NAME- KRISHNA KUMAR MAHTO

REG-16BIT0453

a.)

CODE:

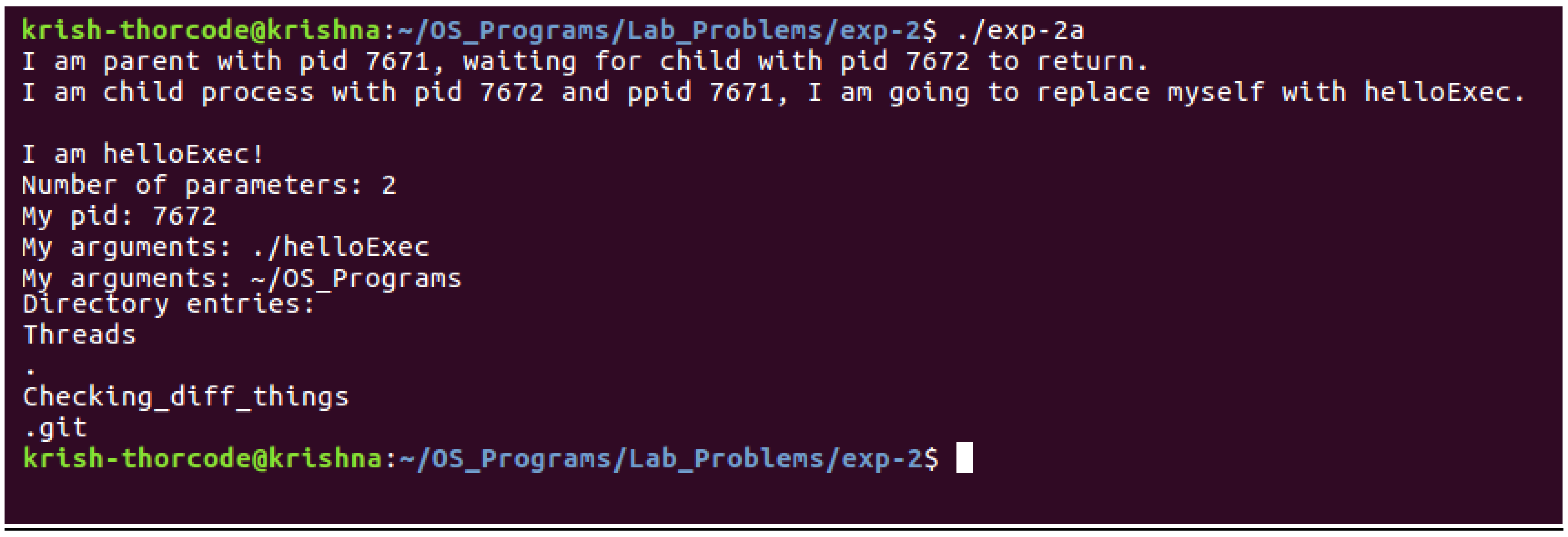
Program that calls exec() system call:

```
/* This file calls a helloExec file and passes directory path as args that  will use directory access commands to access
that directory. */

#include<sys/wait.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h> // for fork, stat
#include<sys/types.h> //for closedir,opendir,pid_t
#include<dirent.h> // for opendir,closedir,DIR*

int main(int argc,char **argv)
{
pid_t pid=fork();
int child_status=0;
if(pid==0) // if child process
{

/* child process will ask a brand new process to replace itself. */
printf("I am child process with pid %d and ppid %d, I am going to replace myself with
helloExec.\n\n",(int)getpid(),(int)getppid());

char *args[]={"./helloExec","~/OS_Programs",NULL}; //an argument for the new process;  as a convention first arg is name
of the file to be executed

execvp("./helloExec",args); //passing arg to the new process- helloExec
fprintf(stderr,"execvp failed!\n\n"); //no conditional block is required for fprintf as execvp returns only if it fails
exit(1);
}

else
{

printf("I am parent with pid %d, waiting for child with pid %d to return.\n",(int)getpid(),(int)pid);

}
return 0;
}
```

helloExec (called by exec() function call):

```
/* This helloExec will access the directory whose address will be passed by the child process */

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h> //for opendir,closedir,pid_t
#include<dirent.h> //for opendir,closedir,DIR*

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

int i;

puts("I am helloExec!");
printf("Number of parameters: %d\n",argc);

printf("My pid: %d\n",(int)getpid());

for(i=0;i<argc;i++)
printf("My arguments: %s",argv[i]);

DIR* dir_handle=opendir(argv[1]);
struct dirent *dir_entry;

do
{
dir_entry=readdir(dir_handle);
printf("%s\n",dir_entry->d_name);
}while(readdir(dir_handle)!=NULL);

closedir(dir_handle);
return 0;
}
```

