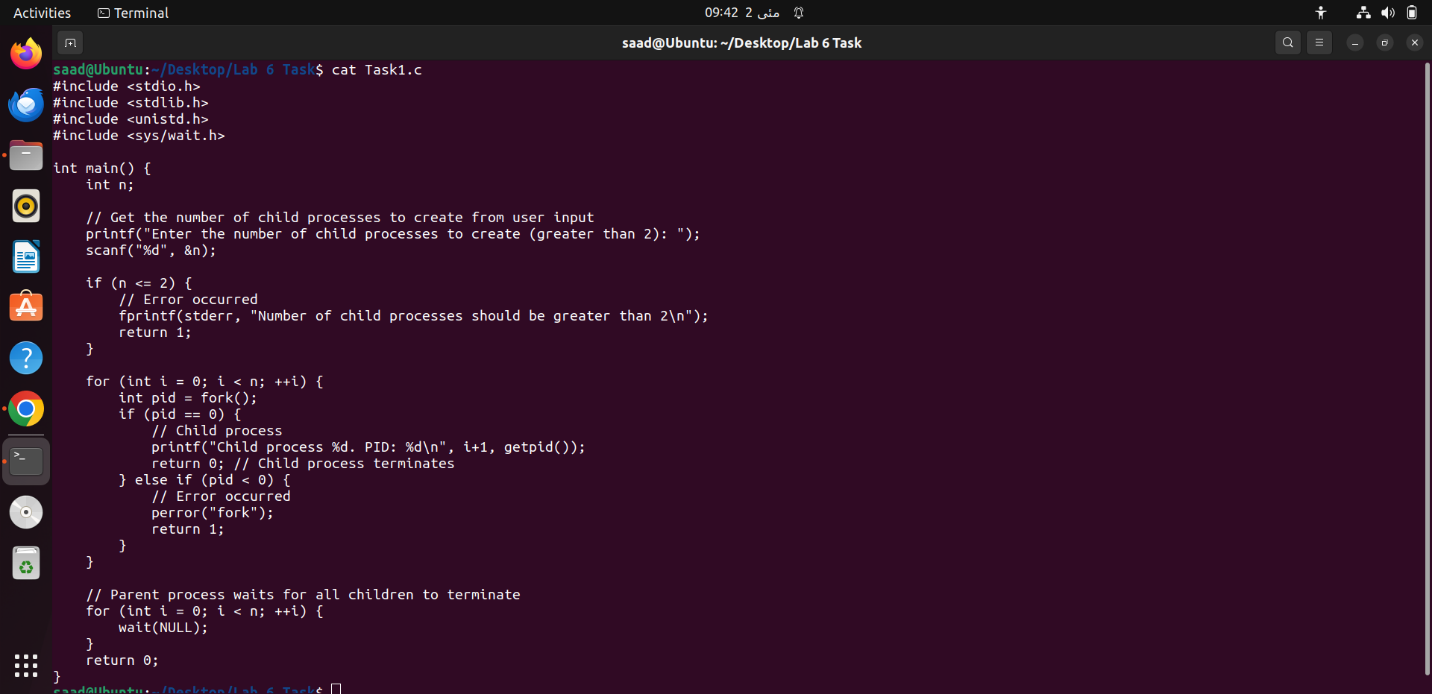
**Operating Systems**

**Lab\_06**

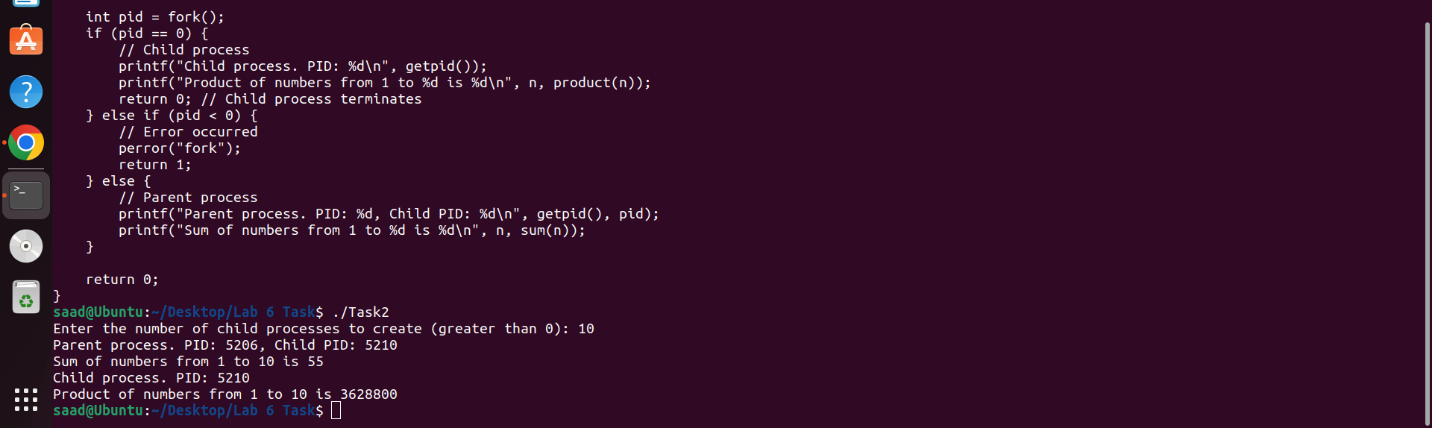
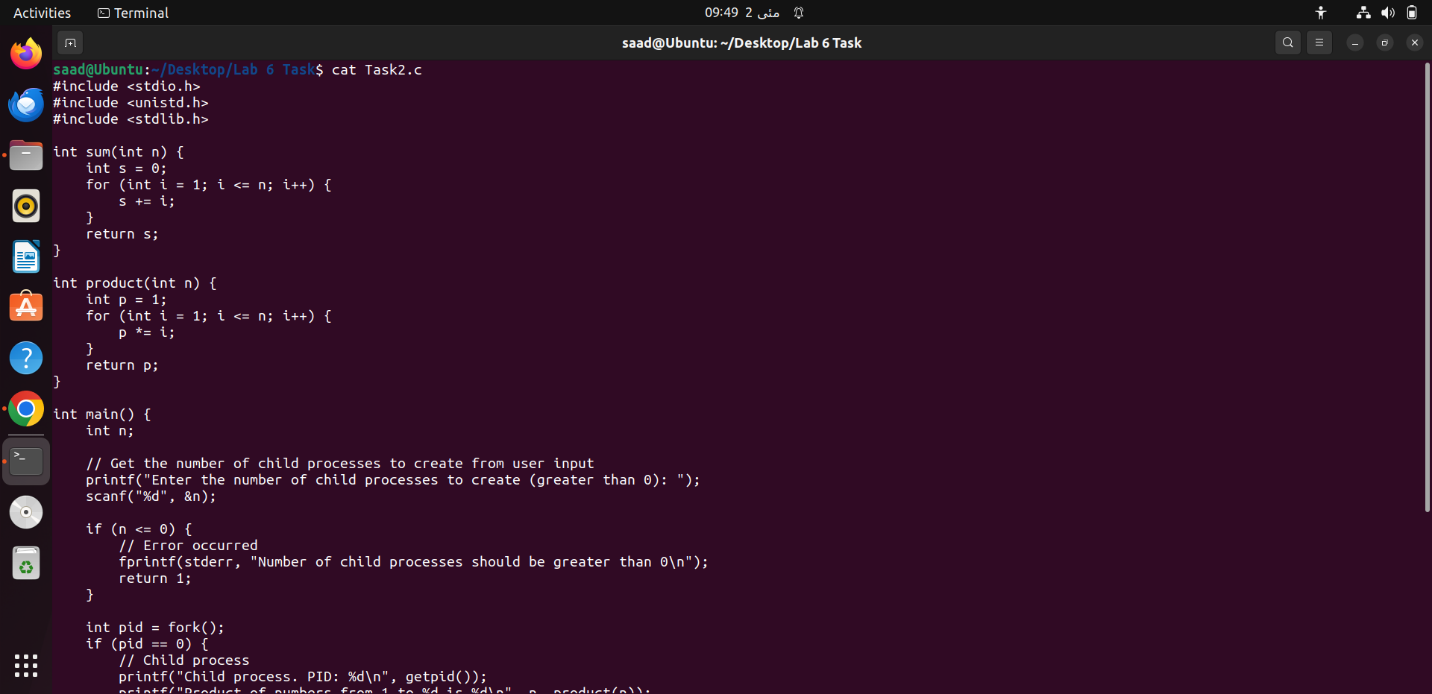
**Roll No. 41**

Task1:



**Description:**  
The above code is a C program that creates a specified number of child processes using the “fork()”system call. The program first prompts the user to input the number of child processes to create, with a minimum value of 3 (since creating 1 or 2 processes would not make sense in this context). The main process then enters a loop where it creates a child process in each iteration. The child process simply prints its process ID and terminates. After creating all the child processes, the main process waits for all of them to finish using the “wait()” system call. This program demonstrates the use of fork() to create multiple processes and wait() to synchronize their execution

