Concurrent Non-Blocking Priority Queue Implementation

CS 550 Advanced Operating System

Prepared by

Nilesh Jorwar (A20405042)

Pradyot Mayank (A20405826)

Table of Contents

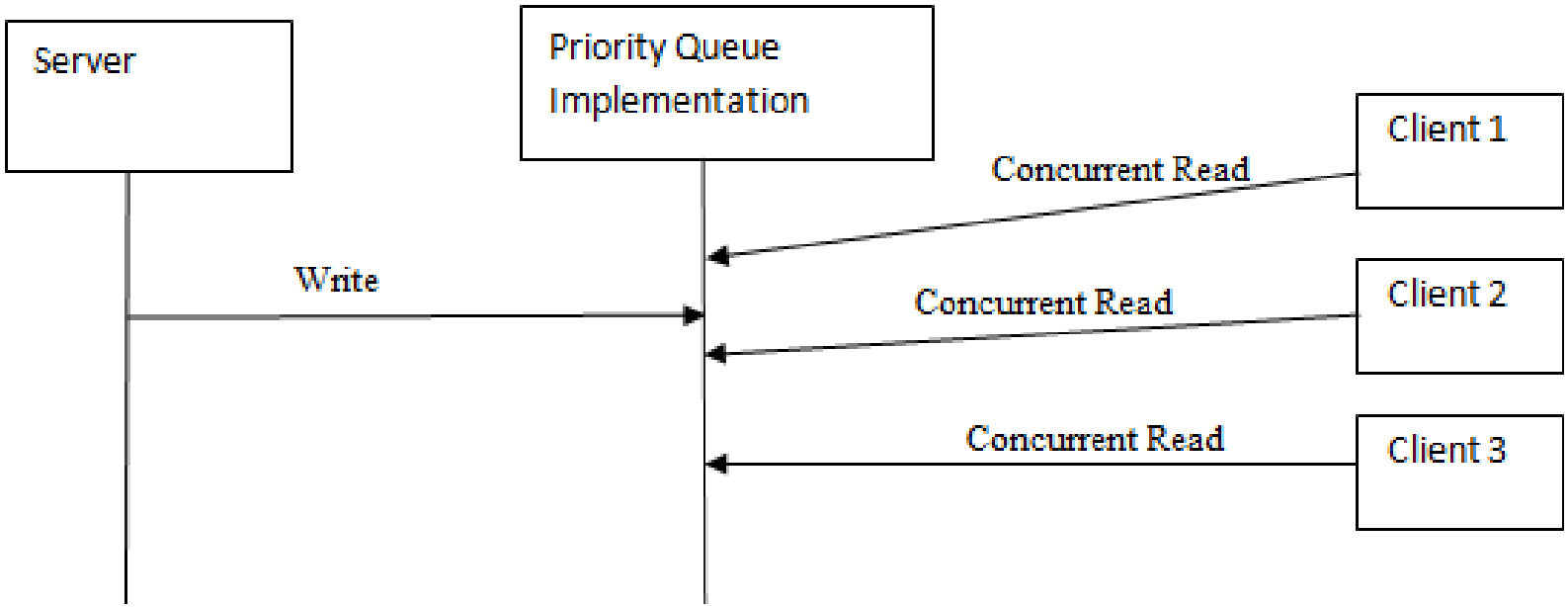
- Introduction
- Literature Review
- Problem Statement
- Proposed Method
- Evaluation
- Results
- Conclusion/Future Scope
- References

Introduction

Introduction

- By word “Non Blocking”, we can infer that either it performs the operation requested by the thread or notifies the thread that it can not perform the operation.
- Multithreaded environment is based on the concept of communication using various data structures like queue, maps, stacks etc.
- We have to build the concurrent algorithm to facilitate the concurrent access to the data structure by multiple threads which makes the data structure a concurrent data structure.
- If the algorithm guards the concurrent data structure is non blocking (no thread suspension), it is non blocking concurrent data structure.

