessary for its execution have been satisfied but the processor is not available.

(d) ***RUNNING:*** The state into which a task progresses from the READY or INTERRUPTED state. When a task has been allocated processor time, it is in the RUNNING state.

(e) ***SUSPENDED:*** A task may be put into the SUSPENDED state when it is brought up during system initialization, or it may be suspended by itself or by another task. The task is essentially put to sleep (stopped) at the point of suspension and will be restarted at this point on execution of the RESUME macro.

(f) ***WAIT:*** A task is in the WAIT state when it cannot continue further processing until some other asynchronous event has completed processing.

(g) ***INTERRUPTED:*** When an interrupt occurs, the currently executing task is suspended temporarily and placed in the INTERRUPTED state while the interrupt handler is executed. Upon completion of interrupt-level processing, the interrupted task may either return to the RUNNING state or to the READY state. If the interrupted processing readied a task of higher priority than the interrupted task, the task in the INTERRUPTED state will move to the READY state and the higher-priority task will move to the RUNNING state. If the interrupted task has a higher priority than the interrupting task, the interrupted task will be returned to the RUNNING state.

(h) ***COMPLETED:*** The state into which a task is placed when it has completed processing (running) and while the operating system is performing table cleanup operations. From the COMPLETED state, the task is moved to the INACTIVE state by the cleanup operations.

D. Task Termination

3.13 A task is terminated by execution of the END_PROGRAM macro. A task may either terminate itself or terminate another task. END_PROGRAM places the terminated task in the COMPLETED state and disables any events for this task. If the terminated task has a "parent" task, the "parent" task is notified that the "child" task was terminated.

4. EVENTS AND EVENT MANAGEMENT

A. Event Purpose

4.01 An event is a signal to a process or task that a previously specified state change has oc-

curred. Each process and task has a unique set of 32 event flags, with 32 associated event mask bits and the ability to link a subroutine, called an event routine, to each event. Events are used to signal the:

- Completion of another process or task
- Completion of an I/O operation
- Completion of a specified time interval
- Receipt of an interprocess message
- Occurrence of certain maintenance actions.

Events provide the application program with the means to schedule the execution of a number of tasks, to overlap I/O operations with the execution of other statements in the task that initiate the operations, and to simulate the occurrence of a specified state change by setting its own events.

B. Event State

4.02 The state of an event flag is determined by its value held in a pair of 32-bit registers called STATE1 and STATE2. The state of an event flag (see Fig. 5) is represented by the corresponding pair (STATE1 and STATE2) of bits that represent the state of the flag settings. ➧Using (0,0) to represent (STATE1 and STATE2), the following state transitions are allowed for the event flags:

- (0,0) changes to (0,1)—Message retrieved or event set by SET_FLAG
- (0,1) changes to (0,0)—WAIT is satisfied
- (0,1) changes to (1,1)—Event routine entered
- (1,1) changes to (0,0)—WAIT satisfied following execution of event routine
- (1,1) changes to (0,1)—Another message received or a SET_FLAG occurred while the event routine was executing.

4.03 Upon the occurrence of an event, its state is changed from (0,0) to (0,1). If no event routine is associated with the selected flag (or if the event routine is disabled), the next WAIT satisfied by the selected event flag will change its state back to (0,0).

4.04 If an event routine is present and enabled, the state of the selected event flag will change to (1,1) when control is passed to the event routine by the local dispatcher. After the event routine completes execution, the next WAIT satisfied by the selected event flag will change its state back to the (0,0) state. If another event occurs while the event routine is being executed or before the next WAIT is satisfied, the state of the selected event flag is changed from (1,1) to (0,1).➧

C. System Events

4.05 Twelve of the 32 event flags are reserved for system use, thereby leaving each process or task 20 event flags. The event flags reserved for system use are:

FLAG (BIT)	SYSTEM USE
0	Maintenance
1	Maintenance
2	Maintenance
3	Maintenance
4	File system
5	Reply received
6	Request received
7 or 8	Unassigned
9	I/O error
10 or 11	Unassigned.

4.06 ➧Maintenance flags are used primarily during system and process initialization phases. Flags 0, 1, and 2 are used to perform initialization of all system I/O devices. Event flag 4 is used by the file system to provide an interface between the process-level and event-level portions of the various device handlers.

