Operating System Lab

Exp 8 - part 2 : Pipes

1. *Objectives:*

- Be familiar with the concept of Pipe.
- Learn the implementation of pipe, reading and writing to the pipe.

2. *Introduction:*

A pipe is used for one-way communication of a stream of bytes. In this lab, we will learn how to create pipes and how processes communicate by reading or writing to the pipe.

The following Linux systemcalls are used in the lab. You may use the man command to learn more about them.

```
ssize_t write(int fd, const void *buf, size_t count);

ssize_t read(int fd, void *buf, size_t count);

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

int pipe(int *fildes);

int system(const char* command);
```

3. *Background Process*

A background process is a computer process that runs "behind the scenes" (i.e. in the background) and without user intervention. Typical tasks for these processes include logging, system monitoring, scheduling, and user notification. to run a process in the background, from a Unix command line, a background process can be launched using the "&" operator.

4. *Named Pipes*

This type of IPC can be used between processes between which no parent-child relationship exits as shown in the example below.

- Compile the two programs, ***p1.c*** and ***p2.c*** and name the executable output as ***p1*** and ***p2*** respectively.
- Run ***p1*** in the background
- Run ***p2*** as: ./p2 hello

- The output that comes from p1 should look like:
  P1 read from the pipe: hello

The code of P1 (p1.c)

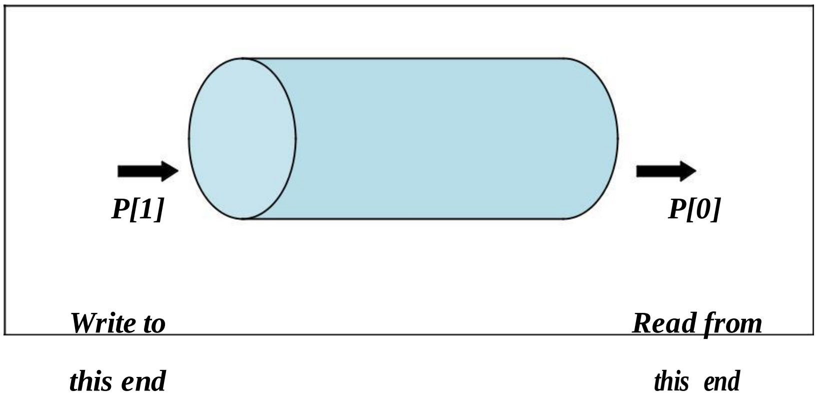
```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#define NAMED_PIPE "/tmp/namedpipe"
#define MAX_BUF_SIZE 255
void main()
{
int fd, ret_val, count, numread;
char buf[MAX_BUF_SIZE];
/* Create the named - pipe */
ret_val = mkfifo(NAMED_PIPE, 0666);
if ((ret_val == -1) && (errno != EEXIST)){
printf("Error creating the named pipe");
exit (1);
}
/* Open the pipe for reading */
fd = open(NAMED_PIPE, O_RDONLY);
/* Read from the pipe */
numread = read(fd, buf, MAX_BUF_SIZE);
buf[numread] = 0;
printf("P1 read from the pipe: %s", buf);
}
```

The code of P2 (p2.c)

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#define NAMED_PIPE "/tmp/namedpipe"
#define MAX_BUF_SIZE 255
void main(int argc, char *argv[])
{
int fd;
/* Check if an argument was specified. */
if (argc != 2) {
printf("Usage : %s <string>\n", argv[0]);
exit (1);
}
/* Open the pipe for writing */
fd = open(NAMED_PIPE, O_WRONLY);
/* Write to the pipe */
write(fd, argv[1], strlen(argv[1]));
}
```

□ Ordinary Pipes

This type of pipes requires parent-child relationship between the communicating pair of processes.



The following example shows how a parent process can send a message to its child to print it on its behalf on the standard output.

General Procedure to connect two processes with a pipe:

1. Make the pipe.
   - Fork to create the reading child.
2. In the child :
   - Close the unused end of the pipe,
   - Make the necessary preparations that are needed and execute the child program.
3. In the parent :
   - Close the unused end of the pipe.
   - o Make the necessary preparations and execute the parent program.

Run the program and observe the output.

```
/***************************************
Ordinary pipes example
File Name: Pipe.c

****************************************/
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define READ 0
#define WRITE 1
char* phrase ="This is OS lab time" ;
void main ( )
{
int fd [2],bytesread ;
char message [100] ;
pipe ( fd) ;
if ( fork ()== 0 ) /*child, writer */
{
close (fd [READ] ) ; /*close unused end */
write( fd [WRITE], phrase, strlen (phrase) + 1) ;
}
else
/*parent,reader  */
{
close ( fd [WRITE] ); /* close unused end */
bytesread = read (fd [READ], message, 100) ;

printf ("Read %d bytes : %s\n", bytesread, message) ;
}
}
```