**CSE2005-Operating Systems Lab**

**Assessment-1 Questions**

1. **Basic Linux Commands**
2. **Shell Programming** 
   1. **Handling the command line arguments**
   2. **String reversal**
   3. **If-Else, Nested If Else, Switch cases in shell**
3. **Parent child process creation using fork( ) and exec() system call**

**A) Checking the Process Identifier**

**B) Assigning new task to child**

**C) Providing the path name and program name to exec()**

**D) Synchronizing Parent and child process using wait()**

1. **The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:**

**n = n/2, if n is even n = 3 × n + 1, if n is odd**

**The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1.**

**For example, if n = 35, the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1.**

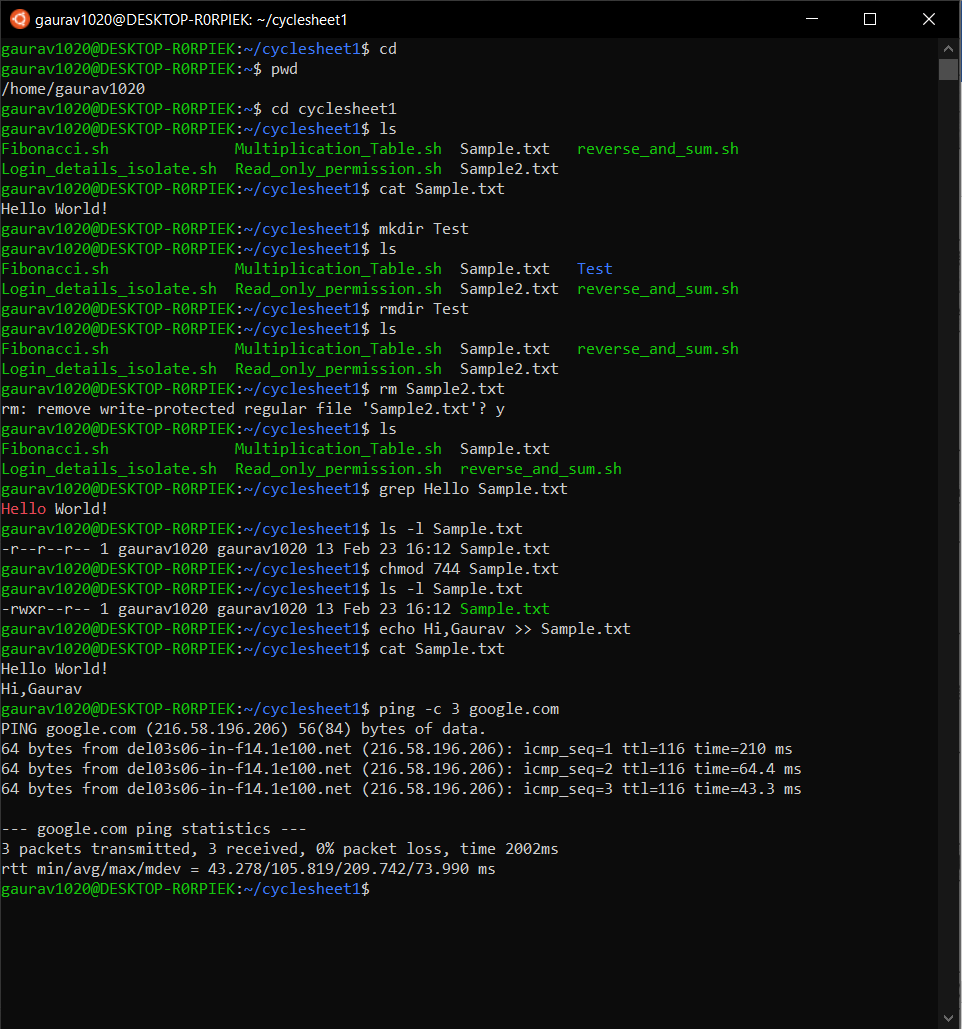
**Write a C program using the fork() system call that generates this sequence in the child process. The starting number will be provided from the command line.**

