SIGNAL HANDLING

UNIX Programming 2014 Fall by Euiseong Seo

What Are Signals?

- A signal is essentially an asynchronous message sent to a process
- Simple message, only carrying an integer
- Signals may be initiated by followings
  - OS due to hardware exceptions
  - User input such as CTRL+C
  - Other processes via kill(2)
  - Alarm by setitimer(2)

User-Initiated Signals

- SIGINT
  - Interrupt
  - CTRL+C
- SIGQUIT
  - Quit
  - CTRL+\
- SIGSTOP
  - Stop
  - CTRL+Z

Behavior on Signal Reception

1. Default action
   A. Ignore
   B. Terminate (some with core dump)
   C. Stop / continue
2. Ignore signal
3. Catch / handle signal
   - By signal handler function
4. Block / mask / hold signal

Sending a Signal

- kill(2) sends a signal to other processes
- A process can send a signal to itself using raise(3)
- Prototype

```
#include <signal.h>
int kill(pid_t pid, int signo);
int raise(int signo);
```

- abort() actually sends a SIGABRT to current process using kill()

```
#include <stdlib.h>
int abort(void)
{
    return kill(getpid(), SIGABRT);
}
```

Signal Sets

- Signals are normally dealt with by name or number, or in groups
- POSIX defines a data type, sigset_t, to represent multiple signals
- The following routines work with sigset_t

```
#include <signal.h>

sigset_t set; /* opaque bit set, one bit represents each signal */

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

Signal Sets

- Example

```
main()
{
    sigset_t set1;

    /* Clear set1 (all bits = 0) */
    sigemptyset( &set1 );

    /* Express interest in SIGINT */
    sigaddset( &set1, SIGINT );

    /* Express interest in SIGQUIT */
    sigaddset( &set1, SIGQUIT );
}
```

Blocking Signals

- If you do not want to take a signal or handle a signal during a certain region of code, you can block it with sigprocmask(2)
- Blocking a signal causes signal action not to occur until you unblock it
- Prototype

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Blocking Signals

- how parameter can be one of followings
  - SIG_BLOCK: add set to current block set
  - SIG_UNBLOCK: remove set from current block set
  - SIG_SETMASK: set replaces current block set
- Example

```
sigset_t toblock, oldblock;

sigemptyset(&toblock);
sigaddset(&toblock, SIGINT);
sigaddset(&toblock, SIGQUIT);
sigprocmask(SIG_BLOCK, &toblock, &oldblock);

sigprocmask(SIG_SETMASK, &oldblock, (Sigset_t *)NULL);
```

Blocking Signals

- Example

```
#include <signal.h>
#include <stdio.h>

main()
{
    int i;
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigprocmask(SIG_BLOCK, &set, (sigset_t *) NULL);
    for(i = 0; i< 100000000 ; i++);
    sigprocmask(SIG_UNBLOCK, &set, (sigset_t *)NULL);
}
```

Pending Signals

- sigpending(2) can be used to determine set of pending signals, which are blocked from delivery and currently pending
- Prototype

```
#include <signal.h>

int sigpending(sigset_t *set);
```

Catching a Signal (Old Skool)

- You can install a function to be called when a specific signal arrives
- This is called catching a signal
- sigset(2) or signal(2) can catch signals
- Prototype

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
sighandler_t sigset(int sig, sighandler_t disp);
```

Catching a Signal (Old Skool)

- Handler can be one of followings
  - SIG_IGN
  - SIG_DFL
  - function
- sigset
  - Keep signal handler after handling signal
  - Block delivered signal while handler is invoked
- signal
  - Restore signal handler after first time signal arrives
  - Does not block delivered signal

Catching a Signal (New Skool)

- sigaction(2) allows more control than signal(2) with sigaction structure
- Prototype

```
#include <signal.h>

int sigaction(int signum, const struct sigaction *act,
struct sigaction *oldact);

struct sigaction {
    void     (*sa_handler)(int);
    sigset_t   sa_mask;
    int        sa_flags;
};
```

Catching a Signal (New Skool)

