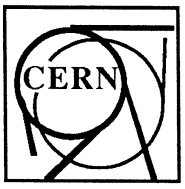
ALEPH 88-147
DATACQ 88-22
A. Miotto
14.11.1988



ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

Handling OS-9 Signals

Alessandro Miotto

A small library to improve signal handling under the OS–9/68K operating system.

Author: A. Miotto.
Network Address: AL0VOL::MIOTTO at CERN
Version of Document: 1.0
Revision date: 26 September 1988
Status: Draft

Changes from last version:

This is a new document.

1. INTRODUCTION

OS-9 provides a way to execute asynchronous user routines via the signal mechanism. After the user has installed the intercept routine, any signal sent to the process (other than `S$Kill` and `S$WakeUp`) causes its execution. Unfortunately if the installation has not been done, any signal (other than `S$WakeUp`) will cause the process to abort. Moreover, because signal are handled directly in the user routine, there is no safe way to provide asynchronous calls from inside a library.

The routines contained in this library allow standard handling as well as connecting selected signals. At any time previously declared connections can be cancelled or signals can be masked. Two further routines are implemented to allow bookkeeping of user signals.

The behaviour of the dispatcher is the following: when a signal is received the table of declared signals (up to 16) is scanned to see whether a specific routine has to been called. If the signal number is not declared, then the default intercept routine (if declared) is called. Finally if no default intercept has been declared, all signal but `S$Intrpt` and `S$Abort` (that cause process to abort) are ignored. If the signal is masked it will be queued, and the corresponding user routine will be executed whenever the signal is unmasked again. Be aware that if even if the signal is masked, a `S$WakeUp` will be sent to the process, causing it to resume execution.

The dispatcher is installed after the first call to a declare routine. Before that, the normal OS-9 behaviour should be expected (i.e. signals are intercepted by the `shell`).

Selected connections can be used from inside library routines to handle specific system signals and execute asynchronous user routines in a transparent way, or to use safely signals different from `S$Wake` (to prevent deadlocks caused by the fact that this signal is not queued) without aborting the user program.

The library filename is `signals.l`.

The include filename (containing error codes) is `sigdef.h`.

2.1 SIGNAL INTERCEPTING

SIG_DECLARE_SIGNAL Declare Selected Signal Intercept Routine.

C Syntax:
```
int sig_declare_signal (signal_num, my_proc);
int signal_num;
void (*my_proc)();
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_DECLARE_SIGNAL (SIGNAL_NUM, MY_PROC)
INTEGER*4 SIGNAL_NUM, MY_PROC
```

Description: Declares the connection to the selected intercept routine. Whenever the signal `signal_num` is sent to the process, the user routine `my_proc` is called.

Return value:

`SIG_NORMAL`	
`SIG_ERROR`	(error code in global variable `errno`)

Possible error codes:

`SIG_TABLE_FULL`	maximum number of connections (16) reached
`SIG_ALREADY_CONN`	signal already connected.

SIG_CANCEL_SIGNAL Cancel Selected Signal Intercept Routine.

C Syntax:
```
int sig_cancel_signal (signal_num);
int signal_num;
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_CANCEL_SIGNAL (SIGNAL_NUM)
INTEGERT*4 SIGNAL_NUM
```

Description: Cancels the connection to the selected intercept routine. Whenever the signal `signal_num` is received it will be handled by the default procedure, if connected. If the default procedure is not connected and the signal is `S$Abort` or `S$Intrpt`, the user process will be aborted, otherwise the signal is ignored. No warning is returned if the signal is masked and its queue is not empty.

Return value:

`SIG_NORMAL`	
`SIG_ERROR`	(error code in global variable `errno`)

Possible error codes:

`SIG_NOT_CONN`	signal not connected.

SIG_DECLARE_DEFAULT Declare Default Intercept Routine.

C Syntax:
```
int sig_declare_default (my_proc);
void (*my_proc)();
/* or */
int intercept (my_proc);
void (*my_proc)();
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_DECLARE_DEFAULT (MY_PROC)
INTEGER*4 MY_PROC
```

Description: Declares the connection to the default intercept routine `my_proc`. Whenever a signal is received for which a selected connection has not been made, the user procedure `my_proc` will be called.

Return value:

`SIG_NORMAL`	
`SIG_ERROR`	(error code in global variable `errno`)

Possible error codes:

`SIG_ALREADY_CONN`	default already connected

SIG_CANCEL_DEFAULT Cancel Default Intercept Routine.

C Syntax:
```
int sig_cancel_default ();
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_CANCEL_DEFAULT ()
```

Description: Cancels the connection to a default intercept routine. Whenever the signal `signal_num` is received it will be handled by the selected signal intercept routine, if connected. If the selected signal is not connected and the signal is `S$Abort` or `S$Intrpt`, the user process will be aborted, otherwise the signal is ignored. All masked signals that are connected to the default intercept routines are deleted, even if their queues are not empty.