- When it comes to thread safe concurrent access to multiple resources at the same time then there exists lot of such data structures like stack, ConcurrentLinkedQueue, linked List.
- These data structures however are implemented in many applications, but there comes the low output considering the efficiency and performance of such applications.
- Hence the term Priority Queue based application with O(log(n)) running performance which is very useful for widely growing distributed applications.



Concurrent Non Blocking Priority Queue Implementation

Literature Review

Literature Review

- Queues are ubiquitous in parallel programs, but their performance is matter of major concern.
- There has been development in the past regarding thread safe, wait free algorithms on various data structures like Stack, Queue[1] by Hunt and Michael L Scott.
- Michael L Scott. And Maged Michael presented the non blocking algorithm with one enqueue and one dequeue operations running concurrently using compare_and_safe and load_linked primitives[1].
- Lately, in 2004 again Michael Scott and William Scherrer came with implementation of dual data structures to describe the concurrent object implementation[2].

Problem Statement

Problem Statement

- With the fast pacing world and emerging technologies it becomes human and technology necessity to have access and availability to those with the higher priority.
- Though the currently available applications are capable enough to provide the services to those which are based on First Come First Serve (FCFS) policy but are subsequently failing to mitigate the exigencies needed for higher preference demanded.
- Widely available parallel applications based on the thought of Concurrent FIFO queues failing to serve clients/applications that need an immediate attention.
- This need leads to increase in indefinite idle time for waiting clients and makes the system slower due to flooding their requests.
- Hence implementing concurrent non blocking priority queue to handle the emergencies as per priorities.