- sa_flags field is a bitwise combination of followings
  - SA_RESETHAND: Restore signal action to default upon entry to signal handler
  - SA_NODEFER: Do not block delivery of the same signal during signal handling
  - SA_NOCLDSTOP: If signum is SIGCHLD, do not receive notification when child processes stop or resume
  - SA_RESTART: A system call that was interrupted is automatically restarted

Catching a Signal (New Skool)

- Example

```
struct sigaction act;
sigset_t set;
void handler(int signo);

sigemptyset(&act.sa_mask);
/* also block SIGQUIT while handling SIGINT */
sigaddset(&act.sa_mask, SIGQUIT);
act.sa_flags = 0;
act.sa_handler = handler;
sigaction(SIGINT, &act, (struct sigaction *)NULL);
```

Signal Handler

- A signal handler is a function
- When catching a signal, system forces your process to suspend what it was doing and call this function
- When a signal handler returns, process resumes where it left off
- Prototype

```
void handler(int signo)
```

Signal Handler

- You should be careful about what you do in a signal handler
  - A handler can be interrupted by following signal
  - A system call may cause a signal too
  - E.g) malloc in a handler generates a malloc call again, what will happen?
- POSIX defines a set of functions that can be reentrant and thus safe to be used in signal handlers
- Usually, it is desirable to have signal handlers to set flags and return immediately

Signal Handler

- Example

```
int count = 0;
void handler(int signo) {
    count ++;
    write(STDOUT_FILENO, "  OUCH!!\n", 10);
}

main()
{
    int i;
    struct sigaction act;
    act.sa_handler = handler;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, SIGINT);
    sigaddset(&act.sa_mask, SIGQUIT);
    act.sa_flags = 0;
    sigaction(SIGQUIT, &act, (struct sigaction *)NULL);
    sigaction(SIGINT, &act, (struct sigaction *)NULL);
    for(i = 0; i< 100000000 ; i++);
    printf("You pressed CTRL+C %d times\n", count);
}
```

Catching SIGCHLD

- When a child process dies or is suspended by a SIGTSTP
- Parent must take care of SIGCHLD or child will be a zombie or unreachable
- This is necessary for job controlling in your shell assignment

Catching SIGCHLD

- Example

```
void childhandler(int signo) {
    int status;
    pid_t pid;
    for(;;) {
        pid = waitpid(-1, &status, WNOHANG);
        if (pid == 0) {
            /* 1. no dead children, but some live ones */
            return;
        } else if (pid == -1 && errno == ECHILD) {
            /* 2. no more children, dead or running */
            return;
        } else if (pid == -1) {
            /* You should not get this */
            abort(1);
        }
        /* 3. Reaped status of one child */
        /* Save status for main program? */
    }
}
```

Alarms

- There are three types of alarms
  - Real-time alarm
    - Counts down wall-clock time
    - A SIGALRM is sent when it expires
  - Virtual-time alarm
    - Counts down while process is executing user code
    - A SIGVTALRM is sent when it expires
  - Profiling alarm
    - Counts down while process is executing in both user and system code
    - A SIGPROF is sent when it expires

Alarms

- The easiest way to schedule real-time alarms is using alarm(3)
- Prototype

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

- You must install a signal handler for SIGALRM before calling this function

Alarms

- More complicated but powerful way to set alarm timer is with interval timer functions, getitimer(2) and setitimer(2)
- Prototype

```
#include <sys/time.h>
struct itimerval {
    struct timeval it_interval; /* next value */
    struct timeval it_value;    /* current value */
};
struct timeval {
    time_t      tv_sec;         /* seconds */
    suseconds_t tv_usec;        /* microseconds */
};
int getitimer(int which, struct itimerval *curr_value);
int setitimer(int which, const struct itimerval *new_value,
struct itimerval *old_value);
```

Some Tips about Shell Assignment

- Your shell must ignore SIGTSP and SIGINT
- Install a SIGCHLD handler
  - Child may be suspended by SIGTSP
  - Child may be terminated
- You should react properly according to status info of a child process
- It would be desirable to create a new process group for a command line requiring fork