

CSC 4304 - Systems Programming
Fall 2008

LECTURE - X SIGNALS

Tevfik Koşar

Louisiana State University
October 7th, 2008

Revisiting Symbolic Links

Getting the actual filename referred by symbolic link:

- command line:
 - `readlink` - display value of a symbolic link
- in C:
 - `#include <unistd.h>`
 - `int readlink(const char *path, char *buf, size_t bufsiz);`

Readlink Example

```
char *readlink_malloc (const char *filename)
{
    int size = 100;
    char *buffer = NULL;

    buffer = (char *) malloc (size);
    int nchars = readlink (filename, buffer, size);
    if (nchars < 0)
    {
        free (buffer);
        return NULL;
    }
    if (nchars < size)
        return buffer;
    size *= 2;
}
```

3

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 are 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

4

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 clock timeout
- SIGUSR[1,2]: User-defined signals
- kill -l will list all signals

5

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)

6

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 signal from 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

Changing Default Action

- `void (* signal (int sig, void (*func)(int)))(int);`
or:
- `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

9

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 );
}
```

10

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");
}
```

11

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 );
}
```

12

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 );
}
```

13

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

14

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");
}
```

15

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.

16

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 */
};
```

17

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);
    }
}
```

18

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);
}
```

19

Interval Time Example (*cont.*)

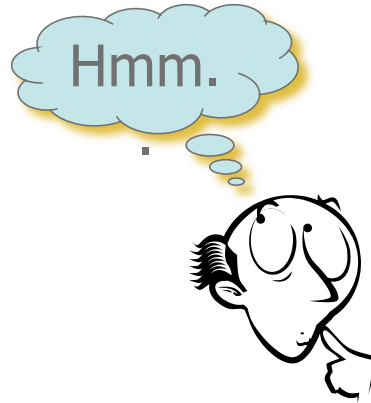
```
void hello(int s)
{
    static int counter = 5;

    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;
    }
}
```

20

Summary

- Signals
 - Signal Types & Actions
 - Catching Signals
 - STOP & CONT Signals
 - ALARM Signals
 - Interval Timers
- Read Ch 10 from Stevens Bok
- Source Codes available in:
/classes/cs4304/cs4304_kos/code



21

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).

22