OUTPUT:



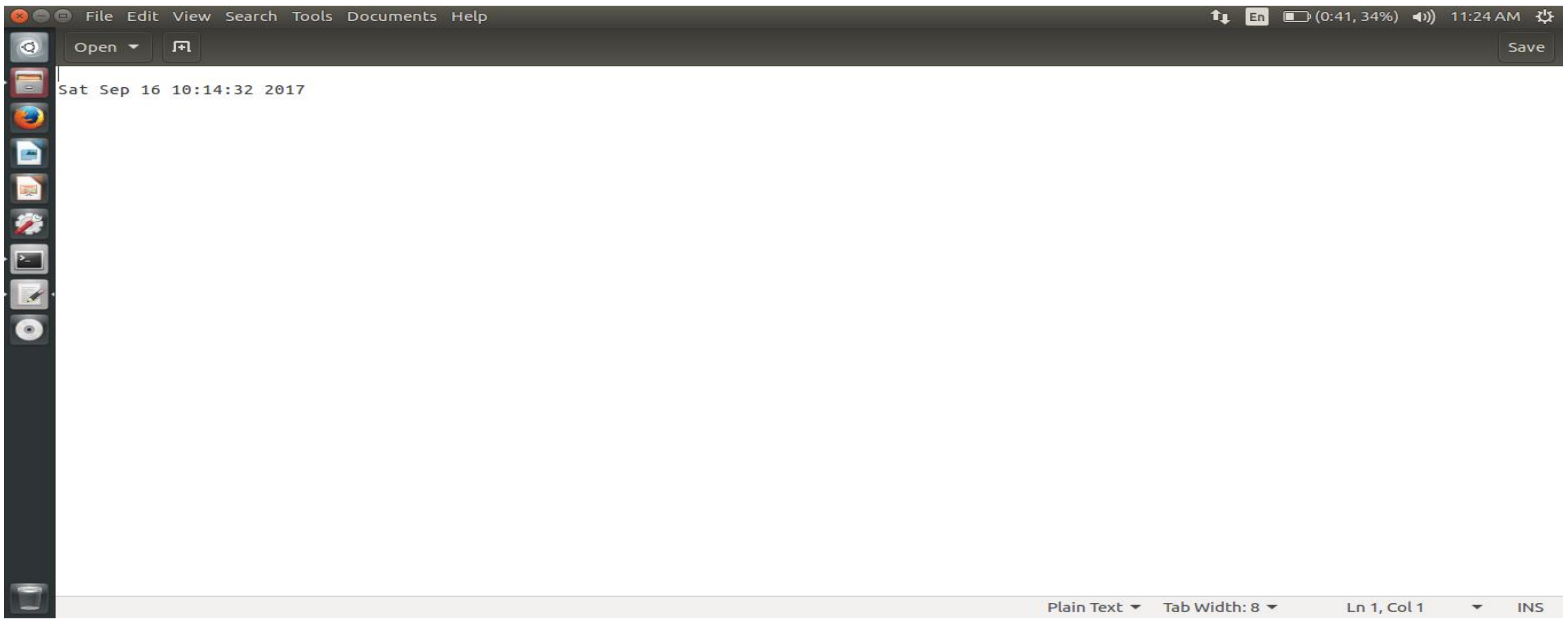
```
krish-thorcode@krishna:~/OS_Programs/Lab_Problems/exp-2$ ./exp-2a
I am parent with pid 7671, waiting for child with pid 7672 to return.
I am child process with pid 7672 and ppid 7671, I am going to replace myself with helloExec.

I am helloExec!
Number of parameters: 2
My pid: 7672
My arguments: ./helloExec
My arguments: ~/OS_Programs
Directory entries:
Threads
.
Checking_diff_things
.git
krish-thorcode@krishna:~/OS_Programs/Lab_Problems/exp-2$ 
```

b.)