**For example, if 8 is passed as a parameter on the Command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence.**

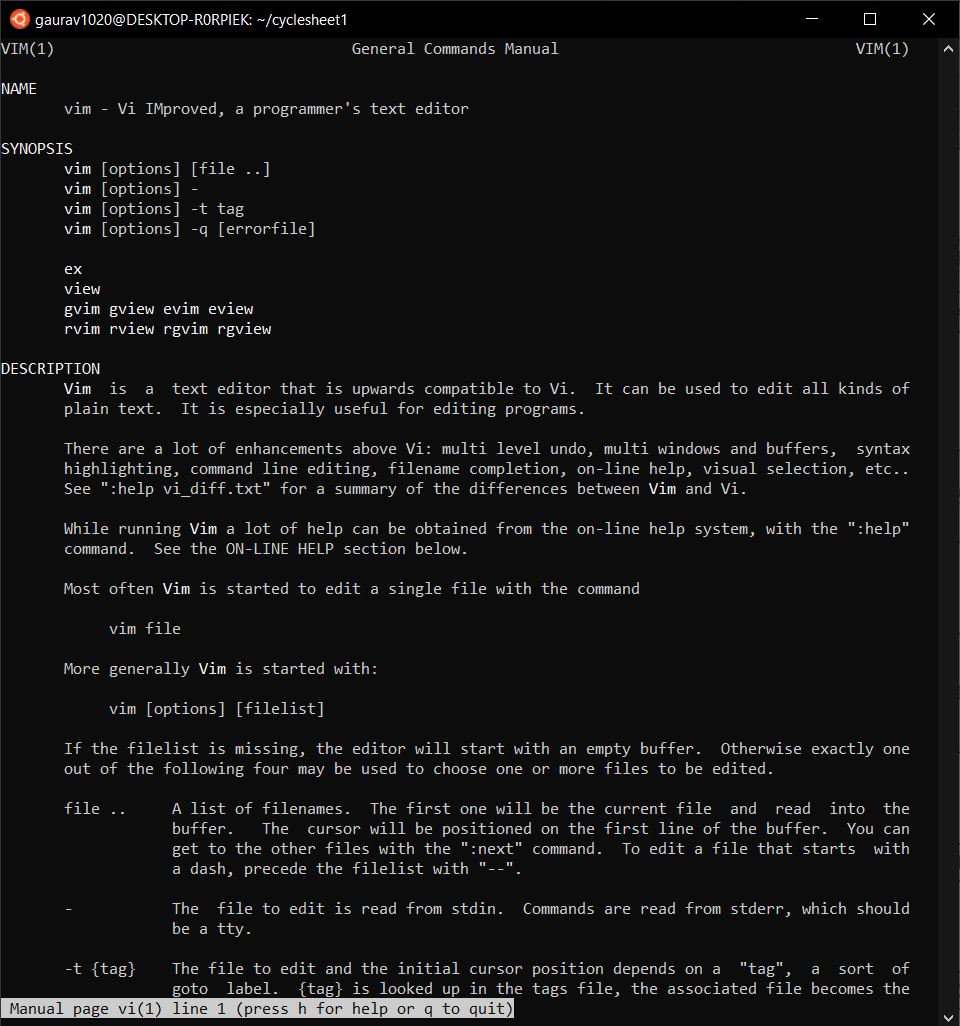
**Have the parent invoke the wait() call to wait for the child process to complete before exiting the program (High).**

**Answer 1) Basic Linux Commands**

|  |  |
| --- | --- |
| **Command** | **Function** |
| pwd | pwd command is used to find out the path of the current working directory (folder). The command will return an absolute (full) path, which is basically a path of all the directories that starts with a forward slash (/) |
| cd | cd is used to navigate through the Linux files and directories.  cd (directoryName) to move down the immediate directory  cd ..  to move one directory up  cd to go straight to the home folder  cd- to move to your previous directory |
| ls | The ls command is used to view the contents of a directory. By default, this command will display the contents of your current working directory.  ls -l will list all the files and directory in the directory along with their modification permissions.  ls -R will list all the files in the sub-directories as well.  ls -a will show the hidden files.  ls -al will list the files and directories with detailed information like the permissions, size, owner, etc. |
| cat | It is used to list the contents of a file on the standard output (sdout). To run this command, type cat followed by the file’s name and its extension. For instance: cat file.txt. |
| vi | vi command is to create new files and open and edit existing files. |
| mkdir | mkdir command to make a new directory |
| rmdir | rmdir command is used to delete a directory. However, rmdir only allows you to delete empty directories. |
| grep | grep lets us search through all the text in a given file. |
| sudo | Short for “SuperUser Do”, this command enables you to perform tasks that require administrative or root permissions. |
| chmod | chmod is used to change the read, write, and execute permissions of files and directories. |
| ping | ping command is used to check our connectivity status to a server. |
| man | Shows a documentation on another linux commands for us to learn how to use those commands |
| echo | This command is used to move some data into a file. |
| rm | The rm command is used to delete directories and the contents within them. If you only want to delete the directory — as an alternative to rmdir — use rm -r |



