NAME : ADITYA RAJ PANDIT

REG NO : 23BRS1157

SYSTEM CALLS

ASSIGNMENT :

1) Create a process and print parent ID and Child ID

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

int main() {
    pid_t pid = fork();  // Create a new process

    if (pid == 0) {
        // Child process
        printf("Child ⇒ PPID: %d, PID: %d\n", getppid(), getpid());
    } else if (pid > 0) {
        // Parent process
        printf("Parent ⇒ PID: %d, Child PID: %d\n", getpid(), pid);
    } else {
        // Fork failed
        printf("Fork failed!\n");
        return 1;
    }

    return 0;
}
```

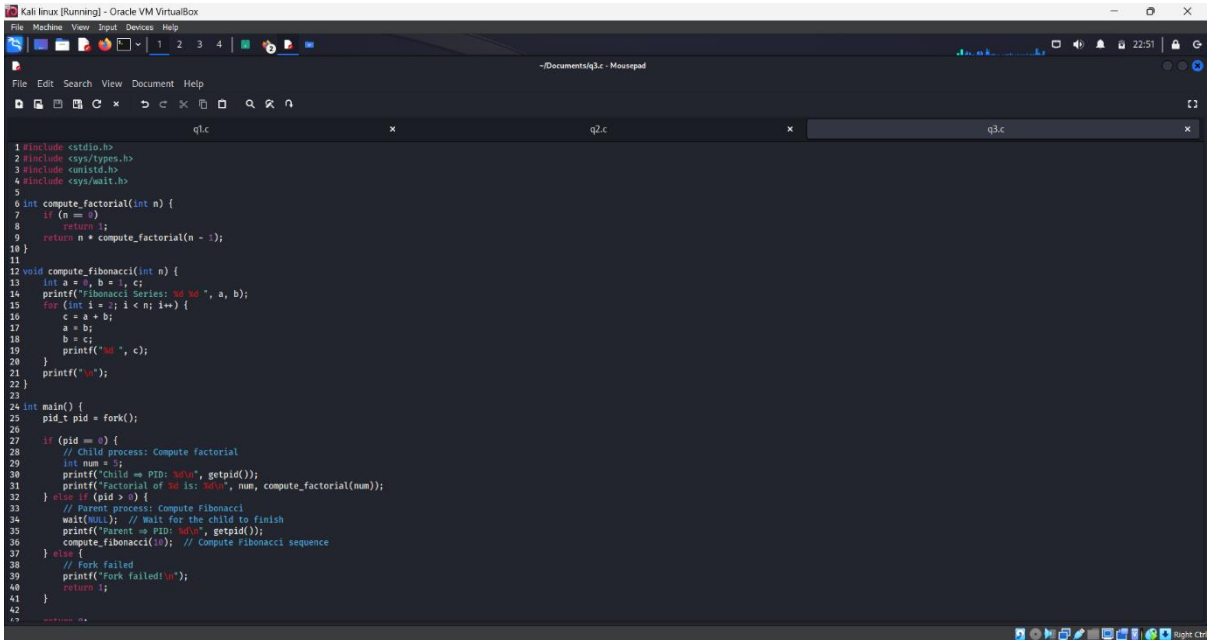
```
adigen@kali: ~/Documents
File  Actions  Edit  View  Help

