Fatima Jinnah Women University

Opening Portals of Excellence Through Higher Education

Rabia batool

2022-BSE-064

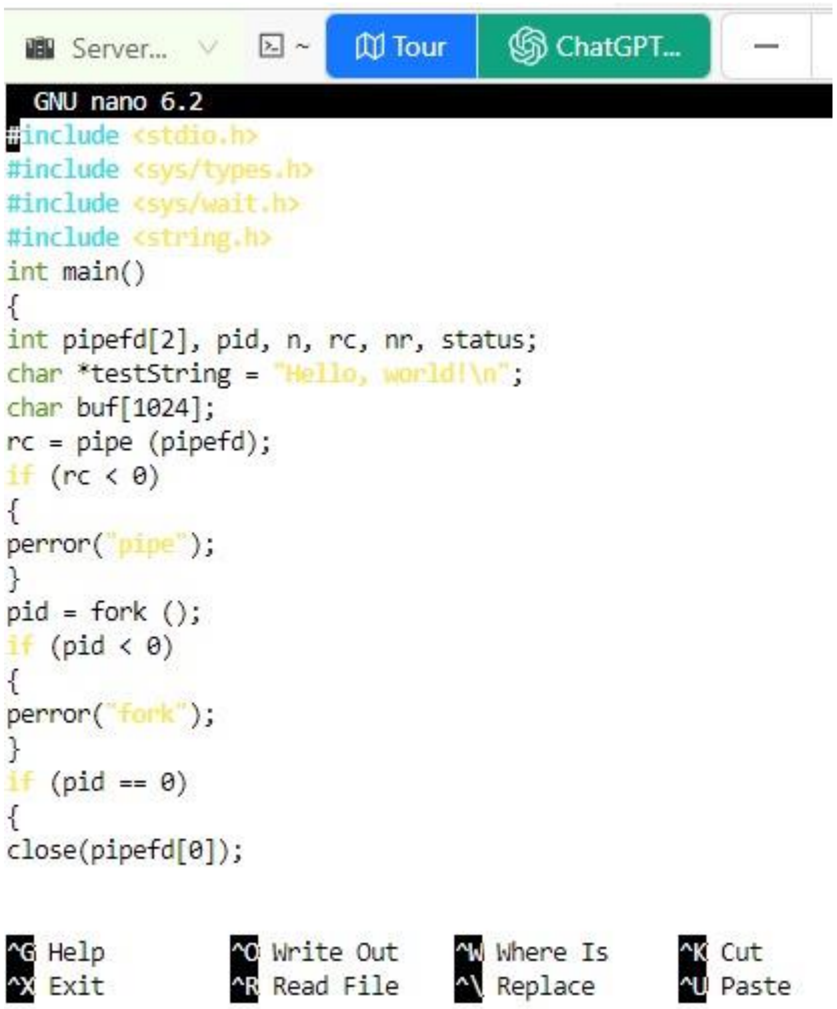
Group#B

Operating System

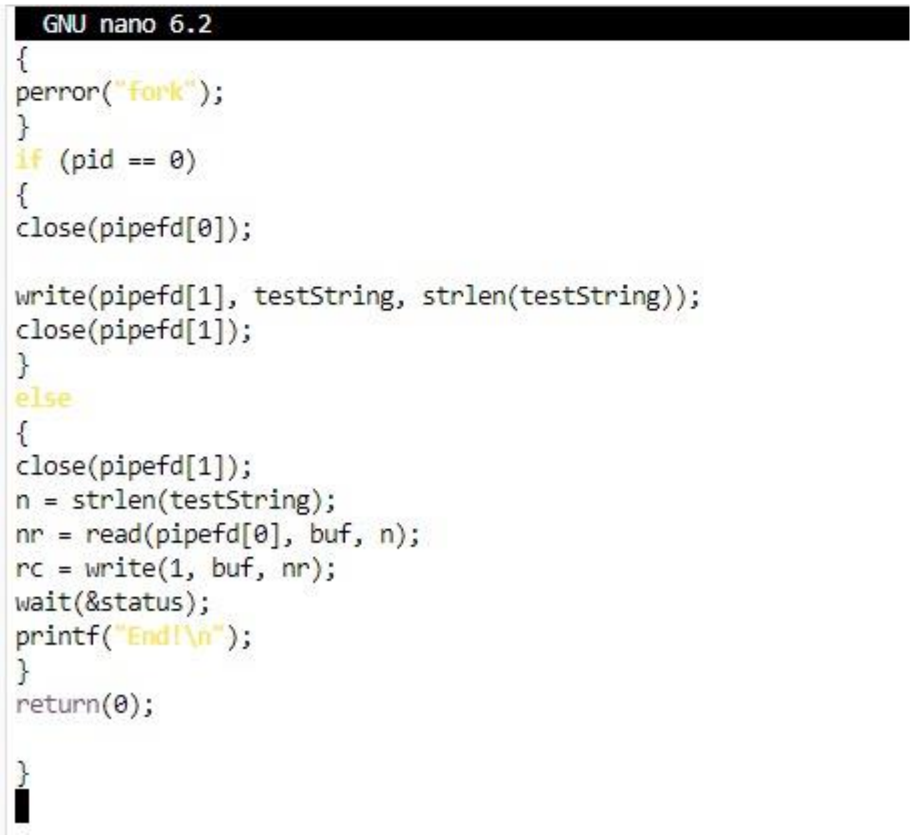
Lab#08

Submitted to Sir Shehzad

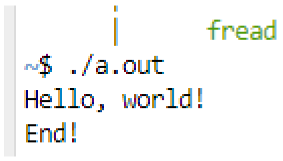
Practice 1



```
GNU nano 6.2
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
int main()
{
int pipefd[2], pid, n, rc, nr, status;
char *testString = "Hello, world!\n";
char buf[1024];
rc = pipe (pipefd);
if (rc < 0)
{
perror("pipe");
