Student Name: Deepanshu Gupta

Registration No.: 11813348

Roll No.:40

Section:K18ZV

Group:2

Email Address: [deepugupta488@gmail.com](mailto:deepugupta488@gmail.com)

Question Code: 1and 26

Question No.1

**Description**

**fork**() creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited.

Under Linux, **fork**() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent’s page tables, and to create a unique task structure for the child.

# **Return Value**

On success, the PID of the child process is returned in the parent’s thread of execution, and a 0 is returned in the child’s thread of execution. On failure, a -1 will be returned in the parent’s context, no child process will be created, and *errno* will be set appropriately.

**Algorithm:**

**Step 1:**create a pid \_t p =fork();

**Step 2:**p holds two values 1 or 0.

**A:**if 1 it is in parent pass wait(Null) to wait the parent to wait until child execute

B: if 0 t is in child perform the operation need to be executed,

**Step 3:**create a for loop which start from value n and go upto 1 and print all the values.

**Step 4:**exit loop;

**Code:**

#include<unistd.h>

#include<fcntl.h>

#include<stdlib.h>

#include<stdio.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<sys/wait.h>

int main(){

pid\_t p;

p=fork();

int n;

scanf("%d",&n);

if(p>0){

wait(NULL);

}

if(p==0){

if(n%2==0){

for(int i=n;i>0;i=i/2){

printf("%d",i);

printf("\n");

}

}

else{

for(int i=n+1;i>0;i=i/2)){

printf("%d",i);

printf("\n");

}

}

}

}

**Command to run the code:**

1. gcc filename.c

2. ./a.out

3. “enter any number”

