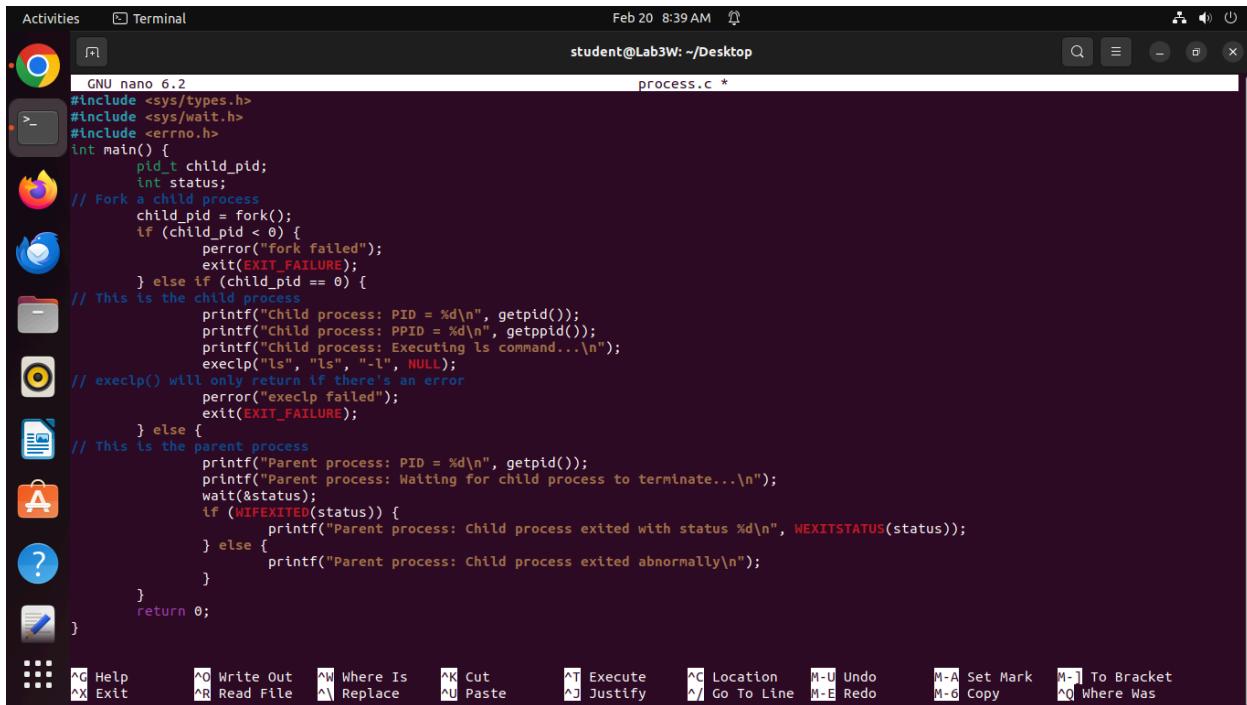


OS Lab 4:

