CLASS: ps
Table of Contents

Functional Description of ps

Process synchronization exists as a monitor of process management to do the following two tasks:

- o maintain the integrity of shared data
- o synchronize the execution of parallel processes

The basic building block of process synchronization is the notion of WAIT and SIGNAL. These two constructs are the mechanism by which one process may communicate with another. Moreover, these constructs are very useful if a process must wait for a communication from another process without being in execution. This need for inter-process communication has given rise to the abstract notion of a condition. A condition can be thought of as a queue. Two operations may be performed on a condition, namely, "WAIT" and "SIGNAL". When a process executes a SIGNAL function, the process enqueued at the head of the queue is again made eligible for execution (i.e., the process is enabled). If there are no processes enqueued on the condition when the SIGNAL function is executed, no action is taken (i.e.,the signal is forgotten). Another use for these conditions is in the notification of events. For example, if a process has reached a point in its execution where it cannot continue until some subsequent event occurs, it can execute a WAIT function on a condition that has been associated with that event. When the event occurs, another process SIGNALS the same condition to let the first process know that the event has occurred.

If the important thing is not that some event occurred after a process WAITed on the associated condition but that it occurred at all, then this mechanism is not sufficient. To handle this case a semaphore facility was implemented. A semaphore is a specialized condition that has a count associated with it. This count is used to remember the SIGNALS that occur when no process is enqueued on the condition. Three operations may be performed on a semaphore: P-op, V-op, and I-op. When a process executes a P-op function the count is looked at and if it is greater than zero, this indicates that there have been more SIGNALS than WAITS performed. In this case the count is decremented by one and the process continues in execution. If the count is less than one, then there are no events being remembered and the count is decremented by one and the process is taken out of execution and enqueued on the condition.

When a process executes a V-op function the count is looked at and if it is negative it is incremented by one and a SIGNAL operation is executed on the condtion. If the count is not negative it is simply incremented by one and no further action takes place.

When a process executes a I-op function the count is looked at and if it is greater than zero, it is decremented by one and the process continues executing knowing the event had occurred. If the count is not greater than zero, no action is taken and the process continues in execution knowing that the event has not yet occurred.

Note that to prevent unauthorized WAITS and SIGNALS on any arbitrary condition, some process must initiate (request) a condition. The initiating (requesting) process is returned a condition identifier (CID) which is a secure token(which really is an illegal T=15 descriptor). The CID must be presented whenever a P-op, V-op or I-op function is to be performed on the condition.

Besides the abstract concepts of a condition and a semaphore, several others have been developed, namely, monitor, critical section, software interrupt, and message semaphore. A monitor is a set of data shared among multiple processes and a set of procedures which are the only procedures permitted to access these shared data. The procedures of the monitor may each have their own private data. The only other data they may access are the parameters passed when the procedure is called.

Only one procedure of a monitor may be executed at a time. If a subsequent call to a monitor occurs while one of its procedures is in execution (by any process), that request must be delayed until the current executing process exits the monitor. In this way potential conflicts resulting from multiple accesses to monitor data are avoided.

Each procedure of a monitor is designed and implemented so as to maintain the data invariant of the monitor. In addition, every monitor has an initialization procedure executed on every monitor start or restart (monitor creation) which establishes the invariant before any calls are performed in normal usage.

Two types of monitors have been defined:

loop-type o access to the monitor procedures in this type of monitor is controlled by a loop gate. When two processes attempt to call a monitor procedure, one of the processes loop on the monitor gate until the other process exits the monitor. This monitor must also be in a type-1 critical section (defined below).

queue-type o access to the monitor procedures in this type of monitor is controlled by a loop gate and a non-message semaphore. When two processes attempt to call the same monitor, one of the processes is enqueued on the semaphore until the other process exits the monitor. This monitor must also be at least in a type-2 critical section (defined below).

Both loop-type and queue-type monitors may execute a WAIT. For a loop-type monitor the monitor is exited and the process is enqueued on thespecified condition. For a queue-type monitor the process is enqueued on the specified conditon and the monitor is exited by that process. If there is another process waiting on the monitor, it is signalled.

A critical section is a state that a process can enter which defines certain limits to the conditions under which the process will give up control of the processor on which it is executing.

Two types of critical sections have been defined:

type-1 o the process cannot give up cortrol of the processor to another process nor can it allow software interrupts to occur for this process. The process can only give up the processor by executing a WAIT function. A type-1 critical section is implemented by using the inhibit interrupt feature of the hardware.

type-2 o the process can relinquish the processor to other processes but re-dispatch to the process must be to the point of interruption within the type-2 critical section. That is, the process cannot be aborted or have software interrupts or courtesy calls paid to it, or allow any other exception processing to occur until it has exited the type-2 critical section.

