CSC 4304 - Systems Programming
Fall 2010


LECTURE - IX
SIGNALS


Tevfik Koşar


Louisiana State University
September 23rd, 2010

---

# What is a Signal?

- A signal is a software interrupt delivered to a process by the OS because:
    – it did something (segfault, FPE)
    – the user did something (pressed ^C)
    – another process wants to tell it something (SIGUSR?)
- Sending a signal is one way a process can communicate with other processes
- Some signals is asynchronous, they may be raised at any time (user pressing ^C)
- Some signals are directly related to hardware (illegal instruction, arithmetic exception, such as attempt to divide by 0) - synchronous signals
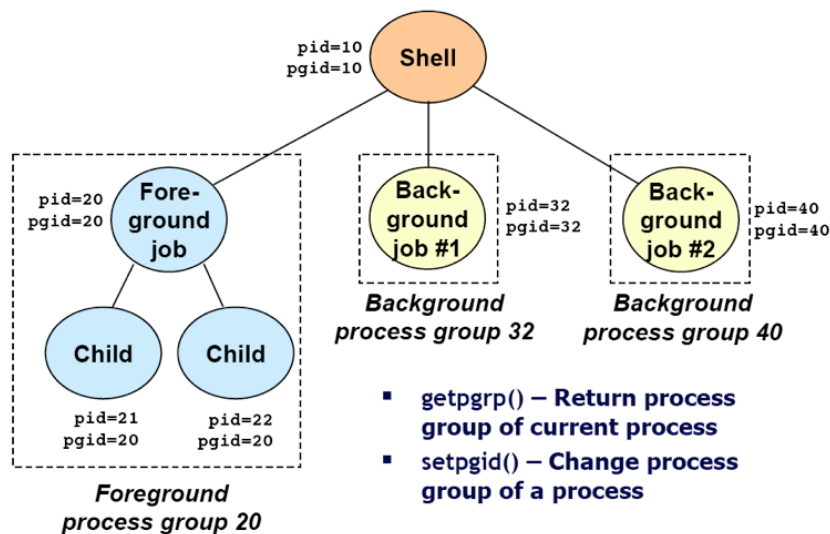
2

# Common Signals

- SIGHUP (1): hangup - sent to a process when its controlling terminal has disconnected
- SIGINT (2): interrupt - Ctrl-C pressed by user
- SIGQUIT (3): quit - Ctrl-\ pressed by user
- SIGILL (4): Illegal instruction (default core)
- SIGABRT (6): Abort process
- SIGKILL (9): kill (cannot be caught or ignored)
- SIGSEGV (11): Segmentation fault
- SIGALRM (14): Alarm cock timeout
- SIGUSR[1,2]: User-defined signals
- kill –l will list all signals

3

# Process Groups

- **Every process belongs to exactly one process group.**



pid=10
pgid=10 — Shell

pid=20
pgid=20 — **Fore-ground job**

Back-ground job #1 — pid=32
pgid=32

Back-ground job #2 — pid=40
pgid=40

*Background process group 32*

*Background process group 40*

Child — pid=21
pgid=20

Child — pid=22
pgid=20

*Foreground process group 20*

- getpgrp() – **Return process group of current process**
- setpgid() – **Change process group of a process**

4

# Sending Signals

- **Sending signals from the keyboard**
  - Typing **ctrl-c** (**ctrl-z**) sends a **SIGINT** (**SIGTSTP**) to every job in the foreground process group.
    - SIGINT: default action is to terminate each process.
    - SIGTSTP: default action is to stop (suspend) each process.

---
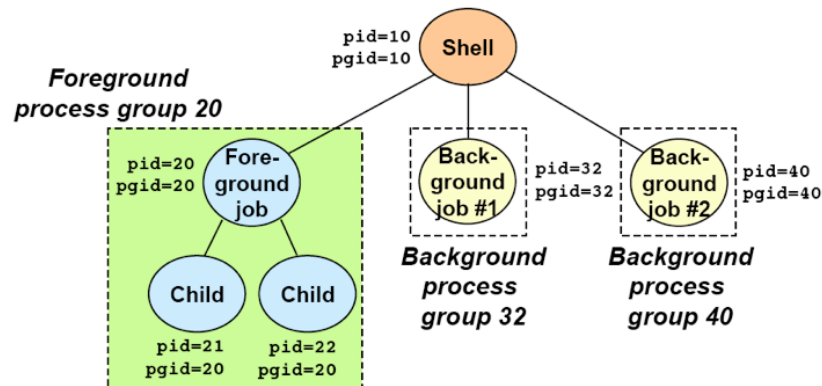
# Signals from Keyboard

The most common way of sending signals to processes is using the keyboard:

- Ctrl-C: Causes the system to send an `INT` signal (`SIGINT`) to the running process.
- Ctrl-Z: causes the system to send a `TSTP` signal (`SIGTSTP`) to the running process.
- Ctrl-\: causes the system to send a `ABRT` signal (`SIGABRT`) to the running process.

