Objectives

- To perform message passing between related processes (parent and child) using an unnamed pipe.
- To perform message passing among a chain of processes using unnamed pipes.
- To perform message passing among fans of processes using unnamed pipes.
- To perform message passing between unrelated processes using a named pipe (Special File).

Pre-Lab Theory

1. ***pipe() synopsis:***

int pipe(int fd[])

Description:

The inter-process communication channel is created and two descriptors are allocated and stored in the 2-element integer array provided as an argument in the pipe() function.

file descriptor fd[0]:

Represents a handle for reading from the pipe. It is the index of the entry created in the process file table.

file descriptor fd[1]:

Represents a handle for writing to the pipe. It is the index of the entry created in the process file table.

Default Descriptors in the process file table with symbolic names:

Standard output: *STDOUT_FILENO*

Standard input: *STDIN_FILENO*

Standard Error: *STDERR_FILENO*

return*: Negative value if the pipe(special file) is not created in case of an error*

Named Pipe: Special File

#include <sys/stat.h>

*int mkfifo(const char *path, mode_t mode);*

Description:

The mkfifo() function shall create a new FIFO special file named by the pathname pointed to by the path. The file permission bits of the new FIFO shall be initialized from mode. The file permission bits of the mode argument shall be modified by the process file creation mask.

***Return**: 0 on successful creation of named pipe and -1 in case of an error*

Example Usage:

```
#include <sys/types.h>
#include <sys/stat.h>


int status;
...
status = mkfifo("/home/OS/Lab4/myfifo", S_IWUSR | S_IRUSR |
    S_IRGRP | S_IROTH);

fd = Open(("/home/OS/Lab4/myfifo",O_RDWR);
read(fd,….,….)
write(fd,…,….)
```

In-Lab Tasks

header file*: unistd.h, stdio.h*

System call: pipe(), fork() read(), write(), open(), close()

Task1(a)

Include the header files and the following line of code in your program. Compile and run the code and describe its behavior.

```
Int fd[2];
char buffer_p[] = "Welcome";
char buffer_c[10];
If (Pipe(fd) > 0)
     Printf("Pipe is successfully created\n");
Else
     Printf("Error");
     exit();
If ((pid = fork()) > 0)
     Printf("Child Process created\n")
     write(fd[1], buffer, sizeof(buffer);
Else if(pid == 0)
     read(fd[0], buffer_c, sizeof(buffer);
     printf("%d Received %s from Parent Process %d,getpid(),
buffer_c,getppid() );
```

Brief the working/output:

Task 1(b)

Include the header files, declare the required variables, extend the code in Task 1(a) to create a two way communication between parent and child process. The child process after receiving and displayed the "Welcome" message from the parent send the "Thanks" message to the parent. Parent displays the message received from the child process to its Standard Output.

Parent Output format:
<process_id> received Thanks from <child_process_id>
Child Output format:

<process_id> received Welcome from <parent_process_id>

Output:

Task 2(a)

Include the header files, declare the required variables, and extend the code in Task 1(a), and Task 1(b) to create a two-way message relay system between a chain of processes. Every child process after receiving and displaying the "Welcome" message from the parent replies with the "Thanks" message and forwards the "Welcome" message to the child process in the chain.

Parent Output format:
<process_id> received Thanks from <child_process_id>
Child Output format:

<process_id> received Welcome from <parent_process_id>

Output:

Task 2(b)

Include the header files, declare the required variables, and extend the code in Task 1(a), and Task 1(b) to create a two-way message relay system between a Fan of processes. Every child process after receiving and displaying the "Welcome" message from the parent replies with the "Thanks" message. The parent process displays all the messages received with the sender process ID.

Parent Output Format:

<process_id> received Thanks from <Child_process_id>

Child Output Format:

<process_id> received Welcome from <Parent_Process_Id>

Output:

Task 3

Create a two-way communication channel between two unrelated processes (Client and Server) using a named pipe (Fifo). The Client Process gets some text from standard input and sends it to the server process with its process Id. The server process displays the message received from the client and replies with a "Welcome" Message with its process id. The client displays the "Welcome" message received from the server and server process id.

Both Processes use the same named pipe for message exchange.

Client Code

Server Code

Server Side Output:

Client Side output: