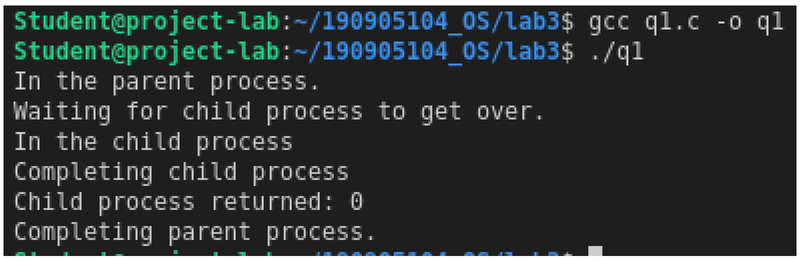
190905104
Lab 3

1)
// Block parent till child completes process using wait

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

int main(){
    pid_t pid;
    int status = 0;

    pid = fork();

    if(pid == -1){
        printf("Error in forking\n");
        exit(1);
    }
    else if(pid == 0){
        printf("In the child process\n");
        printf("Completing child process\n");
    }
    else{
        printf("In the parent process.\nWaiting for child process to get over.\n");
        wait(&status);
        printf("Child process returned: %d\n", status);
        printf("Completing parent process.\n");
    }
    return 0;
}
```



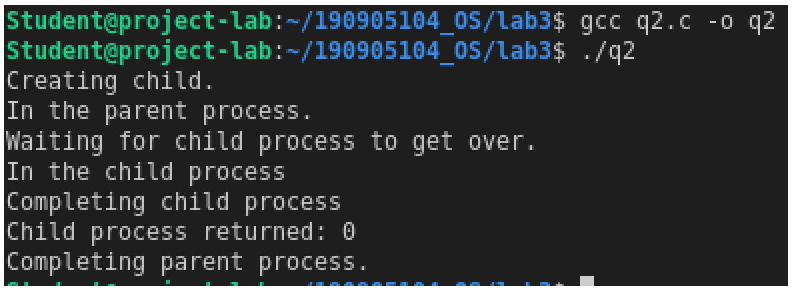
```
Student@project-lab:~/190905104_OS/lab3$ gcc q1.c -o q1
Student@project-lab:~/190905104_OS/lab3$ ./q1
In the parent process.
Waiting for child process to get over.
In the child process
Completing child process
Child process returned: 0
Completing parent process.
```

2)

// Load the binary of the previous program in child and use exec()

```
#include <stdlib.h>
#include <unistd.h>
```

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

int main(){
    pid_t pid;
    int status = 0;

    pid = fork();

    if(pid == -1){
        printf("Error in forking\n");
        exit(1);
    }
    else if(pid == 0){
        execlp("./q1", "q1", NULL);
    }
    else{
        printf("Creating child.\n");
        wait(&status);
    }
    return 0;
}
```

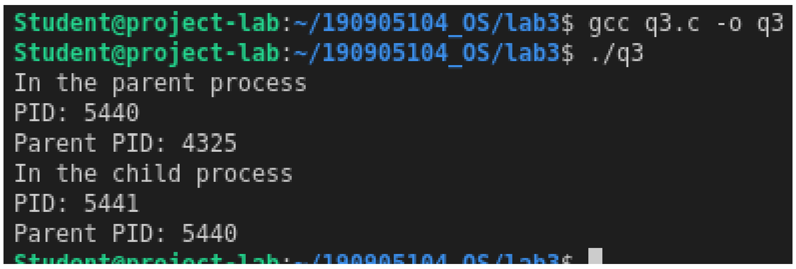


```
Student@project-lab:~/190905104_OS/lab3$ gcc q2.c -o q2
Student@project-lab:~/190905104_OS/lab3$ ./q2
Creating child.
In the parent process.
Waiting for child process to get over.
In the child process
Completing child process
Child process returned: 0
Completing parent process.
```

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.

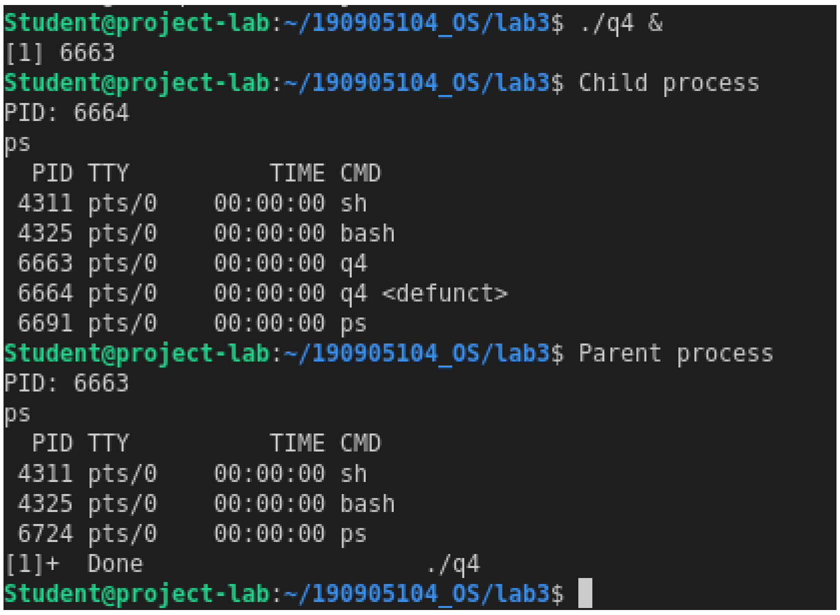
```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(){
    pid_t pid;
    int status = 0;
    pid = fork();

    if(pid == -1){
        printf("Error in forking\n");
        exit(1);
    }
    else if(pid == 0){
        printf("In the child process\n");
        printf("PID: %d\n", getpid());
        printf("Parent PID: %d\n", getppid());
    }
    else{
        printf("In the parent process\n");
        printf("PID: %d\n", getpid());
        printf("Parent PID: %d\n", getppid());
    }
    return 0;
}
```



```
Student@project-lab:~/190905104_OS/lab3$ gcc q3.c -o q3
Student@project-lab:~/190905104_OS/lab3$ ./q3
In the parent process
PID: 5440
Parent PID: 4325
In the child process
PID: 5441
Parent PID: 5440
Student@project-lab:~/190905104_OS/lab3$
```