}
pid = fork ();
if (pid < 0)
{
perror("fork");
}
if (pid == 0)
{
close(pipefd[0]);


^G Help        ^O Write Out   ^W Where Is    ^K Cut
^X Exit        ^R Read File   ^\ Replace     ^U Paste
```

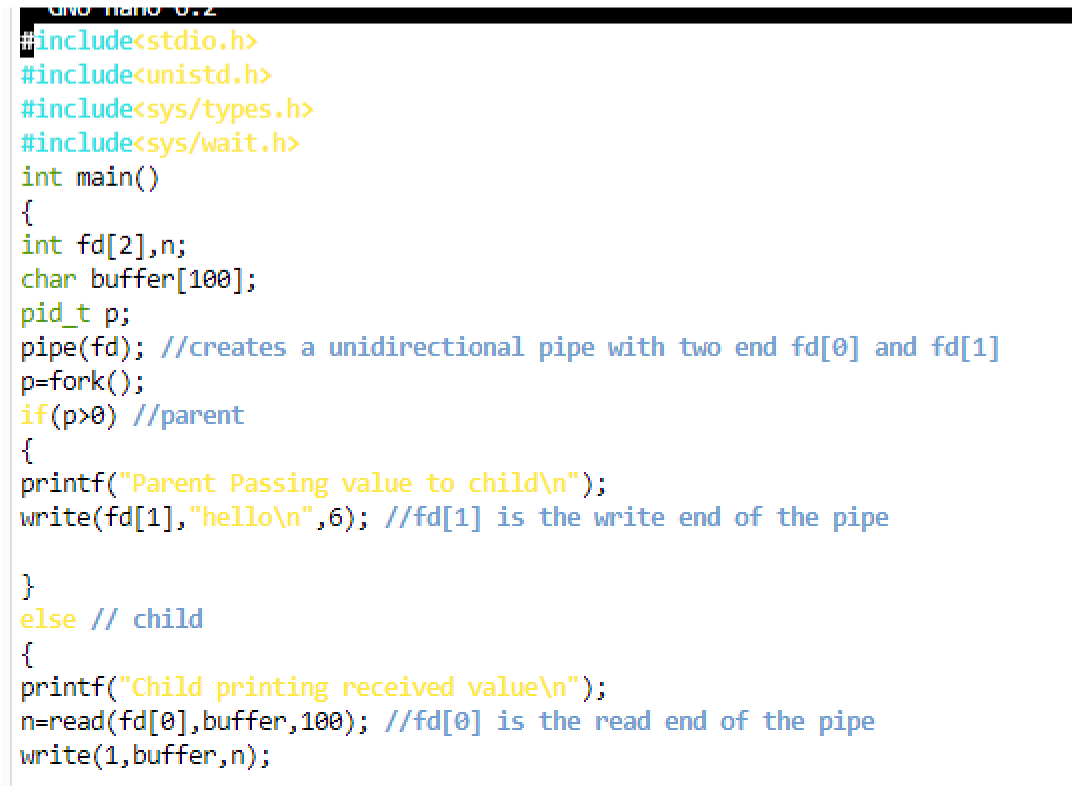


```
GNU nano 6.2
{
perror("fork");
}
if (pid == 0)
{
close(pipefd[0]);

write(pipefd[1], testString, strlen(testString));
close(pipefd[1]);
}
else
{
close(pipefd[1]);
n = strlen(testString);
nr = read(pipefd[0], buf, n);
rc = write(1, buf, nr);
wait(&status);
printf("End!\n");
}
return(0);

}
```

