**Inter-process Communication**

**Lab no# 10**

****

**Fall 2021**

**CSE-302 System Programming Lab**

Submitted by: **Ashfaq Ahmad**

Registration No: **19PWCSE1795**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Engr: Ma’am Madeha sheer**

**March** 6, 2022

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Task 1:**

A program in which a child writes a string to a pipe and the parent reads the string.

**Source Code:**

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

//in this program child send message and parent recieve.

int main()

{

int fd[2];

if(pipe(fd)==-1)

{

printf("Sorry! pipe can't created Successfully\n");

perror("Reason");

return -1;

}

int x=fork();

if(x==-1)

{

printf("Sorry! Child can't created Successfully\n");

perror("Reason");

return -1;

}

if(x==0)

{

char buff[100];

printf("Child: write a message for a parent..\n");

int bytesread=read(STDIN\_FILENO,buff,sizeof(buff));

//child read from STDIN file.

if(bytesread==-1)

{

printf("Sorry! read() can't executed Successfully in child case\n");

perror("Reason");

return -1;

}

int byteswritten=write(fd[1],buff,bytesread);

//and write on pipe. write alway on fd[1]

if(byteswritten==-1)

{

printf("Sorry! Data can't written on pipe successfully\n");

perror("Reason");

return -1;

}

}

else

{

wait(NULL);

char buff[200];

int bytesread=read(fd[0],buff,sizeof(buff));

//parent read from pipe. read alway from fd[0].

if(bytesread==-1)

{

printf("Sorry! Data can't read from pipe successfully\n");

perror("Reason");

return -1;

}

printf("Parent: Message Recieved\n");

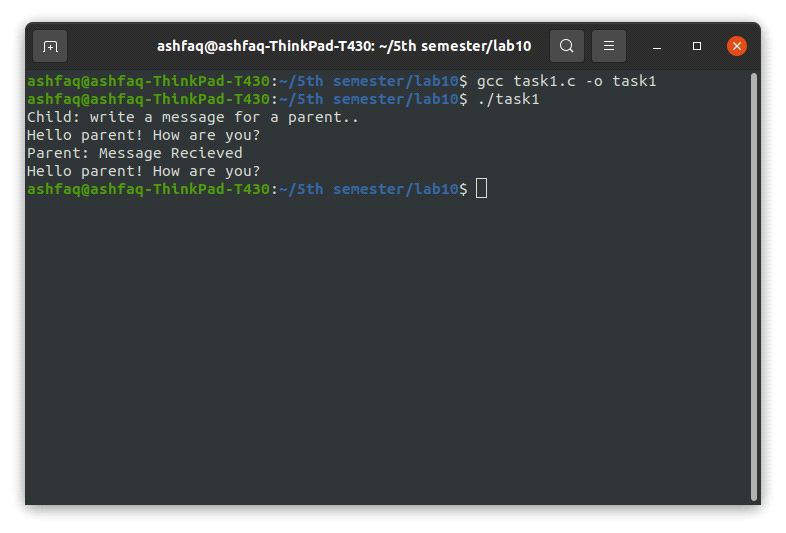
printf("%s",buff);

}

return 0;

}

**Output:**



**Task 2:**

Write a program that creates a process fan. Parent process writes to the pipe and all the child processes read the message from pipe and display it on stdout.

**Source Code:**

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char\* argv[])

{

int fd[2];

int p=pipe(fd);

if(p==-1)

{

printf("Sorry! pipe can't created Successfully\n");

perror("Reason");

return -1;

}

char buffer[500];

for(int i=0; i<5; i++)

{

int x=fork();

if(x==-1)

{

printf("Sorry! Child can't created Successfully\n");

perror("Reason");

return -1;

}

if(x>0) //parent process

{

fprintf(stderr,"Enter message for child %d: ",i+1);

int bytesread=read(STDIN\_FILENO,buffer,sizeof(buffer)); //read from keyboard

if(bytesread==-1)

