Module 6

Signals

Overview

- Objectives
- Relevance

What Are Signals?

- System-Initiated
- User-Initiated
- Process-Initiated

Sending Signals

- `kill()`— Sends a signal to a process or process group
- `sigsend()`— Sends a signal to a process or group of processes
- `raise()`— Enables a process to send a signal to itself
- `abort()`— Sends a message to kill the current process

Using Signal Sets

mysigsetmask.c

```
1   #include <signal.h>
2
3   main() {
4
5     sigset_t set1;
6
7     /* Clear set1 (all bits = 0) */
8     sigemptyset( &set1 );
9
10    /* Express interest in SIGINT  */
11    sigaddset( &set1, SIGINT );
12
13    /* Express interest in SIGQUIT */
14    sigaddset( &set1, SIGQUIT );
15  }
```


Blocking Signals

- `sigprocmask()`— Blocks or unblocks a set of signals
- `sighold()`— Add one signal to list of blocked signals
- `sigrelse()`— Remove one signal from list of blocked signals
- `sigpending()`— Determines posted signals blocked and currently pending

Blocking Signals

```
1   sigset_t toblock, oldblock;
2
3   /* If the following code were interrupted by a */
4   /* control-C, or control-\, the data file may */
5   /* be munged. We have protected against this.*/
6
7   /* Initialize toblock to all 0's  */
8     sigemptyset(&toblock);
9     sigaddset(&toblock, SIGINT);
10    sigaddset(&toblock, SIGQUIT);
11
12    /* Add toblock to mask */
13    oldblock = sigprocmask(SIG_BLOCK, &toblock,
14      &oldblock);
15       ...
16    /* write out the database files */
17       ...
18    sigprocmask(SIG_SETMASK, &oldblock,
19      sigset_t *)NULL);
```


Catching Signals

- `sigset()`— Blocks delivered signal automatically while handler involved
- `signal()`— Resets handler automatically when signal arrives
- `sigaction()`— Provides more control with `sigaction` structure
- `sigignore()`— Ignore signal

Catching Signals

```
1   #include <stdio.h>
2   #include <signal.h>
3   #include <unistd.h>
4   #include <stdlib.h>
5
6   void handler(int signo) {
7
8     write(1, "Hi, I'm the signal handler!\n", 29);
9
10    /* If absent, execution resumes where interrupted */
11    exit(0);
12  }
13
14  main() {
15
16    void  (*ohand)(int);
17
18    /* Install handler() */
19    ohand = sigset(SIGINT, handler);
```



```
20
21    if (ohand == SIG_ERR) {
22      perror("sigset");
23    }
24
25    /* do some stuff */
26
27    /* Reinstall ohand */
28    (void)sigset(SIGINT, ohand);
29  }
```


Catching Signals

Flag	Description
SA_RESTART	If set, a system call that was interrupted is automatically restarted.
SA_RESETHAND	Causes the signal handling to revert to default and caught signal not blocked.
SA_NODEFER	If set, signal is not automatically blocked while it is being caught.
SA_SIGINFO	If set, additional information is passed to the signal handling function.
Others	See man sigaction.

Catching Signals

```
1   #include <signal.h>
2   ...
3     struct sigaction act;
4     void handler(int signo);
5
6     sigemptyset(&act.sa_mask);
7
8     /* also block SIGQUIT while handling SIGINT */
9     sigaddset(&act.sa_mask, SIGQUIT);
10    act.sa_flags = 0;
11    act.sa_handler = handler;
12    sigaction(SIGINT, &act, (struct sigaction *)NULL);
13    ......
```

Using Encapsulation

```
1   #include <stdio.h>
2   #include <signal.h>
3   #include <unistd.h>
4
5   /*********************
6   install_disp  -  An encapsulation function which
7   installs a signal handling disposition using the same
8   interface as signal and sigset BUT has BSD semantics
9   and is POSIXcompliant.Works on both SunOS 4.x and 5.x as
noted.
10  *********************/
11  void (*install_disp(int signo, void (*disp)(int)) ) (int) {
12
13    /* new action to be installed */
14    struct sigaction act;
15
16    /* current action to be replaced */
17    struct sigaction oldact;
18
```



```
19    act.sa_handler = disp;
20    sigemptyset(&act.sa_mask);
21
22    /* For 4.x */
23    act.sa_flags = 0;
24
25    /* OK for 5.x */
26    act.sa_flags = SA_RESTART;
27
28    if( sigaction( signo, &act, &oldact ) == -1 ) {
29      return( SIG_ERR );
30    }
31    return( oldact.sa_handler);
32  }
33
34  void handler(int signo) {
35
36     char *cp = "Hi, I'm in the handler\n";
37     write(1,cp,strlen(cp));
38  }
39
40  main() {
```



```
41
42      printf("\nPress ^\\ or ^Z to call handler,^C to
terminate\n\n");
43      install_disp(SIGQUIT, handler);
44      install_disp(SIGTSTP, handler);
45      while(1)
46         ;
47  }
```


