Bilkent University

Department of Computer Engineering

CS 342 - Operating Systems

2023 - 2024

Spring Semester

Project 3

Murat Çağrı Kara - 22102505 - Section 1

Görkem Kadir Solun - 22003214 - Section 1

Performance Tests:

The algorithm for inserting message queues into the system breaks down the shared memory into three partitions. The first shared memory region consists of fixed memory that stores the message queue headers that have the purpose of storing the size of the message queue, message queue ID, etc. The second region consists of variables such as queue count, free space, active processes, etc. Then, the third shared memory region contains the message queues; it inserts the message queues by considering the next fit.

1. Message Latency

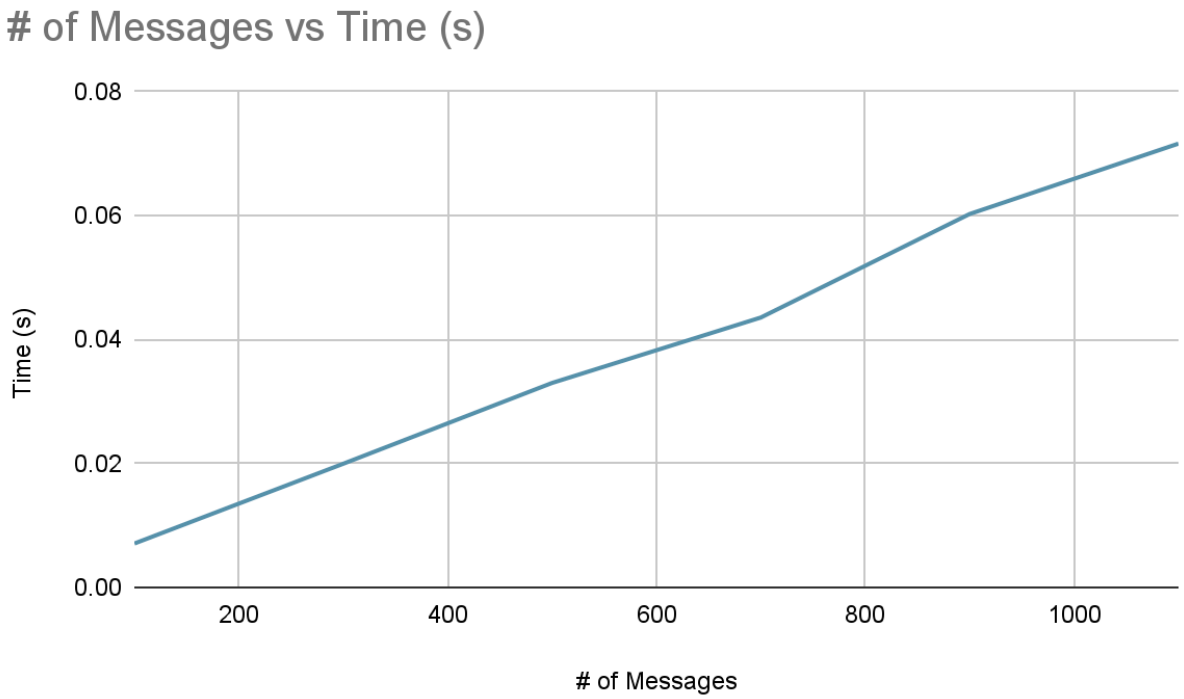
1.1. Sending a Message

Sending messages to the message queue by writing them into operation can cause message latency. Sending from the message queue is available when a process wants to write a message to the message queue. Sending a message can block the caller process until space is available in the queue. This can lead to a time latency. This phenomenon is tested by sending multiple messages by filling the shared memory region, and a process tries to write to the message queue whilst they are full.

1.2. Receiving a Message

Reading from the message queue can cause message latency. Reading from the message queue is available when a process has written a message on the message queue buffer placed in shared memory, and another process wants to retrieve that message. Since messages are sent and received in the order that they are placed (FIFO), the messages that are placed first will be removed from the shared buffer. Since multiple processes are expected to run in this implementation, it was ensured that there would be no synchronization problems. If a pair is using that particular queue for a message, other processes can access messages from other queues but not the ones that are already being used by two processes. The time taken for a message to be received by a process depends on whether the message is available for removal or not. The test includes concurrency behaviour by showing that other processes can receive or send messages (no time difference) in parallel with the to/from for different message queues as well as the time taken for a process to receive a message.

