Computer Science & Engineering Department
I. I. T. Kharagpur

Operating System: CS33007
3rd Year CSE: 5th Semester (Autumn 2006 - 2007)
Lecture VII

Goutam Biswas *Date:* 4th September, 2006

1 Signals

Signals are software version of interrupt or exceptions, triggered by some event. The normal execution of the process is suspended and it is forced to handle the signal.

When the <Ctrl-C> is pressed by the user, the INT signal (SIGINT) is sent to the current process.

A signal may be raised by some error condition e.g. divide-by-zero (SIGFPE), memory protection violation (SIGSEGV), illegal instruction instruction (SIGILL).

Each signal is represented by a small positive integer (it has a a symbolic name). When a signal is sent to a process, its normal execution is suspended and it is forced to call a function called *signal handler*.

Different signals available in a system can be viewed by the command `kill -l`. It is also available from the man page of the syatem call `signal` (`man 7 signal`). A few of them are -

- `SIGINT` - interrupt from the keyboard, terminates the process (default).
- `SIGILL` - illegal instruction, terminate and core dupm.
- `SIGFPE` - floating-point exception, terminate and core dump.
- `SIGKILL` - kill signal, terminate, cannot be caught or ignored.
- `SIGSEGV` - invalid memory reference, terminate and core dump.
- `SIGALRM` - alarm clock, timer signal - terminate.
- `SIGUSR1` and `SIGUSR2` - user defined signals.
- `SIGSTOP` - stop executing, cannot be caught or ignored.
- `SIGCHLD` - child process may be stopped, ignored by default.
- `SIGCONT` - continue if stopped.

There are a few signals e.g. SIGKILL (9), SIGSTOP etc. that the process cannot catch.

1.1 How to Send Signal

- Signal can be sent from the keyboard e.g. `Ctrl-C` for SIGINT, `Ctrl-Z` (SIGTSTP) to suspend a running process, `CTRL-\` is very similar to `Ctrl-C`.

```
/*************************************
 * Ctrl-C terminates the current
 * process:
 * $ ./a.out
 * Press Ctrl-C to terminate
 * Execute again
 * Press Ctrl-Z to suspend
 * $ fg    to restart
 * Try Ctrl-\
 *************************************/
#include <stdio.h>

int main(){
        while(1) printf("What next!\n") ;
        return 0 ;
}
```

- Signal can be sent by executing the comand `kill` from the command line.

```
$ ps -A
        ............
29348 pts/2    00:00:00 vim
29350 pts/4    00:00:00 a.out
29352 pts/6    00:00:00 ps
$ kill -SIGINT 29350
$ ps -A
29348 pts/2    00:00:00 vim
29353 pts/6    00:00:00 ps
$
```

  We already have seen the command '`fg`'.

- Signal can be sent using system call `kill()` (37).

```
/***************************************
 * Sending SIGINT signal to a process:  *
 * $ cc -Wall killProg1.c -o killProg1  *
 * *************************************/

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

```
int main() {
int pid ;

printf("Enter a process ID:") ;
scanf("%d", &pid) ;

if(kill(pid, SIGINT) == -1)
perror("Kill fails\n") ;
return 0 ;
}
```

- A child process can send signal to the parent!

```
/********************************************
 * Send SIGINT to parent:  killProg2.c      *
 * ******************************************/

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

int main() {
     if(fork() > 0) { // Parent
               while(1) {
                    printf("\t\tIn parent: Infinite Loop!\n") ;
                    sleep(1) ;
               }
     }
     else { // Child
                int ppid ;

                sleep(5) ;
                printf("Parent ID is: %d\n", ppid = getppid()) ;
                printf("Sending SIGINT to the parent\n") ;
                kill(ppid, SIGINT) ;
                sleep(2) ;
                printf("Parent ID is: %d\n", getppid()) ;
     }
     return 0 ;
}
```

- Stop, continue and kill a child.

```
/**********************************************
 * Stop a child and then continue
```

```
 * $ cc -Wall sigStopCont.c
 * ********************************************/

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

int main() {
int chPID ;

if((chPID = fork()) != 0) { // Parent
printf("\tInside parent\n") ;
sleep(5) ;
printf("\tChild Stopped\n") ;
kill(chPID, SIGSTOP) ;
sleep(5) ;
printf("\tChild Continue\n") ;
kill(chPID, SIGCONT) ;
sleep(5) ;
kill(chPID, SIGKILL) ;
}
else { // Child
while(1) {
printf("\t\tInside Child\n") ;
sleep(1) ;
}
}
return 0 ;
}
```

1.2 Catch and Handle

When a signal is sent to a process, it is registered in the PCB (process table slot). The process is forced to react to the signal in three different ways -

1. The signal may be ignored.

2. There may be default action associated to the signal predefined by the kernel. The default action depends on the signal type.

3. The signal may be caught and a user defined function to handle the signal, known as the signal handler, can be invoked.

There are some signals that cannot be ignored or caught e.g. SIGKILL, SIGSTOP etc. are of that type. But there are some signals that can be caught by a process, it can be ignored or a user defined handler can be written for it.

- Ignoring SIGINT - the system call `signal()` (48)

```
/************************************************
 * SIGINT is sent to the process if <ctrl-C> is   *
 * pressed. It may be ignored: SIGINTprog1.c      *
 * ***********************************************/

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

int main() {
// signal(SIGINT, SIG_IGN) ; // Ignore <Ctrl-C>
while(1) printf("What next?\n") ;;
return 0 ;
}
```

- A new signal handler can be written for SIGINT.

