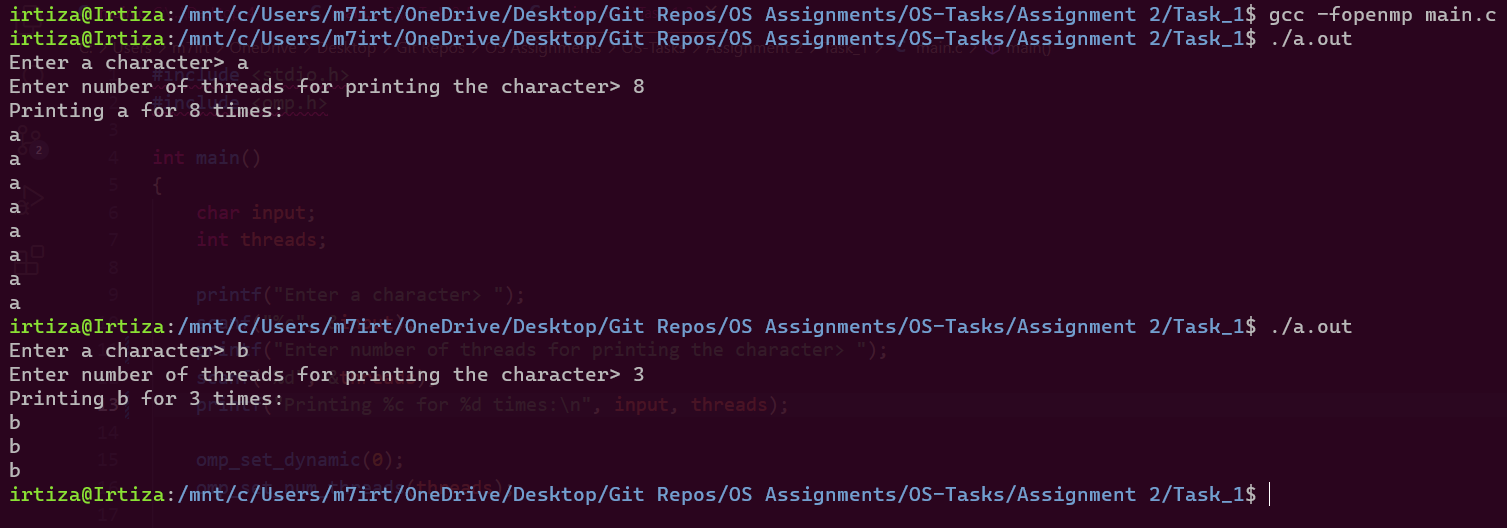
**Question # 1:**

**Method:**

The program takes a character and the number of threads from the user. I have used OpenMp to do multithreading. I set the variable “omp\_set\_dynamic” to 0 and set the variable “omp\_set\_num\_threads” to the required number of threads. In the end I added the “printf” instruction to the “parallel” construct of OpenMP.

**Output:**

****

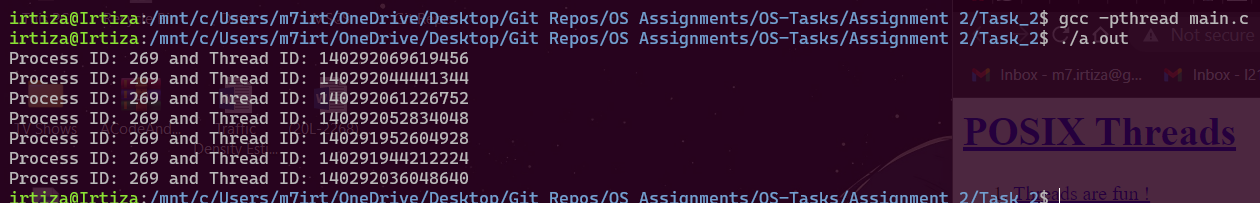
**Question # 2:**

**Method:**

I created an array of type “pthread\_t” and size equal to 7. A for-loop is used to create 7 threads and the function passed to the thread, to be executed, creates a struct of type INFORMATION. This struct holds pid and tid which are initialized from getpid() and pthread\_self(). This struct is returned back to main thread and is received by pthread\_join().