4.07 Event flags 5 and 6 are used exclusively for interprocess communication via messages. Before sending a message to another process, various parameters must be set with the MESSAGE macro. These include the REQUEST event to be set in the receiving process (defaults to event flag 6), whether a reply is expected, the REPLY event to be set upon

receiving the reply (defaults to event flag 5), and whether to wait for the reply. The message is sent to a handler that attaches the message to the receiving process' REQUEST list and sets the receiver's REQUEST event. If the receiving process wishes to reply to the message, the REPLY option must have been specified by the MESSAGE macro. The reply is sent with the SEND_REPLY macro. This results in the message handler attaching the reply message to the original sending process' REQUEST list and setting that process' REPLY event.

4.08 Event flag 9 is used to signal the occurrence of an I/O error. Tasks that use any of the I/O macros must be designed to respond to the setting of event flag 9. Associated with all I/O requests to the EOS file system is a COMPLETION event (defaults to event 5), which is set by the file system when the requested I/O operation has successfully completed. If the requested I/O operation fails, a message is sent from the file system to the process that requested the I/O. Event 9 is the REQUEST event and is set by the receiving process' REQUEST list. The receiving process must retrieve this message using either the RETRIEVE or the TEST_MSG macros.◀

D. Event Routines

4.09 Once a task relinquishes control of the processor, it can be rescheduled for execution only as a result of the occurrence of an event (the only exception is if the process was preempted by time slicing). When an event occurs, its associated task is readied for execution. At this point, the application program must decide if it needs to be notified by the operating system that an event has occurred or if it wishes to schedule its own detection of events.

4.10 The operating system notifies a task that an event has occurred via the use of event routines. Event routines are subroutines that are linked to event flags. When a readied task is dispatched by the operating system, the local dispatcher is given control if event flags are set.

4.11 Each task also has a 32-bit mask register which permits enabling and disabling of associated event routines. A mask bit of zero means the corresponding event routine is disabled, and a mask bit of one means the corresponding event routine is enabled (Fig. 5). Disabling an event does not prevent its event flag from being set. It only inhibits the execution of the event routine. Enabled event routines are processed in order of priority from the highest (event flag 0) to the lowest (event flag 31).

E. Event Macros

4.12 Several macros are provided to enable a task to manipulate the event flags. The macros and their functions are:

- DISABLE—Inhibits all event routines without altering the event mask.
- ENABLE—Reestablishes event routines subject to the current event mask.
- EVENTS—Assembles a block of data for use with the SET_FLAG and SET_MASK macros, ie, declares an event mask.
- EVENT_TV—Declares an event transfer vector.
- EXIT_TO—Directs the exit from an event routine. EXIT_TO serves as a GOTO providing a direct exit from an event routine rather than a return.
- GET_FLAG—Gets the current event flags and loads the event mask in a specified location.
- GET_MASK—Loads the current contents of the event mask into a specified location.
- SET_EVNT_ADR_—Specifies the event routine or transfer vector for use in event processing. The execution of this macro overrides any previous executions. Issuing this macro without operands removes any event routine specification.
- SET_FLAG—Sets or resets the specified event flags.
- SET_MASK—Enables (set) or disables (reset) specified events, ie, sets a new event mask.
- ▶SWAP_EVENT—Enables or disables the setting of an event flag in the current process (or task) whenever the process is placed in the READY state because a higher priority process was readied or the time slice ended for the current process.◀

A list of EOS macros is provided in Section 254-340-106.

5. DISPATCHING

A. Task Dispatching (PIDENT DISPAT)

5.01 The task state change handled by the dispatcher is the movement of a task from the READY state to the RUNNING state (Fig. 3). Previous paragraphs (3.05, 3.08, and 3.12) explain the various states a task can assume in the operating system and some of the movements between these states.

5.02 Each time control leaves a task because of an interrupt or SVC, the dispatcher is called after interrupt or SVC processing (Fig. 6). When another task has been made ready because of an interrupt or SVC processing, the dispatcher must determine if the interrupting task is of higher priority than the previously executing (running) task. When it is not, the previously executing task is restored to the RUNNING state and resumes executing at the point where the task was interrupted.

5.03 When a higher priority task has been readied, the dispatcher must save the state of the previously running (executing) task and return it to the ready list. The newly readied (higher priority) task is then moved from the READY state to the RUNNING state and is allocated the processor.

