

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS



A LAB REPORT ON
Inter Process Communication

SUBMITTED BY

Sushovan Shakya

THA075BEI046

SUBMITTED TO

Department of Electronics and Computer Engineering

26th August, 2021

INTER PROCESS COMMUNICATION

OBJECTIVE

- To understand IPC concepts.

THEORY

IPC stands for inter-process communication. Let us assume there are two processes running in the memory, a child process and a parent process. The requirement is such that the parent has to wait till a key is pressed from within the child and the parent has to exit.

pipe()

The pipe system call accept a pointer to an integer (actually an array of two integers). The integer (pfd[0]) can be used to read using the read call and the second (pfd[1]) can be used for writing using write call.

SOURCES CODES

Program 1

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int exflag = 0;

void main()
{
    char c;

    if(!fork())
    {
        printf("press a key:");
        scanf("%c", &c);
        exflag = 1;
        exit(0);
    }
    else
    {
        while(!exflag);
        printf("i got the character");
        exit(0);
    }
}
```

Output

```
kachila@pop-os:~/Desktop/os_lab/Lab 4$ ./ipc1
press a key:
```

Modified Code

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <errno.h>

int main()
{
    int fd[2];
    pipe(fd);
    char c;
    int exflag=0;

    if(!fork()) {
        close(fd[0]);
        printf("press a key: ");
        scanf("%c",&c);
        exflag=1;
        write(fd[1], &exflag, sizeof(int));
        close(fd[1]);
        exit(0);
    }
    else {
        close(fd[1]);
        int y;
        read(fd[0], &y, sizeof(int));
        close(fd[0]);
        while(!y);
        printf("i got the character\n");
    }
}
```

Output

```
kachila@pop-os:~/Desktop/os_lab/Lab 4$ ./ipc1
press a key: a
i got the character
```

RESULT

Here a character is passed between the main and the child process. This is achieved by using the pipe() function call. The child process allows the user to enter a character and the main process displays a string on receiving the character.

Program 2

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

void main()
{
    int pfd[2];

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

    if(!fork())
    {
        char data;
        printf("I'm child\n");
        printf("press any key to exit: ");
        scanf("%c", &data);
        write(pfd[1], &data, 1);
        printf("child exiting...\n\n");
    }
    else
    {
        char data;
        read(pfd[0], &data, 1);
        printf("I'm parent\n");
        printf("received %c from child\n", data);
        printf("parent exiting...\n");
        exit(0);
    }
}
```

Output

```
kachila@pop-os:~/Desktop/os_lab/Lab 4$ ./ipc1
I'm child
press any key to exit:
child exiting...

I'm parent
received
  from child
parent exiting...
```

RESULT

pipe () system call is used to communicate between processes. The child process allows the user to enter a character which is written on a file and read in the main process from the file through the use of pipe () system call. The child process terminates on getting a character from the user and the parent process terminates when it receives a character from the child process.

Program 3

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

#define MSGSZ 16

int main() {
    char *msg1 = ("hello one");
    char *msg2 = ("hello two");
    char *msg3 = ("hello three");
    char inbuf[MSGSZ];

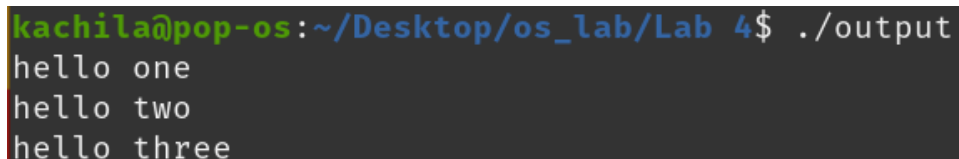
    int p[2];
    pipe(p);

    write(p[1], msg1, MSGSZ);
    write(p[1], msg2, MSGSZ);
    write(p[1], msg3, MSGSZ);

    for(int j = 0; j < 3; ++j) {
        read(p[0], inbuf, MSGSZ);
        printf("%s\n", inbuf);
    }

    exit(0);
    return 0;
}
```

Output

A terminal window with a dark background. The prompt is 'kachila@pop-os:~/Desktop/os_lab/Lab 4\$'. The command './output' has been executed, resulting in three lines of output: 'hello one', 'hello two', and 'hello three'.

```
kachila@pop-os:~/Desktop/os_lab/Lab 4$ ./output
hello one
hello two
hello three
```

RESULT

Three messages are written to a file from the child process and read from the main process and displayed. This communication is made possible by a pipe() call which connects two processes. Similar results are achieved on writing from the parent process and reading from the child process. This can be done by exchanging statements of if and else.

Program 4

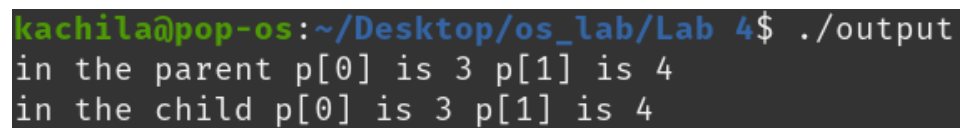
```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main() {
    int p[2], pid;
    pipe(p);
    pid = fork();

    if(pid == 0)
        printf("in the child p[0] is %d p[1] is %d\n", p[0], p[1]);
    else
        printf("in the parent p[0] is %d p[1] is %d\n", p[0], p[1]);

    return 0;
}
```

Output



```
kachila@pop-os:~/Desktop/os_lab/Lab 4$ ./output
in the parent p[0] is 3 p[1] is 4
in the child p[0] is 3 p[1] is 4
```

RESULT

The values of the file descriptors of the both processes are displayed.

Program 5

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

#define msgsz 16

int main() {
    char *msg = "hello!";
    char inbuf[msgsz];
    int p[2], pid, j;

    pipe(p);
    pid = fork();

    if(pid > 0) {
        close(p[0]);
        write(p[1], msg, msgsz);
    }

    if(pid == 0) {
        close(p[1]);
        read(p[0], inbuf, msgsz);
        printf("%s\n", inbuf);
    }

    exit(0);
    return 0;
}
```

Output

A terminal window with a dark background. The prompt is 'kachila@pop-os:~/Desktop/os_lab/Lab 4\$'. The command './output' has been entered, and the output 'hello!' is displayed on the next line.

```
kachila@pop-os:~/Desktop/os_lab/Lab 4$ ./output
hello!
```

RESULT

Message is written to a file in the main process and read in the child process and displayed. If the message is read from the file descriptor already closed then the message cannot be read and will not be displayed. Message cannot be read from or written to a file descriptor already closed.

CONCLUSION

In this section of lab, we learned about pipe () function and other parts of interprocess communication. We implemented some programs and saw examples of interprocess communications.