Operating System
Assignment 4

by Mohammad Abu Musa Rabiul
ID: 220201072

Table Of Contents

Purpose

the purpose of the project is the Comparison of performance of multi-threaded environment and non-multi-threaded environment. Program is built on Linux operating system.

Sequential case

The program begins by generating tasks. A task is either an insert, a delete, or a search a value in a sorted list (in ascending order) with no duplicates. Your program assigns a task_num starting from 0 and incremented by 1 for each task generation, a task_type which is insert, delete, or search task in the list, and a value for the list operation, randomly. Each time it generates a new task, it adds it to a task queue. After the completion of task generation, your program pulls the tasks from the queue and processes them one by one, by inserting, deleting, or searching from the list.

Multi threaded environment case

The main thread begins by starting a user-specified number of threads that immediately go to sleep in a condition wait (the worker threads are idle at first).
The main thread generates tasks to be carried out by the other threads. A task is either an insert, a delete, or a search a value in a sorted list (in ascending order) with no duplicates. Each time the main thread generates a new task by adding it to a task queue, it awakens a thread with a condition
signal. When a thread is awakened, it pulls a task from the queue and processes it. When a thread finishes executing its task, it should return to a condition wait. When the main thread completes generating tasks, it sets a global variable indicating that there will be no more tasks, and awakens all the threads with a condition broadcast.

The program will be implemented using Linux pthreads, mutexes and conditional variable.

Program implementation

Sequential implementation

In sequential implementation, main function in “run.c” file
first checks the user input and the calls **manageTasks** function. The **manageTasks** function takes an integer as argument. Inside the **managerTasks** function, **Task_enqueue** function is called and it generates task queue of size n. Then program dequeue task one by one using **Task_dequeue** function inside a while loop. The objective of **task_dequeue** function is to take a task from the task queue and process that task by calling **processTask** function. The **processTask** function process the task operation which is either search, insert, or delete operation on data linked list. Upon returning from the **processTask** function, the task is deallocated. At the end, the

manageTask function deallocated the data linked list to guard against possible memory leak.
In sequential case, there is no concept of multi-threading.

Multi-threaded implementation

To implement multi threaded program, Linux pthreads are used along with the main thread of the program. User will first define the thread size and according to that threads will be created. The main thread generates a task and other threads process that task and task is dequeued from the task queue after finishing execution.

During the implementation of the program, mutex is used to handle synchronization problem. As **task queue** is a critical section of the program, it is important to maintain the data consistency of the program so that program does not encounter "race condition".

The program also use conditional variable to generate proper signal to other threads when there is a task available to process generated by main thread.

The program pseudo code

```
task_count=0; // keep track of number of generated tasks
available= false; // task is available or not
exit_status= false; // check when no task to generate

main_thread(){
   while(task_count != input_task_size){ //loop until the size is
                                         //equivalent to input task size
         lock();
         else if (! available){ // increment when no task available
            enqueue(); // add task in the queue
            available = true;
            task_count ++;
         }
          else if (available){ // if available (trigger condition) then
                               //call signal
            signal();
         }
         unlock ();
   }
while(1){
//check if still any available task.
//if exist then send signal until other thread process that task.
         lock(); //lock
         if (available){
             signal();
             unlock(); //unlock
```

```
        }

        if (!available) {

            broadcast();
            exit_status =true;
            unlock();
            break;

        }


    }
}

Other_thread (){
while (true){   // each thread tries infinitely until gets a broatcast
signal from main thread.
    lock ();
    while (!available && !exit_status){
        wait();
    }
    if (exit_status){
        unlock();
        thread_exit();
    }
    dequeue(); // dequeue task queue and process task
    available= false;
    unlock();
}
```

task_count variable keep track of number of tasks generated by main thread. When a task is generated, main thread makes **available** variable true. When **available** is true, other thread call **Task_dequeue** function. When there is no task to generate, **exit_status** variable set to true and **broadcast signal** is called by main thread. Other threads in the system get that **broadcast signal** and finish execution.

All child threads of main thread use **`task_thread(void * param)`** function.