Return value:

`SIG_NORMAL`	
`SIG_ERROR`	(error code in global variable `errno`)

Possible error codes:

`SIG_NOT_CONN`	no default connected.

2.2 SIGNAL MASKING

SIG_MASK_SIGNAL Mask Signal.

C Syntax:
```
int sig_mask_signal (signal_num);
int signal_num;
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_MASK_SIGNAL (SIGNAL_NUM)
INTEGERT*4 SIGNAL_NUM
```

Description: The signal number `signal_num` does not cause anymore an user procedure to be executed. The signal is queued instead.

Return value:

`SIG_NORMAL`	
`SIG_ERROR`	(error code in global variable `errno`)

Possible error codes:

`SIG_ALREADY_MASKED`	signal has already been masked.
`SIG_NOT_CONN`	signal not connected.

SIG_UNMASK_SIGNAL Unmask Signal.

C Syntax:
```
int sig_unmask_signal (signal_num);
int signal_num;
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_UNMASK_SIGNAL (SIGNAL_NUM)
INTEGERT*4 SIGNAL_NUM
```

Description: The signal number `signal_num` is unmasked. If the queue is not empty, the user routine connected to the signal is called as many times as the number of signals received while it was masked.

Return value:

`SIG_NORMAL`	
`SIG_ERROR`	(error code in global variable `errno`)

Possible error codes:

`SIG_NOT_MASKED`	signal not masked.
`SIG_NOT_CONN`	signal not connected.

SIG_SHOW_PENDING Show Number of Pending Signals.

C Syntax:
```
int sig_show_pending (signal_num);
int signal_num;
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_SHOW_PENDING (SIGNAL_NUM)
INTEGERT*4 SIGNAL_NUM
```

Description: If the signal number `signal_num` was masked, this function returns the number of signals queued while it was masked.

Return value: number of pending signals in the queue

Possible error codes:

`SIG_ERROR`	(error code in global variable `errno`)
`SIG_OVERFLOW`	too many signals have been queued (maximum is 32767).
`SIG_NOT_MASKED`	signal not masked.
`SIG_NOT_CONN`	signal not connected.

2.3 SIGNAL BOOKKEEPING

SIG_BOOK_SIGNAL Book User Defined Signal.

C Syntax: `int sig_book_signal ();`

FORTRAN Syntax: `INTEGER FUNCTION SIG_BOOK_SIGNAL ()`

Description: Each time the routine is called, a new signal number is returned to the user.

Return value: number of the signal that has been booked (256 + 511)

`SIG_ERROR` (error code in global variable `errno`)

Possible error codes: `SIG_OVERBOOK` too many signals have been booked (maximum is 256).

SIG_UNBOOK_SIGNAL Cancel Booking of User Defined Signal.

C Syntax:
```
int sig_unbook_signal (signal_num);
int signal_num;
```

FORTRAN Syntax:
```
INTEGER FUNCTION SIG_UNBOOK_SIGNAL (SIGNAL_NUM)
INTEGERT*4 SIGNAL_NUM
```

Description: Cancels booking of user defined signal `signal_num`. The same signal can be returned by a subsequent call to `sig_book_signal ()`.

Return value: `SIG_NORMAL`

`SIG_ERROR` (error code in global variable `errno`)

Possible error codes:
- `SIG_NOT_BOOKED` signal not connected
- `SIG_SYSTEM` signal number in the system range (0 + 256)
- `SIG_TOO_BIG` signal number greater than 511.

EXAMPLES

In the first program a default handler is declared, and later the handling routine is changed. This is not a typical application but shows what the library can do.

In the second program only keyboard interrupts are intercepted to exit the program in the correct way. Then a signal is booked and used to execute an user routine whenever an ethernet packet is received.

Example 1.

```
#include <sigdef.h>
...
first_intercept ()
{ ... }        /* here you are if you get a signal */

second_intercept ()
{ ... }

main ()
{
    ...
    sig_declare_default (first_intercept);   /* standard OS-9 intercept */
    ...
    sig_cancel_default ();                   /* change handling routine */
    /* any signal coming now (except keybord interrupts) is lost */
    sig_declare_default (second_intercept);
    ...
}
```

Example 2.

```
#include <sigdef.h>
...
quit ()
{
    ...        /* flush buffers and cleanup */
    exit (1);
}

got_a_packet ()
{ ... }        /* this routine is called on receiving ethernet packets */

main ()
{
    ...
    /* quit on CTRL-C and CTRL-E - ignore all other signals */
    sig_declare_signal (SIGQUIT, quit);
    sig_declare_signal (SIGINT, quit);
    ...
    /* book signal and use it for reading ethernet packets */
    eth_signal = sig_book_signal ();
    sig_declare_signal (eth_signal, got_a_packet);
    eth_read (protocol_number, eth_signal, buffer, length);
    ...
}
```