Report: Thread Pool Implementation – Assignment 2

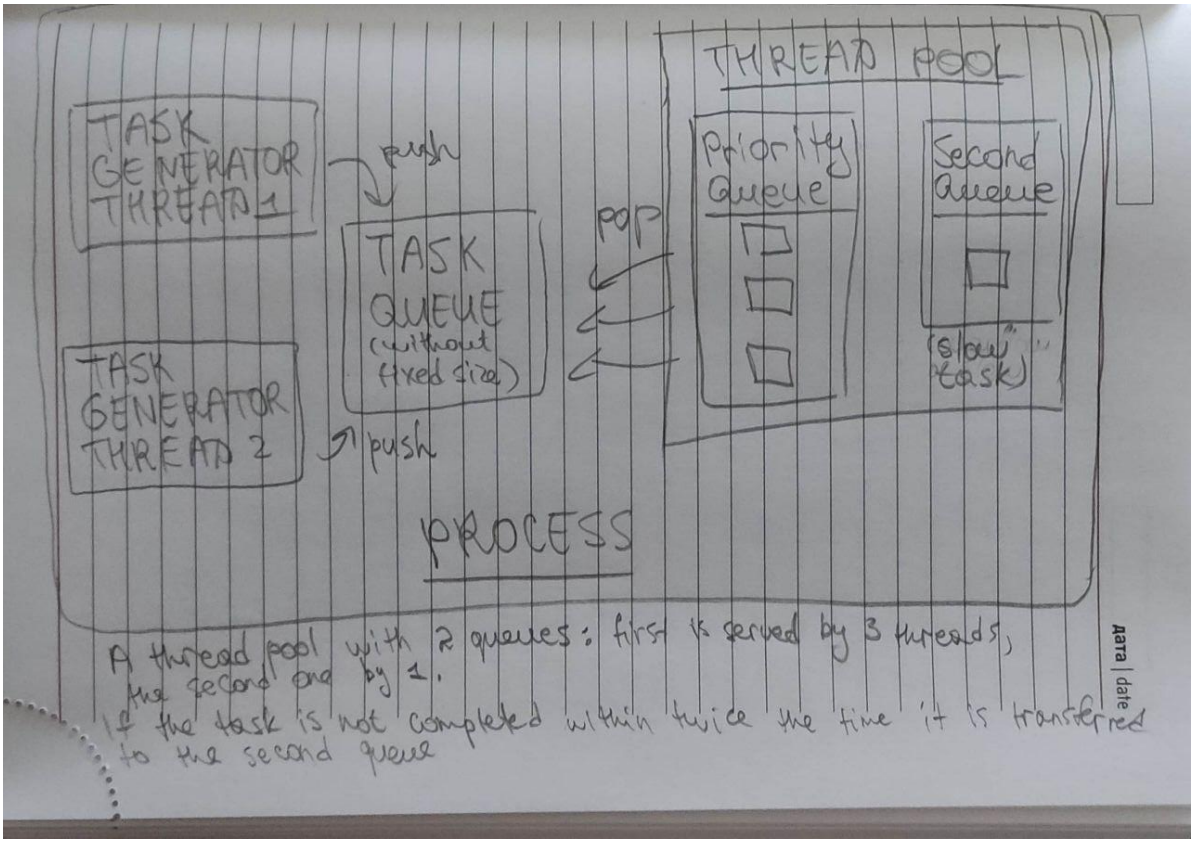
Student Name: Olesia Mykhailyshyn
Group: 1
Chosen Variant: 10
Github Link: https://github.com/VY-Assignments/parallel-assignment-2-olesia-mykhailyshyn.git

Task

Implement a program in which its parts will be synchronized. which includes the implementation of a Thread pool in order not to create threads, then they perform work, are deleted and other new threads are created, and so on in a circle. And here the work was to reduce the load on the central processor using the condition variable and mutex.

System Model

Insert a photo or screenshot of the system model here:



Solution Description

Provide a description of the implemented software solution, explaining how the system was designed and developed:.*)*

The thread pool implementation in this code consists of a General task queue (tasksQueue):
A queue to which new tasks are first added in FIFO order.
A priority queue (firstPriorityQueue):
A second queue (secondQueue):
Used exclusively for slow tasks that have been running too long in the priority queue.

