Ayush Goyal
190905522 CSE D Roll 62

Operating Systems Lab 3: Processes and Signals

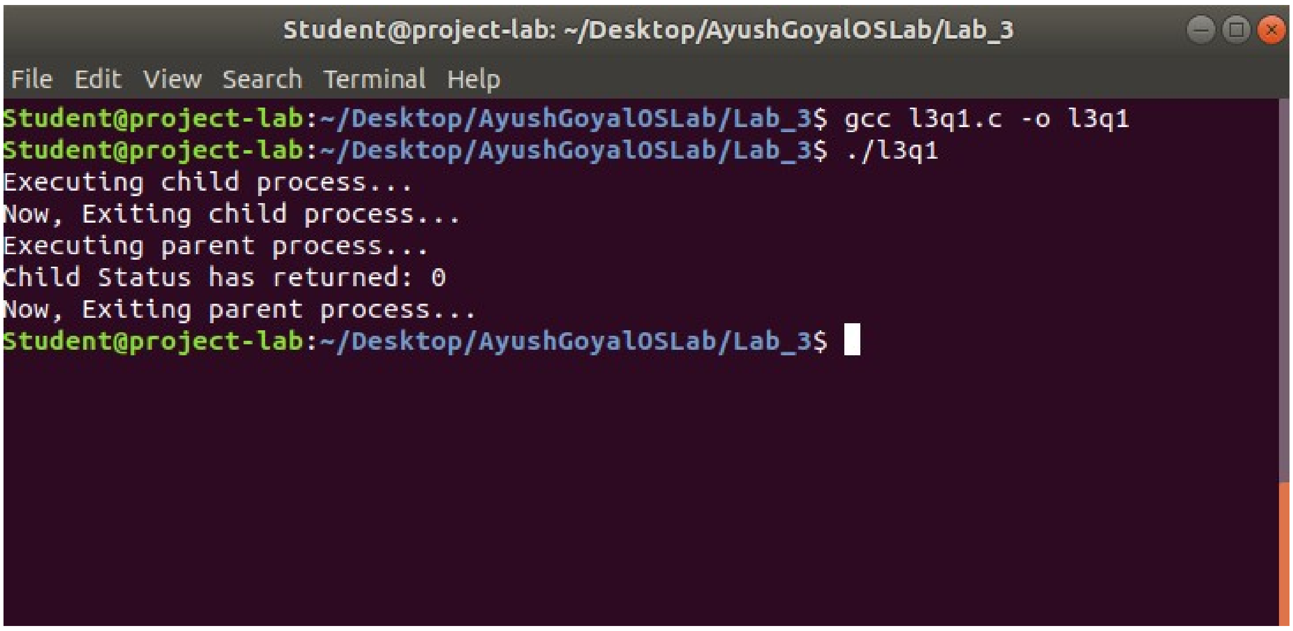
1. Write a C program to block a parent process until the child completes using a wait system call.

Code:

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

int main(){
	pid_t pid;
	int status;
	pid = fork();
	switch(pid){
		case -1: printf("Error occured!...\n");
				 exit(-1);
		case 0: printf("Executing child process...\nNow, Exiting child process...\n");
				exit(0);
		default: wait(&status);
				 printf("Executing parent process...\nChild Status has returned: %d\
nNow, Exiting parent process...\n", status);
	}
	return 0;
}
```

Output:



```
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ gcc l3q1.c -o l3q1
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ ./l3q1
Executing child process...
Now, Exiting child process...
Executing parent process...
Child Status has returned: 0
Now, Exiting parent process...
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$
```

Student@project-lab: ~/Desktop/AyushGoyalOSLab/Lab_3
File Edit View Search Terminal Help

2. Write a C program to load the binary executable of the previous program in a child process using the exec system call.

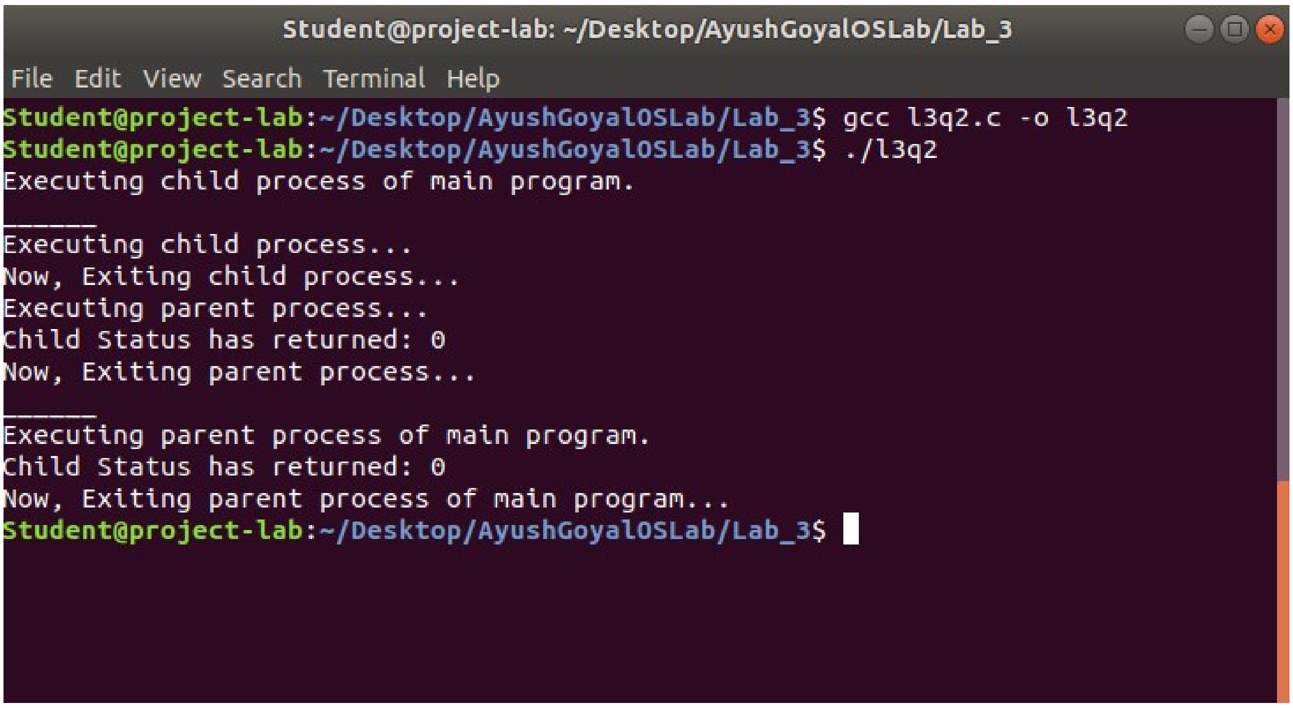
Code:

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

int main(){
        pid_t pid;
        int status;
        pid = fork();
        switch(pid){
                case -1: printf("Error occured!...\n");
                                 exit(-1);
                case 0: printf("Executing child process of main program.\n");
                                printf("______\n");
                                execlp("./l3q1", "l3q1", NULL);
                                exit(0);
                default: wait(&status);
                                 printf("______\nExecuting parent process of main program.\nChild
Status has returned: %d\nNow, Exiting parent process of main program...\n", status);

        }
}
```

Output:


Student@project-lab: ~/Desktop/AyushGoyalOSLab/Lab_3
File Edit View Search Terminal Help

```
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ gcc l3q2.c -o l3q2
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ ./l3q2
Executing child process of main program.
______
Executing child process...
Now, Exiting child process...
Executing parent process...
Child Status has returned: 0
Now, Exiting parent process...
______
Executing parent process of main program.
Child Status has returned: 0
Now, Exiting parent process of main program...
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$
```