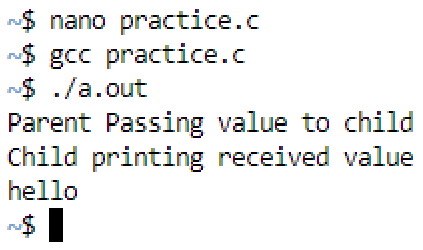


```
        |      fread
~$ ./a.out
Hello, world!
End!
```

Practice 2



```
  GNU nano 6.2
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
int fd[2],n;
char buffer[100];
pid_t p;
pipe(fd); //creates a unidirectional pipe with two end fd[0] and fd[1]
p=fork();
if(p>0) //parent
{
printf("Parent Passing value to child\n");
write(fd[1],"hello\n",6); //fd[1] is the write end of the pipe

}
else // child
{
printf("Child printing received value\n");
n=read(fd[0],buffer,100); //fd[0] is the read end of the pipe
write(1,buffer,n);
```



```
~$ nano practice.c
~$ gcc practice.c
~$ ./a.out
Parent Passing value to child
Child printing received value
hello
~$
```

Viva Questions on Program for IPC using pipe() function

Q1. Which kind of data channel is created by pipe() system call: Unidirectional or bidirectional?

ANS:Unidirectional.

Q2. What does the pipe() system call return on success?

ANS: The pipe() system call returns 0 (zero) on success. This indicates that the pipe was created successfully and the two file descriptors are ready to be used for reading and writing.

Q3. What does the pipe() system call return on failure?

ANS: the pipe() system call returns -1 (negative one) on failure. This indicates that the pipe creation was unsuccessful, and you cannot proceed with using it for data transfer.

Q4. Why fork() is used in the above program?

ANS: To create child process.

Q5. Which process (parent or child) in the above code, is using the writing end of the pipe?

ANS: The parent process is using the writing end of the pipe.

TASK 1

o Compute the Factorial of a number using IPC (PIPE implementation).

