Lab # 05

IPC-I (inter-process communication)

In this lab we will look at the following IPC techniques in detail.

i. **Signals**
ii. **Shared Memory**

1. Signals

Signals are an inter-process communication mechanism provided by operating systems which facilitate communication between processes. It is usually used for notifying a process regarding a certain event. So, we can say that signals are an event-notification system provided by the operating system. Event (interrupt or exception) is basically the situation where OS attention is required.

1.1 Signal Delivery Using Kill

Kill is the delivery mechanism for sending a signal to a process. Unlike the name, a kill () call is used only to send a signal to a process. It does not necessarily mean that a process is going to be killed (although it can do exactly that as well). The Kill facility can be used in two ways; as a command from your command prompt, or via the kill () call from your program.

The syntax of the kill command is as such:

Kill -s PID

Here, kill is the command itself, -s is an argument which specifies the type of signal to send, and PID is the integer identifier of the process to which a signal is going to be delivered. The list of signals for -s argument can be seen from the following:

Kill -l

Hence, supposing that we want to terminate a process, we may enter

Kill -9 12345

Where 12345 will be a process id of any active process in the system.

Usage of the Kill () system call is as such: kill (int, int);
For this to work, we will require the sys/types.h & signal.h C libraries. Here, the 1st parameter is the PID of the process to which signal is to be delivered, and the 2nd parameter is the integer

number of signal type (again, can be checkable from kill -l). As an example, if we want to send the Terminate signal to the current process, we will use

kill(getpid(),SIGTERM)

or

kill(getpid(),9)

1.2 Signal Handling Using Signal

Signal delivery is handled by the kill command or the kill () call. The process receiving the signal can behave in a number of ways, which is defined by the signal () call. The syntax of the call is as such:

Signal (int, conditions);

The signal will require the signal.h C library to work. From the syntax above, signal () is the system call, the 1st parameter int is the integer identifier of the respective signal which is sent, and the last parameter is either of the following:

i. SIG_DFL which will perform the default mechanism provided by the operating system for that particular signal

ii. SIG_IGN which will ignore that particular signal if it is delivered

iii. Any function name (for programmer-defined signal handling purposes)

As an example, we are going to send a process the SIGINT signal and count the total number of times that it is received. See the code below for this purpose:

Example 1:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
int sigCounter = 0;
void sigHandler(int sigNum)
{
printf("Signal received is %d\n", sigNum);
++sigCounter;
printf("Signals received %d\n", sigCounter);
}
int main()
{
```

```
signal(SIGINT, sigHandler);
while(1)
{
printf("Hello Dears\n");
printf("Hello Dears %d\n",getpid());
sleep(1);
}
return 0;
}
```

When we run the program, we are instructing our program that if in case the SIGINT signal is detected, we will perform the steps provided in the sigHandler function. As long as there is no event, the program will keep on executing the while loop. The event can be delivered by pressing CTRL+C. Try pressing CTRL+C with, and without the signal () call and you will understand the difference yourselves.

Also the following program can be used to send signal to the above program.

Example 2:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
while(1)
{
kill(PID,SIGINT);
printf("Hello Dears %d\n",getpid());
sleep(1);
}
return 0;
}
```

2. Shared Memory

- Region of Memory that is shared by cooperating processes
- Processes exchange Data by reading/writing to the shared region

Example 3:

shm_server.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define SHMSZ  27
main()
```

```
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;
    key = 5678;
  if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }
s = shm;

    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
    *s = NULL;
    while (*shm != '*')
        sleep(1);

    exit(0);
}
```

shm_client.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define SHMSZ     27
main()
{
    int shmid;
    key_t key;
    char *shm, *s;
    key = 5678;
    if ((shmid = shmget(key, SHMSZ, 0666)) < 0)
     {
        perror("shmget");
        exit(1);
    }
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);

   for (s = shm; *s != NULL; s++)
        putchar(*s);
        putchar('\n');
       *shm = '*';
        shmdt(shm);
shmctl(id,IPC_RMID,NULL);
       exit(0);

}
```