A software interrupt is a mechanism by which a process can be interrupted by another process. When performed a software interrupt forces the execution of the target (to be interrupted) process to be continued at the specified "interrupt handling routine". The process number (KPX) of the target process and the entry descriptor to the interrupt handling routine must be supplied by the user. The software interrupt is paid via the courtesy call mechanism from within the dispatcher. The user should note that a software interrupt for a swapped process or one within a type-2 critical section will be queued and not paid until the process is swapped back into core or the process has exited the type-2 critical section.

Mission Description   Runoff: 03/05/79

A message semaphore is a semaphore which has associated with it a two-word message. The message is passed in the AQ with the invoked VMSEM macro and it is returned in the AQ from the invoked PMSEM or TMSEM macro if an event has occurred.

The monitor maintains several lists in order to keep track of processes. At node initialization time there are three lists which have entries on them:

- o Free Conditions (CON)
- o Free Entry Definitions (ENT)
- o Assigned Process Definitions (PRD)

In addition there exists two empty lists:

- o Assigned CON's
- o Assigned ENT's

Usage Information of ps

I. Overview and Introduction

There are a number of macros which when used in conjunction with process synchronization provide users with an easy way of using both loop-type and queue-type monitors. These macros can be executed in either slave or master mode. The following pages will describe each macro by giving a brief description of the macro followed by its argument requirements and any other notes the user of the macro should be cognizant of. Note, that only the argument names are given when the macro is described below. However, a glossary containing the meaning of each argument is included following the last macro description.

II. ICOND - Initialize Condition

A. Description

This macro is providied to initialize conditions for the user. The condition identifer (CID) returned to the user (in the descriptor space provided by the user) is a secure token. This token takes the form of a T=15 descriptor (an illegal type) which contains in the descriptor base field the protected data. This descriptor cannot be shrunk and therefore cannot be modified. This descriptor can be used by other macros to coordinate processes by doing WAITs and SIGNALs.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
QTYPE	3	Optional
FLAGS	4	Reserved
COUNT	5	Optional
CIDOFF	6	Optional

C. Notes

Index register zero (X0) is destroyed.

III. SIGNL - Signal Condition

A. Description

This macro is provided to allow a process to notify another process(es) that some event has occurred. If there are no process(es) waiting on the condition, the signal is lost (i.e., the event is not rembered). If desired, a process may signal a condition and specify that all processes waiting on the condition are to be signalled rather than just the one on top of the waiting queue.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
REASON	3	Optional
BRDCST	4	Optional
CIDOFF	5	Optional

C. Notes

Accumulator register (AR) is destroyed. This function should be invoked from within a monitor.

IV. NTRLM - Enter Loop Monitor

A. Description

This macro is used to enter a loop-type monitor. It shuts the gate specified by the user and increments the loop monitor count in the Process Control Block (PCB). The user must insure that interrupts are inhibited while executing within a loop-type monitor.

B. Argument List

NAME	NUMBER	USAGE
GATOFF	1	Optional
IRMOD	2	Optional
GATSEG	3	Required
PCBSEG	4	Required

C. Notes

Index register zero (X0) is destroyed.

V. XITLM - Exit Loop Monitor

A. Description

This macro is used to exit a loop-type monitor. It opens the gate specified by the user and decrements the loop monitor count in the PCB.

B. Argument List

NAME	NUMBER	USAGE
GATOFF	1	Optional
IRMOD	2	Optional
GATSEG	3	Required
PCBSEG	4	Required

C. Notes

Index register zero (X0) is destroyed.

VI. WAITLM - Wait From Loop Monitor

A. Description

This macro is provided to allow the user to perform a WAIT function on a condition from inside a loop-type monitor. If the condition was initiated with a priority queue the priority may be specified by the user. A timer may also be specified which will cause the process to be placed into execution if it has not been signalled within that time. The WAIT function will open the loop monitor gate specified by the user so that the process will no longer be in the loop monitor when it resumes execution.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
PRIOL	3	Optional
TIMER	4	Optional
CIDOFF	5	Optional
GATOFF	6	Optional

C. Notes

Accumulator and quotient registers (AQ) are destroyed. This function should be invoked only within a loop monitor.

VII. ISEM - Initialize Semaphore

A. Description

This macro is provided to initialize a semaphore. Arguments can be provided which specify the attributes of the condition and the initial value of the count can be specified. Since the count is a shared data item it must be in a loop monitor and the gate for this monitor is in the word before the count. The gate word must be on an even word boundary.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
SEMSEG	3	Required
ICOUNT	4	Optional
QTYPE	5	Optional
FLAGS	6	Reserved
CIDOFF	7	Optional
GATOFF	8	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed.

