Signal Management

Signals

- A signal is a notification to a process that an event has occurred.
- Signals are sometimes described as software interrupts.
- A signal may be received either synchronously or asynchronously, depending on the source of and the reason for the event being signaled.

1)Synchronous signals are delivered to the same process that performed the operation that caused the signal .

2)When a signal is generated by an event external to a running process, that process receives the signal asynchronously.

Signals (contd ...)

- The following small code tries to divide by zero resulting in the program termination by floating point exception signal (SIGFPE). This signal occurred synchronously.

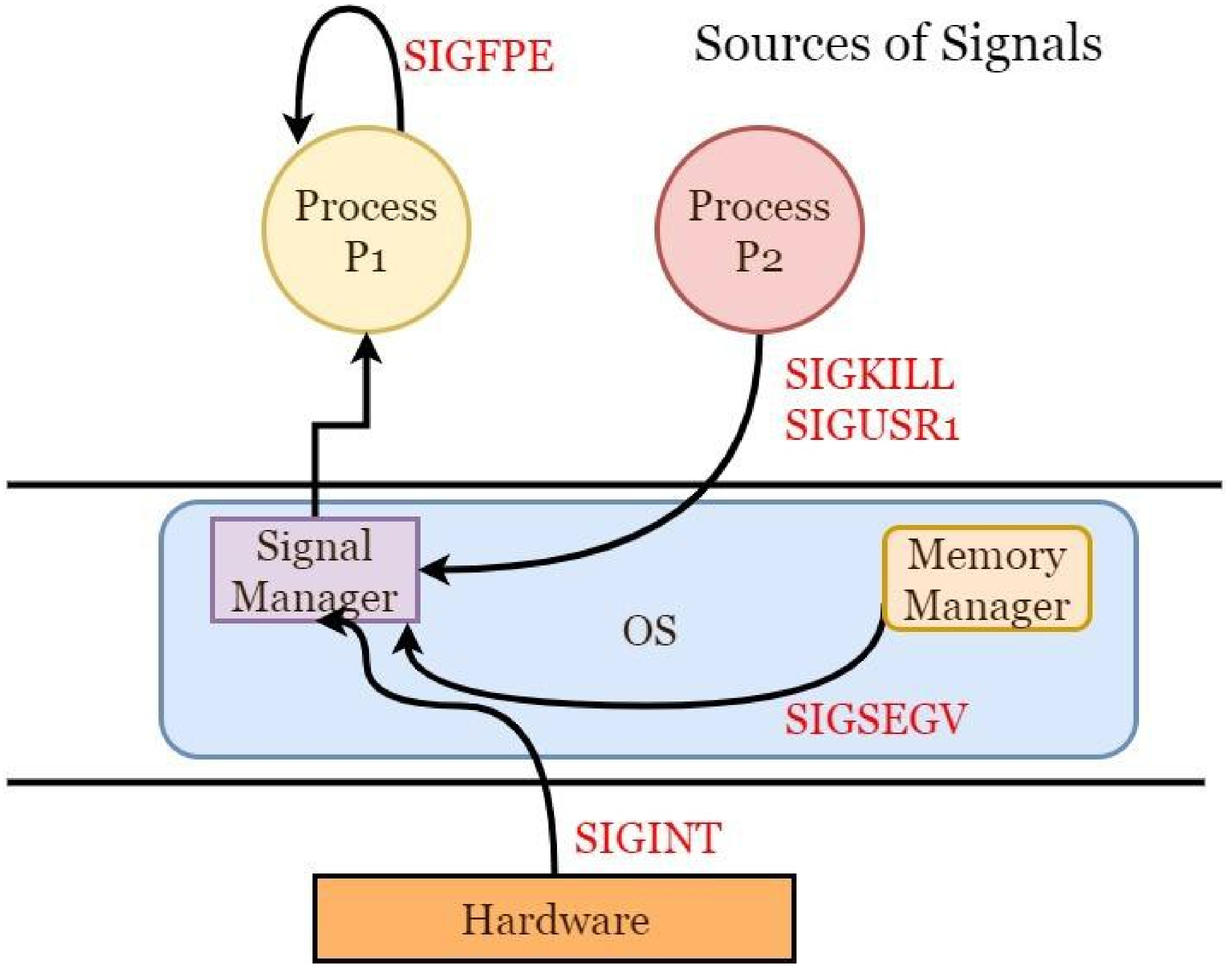
```
main()
{
int a=5,b=0,c;
 c=a/b;
 printf("%d\n",c);
 }
$ ./a.out  Floating point exception (core dumped)
```

Signals (contd ...)