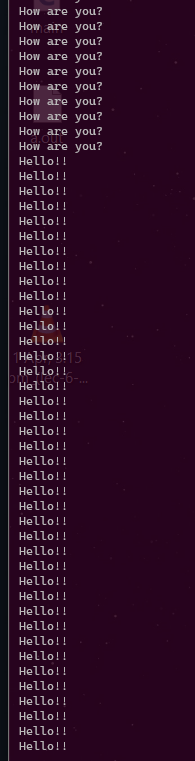
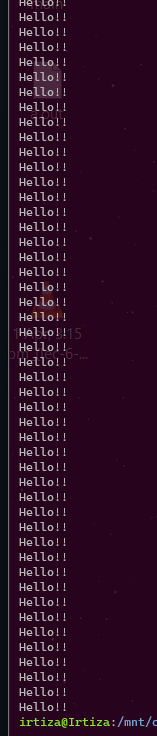
|  |  |
| --- | --- |
| **Struct** | **Thread Handler Function** |
|  |  |

**Output:**

****

**Question # 3:**

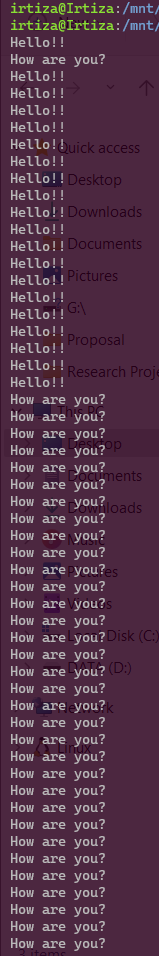
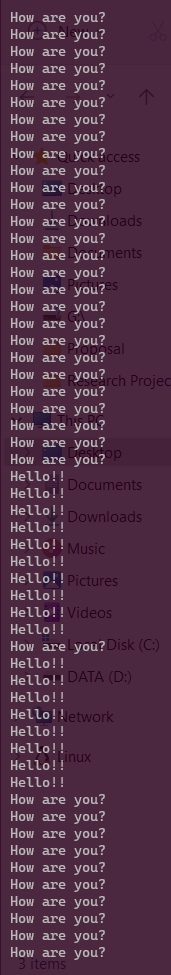
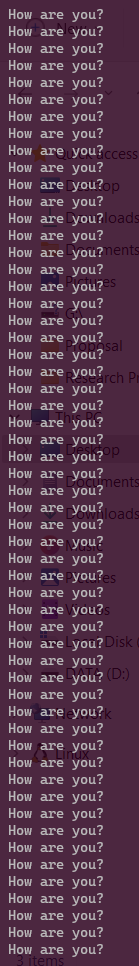
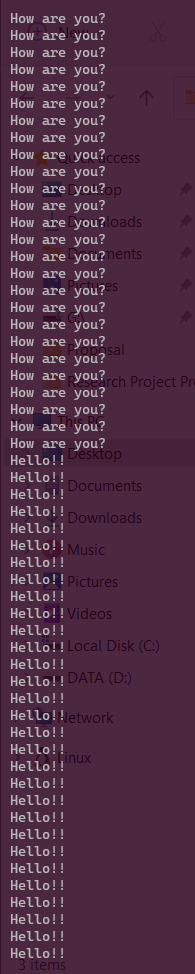
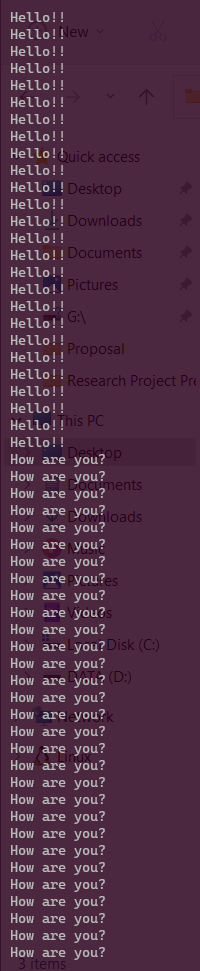
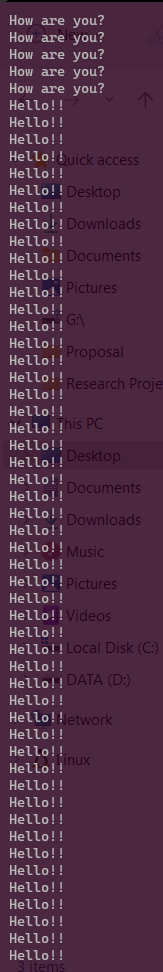
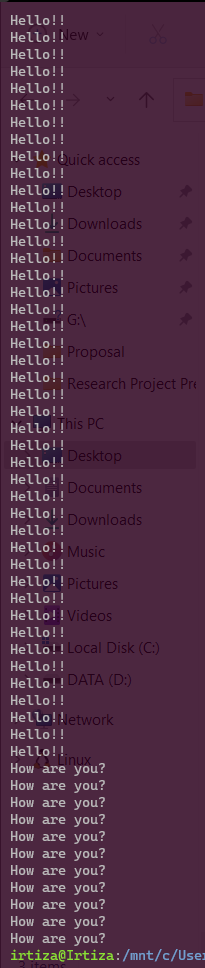
**Part A**

