Signals

Introduction

- Signals
  - Software interrupts
  - provides a way of handling asynchronous events.
    - E.g. A user types the interrupt key to stop a program.

- Signal name
  - Begins with ‘SIG’.
    - E.g. SIGABRT, SIGTERM, SIGALRM, …
  - Is defined by positive integer constants in <signal.h>
    - E.g. #define SIGHUP   1
    - Depends on architecture and OS.

Introduction

- Examples of signal generation
  - When user press ‘Ctrl-C’ on the terminal.
    - Generates SIGINT signal.
  - When executes an invalid memory references.
    - Generates SIGSEGV signal. (SEGmentation Violation)
  - When superuser want to kill a process.
    - Generates SIGKILL signal.
  - When a process writes to a pipe after the reader has terminated.
    - Generates SIGPIPE signal.

Introduction

- Disposition of the signal(called action).
  - Ignore the signal
    - SIGKILL and SIGSTOP cannot be ignored.
  - Catch the signal
    - We should tell the kernel to call a signal handler function whenever the signal occurs.
  - Execute the default action
    - The default action for most signals is to terminate.

Signals

- Signals for terminating processes
  - SIGHUP
    - This signal is sent to the controlling process(session leader) associated with a controlling terminal if a disconnect is detected.
    - termination

Signals

- Signals for terminating processes(cont.)
  - SIGINT
    - It is often used to terminate a runaway program.
    - It is sent to all foreground processes.
    - [CTRL-C]
    - termination
  - SIGQUIT
    - Is similar to SIGINT, but generates a core file.
    - [CTRL-\]
    - termination with core

"core" means that a memory image of the process is left in the file named core of the current working directory.
It can be used for debugging.

Signals

- Signals for terminating processes(cont.)
  - SIGABRT
    - abnormal termination (abort()).
    - terminate
  - SIGKILL
    - irrevocable termination signal.
    - It provides the superuser with a sure way to kill process.
    - cannot be caught or ignored.
    - terminate
  - SIGTERM
    - default signal sent out by the kill command.
    - terminate

Signals

- Signals for terminating processes(cont.)
  - SIGCHLD(or SIGCLD)
    - When a process terminates, it is sent to parent.
    - Ignore
    - The parent must catch using wait().

Signals

- Signals for suspending or resuming.
  - SIGCONT
    - Continue a stopped process.
    - resume
  - SIGSTOP
    - Stop a process.
    - Cannot be caught or ignored.
    - suspend

Signals

- Signals for suspending or resuming(cont).
  - SIGTSTP
    - When we type the terminal suspend key.
    - [CTRL-Z]
    - suspend
  - SIGTTIN
    - When a background process tries to read from terminal.
    - suspend
  - SIGTTOU
    - When a background process tries to write to terminal.
    - suspend

Signals

- Signals triggered by a physical circumstance
  - SIGILL
    - illegal hardware instruction
    - terminate
  - SIGTRAP
    - An implementation-defined hardware fault.
    - use this signal to transfer control to a debugger when a breakpoint instruction is executed.
    - terminate with core
  - SIGBUS
    - bus error
    - terminate

Signals

- Signals triggered by a physical circumstance(cont.)
  - SIGFPE
    - arithmetic error (floating point exception)
    - terminate
  - SIGSEGV
    - Invalid memory reference
    - terminate with core

Signals

- Signals available for use by the programmer
  - SIGUSR1, SIGUSR2
    - User-defined signal, for use in application programs
    - terminate

- Signal generated when a pipe is closed
  - SIGPIPE
    - pipe without reader
    - terminate

- Refer the textbook for entire list of signals!

signal()

```
#include <signal.h>

void (*signal(int signo, void (*func)(int)))(int);
                              Returns: previous disposition of signal if OK, SIG_ERR on error
```

- installs a signal handler for the signal with *signo*.
  - *signo* is the name of the signal.
  - *func* is one of the followings.
    - SIG_IGN: Ignore the signal.
    - SIG_DFL: set the action of the signal to its default value.
    - a user-specified function(signal handler).
  - It possible to use one signal handler for several signals.
  - Return value is the previous signal handler.

signal()

- Example

```
#include        <signal.h>

void myhandler(int signo)
{
        switch (signo) {
        case SIGQUIT : printf("SIGQUIT(%d) is caught\n",SIGQUIT);
                break;
        case SIGTSTP : printf("SIGTSTP(%d) is caught\n",SIGTSTP);
                break;
        case SIGTERM : printf("SIGTERM(%d) is caught\n",SIGTERM);
                break;
        case SIGUSR1 : printf("SIGUSR1(%d) is caught\n",SIGUSR1);
                break;
        default: printf("other signal\n");
        }
        return;
}
```

signal()

Example(cont.)

```
int main(void)
{
        signal(SIGQUIT, myhandler);
        signal(SIGTSTP, SIG_DFL);                       //use default handler
        signal(SIGTERM, myhandler);
        signal(SIGUSR1, myhandler);

        for (;;)
                pause();
}
```

Stop until it receive a signal.

signal()

- 실행

```
$ ./a.out
^\
SIGQUIT(3) is caught
^Z
[1]+  Stopped                 ./a.out
$ ps
  PID TTY          TIME CMD
15554 pts/2    00:00:00 bash
15587 pts/2    00:00:00 a.out
15588 pts/2    00:00:00 ps
$ kill 15587
SIGTERM(15) is caught
$ kill -USR1 15587
SIGUSR1(10) is caught
$
```

kill()

```
#include <signal.h>

int kill(pid_t pid, int signo);
                                        Both return: 0 if OK, -1 on error
```

- Sends a signal to a process or a group of processes.

kill()

- *pid* argument
  - pid > 0
    - The signal is sent to process with *pid*.
  - pid == 0
    - The signal is sent to all processes in the process group of the current process.
  - pid == –1
    - The signal is sent to all processes on the system for which the sender has permission to send the signal.
  - pid < -1
    - The signal is sent to all processes whose process group ID equals the absolute value of *pid*.

raise()

```
#include <signal.h>

int raise(int signo);
                                        Both return: 0 if OK, -1 on error
```

- Sends a signal to itself.
  - raise(signo); is equivalent to kill(getpid(), signo);

alarm()

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
                    Returns: 0 or number of seconds until previously set alarm
```

- Set a timer that will expire at a specified time in the future.
  - When the timer expires, SIGALRM is generated.
  - Default action is to terminate the process, but most processes catch this signal.
  - There is only one alarm clock per process.
    - If, when we call alarm, a previously registered alarm clock for the process has not yet expired, the number of seconds left is returned. The previously registered alarm clock is replaced by the new one.

pause()

```
#include <unistd.h>

int pause(void);
                                        Returns: -1 with errno set to EINTR
```

- Suspends the calling process until a signal is caught.

abort()

```
#include <stdlib.h>

void abort(void);
                                        This function never returns
```

- Sends the SIGABRT to the caller.

sleep()

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
                                        Returns: 0 or number of unslept seconds
```

- Causes the calling process to be suspended until
  - the amount of time specified by seconds has elapsed, or
  - a signal is caught by the process.
  - return value
    - 0 if the requested time has elapsed, or the number of seconds left to sleep.