Two threads (TaskGenerator) create tasks and add them to the tasksQueue.
A separate thread that monitors the tasksQueue and passes the tasks to the firstPriorityQueue.
First-queue workers Process tasks from the firstPriorityQueue.
Second-queue workers Process only "slow tasks" from the secondQueue.

Synchronization techniques
a. Mutexes (std::mutex)
Provide exclusive access to shared resources:
b. Condition variables (std::condition_variable)
Used to communicate between threads, allowing them to efficiently wait and send signals
c. Atomic variables (std::atomic)
Manage global states of threads without using locks:
stop: Signals threads to stop processing.
immediateStop: Allows all operations to be stopped immediately.
paused: Pauses task generation if necessary.
activeThreads: Tracks the number of active threads.

Testing

- **Describe how the testing was conducted:** *(Explain how you tested the thread pool's functionality, including task addition, execution, and safe shutdown. Mention any specific tests you performed to ensure thread safety and the correct handling of synchronization primitives.) It was just debugging and following the consol output.*

- **Performance Evaluation:** *(Perform time-limited testing to monitor and record the number of threads created and the average time a thread spends in the waiting state. Evaluate the performance of your thread pool by determining the average length of each queue and the average task execution time for unlimited queues, or ascertain the maximum and minimum times until a queue is filled and the number of rejected tasks for limited queues*

  Metrics such as wait time, execution time, and number of jobs processed are tracked using variables (totalWaitTimeFirstQueue, totalExecutionTimeFirstQueue, etc.). Metrics are calculated using timestamps. Results include: Average wait and execution time for each queue. Total number of jobs processed in each queue.

Insert screenshots of the program during testing: *(Show the console input/output that verifies the thread pool's behavior under various scenarios, like loading tasks, pausing, and stopping the pool.) Pausing and stopping threads*

Paused:
Task generators stop running if the system temporarily pauses processing.
Threads resume when paused changes to false.
Stop and immediateStop:
A soft stop allows you to complete all tasks that are in the queue.
An immediate stop stops all operations immediately.

The pause mechanism allows you to temporarily stop task generation without terminating the thread.
The code uses the paused flag and the taskQueueCV conditional variable in the TaskGenerator function. Here's how it works:
```
if (paused) {
std::cout << "Generator paused by thread " << threadId << std::endl;
taskQueueCV.wait(lock, [] { return !paused || immediateStop; });
if (immediateStop) {
return; // Exit the thread if paused.
}
}
```
If paused == true, task generation is stopped.
The thread waits for the paused flag to change state via taskQueueCV.
When paused == false, the thread resumes.

The pool is resumed by setting paused = false and notifying all threads via taskQueueCV:

paused = false;
taskQueueCV.notify_all(); // Resume all threads.
Things to consider:
Before resuming work, you need to make sure that the system is ready to accept tasks (for example, not all queues are full).
Threads waiting in wait will immediately continue their work.

Smooth stop
The stop flag is used to terminate the thread pool. This allows you to:

Wait for all active threads to complete their work.
Make sure that all tasks remaining in the queues have been completed.
In the code:
stop = true;
taskQueueCV.notify_all(); // Notify threads that the queues will no longer be populated.
firstPriorityQueueCV.notify_all(); // Notify workers.
secondQueueCV.notify_all(); // Notify second queue.
Immediate stop
To stop work immediately, set immediateStop = true. All threads check this condition and terminate immediately:
if (immediateStop) {
return; // The thread terminates execution immediately.

1 task with 50%:

```
Task 1 generated by thread 1 with duration 10
Task 1 transferred from tasksQueue to firstPriorityQueue
Task 1 added to firstPriorityQueue
Maximum number of tasks generated
Worker 1 took task 1 from firstPriorityQueue
Maximum number of tasks generated
Executing task 1 that took 10 seconds
Worker 1 successfully completed task 1 in firstPriorityQueue


=== Metrics ===
First Queue - Average wait time: 0.0034182 seconds
First Queue - Average execution time: 10.0012 seconds
First Queue - Total tasks processed: 1
All tasks are completed.
```

```
Task 1 generated by thread 1 with duration 10
Maximum number of tasks generated
Task 1 transferred from tasksQueue to firstPriorityQueue
Task 1 added to firstPriorityQueue
Worker 1 took task 1 from firstPriorityQueue
Maximum number of tasks generated
Executing task 1 that took 30 seconds
Worker 1 moved task 1 to secondQueue


=== Metrics ===
First Queue - Average wait time: 0.0045734 seconds
First Queue - Average execution time: 30.0099 seconds
First Queue - Total tasks processed: 1
All tasks are completed.
```