VIII. TSEM - Terminate Semaphore

A. Description

This macro is used to delete a semaphore. The CID must be returned to the system and the CID is made invalid.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
CIDOFF	3	Optional

C. Notes

Accumulator register (AR) is destroyed.

IX. PSEM - P-op Semaphore

A. Description

This macro performs a P-operation on the semaphore specified by the user. If an event has occurred the count is decremented and the process will continue in execution. If an event has not occurred, the process will be taken out of execution and enqueued on the condition until it is signalled or until the timer specified by the user has elapsed.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
ARGD	2	Required
ARGBD	3	Optional
SEMSEG	4	Optional
PRIOL	5	Optional
TIMER	6	Optional
CIDOFF	7	Optional
GATOFF	8	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed. There are two exits from this macro, EXIT #0 and EXIT #1 (defined below).

X. VSEM - V-op Semaphore

A. Description

This macro performs a V-operation on the semaphore specified by the user. If any process(es) are enqueued on the condition, the one at the front of the queue will be signalled. If there are no process(es) enqueued, the event will be remembered by incrementing the count field.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
ARGD	2	Required
ARGBD	3	Optional
SEMSEG	4	Optional
REASON	5	Optional
CIDOFF	6	Optional
GATOFF	7	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed. There are two exits from this macro, EXIT #0 and EXIT #1 (defined below).

XI. TSTSEM - Test Semaphore

A. Description

This macro provides the user with the ability to test if an event has occured. If so, the count field is decremented by one. If no events occurred, the process does not wait but continue in execution.

B. Argument List

NAME	NUMBER	USAGE
GATOFF	1	Optional
IRMOD	2	Optional
GATSEG	3	Required
PCBSEG	4	Required

C. Notes

Accumulator register (AR) and index register zero (X0) are destroyed. There are two exits from this macro, EXIT #0 and EXIT #1 (defined below).

XII. INITQM - Initialize Queue Monitor

A. Description

This macro initializes a queue-type monitor. Since a queue monitor is realized with a semaphore, this macro simply initializes a semaphore with an initial count of one. This semaphore can then be used for queue monitor operations.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
SEMSEG	3	Required
QTYPE	4	Optional
FLAGS	5	Reserved
CIDOFF	6	Optional
GATOFF	7	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed.

XIII. TERMQM - Terminate Queue Monitor

A. Description

This macro terminates a queue-type monitor. This is simply a termination of the semaphore used by the monitor. After this is executed, no more queue monitor operations may take place using this semaphore.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
CIDOFF	3	Optional

C. Notes

Accumulator register (AR) is destroyed.

XIV. NTRQM - Enter Queue Monitor

A. Description

This macro allows the user to enter a queue-type monitor. If the monitor is busy, the process will be taken out of execution and enqueued on the condition associated with the queue monitor semaphore. When the monitor becomes available (via XITQM) the condition will be signalled and the process will be placed into execution. This macro also places the process into a type-2 critical section and increments the count field in the PCB.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
ARGD	2	Required
ARGBD	3	Optional
SEMSEG	4	Required
PRIOL	5	Optional
TIMER	6	Optional
CIDOFF	7	Optional
GATOFF	8	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed. Because this macro invokes the PSEM macro its exits are the same as those for the PSEM macro.

XV. XITQM - Exit Queue Monitor

A. Description

This macro allows the user to exit a queue-type monitor. A V-operation is performed on the monitor semaphore and if some process(es) are enqueued, the one at the front of the queue be signalled. The user process exits the type-2 critical section.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
ARGD	2	Required
ARGBD	3	Optional
SEMSEG	4	Optional
REASON	5	Optional
CIDOFF	6	Optional
GATOFF	7	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed. Because this macro invokes the VSEM macro its exits are the same as those for the VSEM macro.

XVI. WAITQM - Wait From Queue Monitor

A. Description

This macro allows a user to wait on some event from a queue-type monitor. A V-operation is performed on the monitor semaphore and if some process(es) are enqueued, the one at the head of the queue is signalled. The user process then performs a WAIT function on the condition specified for the event. It is important to note that the type-2 critical section is not exited until the process is placed back into execution after the WAIT has been broken. Therefore, the process cannot be interrupted by a software interrupt while waiting in this situation. After this macro is finished, the process is no longer within the queue monitor.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
SARGD	2	Optional
SARGBD	3	Optional
SEMSEG	4	Optional
REASON	5	Optional
WARGD	6	Required
WARGBD	7	Optional
PRIOL	8	Optional
TIMER	9	Optional
SCIDOF	10	Optional
WCIDOF	11	Optional
SGATOF	12	Optional
WGATOF	13	Optional

C. Notes

Accumulator and quotient registers (AQ) and index register zero (X0) are destroyed.