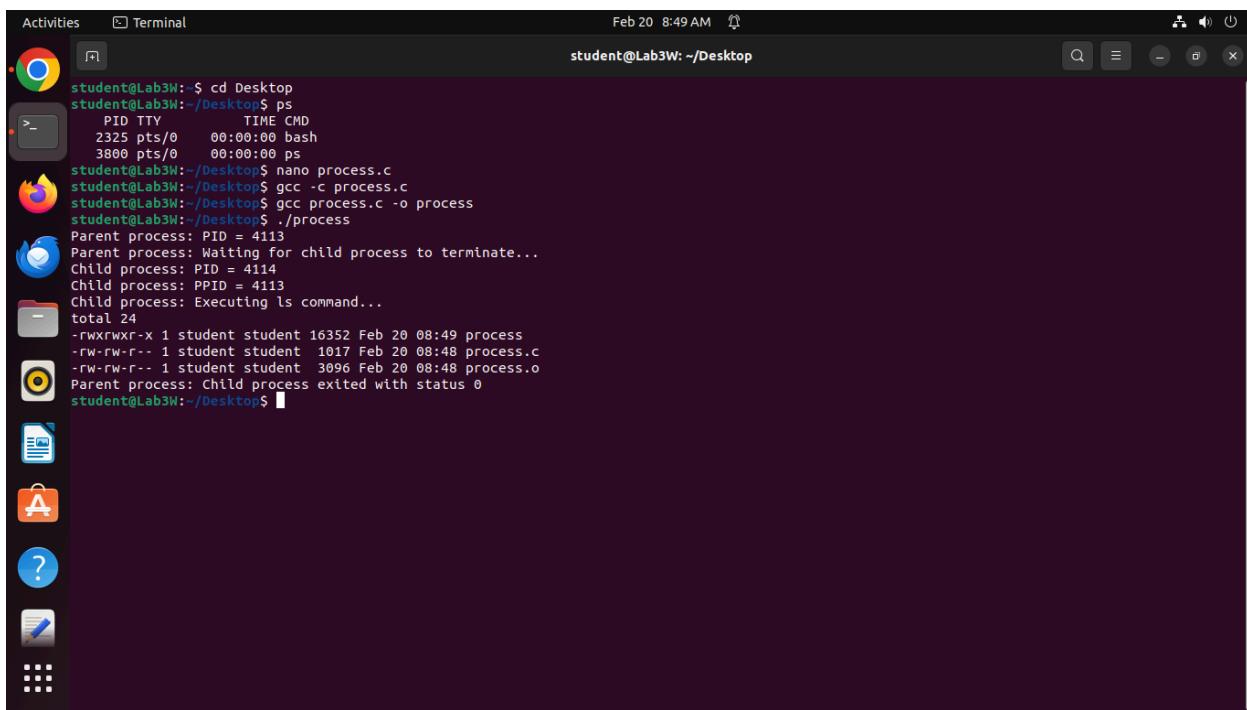
24k-0810

IN-LAB TASK



The screenshot shows a terminal window titled "Terminal" with the command "student@Lab3W: ~/Desktop". The window displays the source code for "process.c". The code includes #include directives for sys/types.h, sys/wait.h, and errno.h. It defines a main() function that forks a child process. If the fork fails, it exits with EXIT_FAILURE. If the child process is successful, it prints its PID and PPID, executes an ls command, and then exits with EXIT_FAILURE. If the parent process is successful, it waits for the child to terminate. If the child exits normally, it prints the status; if abnormally, it prints an error message. Finally, it returns 0.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
int main() {
    pid_t child_pid;
    int status;
    // Fork a child process
    child_pid = fork();
    if (child_pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // This is the child process
        printf("Child process: PID = %d\n", getpid());
        printf("Child process: PPID = %d\n", getppid());
        printf("Child process: Executing ls command...\n");
        execvp("ls", "ls", NULL);
        // execvp() will only return if there's an error
        perror("execvp failed");
        exit(EXIT_FAILURE);
    } else {
        // This is the parent process
        printf("Parent process: PID = %d\n", getpid());
        printf("Parent process: Waiting for child process to terminate...\n");
        wait(&status);
        if (WIFEXITED(status)) {
            printf("Parent process: Child process exited with status %d\n", WEXITSTATUS(status));
        } else {
            printf("Parent process: Child process exited abnormally\n");
        }
    }
    return 0;
}
```



The screenshot shows a terminal window titled "Terminal" with the command "student@Lab3W: ~/Desktop". The user runs "ps" to show processes, then "nano process.c" to edit the source code. They compile it with "gcc -c process.c" and run it with "./process". The output shows the parent process (PID 4113) waiting for the child process (PID 4114). The child process has a PPID of 4113 and is executing an ls command, which lists 24 files. Finally, the parent process exits with status 0.