┌──(adigen㉿kali)-[~/Documents]
└─$ gcc q1.c -o q1

┌──(adigen㉿kali)-[~/Documents]
└─$ ./q1
Parent ⇒ PID: 47866, Child PID: 47867

Child ⇒ PPID: 1, PID: 47867
┌──(adigen㉿kali)-[~/Documents]
└─$
```

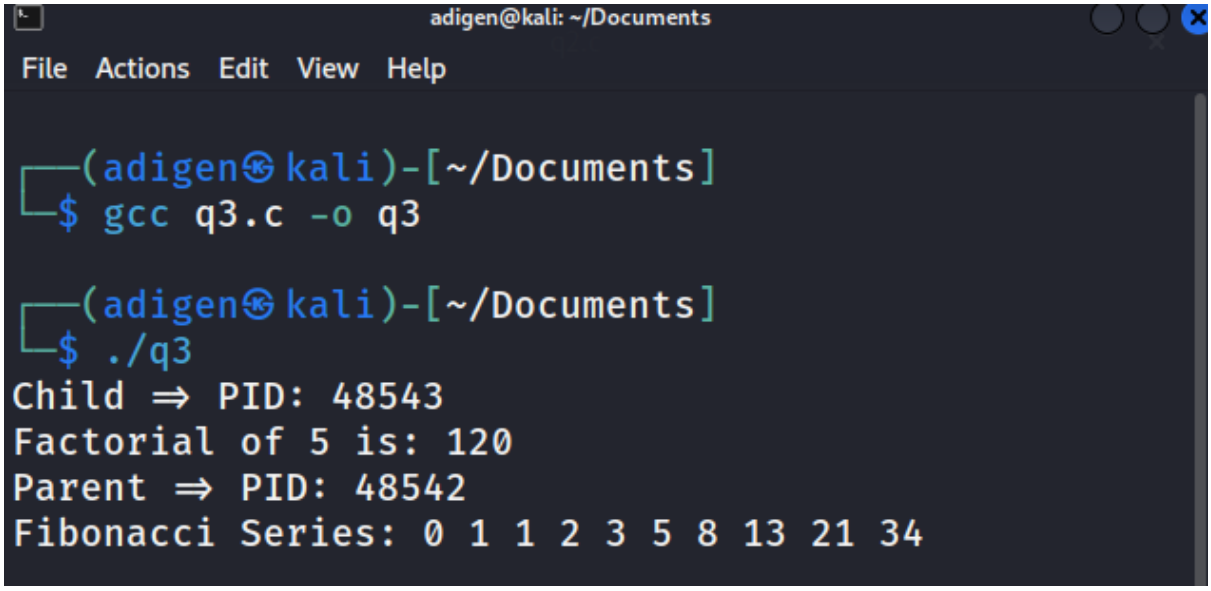
2) Create a Process and let the child do some task like computing Fibonacci

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

void compute_fibonacci(int n) {
    int a = 0, b = 1, c;
    printf("Fibonacci Series: %d %d ", a, b);
    for (int i = 2; i < n; i++) {
        c = a + b;
        a = b;
        b = c;
        printf("%d ", c);
    }
    printf("\n");
}

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child => PID: %d\n", getpid());
        compute_fibonacci(10);  // Compute Fibonacci sequence
    } else if (pid > 0) {
        // Parent process
        wait(NULL);  // Wait for the child to finish
        printf("Parent => PID: %d\n", getpid());
    } else {
        // Fork failed
        printf("Fork failed!\n");
        return 1;
    }

    return 0;
}
```

```
adigen@kali: ~/Documents
File  Actions  Edit  View  Help

┌──(adigen㉿kali)-[~/Documents]
└─$ gcc q2.c -o q2

┌──(adigen㉿kali)-[~/Documents]
└─$ ./q2
Child => PID: 48181
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
Parent => PID: 48180
```

3) Create a process and compute factorial in child and Fibonacci in parent as executable





CLASS WORK :

1) **Write programs using the following system calls of UNIX operating system: fork, getpid, getppid, exit**

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

int main(void)
{
    pid_t pid = fork();

