SLIIT
Discover Your Future

BSc (Hons) in Information Technology
IT– Year 2

Lab Exercise 5

IT2060 – Operating Systems and System Administration
Semester 1, 2022

Learning Objectives: In this lab, you will learn about Intercrosses communication with signals and pipes.

The Linux IPC (Inter-process communication) facilities provide a method for multiple processes to communicate with one another. There are several methods of IPC available to Linux C programmers:

- Pipes
- Signals
- Message queues
- Semaphore sets
- Shared memory segments
- Sockets

Pipes are known as the oldest communication mechanism under UNIX. To create a simple pipe with C, we make use of the pipe() system call. It takes a single argument, which is an array of two integers, and if successful, the array will contain two new file descriptors to be used for the pipeline. A pipe is created by calling the pipe () function in the following way.

```
int fd[2];
 if (pipe(fd) <  0)
        printf(“Error!\n”);
```

The pipe() function is invoked. This returns two valid file descriptors in the array given as the argument. The input of the first file descriptor (fd[0]) is the output of the second file descriptor (fd[1]).

Exercise 01: Write the following program and execute to understand the concept of pipe.

```
#include <stdio.h>
main()
{
  int pipefd[2];
  int i;
  char s[1000];
  char *s2;

  if (pipe(pipefd) <  0)
 {
    perror("pipe");
    exit(1);
  }
```

```
  s2 = "This is the message";

  write(pipefd[1], s2, strlen(s2));

  i = read(pipefd[0], s,  1000);
                    s[i] = '\0';

                  printf("Read %d bytes from the pipe: '%s'\n", i, s);

}
```

Exercise 02: Write the following program and execute to understand the concept of pipe with parent and child processes.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
        int     fd[2], nbytes;
        pid_t   childpid;
        char    string[] = "Hello, world!\n";
        char    readbuffer[80];
        pipe(fd);
                if((childpid = fork()) == -1)
                {
                        perror("fork");
                        exit(1);
                }
                if(childpid == 0)
                {
                        close(fd[0]);
                        write(fd[1], string, strlen(string));
                        exit(0);
                }
                else
                {
                       close(fd[1]);
                       nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
                       printf("Received string: %s",  readbuffer);
                 }
         return(0);
}
```

Lab Exercise 5

IT2060 – Operating Systems and System Administration
Semester 1, 2018

Signal Handling

Signals are the means to notify a process or thread of the occurrence of an event. Signals are one of the oldest inter-process communication methods used by Unix. Signals are a way of sending simple messages to processes. Most of these messages are already defined and can be found in <linux/signal.h>. However, signals can only be processed when the process is in user mode. If a signal has been sent to a process that is in kernel mode, it is dealt with immediately on returning to user mode.

Exercise 03:

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

void sigproc(void);
void quitproc(void);

main()
{
        signal(SIGINT, sigproc);
        signal(SIGQUIT, quitproc);
        printf("`ctrl-c disabled use ctrl. \\ to quit \n");
        for(;;); /* infinite loop  */}

        void sigproc()
        {
                signal(SIGINT, sigproc);
                printf("`you have pressed ctrl-c \n'");
        }

        void quitproc()
        {               printf("ctrl- \\ pressed to quit \n' ");
                        exit(0); /* normal exit status */
        }
}
```