• When you press ^c for infinitely running program, the program terminates on reception of quit process signal (SIGINT). This is asynchronous signal.

```
main()
{
 printf("Hi...\n");
 while(1);
 }
$ ./a.out  Hi... ^\c
```

Signals (contd ...)



Signals (contd ...)

- Each signal is defined as a unique (small) integer, starting sequentially from 1.
- These integers are defined in <signal.h> with symbolic names of the form SIGxxxx.
- . For example, when the user types the interrupt character (^c), SIGINT (signal number 2) is delivered to a process.
- Signals fall into two broad categories. The first set constitutes the traditional or standard signals, which are used by the kernel to notify processes of events. On Linux, the standard signals are numbered from 1 to 31.

Signals (contd …)

- The other set Linux kernel defines different real time signals, generally numbered from 32 to 63.
- We can list the signals on your system with command —kill -l .

```
$ kill -l
1) SIGHUP     2) SIGINT   3) SIGQUIT    4) SIGILL   5) SIGTRAP
6) SIGABRT   7) SIGBUS   8) SIGFPE      9) SIGKILL 10) SIGUSR1
11) SIGSEGV   12) SIGUSR2      13) SIGPIPE         14) SIGALRM
15) SIGTERM  16) SIGSTKFLT   17) SIGCHLD    18) SIGCONT
19) SIGSTOP  20) SIGTSTP       21) SIGTTIN     22) SIGTTOU
 23) SIGURG   24) SIGXCPU    25) SIGXFSZ …. 64) SIGRTMAX
```

Signals (contd …)

- All signals, whether synchronous or asynchronous, follow the same pattern:

  1) A signal is said to be generated by some event.

  2) Once generated, a signal is later delivered to a process.

  3) The process then takes some action in response to the signal i.e. signal is handled.

- Between the time it is generated and the time it is delivered, a signal is said to be pending.
- life of signal starts when signal is generated and its life ends when signal manager delivers it to respective process.

Signals (contd ...)

- Every signal may be handled by one of two possible handlers:

1) A default signal handler .
2) A user-defined signal handler.

- Every signal has a default signal handler that is run by the kernel when handling that signal. This default action can be overridden by a user-defined signal handler that is called to handle the signal.

Signals (contd ...)

- Upon delivery of a signal, a process carries out one of the following default actions, depending on the signal:

1) Ignore: The signal is ignored.

2) Terminate: The process is terminated (killed).

3) Core Dump: A core dump file is generated, and the process is terminated.

4)Stop: The process is stopped—execution of the process is suspended.

5)Continue: Execution of the process is resumed after previously being stopped.

Signals (contd ...)

- Instead of accepting the default for a particular signal, a program can change the action that occurs when the signal is delivered. This is known as setting the disposition of the signal. A program can set one of the following dispositions for a signal:

1) SIG_DFL: The default action should occur. This is useful to undo an earlier change of the disposition of the signal to something other than its default.

2) SIG_IGN: The signal is ignored. This is useful for a signal whose default action would be to terminate the process.

3) A signal handler is executed.

Sending and Receiving signals

•To send signals from one process to other process you can use a system call kill( ).

```
#include <signal.h>

int kill(pid_t pid, int sig);
```

returns 0 on success, or –1 on error .

•If no process matches the specified pid, kill() fails and sets errno to ESRCH (—No such process).

raise()… a library function

- Sometimes, it is useful for a process to send a signal to itself. The raise() function performs this task.

```
#include <signal.h>
int raise(int sig);
```

returns 0 on success, or nonzero on error

- a call to raise() is equivalent to the following call to kill()

```
kill(getpid(), sig);
```