Signal Handling

Signal Handling

- A signal is used in UNIX systems to notify a process that a particular event has occurred.
- A signal is a notification to a process that an event has occurred. Signals are sometimes called “software interrupts”.
- Features of Signal

  - The process does not know ahead of time exactly when a signal will occur.

  - Signal can be sent by one process to another process (or to itself) or by the kernel to a process.

Predefined Signals

- SIGALRM: Alarm timer time-out. Generated by ***alarm()*** API.
- SIGILL: Execution of an illegal machine instruction.
- SIGINT: Process interruption. Can be generated by *<Delete> or <ctrl_C>* keys.
- SIGSEGV: Segmentation fault. generated by de-referencing a NULL pointer.
- SIGTERM: process termination. Can be generated by
- "kill <process_id>" command.
- SIGCHLD: Sent to a parent process when its child process has terminated.

Signal Handling

- **Synchronous** signals are delivered to the same process that performed the operation that caused the signal.
- Synchronous signals include illegal memory access and division by 0.
- If a running program performs either of these actions, a signal is generated and sent to same process.

Signal Handling

- **Asynchronous** signal is sent to another process.
- When a signal is generated by an event external to a running process, that process receives the signal asynchronously.
- Examples of such signals include terminating a process with specific keystrokes (such as <control><C>) and having a timer expire.

Sources for Generating Signals

- Hardware

  - A process attempts to access addresses outside its own address space.

- Kernel

  - Notifying the process that an I/O device for which it has been waiting is available.

- Other Processes

  - A child process notifying its parent process that it has terminated.

- User

  - Pressing keyboard sequences that generate a quit, interrupt or stop signal.

Signal Handling

A signal may be handled by one of three possible handlers:

1. Ignore the signal
   - A process can do ignoring with all signal.
2. A default signal handler
   - Every signal has a **default signal handler** that the kernel runs when handling that signal.
3. A user-defined signal handler
   - This default action can be overridden by a **user-define** signal handler that is called to handle the signal.

Kernel support

- Process table has an array of signal flags
  - Flag=1 ignore
  - Flag=0 default action
  - Flag= any other number user defined

Signal Handling

- Specify a signal handler function to deal with a signal type.

```
#include <signal.h>
int  (*signal(int sig_no, void(*handler)(int)))(int);
```

- The sig_no argument is the name of the signal such as SIGALRM, SIGCHLD, etc.
- sig_no is an integer.
- The second argument, **handler**, is a pointer to a function that takes a single integer argument and returns nothing.

Example

```
int main()
 {
    signal( SIGINT, user_fun);
    /* do usual things until SIGINT */
 return 0;
 }

void user_fun( int sig_no )
 {
      /* deal with SIGINT signal */

 return;       /* return to program */
 }
```

Example

```
#include<signal.h>
void catch_sig(int sig_num)
{
print("signal caught %d", sig_num);
}

int main()
    {
     signal(SIGTERN, catch_sig);
     signal( SIGINT, SIG_IGN);
     signal(SIGSEGV, SIG_DFL);
     pause();
     return 0;
    }
```

Signal Handling- Alternate

- Unix V3 and V4 did not support signal().
- The alternate of signal().

```
#include <signal.h>
int  (*sigset(int sig_no, void(*handler)(int)))(int);
```

- The sig_no argument is the name of the signal.
- sig_no is an integer.
- The second argument, **handler**, is a pointer to a function.

Signal Handling

- SIG_DFL =0 Request for default signal handling.
- SIG_IGN=1 Request that signal be ignored.
- SIG_ERR Return value from signal() in case of error.

Signal Mask

- A process can query or set its signal mask via the ***sigprocmask***.

  signal.h

- *int* ***sigprocmask*** *(int cmd, const sigset_t *new_mask,*

  *sigset_t *old_mask);*

- Return: Success: 0 and Failure: -1
- *cmd*: specifies how the new_mask value is to be used:

  - SIG_SETMASK: Overrides the calling process signal mask with the value specified in the new_mask argument.

  - SIG_BLOCK: Adds the signals specified in the new_mask argument to the calling process signal mask.

  - SIG_UNBLOCK: Removes the signals specified in the new_mask argument from the calling process signal mask.

Signal Mask

- *new_mask*: defines a set of signals to be set or reset in a calling process signal mask.

  new_mask = NULL, current process signal mask unaltered.
- *old_mask*: Address of a *sigset_t* variable that will be assigned the calling processing's original signal mask.

  old_mask = NULL, no previous signal mask will be return.

- sigset_t: Integer or structure type of an object used to represent sets of signals.

Signal Mask

- *int* ***sigemptyset*** *(sigset_t* sigmask);*

  Clears all signal flags in the *sigmask* argument.
- *int* ***sigaddset*** *(sigset_t* sigmask, const int signal_num);*

  Sets the flag corresponding to the *signal_num* signal in the *sigmask* argument.
- *int* ***sigdelset*** *(sigset_t* sigmask, const int signal_num);*

  Clears the flag corresponding to the *signal_num* signal in the *sigmask* argument.
- *int* ***sigfillset****(sigset_t* sigmask);*

  Sets all the signal flags in the *sigmask* argument.
- *int* ***sigismember****(sigset_t* sigmask, const int signal_num);*

  Return 1 if the specified signal is a member of the specified set, or 0 if it is not. Otherwise, it return -1, to indicate the error.

sigprocmask Example

```
int main( )
{
sigset_t  sigmask;
sigemptyset(&sigmask);    /*initialize set */

    if(sigprocmask(0,0,&sigmask)==-1)/*get current signal mask*/
    {
     perro("sigprocmask");
     exit(1);
    }
    else
     sigaddset(&sigmask, SIGINT); /* set SIGINT flag*/

sigdelset(&sigmask, SIGSEGV); /* clear SIGSEGV flag */

    if (sigprocmask(SIG_SETMASK,&sigmask,0) == -1)
     perro("sigprocmask"); /* set a new signal mask */
}
```

sigprocmask Example

- The example checks whether the SIGINT signal is present in a process signal mask and
- Adds it to the mask if it is not there.
- It clears the SIGSEGV signal from the process signal mask.