```
student@Lab3W: $ cd Desktop
student@Lab3W:~/Desktop$ ps
   PID TTY      TIME CMD
 2325 pts/0    00:00:00 bash
 3800 pts/0    00:00:00 ps
student@Lab3W:~/Desktop$ nano process.c
student@Lab3W:~/Desktop$ gcc -c process.c
student@Lab3W:~/Desktop$ gcc process.c -o process
student@Lab3W:~/Desktop$ ./process
Parent process: PID = 4113
Parent process: Waiting for child process to terminate...
Child process: PID = 4114
Child process: PPID = 4113
Child process: Executing ls command...
total 24
-rwxrwxr-x 1 student student 16352 Feb 20 08:49 process
-rw-rw-r-- 1 student student 1017 Feb 20 08:48 process.c
-rw-rw-r-- 1 student student 3096 Feb 20 08:48 process.o
Parent process: Child process exited with status 0
student@Lab3W:~/Desktop$
```

Activities Terminal Feb 20 8:57 AM student@Lab3W: ~/Desktop

```
GNU nano 6.2                                     hello.c
#include <stdio.h>
int main(){
    printf("HELLO BEAUTIFUL PEOPLE <3\n HAPPY RAMADAN QUEENS");
    return 0;
}
```

[Read 5 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-1 To Bracket
^X Exit ^R Read File ^N Replace ^U Paste ^J Justify ^V Go To Line M-E Redo M-6 Copy ^O Where Was

Activities Terminal Feb 20 8:56 AM student@Lab3W: ~/Desktop

```
GNU nano 6.2                                     process.c *
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
int main() {
    pid_t child_pid;
    int status;
    // Fork a child process
    child_pid = fork();
    if (child_pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // This is the child process
        printf("Child process: PID = %d\n", getpid());
        printf("Child process: PPID = %d\n", getppid());
        printf("Child process: Executing ls command...\n");
        execvp("./hello", "hello", NULL);
        // execvp() will only return if there's an error
        perror("execvp failed");
        exit(EXIT_FAILURE);
    } else {
        // This is the parent process
        printf("Parent process: PID = %d\n", getpid());
        printf("Parent process: Waiting for child process to terminate...\n");
        wait(&status);
        if (WIFEXITED(status)) {
            printf("Parent process: Child process exited with status %d\n", WEXITSTATUS(status));
        } else {
            printf("Parent process: Child process exited abnormally\n");
        }
    }
}
```

Save modified buffer?
Y Yes N No ^C Cancel

Activities Terminal Feb 20 9:04 AM student@Lab3W: ~/Desktop

```
GNU nano 6.2 file.c *
```

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

int main() {
    int fd1, fd2;
    ssize_t bytes_read, bytes_written;
    char buffer[1024];
    // Open input file "input.txt" for reading
    fd1 = open("input.txt", O_RDONLY);
    if (fd1 == -1) {
        perror("Failed to open input file");
        exit(EXIT_FAILURE);
    }
    // Open output file "output.txt" for writing (create if not exists, truncate if exists)
    fd2 = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (fd2 == -1) {
        perror("Failed to open output file");
        exit(EXIT_FAILURE);
    }
    // Read from input file and write to output file
    while ((bytes_read = read(fd1, buffer, sizeof(buffer))) > 0) {
        bytes_written = write(fd2, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("Write error");
            exit(EXIT_FAILURE);
        }
    }
    if (bytes_read == -1) {
        perror("Read error");
        exit(EXIT_FAILURE);
    }
}
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-1 To Bracket
^X Exit ^R Read File ^N Replace ^U Paste ^J Justify ^Y Go To Line M-E Redo M-6 Copy ^O Where Was

Activities Terminal Feb 20 9:06 AM student@Lab3W: ~/Desktop

```
GNU nano 6.2 input.txt *
```

```
THIS IS OPERATING SYSTEMM Lab - 04
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-1 To Bracket
^X Exit ^R Read File ^N Replace ^U Paste ^J Justify ^Y Go To Line M-E Redo M-6 Copy ^O Where Was

Activities Terminal Feb 20 9:15 AM

```
student@Lab3W:~/Desktop$ nano hello.c
student@Lab3W:~/Desktop$ gcc hello.c -o hello
student@Lab3W:~/Desktop$ nano process.c
student@Lab3W:~/Desktop$ nano hello.c
student@Lab3W:~/Desktop$ ls
hello hello.c process process.o
student@Lab3W:~/Desktop$ rm -r process.o
student@Lab3W:~/Desktop$ ls
hello hello.c process process.o
student@Lab3W:~/Desktop$ gcc process.c -o process
student@Lab3W:~/Desktop$ ./process
Parent process: PID = 4250
Parent process: Waiting for child process to terminate...
Child process: PID = 4251
Child process: PPID = 4250
Child process: Executing ls command...
HELLO BEAUTIFUL PEOPLE <3>
HAPPY RAMADAN QUEENParent process: Child process exited with status 0
student@Lab3W:~/Desktop$ nano file.c
student@Lab3W:~/Desktop$ nano input.txt
student@Lab3W:~/Desktop$ gcc file.c -o file
student@Lab3W:~/Desktop$ ./file
File copied successfully.
```