B. Local Event Dispatcher (PIDENT EVTDIS)

5.04 The local dispatcher receives control from the system dispatcher when the task having the highest priority on the ready list has at least one event flag enabled and is in the "message received" state (Fig. 5). The local event dispatcher functions as the prologue and epilogue for all event routines by examining the event flag register (located in the task or descriptor table). If an event flag is enabled and in the "message received" state, then the local event dispatcher performs the following:

(a) Stores the contents of the process or task program counter in the event routine save area

(b) Sets up the return address

(c) Transfers control to the specified event routine.

Upon completion of the event routine, control is returned to the local dispatcher via the return address. The task program counter is restored from the event routine save area before the event dispatcher processes the rest of the event flags. When all event flags have been processed, the event dispatcher transfers control to the task via the address in the task program counter located in the task descriptor table (see Fig. 6). Accordingly, each event routine functions as a normal subprogram with the local event dispatcher providing the necessary linkage.

5.05 Since tasks and event routines may be time-sliced (it did not complete execution during the time allocated by the processor and must wait for another time-slice to complete its execution), the local dispatcher may dispatch a higher priority task (event routine) before the execution of the interrupted task is again resumed. When a task is time-sliced, the task is returned to the ready list in the READY state. By saving the contents of the task and program counter upon exit and restoring it upon entry, tasks and routines may be interrupted as necessary. If the task has the highest priority of all tasks on the ready list, the local dispatcher returns control to the task and it resumes execution where it left off; or if an event has occurred for which an event routine exists, execution may start at the event routine. The local dispatcher checks the status of the event flag; and if the event routine has the higher priority, the event routine is executed.

6. INTERPROCESS COMMUNICATION

A. Interprocess/Task Overview

6.01 Two general intertask communication facilities are supported by EOS: messages and events. Most of the EOS functions are implemented as a set of tasks. Included in this set are the task creator, file system manager, and terminal administrator. The system services are obtained by using the appropriate system macro which, at execution time, causes a message to be sent to its associated system task. Most application program functions are also implemented as a set of tasks. The only way these tasks communicate with each other is via messages and/or events.

6.02 Events are used for intertask communication when no message text is required to be transferred between tasks. As long as tasks know the meaning associated with each event, it is sufficient for one task to set an event flag for use by another task. The receiving task must correctly interpret the

setting of the event flag. The use of event flags for intertask communications requires much less use of system resources than the use of message text. Message text is used only when text must be communicated between tasks. (Event management is described in Part 4.)

6.03 The following system macros are used by EOS and application programs for intertask communication:

- MESSAGE—Sets up a message header and declares a message buffer
- NBR_MSG—Returns the number of messages received and not retrieved, and returns the number of outstanding messages
- RETRIEVE—Copies the message number into a specified buffer location
- SEND_MSG—Sends a message in a specified buffer to a specified task
- SEND_REPLY—Sends a reply in a specified buffer to the originating task
- TEST_MSG—Searches the list of messages for a message to the current task and if a message exists it is retrieved
- FORWARD—Takes the message originally retrieved in the buffer and forwards it to the specified task
- END_MESSAGE—Delimits the end of message buffer declared by the MESSAGE macro
- MSG_STATUS—Determines message status.

6.04 Before messages can be sent or retrieved, a message buffer must be established in writable user address space (ie, in a DATASECT). The message buffer must have a 5-word message header. The MESSAGE macro is used to set up the message header and to declare the length (number of words) of the message buffer. The buffer end (delimiter) is delcared by the END_MESSAGE macro. The message text follows the message header.

6.05 Once a message buffer has been declared by the MESSAGE macro, the message can be sent to another task using the SEND_MSG macro or retrieved from another task using the RETRIEVE or TEST_MSG macros. The parameters for the SEND_MSG macro specify the address of the message buffer and the system ID of the receiving task. The system ID is preassigned at system initialization using the system operating tables in LOSTAB. In order to send a message to a task, the task must have been defined during system initialization.