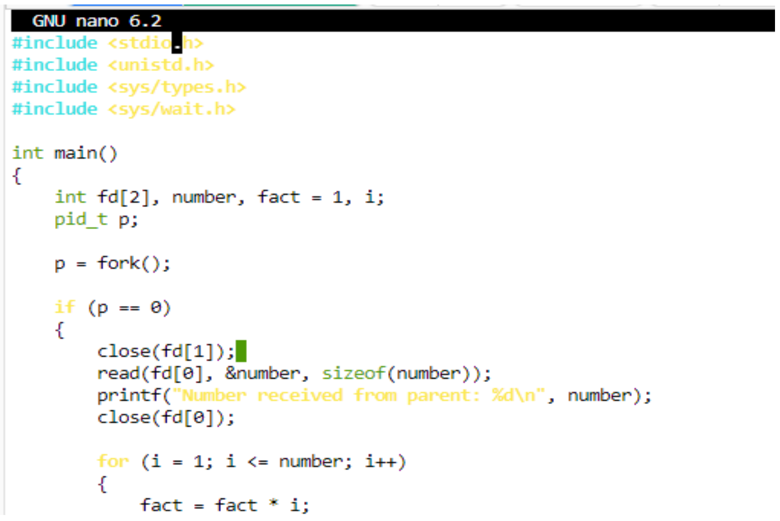
o Parent creates pipe

o Forks a child

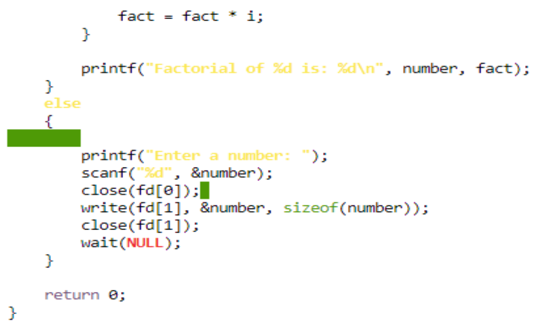
o Parent writes into pipe (the number whose factorial is to be calculated, take the number

from the user)

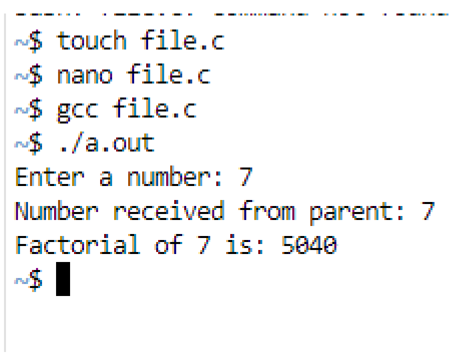
o Child reads from pipe and compute the Factorial of a number written by Parent



```
  GNU nano 6.2
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    int fd[2], number, fact = 1, i;
    pid_t p;

    p = fork();

    if (p == 0)
    {
        close(fd[1]);
        read(fd[0], &number, sizeof(number));
        printf("Number received from parent: %d\n", number);
        close(fd[0]);

        for (i = 1; i <= number; i++)
        {
            fact = fact * i;
```



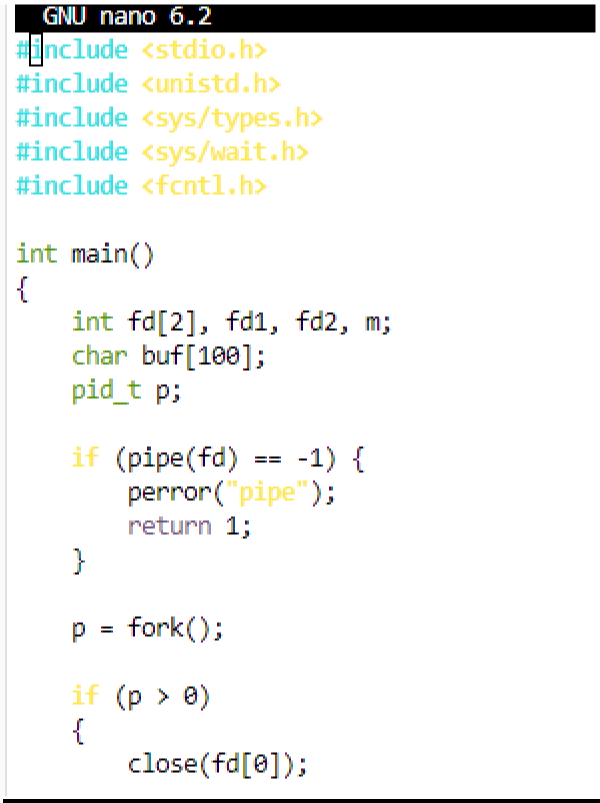
```c
            fact = fact * i;
        }

        printf("Factorial of %d is: %d\n", number, fact);
    }
    else
    {

        printf("Enter a number: ");
        scanf("%d", &number);
        close(fd[0]);
        write(fd[1], &number, sizeof(number));
        close(fd[1]);
        wait(NULL);
    }

    return 0;
}
```



