

DATE: September 22, 1983
TO: R and D Personnel
FROM: Jerry Kazin
SUBJECT: Software Interrupt Control Module Functional Spec.
REFERENCE: PRIMOS Subroutines Guide (BREAK\$ Function)
Software Interrupt Mechanism
PE-TI-879
Software Interrupt Control Module Proposal
PE-T-1004
Software Interrupt Control Module Design Spec.
PE-TI-1006
KEYWORDS: Software Interrupts, QUIT

ABSTRACT

Prior to Rev. 18, PRIMOS contained only one software interrupt known as the QUIT\$ condition. This condition may be enabled/disabled by calling the BREAK\$ module. At both Rev. 18 and Rev. 19 new software interrupts were added to the system. This paper discusses four new modules, SW\$INT, SW\$MKRCS, SW\$ROOFF, SW\$RAOF, and SW\$ON which allow users to enable/disable these new interrupts. It also discusses when and how to use the module.

This document is classified PRIME RESTRICTED.
It must not be distributed to non-Prime Personnel.
When there is no longer a need for this document,
it should be returned to the Bldg. 10 Information
Center by special delivery inter-office mail - or
destroyed.

©Prime Computer, Inc., 1983
All Rights Reserved

*** PRIME RESTRICTED ***

Table of Contents

1 INTRODUCTION.....3

2 WHY DO WE NEED SW\$INT.....3

3 MODULE DESCRIPTIONS.....4

 3.1 SW\$INT.....5

 3.2 SW\$MKRCS.....8

 3.3 SW\$ROOFF.....9

 3.4 SW\$RAOF.....10

 3.5 SW\$ON.....11

4 HOW TO MAKE A CRITICAL SECTION.....12

 4.1 In Ring 3.....12

 4.1.1 PL/P.....12

 4.1.2 FORTRAN.....13

 4.1.3 PMA.....13

 4.2 In Ring 0.....14

 4.2.1 Programming Example.....14

5 HOW TO MAKE A REVERSE CRITICAL SECTION.....15

 5.1 PL/P.....16

 5.2 FORTRAN.....16

 5.3 PMA.....17

6 ENABLING/DISABLING SELECTED SOFTWARE INTERRUPTS.....17

 6.1 Programming Example.....17

7 MORE INFORMATION ON THE SOFTWARE INTERRUPT MECHANISM.....19

1 INTRODUCTION

This document provides a description of the functionality provided by the Software Interrupt Control Module, SW\$INT and its streamlined friends SW\$MKRCS, SW\$ROOFF, SW\$RAOF, and SW\$ON. Up till now, a user was able to turn off only one kind of software interrupt, QUIT. This was done via the BREAK\$ module. For more information on this module consult PRIMOS Subroutines Guide (BREAK\$ Function).

At Rev. 19.0 six software interrupt types have been defined for PRIMOS. They are the following:

- 1) CPU watchdog timer (CPU_TIMER\$ condition),
- 2) Real Time watchdog timer (ALARM\$ condition),
- 3) Phantom Logout Notification (PH_LOGO\$),
- 4) Cross Process Signalling (CPS\$),
- 5) Logout (LOGOUT\$), and
- 6) Terminal QUIT (QUIT\$).

CPS\$ is an internal condition and will never be released to the general public.

At Rev. 19.3 a seventh interrupt type has been added.

- 1) IPC Message Waiting (IPC_MSG_WAITING\$)

SW\$INT and its associated mechanism allow a user to independently and selectively enable/disable any or all of these interrupts. SW\$MKRCS, SW\$ROOFF, SW\$RAOF, and SW\$ON allow a user to enable/disable all of these interrupts at one time.

2 WHY DO WE NEED SW\$INT

The software interrupt mechanism normally enables the receipt of interrupts in ring 3 and normally disables their receipt in ring 0. In light of this fact, there are three main reasons why SW\$INT has been created.

- 1) It often happens that programmers need to create sections of code that are non-interruptable. These sections of code are known as critical sections. When a software interrupt is signalled, an asynchronous event occurs. If these asynchronous events were allowed to occur, a programmer would not be able to create a critical section. Since SW\$INT and SW\$RAOF allow programmer's to disable all software interrupts, the use of these modules gives the programmer the needed ability to create a critical section. As ring 0 is normally software interrupt inhibited, this point only relates to the ring 3 programmer.