The screenshot shows a standard Ubuntu desktop environment. On the left, there's a dock with icons for Dash, Home, Help, and others. In the center, a terminal window is open with the title 'Terminal' and the command 'student@Lab3W: ~/Desktop'. The terminal displays a sequence of shell commands and their outputs related to file creation, compilation, and execution. On the right side of the screen, there's a vertical column of icons for 'input.txt', 'hello', 'process', 'output.txt', 'file', 'file.c', 'process.c', 'Home', and 'hello.c'. The desktop background is a purple and red abstract design.

Q1:

Activities Terminal Feb 20 9:29 AM

```
student@Lab3W:~/Desktop$ q1.c
```

This screenshot shows a terminal window with the title 'Terminal' and the command 'student@Lab3W: ~/Desktop'. Inside the terminal, a C program named 'q1.c' is displayed. The code uses the 'fork()' function to create a child process. It then enters a loop where it prints odd numbers from 1 to 10. The parent process continues to print even numbers from 2 to 10. The terminal window has a dark theme and includes a menu bar at the top with options like 'File', 'Edit', 'View', etc., and a bottom bar with various keyboard shortcut keys.

Activities Terminal Feb 20 9:29 AM student@Lab3W: ~/Desktop

GNU nano 6.2 q1.c

```
int main() {
    pid_t child_pid;
    int status;
    child_pid = fork();

    if (child_pid < 0) {
        printf("Unsuccessful Child Process Creation\n");
        exit(EXIT_FAILURE);
    }

    else if (child_pid > 0) {
        wait(&status);
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                printf("Parent prints: %d\n", i);
            }
        }
        printf("Parent Ends\n");
    }

    else {
        printf("Child: Parent ID = %d\n", getppid());
        for (int i = 1; i <= 10; i++) {
            if (i % 2 != 0) {
                printf("Child prints: %d\n", i);
            }
        }
        printf("Child Ends\n");
    }
    return 0;
}
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-1 To Bracket
^X Exit ^R Read File ^H Replace ^U Paste ^J Justify ^V Go To Line M-E Redo M-6 Copy ^O Where Was

Activities Terminal Feb 20 9:31 AM student@Lab3W: ~/Desktop

```
student@Lab3W:~/Desktop$ nano q1.c
student@Lab3W:~/Desktop$ gcc q1.c -o q1
student@Lab3W:~/Desktop$ ./q1
Child: Parent ID = 4609
Child prints: 1
Child prints: 3
Child prints: 5
Child prints: 7
Child prints: 9
Child Prints
Parent prints: 2
Parent prints: 4
Parent prints: 6
Parent prints: 8
Parent prints: 10
Parent Ends
student@Lab3W:~/Desktop$
```

Q2:

The screenshot shows a terminal window titled "student@Lab3W: ~/Desktop". The window title bar also displays the date and time: "Feb 20 9:37 AM". The terminal content is the source code for a C program named "q2.c". The code uses the `fork()` system call to create multiple child processes. It includes error handling for failed forks and prints the PID of each process.

```
GNU nano 6.2
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int main() {
    int n;
    printf("Enter number of child processes to create: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        pid_t pid = fork();

        if (pid < 0) {
            printf("Fork failed\n");
            exit(EXIT_FAILURE);
        }
        else if (pid == 0) {
            printf("Child %d created: PID = %d, Parent PID = %d\n", i + 1, getpid(), getppid());
            exit(0);
        }
        else{
            perror("fork failed");
        }
    }
    printf("Parent process ends. PID = %d\n", getpid());
}

return 0;
}
```

At the bottom of the terminal window, there is a menu bar with various keyboard shortcuts for file operations like Help, Exit, Write Out, Read File, Replace, Cut, Paste, Execute, Location, Go To Line, Undo, Redo, Set Mark, Copy, To Bracket, and Where Was.

The screenshot shows a terminal window titled "student@Lab3W: ~/Desktop". The window title bar also displays the date and time: "Feb 20 9:37 AM". The terminal content shows the execution of the "q2.c" program. The user first compiles the code with "gcc q2.c -o q2", runs it with "./q2", and then enters the number of child processes to create (3). The program outputs the PIDs of the created child processes and the parent process.

