ASSIGNMENT

CSE316

Name: P.Harshavardhan

Registration No: 11811680

Roll No: 14

Section: K18AW

Email: harshavardhanpotula@gmail.com

Solution No: 1

Code:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        int Const=0;
        pid_t pid;
       do
          {
             printf("Enter a Number Greater than 0 \n");
                         scanf("%d", &Const);
```

```
 }while (Const <= 0);
pid = fork();
if (pid == 0)
 {
   printf("Child is Running...\n");
    printf("%d\n",Const);
          while (Const!=1)
          {
                   if (Const%2 == 0)
                   {
                            Const = Const/2;
                   }
                   else if (Const%2 == 1)
                   {
                            Const = 3 * (Const) + 1;
                   }
                printf("%d\n",Const);
          }
      printf("Child process Completed\Num");
 }
 else
 {
```

```
                        printf("Parent is waiting for child process to
complete...\n");
                        wait();
                        printf("Parent process is Completed.\n");
                }
        return 0;
}
```

Explanation in detail:

The child process can be generated from parent main process by using this fork() function and so, this can be called as system call function. Both child and parent processes have the copies of the data in their own method but for a child process it is mandatory that the output should be in sequence. The wait() function call is invoked by the parent process to wait for its child process to complete before the program is exited. In the process line is any non positive integer is included then the error checking starts to find whether the entered integer is valid or invalid.

Algorithm:

Step1: start declare(Const=0 and pid)# declare integer Const and also declare pid as a variable for data type pid which is used to represent process IDs.

Step2: do take the value of the Const while Const<=0.

Step3: Create a child process for the parent process with variable pid.

Step 4: if(pid == 0) then print Child process is running and also print Const value.

```
Step 5: repeat step 6 , 7 and 8 while ( Const!=1)
Step 6: if( Const%2 == 0)
            {
               Const= Const/2
             }
Step7: else if(Const%2==1)
                 {
                   Const= 3*(Const)+1
                 }
```

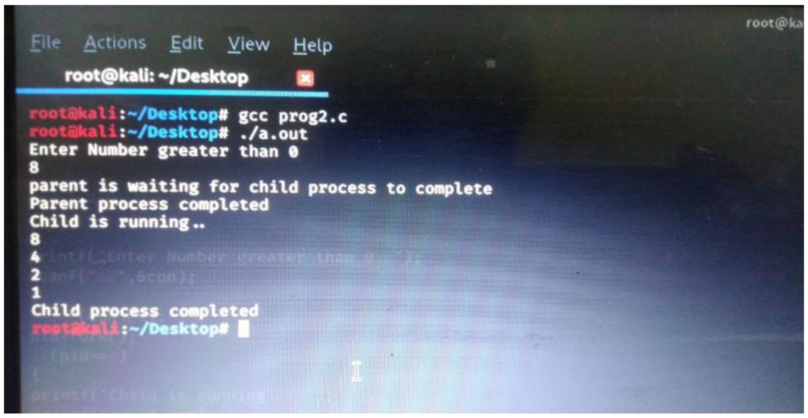
Step8: print the value of Const and also print that Child process Completed.

Step9: If step 4 is not satisfied then print that parent is waiting for child process to complete, then wait for parent process to complete and then print parent process is completed.

Step10: Exit

Test Case:

Output:



```
root@kali:~/Desktop# gcc prog2.c
root@kali:~/Desktop# ./a.out
Enter Number greater than 0
8
parent is waiting for child process to complete
Parent process completed
Child is running..
8
4
2
1
Child process completed
root@kali:~/Desktop#
```

Solution No: 26

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
    int FileDesc[2];
         int fileSize;
         pid_t ProcessId;
    char text[1024];
         char copy[1024];
    char* SourceFile = argv[1];
    char* DestFile = argv[2];
    pipe(FileDesc);
    if (pipe(FileDesc)==-1)
    {
        fprintf(stderr,"Pipe Is Failed" );
        return 1;
    }
```

```
ProcessId = fork();
 if (ProcessId < 0)
{
    fprintf(stderr, "Fork Is Failed" );
    return 1;
}
if (ProcessId > 0)
{
        int sourceFileNum;
        ssize_t numBytes;
    close(FileDesc[0]);
    sourceFileNum=open(SourceFile, O_RDONLY);
    numBytes=read(sourceFileNum, text,sizeof(text));
    write(FileDesc[1],text, numBytes);
    close(FileDesc[1]);
}else if (ProcessId == 0){
     int destDesc;
    close(FileDesc[1]);
    ssize_t BytesNumber;
    BytesNumber =read(FileDesc[0], copy,sizeof(copy));
    close(FileDesc[0]);
    destDesc=open(DestFile, O_CREAT | O_WRONLY);
    write(destDesc, copy, BytesNumber);
    }
    return 0;
}
```