6.06 The RETRIEVE and TEST_MSG macros are used to copy specified message from system dynamic memory into the specified message buffer. The MESSAGE macro must have previously declared the message buffer before a message can be copied into the buffer. The MESSAGE macro sets up the message header and specifies the number of words (length) of the message buffer. This provides the means whereby the RETRIEVE and TEST_MSG macros may perform range checking. (The number of words in the retrieved message cannot exceed the number of words declared for the message buffer.) A message will not be copied if it is larger than the message buffer. RETRIEVE will copy the first, last, or nth message from the current task message list. The TEST_MSG macro will search the list of messages to the current task for one from another task or one on an event and will return its ordinal position in the message list. If the MSG parameter (address of buffer declared by the MESSAGE macro) is specified, the TEST_MSG macro will copy the message from system dynamic memory into the addressed message buffer.

6.07 The SEND_MSG macro results in an SVC 2 to the kernel. After the receiving task is located, the message is copied from the sending task address space into a message block in system dynamic memory. The message block is then placed on the bottom of the receiver message list (ie, it is queued on the REQUEST list of the receiver task descriptor block). The send event (specified by the sender MESSAGE macro) is set causing the receiving task to be readied for execution.

6.08 The system ID must be known before a message can be retrieved from a given task. A common form of intertask communication is to send a message designating the receiver task. SEND_MSG will result in the event flag being set in the receiver task and the receiver task being readied for execution. After receiving control as a result of the event flag being set, the receiver task can then retrieve all

messages that are associated with that event associated with the task.

B. Intertask Communication and Miscellaneous Administration

6.09 Miscellaneous administration of intertask communication is accomplished by application and EOS routines making the appropriate macro calls. Four macros are available to perform this service: RETRIEVE, TEST_MSG, NBR_MSG, and MSG_STATUS. All of these macros expand into an SVC 3 which sets a modifier. ➧The SVC 3 transfer vector in the LOSTAB causes a transfer to the MSGMGR program at entry point MSGSVC3. The MSGMGR saves the system–state and error checks for any incorrect modifiers.➧ When an incorrect modifier is detected, a return is made to the dispatcher. The modifier is decoded and this value causes control to be transferred to one of four intertask communication administrative programs.

7. GLOSSARY

7.01 The following terms and definitions are used in this section to describe the creation of tasks, events and event management, and intertask communications.

Address—A combination of bits that identifies a location in a storage device or equipment unit.

Application—A set of functional system programs which use the services of the Extended Operating System.

Assembly Unit—A collection of codes that is assembled or compiled as one entity. The assembly unit is the highest level of a modular program structure and may or may not contain functionally related subunits.

Bit—The binary unit of information which is represented by one of two possible conditions; such as the digits 0 and 1 or on and off.

CSECT—Pseudo-operation used to specify the beginning of a sequence of instructions.

DATASECT—A block of relocatable data, ie, a data CSECT.

Disable—Inhibits the event routine without altering the mask.

Enable—Establishes event routines subject to the current mask.

Entry—A labeled location at which a CSECT or any logical block of code may be entered.

Epilog—A function of the local event dispatcher in restoring the program counter from the event routine save area after an event flag has been processed.

Event—A signal to a task that a previously specified state change has occurred.

Flag—A set of bits used to signify the state of an event.

Initialization—An action taken to provide the system with a known good operating configuration.

Interrupt—A hardware generated signal that causes the task currently executing to be preempted and a higher priority task to be performed immediately.

LOSTAB—Operating system tables used by the application programs to define system resources, configuration, parameters, etc.

MACRO—A sequence of operations labeled with abbreviated notation. A macro may generate different sequences of code depending on the parameters supplied in the macro call.

Mask (Event)—A 32-bit register which is ANDed with the current event flags.

PIDENT—Program identifier is the name by which an assembly unit is identified.

Process—The execution of a series of programs with the order of execution specified by a file of commands.

Program Unit—A collection of codes within an assembly unit which performs a well defined function.

Prolog—A function of the local event dispatcher in storing the program counter in the event routine save area prior to servicing an event flag.

Routine—A sequence of instructions called from within another section of instructions to perform a specific function.

Symbol—A group of characters which represent a fixed value or address or structure.

Task—A related group of programs and routines which perform a defined function and are run asynchronously based on a priority system. The task defines all resources required to accomplish the specific function.

Transfer Vector—A series of data words collected in an intermediate location which are pointers to destination locations.

4K Block—4096 consecutive memory words (1K=1024).