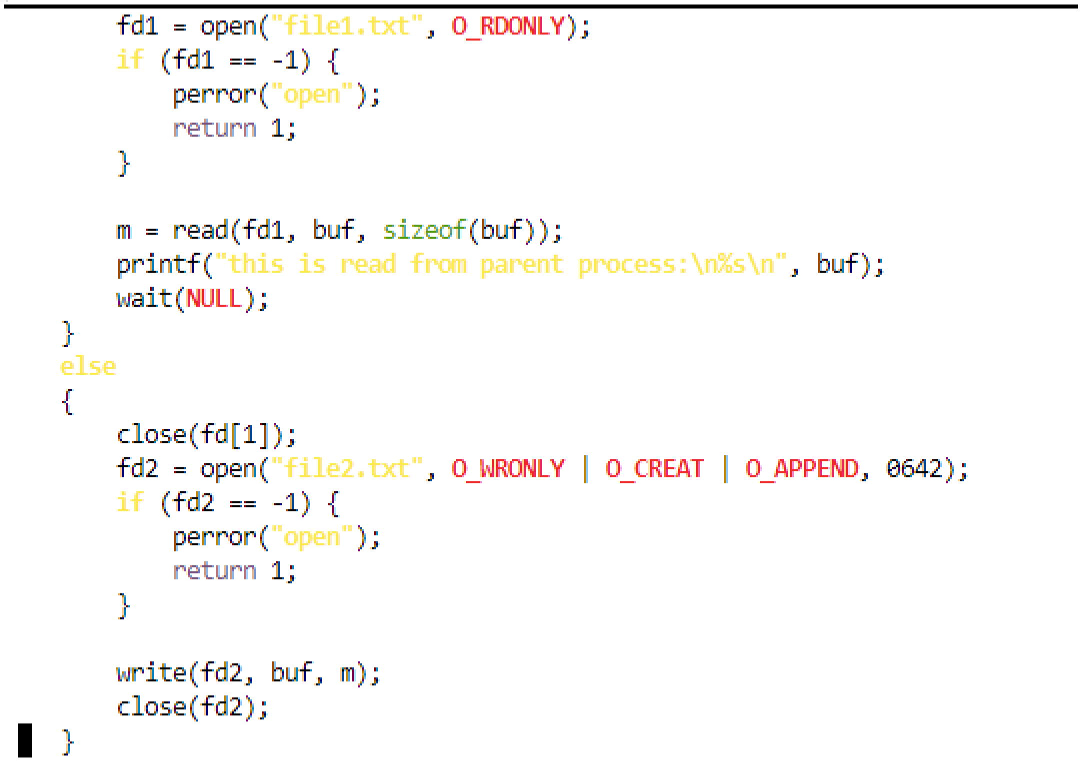
```
~$ touch file.c
~$ nano file.c
~$ gcc file.c
~$ ./a.out
Enter a number: 7
Number received from parent: 7
Factorial of 7 is: 5040
~$
```

TASK 2

Using pipes, parent read data from one file, and child write data into another file.