XVII. NTR2CS - Enter Type-2 Critical Section

A. Description

This macro enters a type-2 critical section. A nesting count of type-2 critical sections is kept in the PCB and is incremented. While in a type-2 critical section, a process will be re-dispatched only at the point of interruption. Thus, all other events, (such as, termination, exception processing, software interrupts, courtesy calls, etc.) are delayed until the process is no longer within the type-2 critical section.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required

C. Notes

Accumulator register (AR) is destroyed.

XVIII. XIT2CS - Exit Type-2 Critical Section

A. Description

This macro exits a type-2 critical section. The nesting count of type-2 critical sections in the PCB is decremented.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required

C. Notes

Accumulator register (AR) is destroyed.

XIX. IMSEM - Initialize Message Semaphore

A. Description

This macro is provided to initialize message semaphores for the user. A condition is requested and the semaphore gate is opened. The user is allowed to define the queueing strategies for the process and the messages, where either can be queued FIFO, LIFO or based on some priority.

B. Argument List

NAME	NUMBER	USAGE
ARGD	1	Required
ARGBD	2	Optional
SEMSEG	3	Required
FLAGS	4	Reserved
PQTYPE	5	Optional
MQTYPE	6	Optional

C. Notes

Accumulator and quotient registers (AQ), operand descriptor register zero (ODR0) and index regiter zero (X0) are destroyed.

XX. PMSEM - P-op Message Semaphore

A. Description

This function performs a P-operation on the message semaphore specified by the user. If an event has occurred the two-word semaphore message is put into the AQ for the user. If no event occurred, a WAIT is performed. If the WAIT is broken by a reason other than a SIGNAL no semaphore message is returned in the AQ.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
ARGD	2	Required
ARGBD	3	Optional
SEMSEG	4	Required
PRIOL	5	Optional
TIMER	6	Optional

C. Notes

Accumulator and quotient registers (AQ), operand descriptor register zero (ODR0) and index regiter zero (X0) are destroyed. The message queue index registers LPRIOR, LNEXT and LCURR (defined below) are updated.

XXI. TMSEM - Test Message Semaphore

A. Description

This function performs a test on the message semaphore specified by the user. If an event has occurred the two-word semaphore message is put into the AQ for the user and the semaphore count is decremented by one. If no event occurred, no mesage is returned and the process does not wait but continue its execution.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
SEMSEG	2	Required

C. Notes

Accumulator and quotient registers (AQ), operand descriptor register zero (ODR0) and index regiter zero (X0) are destroyed. The message queue index registers LPRIOR, LNEXT and LCURR (defined below) are updated.

XXII. VMSEM - V-op Message Semaphore

A. Description

This function performs a V-operation on the message semaphore specified by by the user. If any process(es) are enqueued on the condition, the one at the front of the cueue is signalled and the two-word semaphore message is passed to the waiting process. If no process is waiting, the event is remembered and the two-word message is stored into the next available message queue.

B. Argument List

NAME	NUMBER	USAGE
PCBSEG	1	Required
ARGD	2	Required
ARGBD	3	Optional
SEMSEG	4	Required
MPRIO	5	Optional

C. Notes

Accumulator and quotient registers (AQ), operand descriptor register zero (ODR0) and index regiter zero (X0) are destroyed. The message queue index registers LPRIOR, LNEXT and LCURR (defined below) are updated.

---

glossary of ps

This section gives a brief description of the terms contained in the argument lists for the preceding macros.

EXIT #0 o this is the first instruction following the macro call. Return to this location implies that the invoked PMME was executed successfully.

EXIT #1 o this is the second instruction following the macro call. Return to this location implies that the invoked PMME was not executed successfully. It is the responsibility of the macro user to interpret the meaning of the return code as returned from the invoked PMME.

ARGD o this operand descriptor register (ODR) frames the arguments for the invoked PMME. This argument must not be in ODR0 if ARGBD is not specified and any other optional arguments are specified. ARGD will frames two or three descriptors/vectors depending upon the invoked PMME, the first of which must be ARGBD.

ARGBD o this ODR contains the descriptor which frames the argument block for the invoked PMME. If this argument is not specified it will be loaded into ODR0 from the first descriptor/vector framed by ARGD if any other optional argument is specified. If none of those arguments are given and ARGBD is not specified, it is assumed that the user has already initialized the argument block. Depending upon the function, the argument block can be either 3, 4 or 5 words with the first two words containing return codes. The remaining words contain the request data for the invoked PMME.