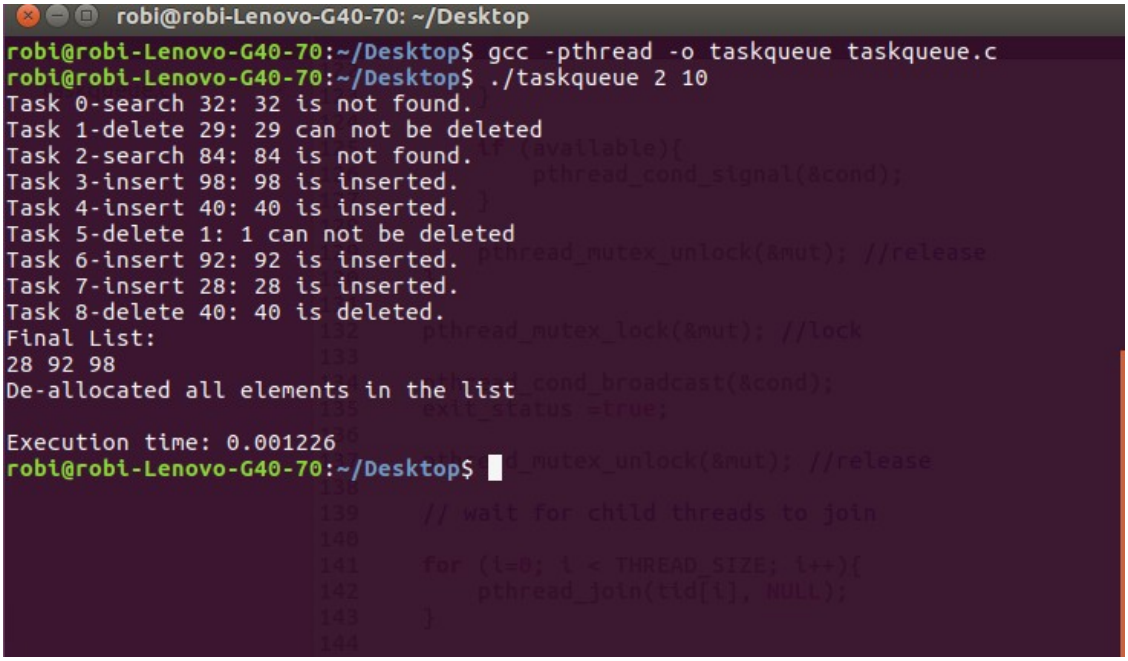
Thread function

```
void * task_thread(void * param){
     while (true) {    // each thread tries infinitely until gets a
broatcast signal from main thread.
        pthread_mutex_lock(&mut); //lock
        while (!available && !exit_status) {
            pthread_cond_wait(&cond, &mut);
        }
```

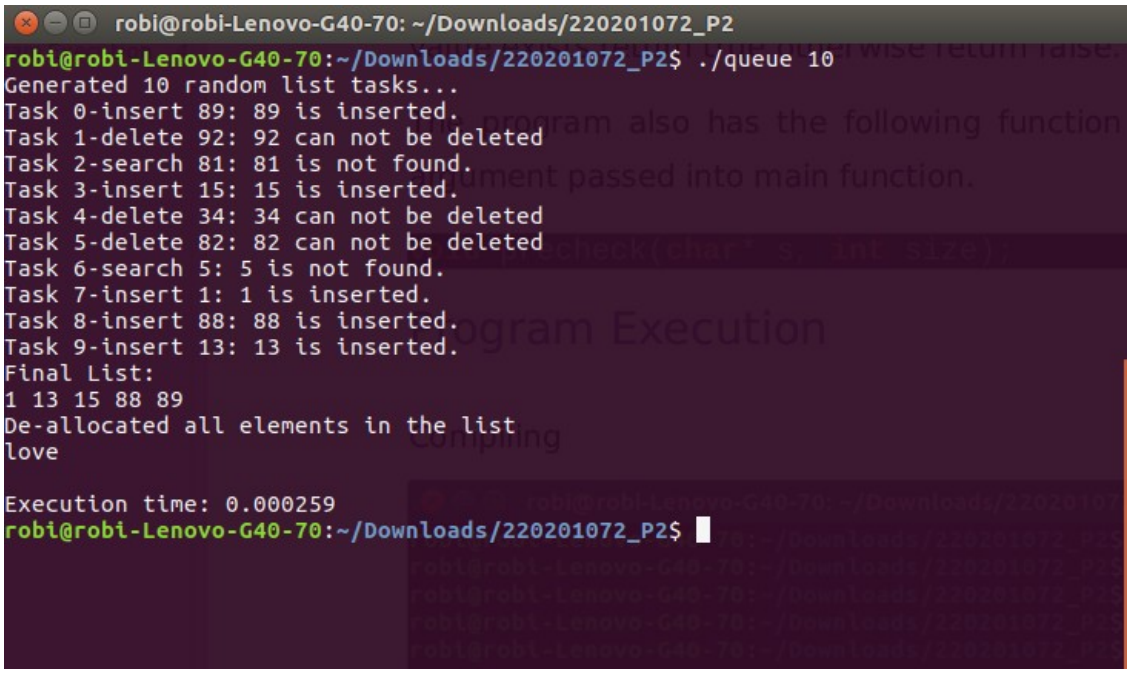
```
        if (exit_status) {
            pthread_mutex_unlock(&mut);
            pthread_exit(NULL);
        }
        Task_dequeue();
        available = false;
        pthread_mutex_unlock(&mut);
    }
}
```