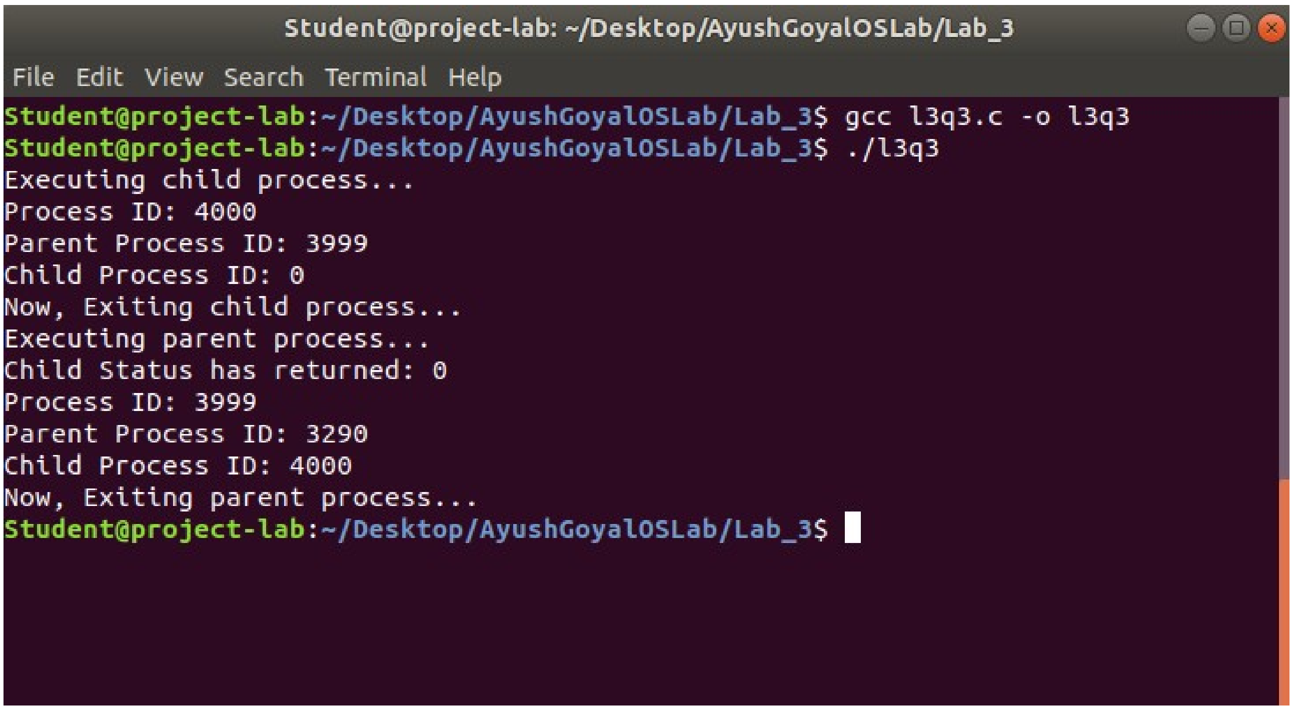
3. Write a program to create a child process. Display the process IDs of the process, parent and child(if any) in both the parent and child processes.

Code:

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

int main(){
	pid_t pid;
	int status;
	pid = fork();
	switch(pid){
		case -1: printf("Error occured!...\n");
				 exit(-1);
		case 0: printf("Executing child process...\nProcess ID: %d\nParent Process ID: %d\
nChild Process ID: %d\nNow, Exiting child process...\n", getpid(), getppid(), pid);
				exit(0);
		default: wait(&status);
				 printf("Executing parent process...\nChild Status has returned: %d\
nProcess ID: %d\nParent Process ID: %d\nChild Process ID: %d\nNow, Exiting parent process...\
n", status, getpid(), getppid(), pid);
	}
	return 0;
}
```

Output:


Student@project-lab: ~/Desktop/AyushGoyalOSLab/Lab_3
File Edit View Search Terminal Help