QTYPE o a literal which describes the queueing strategy to be maintained for processes waiting on a condition. The possible literal argument can be 'F', 'L' or 'P', which respectively indicates a First-in,First-out (FIFO), Last-in,First-out (LIFO) or priority queueing strategy to be used. If not specified, FIFO is assumed.

FLAGS o this argument is reserved for compatability and is no longer relevant to the user.

COUNT o this is the number of CID's being requested and if not specified will be set to one.

CIDOFF o this is the offset which specifies where in the segment the CID can be found/saved. If not specified it is assumed to be zero. If specified this argument can have the form <constant> or the form (<constant>,<IRmod>) where either part is optional.

REASON o this is the code that can be passed to a signalled process to let it know why it was signalled. The range for the reason code is 0 <= REASON <= 2047. If not specified it is set to zero. If specified the argument can be of the form <constant> of the form (<constant>,<IRmod>) where either part is optional.

BRDCST o if specified this argument must be the literal 'B'. If given all processes currently waiting on the specified condition will be signalled.

GATOFF o this is the offset which specifies where in the segment the semaphore gate is found. If not specified it is assumed to be zero.

IRMOD o this is the index register modification to be applied during address development of the semaphore gate. If not specified no index register modification is applied.

GATSEG o this is the ODR modification that is used to reference the semaphore gate.

PCBSEG o this is the ODR modification that is used to reference items within the Process Control Block (PCB).

PRIOL o this is the priority level (0-63) with which the process will wait on the specified condition. This argument is assumed to be zero if it is not specified. If the condition was not requested with a priority, it will be ignored.

TIMER o this is the time limit (in milliseconds) beyond which the process is unwilling to wait for a SIGNAL. This argument can take either of three forms: (i) null, in which a default timer is used, (ii) "MAX", in which a timer of 30 bits all set to one (the largest allowable value) is used, and (iii) a symbolic location/constant. (III) can further be specified in either of the following forms: <constant>, (<constant>,<IRmod>) or (<constant>,<IRmod>,<ODRmod>) where either part is optional.

SEMSEG o this ODR frames the segment containing the gate and count field to be used by the semaphore. The gate is at the location specified by GATOFF and the count field is at the location plus one. This argument must not be in ODR0 if ARGBD is not specified.

ICOUNT o the value with which the count field in SEMSEG will be initialized. This argument is assumed to be zero if not specified.

SARGD o this ODR is exactly as ARGD in its nature except that it specifically frames the arguments for PMME SIGNAL for the WAITQM macro.

SARGBD o this ODR is exactly as ARGBD in its nature except that it specifically frames the argument block for PMME SIGNAL for the WAITQM macro.

WARGD o this ODR is exactly as ARGD in its nature except that it specifically frames the arguments for PMME WAIT for the WAITQM macro.

WARGBD o this ODR is exactly as ARGBD in its nature except that it specifically frames the argument block for PMME WAIT for the WAITQM macro.

SCIDOF o this offset is exactly as CIDOFF in its nature except that it specifically is used as the offset to the CID for PMME SIGNAL for the WAITQM macro.

WCIDOF o this offset is exactly as CIDOFF in its nature except that it specifically is used as the offset to the CID for PMME WAIT for the WAITQM macro.

SGATOF o this offset is exactly as GATOFF in its nature except that it specifically is used as the offset to the gate for PMME SIGNAL for the WAITQM macro.

WGATOF o this offset is exactly as GATOFF in its nature except that it specifically is used as the offset to the gate for PMME WAIT for the WAITQM macro.

PQTYPE o this is the process queueing strategy. It has the same function and conventions as QTYPE.

MQTYPE o this is the message queueing strategy. It has the same function and conventions as QTYPE. The default is 'priority'.

MPRIO o an integer indicating the index register containing the priotity (0-63) associated with the V-operation used in the VMSEM macro. This argument is relevant only if the queueing strategy was defined as 'priority' at semaphore initialization (via IMSEM). The default value of priority is zero. Index register zero (X0) may not be used.

LPRIOR o this index register contains the pointer to the prior messaqe queue entry. Index register four (X4) is the default register if LPRIOR is not set by the user.

LNEXT o this index register contains the pointer to the next message queue entry. Index register two (X2) is the default register if LNEXT is not set by the user.

LCURR o this index register contains the pointer to the current message queue entry. Index register three (X3) is the default register if LCURR is not set by the user.