5 tasks with 40%:

```
"C:\KSE\Parallel and Client-Server Programming\parallel-assignment-2-c
Task 1 generated by thread 1 with duration 10
Task 2 generated by thread 2 with duration 10
Task 1 transferred from tasksQueue to firstPriorityQueue
Task 1 added to firstPriorityQueue
Task Worker 1 took task 1 from firstPriorityQueue
2 transferred from tasksQueue to firstPriorityQueue
Task 2 added to firstPriorityQueue
Worker 2 took task 2 from firstPriorityQueue
Task 3 generated by thread 1 with duration 10
Task 4 generated by thread 2 with duration 10
Task 3 transferred from tasksQueue to firstPriorityQueue
Task 3 added to firstPriorityQueue
Task 4 transferred from tasksQueue to firstPriorityQueue
Worker 3 took task 3 from firstPriorityQueue
Task 4 added to firstPriorityQueue
Task 5 generated by thread 2 with duration 9
Maximum number of tasks generated
Task 5 transferred from tasksQueue to firstPriorityQueue
Task 5 added to firstPriorityQueue
Maximum number of tasks generated
Executing task Executing task 1 that took 10 seconds
Worker 1 successfully completed task 1 in firstPriorityQueue
2 that took 10 seconds
```

```
2 that took 10 seconds
Worker 1 took task 5 from firstPriorityQueue
Worker 2 successfully completed task 2 in firstPriorityQueue
Worker 2 took task 4 from firstPriorityQueue
Executing task 3 that took 10 seconds
Worker 3 successfully completed task 3 in firstPriorityQueue
Executing task 5 that took 9 seconds
Worker 1 successfully completed task 5 in firstPriorityQueue
Executing task 4 that took 10 seconds
Worker 2 successfully completed task 4 in firstPriorityQueue


=== Metrics ===
First Queue - Average wait time: 3.9511 seconds
First Queue - Average execution time: 9.80668 seconds
First Queue - Total tasks processed: 5
All tasks are completed.
```