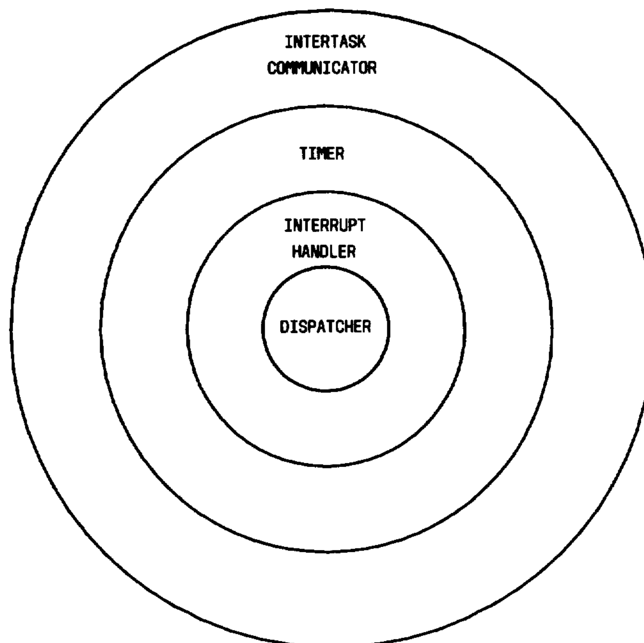


Fig. 1—EOS Kernel Hierarchy

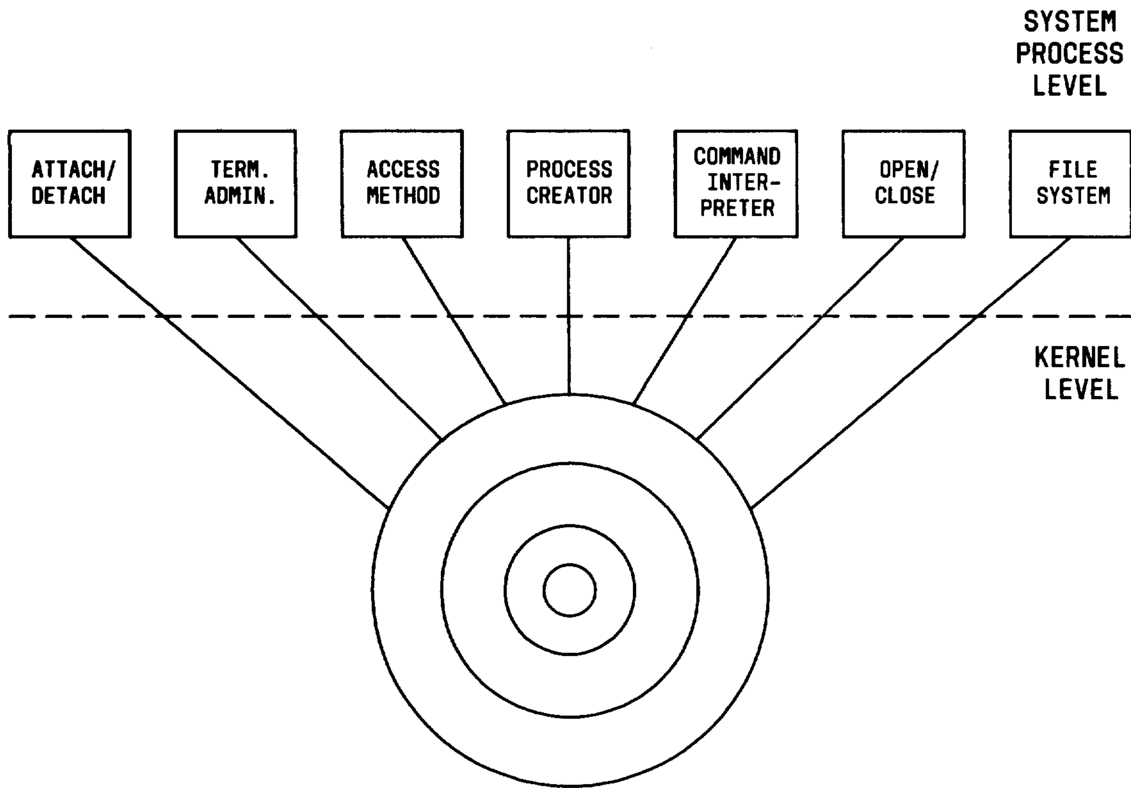


Fig. 2—EOS Functional Structure