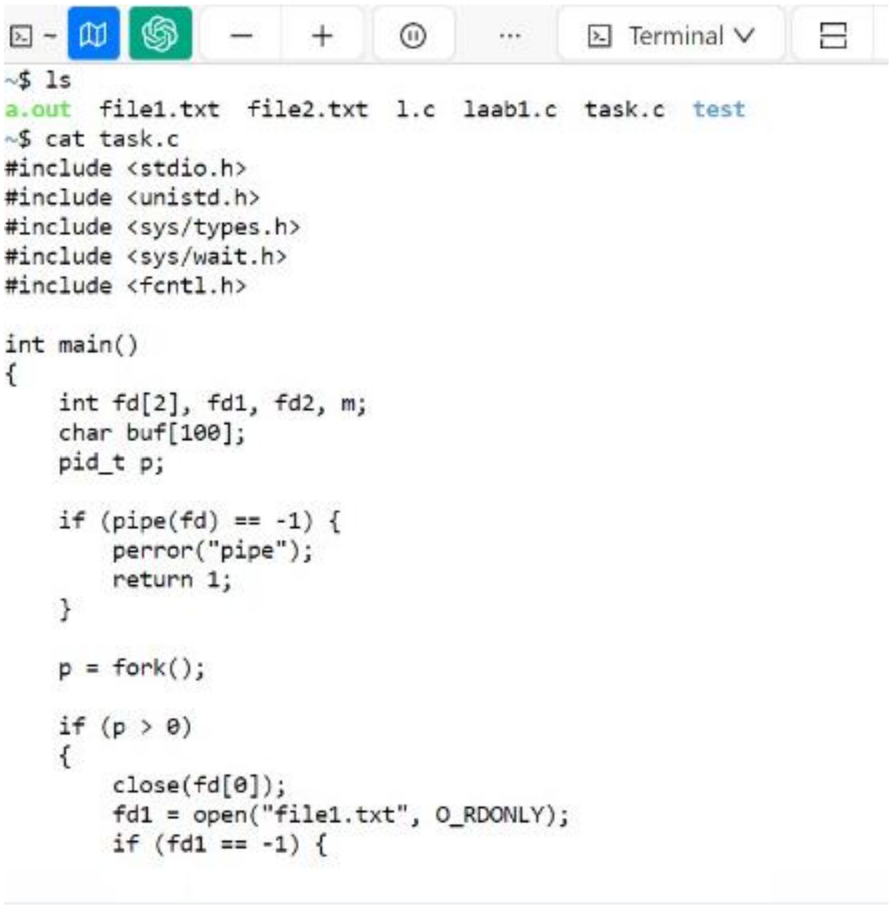


```
  GNU nano 6.2
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main()
{
    int fd[2], fd1, fd2, m;
    char buf[100];
    pid_t p;

    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }

    p = fork();

    if (p > 0)
    {
        close(fd[0]);
```