Proposed Method

Proposed Method

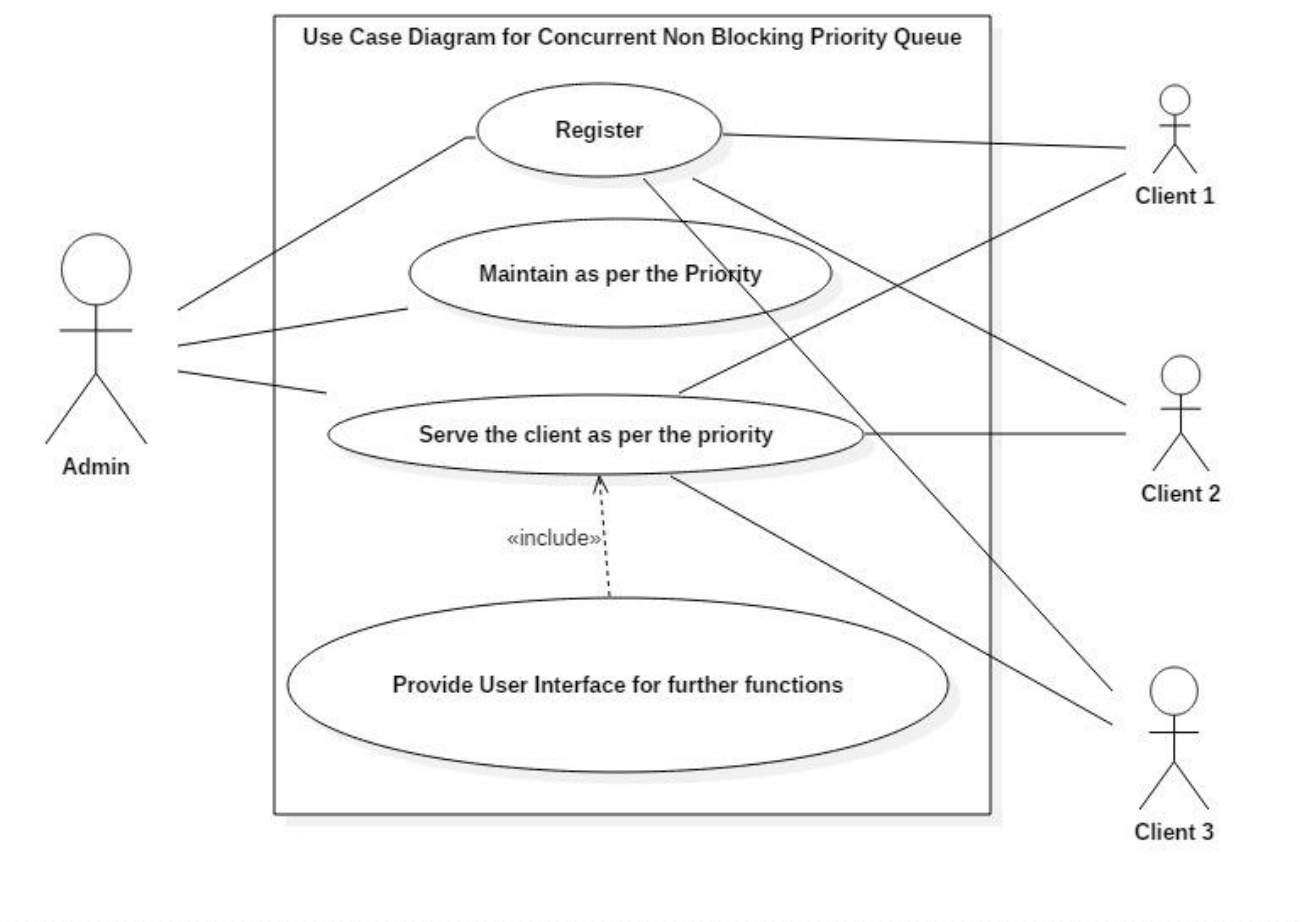
Objective-

- To implement the concurrent priority queue for the applications that allow higher priority threads/applications to get immediate access in case of emergencies.