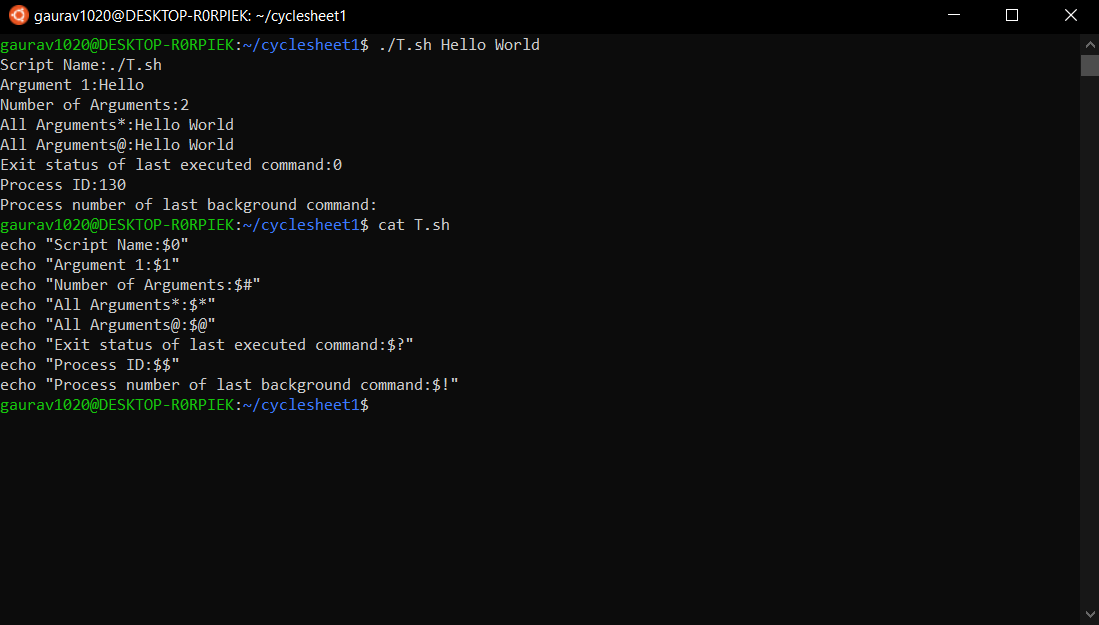




**Answer 2) Shell Programming**

1. **Handling Command Line Arguments**

|  |  |  |
| --- | --- | --- |
| S. No. | Parameter | Description |
| 1 | $0 | Returns filename of the script |
| 2 | $n | n is positive integer. Returns the nth argument given to the script when the script was invoked |
| 3 | $# | Returns the number of arguments given to the script when it was invoked |
| 4 | $\* | Returns all arguments given to the script when it was invoked |
| 5 | $@ | Returns all arguments given to the script when it was invoked |
| 6 | $? | Returns the exit status of last command executed |
| 7 | $$ | Returns process number of the current shell i.e. process id under which the script is executing |
| 8 | $! | The process number of last background command |



1. **String Reversal**

**Code:**

String="$1"

len=${#String}

for ((i = $len-1; i >= 0; i--))