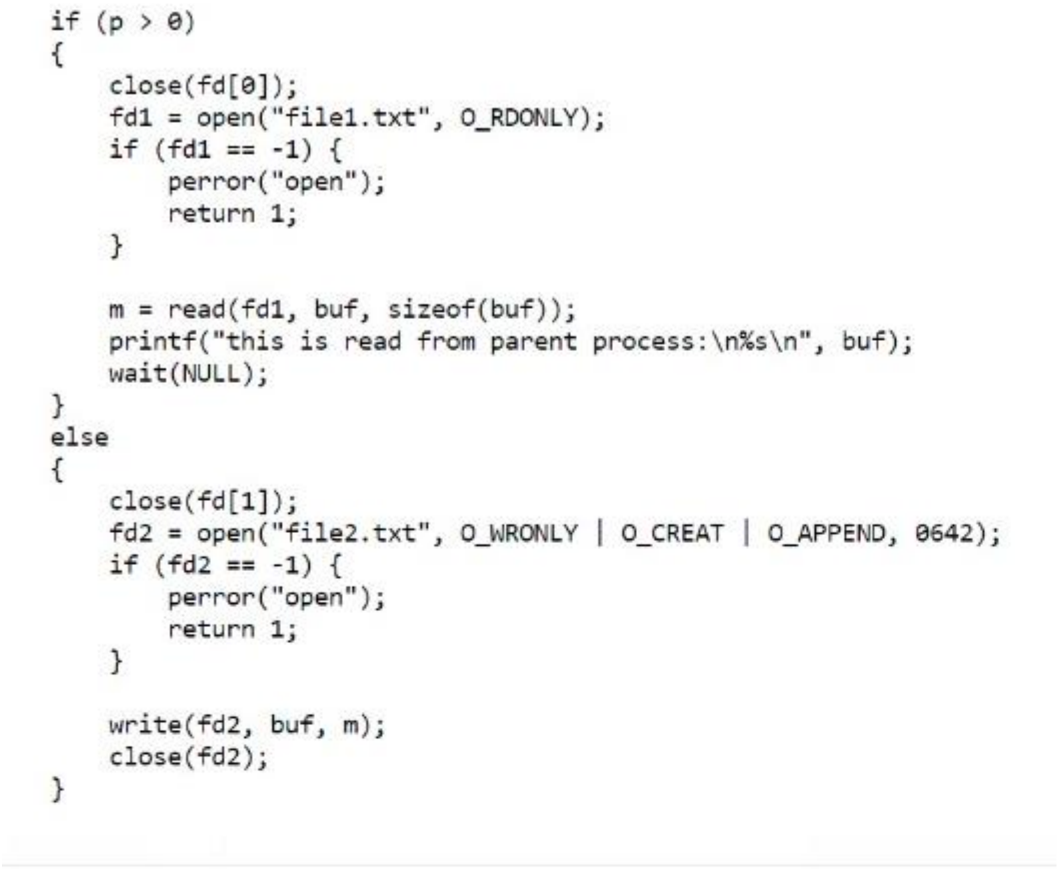


```
        fd1 = open("file1.txt", O_RDONLY);
        if (fd1 == -1) {
            perror("open");
            return 1;
        }

        m = read(fd1, buf, sizeof(buf));
        printf("this is read from parent process:\n%s\n", buf);
        wait(NULL);
    }
    else
    {
        close(fd[1]);
        fd2 = open("file2.txt", O_WRONLY | O_CREAT | O_APPEND, 0642);
        if (fd2 == -1) {
            perror("open");
            return 1;
        }

        write(fd2, buf, m);
        close(fd2);
    }
```

