Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-09
Inter-Process Communication using Pipes & FIFO
Practical Programming with C (CSE 3544)

Publish on: 14-12-2024 **Submission on:** 21-12-2024
Course Outcome: CO_4 **Program Outcome:** PO_3 **Learning Level:** L_5

Problem Statement:

Experiment with inter-process communication mechanism(IPC) using pipes and FIFOs(named pipes).

Assignment Objectives:

Students will be able to differentiate independent processes and co-operative processes. They will be able to use the tools, *pipes and FIFOs*, to communicate between processes.

Answer the followings:

1. Determine the number of file descriptors (fds) will be opened for the program that becomes process. Write theirfd numbers.

```
int main(void) {
    fprintf(stderr,"ITER\n");
    while(1);
    return 0;
}
```

# of FDs and their number

Standard input : FD 0
Standard output : FD 1
Standard Error : FD 2

FD are 3 and they are 0, 1 and 2.

2. State the number of FDs will be opened for each of the process.

```
int main(void){
  fork();
  fork();
  fprintf(stderr,"hello\n");
  return 0;
}
```

# of processes and FDs for each process

Number of process = 4 but 3FD.
- Standard input (Stdin) : FD0
- Standard output (Stdout) : FD1
- Standart Error (Stderr) : FD2

3. Check the directory; **`$ls /proc/PID/fd`** to find the number of FDs the following program generates when it becomes process. Assume **`read.c`** is an existing file in your PWD.

```
int main(){
  int fd,i;
  for(i=0;i<10;i++){
     fd=open("read.c",O_RDONLY);
     if(fd==-1){
        perror("Open error");
        return 1;
     }
     sleep(2);
     printf("FD Number=%d\n",fd);
  }
   return 0;
}
```

FDs opened and output at the printf line

The loop runs 10 times, each iteration opens a new file descriptor for "read.c"

FD Number = 3
FD Number = 4
FD Number = 5
FD Number = 6
FD Number = 7
FD Number = 8
FD Number = 9
FD Number = 10
FD Number = 11
FD Number = 12

4. Check the directory; **`$ls /proc/PID/fd`** to find the number of FDs the following program generates when it becomes process. Assume **`read.c`** is an existing file in your PWD.

```
int main(){
  int fd,i;
  for(i=0;i<10;i++){
     fd=open("read.c",O_RDONLY);
     if(fd%2==0)
         printf("%d..\n",fd);
  }
   sleep(2);
   return 0;
}
```

FDs opened and output at the printf line	
FD, The Loop runs 10 times and each iteration opens a new ~~life~~ file descriptor for "read.c"	4... 6 8 10 12 14 16 18 20 20

5. Study the use of **`read()`** and **`write()`** system calls for unformatted I/O.

```
#define BLKSIZE 8
int main(void){
  char buf[BLKSIZE];
  read(STDIN_FILENO, buf, BLKSIZE);
  write(STDOUT_FILENO, buf, BLKSIZE);
  return 0;
}
```

Input to the program	Observe the corresponding output
(a) STUDENT↵ [Exactly 8 bytes including enter]	STUDENT in
(b) STUDENTS↵ [Exactly 9 bytes including enter]	STUDENTS
(c) STUDENTSpwd↵ [More than 8 byte]	STUDENTS
(d) STUDENTSpwd;who↵ [More than 8 byte]	STUDENTS.
(e) STUDENTSecho $$↵ [More than 8 byte]	STUDENTS
(f) STUDENTSBETCH↵ [More than 8 byte]	STUDENTS
(g) studentsecho "USP IS TO PRACTICE"↵ [More than 8 byte]	Steedents
(h) STUD↵ [Less than 8 byte]	STUD.

6. Write the number of file descriptors will be opened for the following code snippet. Verify the descriptor numbers by exploring the **fd** folder for the process in the directory **/proc/PID**.

```
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
int main(void)
{
  int fd[2],fs[2],fds[2];
  pipe(fd);
  pipe(fs);
  pipe(fds);
  return 0;
}
```

# of Processes& FDs with number	
Total no. of FD by the program will be 9 13 initial FDs + 6 FDs created by pipe(). pipe fd: 3(read end), 4(write end) pipe fd: 5(read end), 6(write end) pipe fd: 7(read end), 8 (write end).	Standard input: 0 Standard output: 1 Standard Error: 2

7. State the number of file descriptors will be opened for the below given code. Can you able to show the file descriptors in your machine? [Y — N]

```
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
int main(void)
{
  int fd[2],fs[2],fds[2];
  if(pipe(fd) == -1){
    perror("Failed to create the pipe");
    return 1;
}
if(pipe(fs) == -1){
   perror("Failed to create the pipe");
   return 2;
}
if(pipe(fds) == -1){
   perror("Failed to create the pipe");
   return 3;
}
return 0;
}
```

# of Processes & FDs with numbers	
total no. of the descriptors opened by the program will be 4.	
FD numbers.	
Standard input : 0	pipe fd : 3 (read end), 4 (write end)
Standard output : 1	pipe fd : 5 (read end), 6 (write end)
Standard error : 2	pipe fd : 7 (read end), 8 (write end).