**Test cases**

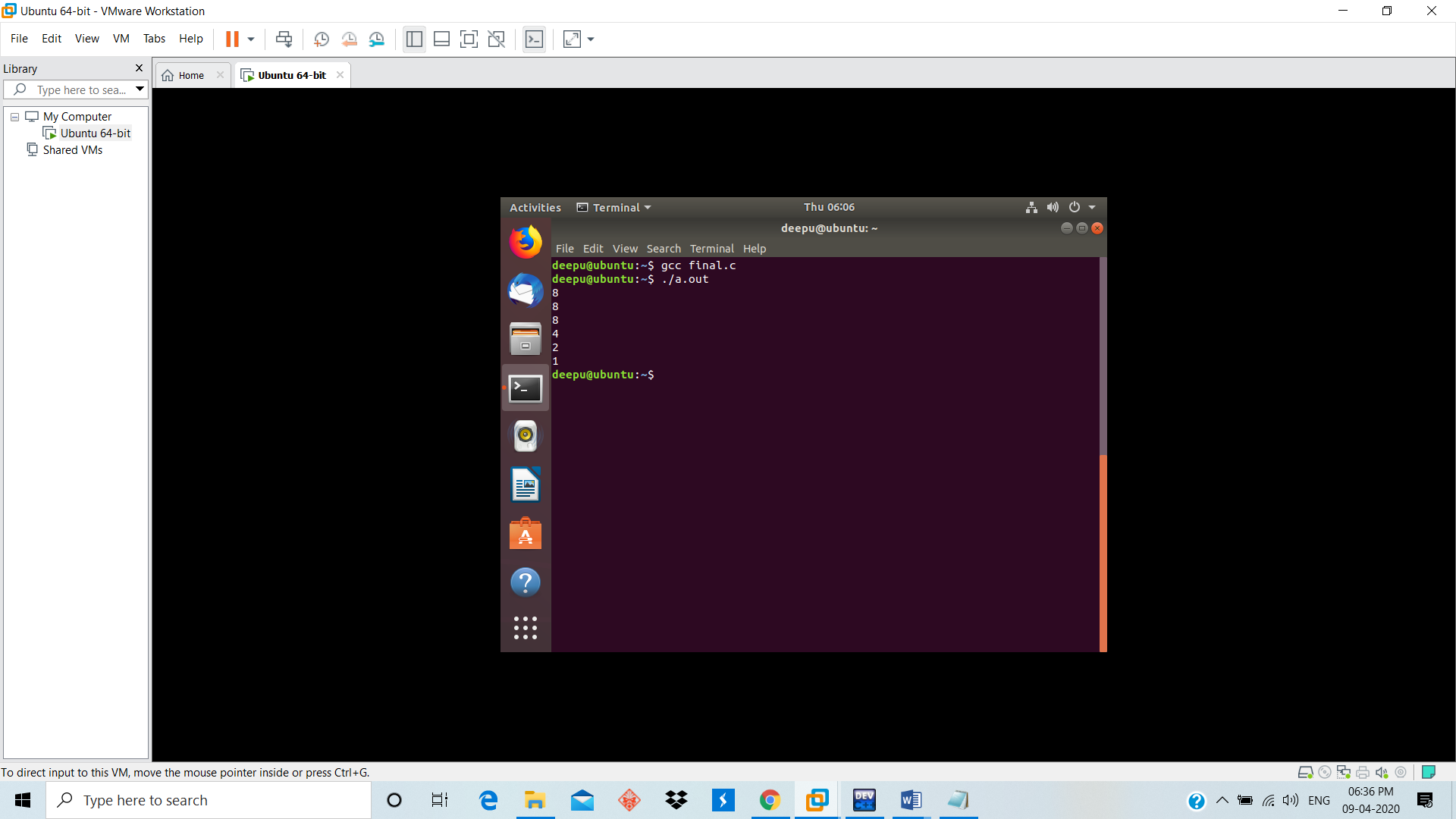
If number is 8

Ans is 8, 4, 2, 1

If number is 16

Ans is 16,8,4,2,1

**Output**



**Question No. 26**

**Algorithm**

Step1: Create pipe file descriptor

Step2: found out the pipe, open both input and output files. Make sure your program closes file handles.

Step3: Create the processes. Read from the input data in parent process and write to the pipe. Write to the computer file within the child process. Make sure your program closes file handles.

Step4: Add a timestamp within the child process to record the time when the reading-writing is completed . Print the timestamp to console output.

**C-code outline**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include

#include

#include

#include

#define READ\_END 0

#define WRITE\_END 1

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

{

int rv;

pid\_t pid;

int c;

char rb[2], wb[2]; **/\* bytes for reading/writing \*/**

int ffd[2]; **/\* file descriptor \*/**

**/\* Step1: Create pipe descriptor \*/**

pipe(ffd); //??

**/\* open the input file \*/**

ffd[READ\_END] = open(argv[1], O\_RDONLY);

if (ffd[READ\_END] < 0) {

fprintf(stderr,"Unable to open %s\n",argv[1]);

return 1;

}

/\* open the output file \*/

ffd[1] = open(argv[2], O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR);

if (ffd[1] < 0) {

fprintf(stderr,"Unable to open %s\n",argv[2]);

/\* close the input file \*/

close(ffd[0]);

return 1;

}

**/\* Step2: set up the pipe \*/**

**/\* make sure your program closes file handles\*/**

**/\* Step3: create the processes \*/**

**/\* read from the input data and write to the pipe \*/**

**/\* read from the pipe and write to the computer file \*/**

**Code:**

|  |
| --- |
| #include<stdio.h>  #include<sys/types.h>  #include<sys/stat.h>  #include<fcntl.h>  #include<string.h>  #include<stdlib.h>  #include<unistd.h>  #include <sys/wait.h>  #define size 50  int main(int arguments, char\* a[])  {  int files[2];  pid\_t cid;  char buffer[size];  pipe(files);  if( arguments != 3 )  {  printf("ERROR: 2 parameters required.\n");  exit(1);  }  int filesOpen = open(a[1], 0);  int Target = open(a[2], O\_RDWR|O\_CREAT|O\_APPEND, 0666);  if (filesOpen == -1 || Target == -1)  {  printf("Files can not be opend\n");  exit(1);  }  cid = fork();  if (cid == 0)  {  close(files[1]);  while (read(files[0], buffer, sizeof(buffer)) > 0)  {  write(Target, buffer, strlen(buffer) - 1);  }  close(files[0]);  close(Target);  }  else  {  close(files[0]);  while (read(filesOpen, buffer, sizeof(buffer)) > 0)  {  write(files[1], buffer, sizeof(buffer));  memset(buffer, 0, size);  }  close(files[1]);  close(filesOpen);  wait(NULL);  }  }  **Command to run the code:**  1. gcc filename.c -o cp  2. echo "enter any data here." > input.txt  3. ./cp input.txt copy.txt  4. cat copy.txt  **Output:** |