Sample Executions

Multi threaded case



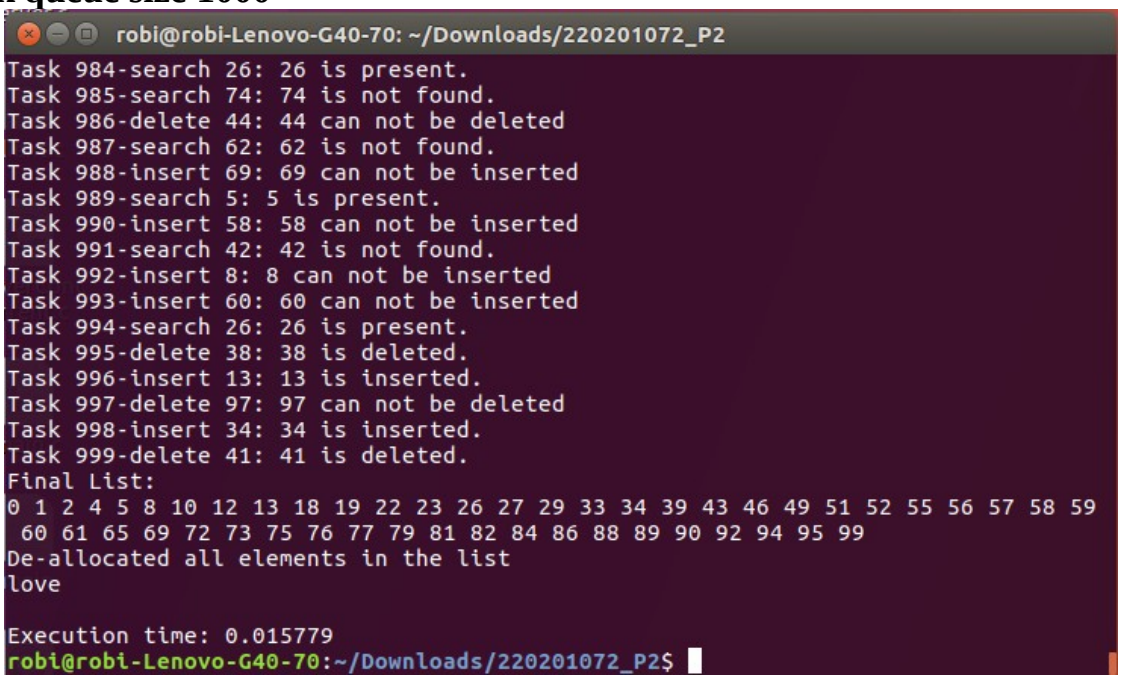
Sequential case



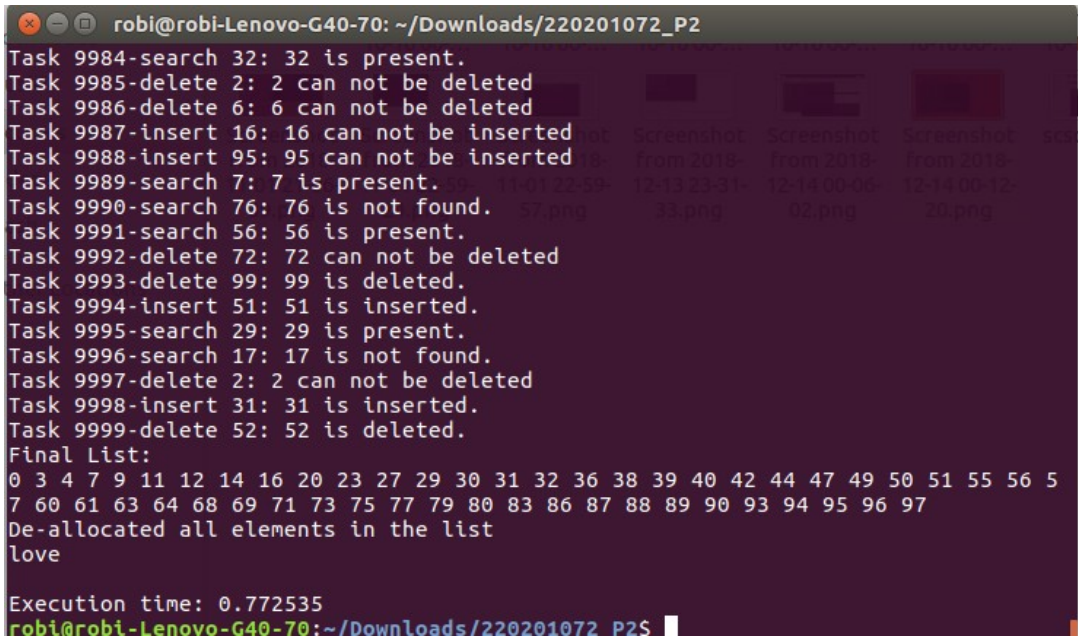
Experiment

Sequential

- Task queue size 1000



- Task queue size 10000



- Task queue size 100000

```
robi@robi-Lenovo-G40-70: ~/Downloads/220201072_P2
Task 99984-delete 84: 84 can not be deleted
Task 99985-insert 91: 91 is inserted.
Task 99986-search 98: 98 is not found.
Task 99987-delete 18: 18 is deleted.
Task 99988-search 69: 69 is not found.
Task 99989-delete 59: 59 can not be deleted
Task 99990-insert 67: 67 is inserted.
Task 99991-delete 88: 88 can not be deleted
Task 99992-search 91: 91 is present.
Task 99993-delete 84: 84 can not be deleted
Task 99994-search 63: 63 is present.
Task 99995-insert 88: 88 is inserted.
Task 99996-insert 16: 16 can not be inserted
Task 99997-insert 18: 18 is inserted.
Task 99998-delete 58: 58 can not be deleted
Task 99999-insert 19: 19 is inserted.
Final List:
0 1 4 6 8 10 11 15 16 18 19 23 26 28 29 31 34 37 38 40 43 44 46 47 49 50 51 53 5
7 61 62 63 67 71 72 74 75 76 78 79 81 85 86 88 89 91 92 95 96 99
De-allocated all elements in the list
love

Execution time: 73.573310
robi@robi-Lenovo-G40-70:~/Downloads/220201072_P2$
```

Multi threaded

<u>Thread 1 Case</u>
- Task queue size 1000

```
Task 998-delete 38: 38 can not be deleted
Task 999-insert 4: 4 can not be inserted
Final List:
0 1 4 5 6 10 11 13 15 20 23 25 26 27 29 31 34 41 45 47 48 51 53 55 56 57 63 64 6
7 68 69 71 75 76 78 84 85 86 88 92 93 95 98 99
De-allocated all elements in the list

Execution time: 0.032689
robi@robi-Lenovo-G40-70:~/Desktop$
```