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

int main(int argc, char const *argv[])
{
    pid_t pid;
    pid = fork();

    if(pid == -1){
        printf("Error\n");
        exit(-1);
    }

    if(pid == 0){
        printf("Child process\n");
        printf("PID: %d\n", getpid());
        exit(0);
    }

    else{
        sleep(5);
        printf("Parent process\n");
        printf("PID: %d\n", getpid());
    }

    return 0;
}
```

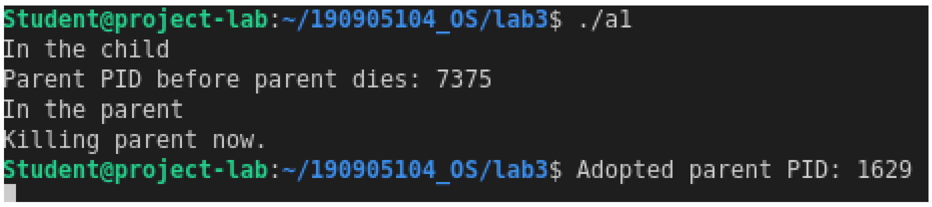


```
Student@project-lab:~/190905104_OS/lab3$ ./q4 &
[1] 6663
Student@project-lab:~/190905104_OS/lab3$ Child process
PID: 6664
ps
  PID TTY          TIME CMD
 4311 pts/0    00:00:00 sh
 4325 pts/0    00:00:00 bash
 6663 pts/0    00:00:00 q4
 6664 pts/0    00:00:00 q4 <defunct>
 6691 pts/0    00:00:00 ps
Student@project-lab:~/190905104_OS/lab3$ Parent process
PID: 6663
ps
  PID TTY          TIME CMD
 4311 pts/0    00:00:00 sh
 4325 pts/0    00:00:00 bash
 6724 pts/0    00:00:00 ps
[1]+  Done                    ./q4
Student@project-lab:~/190905104_OS/lab3$
```

Additional

1) Orphan – Parent dies before child

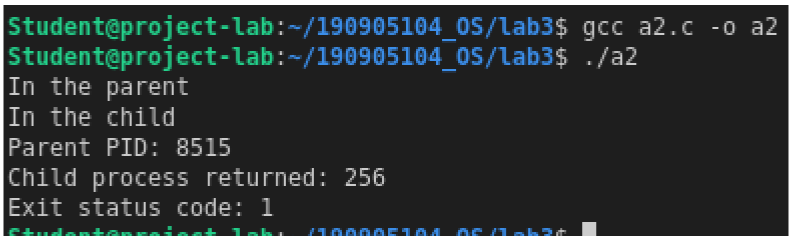
```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(){
    pid_t pid;
    int status = 0;

    pid = fork();

    if(pid == -1){
        printf("Error in forking\n");
        exit(1);
    }
    else if(pid == 0){
        printf("In the child\n");
        printf("Parent PID before parent dies: %d\n", getppid());
        sleep(7);
        printf("Adopted parent PID: %d\n", getppid());
    }
    else{
        sleep(2);
        printf("In the parent\nKilling parent now.\n");
        exit(0);
    }
    return 0;
}
```



```
Student@project-lab:~/190905104_OS/lab3$ ./a1
In the child
Parent PID before parent dies: 7375
In the parent
Killing parent now.
Student@project-lab:~/190905104_OS/lab3$ Adopted parent PID: 1629
```

2)

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

int main(){
    pid_t pid;
    int status = 0;

    pid = fork();

    if(pid == -1){
        printf("Error in forking\n");
        exit(1);
    }
    else if(pid == 0){
        printf("In the child\n");
        printf("Parent PID: %d\n", getppid());
        sleep(5);
        exit(1);
    }
    else{
        // sleep(2);
        printf("In the parent\n");
        wait(&status);
        printf("Child process returned: %d\n", status);
        int es = WEXITSTATUS(status);
        printf("Exit status code: %d\n", es);
    }
    return 0;
}
```



```
Student@project-lab:~/190905104_OS/lab3$ gcc a2.c -o a2
Student@project-lab:~/190905104_OS/lab3$ ./a2
In the parent
In the child
Parent PID: 8515
Child process returned: 256
Exit status code: 1
Student@project-lab:~/190905104_OS/lab3$
```