Chapter 7.4

Signal System Calls

IPCs

One important job of Operating Systems is to manage communication between processes. This is called Inter-Process Communication (IPC).

System calls from the O/S kernel to a user process is sometimes referred to as a signal.

Signal Types

Signals can be generated in three different ways:

User – The user can signal the O/S that it wants the user process to terminate. (`CTRL-C`).

Kernel – The kernel can send a signal to the user process when it does something wrong. (User Process divides by zero.)

Other Process – Send a signal using the `kill()` system call (Parent process terminates Child process.)

Signal Types

Signals can divided into two types:

Synchronous – The signal happens at a predictable time or interval. Deterministic.
Ex. Floating Point Arithmetic Error.

Asynchronous – The signal happens at an unpredictable time. Random.
Ex. `CTRL-C` *input from user.*

Signal Handling

The `signal()` function determines what happens *when* a process receives a signal. (It *does not produce* a signal.)

```
#include <signal.h>
typedef void(*s_func)(int);
s_func signal(int num, s_func func);
```

`num` – The signal index. What type of signal.

`func` – What to execute when the signal happens.

Signal Handling

The process has three choices as to what to do when a `signal()` happens.

```
signal(SIGINT, SIG_DFL);
signal(SIGINT, SIG_IGN);
signal(SIGINT, sig_func);
```

When receive `SIGINT (CTRL-C)`, do `SIG_DFL` i.e. what O/S wants (usually death).

When receiving `SIGINT (CTRL-C)`, do `SIG_IGN` i.e. Ignore the signal altogether.

When receiving `SIGINT (CTRL-C)`, execute the function `sig_func.`

Signal Handling

Notice that the signal function `signal(signum, handler)` does not ***produce*** a signal.

It simply defines what the Operating System is to **do** when the specified signal happens.

`kill(pid, signum)` send a signal.

`alarm(secs)` send a signal to yourself

`pause()` block and wait for a signal.

Signal Example 1

Consider the program `signal1.c.`

```
signal(SIGINT,(fptr)SignalHandler);

while (1)
{ printf("Can you hear me now?\n");
  sleep(1);
}
```

What happens when we run

```
kill -s 2 pid#
```

in another window?

Notice use of **`static int i`**

Signal Example 2

Now Consider the program `signal2.c`.

```
signal(SIGUSR1, (fptr)Left);
signal(SIGUSR2, (fptr)Right);
```

Notice use of **`global`** variables

What happens when we run

```
kill -10 pid#
```

or

```
kill -12 pid#
```

Signal Example 3

Now Consider the program `signal3.c.`

Child waits for a signal

```
    signal(SIGUSR1, (fptr)f);
```

Parent uses:

`scanf` *on* `stdin` (`scanf` ***blocks***)

`kill` ***to send signal to child***

```
    kill(i, SIGUSR1);
// i is pid of child process
```

Notice use of **`fork`** and **`global`** variable

Signal Example 4

Signals can be sent either from child to parent or parent to child

Provides a very limited form of interprocess communication

signal4.c

Child

```
signal(SIGUSR1, f1);
kill(parent_pid, SIGUSER2);
```

Parent

Notice `pause()` – wait for signal

```
signal(SIGUSR2, f2);
kill(child_pid, SIGUSR1);
```

Use Socket system calls for interprocess communication (over a network)

`alarm()` Function

C allows for an `alarm()` function to send an interrupt to a process after an elapsed number of seconds.

`alarm.c` `alarm2.c`

```
AlarmHandler(void)
{ printf(“Alarm!\n”);
  alarm(1 + rand()%5);
}

signal(SIGALRM,(fptr)AlarmHandler);

alarm(1);
```

Signal.h Definitions

```
#define SIGHUP     1
#define SIGINT     2
#define SIGQUIT    3
#define SIGILL     4
#define SIGTRAP    5
#define SIGABRT    6
#define SIGIOT     6
#define SIGUNUSED  7
#define SIGFPE     8
#define SIGKILL    9
#define SIGUSR1   10
#define SIGSEGV   11
#define SIGSEGV   11
#define SIGUSR2   12
#define SIGPIPE   13
#define SIGALRM   14
#define SIGTERM   15
#define SIGSTKFLT 16
#define SIGCHLD   17
#define SIGCONT   18
#define SIGSTOP   19
#define SIGTSTP   20
#define SIGTTIN   21
#define SIGTTOU   22
```

Signals (POSIX.1)

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25		Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.

Signal Example 5

What does the program `signal5.c` *do?*

```
signal(SIGINT, SIG_IGN);
```

Now look at `signalsegfault.c`.

```
signalsegfault.c
```