```
student@Lab3W:~/Desktop$ nano q1.c
student@Lab3W:~/Desktop$ gcc q1.c -o q1
student@Lab3W:~/Desktop$ ./q1
Child: Parent ID = 4609
Child prints: 1
Child prints: 3
Child prints: 5
Child prints: 7
Child prints: 9
Child Ends
Parent prints: 2
Parent prints: 4
Parent prints: 6
Parent prints: 8
Parent prints: 10
Parent Ends
student@Lab3W:~/Desktop$ nano q2.c
student@Lab3W:~/Desktop$ gcc q2.c -o q2
student@Lab3W:~/Desktop$ ./q2
Enter number of child processes to create: 3
fork failed: Success
Child 1 created: PID = 4669, Parent PID = 4668
fork failed: Success
Child 2 created: PID = 4670, Parent PID = 4668
fork failed: Success
Parent process ends. PID = 4668
Child 3 created: PID = 4671, Parent PID = 4668
student@Lab3W:~/Desktop$
```

Q5a:

A screenshot of a Linux desktop environment. In the top panel, there's an 'Activities' button, a 'Terminal' icon, and a date/time indicator 'Feb 20 9:50 AM'. The main window is a terminal titled 'student@Lab3W: ~/Desktop'. It shows a file named 'q5a.c' being edited in the 'GNU nano 6.2' editor. The code in the file is a C program that demonstrates process creation using fork(). It prints the PID of the current process and creates four child processes. If any fork fails, it prints an error message and exits with status 1. The terminal window has a menu bar with various keyboard shortcuts for file operations like Help, Write Out, Read File, Replace, Cut, Paste, Execute, Location, Undo, Redo, Set Mark, Copy, and Where Was.

```
student@Lab3W:~/Desktop$ cat q5a.c
#include <stdio.h>
#include <stdlib.h>
#include <unstd.h>
#include <sys/wait.h>

int main(){
    pid_t pid5, pid3, pid4;
    printf("I am Process 1 and my PID is %d\n", getpid());
    pid5=fork();
    pid5 = fork();
    if (pid5 < 0) {
        perror("Fork failed");
        exit(1);
    }
    if (pid5 == 0) {
        printf("I am Process 5 and my PID is %d\n", getpid());
        exit(0);
    }
    pid3 = fork();
    if (pid3 < 0) {
        perror("Fork failed");
        exit(1);
    }
    if (pid3 == 0) {
        printf("I am Process 3 and my PID is %d\n", getpid());
        pid4 = fork();
        if (pid4 < 0) {
            perror("Fork failed");
            exit(1);
        }
        if (pid4 == 0) {
```

A screenshot of a Linux desktop environment. In the top panel, there's an 'Activities' button, a 'Terminal' icon, and a date/time indicator 'Feb 20 9:50 AM'. The main window is a terminal titled 'student@Lab3W: ~/Desktop'. It shows a command-line session where the user runs 'gcc q1.c -o q1', then './q1'. The output shows a parent process (PID 4669) printing numbers from 1 to 10, and three child processes each printing a single number. The user then runs 'gcc q2.c -o q2', enters '3' when prompted for the number of child processes, and sees three child processes created with PIDs 4669, 4670, and 4668 respectively. Finally, the user runs 'gcc q5a.c -o q5a', then './q5a', which results in five processes (parent and four children) each printing their own PID. The terminal window has a menu bar with various keyboard shortcuts for file operations like Help, Write Out, Read File, Replace, Cut, Paste, Execute, Location, Undo, Redo, Set Mark, Copy, and Where Was.