2) Sometimes it is necessary only to turn off a specific software interrupt, not all interrupts. For example, a program may need to impede terminal quits, but yet allow its user to see phantom logout notification. (It may be spawning phantoms on behalf of its user.) If only all interrupts could be turned off, it would not be possible to achieve this state. SW\$INT allows a programmer to selectively enable/disable the various interrupt types.

3) There are pieces of code in ring 0 that need to receive software interrupts. Since ring 0 is normally disabled for software interrupts, it must be possible to enable software interrupts. SW\$INT and SW\$MKRCS allow software interrupts to be taken while in ring 0. The section of ring 0 code that is allowed to take these interrupts is called a reverse critical section.

3 MODULE DESCRIPTIONS

The following module descriptions contain references to the variables selection and value. These variables have a three word data structure as follows:

```
dcl 1 bit_struct,
    2 len fixed bin,      /* number of bits available */
    2 first16 bit(16),   /* first set of 16 bits */
    2 second16 bit(16); /* second set of 16 bits */
```

Presently seven interrupts types have been defined. Their positions within selection and value are:

- terminal quit - bit 1
- cpu timer - bit 2
- real time timer - bit 3
- logout - bit 4
- phantom logout notification - bit 5
- cross process signalling - bit 6
- ipc message waiting - bit 7

3.1 SW\$INT

Software Interrupt Control Module

| SW\$INT |

| SW\$INT |

Subroutine

May 25, 1982

Name: SW\$INT

Purpose:

SW\$INT is used to control the enable/disable status of the software interrupts. It does this by setting/resetting the enable bit(s) of the software interrupt ring control words located in PUDCOM.

SW\$INT is able to set on, set off, or simply read the status of the interrupt type chosen by its caller. The type is chosen by either setting a bit(s) on in the input argument, SELECTION, or using one of the "all" keys. For read, the current setting is returned in the argument, VALUE.

A user may enable/disable and/or read any interrupt(s) in an outer ring by including the optional outer ring argument.

!During enabling if an interrupt is found pending which is enabled, it will be handled immediately.

Software interrupts are normally off in ring 0 and normally on in the outer rings.

The module is much more powerful in ring 0. It can additionally return the already deferred interrupt setting. Additionally, to ease use in ring0 this module has two names, SW\$INT and SW\$IN. In PL/P one can be used as the function call and one as the procedure call.

When SW\$INT is called to inhibit or enable interrupts with any of the "all" keys (see below for key definitions), the terminal quit counters will not be incremented/decremented if they already indicate the desired setting. In other words, if SW\$INT is called to inhibit all interrupts and the quit counter is already positive meaning terminal quits are already inhibited, the counter will not be incremented. This is analogous for the enable case.

Usage:

```
From Any Ring: dcl sw$int entry(fixed bin, 1, 2 fixed bin,
                             2 bit(16), 2 bit(16),
                             1, 2 fixed bin, 2 bit(16),
                             2 bit(16), fixed bin [,
                             fixed bin]);
                call sw$int(key, selection, value, ercode [,
                             outer_ring]);
```

Additionally

```
From Ring 0:  dcl sw$int entry(fixed bin, 1, 2 fixed bin,
                          2 bit(16), 2 bit(16),
                          1, 2 fixed bin, 2 bit(16),
                          2 bit(16), fixed bin [,
                          fixed bin])
                          returns(fixed bin);
```

```
;          swi_already_deferred = sw$int(key, selection,
                                          value, ercode [,
                                          outer_ring]);
```

key - input key

valid keys and their values are:

```
k$read = 1   Read present status
k$on    = 2   Turn on interrupt(s)
k$off   = 3   Turn off interrupt(s)
k$rdon  = 4   Read present status and
              turn on interrupt(s)
k$rdof  = 5   Read present status and
              turn off interrupt(s)
k$rdal  = 6   Read present status of all
              interrupts
k$alon  = 7   Turn on all interrupts
k$alof  = 8   Turn off all interrupts
k$raon  = 9   Read present status and
              turn on all interrupts
k$raof  = 10  Read present status and
              turn off all interrupts
```

selection - specific interrupt type chosen

value - when a read is issued and the interrupt type chosen in selection or by use of an "all" key is found enabled the bit as defined in selection above is turned on whenever a read is to be performed the len field of value must be initialized to a number ≥ 7 (number of interrupt types) so SW\$INT knows that the user has presented a buffer large enough for it to write into

ercode - standard PRIMOS error code
only ones used are:

```
bad key - unknown key
bad parameter - bad length field in selection
               bad optional outer ring
buffer too small - bad length field in value
```

outer_ring - optional input argument that tells SW\$INT to act upon the ring specified in outer_ring

| swi_already_deferred - if = 1 then a software interrupt has already
| been deferred in ring 0