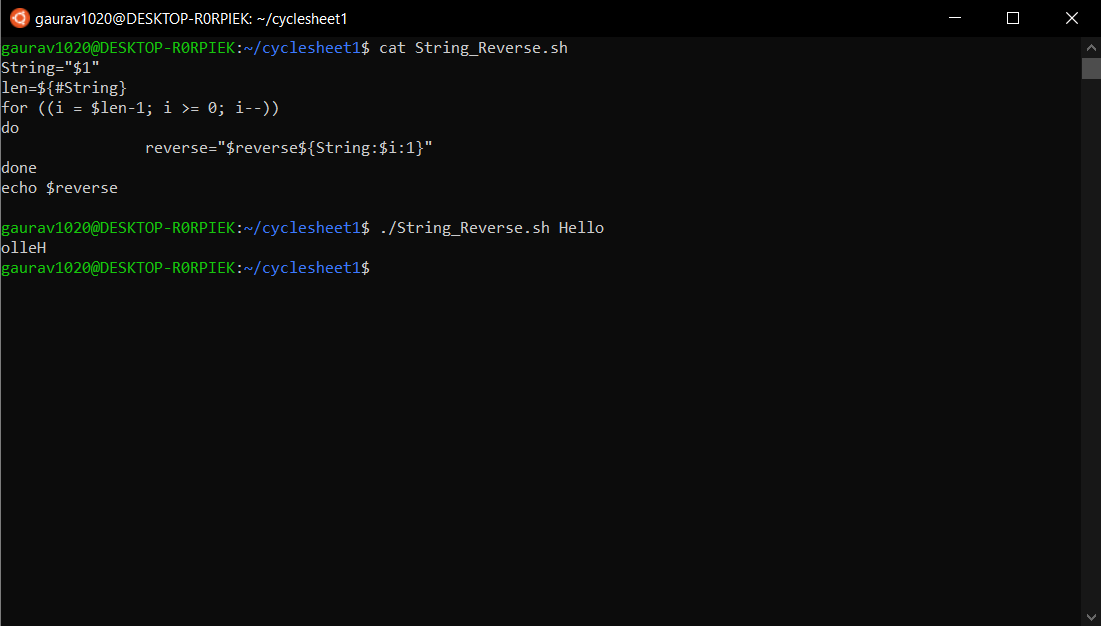
do

reverse="$reverse${String:$i:1}"

done

echo $reverse

**Output:**



1. **If-Else, Nested If Else, Switch cases in shell**

**Code:**

echo "IF ELSE STATEMENT"

if (( $1 == $2 ))

then

echo "$1 is Equal to $2"

else

echo "$1 is NOT Equal to $2"

fi

echo "NESTED IF ELSE STATEMENT"

if (( $1 >= 0 ))

then

echo "$1 is Non Negative Integer"

if (( $1 == 0 ))

then

echo "$1 is Equal to Zero"

else

echo "$1 is NOT Equal to Zero"

fi

else

echo "$1 is Negative Integer"

fi

echo "SWITCH CASE STATEMENT"

case $3 in

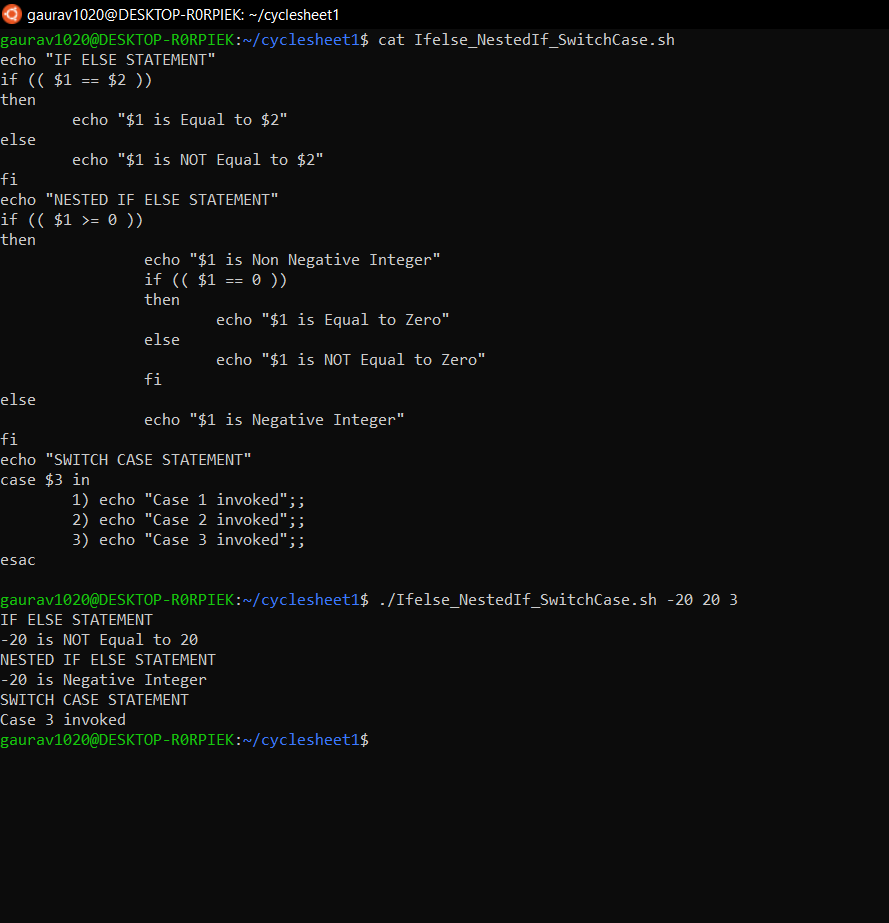
1) echo "Case 1 invoked";;

