National University of Computer and Emerging Sciences

Laboratory Manual

for

Operating Systems Lab

(CL-220)

	Mr. Aamir Rahim
Lab Instructor(s)	Zumirrah Khalid Amir Waseem
Section	4D
Semester	Spring 2021

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives

In this lab, students will practice:

1. Inter-process communication using unnamed pipes
2. Reading and writing files using open, write, and read system calls

Important Note:

- **Comment your code intelligently.**
- **Indent your code properly.**
- **Use meaningful variable names.**
- **Use meaningful prompt lines/labels for input/output.**
- **Use meaningful project and C/C++ file name**

1 Pipes

Ordinary pipes allow two processes to communicate in standard producer consumer fashion: the producer writes to one end of the pipe (the write-end) and the consumer reads from the other end (the read-end). As a result, ordinary pipes are unidirectional, allowing only one-way communication. If two-way communication is required, two pipes must be used with each pipe sending data in a different direction.

References: Operating System concepts Page no 142 section 3.6.3.1

A pipe has a read end and a write end.

Data written to the write end of a pipe can be read from the read end of the pipe.

2 Creating a pipe

On UNIX and Linux systems, ordinary pipes are constructed using the function

• int pipe(int fd[2]) -- creates a pipe

• returns two file descriptors, fd[0], fd[1].

• fd[0] is the read-end of the pipe

• fd[1] is the write-end.

- fd[0] is opened for reading,
- fd[1] for writing. pipe() returns 0 on success, -1 on failure and sets errno accordingly.
- The standard programming model is that after the pipe has been set up, two (or more) cooperative processes will be created by a fork and data will be passed using read() and write().
- Pipes opened with pipe() should be closed with
close(int fd).

Reference: http://linux.die.net/man/2/pipe

3 Example

Listing 1

```
 #include <iostream>
 #include <unistd.h>
 #include <sys/types.h>
       #include <sys/stat.h>
       #include <fcntl.h>

using namespace std;

int main()

{
char buf[5];


 int fd = open ("file.txt" , O_RDONLY, S_IRUSR);
 ssize_t size = read(fd, buf, sizeof (buf));

write(2, buf, size);


while (size > 0)
{

size = read(fd, buf, sizeof (buf));
write(2, buf, size);


}

return 0;
}
```

Listing 2

```
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1
int main(void)
{
char write_msg[BUFFER_SIZE] = "Greetings";
char read_msg[BUFFER_SIZE];
int fd[2];
pid_t pid;

/* create the pipe */
if (pipe(fd) == -1) {
fprintf (stderr,"Pipe failed");
return 1;
}
/* fork a child process */
pid = fork();
if (pid < 0) { /* error occurred */
fprintf(stderr, "Fork Failed");
return 1;
}
if (pid > 0) { /* parent process */
/* close the unused end of the pipe */
close(fd[READ_END]);
/* write to the pipe */
write(fd[WRITE_END], write_msg, strlen(write_msg)+1);
/* close the write end of the pipe */
close(fd[WRITE_END]);
}
else { /* child process */
/* close the unused end of the pipe */
close(fd[WRITE_END]);
/* read from the pipe */
read( fd[READ_END], read_msg, BUFFER_SIZE);
printf("read %s",read_msg);
/* close the write end of the pipe */
close(fd[READ_END]);
}
return 0;
}
```

4 Failure

When pipe() System Call Fails:
The pipe() system call fails for many reasons, including the following:
1 At least two slots are not empty in the FDT—too many files or pipes are open in the process.

5 Open System call for Filing

Open system call is used for opening a file.
int open(const char ****pathname*, int** *flags***, mode_t** *mode***);**

1. pathname is a file name
2. The argument *flags* must include one of the following *access modes* **O_RDONLY**, **O_WRONLY**, or **O_RDWR**. These request opening the file in read-only, write-only, or read/write modes, respectively. Apart from above, flags can also have any of the following:

(A) O_APPEND (file is opened in append mode)
(B) O_CREAT   (If *pathname* does not exist, create it as a regular file.)
(C) O_EXCL Ensure that this call creates the file: if this flag is specified in conjunction with **O_CREAT**, and *pathname* already exists, then **open**() fails.

Note: to use two flags at once use bitwise OR operator, i.e., O_WRONLY | O_CREAT

3. **Mode is only required when a new file is created and is used to set permissions on the new file**

S_IRWXU	00700	user (file owner) has read, write, and execute permission
S_IRUSR	00400	user has read permission
S_IWUSR	00200	user has write permission
S_IXUSR	00100	user has execute permission
S_IRWXG	00070	group has read, write, and execute permission
S_IRGRP	00040	group has read permission
S_IWGRP	00020	group has write permission
S_IXGRP	00010	group has execute permission
S_IRWXO	00007	others have read, write, and execute permission
S_IROTH	00004	others have read permission
S_IWOTH	00002	others have write permission
S_IXOTH	00001	others have execute permission

6 Inlab Question

Q1. Design a program using ordinary pipes in which one process sends a string message to a second process, and the second process reverses the   case of each character in the message and sends it back to the first process. For example, if the first process sends the message Hi There, the second process will return hI tHERE. This will require using two pipes, one for sending the original message from the first to the second process, and the other for sending the modified message from the second back to the first process.

Q2. Design a file-copying program named FileCopy using ordinary pipes. This program will be passed two parameters: the first is the name of the file to be copied, and the second is the name of the copied file. The program will then create an ordinary pipe and write the contents of the file to be copied to the pipe. The child process will read this file from the pipe and write it to the destination file. For example, if we invoke the program as follows: FileCopy input.txt copy.txt the file input. txt will be written to the pipe. The child process will read the contents of this file and write it to the destination fil copy. txt.

Note: Use only read, write and open system calls. Use of Cin, cout, prinf, ofstream, ifstream etc. will result in zero marks.