3.2 SW\$MKRCS

Make A Reverse Critical Section

| SW\$MKRCS |

| SW\$MKRCS |

TUE, 13 SEP 1983

| Subroutine

| Name: SW\$MKRCS| Purpose:

| Allows software interrupts to be seen in ring 0. Callable only in ring 0.

| During enabling if an interrupt is found pending which is enabled, it will be handled immediately.

| Usage:| dcl sw\$mkrcs entry(1, 2 fixed bin, 2 bit(16), 2 bit(16))
| returns(fixed bin);

| swi_already_deferred = sw\$mkrcs(value);

| value - returned indication of inhibited interrupts
| at time of call| swi_already_deferred - if = 1 then a software interrupt has already
| been deferred in ring 0

3.3 SW\$ROOFF

Turns Software Interrupts Off

| SW\$ROOFF |

| SW\$ROOFF |

| Subroutine

TUE, 13 SEP 1983

| Name: SW\$ROOFF

| Purpose:

| Turns selected software interrupts off. Only callable in ring 0.

| Usage:

| dcl sw\$r0off entry(1, 2 fixed bin, 2 bit(16), 2 bit(16));

| call sw\$r0off(selection);

| selection - specific interrupt type chosen

3.4 SW\$RAOF

Makes A Critical Section

| SW\$RAOF |

| SW\$RAOF |

| Subroutine

TUE, 13 SEP 1983

| Name: SW\$RAOF| Purpose:| Turns off all software interrupts. Should only be called from ring 3
| as it only turns off interrupts for ring 3.| Usage:| dcl sw\$raof entry(1, 2 fixed bin, 2 bit(16), 2 bit(16));
| call sw\$raof(value);| value - returned indication of enabled interrupts
| at time of call

3.4 SW\$RAOF

Makes A Critical Section

| SW\$RAOF |

| SW\$RAOF |

| Subroutine

TUE, 13 SEP 1983

| Name: SW\$RAOF

| Purpose:

| Turns off all software interrupts. Should only be called from ring 3
| as it only turns off interrupts for ring 3.

| Usage:

| dcl sw\$raof entry(1, 2 fixed bin, 2 bit(16), 2 bit(16));

| call sw\$raof(value);

| value - returned indication of enabled interrupts
| at time of call

3.5 SW\$ON

Turns Software Interrupts On

| SW\$ON |

| SW\$ON |

| Subroutine

TUE, 13 SEP 1983

| Name: SW\$ON| Purpose:| Enables the selected software interrupts. Should only be called from
| ring 3 as it only turns on interrupts for ring 3.| During enabling if an interrupt is found pending which is enabled, it
| will be handled immediately.| Usage:| dcl sw\$on entry(1, 2 fixed bin, 2 bit(16), 2 bit(16));
| call sw\$on(selection);

| selection - specific interrupt type chosen

4 HOW TO MAKE A CRITICAL SECTION

Critical sections allow a programmer to write non-interruptable code. Since interrupts are normally enabled in ring 3 and disabled in ring 0, only ring 3 users normally have to worry about creating critical sections. Unfortunately, on rare occasions it may be necessary to insure that ring 0 is in a critical section. This may occur because a prior module in ring 0 has enabled any or all of the software interrupts.

This section is therefore split into two parts, making a critical section in ring3 and making a critical section in ring 0.

4.1 In Ring 3

Generically a critical section in ring 3 can be made by:

- 1) Finding out what interrupts are presently enabled.
- 2) Turning off the found interrupts.

The critical section is ended by:

- 1) Turning back on the found interrupts.

Note that this method of creating a critical section leaves the state of a program's execution environment the same after the critical section as before the critical section.

These three operations can be achieved in two atomic steps:

- 1) Finding out which interrupts are presently on while turning them off.
- 2) Turning back on the found interrupts.

The following sections describe how to make a critical section in PL/P, FORTRAN, and PMA.