2) echo "Case 2 invoked";;

3) echo "Case 3 invoked";;

esac

**Output:**



**Answer 3)**

**A) Checking the Process Identifier**

**Code:**

#include <stdio.h>

#include <sys/types.h>

void main(void)

{

pid\_t pid;

pid = getpid();

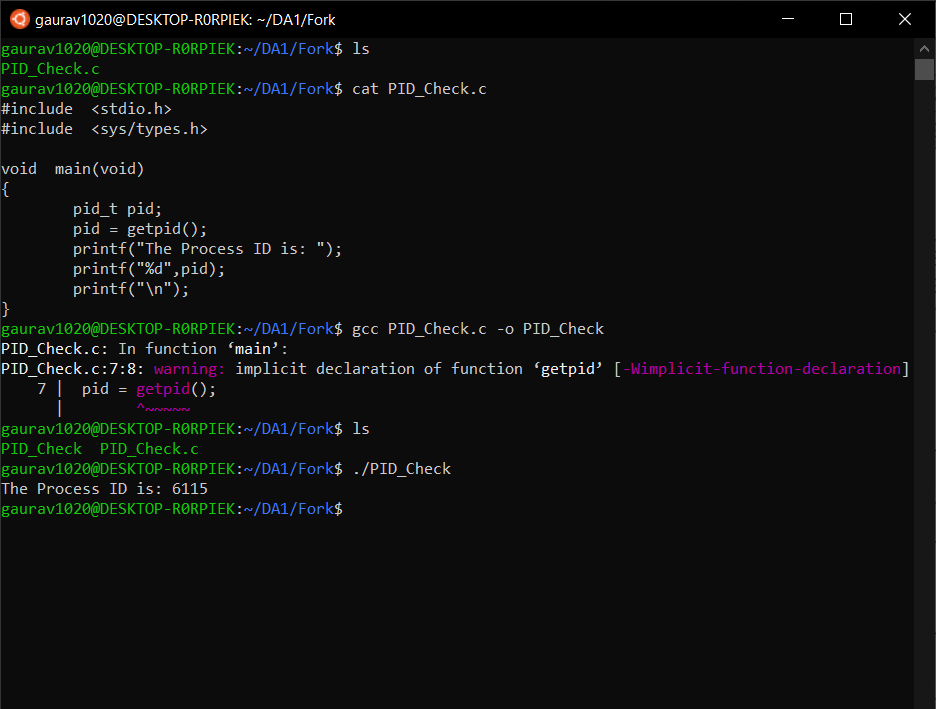
printf("The Process ID is: ");

printf("%d",pid);

printf("\n");

}

**Output:**



**B) Assigning new task to child**

**Code:**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main() {

pid\_t pid;

pid=fork();

if(pid<0) {

printf("Child process not created");

}

else if(pid==0) {

printf("This is child process with Process ID : %d",getpid());

printf("\n");