Using Signal Handlers

- Defining signal handlers
- Writing signal handlers
- Catching `sigchild`

Using Signal Handlers

```
1   #include <signal.h>
2   #include <stdio.h>
3   #include <unistd.h>
4   #include <sys/types.h>
5   #include <time.h>
6
7   int   count;
8
9   void handler(int signo) {
10     count++;
11     write(1, "  OUCH!!\n", 10 );
12  }
13
14  main() {
15    int x;
16    struct sigaction act;
17    time_t start;
18
19    /* Install handler() for SIGINT and SIGQUIT */
```



```
20    act.sa_handler = handler;
21    sigemptyset(&act.sa_mask);
22    sigaddset(&act.sa_mask, SIGINT);
23    sigaddset(&act.sa_mask, SIGQUIT);
24    act.sa_flags = 0;
25    sigaction(SIGQUIT, &act, (struct sigaction *)NULL);
26    sigaction(SIGINT,  &act, (struct sigaction *)NULL);
27
28    printf("Do not press control-C or control-\\.\n");
29
30    /* Waste time while user may or may not ^C or ^\ */
31    start = time((time_d*)0);
32    while (difftime(time((time_t*)0), start) < 10);
33
34    if (count) {
35       printf("\nYou pressed ^C or ^\\ %d times!\n",
36         count);
37    } else {
38      printf ("\nYou follow directions well.\n");
39    }
40  }
```


childhandler.c

```
1   void childhandler(int signo) {
2
3     int status, saveerrno;
4     pid_t pid;
5     extern int errno;
6
7     saveerrno = errno;
8
9     /* get all outstanding terminated children */
10    for (;;) {
11      pid = waitpid(-1, &status, WNOHANG);
12      if (pid == 0) {
13        /* 1. no dead children, but some live ones */
14        break;
15      } else if (pid == -1 && errno == ECHILD) {
16        /* 2. no more children, dead or running */
17        break;
18      } else if (pid == -1) {
19        /* should not get this */
20        perror("waitpid");
```



```
21        abort();
22      }
23      /* 3. status contains the reaped status of one child
24      If desired, save status for main program. */
25    }
26    errno = saveerrno;
27    return;
28  }
```

Waiting For a Signal

```
1   sigset_t oldmask;
2   sigset_t blockmask;
3   sigset_t noracemask;
4
5   /* Install alarm_handler for signal SIGALRM */
6   sigset(SIGALRM, alarm_handler);
7
8   /* Fill mask with 1's */
9   sigfillset(&blockmask);
10  sigfillset(&noracemask);
11
12  /*Block every signal but SIGALRM in blockmask */
13  sigdelset(&blockmask,SIGALRM);
14
15  /*Avoid race condition; block all; save original */
16  sigprocmask(SIGBLOCK, &noracemask,&oldmask);
17
18  alarm(10);
19
```



```
20  /* Suspends the process until SIGALRM (10 seconds)
21  then calls alarm_handler() */
22  sigsuspend(&blockmask);
23  .....
24  sigprocmask(SIG_SETMASK, &oldmask, NULL);
```


Using Interval Timers

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <signal.h>
4
5   #define REPEAT_RATE 2 /* seconds. */
6
7   static char beep  = '\007';
8
9   void handler(int i) {
10    write (STDOUT_FILENO, &beep, sizeof(beep));
11  }
12
13  main(int argc, char *argv[]) {
14    struct itimerval it;/* Structure for loading the timer. */
15    sigset_t mask;/* Mask for blocking signals. */
16    int seconds;
17
18    if ((argc != 2) || ((seconds = atoi(argv[1])) <= 0)) {
19      fprintf (stderr, "Usage: %s <seconds>\n", argv[0]);
```



```
20      exit (1);
21    }
22
23    /* Set up mask of signals to block.
24    Block all except the SIGALRM and ^C. */
25    sigfillset(&mask);
26    sigdelset(&mask,SIGALRM);
27    sigdelset(&mask,SIGINT);
28
29    /* Install handler to receive the alarm signal */
30    (void)sigset(SIGALRM, handler);
31
32    /* Init structure to load into setitimer. */
33    it.it_value.tv_sec = seconds;
34    it.it_value.tv_usec = 0;
35    it.it_interval.tv_sec = REPEAT_RATE;
36    it.it_interval.tv_usec = 0;
37
38    /* Load and set timer. */
39    if (setitimer(ITIMER_REAL, &it,
40      (struct itimerval *)NULL) == -1) {
41      perror("setitimer");
```

```
42      exit(1);
43    }
44
45    /* Repeatedly block then call handler. */
46    while (1) {
47      sigsuspend(&mask);
48    }
49  }
```


Exercise: Signals

- Objectives
- Tasks
- Discussion
- Solutions