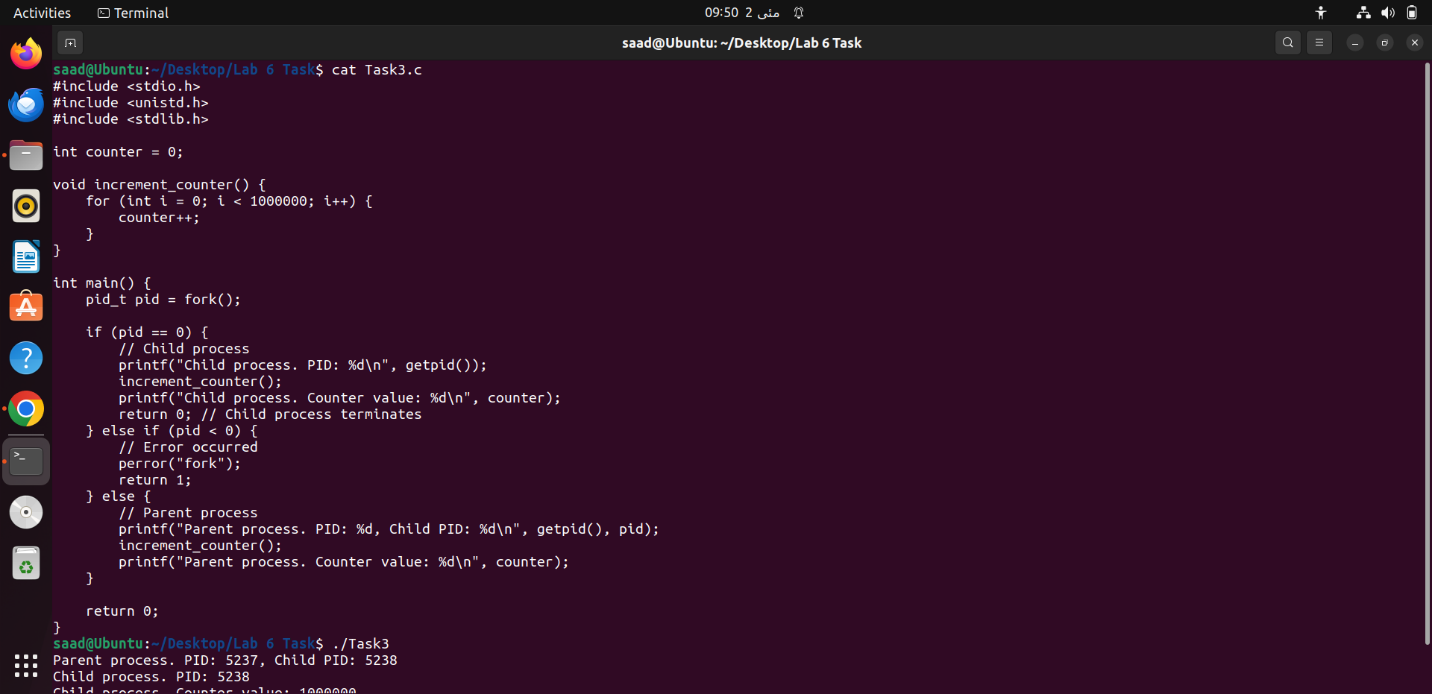
# Task2:



**Description:**

The above code is a C program that creates a single child process using the fork() system call. The program first prompts the user to input the number of child processes to create, with a minimum value of 1. The main process then creates a child process using fork(). The child process calculates and prints the product of numbers from 1 to the input number using the product() function, and then terminates. The parent process calculates and prints the sum of numbers from 1 to the input number using the sum() function. Both the parent and child processes print their respective process IDs. This program demonstrates the use of fork() to create a single child process and how the child process can execute a different set of instructions than the parent process.

# Task3:



Description:  
The above code is a C program that creates a single child process using the fork() system call. The program defines a global variable counter and a function increment\_counter() that increments the counter variable by 1,000,000. The main process creates a child process using fork(). The child process increments the counter variable using increment\_counter() and prints its value. The parent process also increments the counter variable and prints its value. Since the child and parent processes have separate address spaces, they have their own copies of the counter variable. Therefore, the counter value printed by the child process will be different from the counter value printed by the parent process. This program demonstrates the use of fork() to create a single child process and how the child process can have its own copy of global variables.