-Task queue size 10000

```
Task 9997-delete 88: 88 can not be deleted
Task 9998-delete 77: 77 is deleted.
Task 9999-search 51: 51 is not found.
Final List:
1 4 6 7 9 12 13 18 20 22 24 26 29 30 31 32 33 34 35 36 41 42 43 50 52 53 54 55 5
9 60 62 65 66 67 72 73 76 78 82 83 85 86 89 91 97 98 99
De-allocated all elements in the list

Execution time: 0.270050
robi@robi-Lenovo-G40-70:~/Desktop$
```

-Task queue size 100000

```
Task 99997-delete 96: 96 can not be deleted
Task 99998-search 68: 68 is present.
Task 99999-delete 22: 22 can not be deleted
Final List:
0 1 4 5 6 8 11 13 15 16 18 20 21 24 27 29 30 32 33 35 38 39 40 43 46 47 50 51 53
 56 57 60 61 65 68 69 71 74 75 76 77 81 85 86 87 90 92 97 98
De-allocated all elements in the list

Execution time: 2.555347
robi@robi-Lenovo-G40-70:~/Desktop$
```

Thread 2 Case

- Task queue size 1000

```
Task 998-delete 24: 24 is deleted.
Task 999-search 63: 63 is not found.
Final List:
1 2 3 5 9 10 13 18 23 25 26 27 28 36 38 39 40 42 44 46 53 55 60 61 62 64 65 67 7
1 73 75 76 78 81 83 85 89 93 94 97 99
De-allocated all elements in the list

Execution time: 0.029798
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 10000

```
Task 9998-delete 27: 27 is deleted.
Task 9999-insert 23: 23 can not be inserted
Final List:
1 3 5 6 7 9 10 12 16 18 19 20 23 29 30 31 34 36 37 39 41 42 43 44 45 48 49 51 53
 55 57 58 61 62 63 65 66 67 68 72 75 76 79 85 88 89 92 96 98
De-allocated all elements in the list

Execution time: 0.264053
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 100000

```
Task 99998-search 98: 98 is present.
Task 99999-delete 24: 24 is deleted.
Final List:
0 2 3 5 6 7 8 11 12 13 16 17 18 19 20 27 28 29 31 32 34 41 44 46 48 49 50 51 52
53 55 59 60 65 75 77 78 80 81 82 83 94 95 98 99
De-allocated all elements in the list

Execution time: 2.536527
robi@robi-Lenovo-G40-70:~/Desktop$
```

Thread 4 Case

- Task queue size 1000

```
Task 998-insert 1: 1 can not be inserted
Task 999-delete 95: 95 can not be deleted
Final List:
1 4 6 8 10 11 12 13 14 15 18 19 21 25 26 29 30 31 32 33 34 35 37 38 39 43 44 46
49 52 53 54 55 56 57 59 60 61 62 63 65 66 67 68 69 70 71 73 74 78 79 80 81 82 85
 86 87 89 90 91 92 93 98 99
De-allocated all elements in the list

Execution time: 0.042885
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 10000

```
Task 9998-search 41: 41 is not found.
Task 9999-search 62: 62 is present.
Final List:
4 8 9 10 11 13 16 19 27 28 29 30 31 32 35 36 39 40 43 47 49 51 58 59 61 62 63 64
 67 68 69 71 73 75 78 81 82 83 86 87 89 91 94 95 96
De-allocated all elements in the list

Execution time: 0.417577
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 100000

```
Task 99998-insert 90: 90 is inserted.
Task 99999-search 20: 20 is not found.
Final List:
0 1 2 6 12 17 18 22 25 26 29 32 33 36 41 43 45 46 47 53 54 57 58 62 64 65 67 68
69 72 74 75 76 79 80 83 84 85 86 88 89 90 91 93 94 95 96 98
De-allocated all elements in the list

Execution time: 3.963512
robi@robi-Lenovo-G40-70:~/Desktop$
```

Thread 8 Case

- Task queue size 1000

```
Task 998-insert 83: 83 can not be inserted
Task 999-search 38: 38 is present.
Final List:
0 2 3 5 6 10 11 13 14 15 19 22 23 24 27 29 32 34 35 38 42 43 44 48 49 51 54 57 6
0 65 66 73 74 75 76 79 83 88 93 96 97 98
De-allocated all elements in the list

Execution time: 0.055416
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 10000