```
student@Lab3W:~/Desktop$ gcc q1.c -o q1
student@Lab3W:~/Desktop$ ./q1
Child: Parent ID = 4669
Child prints: 1
Child prints: 3
Child prints: 5
Child prints: 7
Child prints: 9
Child Ends
Parent prints: 2
Parent prints: 4
Parent prints: 6
Parent prints: 8
Parent prints: 10
Parent Ends
student@Lab3W:~/Desktop$ nano q2.c
student@Lab3W:~/Desktop$ gcc q2.c -o q2
student@Lab3W:~/Desktop$ ./q2
Enter number of child processes to create: 3
fork failed: Success
child 1 created: PID = 4669, Parent PID = 4668
fork failed: Success
child 2 created: PID = 4670, Parent PID = 4668
fork failed: Success
Parent process ends. PID = 4668
Child 3 created: PID = 4671, Parent PID = 4668
student@Lab3W:~/Desktop$ nano q5a.c
student@Lab3W:~/Desktop$ gcc q5a.c -o q5a
student@Lab3W:~/Desktop$ ./q5a
I am Process 1 and my PID is 4773
I am Process 5 and my PID is 4775
I am Process 5 and my PID is 4776
I am Process 3 and my PID is 4777
I am Process 3 and my PID is 4778
I am Process 4 and my PID is 4779
I am Process 4 and my PID is 4780
student@Lab3W:~/Desktop$ $
```

Q5b:

A screenshot of a Linux desktop environment. In the top-left corner, there's a dock with icons for various applications like a web browser, file manager, and terminal. The main focus is a terminal window titled "student@Lab3W: ~/Desktop". The terminal shows the following C code:

```
GNU nano 6.2
#include <stdio.h>
#include <stdlib.h>
#include <unstd.h>
#include <sys/wait.h>

int main() {
    pid_t pid9;

    printf("I am Process 6 and my PID is %d\n", getpid());
    pid9 = fork();
    if (pid9 < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid9 == 0) {
        printf("I am Process 9 and my PID is %d\n", getpid());
        exit(0);
    }

    wait(NULL);

    return 0;
}
```

The terminal window has a dark background with light-colored text. The menu bar at the top says "Activities" and "Terminal". The status bar at the bottom shows keyboard shortcuts for various functions.

A screenshot of a Linux desktop environment, continuing from the previous one. The terminal window now displays the output of the C program. It shows the parent process (PID 4668) printing its ID and creating two child processes (PIDs 4669 and 4670). The child processes then print their own IDs. The terminal window has a dark background with light-colored text. The menu bar at the top says "Activities" and "Terminal". The status bar at the bottom shows keyboard shortcuts for various functions.

```
Child prints: 7
Child prints: 9
Child Ends
Parent prints: 2
Parent prints: 4
Parent prints: 6
Parent prints: 8
Parent prints: 10
Parent Ends
student@Lab3W:~/Desktop$ nano q2.c
student@Lab3W:~/Desktop$ gcc q2.c -o q2
student@Lab3W:~/Desktop$ ./q2
Enter number of child processes to create: 3
fork failed: Success
child 1 created: PID = 4669, Parent PID = 4668
fork failed: Success
child 2 created: PID = 4670, Parent PID = 4668
fork failed: Success
Parent process ends. PID = 4668
Child 3 created: PID = 4671, Parent PID = 4668
student@Lab3W:~/Desktop$ nano q2.c
student@Lab3W:~/Desktop$ nano q5a.c
student@Lab3W:~/Desktop$ gcc q5a.c -o q5a
student@Lab3W:~/Desktop$ ./q5a
I am Process 1 and my PID is 4773
I am Process 5 and my PID is 4775
I am Process 5 and my PID is 4776
I am Process 3 and my PID is 4777
I am Process 3 and my PID is 4778
I am Process 4 and my PID is 4779
I am Process 4 and my PID is 4780
student@Lab3W:~/Desktop$ nano q5a.c
student@Lab3W:~/Desktop$ nano q5b.c
student@Lab3W:~/Desktop$ gcc q5b.c -o q5b
student@Lab3W:~/Desktop$ ./q5b
I am Process 6 and my PID is 4827
I am Process 9 and my PID is 4828
student@Lab3W:~/Desktop$
```