# Signals from Command-Line

- The `kill` command has the following format:

  `kill [options] pid`
  - `--l` lists all the signals you can send
  - `--signal` is a signal number
  - the default is to send a TERM signal to the process.

- The `fg` command will resume execution of the process (that was suspended with Ctrl-Z), by sending it a CONT signal.

```
$ kill 10231        // SIGTERM : default signal
$ kill -9 10231     // SIGKILL
```
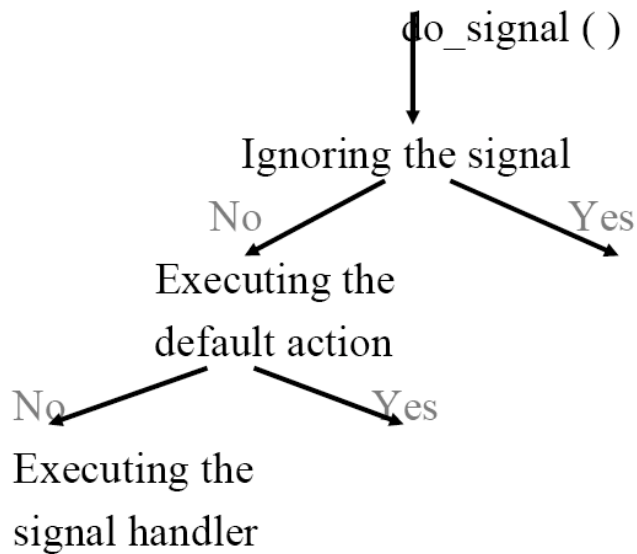
# Signal Disposition

- Ignore the signal (most signals can simply be ignored, except SIGKILL and SIGSTOP)
- Handle the signal disposition via a *signal handler* routine.  This allows us to gracefully shutdown a program when the user presses Ctrl-C (SIGINT).
- Block the signal.  In this case, the OS queues signals for possible later delivery
- Let the default apply (usually process termination)

# Actions on Signal

do_signal ( )

Ignoring the signal

No           Yes

Executing the
default action

No           Yes

Executing the
signal handler

---

# Default Actions

- <u>Abort</u> – terminate the process after generating a dump
- <u>Exit</u> – terminate the process without generating a dump
- <u>Ignore</u> – the signal is ignored
- <u>Stop</u> – suspends the process
- <u>Continue</u> – resumes the process, if suspended
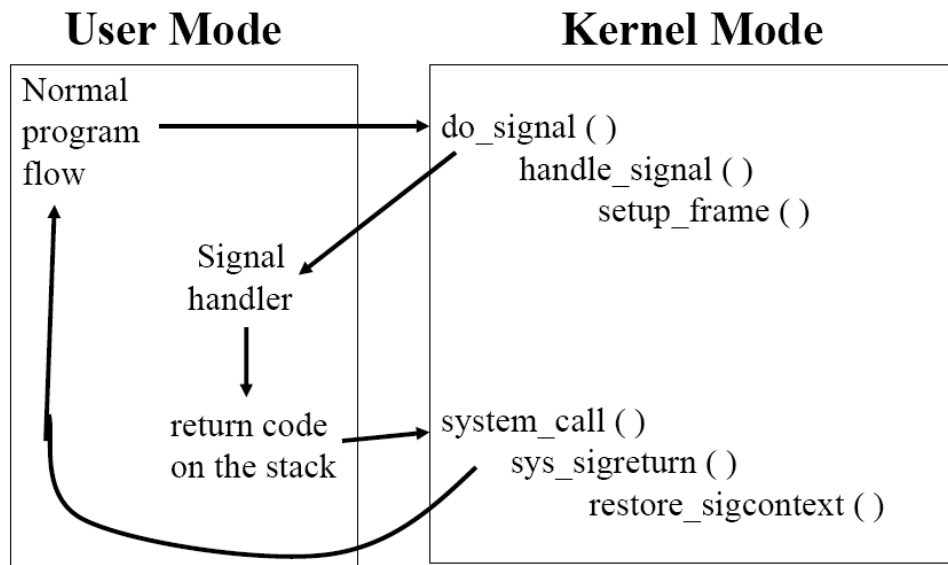
# Default Signal Actions (BSD)