```
Task 9997-delete 8: 8 is deleted.
Task 9998-search 2: 2 is present.
Task 9999-search 32: 32 is present.
Final List:
0 2 4 5 6 7 14 15 16 18 20 21 22 28 29 31 32 33 34 37 38 39 40 41 44 47 48 50 51
 52 53 54 55 56 57 62 63 66 67 68 69 70 73 74 75 76 78 79 81 82 83 84 86 87 88 8
9 90 92 97 99
De-allocated all elements in the list

Execution time: 0.445301
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 100000

```
Task 99998-delete 87: 87 can not be deleted
Task 99999-search 54: 54 is present.
Final List:
0 1 2 3 4 7 8 9 12 13 14 16 19 23 24 25 32 34 35 37 40 41 46 49 52 54 55 57 58 6
2 64 65 67 69 71 72 73 77 80 81 82 86 91 92 93 95 96 98
De-allocated all elements in the list

Execution time: 4.382528
robi@robi-Lenovo-G40-70:~/Desktop$
```

Thread 16 Case

- Task queue size 1000

```
Task 997-search 14: 14 is not found.
Task 998-insert 92: 92 is inserted.
Task 999-insert 16: 16 is inserted.
Final List:
6 7 10 15 16 18 19 21 22 23 25 27 29 30 31 34 36 37 38 40 44 48 50 51 53 55 56 5
7 58 60 61 62 64 65 67 68 71 72 77 78 79 80 82 83 87 88 92 93 94 95 98
De-allocated all elements in the list

Execution time: 0.057484
robi@robi-Lenovo-G40-70:~/Desktop$
```

- Task queue size 10000

```
Task 9998-insert 88: 88 is inserted.
Task 9999-insert 19: 19 is inserted.
Final List:
1 3 4 5 7 8 9 10 15 17 19 20 21 26 27 35 36 37 39 40 41 45 46 48 49 53 54 55 60
63 64 65 66 67 68 73 74 78 81 82 83 84 87 88 89 92 96
De-allocated all elements in the list

Execution time: 0.519508
robi@robi-Lenovo-G40-70:~/Desktop$
```