CODE:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>

int main(int c,char **argv)
{
pid_t pid;

pid=fork();

if(pid==0)
{
//child process
printf("I am child and my pid is %d and my parent's pid is: %d\n\n ",getpid(),getppid());
sleep(10); //child goes to sleep
}

else
{
//parent process

printf("I am parent process and my pid is: %d\n",getpid());

/*as the parent will exit in the next line and the child process will still be running the sleep() system call, the child
will become orphan.*/

exit(0);
}

return 0;

}
```

OUTPUT (opened an empty file and written current date and time):


File Edit View Search Tools Documents Help
En (0:41, 34%) 11:24 AM
Open
Save
Plain Text Tab Width: 8 Ln 1, Col 1 INS

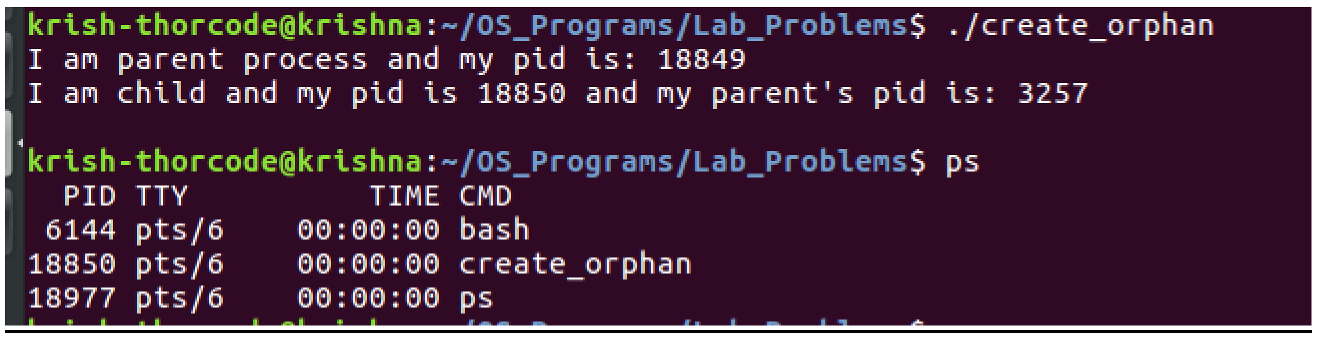
```
Sat Sep 16 10:14:32 2017
```

c.)

CODE:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>

int main(int c,char **argv)
{
pid_t pid;

pid=fork();

if(pid==0)
{
//child process
printf("I am child and my pid is %d and my parent's pid is: %d\n\n ",getpid(),getppid());
sleep(10); //child goes to sleep
}

else
{
//parent process

printf("I am parent process and my pid is: %d\n",getpid());

/*as the parent will exit in the next line and the child process will still be running the sleep() system call, the child
will become orphan.*/

exit(0);
}

return 0;

}
```

OUTPUT:



```
krish-thorcode@krishna:~/OS_Programs/Lab_Problems$ ./create_orphan
I am parent process and my pid is: 18849
I am child and my pid is 18850 and my parent's pid is: 3257

krish-thorcode@krishna:~/OS_Programs/Lab_Problems$ ps
  PID TTY          TIME CMD
 6144 pts/6    00:00:00 bash
18850 pts/6    00:00:00 create_orphan
18977 pts/6    00:00:00 ps
```

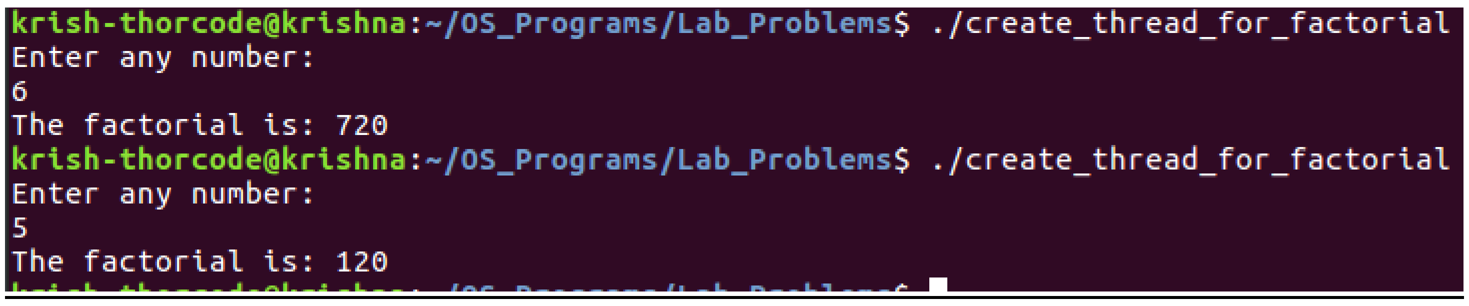
Child process is still running, parent process has exited.

d.)