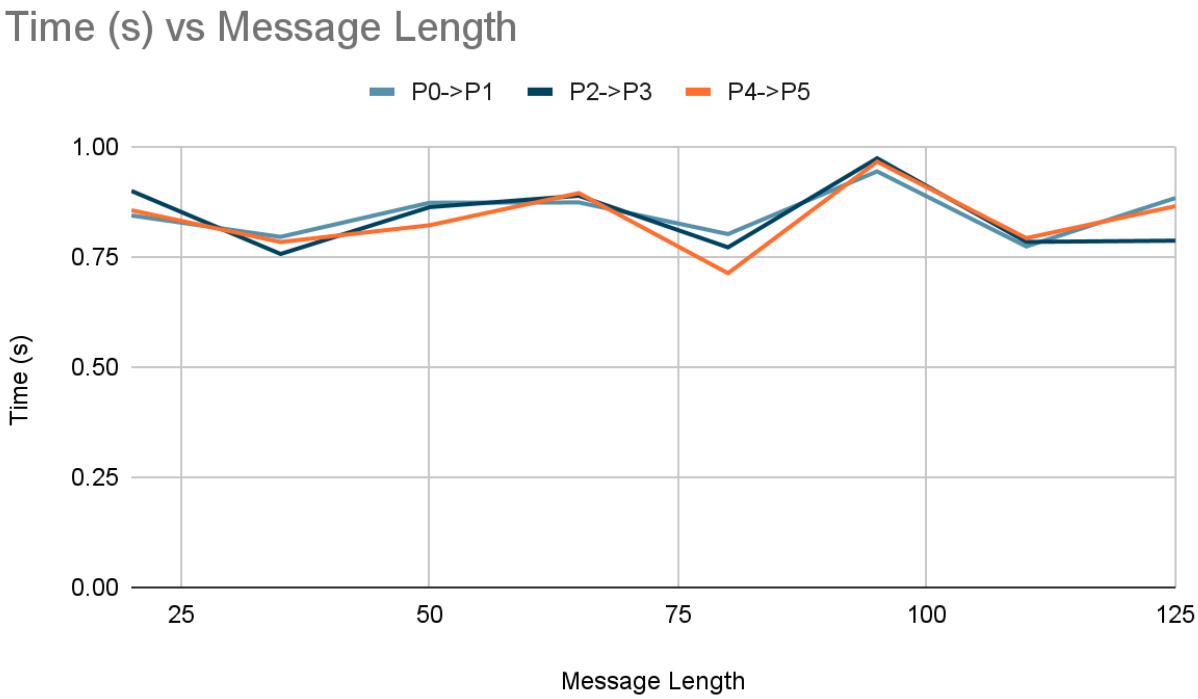
In producer-consumer relations, the number of messages yields more time as it will need more time to consume the data brought by the producer. The test includes the consumer from the start of the first received message until consuming all of the received messages.



As observed, # of messages and time taken for the consumer to consume the data has linear progression. Our interpretation of the consumer-producer relation was correct.

1.3. Receiving & Send Graph

Graph for one process sending a message to the message queue and one process receiving the message from the message queue depending on message sizes (concurrent process structure):



It is observed that concurrent processes work in parallel since they are unaffected by the send-and-receive relationship between the processes. It is observed that sending a message to the message queue and retrieving it to another process yields very similar times in different message sizes. Fluctuations in the graph can be caused by the cache accessing some memory locations faster, resulting in some message queues being accessed at a faster rate and some slower, thus causing a fluctuating graph.