8. Write the descriptor numbers attached to both parent and child process file descriptor table(PFDT). Verify the descriptor numbers by exploring the **fd** folder for the process in the directory **proc.**

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
int main(void) {
 int fd[2],fs[2],fds[2];
 pid_t pid;
 pipe(fd);
 pid=fork();
 if(pid==0){
    pipe(fs);
    pipe(fds);
 }
 else{
   wait(NULL);
   printf("Parent waits\n");
 }
  return 0;
}
```

Parent Process FDs	Child Process FDs
Parent Process FDs.	Standard input : 0
Standard input (stdin) : 0	Standard output : 1
Standard output : 1	Standard Error : 2
Standard Error : 2	fd : 3 ; 4 (write end
pipe fd : 3 (read end), 4 (write end)	fs : 5 (read end), 6 (write end).
	fds : 7 (read end), 8 (write end)

9. Write the descriptor numbers attached to both parent and child process file descriptor table(PFDT). Verify the descriptor numbers by exploring the **fd** folder for the process in the directory **proc.**

```
int main(void){int fd[2],fs[2],fds[2];
  pipe(fd);
  pid_t pid=fork();
  if(pid!=0){
     pipe(fs);
     pipe(fds);
  }
  else{
     wait(NULL);
     printf("Parent waits\n");
 }return 0;}
```

Parent Process FDs	Child Process FDs
Standard ip : 0	Standard input : 0
Standard op : 1	Standard output : 1
Standard error : 2.	Standard error : 2
fd : 3 (read), 4 (write)	pipe fd : 3 (read),
fs : 5 (read), 6 (write)	4 (write)
fds : 7 (read), 8 (write)	

10. Develop a program to write and read a message using **`pipe()`** for a single process. (***Hint:*** *No need to use* **`fork()`** *and the main process will create and implement the pipe for both writing and reading.*)

Code here | **Specify: Input & output**

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define BUFFER_SIZE 100
int main(){
   int pipefd[2];
   char buffer[BUFFER_SIZE];
   if(pipe(pipefd)==-1){
      perror("pipe");
   }  return 1;
   const char *message="Hello, this is a message";
   write(pipefd[1], message, strlen(message)+1);
   read(pipefd[0], buffer, BUFFER_SIZE);
   printf("Read from pipe : %s\n", buffer);
}  return 0;
```

output

```
Read from pipe : Hello, this is
                 a message!
```

11. Here, use the **`fork()`** system call to create a child process. The child process will write a message into the pipe and the parent process will read the message from the pipe. The parent process will display the message on *stdout*. Design a program to establish the communication using pipe between the processes.

Code here | **Specify: input & output**

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
int main(){
   int pipefd[2];
   char buffer[100];
   pipe(pipefd);
   if(fork()==0){
      const char *message="Hello from the child process!";
      write(pipefd[1], message, strlen(message)+1);
      close(pipefd[1]);
   }
   else{
      wait(NULL);
      read(pipefd[0], buffer, sizeof(buffer));
      printf("Read from pipe : %s\n", buffer);
      close(pipefd[0]);
   }
}  return 0;
```

12. Develop a program to communicate between two processes using a named pipe (FIFO). The program will demonstrate how data is written into a FIFO and how data is read from a FIFO. Implement FIFO in two aspects;

    CASE-I: Between parent process and child process (co-operative processes)

    CASE-II: Between two different process (i.e. two independent processes)

CASE I

Code here | **Specify: input & output**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FIFO_NAME "my-fifo"
#define BUFFER_SIZE 100

int main(void){
  char buffer[BUFFER_SIZE];
  pid_t pid;
  mkfifo(FIFO_NAME, 0666);
  pid = fork();
  if (pid < 0){
     perror("fork");
     exit(1);
  }else if (pid == 0){
     int fd = open(FIFO_NAME, O_WRONLY);
     const char *message = "Hello from child process!";
     write(fd, message, strlen(message)+1);
  } close(fd);
  else {
     int fd = open(FIFO_NAME, O_RDONLY);
     read(fd, buffer, sizeof(buffer));
     printf("Parent received: %s\n", buffer);
     close(fd);
     wait(NULL);
      unlink(FIFO_NAME);
  }
  return 0;
}
```

CASE II

Code here | **Specify: input & output**

writer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FIFO_NAME "my_fifo"

int main(void) {
    mkfifo(FIFO_NAME, 0666);
    int fd = open(FIFO_NAME, O_WRONLY);
    const char *message = "Hello from the independent process!\n";
    write(fd, message, strlen(message)+1);
    close(fd);
    return 0;
}
```

---

reader.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FIFO_NAME "my_fifo"
#define BUFFER_SIZE 100

int main(void) {
    char buffer[BUFFER_SIZE];
    int fd = open(FIFO_NAME, O_RDONLY);
    read(fd, buffer, sizeof(buffer));
    printf("Independent process received: %s\n", buffer);
    close(fd);
    unlink(FIFO_NAME);
    return 0;
}
```