**Explanation:**

The output of the program is not synchronized. We can see that the **thread1()** was executed for 7 times and printed “Hello!!” but the system then scheduled the **thread2()** to run which printed “How are you?” for some number of times. Then again **thread1()** was executed in the end.

**Part B**

**      **

**Explanation:**

As the output shows the scheduling of 4 threads was not uniform. Some data from a thread was printed then it was context switched and the data from another thread was printed.

**Question # 4:**

**Results for Sequential Code**

|  |  |  |
| --- | --- | --- |
| Run 1 | 8.103 |  |
| Run 2 | 8.119 |
| Run 3 | 8.178 |
| Average Time Taken | 8.133 seconds |

**Results for Parallel Code**

|  |  |  |  |
| --- | --- | --- | --- |
| Threads = 2 | Run 1 | 7.487 |  |
| Run 2 | 7.510 |
| Run 3 | 7.472 |
| Average Time Taken | 7.489 seconds |
| Threads = 3 | Run 1 | 7.579 |  |
| Run 2 | 7.385 |
| Run 3 | 8.037 |
| Average Time Taken | 7.667 seconds |
| Threads = 4 | Run 1 | 6.955 |  |
| Run 2 | 7.202 |
| Run 3 | 6.841 |
| Average Time Taken | 6.999 seconds |

**Question # 5:**

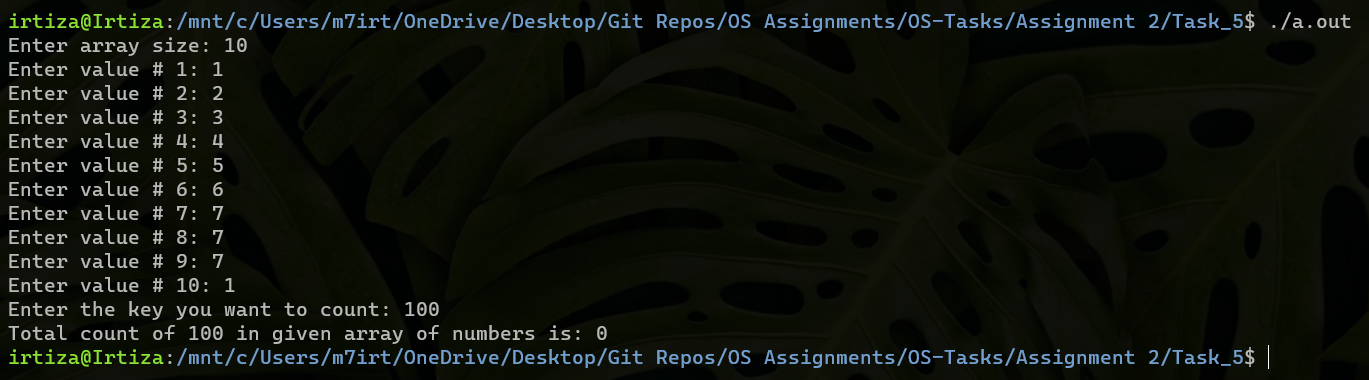
**Method:**

The program takes size of the array and the values to be inserted in the array of numbers. Then the key, that is to be counted from the given array, is taken. The program uses following OpenMP directive to apply linear search with multiple threads:

* parallel
* for
* schedule(static)
* reduction

**Output:**

****

****