Explanation in detail:

The communication between related processes can be done using pipes. Also pipes used for unrelated process communication like if we want to execute the client program from one terminal. One-way communication is done using one pipe and similarly for bi-directional communication two pipes are used. But the same can’t be applied for Named Pipes as single named pipe can be used in two-way communication. So, we can say bi-directional communication is possible by using single named pipe in Named Pipes concept.

Algorithm:

```
Step1: Start.
Step2: Declare file descriptors,and size of each file and processids.
Step3: Declare source file and  destination files.
Step4: pipe(FileDesc).
Step5: if pipe of filedescriptor is not created then print pipe is failed.
Step6: create a Chile process for the parent process and if it not created print "Fork creation failed".
Step7: else if Declare source file Num and numBytes.
Step8: close the first file.
Step9: open source file for reading it , then write the text to the second file and then close the second file also.
```

Step10: else if(PROCESSID ==0) declare destination file discripter and close the second file.

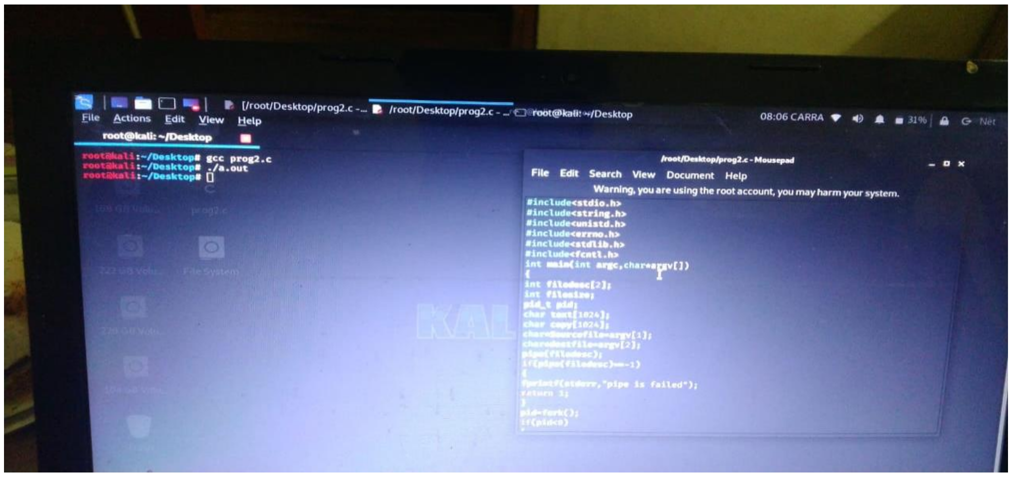
Step11: read text from the first file and close the first file

Step12: write the text to the destination file

Step13: Exit.

Test Case:

Output:


File Actions Edit View Help
[/root/Desktop/prog2.c -...
/root/Desktop/prog2.c - ...
root@kali: ~/Desktop
08:06 CARRA
31%
/root/Desktop/prog2.c - Mousepad
File Edit Search View Document Help

```
root@kali:~/Desktop# gcc prog2.c
root@kali:~/Desktop# ./a.out
root@kali:~/Desktop# 
```

Warning, you are using the root account, you may harm your system.

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<errno.h>
#include<stdlib.h>
#include<fcntl.h>
int main(int argc,char*argv[])
{
int filedesc[2];
int filesize;
pid_t pid;
char text[1024];
char copy[1024];
char*sourcefile=argv[1];
char*destfile=argv[2];
pipe(filedesc);
if(pipe(filedesc)==-1)
{
fprintf(stderr,"pipe is failed");
return 1;
}
pid=fork();
if(pid<0)
```