```
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ gcc l3q3.c -o l3q3
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ ./l3q3
Executing child process...
Process ID: 4000
Parent Process ID: 3999
Child Process ID: 0
Now, Exiting child process...
Executing parent process...
Child Status has returned: 0
Process ID: 3999
Parent Process ID: 3290
Child Process ID: 4000
Now, Exiting parent process...
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$
```

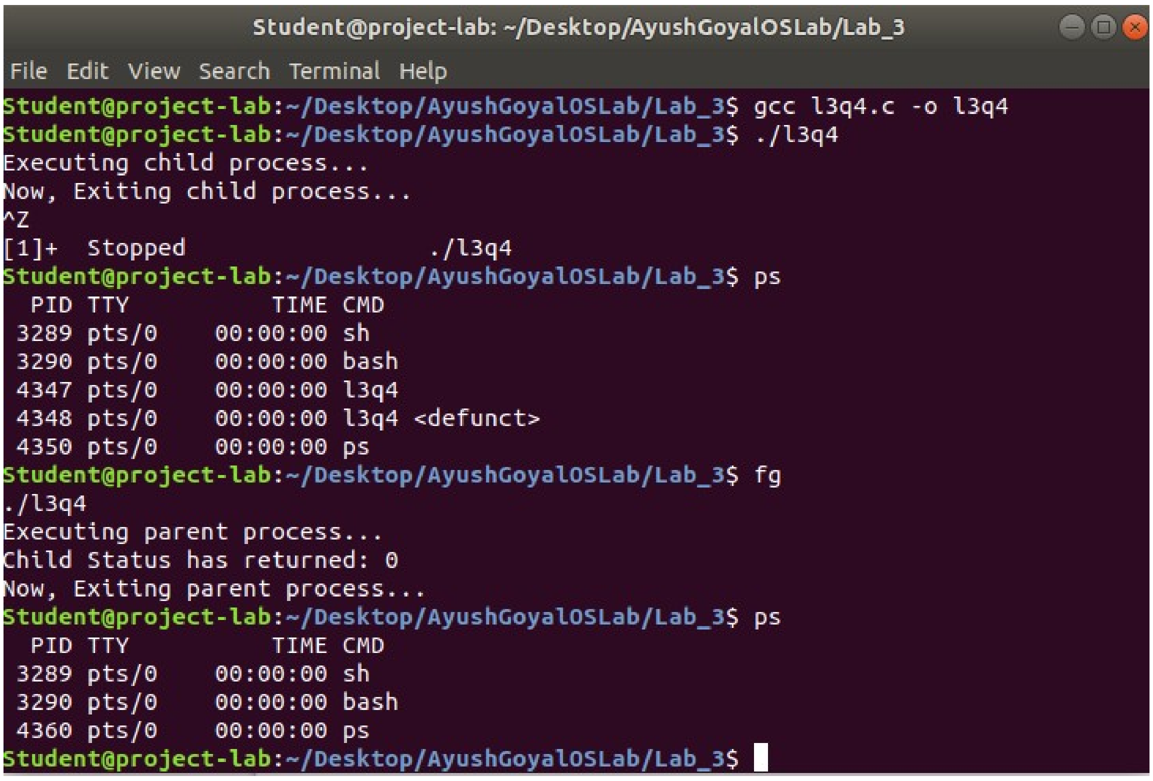
4. Create a zombie(defunct) child process(a child with exit() call, but no corresponding wait() in the sleeping parent) and allow the init process to adopt it(after parent terminates). Run the process as a background process and run the "ps" command.

Code:

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

int main(){
        pid_t pid;
        int status;
        pid = fork();
        switch(pid){
                case -1: printf("Error occured!...\n");
                                 exit(-1);
                case 0: printf("Executing child process...\nNow, Exiting child process...\n");
                                exit(0);
                default: sleep(5);
                                 printf("Executing parent process...\nChild Status has returned: %d\
nNow, Exiting parent process...\n", status);
        }
        return 0;
}
```

Output:



Student@project-lab: ~/Desktop/AyushGoyalOSLab/Lab_3
File Edit View Search Terminal Help

```
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ gcc l3q4.c -o l3q4
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ ./l3q4
Executing child process...
Now, Exiting child process...
^Z
[1]+  Stopped                 ./l3q4
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ ps
  PID TTY          TIME CMD
 3289 pts/0    00:00:00 sh
 3290 pts/0    00:00:00 bash
 4347 pts/0    00:00:00 l3q4
 4348 pts/0    00:00:00 l3q4 <defunct>
 4350 pts/0    00:00:00 ps
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ fg
./l3q4
Executing parent process...
Child Status has returned: 0
Now, Exiting parent process...
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ ps
  PID TTY          TIME CMD
 3289 pts/0    00:00:00 sh
 3290 pts/0    00:00:00 bash
 4360 pts/0    00:00:00 ps
Student@project-lab:~/Desktop/AyushGoyalOSLab/Lab_3$ 
```

THE END