Functional Description of wait

This function provides the user with the ability to suspend the execution of a process. If a process has reached a point in its execution where it cannot continue until some subsequent event occurs, then this function should be used. A time value (in milliseconds) may be specified which will cause the suspended process to resume execution if it has not been signalled within that time. This function will open the loop monitor gate specified by the user so that the process will no longer be in the loop monitor when execution is resumed.

Usage Information of wait

The wait function has two externally visible interfaces, one for privileged programs executing in master mode (.CALL) and one for slave mode users (PMME). The .CALL interface will be described first followed by the PMME interface.

-- .CALL Interface --

Coding Format: .CALL .MSYNC,1

Input State:

Condition Identifier (X1)
Gate Pointer (ODR2)
Time Limit (QR)

Output State:

Return Code (X0)
Reason Code (X2)

Argument Declaration:

```
dcl 01 Return_Code,
       02 Fill Bit (2),
       02 Result Bit (16); /* 0 = Successful
                              3 = Timer Runout
                              4 = Software Interrupt */
```

```
dcl 01 Reason_Code,
       02 Fill Bit (6),
       02 Reason Fixed Bit (12); /* 0 = Null
                                   -1 = Timer Runout
                                   >0 = Via SIGNAL */


dcl 01 Condition_Identifier,
       02 CID Bit (18); /* This is the CID pointer
                           returned by a call to REQCID */


dcl 01 Gate_ptr,
       02 Address Bit (18), /* Gate offset */
       02 Gate_Seg DESC (0); /* This segment must contain
                                the monitor gate which will
                                be opened */


dcl 01 Time_Limit,
       02 Priority_Level Bit (6), /* This is the priority
                                     level with which the
                                     process will wait on
                                     the condition if the
                                     condition was requested
                                     with priority */
       02 Time Bit (30); /* Time (msecs) (0 = Default) */
```

Mission Description Runoff: 03/05/79

-- PMME Interface --

Coding Format: PMME WAIT

Input Variables:

Argument Block (.PS+0)
Condition Identifier (.PS+1)
Gate Segment (.PS+2)

Argument Declaration:

```
dcl 01 Argument_Block,
       02 Immediate_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Fill Bit (2),
          03 Return Bit (18), /* 0 = Successful
                                 1 = Illegal CID
                                 3 = Timer Runout
                                 4 = Software Interrupt */
       02 Original_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Fill Bit (2),
          03 Return Bit (18), /* 0 = Successful
                                 1 = Illegal CID
                                 3 = Timer Runout
                                 4 = Software Interrupt */
       02 Offset_List,
          03 CID_offset Bit (18), /* CID offset */
          03 Gate_offset Bit (18), /* Gate offset */
       02 Reason_Code,
          03 Fill Bit (24),
          03 Reason Fixed Bit (12), /* 0 = Null
                                      -1 = Timer Runout
                                      >0 = Via SIGNAL */
       02 Time_Limit,
          03 Priority_Level Bit (6), /* This is the priority
                                        level with which the
                                        process will wait on
                                        the condition if the
                                        condition was made
                                        with priority */
          03 Time Bit (30); /* Time (msecs) (0 = Default) */
```

```
    dcl 01 Condition_Identifier DESC (1);
 /* This frames the token provided by the call to PMME REQCID
    and is used to identify the condition being waited on.
    The token is an illegal descriptor (T=15) with bits 0-35 of
    word 1 being the condition identifier and bits 0-17 of
    word 0 being the key and is located at the offset specified
    by CID_offset. */


    dcl 01 Gate_Segment DESC (0);
 /* This descriptor frames the segment in which the gate
    resides at the offset specified by Gate_offset. */
```

Mission Description   Runoff: 03/05/79

Functional Description of signal

This function permits a process to notify (signal) another process(es) that some event has occurred. If there is some process queued waiting on a condition, then the process is re-entered into the dispatch queue. If there are no process waiting, the signal is ignored. If the broadcast option of this function is used, then all processes waiting on the specified condition are signalled rather than just the one on top of the waiting queue.

Usage Information of signal

The signal function has two externally visible interfaces, one for privileged programs executing in master mode (.CALL) and one for slave mode users (PMME). The .CALL interface will be described first followed by the PMME interface.