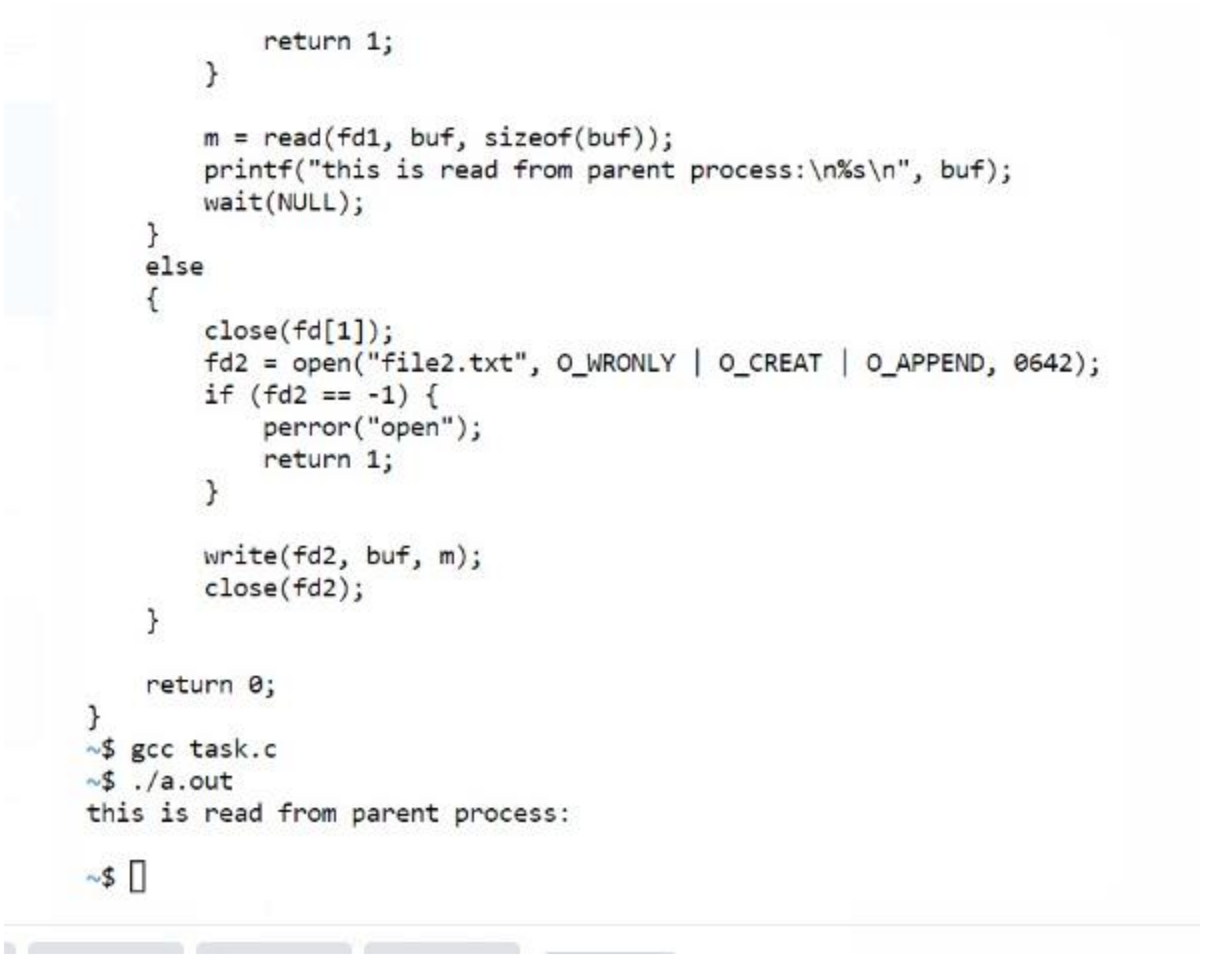
Output:



```
~$ ls
a.out  file1.txt  file2.txt  l.c  laab1.c  task.c  test
~$ cat task.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main()
{
    int fd[2], fd1, fd2, m;
    char buf[100];
    pid_t p;

    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }

    p = fork();

    if (p > 0)
    {
        close(fd[0]);
        fd1 = open("file1.txt", O_RDONLY);
        if (fd1 == -1) {
```



```
if (p > 0)
{
    close(fd[0]);
    fd1 = open("file1.txt", O_RDONLY);
    if (fd1 == -1) {
        perror("open");
        return 1;
    }

    m = read(fd1, buf, sizeof(buf));
    printf("this is read from parent process:\n%s\n", buf);
    wait(NULL);
}
else
{
    close(fd[1]);
    fd2 = open("file2.txt", O_WRONLY | O_CREAT | O_APPEND, 0642);
    if (fd2 == -1) {
        perror("open");
        return 1;
    }

    write(fd2, buf, m);
    close(fd2);
}
```



```
            return 1;
        }

        m = read(fd1, buf, sizeof(buf));
        printf("this is read from parent process:\n%s\n", buf);
        wait(NULL);
    }
    else
    {
        close(fd[1]);
        fd2 = open("file2.txt", O_WRONLY | O_CREAT | O_APPEND, 0642);
        if (fd2 == -1) {
            perror("open");
            return 1;
        }

        write(fd2, buf, m);
        close(fd2);
    }

    return 0;
}
~$ gcc task.c
~$ ./a.out
this is read from parent process:

~$ 
```