| No | Name | Default Action | Description |
|----|------|----------------|-------------|
| 1 | SIGHUP | terminate process | terminal line hangup |
| 2 | SIGINT | terminate process | interrupt program |
| 3 | SIGQUIT | create core image | quit program |
| 4 | SIGILL | create core image | illegal instruction |
| 5 | SIGTRAP | create core image | trace trap |
| 6 | SIGABRT | create core image | abort program |
| 7 | SIGEMT | create core image | emulate instruction |
| 8 | SIGFPE | create core image | floating-point exception |
| 9 | SIGKILL | terminate process | kill program |
| 10 | SIGBUS | create core image | bus error |
| 11 | SIGSEGV | create core image | segmentation violation |
| 12 | SIGSYS | create core image | non-existent system call |
| 13 | SIGPIPE | terminate process | write on a pipe with |
| 14 | SIGALRM | terminate process | real-time timer expired |
| 15 | SIGTERM | terminate process | software termination |

# Default Signal Actions (BSD)

| No | Name | Default Action | Description |
|----|------|----------------|-------------|
| 16 | SIGURG | discard signal | urgent condition present on socket |
| 17 | SIGSTOP | stop process | stop (cannot be caught or ignored) |
| 18 | SIGTSTP | stop process | stop signalfrom keyb |
| 19 | SIGCONT | discard signal | continue after stop |
| 20 | SIGCHLD | discard signal | child status has changed |
| 21 | SIGTTIN | stop process | background read attempt |
| 22 | SIGTTOU | stop process | background write attempt |
| 23 | SIGIO | discard signal | I/O is possible on a desc |
| 24 | SIGXCPU | terminate process | cpu time limit exceeded |
| 25 | SIGXFSZ | terminate process | file size limit exceeded |
| 26 | SIGVTALRM | terminate process | virtual time alarm |
| 27 | SIGPROF | terminate process | profiling timer alarm |
| 28 | SIGWINCH | discard signal | Window size change |
| 29 | SIGINFO | discard signal | status request from keyboard |
| 30 | SIGUSR1 | terminate process | User defined signal 1 |
| 31 | SIGUSR2 | terminate process | User defined signal 2 |
| 32 | SIGTHR | terminate process | thread interrupt |

# Catching the Signal

**User Mode**                    **Kernel Mode**

Normal
program          →         do_signal ( )
flow                              handle_signal ( )
                                       setup_frame ( )

        Signal
        handler

        return code         →    system_call ( )
        on the stack                 sys_sigreturn ( )
                                          restore_sigcontext ( )

13

---

# Changing Default Action

- typedef void (*sig_t) (int);
- sig_t **signal**(int sig, sig_t func);

**Actions:**

•SIG_DSL: Reset to default Action
•SIG_IGN: Ignore Signal
•func(): user defined function

14

# Non-Catchable Signals

- Most signals may be caught by the process, but there are a few signals that the process cannot catch, and cause the process to terminate.
  - For example: KILL and STOP.
- If you install no signal handlers of your own the runtime environment sets up a set of default signal handlers.
  - For example:
    - The default signal handler for the TERM signal calls the exit().
    - The default handler for the ABRT is to dump the process's memory image into a file, and then exit.