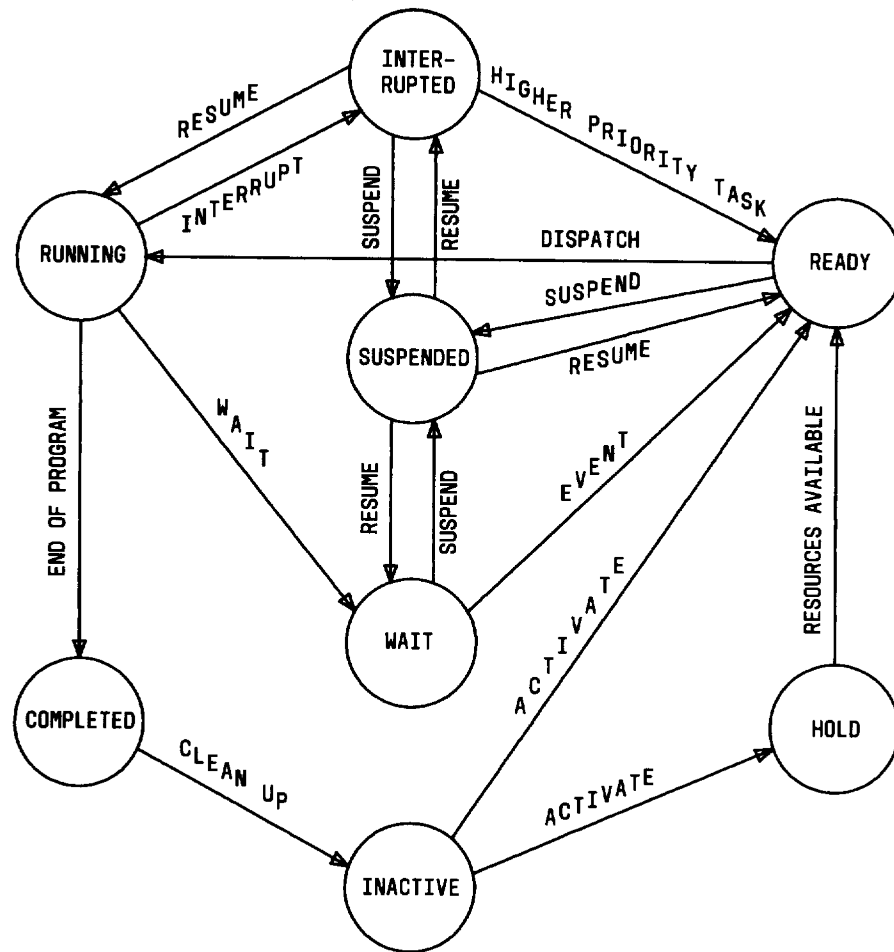


Fig. 3—Task States

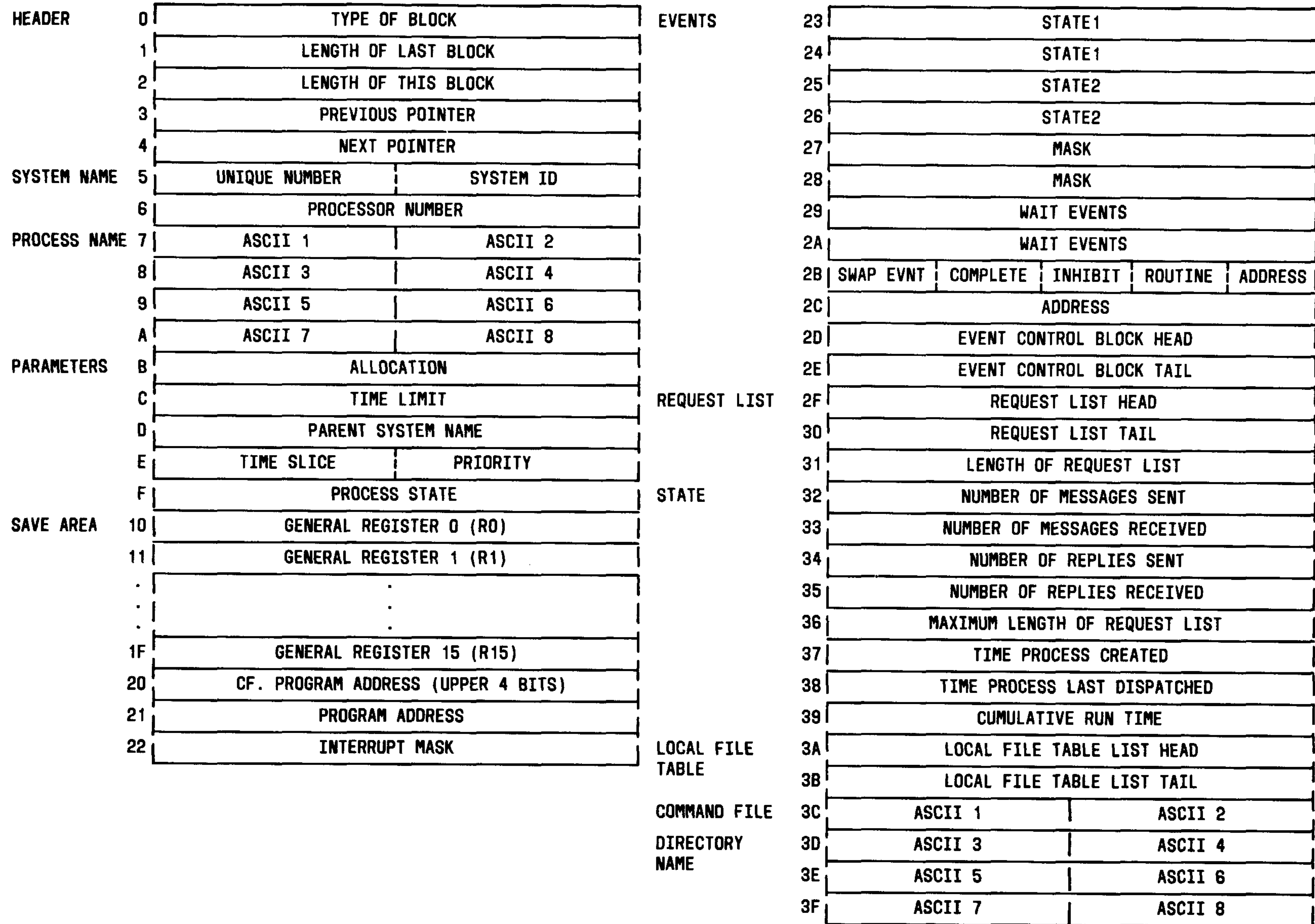
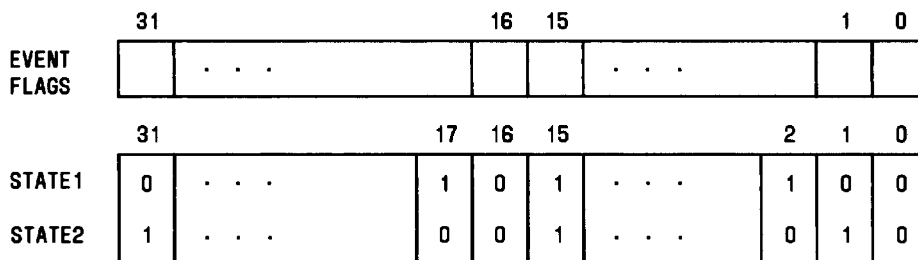
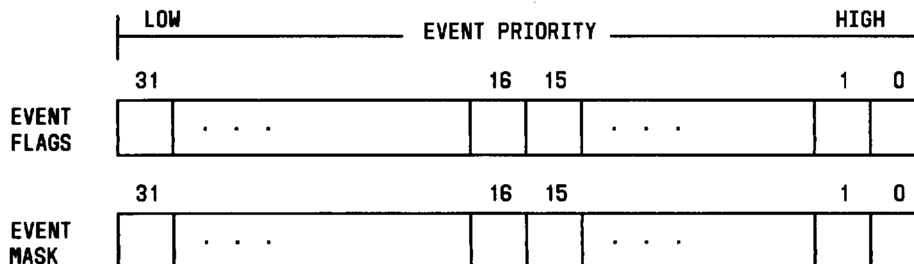


Fig. 4—Task Descriptor Layout



EACH OF THE 32 EVENT FLAGS MAY HAVE ONE OF THE THREE FOLLOWING STATES:

