

# BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

## IS F462 – Network Programming

### Lab#2

---

**Note: Please use programs under *Code\_lab2* directory supplied with this sheet. Do not copy from this sheet.**

The lab has the following objectives:

Giving practice programs for setjmp, zombie, exec, and signals

### Non-local goto:

Run the following program.

#### //jump.c

```
#include <stdio.h>
#include <stdlib.h>
#include <setjmp.h>
#include <math.h>

void func(void);
jmp_buf place;
double square1(double n2){
    if(n2<=0) longjmp(place, 2);
    return n2*n2;
}
double compute(int n){
    if (n<=0) longjmp(place, 1);
    double n1=sqrt((double)n);
    double n2=square1(n1);
    return(n2-n);
}

main(){
    int retval;
    int i=1, n;
    /*
     * First call returns 0,
     * a later longjmp will return non-zero.
     */
    if((retval=setjmp(place)) != 0){
        printf("Returned using longjmp\n");
        printf("Ret value: %d", retval);
    }


    while(i<100){
```

```

        printf("Enter value %d: ", i);
        scanf("%d", &n);
        double n3=compute(n);
        printf("Result=%ld\n", n3);
        i++;
    }
}

```

## Q?

1. Enter the value 0 or -1 and see the next i value being asked for. What do you notice from this? 
2. Is it possible for a `longjmp()` to have many places to jump to? How can you do this?

## Zombie Process:

### //zombie.c

```

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




int
main()
{
    pid_t pid;

    pid=fork();

    if(pid<0)
        perror("fork:");
    else if(pid==0){
        printf("In Child: pid = %d, parent pid= %d", getpid(), getppid());
        printf("child finishing...");
        exit(0);
    }
    else if (pid>0){
        while(1);
    }
}

```

## Q?

1. Run the above program. Open another terminal and run 'ps -al' command. What is the state of child process? 
2. Modify the above program to be free from creating zombie processes. 
3. Write an exit handler such that it clears all the zombies of this process. 



## Orphan Process:

Orphan process is one whose parent has got terminated and the init process has become the parent of it.

**//orphan.c**

```
# include <stdio.h>

int main()
{
    int pid;
    pid=getpid();

    printf("Current Process ID is : %d\n",pid);

    printf("[ Forking Child Process ... ] \n");
    pid=fork();
    if(pid < 0)
    {
        /* Process Creation Failed ... */

        exit(-1);
    }
    elseif(pid==0)
    {
        /* Child Process */

        printf("Child Process is Sleeping ...");
        sleep(5);

        printf("\nOrphan Child's Parent ID : %d",getppid());
    }
    else
    {
        /* Parent Process */

        printf("Parent Process Completed ...");
    }
    return 0;
}
```

# Q?

1. Run the above program. Open another terminal and run 'ps -al' command. What is the state of child process?





## exec() calls:

Consider the following program.

```
//excl.c
int
main ()
{
    execl ("/bin/ls", "ls", "-l", (char *) 0);
    printf ("hello");
}
```

## Q?

1. When you execute the above program do you see the 'hello' being printed? Why or why not? 
2. Modify the program so that you need not give the absolute path of ls program. Also the arguments are passed in vector. 
3. Why is it stated that exec calls don't return anything upon success?

Consider the following program to demonstrate the environment passing to a child process. Also consider the program given in lab1 to print command-line arguments and environment variables. Compile it to 'echoall' executable.

```
//exec.c
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>

char *env_init[] = { "USER=unknown", "PATH=/tmp", NULL };

int
main (void)
{
    pid_t pid;

    if ((pid = fork ()) < 0)
    {
        perror ("fork error");
    }
    else if (pid == 0)
    {
        /* specify pathname, specify environment */
        if (execle ("/home/students/f2007080/echoall", "echoall", "myarg1",
                    "MY ARG2", (char *) 0, env_init) < 0)
            perror ("execle error");
    }
    if (waitpid (pid, NULL, 0) < 0)
        perror ("wait error");

    if ((pid = fork ()) < 0)
    {
        perror ("fork error");
    }
}
```

```

    }
    else if (pid == 0)
    {
        /* specify filename, inherit environment */
        if (execlp ("echoall", "echoall", "only 1 arg", (char *) 0) < 0)
            perror ("execlp error");
    }

    exit (0);
}

```

## Q?

1. What did you observe in the output? You will see that the first child will inherit only that environment as specified in the command-line and the second will inherit the whole of the parent's environment.

Run the following program and see its results.

### //execlp.c

```

#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#define MAX 10
void pr_exit (int status);
main (int argc, char *argv[])
{
    pid_t pid[MAX];
    int i, status;
    for (i = 1; i < argc; i++)
    {
        if ((pid[i] = fork ()) < 0)
            perror ("fork error");
        else if (pid[i] == 0)
        {
            printf ("Executing %s \n", argv[i]);
            execlp (argv[i], argv[i], (char *) 0);
            perror ("error in execlp");
        }
        else if (pid[i] > 0)
        {
            wait (&status);
            pr_exit (status);
        }
    }
}

void
pr_exit (int status)
{
    if (WIFEXITED (status))
        printf ("normal termination, exit status = %d\n", WEXITSTATUS (status));
    else if (WIFSIGNALED (status))

```

```


    printf ("abnormal termination, signal number = %d \n", WTERMSIG (status));
else if (WIFSTOPPED (status))
    printf ("child stopped, signal number = %d\n", WSTOPSIG (status));
}

```

## Q?

1. The above program runs the commands that are specified on the command line;

*./a.out ls cut pwd*

The program forks a child to run each command. What is the exit status of the second command cut? 