2. Shared Memory Access

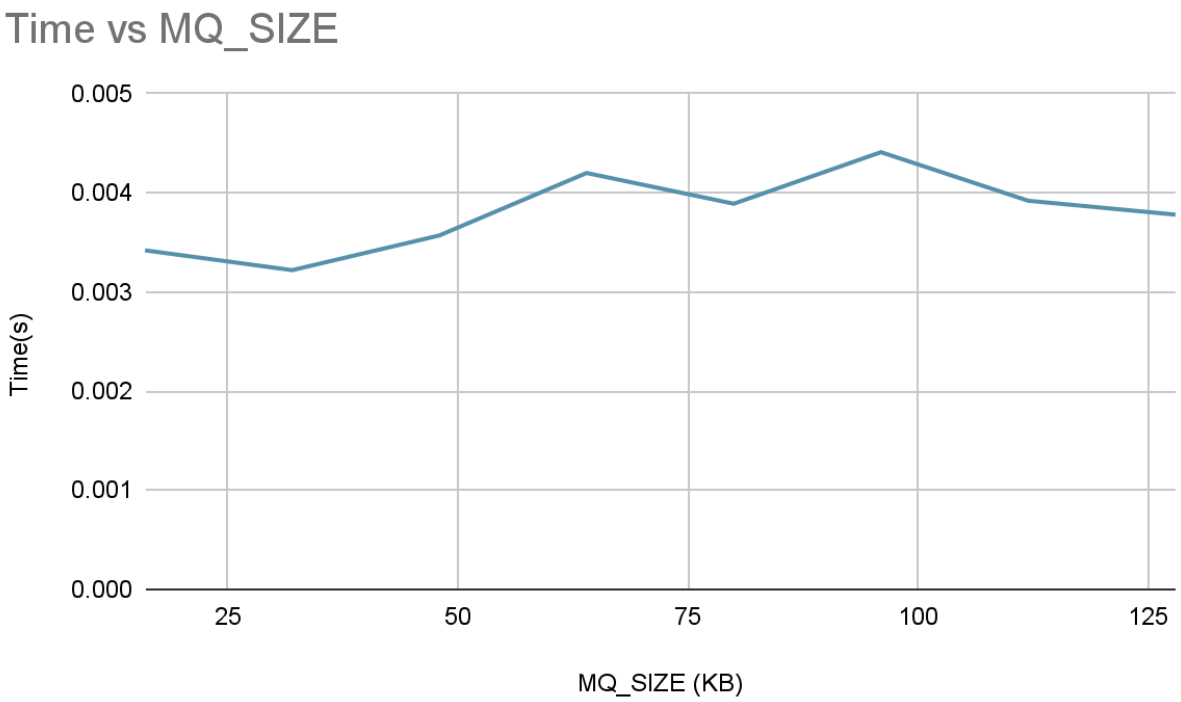
2.1. Shared Memory Reading & Writing Message Queue

2.1.1. Removing From the Shared Memory

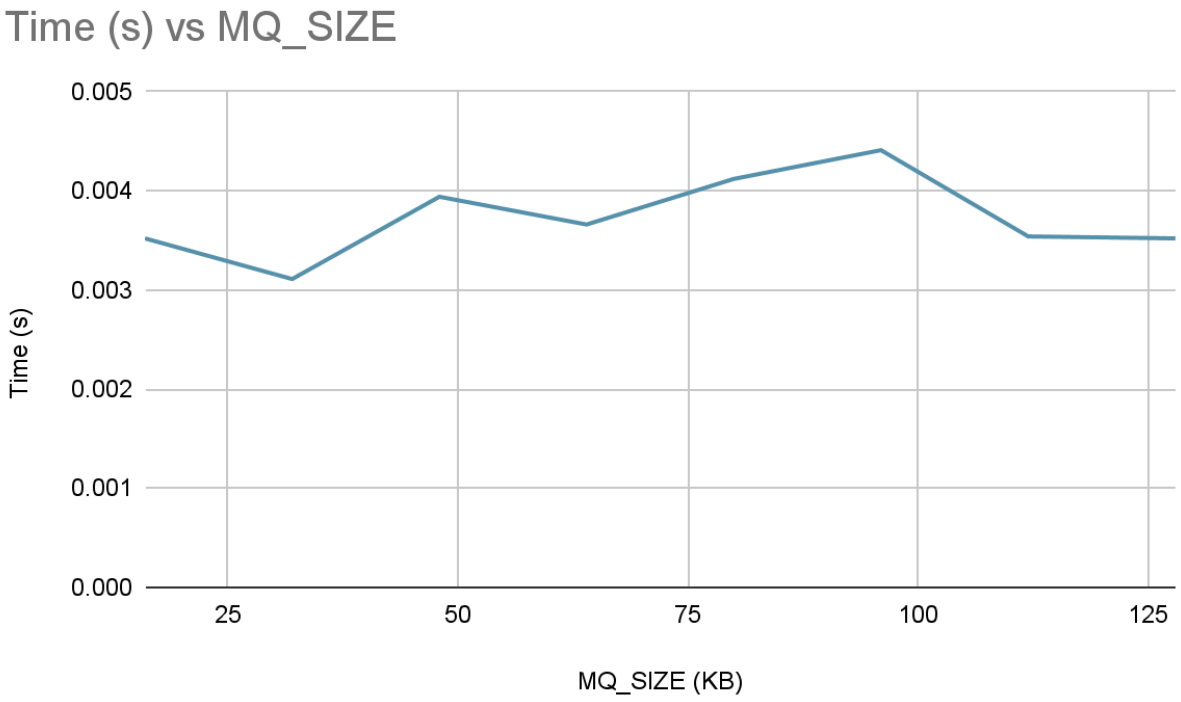
Since the implementation of putting message queues inside of shared memory is through a next fit algorithm, given the message queue name, in the fixed region, we will search for the name of the message queue, and if we have a match, we will perform the removal procedure (reading from the shared memory). Moreover, as the search takes place on the fixed side of the shared memory, the actual address of the message queue is to be accessed using index operations (writing from the shared memory). Since message queues will be written in the fixed region one after the other, removing the element which is placed first in the shared memory will require less time, as getting the index information of the message queue elements requires less time when searched. The test first includes removing message elements