-- .CALL Interface --

Coding Format: .CALL .MSYNC,2

Input State:

Condition Identifier (X1)
Reason Code (X2)

Output State:

Return Code (X0)

Argument Declaration:

```
dcl 01 Return_Code,
       02 Fill Bit (2),
       02 Return Bit (16); /* 0 = Successful
                              2 = Queue Empty
                              3 = Program Not Enabled */


dcl 01 Reason_Code,
       02 Broadcast Bit (1), /* If this bit is on, all
                                processes currently waiting
                                on the  condition will be
```

CLASS: ps
FUNCTION: signal

Mission Description Runoff: 03/05/79

```
                                    signalled */
          02 Fill Bit (5),
          02 Reason Fixed Bit (12), /* This code will be passed
                                       to the signalled process
                                       via its reason code */


     dcl 01 Condition_Identifier,
          02 CID Bit (18); /* This is the CID pointer
                                 returned by a call to REQCID */
```

CLASS: ps
FUNCTION: signal

Mission Description   Runoff: 03/05/79

-- PMME Interface --

Coding Format: PMME SIGNAL

Input Variables:

Argument Block (.PS+0)
Condition Identifier (.PS+1)

Argument Declaration:

```
     dcl 01 Argument_Block,
            02 Immediate_Return_Code,
               03 Module_Number Bit (12),
               03 Entry_Point Bit (6),
               03 Fill Bit (2),
               03 Return Bit (16), /* 0 = Successful
                                      1 = Illegal CID
                                      2 = Queue Empty
                                      3 = Program Not Enabled */
            02 Original_Return_Code,
               03 Module_Number Bit (12),
               03 Entry_Point Bit (6),
               03 Fill Bit (2),
               03 Return Bit (16), /* 0 = Successful
                                      1 = Illegal CID
                                      2 = Queue Empty
                                      3 = Program Not Enabled */
            02 Reason_Code,
               03 CID_offset Bit (18), /* CID offset */
               03 Broadcast Bit (1), /* If this bit is on, all
                                       processes currently
                                       waiting on the condition
                                       will be signalled */
               03 Fill Bit (5),
               03 Reason Fixed Bit (12), /* 0 = Null
                                           -1 = Timer Runout
                                           >0 = Via SIGNAL */


     dcl 01 Condition_Identifier DESC (1);
  /* This frames the token provided by the call to PMME REQCID
     and is used to identify the condition being waited on.
     The token is an illegal descriptor (T=15) with bits 0-35 of
     word 1 being the condition identifier and bits 0-17 of
     word 0 being the key and is located at the offset specified
```

```
    by CID_offset. */
```

CLASS: ps
FUNCTION: reqcid

Mission Description Runoff: 03/05/79

Functional Description of reqcid

This function returns upon request one or more condition identifiers (CIDs) in the form of secure tokens. These tokens, which really are illegal T=15 descriptors, provide protected data that cannot be modified. These descriptors are the basic constructs used to coordinate WAITs and SIGNALs among processes.

Usage Information of reqcid

The reqcid function has two externally visible interfaces, one for privileged programs executing in master mode (.CALL) and one for slave mode users (PMME). The .CALL interface will be described first followed by the PMME interface.

-- .CALL Interface --

Coding Format: .CALL .MSYNC,3

Input State:

Request Data (X1)
Block Pointer (P2)

Output State:

Return Code (X0)
Request Data (X1)

Argument Declaration:

```
dcl 01 Return_Code,
       02 Fill Bit (2),
       02 Return Bit (16); /* 0 = Successful
                              1 = Request Not Fulfilled
                              2 = Count Was Zero */


dcl 01 Request_Data,
       02 Fill Bit (4),
       02 Queue_Type Bit (2), /*  0 = FIFO
                                  1 = LIFO
                                  2 = Priority
                                  3 = Undefined */
```

```
            02 Return_Count Bit (6), /* Output */
            02 Request_Count Bit (6); /* Input */


     dcl 01 Block_Pointer,
            02 Address Bit (18),
            02 Block_Segment DESC (0);  /* The CIDs returned are
                                           18 bit entities and
                                           each one is returned
                                           in the upper half */
```

CLASS: ps
FUNCTION: reqcid

Mission Description Runoff: 03/05/79

-- PMME Interface --

Coding Format: PMME REQCID

Input Variables:
Argument Block (.PS+0)
Block Segment (.PS+1)

Argument Declaration:

```
dcl 01 Argument_Block,
       02 Immediate_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Fill Bit (2),
          03 Return Bit (16), /* 0 = Successful
                                 1 = Request Not Fulfilled
                                 2 = Count Was Zero */
       02 Original_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Fill Bit (2),
          03 Return Bit (16), /* 0 = Successful
                                 1 = Request Not Fulfilled
                                 2 = Count Was Zero */
       02 Request_Data,
          03 Start_offset Bit (18),  /* Starting offset */
          03 Fill Bit (4),
          03 Queue_Type Bit (2), /*  0 = FIFO
                                     1 = LIFO
                                     2 = Priority
                                     3 = Undefined  */
          03 Return_Count Bit (6), /* Output */
          03 Request_Count Bit (6); /* Input */


dcl 01 Block_Segment DESC (1);  /* The CIDs returned are
                                   illegal descriptors
                                   (T = 15) and each one is
                                   returned in the two-word
                                   pair starting at the
                                   location specified by
                                    Start_offset */
```

Functional Description of retcid

This function deletes one or more condition identifiers (CIDs) as requested by the user. Once made invalid the returned CIDs can no longer be used to cooridnate WAITs and SIGNALs between processes.