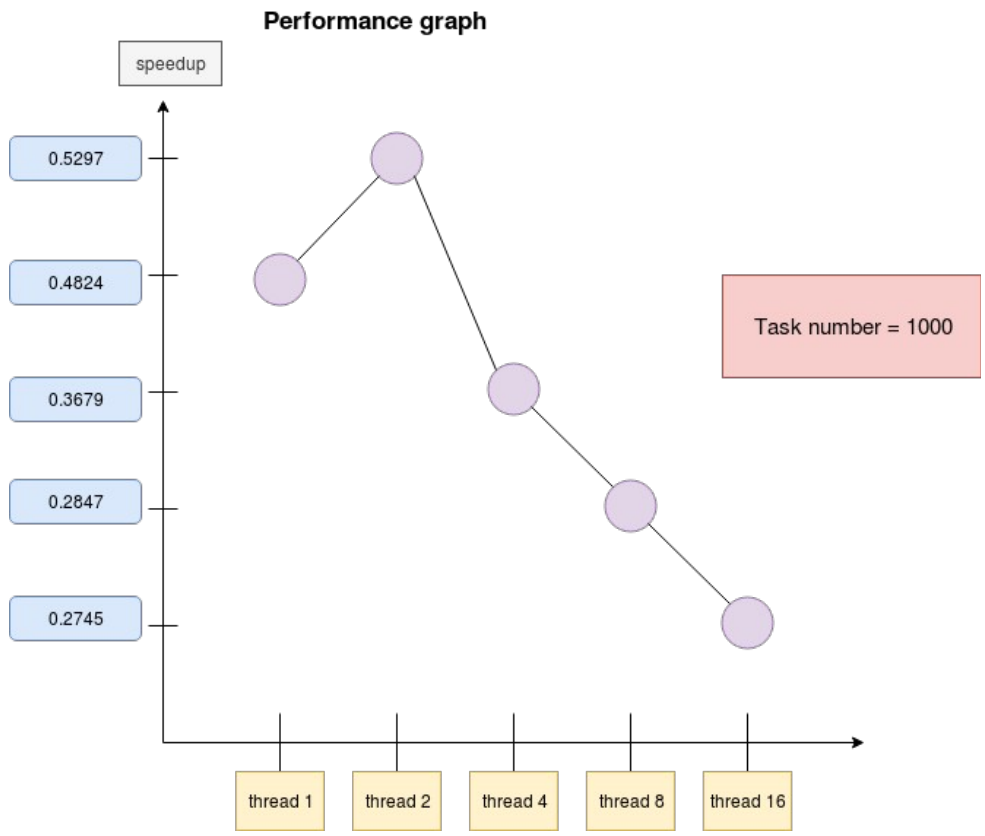
- Task queue size 100000

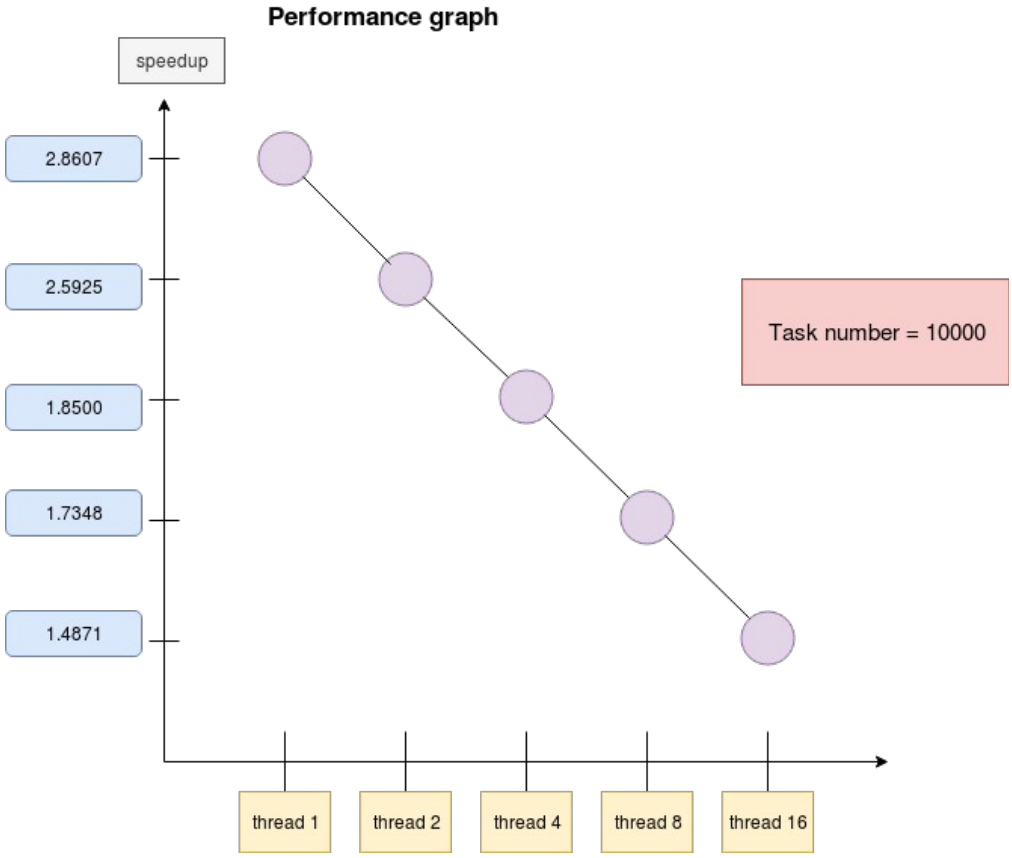
```
Task 99998-search 99: 99 is present.
Task 99999-insert 50: 50 is inserted.
Final List:
4 5 7 8 9 10 15 18 19 21 22 23 26 28 29 32 33 37 40 42 43 44 45 48 49 50 54 55 5
8 60 63 64 67 71 73 75 76 79 81 86 87 88 89 90 91 93 95 97 98 99
De-allocated all elements in the list

Execution time: 4.890851
robi@robi-Lenovo-G40-70:~/Desktop$
```