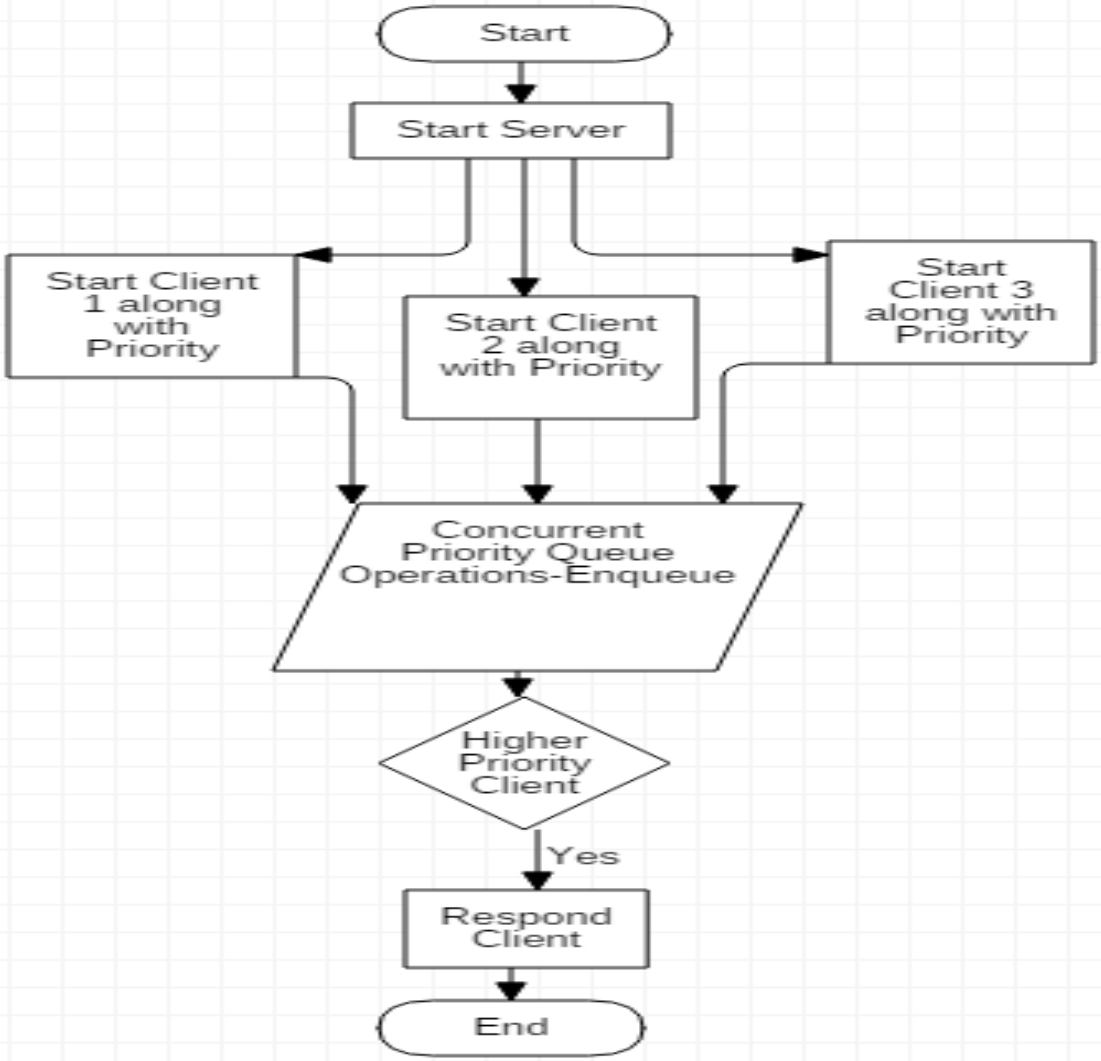
Methodology-

- The project follows an Incremental Development Methodology. Multiple development cycles would be taking place here. Cycles are divided up into smaller, more easily managed modules. Each module passes through the requirements, design, implementation and testing phases.
- This model is more flexible – less costly to change scope and requirements and it is easier to test and debug during a smaller iteration.