that are inserted beforehand in sorted order and then removing message queue elements inserted beforehand in reversely sorted order.

Graph for removing message queues in the normal order that they are inserted:



Graph for removing message queues in reverse order that they are inserted:

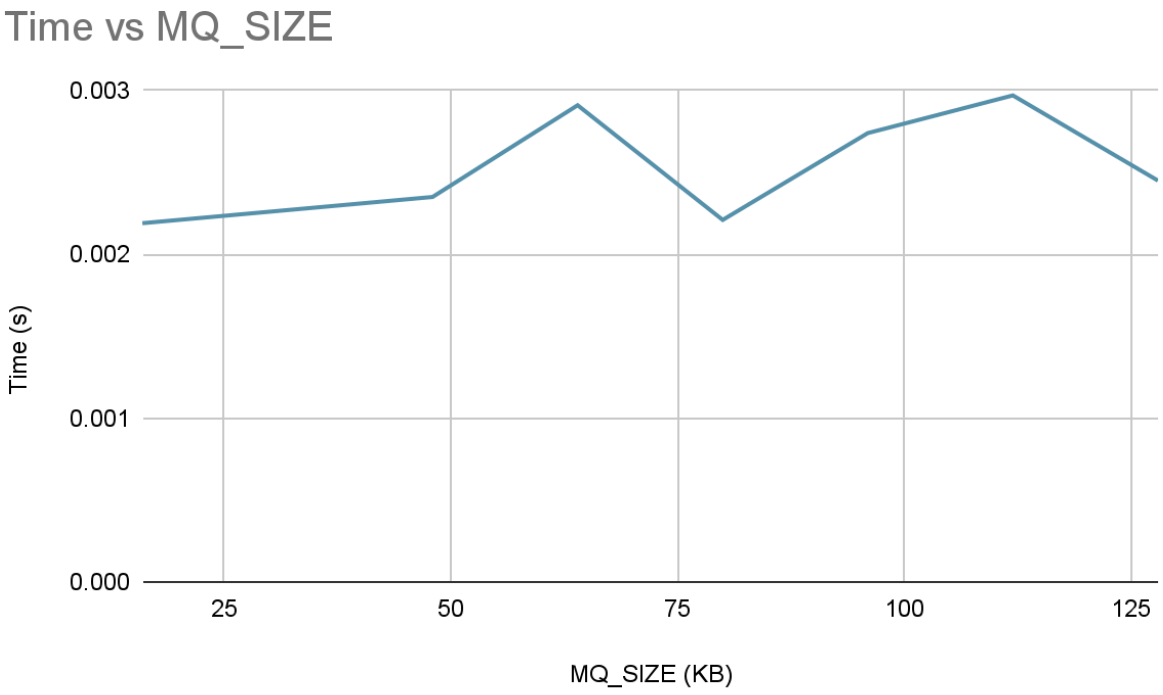


It is observed that removing messages in reverse order and removing messages in the normal order of queues have negligible effects on the system. It is deduced that the next fit algorithm in a small amount of shared memory region contributes no significant change in the overall run time of the system. Fluctuations in the graph can be caused by the cache accessing some memory locations faster, resulting in some message queues being accessed at a faster rate and some slower, thus causing a fluctuating graph.

