Producer-Consumer problem using Thread pool

A.M. Abhishek Sai
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri , India
amenu4aie20101@am.students.amrita.edu

Dinesh Reddy
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri , India
amenu4aie20130@am.students.amrita.edu

Perumalla Raghavendra
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri , India
amenu4aie20156@am.students.amrita.edu

G.Yashwanth Kiran
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri , India
amenu4aie20175@am.students.amrita.edu

Rejeenth V R
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri , India
rejeenthvr@am.amrita.edu

***Abstract*—**

Many server programs perform repetitive operations on a large number of short tasks. As there is a need for the improvement of the efficiency in doing many concurrent processes, thread-pools came into existence to improve the performance of these concurrent servers. The thread-pool-based solution to the producer-consumer problem is discussed in this paper. We used semaphores to prevent multiple processes from accessing the same buffer at the same time. We've discussed how the proposed producer-consumer problem plays out in various scenarios. We observed that using thread-pool instead of threads made this model more efficient in terms of time. This paper describes a real-life scenario in which this approach can be employed (Delivery System).

***Index Terms*—semaphores , blocking queue , buffer , Thread-pool, Delivery System**

I. INTRODUCTION

The producer-consumer problem is a classic example of the multi-process synchronization problem in computer science. The most common and important problem in software design and its development is Synchronization. In many applications such as Synchronous Product and FlexSync, uses synchronization as their method. Synchronization is required when more than one threads or processors access a critical resource [1].

The producer-consumer problem can be implemented in a variety of applications and run on single and multi-processor systems. Many scholars have investigated this subject because of its importance and are commonly employed to visualize the problem's behavior.

Multi-process synchronization, or synchronization between several processes, is solved using the Producer-Consumer problem.

One producer and one customer are involved in this problem. The producer creates something, and the consumer consumes something generated by the producer. A constant size memory buffer is shared between the producer and the consumer. By reusing previously created threads, a thread pool solves the problem of resource thrashing and excessive thread cycle overhead. The time taken to create threads are eliminated as Thread-pool reuses the thread, which makes the application more effective.

Thread pools follow a set of rules to control the life cycle of threads. Rather than creating a new thread for every new request, the thread pool holds a certain number of idle threads and reuses them to complete tasks. Thread starvation will thus be prevented.

II. BACKGROUND

As a consultant for the Electrologica X1 and X8 computers, Dijkstra discovered the solution to the producer-consumer problem. The first implementation of producer-consumer was 'partly software, partly hardware. [2].

Niklaus Wirth and Per Brinch Hansen recognized the difficulty with semaphores early on, and they came to the conclusion about semaphores that are not suited for higher-level languages.

A. *Related works*

The Producer-Consumer Problem model by Liangzhou Wang and Chaobin Wangwas was proposed in 2020, but we have added semaphores additional to it. Our proposed solution has all the properties that a producer-consumer problem has and can handle the corner cases [3].

Aspect-Oriented Programming was used by Yang Zhang and Jingjun Zhang to implement and test the Producer-Consumer Problem, whereas the proposed model is implemented using object-oriented programming [4].

Syed Nasir Mehmood, Nazleeni Haron has implemented the producer-consumer problem using threads, but we have proposed a model using thread pool which is optimum than threads [5].

III. Producer-Consumer Problem and Thread pool

Coming to the Producer and consumer problem, this is a basic concurrency problem that will arise when a process is producing some data and simultaneously another process is using the same data (consumer) [6]. At times the rate between the processes may vary, consumers may work faster or producers may work faster. So to support this we used a blocking queue as a buffer between the producer and consumer as the BlockingQueue interface supports managing rates of data transmission by introducing blocking if it is full or empty. A thread attempting to enqueue an element in a full queue will be halted until another thread clears the queue fully or dequeues one or more elements. It also prevents a thread from deleting from an empty queue until one or more other threads insert an item. Deadlocks can be explained by the collaboration between the producer and the consumer [7]. When one thread's execution must be stopped by another, the multithreading approach comes into play [8].The semaphores and queues handle task synchronisation [9].

The main 3 requirements for our project include :

1) The buffer (blocking queue) cannot be accessed simultaneously by the Producer and the Consumer.