```
/************************************************
 * SIGINT is sent to the process if <ctrl-C> is   *
 * pressed. There may be a different signal       *
 * handler: SIGINTprog2.c                         *
 * ***********************************************/
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void mySignalHandler(int n) {
signal(SIGINT, SIG_DFL) ; // <ctrl-C> default
printf("\t\tInside mySignalHandler()\n") ;
}
int main() {
signal(SIGINT, mySignalHandler) ; // <Ctrl-C> mySignalHandler()
while(1) { printf("What next?\n") ; sleep(1); }
return 0 ;
}
```

- Handler for segmentation violation SIGSEGV.

```
/****************************************
 * SIGSEGV handler
 * SIGSEGVprog1.c
 ****************************************/
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void mySEGVhandler(int n){
     signal(SIGSEGV, mySEGVhandler) ; // SEGV default
```

```
     printf("\t\tInside mySignalHandler()\n") ;
     _exit(0) ;
}

int main() {
        int *p = (int *)100 ;

        // signal(SIGSEGV, mySEGVhandler) ; // SEGV mySEGVhandler()
        *p = 10 ;
        printf("Not printed\n") ;
        return 0 ;
}
```

- By-pass the segmentation fault using `setjmp` and `longjmp`.

```
/*****************************************
 * SIGSEGV handler - setjmp, longjmp
 * SIGSEGVprog2.c
 *****************************************/
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <setjmp.h>

jmp_buf env ;

int fact(int n){
    if(n == 0) return 1 ;
    return n*fact(n-1) ;
}

void mySEGVhandler(int n){
     signal(SIGSEGV, mySEGVhandler) ; // SEGV default
     printf("\t\tInside mySignalHandler()\n") ;
     longjmp(env, 1) ;
}

int main() {
        int *p = (int *)100 ;

        // signal(SIGSEGV, mySEGVhandler) ; // SEGV mySEGVhandler()
        if(setjmp(env) == 0) *p = 10 ;
        printf("Print after exception: %d! = %d\n", 5, fact(5)) ;
        return 0 ;
}
```

- The function call `pause()` () makes the process wait for a signal

```
/*********************************************
 * Sending an alarm signal in future
 * $ cc -Wall alarmClock.c
 * ********************************************/

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void ringTheBell(int n){
printf("\t\t\tAlarm signal received\n") ;
}
int main() {
int chPID ;

if((chPID = fork()) != 0) { // Parent
printf("\tInside parent\n") ;
sleep(5) ;
kill(chPID, SIGALRM) ;
}
else { // Child
signal(SIGALRM, ringTheBell) ;
printf("\t\tInside child\n") ;
printf("\t\tNo alarm yet\n") ;
pause() ;  // wait for a signal
}
return 0 ;
}
```

- Floating-point Exception: SIGFPE

```
/***********************************************
 * SIGFPE is sent to the process if there is an   *
 * arithmetic error: SIGFPEprog1.c                *
 * **********************************************/

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

void mySIGFPEhandler(int sigNum) {
printf("\t\tDivide-by-zero\n") ;
exit(0) ;
}

int main() {
```

```
int n ;

signal(SIGFPE, mySIGFPEhandler) ;
printf("Enter an integer:") ;
scanf("%d", &n) ;
printf("1/n = %d\n", 1/n) ;
return 0 ;
}
```

- Two processes can communicate by sending signals.

```
/***********************************************
 * Communicating between parent and child
 * using SIGUSR1
 * SIGUSR1prog1.c
 ***********************************************/
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void sigUSR1handler(int n){
     signal(SIGUSR1, sigUSR1handler) ;
     printf("---> Parent receives\n") ;
}

void sigUSR2handler(int n){
     signal(SIGUSR2, sigUSR2handler) ;
     printf("\t\tChild receives ---> ") ;
}

int main()
{
    int chPID, count = 0 ;

    if((chPID = fork()) != 0) { // Parent
        signal(SIGUSR1, sigUSR1handler) ;
        while(1){
             printf("\t%d: P --> C ", ++count) ;
             // sleep(1) ;
             kill(chPID, SIGUSR2) ;
             pause() ;
        }
    }
    else { // Child
        int pPID = getppid() ;
        signal(SIGUSR2, sigUSR2handler) ;
```

```
        while(1){
            pause() ;
            printf("%d: C --> P\n", ++count) ;
            sleep(1) ;
            kill(pPID, SIGUSR1) ;
        }
    }
    return 0 ;
}
```

References

[1] http://users.actcom.co.il/ choo/lupg/tutorials/signals/signals-programming.html