2. Write a program called inf.c as follows.

```

main()
{
    while(1);
}

```

Compile it: `gcc inf.c -o inf`

Now execute `./a.out ls ./inf`

Open another terminal and find out pid of inf process using `ps -a` command. Kill the inf process using `kill -9 <pid of inf>`.

Now check the status reported by the a.out program. 


3. Write a small program like this in fpe.c. Compile it. `gcc fpe.c -o fpe`

```

main()
{
    int x;
    x=x/0;
}

```

Run this command. `./a.out fpe`. What do you observe?

Try to debug this program using gdb. 

4. Modify the above program such that it takes a command and its arguments on the input line just like shell.
5. Is it possible to run the commands in the background? *./a.out ls wc* In this, run ls and wc in the background which means the program should not wait for the result of the current command and proceeds to next command.
6. If you modify the program to be like in 3, then ensure that there will be no zombies created?

## **Signals:**


**Run the following program.**

**//signal.c**

```
#include <signal.h>
#include <stdio.h>
void int_handler(int signo);
main ()
{
    signal (SIGINT, int_handler);
    printf ("Entering infinite loop\n");
    while (1)
    {
        sleep (10);
    }
    printf (" This is unreachable\n");
}

/* will be called asynchronously, even during a sleep */
void int_handler(int signo)
{
    printf ("Running int_handler\n");
}
```

## **Q?**

- Press Ctrl-C (SIGINT) a few times. See what is happening. How to exit this process now? You can generate Ctrl-Z (SIGTSTP) or Ctrl-\ (SIGQUIT) to terminate the process.
- In the signal function, change the second argument to SIG\_DFL and run it. Now will it stand the Ctr-C signal? Similarly check by giving SIG\_IGN.
- Implement the handlers for SIGTSTP and SIGQUIT signals. If you handle all these three signals, is there way to terminate the process? 
- Modify the program to prevent any signal disturbing the sleep of main except the SIGKILL and SIGSTOP? After completing sleep for 10 seconds, check for the pending signals and print if there are any known signals pending at kernel. [Hint: use sigprocmask() see slides for syntax]
- Generally while writing C programs we often face segmentation fault error. When invalid memory reference occurs, SIGSEGV signal is generated. Consider the following program.

```
#include <stdio.h>
main()
{
    int n;
    printf("enter a number");
    scanf("%d", n);
}
```

Write a signal handler to catch SISEGV signal and see if you can continue further executing the program.

## alarm()

Consider the following program. The program demonstrates the use of alarm() as a timer.

### //alarm.c

```
#include <signal.h>

int n;

void sigalrm(int signo){
    alarm(1);
    n=n+1;
    printf("%d seconds elapsed\n",n);
}

main(){
    signal(SIGALRM, sigalrm);
    alarm(1);
    while(1) pause();
}
```

## Q?

1. Modify the program to implement sleep(long secs). The program should sleep for the specified number of seconds using alarm() and pause() calls.

## Synchronization using signals

### //sync.c

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
main ()
{
    int i = 0, j = 0;
    pid_t ret;
    int status;

    ret = fork ();
    if (ret == 0)
    {
        for (i = 0; i < 5000; i++)
            printf ("Child: %d\n", i);
        printf ("Child ends\n");
    }
    else
    {
        wait (&status);
    }
}
```



```

        printf ("Parent resumes.\n");
        for (j = 0; j < 5000; j++)
            printf ("Parent: %d\n", j);
    }
}

```

# Q?

1. Synchronize the printing of child and parent in such way that child and parent print the lines alternatively? (hint: use pause() and kill() calls)
2. Examine when fork () is called, does the child inherit the signal mask of parent? Also examine when you do exec() , does the new program inherit the current signal mask? [Hint: Use sigprocmask() to print the singnal mask]

## Signal Blocking & Unblocking

### //critical.c

```

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

void err_sys(char* str)
{
    perror(str);
    exit(-1);
}

static void  sig_quit(int);

int
main(void)
{
    sigset_t      newmask, oldmask, pendmask;

    if (signal(SIGQUIT, sig_quit) == SIG_ERR)
        err_sys("can't catch SIGQUIT");

    sigemptyset(&newmask);
    sigaddset(&newmask, SIGQUIT);
    /* block SIGQUIT and save current signal mask */
    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
        err_sys("SIG_BLOCK error");
    /* critical section starting*/

    sleep(5);                /* SIGQUIT here will remain pending */

    if (sigpending(&pendmask) < 0)
        err_sys("sigpending error");
    if (sigismember(&pendmask, SIGQUIT))
        printf("\nSIGQUIT pending\n");

    /*critical section ending*/

    /* reset signal mask which unblocks SIGQUIT */
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)

```

```

        err_sys("SIG_SETMASK error");
        printf("SIGQUIT unblocked\n");

        sleep(5);          /* SIGQUIT here will terminate with core file */

        exit(0);
    }

    static void
    sig_quit(int signo)
    {
        printf("caught SIGQUIT\n");

        if (signal(SIGQUIT, SIG_DFL) == SIG_ERR)
            err_sys("can't reset SIGQUIT");
        return;
    }

```

## Q?

- 1.** Observe the blocking and unblocking of signals within the critical section. After executing the program, press Ctrl-\ and see. If it is pressed during critical section, the signal will remain pending. If it is pressed after critical section, the signal will be directly delivered.
- 2.** Also block SIGINT signal while in critical section?
- 3.** In the above program only SIGQUIT signal is blocked. Block all signals being delivered while you are in critical section?
- 4.** In the above program, only SIGQUIT signal is being check for whether it is pending? How can you check all the signals that are pending?

**End of lab2**