    if(pid == 0)
    {
        printf("Child ⇒ PPID: %d PID: %d\n", getppid(), getpid());
        exit(EXIT_SUCCESS);
    }
    else if(pid > 0)
    {
        printf("Parent ⇒ PID: %d\n", getpid());
        printf("Waiting for child process to finish.\n");
        wait(NULL);
        printf("Child process finished.\n");
    }
    elseSS
    {
    printf("Unable to create child process.\n");
    }
    return EXIT_SUCCESS;
}
```

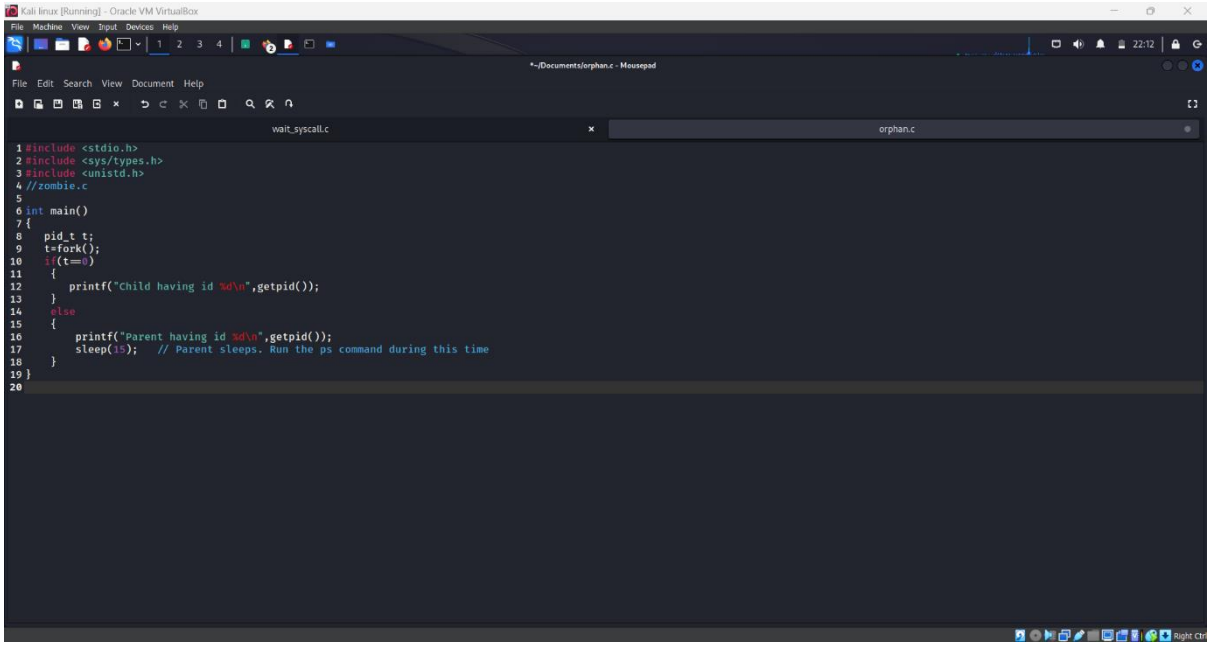
```
adigen@kali: ~/Documents
File  Actions  Edit  View  Help
systemcalls.c

┌──(adigen㉿kali)-[~/Documents]
└─$ gcc systemcalls.c

┌──(adigen㉿kali)-[~/Documents]
└─$ ls
a.out  systemcalls.c

┌──(adigen㉿kali)-[~/Documents]
└─$ ./a.out
Parent ⇒ PID: 12313
Waiting for child process to finish.
Child ⇒ PPID: 12313 PID: 12314
Child process finished.