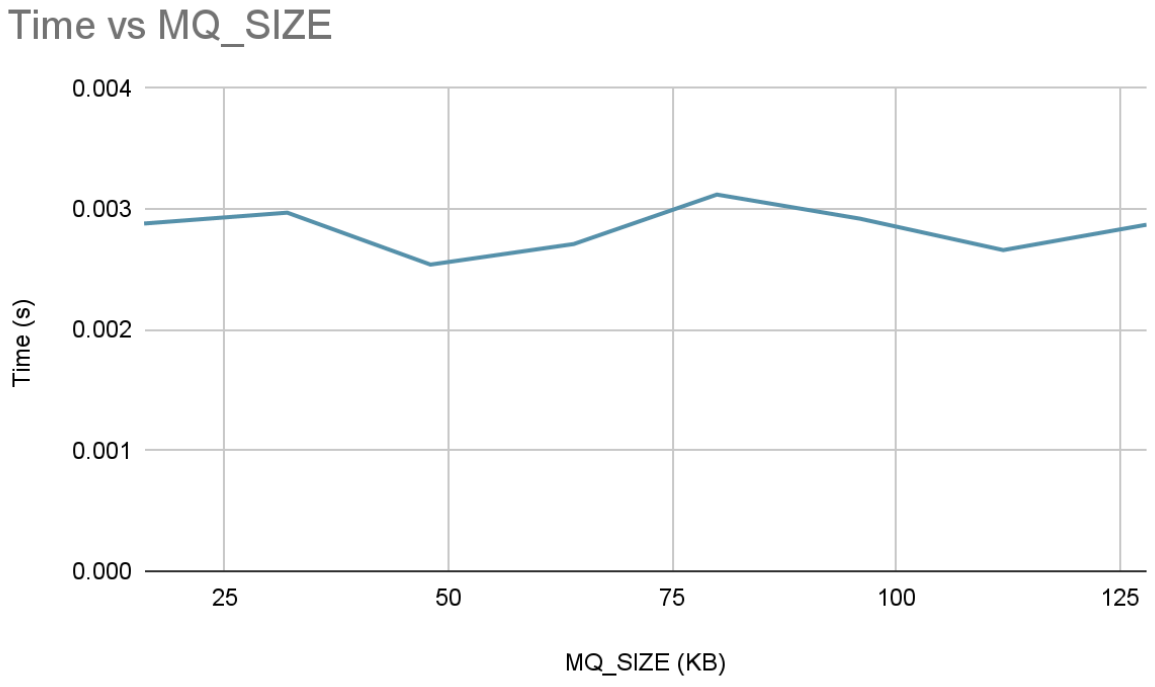
2.1.2. Writing to Shared Memory

As the next fit algorithm implementation searches the available shared memory segment for the given message queue, when the holes are distributed as hole->message queue->hole…, and the hole regions have smaller sizes than the given message queue size that will be inserted into the shared memory, this yields the worst possible outcome. Thus, the test consists of first inserting message queues into a fixed memory region and plotting their elapsed time. Then, the test continues by removing message queue elements smaller in size than the ones that enter the shared memory region (ex. Removing a 16KB message queue and inserting a 32KB message queue).

Graph for inserting message queues on an empty shared memory:



Graph for inserting message queues on a shared memory where the first couple of hole regions(done by removing message queues smaller than the ones that enter) are smaller than the newly inserted message queues:



It is observed that the time when mq_sizes cannot fit the first couple of hole regions proved to have small inefficiency in contrast to the ones that entered into an empty shared memory region. Noting that these changes are so small that they can be considered negligible. Fluctuations in the graph can be caused by the cache accessing some memory locations faster, resulting in some message queues being accessed at a faster rate and some slower, thus causing a fluctuating graph.