{

printf("Sorry! read() can't executed Successfully in parent case\n");

perror("Reason");

return -1;

}

int byteswritten=write(fd[1],buffer,bytesread);

//parent writes to the pipe. write alway on fd[1]

if(byteswritten==-1)

{

printf("Sorry! Data can't written on pipe successfully\n");

perror("Reason");

return -1;

}

int w=wait(NULL);

if(w==-1)

{

printf("Sorry! Parent can't waited for a child to terminate successfully\n");

perror("Reason");

return -1;

}

}

else

{

int bytesread=read(fd[0],buffer,sizeof(buffer));

//child reads the pipe. read alway from fd[0]

if(bytesread==-1)

{

printf("Sorry! read() can't executed Successfully in child case\n");

perror("Reason");

return -1;

}

fprintf(stderr,"Child %d recieved message: ",i+1);

write(1,buffer,bytesread); //1=STDIN\_FILENO

//if i use printf rather than write then it print some garbige value with orignal text.

printf("\n");

break;

}

}

return 0;

}

**Output:**



**Task 3:**

**Chatting between two process using FIFO**

Write a Chatting application in which two processes can communicate using FIFO. Your program should satisfy the following specifications.

The program should take the name of FIFO, will create the FIFO (if not created yet) and should open it for reading and writing. Program should take input from standard input and write it to FIFO and should read from FIFO and write to standard output in another process. Both reading and writing shall be done concurrently.

**Source Code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/select.h>

#include <fcntl.h> //for mode i,e O\_RDONLY etc.

#include <sys/stat.h> //for fifo function

#include <errno.h>

int main(int argc, char\* argv[])

{

if(argc!=2)

{

printf("fifo name is not provided\n");

return -1;

}

int x=mkfifo(argv[1],S\_IRWXU); //makes named pipe..

if(x==-1 && errno!=EEXIST) //return -1 if mkfifo failed but not in case of EEXIST.

{

printf("Filed to create fifo\n");

perror("Reason");

return -1;

}

int fd=open(argv[1],O\_RDWR); //open pipe

if(fd==-1)

{

printf("Filed to open fifo\n");

perror("Reason");

return -1;

}

char buffer[1000];

fd\_set readset;

while(1)

{

FD\_ZERO(&readset);

FD\_SET(fd,&readset);

FD\_SET(STDIN\_FILENO,&readset);

int s=select(fd+1,&readset,NULL,NULL,NULL);

//will check which terminal is ready for reading and which for writing.

if(s==-1)

{

printf("Sorry an Error occured with select function\n");

perror("Reason");

return -1;

}

if(FD\_ISSET(STDIN\_FILENO,&readset))

{

int bytesread=read(STDIN\_FILENO,buffer,sizeof(buffer));

if(bytesread==-1)

{

printf("Sorry! data can't read from stdin\n");

printf("Reason");

return -1;

}

int byteswritten=write(fd,buffer,bytesread);

if(byteswritten==-1)

{

printf("Sorry! data can't written on fifo\n");

printf("Reason");

return -1;

}

sleep(1); //sleep that terminal which write on fifo.

//if it doesn't go to sleep then same termial may read fifo also.

//i,e from same terminal text will read and displayed..

}

if(FD\_ISSET(fd,&readset))

{

int bytesread=read(fd,buffer,sizeof(buffer));

if(bytesread==-1)

{

printf("Sorry! data can't read from fifo\n");

printf("Reason");

return -1;

}

fprintf(stderr,"Recieved: "); //alway print to file rather than stdout.

if(write(STDIN\_FILENO,buffer,bytesread)==-1)

//here i write buffer rather than print it.

//if i print buffer. it will print some garbige value along with original text.

//because printf function print all contents in a buffer upto \0 character.

//rather than orignal text only.

{

printf("Sorry! data can't written on Monitor\n");

printf("Reason");

return -1;

}

}

}

}

**Output:**