4.1.1 PL/P

```
del sw$raof entry(1, 2 fixed bin, 2 bit(16), 2 bit(16));
                1, 2 fixed bin, 2 bit(16), 2 bit(16), fixed bin),
1 value,
  2 len fixed bin,
  2 first16 bit(16),
  2 second16 bit(16);
```

```
/* ***** BEGIN CRITICAL SECTION ***** */
```

```
|call sw$raof(value);
```

```
|           {User Code}
```

```
|call sw$on(value);
```

```
/* ***** END CRITICAL SECTION ***** */
```

4.1.2 FORTRAN

```
| INTEGER*2 VALUE(3)
```

```
|C /* ***** BEGIN CRITICAL SECTION ***** */
```

```
| CALL SW$RAOF(VALUE)
```

```
|           {User Code}
```

```
| CALL SW$ON(VALUE)
```

```
|C /* ***** END CRITICAL SECTION ***** */
```

4.1.3 PMA

```
| DYNAM VALUE(3)
```

```
/* /* ***** BEGIN CRITICAL SECTION ***** */
```

```
| CALL SW$RAOF  
| AP VALUE,SL
```

```
| {User Code}
```

```
| CALL SW$ON  
| AP VALUE,SL
```

```
/* /* ***** END CRITICAL SECTION ***** */
```

4.2 In Ring 0

Generically a critical section in ring 0 can be made by:

- 1) Get and save the present value for all counted software interrupts
- 2) Set the value for all counted interrupts to 0
- 3) Call SW\$INT with the read all off key to get present enabled state which must be saved

To end the critical section, do the following:

- 1) Call SW\$INT with the on key and the saved present enable state
- 2) Restore the saved value for all counted software interrupts

Note that this method of creating a critical section leaves the state of a program's execution environment the same after the critical section as before the critical section.

The following section contains a programming example in PL/P of how to make a critical section in ring 0.

4.2.1 Programming Example

```
dcl break$ entry (bit(16), fixed bin),
  sw$int entry(fixed bin, 1, 2 fixed bin, 2 bit(16),
              2 bit(16), 1, 2 fixed bin, 2 bit(16),
              2 bit(16), fixed bin),
  brk_count fixed bin,
  1 selection,
  2 len fixed bin,
  2 first16 bit(16),
  2 second16,
  3 nu bit(16);
dcl 1 value like selection,
  ercode fixed bin;

      brk_set by '0004'b4,
      k$raof by 10,
      k$off by 3,

/* ***** BEGIN CRITICAL SECTION ***** */

call break$(brk_get, brk_count);
call break$(brk_set, 0);
value.len = 32;
call sw$int(k$raof, selection, value, ercode);
```

```

|           {User Code}

|call sw$int(k$off, selection, value, ercode);
|call break$(brk_set, brk_count);

|/* ***** END CRITICAL SECTION ***** */

```

5 HOW TO MAKE A REVERSE CRITICAL SECTION

Software interrupts are normally disabled in ring 0 to make kernal operations normally atomic. This allows ring 0 programmers to forget about the need for critical sections, a very useful convenience. However, sometimes ring 0 code, such as C1IN\$, needs to enable software interrupts. As was previously mentioned, these sections of code are termed reverse critical sections.

Generically a reverse critical section can be made by:

- 1) Finding out what interrupts are presently disabled.
- 2) Turning on the found interrupts.

The reverse critical section is ended by:

- 1) Turning back off the found interrupts.

Note that this method of creating a critical section leaves the state of a program's execution environment the same after the reverse critical section as before the reverse critical section.

These three operations can be achieved in two atomic steps:

- 1) Finding out which interrupts are presently off while turning them on.
- 2) Turning back off the found interrupts.

One other item needs to be considered in ring 0 when creating a reverse critical section. Since interrupts are normally off in ring 0, there exists the possibility that a software interrupt occurred while we were in ring 0 with interrupts disabled. When the system detects such a situation and finds the interrupts to be enabled in the outer rings, it defers the interrupt until the first exit from ring 0. Only one interrupt type may ever be found deferred. Therefore, when creating a reverse critical section, the programmer must read the SWI_ALREADY_DEFERRED status returned. If there already is a deferred interrupt, the program should return as soon as possible. The return from ring 0 will then catch the deferred interrupt. Since the programmer was trying to set up to see interrupts, this is the

functionality that the programmer desires.