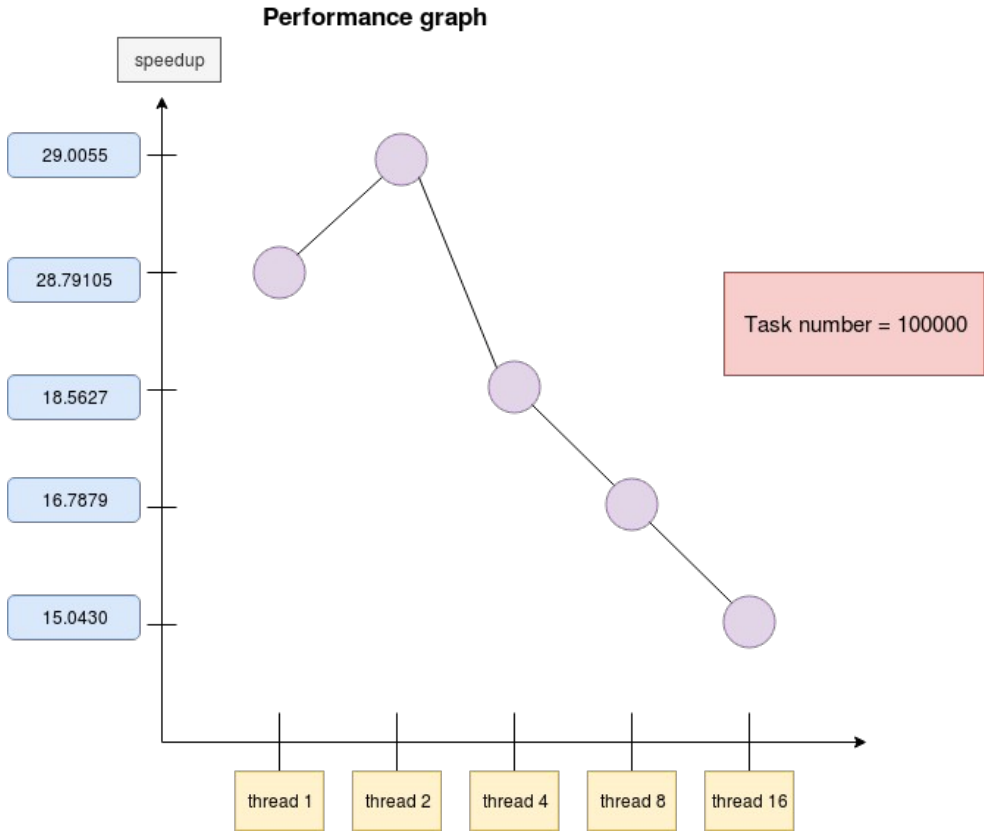
Performance measurement

To evaluate the performance of your parallel implementation, first speedup o the program is measured respect to number of threads and the formula is used to calculate the speedup is $\mathbf{S = T_s / T_p}$ where $\mathbf{T_s}$ is the execution time of the sequential version, $\mathbf{T_p}$ is the execution time of the multi-threaded implementation. Using the formula above we get following performance graphs.

Performance graph




Performance graph
speedup
2.8607
2.5925
1.8500
1.7348
1.4871
Task number = 10000
thread 1
thread 2
thread 4
thread 8
thread 16


Performance graph
speedup
29.0055
28.79105
18.5627
16.7879
15.0430
Task number = 100000
thread 1
thread 2
thread 4
thread 8
thread 16

Observation

According to execution time of the programs and performance of implementation graph it is clear that whenever number of queue size is increased total execution time of the program also increases.

It is observed from the graph that using two thread we get the most speedup when queue size is 1000 and 100000. It is important to notice that speedup decreases progressively when more that two thread is used. This is because child threads wait for main thread to create a task ,so whenever a new task is created one of threads wakes up, take the task has been created and do one of operations insertion, deletion and checking. So we limit the parallelism of threads. Therefore incrementing thread size will increase the execution time , as thread will wait in the waiting queue to get signal from main thread.

Finally, In this program using two thread will increase the overall speedup performance of the program.