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

MINOR ASSIGNMENT-007

UNIX Systems Programming (CSE 3041)

Working with UNIX pipes, named pipes(FIFO)

1. Write a C program in which the original process will create a pipe before forking a child. The parent process then writes a string to the pipe and prints a message to standard error and the child process will read the string from the pipe and then prints to standard error.

2. 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?

```
#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;
}
```

3. The above code is modified to use **`fork`**. Now 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?

```
#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;
```

```
    }
    fork();
  return 0;
}
```

4. Write a C code using **pipe** to simulate the shell pipe command **ls | sort -r**. First run on the command on the terminal and observe the output, and then create your code to meet the desired output.

5. Create C program to prepare a ring of single process that will connect the standard output of a process to its standard input through a pipe. The process will write a value of i to `stdout`, then read a value from `stdin` and store in variable j, then display value of j in `stderr`.

6. Write your C code to simulate the shell command **man ls | grep ls | wc -l** to count the number of times **ls** present in the manual page.

7. Write a C program to create a ring of two processes. Initialize the value of two variables i,j in original process. Parent process will update the value of i and pass the updated value of i to child process through pipe. Similarly child process will updtae the value of j and pass the updated value of j to parent process through pipe. Both the processes will display their corresponding value of i, j through stderr.

8. Write a C code to create a FIFO and put a string on the FIFO. Now create another C code to read the content present in the FIFO and then unlink the FIFO file.

9. Demonstarte the use the shell provided **mkfifo** command to create named pipe.

10. 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**.

```
#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;
}
```

11. 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;
}
```

12. 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;
}
```

13. How many pipe required to simulate the pipeline command **`cat demo.txt | wc`** in C.

14. Write the descriptor numbers attached to the process file descriptor table(PFDT). Verify the descriptor numbers by exploring the **fd** folder for the process in the directory **proc**. Let make a note to the size of the array **fd[6]**, does it affect the file descriptor table entry. Do more possible choice on the size of array and conclude why the size of the array **fd** is 2.

```
int main(void) {
  int fd[6],r;
  r=pipe(fd);
  if(r==-1){
      printf(Pipe create error\n");
      return 1;
  }
  while(1);
  return 0;}
```