STATE1	STATE2	EVENT FLAG EXAMPLE	STATE
0	0	0,16	NOTHING HAPPENED
0	1	1,31	EVENT OCCURRED
1	1	15	EVENT ROUTINE ENTERED



EXAMPLE:

EVENT MASK SETTING	EVENT ROUTINES	EVENT ROUTINES ARE:
ALL 0 s	0-31	DISABLED
ALL 1 s	0-31	ENABLED

Fig. 5—♦Event Flag and Mask Layout♦

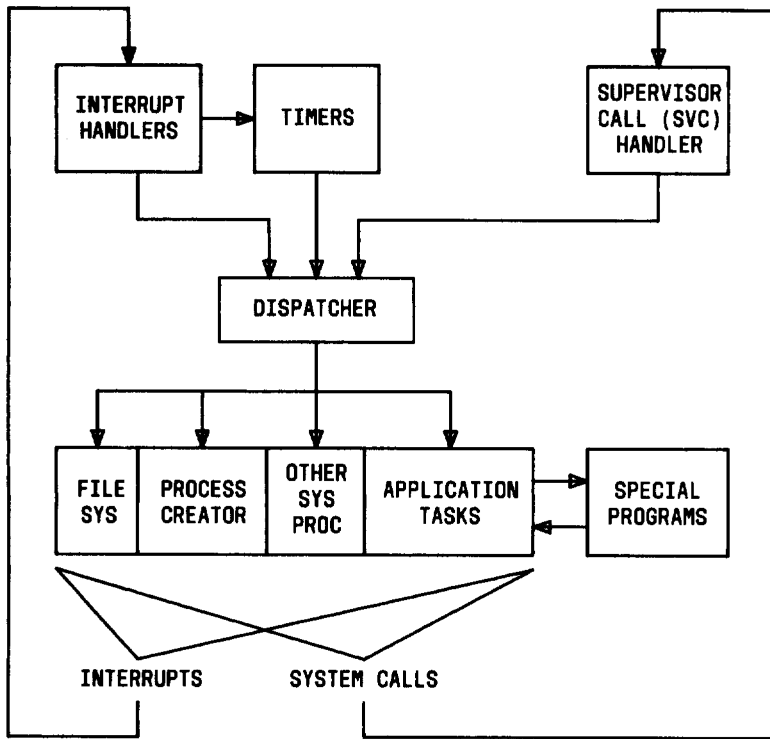


Fig. 6—EOS Call Structure

TABLE A

ABBREVIATIONS AND ACRONYMS

ABBREVIATION/ACRONYM	TERM
CC	Central Control
EOS	Extended Operating System
I/O	Input/Output
ID	Identification
K	One Thousand (Actually 1024 Memory Locations)
PIDENT	Program Identification
PR	Program Listing
SVC	Supervisor Call

TABLE B

ASSEMBLY UNIT IDENTIFICATION

NAME	TITLE	PR NUMBER
AUDIT	KERNAL AUDIT	PR-4C157
CONVTD	OPERATING SYSTEM DATA CONVERSION ROUTINES	PR-4C149
DEVATT	INITIALIZE PERIPHERY CREATE DEVICE TABLES	PR-4C104
DISPAT	OPERATING SYSTEM PROCESS DISPATCHER	PR-4C150
EVTDIS	PROCESS EVENT DISPATCHER	PR-4C151
EVTMGR	EVENT MANAGER	PR-4C152
INTRPT	INTERRUPT HANDLING PROGRAM	PR-4C148
INTSRV	INTERRUPT SERVICE ROUTINE	PR-4C113
LOSTAB	LAB OPERATING SYSTEM TABLES	PR-4C147
MAICCI	EOS INITIALIZATION PROGRAM	PR-4C605
MSGMGR	MESSAGE MANAGER	PR-4C153
PCRTRM	PROCESS CREATOR AND TERMINATOR	PR-4C154
PROCON	MISCELLANEOUS PROCESS CONTROL	PR-4C155
PRSTAT	PROCESS STATE TRANSITION MANAGER	PR-4C156
SYSMM	SYSTEM MEMORY MANAGEMENT	PR-4C141
TCONAU	CONVERT BINARY TIME TO CHARACTER	PR-4C142
TIMEAU	CONTROL SYSTEM INTERVAL TIMING IN THE SYSTEM	PR-4C144