}

else {

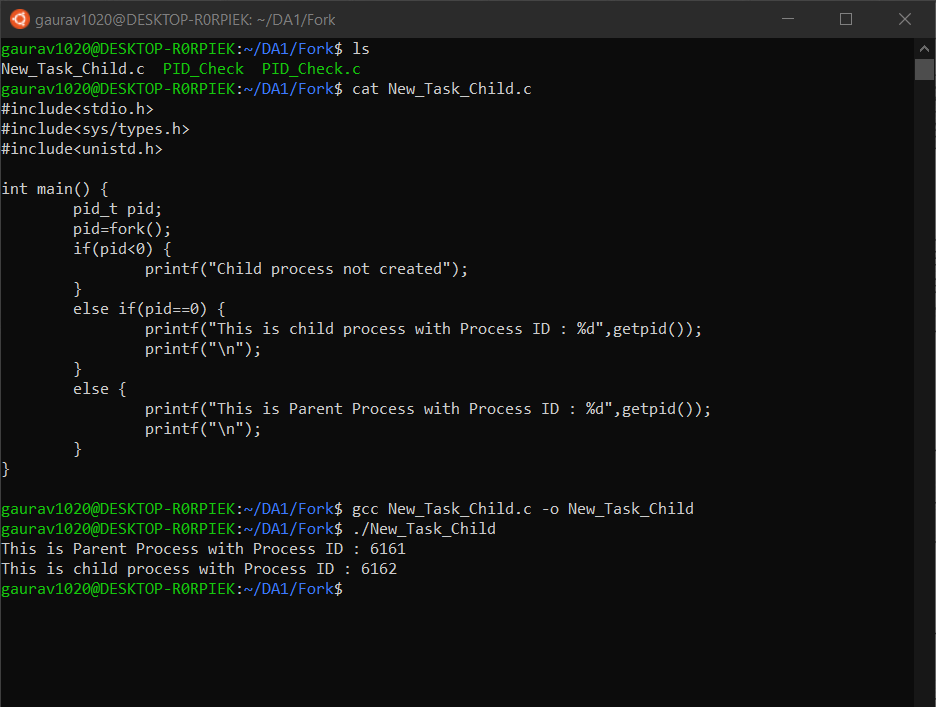
printf("This is Parent Process with Process ID : %d",getpid());

printf("\n");

}

}

**Output:**



**C) Providing the path name and program name to exec()**

**Code:**

**ex1.c**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

printf("This is in program ex1.c with Process ID : %d",getpid());

printf("\n");

char \* arg[] = {"19","BCE","2119",NULL};

execv("./ex2", arg);

return 0;

}

**ex2.c**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

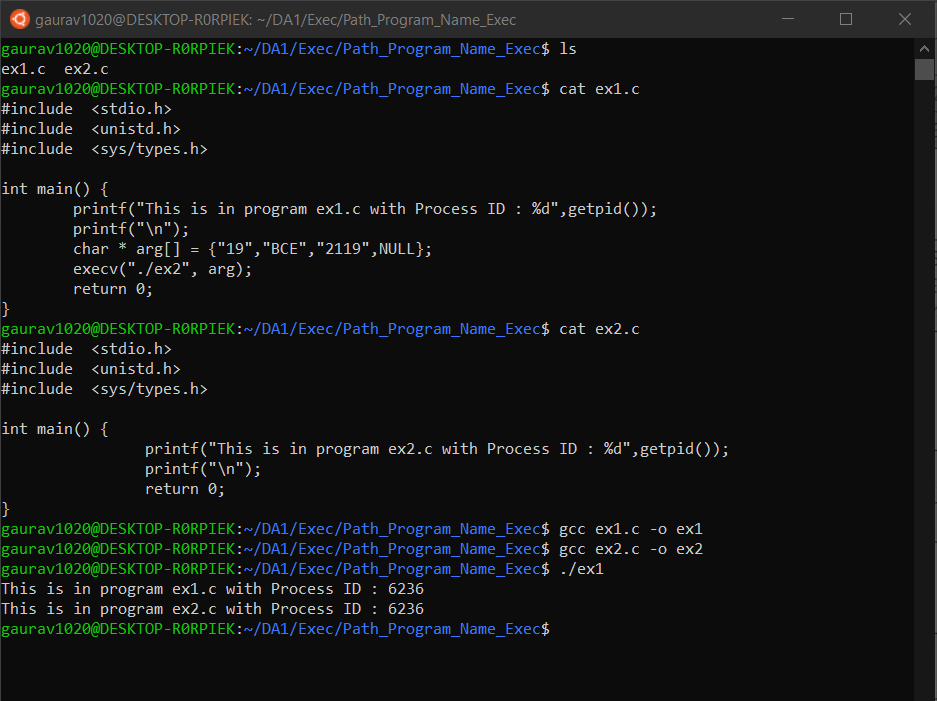
printf("This is in program ex2.c with Process ID : %d",getpid());

printf("\n");

return 0;

}

**Output:**



**D) Synchronizing Parent and child process using wait()**

**Code:**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main() {

pid\_t pid;

pid=fork();

if(pid<0) {

printf("Child process not created");