15

# Catching a Signal

```
main(int ac, char *av[])
{
        void    inthandler(int);
        void    quithandler(int);
        char    input[100];

        signal( SIGINT,  inthandler );    //set trap
        signal( SIGQUIT, quithandler );   //set trap

        do {
                printf("\nType a message\n");
                if ( gets(input) == NULL )
                        perror("Saw EOF ");
                else
                        printf("You typed: %s\n", input);
        }
        while( strcmp( input , "quit" ) != 0 );
}
```

16

# Catching a Signal *(cont.)*

```
void inthandler(int s)
{
        printf(" Received signal %d .. waiting\n", s );
        sleep(2);
        printf("  Leaving inthandler \n");

}


void quithandler(int s)
{
        printf(" Received signal %d .. waiting\n", s );
        sleep(3);
        printf("  Leaving quithandler \n");
        exit(0)
}
```

17

# User Defined Signals

```
main(int ac, char *av[])
{
        void    signalhandler(int);

        signal( SIGUSR1,  signalhandler );
        signal( SIGUSR2,  signalhandler );

        while(1) pause();
}


void signalhandler(int s)
{
        printf(" Received signal %d\n", s );

}
```

18

# STOP & CONT Signals

```
main(int ac, char *av[])
{
    signal( SIGSTOP,  signalhandler );
    signal( SIGCONT,  signalhandler );

    int i=0;
    while(1){
                printf("i=%d\n", i++);
                sleep(1);
    }
}


void signalhandler(int s)
{
        printf(" Received signal %d\n", s );
}
```

19

# Alarming Signals

- SIGALRM can be used as a kind of "alarm clock" for a process
- By setting a disposition for SIGALRM, a process can set an alarm to go off in x seconds with the call:
  – unsigned int alarm(unsigned int numseconds)
- Alarms can be interrupted by other signals
- *Examples:*  mysleep.c, impatient.c

20

# Alarm Signal

```
main()
{
        void    wakeup();

        printf("about to sleep for 4 seconds\n");
        signal(SIGALRM, wakeup);      /* catch it    */
        alarm(4);                     /* set clock   */
        pause();                      /* sleep */
        printf("Morning so soon?\n");      /* back to work */
}


void
wakeup()
{
        printf("Wakeup: Alarm received from kernel!\n");

}
```

21

---

# Interval Timers

```
  #include <sys/time.h>
• int getitimer(int which, struct itimerval *value);
• int setitimer(int which, const struct itimerval *value,
               struct itimerval *ovalue);
```

**Three Timers:**
- ITIMER_REAL:       decrements in real time
- ITIMER_VIRTUAL:  decrements only when the process is executing
- ITIMER_PROF:      decrements both when the process executes and when the system is executing on behalf of the process.

22

# Interval Timer Struct

```
struct itimerval {
        struct timeval it_interval; /* next value */
        struct timeval it_value;    /* current value */
    };


struct timeval {
        long tv_sec;             /* seconds */
        long tv_usec;             /* microseconds */
    };
```

23

# Interval Time Example

```
#include        <stdio.h>
#include        <signal.h>
#include        <sys/time.h>


void main()
{
      char    x[200];

      signal(SIGALRM, hello);
      set_ticker(5,1);

      while(1)
      {
            printf("enter a word: ");
            fgets(x, 200, stdin);
            printf(">>> %s", x);

      }
}
```

24

## Interval Time Example *(cont.)*

```
void set_ticker(int start, int interval)
{
        struct itimerval new_timeset;

        new_timeset.it_interval.tv_sec  = interval;
        new_timeset.it_interval.tv_usec = 0;
        new_timeset.it_value.tv_sec     = start  ;
        new_timeset.it_value.tv_usec    = 0 ;

        return setitimer(ITIMER_REAL, &new_timeset, NULL);
}
```

## Interval Time Example *(cont.)*
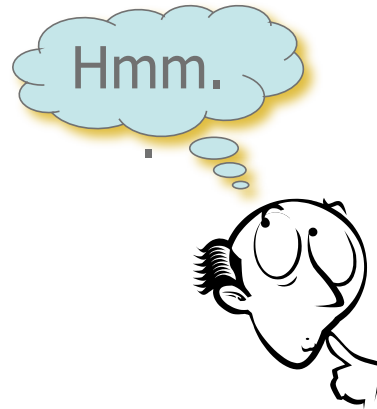
```
void hello(int s)
{
        static int      counter = 5;

        printf("hello\n");
        counter--;
        printf("* TICK: counter is now %d\n", counter);
        if ( counter == 0 ){
                printf("* TICK: Time is up!\n");
                exit(0);
                counter = 5;
        }
}
```

# Summary

- Signals
  - Signal Types & Actions
  - Catching Signals
  - STOP & CONT Signals
  - ALARM Signals
  - Interval Timers
  - Generating & Catching Signals

- Read Ch 10 from Stevens Book

Hmm.

# Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), B. Knicki (WPI), and M. Shcklette (UChicago).