Usage Information of retcid

The retcid function has two externally visible interfaces, one for privileged programs executing in master mode (.CALL) and one for slave mode users (PMME). The .CALL interface will be described first followed by the PMME interface.

-- .CALL Interface --

Coding Format: .CALL .MSYNC,4

Input State:

Request Data (X1)
Block Pointer (P2)

Output State:

Return Code (X0)
Request Data (X1)

Argument Declaration:

```
dcl 01 Return_Code,
       02 Fill Bit (2),
       02 Return Bit (16); /* 0 = Successful
                              1 = Someone Waiting
                              2 = Count Was Zero */

dcl 01 Request_Data,
       02 Fill Bit (4),
       02 Queue_Type Bit (2), /*  0 = FIFO
                                  1 = LIFO
                                  2 = Priority
                                  3 = Undefined */
       02 Return_Count Bit (6), /* Output */
       02 Request_Count Bit (6); /* Input */
```

```
dcl 01 Block_Pointer,
       02 Address Bit (18),
       02 Block_Segment DESC (0);  /* The CIDs returned are
                                      18 bit entities and
                                      each one is returned
                                      in the upper half */
```

CLASS: ps
FUNCTION: retcid

Mission Description Runoff: 03/05/79

-- PMME Interface --

Coding Format: PMME RETCID

Input Variables:
Argument Block (.PS+0)
Block Segment (.PS+1)

Argument Declaration:

```
dcl 01 Argument_Block,
       02 Immediate_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Fill Bit (2),
          03 Return Bit (16), /* 0 = Successful
                                 1 = Someone Waiting
                                 2 = Count Was Zero
                                 3 = Request Not Fulfilled */
       02 Original_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Fill Bit (2),
          03 Return Bit (16), /* 0 = Successful
                                 1 = Someone Waiting
                                 2 = Count Was Zero
                                 3 = Request Not Fulfilled */
       02 Request_Data,
          03 Start_offset Bit (18),  /* Starting offset */
          03 Fill Bit (4),
          03 Queue_Type Bit (2), /*  0 = FIFO
                                     1 = LIFO
                                     2 = Priority
                                     3 = Undefined  */
          03 Return_Count Bit (6), /* Output */
          03 Request_Count Bit (6); /* Input */


dcl 01 Block_Segment DESC (1);  /* The CIDs returned are
                                   illegal descriptors
                                   (T = 15) and each one is
                                   returned in the two-word
                                   pair starting at the
                                   location specified by
                                   Start_offset */
```

Functional Description of sfwint

This function allows a process to interrupt another process. Specifically, the execution of the target (to be interrupted) process is forced to a specific "interrupt handling routine" at the next dispatch to the target process.

Usage Information of sfwint

The sfwint function has two externally visible interfaces, one for privileged programs executing in master mode (.CALL) and one for slave mode users (PMME). The .CALL interface will be described first followed by the PMME interface.

-- .CALL Interface --

Coding Format: .CALL .MSYNC,5

Input State:

Target Process (AU)
User Entry Descriptor (ODR2)

Output State:

Return Code (X0)

Argument Declaration:

```
dcl 01 Return_Code,
       02 Fill Bit (2),
       02 Result Bit (16); /* 0 = Successful
                              1 = Interrupt Not Paid */


dcl 01 User_Entry_Desc DESC (11);
/* Descriptor to segment where control is to be passed
   after the software interrupt has been paid */
```

CLASS: ps
FUNCTION: sfwint

Mission Description   Runoff: 03/05/79

-- PMME Interface --

Coding Format: PMME SFWINT

Input Variables:

Argument Block (.PS+0)
User Entry Descriptor (.PS+1)

Argument Declaration:

```
dcl 01 Argument_Block,
       02 Immediate_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Return Bit (18), /* 0 = Successful
                                 1 = Interrupt Not paid */
       02 Original_Return_Code,
          03 Module_Number Bit (12),
          03 Entry_Point Bit (6),
          03 Return Bit (18), /* 0 = Successful
                                 1 = Interrupt Not paid */
       02 Request_Data,
          03 Target_KPX Bit (18),
          03 Fill Bit (18);

dcl 01 User_Entry_Desc DESC (11);
/* Descriptor to segment where control is to be passed
   after the software interrupt has been paid */
```