Use Case Diagram

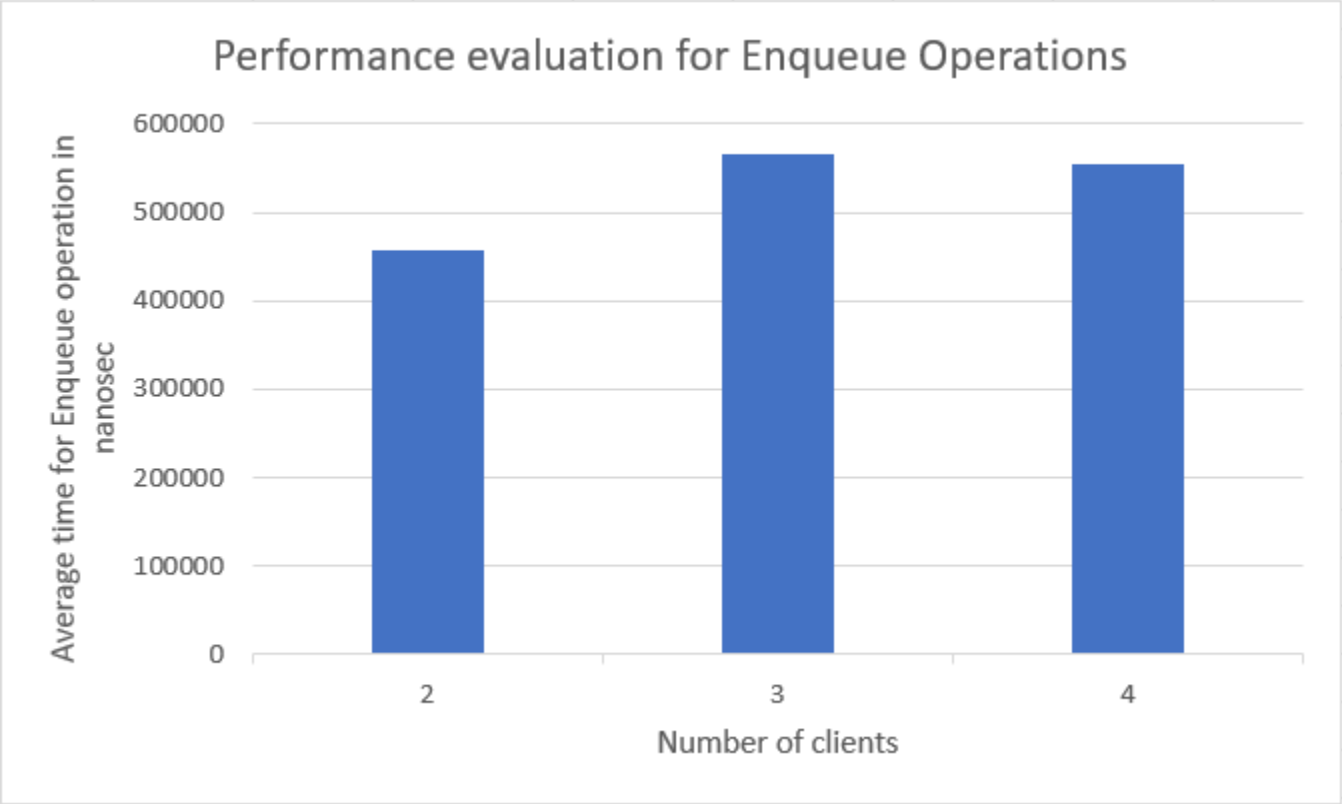


Flow Chart Diagram

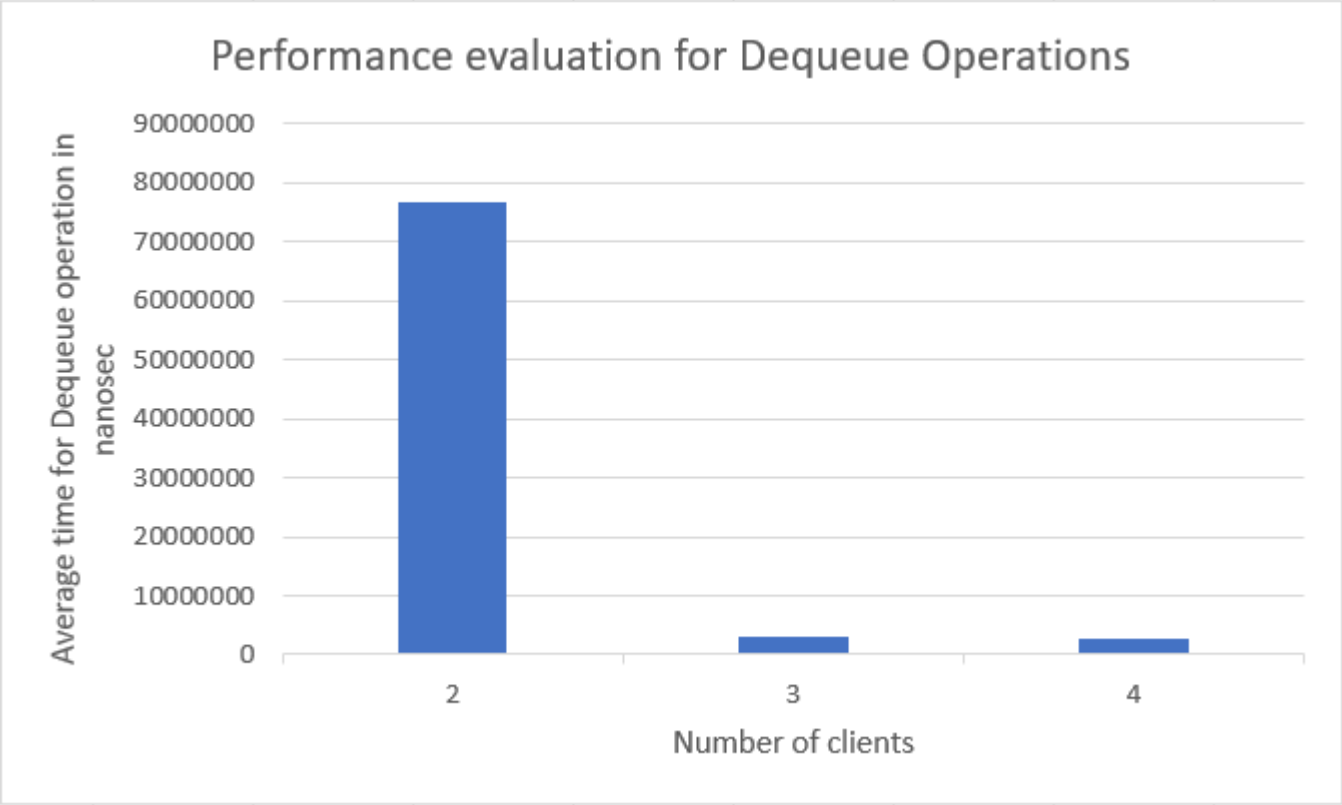


Evaluation

Performing all the operations of the queue without affecting the existing complexities.



Performing all the operations of the Dequeue without affecting the existing complexities



Results

Average time for enqueue operation in nanoseconds---455957 for 2 users
Array after Sorted--Client1 2 Thread[Thread-0,5,main]
Socket[addr=/192.168.1.112,port=49724,localport=9000]
Array after Sorted--Client2 10 Thread[Thread-1,5,main]
Socket[addr=/192.168.1.112,port=49725,localport=9000]
Average time for Dequeue operation in nanoseconds---76422438

Average time for enqueue operation in nanoseconds---565456 for 3 users
Array after Sorted--Client1 2 Thread[Thread-1,5,main]
Socket[addr=/192.168.1.112,port=49727,localport=9000]
Array after Sorted--Client3 4 Thread[Thread-0,5,main]
Socket[addr=/192.168.1.112,port=49726,localport=9000]
Array after Sorted--Client2 10 Thread[Thread-2,5,main]
Socket[addr=/192.168.1.112,port=49728,localport=9000]
Average time for Dequeue operation in nanoseconds---2729755

Average time for enqueue operation in nanoseconds---553479 for 4 users
Array after Sorted--Client4 0 Thread[Thread-1,5,main]
Socket[addr=/192.168.1.112,port=49735,localport=9000]
Array after Sorted--Client1 2 Thread[Thread-0,5,main]
Socket[addr=/192.168.1.112,port=49734,localport=9000]
Array after Sorted--Client3 4 Thread[Thread-2,5,main]
Socket[addr=/192.168.1.112,port=49736,localport=9000]
Array after Sorted--Client2 10 Thread[Thread-3,5,main]
Socket[addr=/192.168.1.112,port=49737,localport=9000]
Average time for Dequeue operation in nanoseconds---2660463

Conclusion/Future Scope

Conclusion/Future Scope

- This module could be used for implementation Producer Consumer problem with priority.
- It can be used for implementation of hospital management where there is only one staff i.e. doctor.
- Scanning through a large collection of statistics to report the top N items - N busiest network connections, N most valuable customers, N largest disk users
- This can be further enhanced as much as to be implemented for the applications stated above.

References

[1] Maged M. Michael Michael L. Scott. Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms.

[2] William N. Scherer III and Michael L. Scott. Nonblocking Concurrent Data Structures with Condition Synchronization.

[3] Carole Delporte, Hugues Fauconnier,Michel Raynal. An Exercise in Concurrency: From Non-blocking Objects to Fair Objects ,2014