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<pthread.h>

void *factorial_thread(void *num_recv)
{
int num=*((int*)num_recv);
int *fact=(int*)malloc(sizeof(int));
int i;
*fact=1;
for(i=num;i>1;i--)
(*(fact))*=i;

printf("The factorial is: %d\n",*fact);

return NULL;
}

int main(int argc,char **argv)
{
int n; //number whose factorial will be calculated
static int *r; //result of the factorial's address
pthread_t thread_id;

puts("Enter any number: ");
scanf("%d",&n);

pthread_create(&thread_id,NULL,&factorial_thread,(void*)&n);
pthread_join(thread_id,NULL);

return 0;
}
```

OUTPUT:



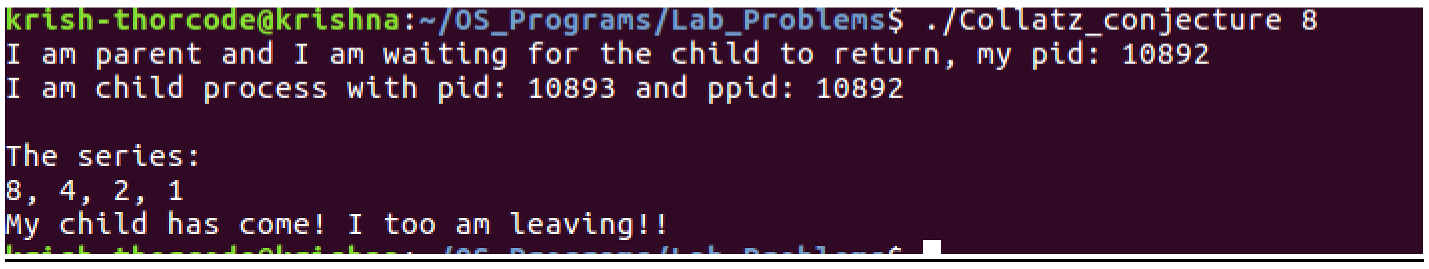
```
krish-thorcode@krishna:~/OS_Programs/Lab_Problems$ ./create_thread_for_factorial
Enter any number:
6
The factorial is: 720
krish-thorcode@krishna:~/OS_Programs/Lab_Problems$ ./create_thread_for_factorial
Enter any number:
5
The factorial is: 120
```

e.)

CODE:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<string.h>
#include<math.h>

int main(int argc,char** argv)
{
if(*(argv+1)[0]=='-')
{
puts("Please enter a positive number and come back.\nExiting...");
exit(1);
}
int count=0,i=0,num=0,n;

if(*(argv+1)[0]!='+')
{
count=strlen(*(argv+1));

for(i=0;i<count;i++)
num+=((*(argv+1)[count-i-1])-48)*pow(10,i);
}

else
{
count=strlen(*(argv+1));

for(i=0;i<count-1;i++)
num+=((*(argv+1)[count-i-1])-48)*pow(10,i);
}
/* it is important to initialise child_status because it's address will be used in wait() in parent process. Until un-
initialised, it will have some garbage value. */

n=num;
int child_status=0;
pid_t pid=fork();

if(pid==0) //child process
{
printf("I am child process with pid: %d and ppid: %d\n\n",(int)getpid(),(int)getppid());

puts("The series: ");
for(;n!=1;)
{
if(n%2==0)
{
printf("%d, ",n);
n=n/2;
}
else
{
printf("%d, ",n);
n=(3*n)+1;
}
}
printf("%d\n",n);

return 0;
}

else //parent process
{
printf("I am parent and I am waiting for the child to return, my pid: %d\n",(int)getpid());

wait(&child_status); //waiting for the child to come

puts("My child has come! I too am leaving!!");

}

return 0;
}
```

OUTPUT:



```
krish-thorcode@krishna:~/OS_Programs/Lab_Problems$ ./Collatz_conjecture 8
I am parent and I am waiting for the child to return, my pid: 10892
I am child process with pid: 10893 and ppid: 10892

The series:
8, 4, 2, 1
My child has come! I too am leaving!!
```