20 tasks with 50%

```
"C:\KSE\Parallel and Client-Server Programming\parallel-assignmen
Task 1 generated by thread 1 with duration 10
Task 2 generated by thread 2 with duration 10
Task 1 transferred from tasksQueue to firstPriorityQueue
Task 1 added to firstPriorityQueue
Task 2 transferred from tasksQueue to firstPriorityQueue
Worker 1 took task 1 from firstPriorityQueue
Task 2 added to firstPriorityQueue
Worker 2 took task 2 from firstPriorityQueue
Task 3 generated by thread 2 with duration 10
Task 4 generated by thread 1 with duration 10
Task 3 transferred from tasksQueue to firstPriorityQueue
Task 3 added to firstPriorityQueue
Task 4 transferred from tasksQueue to firstPriorityQueue
Worker 3 took task 3 from firstPriorityQueue
Task 4 added to firstPriorityQueue
Task 5 generated by thread 2 with duration 9
Task 6 generated by thread 1 with duration 9
Task 5 transferred from tasksQueue to firstPriorityQueue
Task 5 added to firstPriorityQueue
Task 6 transferred from tasksQueue to firstPriorityQueue
Task 6 added to firstPriorityQueue
Task 7 generated by thread 2 with duration 9
Task 8 generated by thread 1 with duration 9
Task 7 transferred from tasksQueue to firstPriorityQueue
```

```
Task 7 added to firstPriorityQueue
Task 8 transferred from tasksQueue to firstPriorityQueue
Task 8 added to firstPriorityQueue
Task 9 generated by thread 1 with duration 10
Task 10 generated by thread 2 with duration 10
Task 9 transferred from tasksQueue to firstPriorityQueue
Task 9 added to firstPriorityQueue
Task 10 transferred from tasksQueue to firstPriorityQueue
Task 10 added to firstPriorityQueue
Task 11 generated by thread 2 with duration 9
Task 12 generated by thread 1 with duration 9
Task 11 transferred from tasksQueue to firstPriorityQueue
Task 11 added to firstPriorityQueue
Task 12 transferred from tasksQueue to firstPriorityQueue
Task 12 added to firstPriorityQueue
```

```
Task 13 generated by thread 2 with duration 5
Task 14 generated by thread 1 with duration 5
Task 13 transferred from tasksQueue to firstPriorityQueue
Task 13 added to firstPriorityQueue
Task 14 transferred from tasksQueue to firstPriorityQueue
Task 14 added to firstPriorityQueue
Task 15 generated by thread 2 with duration 5
Task 16 generated by thread 1 with duration 5
Task 15 transferred from tasksQueue to firstPriorityQueue
Task 15 added to firstPriorityQueue
Task 16 transferred from tasksQueue to firstPriorityQueue
Task 16 added to firstPriorityQueue
Task 17 generated by thread 1 with duration 9
Task 18 generated by thread 2 with duration 9
Task 17 transferred from tasksQueue to firstPriorityQueue
Task 17 added to firstPriorityQueue
Task 18 transferred from tasksQueue to firstPriorityQueue
Task 18 added to firstPriorityQueue
Task 19 generated by thread 1 with duration 7
Task 20 generated by thread 2 with duration 7
Task 19 transferred from tasksQueue to firstPriorityQueue
```

```
Task 19 added to firstPriorityQueue
Task 20 transferred from tasksQueue to firstPriorityQueue
Task 20 added to firstPriorityQueue
Maximum number of tasks generated
Maximum number of tasks generated
Executing task Executing task 1 that took 30 seconds
Worker 1 moved task 12 to secondQueue
Worker 1 took task 13 from firstPriorityQueue
 that took 30 seconds
Worker 2 moved task 2 to secondQueue
Worker 2 took task 15 from firstPriorityQueue
Executing task 3 that took 30 seconds
Worker 3 moved task 3 to secondQueue
Worker 3 took task 16 from firstPriorityQueue
Executing task Executing task 13 that took 5 seconds15 that took 5 seconds
Worker 2
Worker 1 successfully completed task 15 in firstPriorityQueue
Worker 2 successfully completed task 13 in firstPriorityQueue
 took task 14 from firstPriorityQueue
Worker 1 took task 19 from firstPriorityQueue
Executing task 16 that took 5 seconds
Worker 3 successfully completed task 16 in firstPriorityQueue
Worker 3 took task 20 from firstPriorityQueue
Executing task 14 that took 15 seconds
```

```
Executing task 16 that took 5 seconds
Worker 3 successfully completed task 16 in firstPriorityQueue
Worker 3 took task 20 from firstPriorityQueue
Executing task 14 that took 15 seconds
Worker 2 moved task 14 to secondQueue
Worker 2 took task 17 from firstPriorityQueue
Executing task 19 that took 21 seconds
Worker 1 moved task 19 to secondQueue
Worker 1 took task 18 from firstPriorityQueue
Executing task 20 that took 21 seconds
Worker 3 moved task 20 to secondQueue
Worker 3 took task 6 from firstPriorityQueue
Executing task 17 that took 27 seconds
Worker 2 moved task 17 to secondQueue
Worker 2 took task 11 from firstPriorityQueue
Executing task 18 that took 27 seconds
Worker 1 moved task 18 to secondQueue
Worker 1 took task 8 from firstPriorityQueue
Executing task 6 that took 27 seconds
Worker 3 moved task 6 to secondQueue
Worker 3 took task 5 from firstPriorityQueue
Executing task 11 that took 9 seconds
Worker 2 successfully completed task 11 in firstPriorityQueue
Worker 2 took task 7 from firstPriorityQueue
Executing task 8 that took 9 seconds
```

```
Worker 1 successfully completed task 8 in firstPriorityQueue
Worker 1 took task 12 from firstPriorityQueue
Executing task 5 that took 9 seconds
Worker 3 successfully completed task 5 in firstPriorityQueue
Worker 3 took task 9 from firstPriorityQueue
Executing task 7 that took 27 seconds
Worker 2 moved task 7 to secondQueue
Worker 2 took task 10 from firstPriorityQueue
Executing task 12 that took 27 seconds
Worker 1 moved task 12 to secondQueue
Worker 1 took task 4 from firstPriorityQueue
Executing task 9 that took 30 seconds
Worker 3 moved task 9 to secondQueue
Executing task 10 that took 10 seconds
Worker 2 successfully completed task 10 in firstPriorityQueue
Executing task 4 that took 10 seconds
Worker 1 successfully completed task 4 in firstPriorityQueue


=== Metrics ===
First Queue - Average wait time: 54.6643 seconds
First Queue - Average execution time: 18.7069 seconds
First Queue - Total tasks processed: 20
All tasks are completed.
```