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	

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	

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	

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	

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]	
(b) STUDENTS↵ [Exactly 9 bytes including enter]	
(c) STUDENTSpwd↵ [More than 8 byte]	
(d) STUDENTSpwd;who↵ [More than 8 byte]	
(e) STUDENTSecho $$↵ [More than 8 byte]	
(f) STUDENTSBETCH↵ [More than 8 byte]	
(g) studentsecho ”USP IS TO PRACTICE”↵ [More than 8 byte]	
(h) STUD↵ [Less than 8 byte]	

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	

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");
   eturn 3;
}
return 0;
}
```

# of Processes & FDs with numbers

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

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

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**

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**

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)

Code here	**Specify: input & output**

Code here **Specify: input & output**