┌──(adigen㉿kali)-[~/Documents]
└─$
```

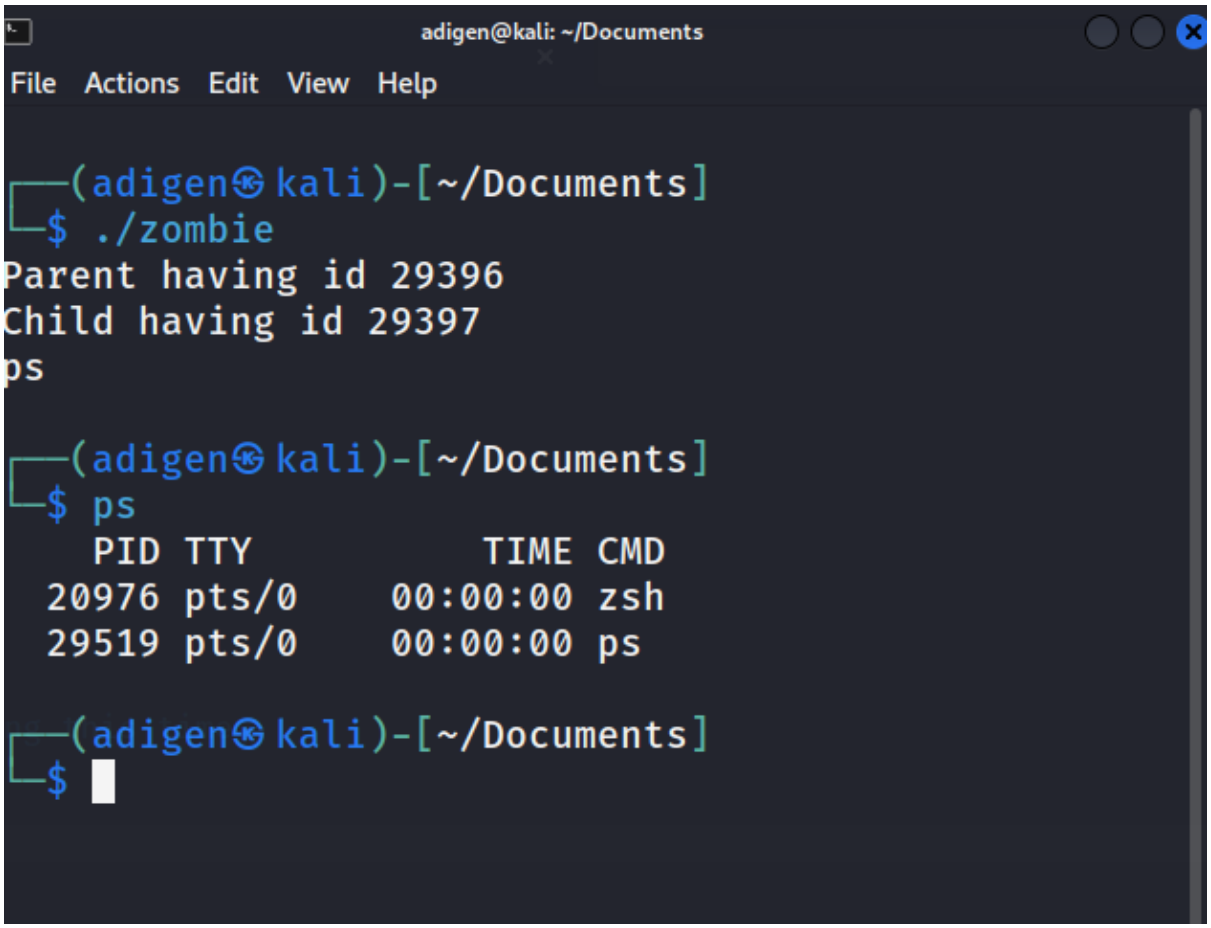
2) **Program for wait () system call which makes the parent process wait for the child to finish.**

```c
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
#include<sys/wait.h>
int main()
{
pid_t p;
printf("before fork\n");
p=fork();
if(p==0)//child
{
printf("I am child having id %d\n",getpid());
printf("My parent's id is %d\n",getppid());
}
else//parent
{
wait(NULL);
printf("My child's id is %d\n",p);
printf("I am parent having id %d\n",getpid());
}
printf("Common\n");
}
```

```
└─$ gcc wait_syscall.c -o wait

┌──(adigen㉿kali)-[~/Documents]
└─$ ls
a.out  systemcalls.c  wait  wait_syscall.c

┌──(adigen㉿kali)-[~/Documents]
└─$ ./wait
before fork
I am child having id 17889
My parent's id is 17888
Common
My child's id is 17889
I am parent having id 17888
Common

┌──(adigen㉿kali)-[~/Documents]
└─$
```

3) Demonstrates the creation and termination of a zombie process





4) Demonstrate the creation of an orphan process

Kali linux [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

~/Documents/orphan1.c - Mousepad

File Edit Search View Document Help

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

int main()
{
pid_t p;
p=fork();
      if(p==0)
     {
         sleep(5); //child goes to sleep and in the meantime parent terminates
         printf("I am child having PID %d\n",getpid());
         printf("My parent PID is %d\n",getppid());
     }
     else
     {
         printf("I am parent having PID %d\n",getpid());
         printf("My child PID is %d\n",p);
     }
     return 0;
}
```

adigen@kali: ~/Documents

File Actions Edit View Help

```
┌──(adigen㉿kali)-[~/Documents]
└─$ gcc orphan1.c -o orph

┌──(adigen㉿kali)-[~/Documents]
└─$ ./orph
I am parent having PID 33658
My child PID is 33659

┌──(adigen㉿kali)-[~/Documents]
└─$ I am child having PID 33659
My parent PID is 1
```