}

else if(pid==0) {

printf("This is child process with Process ID : %d",getpid());

printf("\nAdding Two Numbers: \n");

int a, b;

scanf("%d %d",&a,&b);

printf("Sum is: %d",a+b);

}

else {

wait();

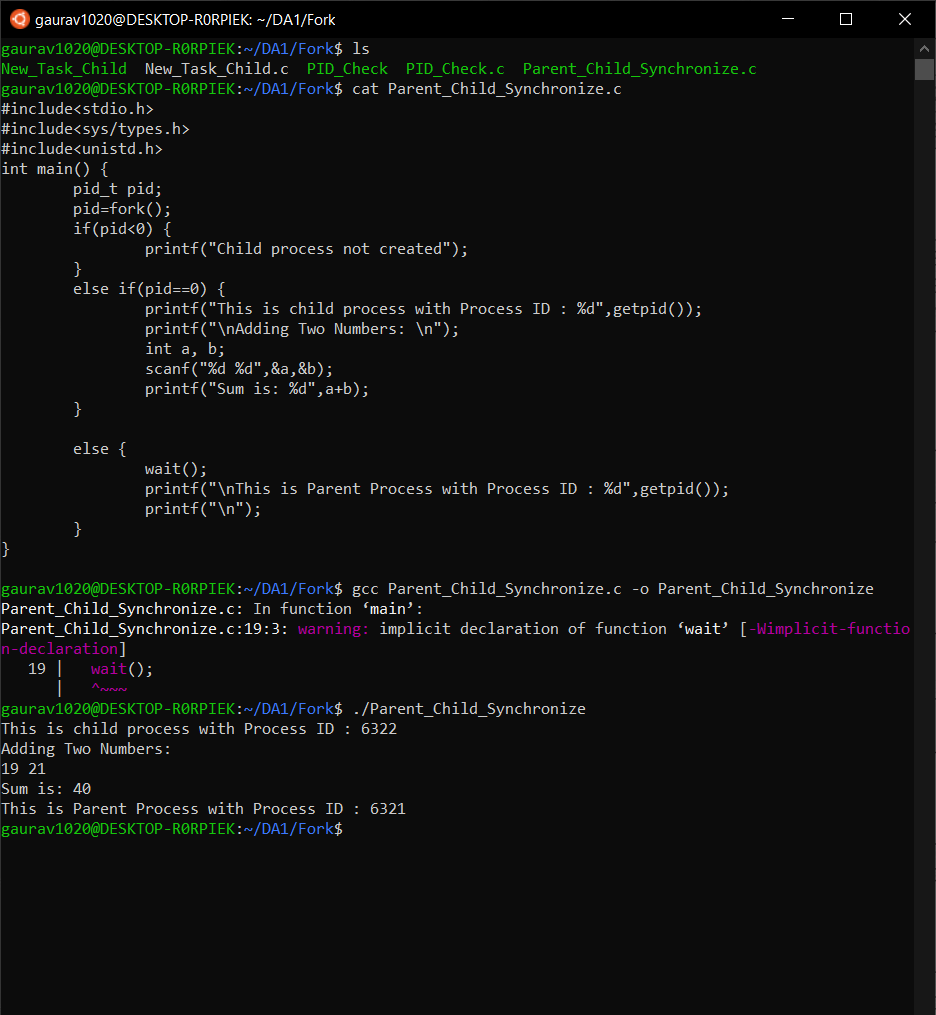
printf("\nThis is Parent Process with Process ID : %d",getpid());

printf("\n");

}

}

**Output:**



**Answer 4)**

**Code:**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

int k = 0;

pid\_t pid;

do {

printf("Enter a valid number to run Collatz Conjecture on:.\n");

scanf("%d", & k);

} while (k < 0);

pid = fork();

if (pid == 0) {

printf("Child Process is Running having Process ID: %d\n",getpid());

printf("%d\n", k);

while (k != 1) {

if (k % 2 == 0) {

k = k / 2;

} else if (k % 2 == 1) {

k = 3 \* (k) + 1;

}

printf("%d\n", k);

}

printf("Child process is done.\n");

}

else {

printf("Parents is waiting for the completion of child process.\n");

wait();

printf("Parent process is done having Process ID:%d\n",getpid());

}

return 0;}

**Output:**