Two other rules must also be followed by the ring 0 user. When a deferred interrupt is detected, the initial call into ring 0 is set up to be retried. Therefore, anytime a programmer is going to create a reverse critical section, no static changes can be made to the user environment at any point before the reverse critical section is made.

Basically, the above two paragraphs show that the only time a programmer needs to pay attention to the SWI_ALREADY_DEFERRED status is when the program is in ring 0 and software interrupts are to be enabled.

For more information consult Software Interrupt Mechanism PE-TI-879.

The following sections describe how to make a reverse critical section in PL/P, FORTRAN, and PMA.

5.1 PL/P

```
|dcl sw$mkrcs entry(1, 2 fixed bin, 2 bit(16), 2 bit(16))
|      returns(fixed bin),
|      1 value,
|      2 len fixed bin,
|      2 first16 bit(16),
|      2 second16 bit(16);
```

```
|/* ***** BEGIN REVERSE CRITICAL SECTION ***** */
```

```
|if sw$mkrcs(value) = 1
|  then return;
```

```
|      {User Code}
```

```
|call sw$r0off(value);
```

```
|/* ***** END REVERSE CRITICAL SECTION ***** */
```

5.2 FORTRAN

```
|      INTEGER*2 VALUE(3)
```

```
|C      /* ***** BEGIN REVERSE CRITICAL SECTION ***** */
```

```
|      IF (SW$MKRCS(VALUE) .EQ. 1) RETURN
```

```
|      {User Code}
```

```

|      CALL SW$ROOFF(VALUE)
|C     /* ***** END REVERSE CRITICAL SECTION ***** */

```

5.3 PMA

```

|      DYNM VALUE(3)
|*     /* ***** BEGIN REVERSE CRITICAL SECTION ***** */
|
|      CALL SW$MKRCS
|      AP VALUE,SL
|      S1A          INTERRUPT
|      SNZ          PENDING?
|      PRTN        YES - RETURN
|
|      {User Code}
|
|      CALL SW$ROOFF
|      AP VALUE,SL
|*     /* ***** END REVERSE CRITICAL SECTION ***** */

```

6 ENABLING/DISABLING SELECTED SOFTWARE INTERRUPTS

|SW\$INT can be used to selectively enable or disable specific interrupts. As ring 3 is most likely to use this feature, an program which runs in this ring will be used as an example.

|Only PL/P will be used as the programming example.

6.1 Programming Example

|A program running in ring 3, such as EMACS, decides that it does not want the phantom logout notification message to be printed immediately. The following code show how EMACS would disable this interrupt.

```

|dcl sw$int entry(fixed bin, 1, 2 fixed bin, 2 bit(16), 2 bit(16),
|                1, 2 fixed bin, 2 bit(16), 2 bit(16), fixed bin),
|  key fixed bin,
|  1 selection,          /* selected interrupt */
|  2 len fixed bin,
|  2 first16 bit(16),
|  2 second16 bit(16);
|dcl 1 dummy like selection, /* dummy value arg */
|  ercode fixed bin;        /* standard system error code */
|%replace ph_logo_bit by '0800',

```

```
k$on by 2,  
k$off by 3,  
max_num_swis by 7; /* max number software ints. */
```

```
.  
.  
.  
EMACS Code During Which It  
Is Alright To See Phantom  
Logout Notifications  
.  
.  
.
```

```
/* turn off phantom logout notification */
```

```
selection.len = max_num_swis;  
selection.first16 = ph_logo_bit;  
call sw$int(k$off, selection, dummy, ercode);
```

```
.  
.  
.  
EMACS Code During Which It  
Is Not Alright To See Phantom  
Logout Notifications  
.  
.  
.  
.  
.
```

```
/* turn on phantom logout notification */
```

```
call sw$int(k$on, selection, dummy, ercode);
```

```
EMACS Code During Which It  
Is Alright To See Phantom  
Logout Notifications  
.  
.  
.
```

```
Remember that any combination of interrupts may be selected.
```

7 MORE INFORMATION ON THE SOFTWARE INTERRUPT MECHANISM

|For information on the details of the software interrupt mechanism
|consult Software Interrupt Mechanism PE-TI-879.

|For information relating to why this new software interrupt control
|module was built consult Software Interrupt Control Module Proposal
|PE-T-1004.

|For information relating to the design details of the software
|interrupt control mechanism consult Software Interrupt Control Module
|Design Spec. PE-TI-1006.