2) If the memory buffer is full, the producer cannot produce data. It signifies that when the memory buffer isn't full, the data can be produced by the producer.

3) Only when the memory buffer is not empty, the consumers can consume data. When the memory buffer is empty, the consumer is not permitted to access the data stored inside.

Manually, creating and destroying threads is not a good idea. Thread generation is a time-consuming process [10] [11]. The following steps are involved in the creation of each thread:

1. It allocates memory for a thread stack

2. The operating system generates a native thread that corresponds to the Java thread.

3. Thread-related descriptors are added to the JVM's internal data structures.

Aside from the cost of creating them manually, another issue is, we have no control of threads that are running parallel. Let us consider a example, if a server app receives millions of requests, and each request generates a new thread, so now millions of threads will operate in parallel, potentially resulting in thread starvation.

You must put certain jobs in the task queue in order to run them simultaneously. When a thread becomes available, it will accept and execute a job. The more threads accessible, the more work can be completed concurrently.

Aside from thread lifetime management, another benefit of working with a thread pool is that you can think more functionally when planning how to partition the work to be completed concurrently. The task has replaced the thread as the unit of parallelism. You create tasks that are executed simultaneously rather than threads that share a common memory and run in parallel. Deadlocks and data races are two frequent multithreading issues that can be avoided by thinking functionally [12]. The task of parallel and synchronised thread execution is delegated to a thread pool executor [11]. Processes and their threads even can be used to run Android applications [13]. Various processes, such as the TLS handshake, audio sample capturing and real-time prediction of distraction,from a video input were synchronised using Java threads [14] [15].

One advantage of using a thread pool versus creating a new thread for each operation is that the overhead of thread creation and destruction is limited to the pool's initial formation. Destroying and creating thread is a time-consuming procedure.

There is also a small issue with the thread-pool. Managing a large thread pool can waste processing and memory resources, if there are many unused threads in the pool. To handle new requests, additional threads have to be created and destroyed as needed if the thread pool is too small. Solution to this problem is to maintain a optimal thread pool size [16]. The heuristic algorithm also helps to determine the optimal thread pool size [17]. There was also a research of implementing a prediction-based dynamic thread pool model using customized exponential average which improves the server's response time [18]. The other way to find optimal size of thread-pool is, by modeling the rate change in the number of threads requested using a Gaussian distribution. Then we can predict the number of threads required in the thread-pool [19].

IV. METHODOLOGY

Thread pool model based on producer-consumer model

A. Working of Semaphore

If semaphore count > 0, the thread acquires a permit, and the semaphore's count gets decreased. Otherwise, until a permit is obtained, the thread will be locked. The permit is released and the semaphore count is incremented when a thread no longer requires access to a shared resource.

acquire() - acquires a permit if one is available

release() – release a permit

B. Key-Terms

The Java executor service interface allows us to asynchronously execute tasks on threads, which is present in the java.util.concurrent package. It manages Thread-pool and assigns tasks to them [20].

In Java, we use the ExecutorService to build a thread pool of a given size. [21].

It makes asynchronous task execution easier.

execute(): The execute() method of the Java ExecutorService accepts a runnable object and performs its task asynchronously. We call the shutdown method after calling the execute method, which prevents any more tasks from being queued in the executor services.

submit(): Callable or Runnable tasks are submitted to the ExecutorService and return the result of type future

shutdown(): When there are no tasks to process, the ExecutorService will not be automatically destroyed. It will continue to exist and wait for new tasks to be assigned to it.

This function does not destroy the ExecutorService immediately. After all running threads finish their present work, the ExecutorService will stop taking new tasks and shut down.

runnable: The Java runnable interface is used to execute code on several threads at the same time.

Java BlockingQueue:

put(): To Insert elements into the queue we use this function. It waits for the spot to become available if the queue is full.

take(): This function retrieves the element at the top of the queue and removes it. It waits for the element to be available if the queue is empty

C. Procedure :

producer class: this class actually implements the runnable interface which is inbuilt in java and overrides the run method to execute.

In the producer class at first, we acquire a permit from the semaphore and release the permit after completion. If a permit was not available, it waits until it gets a permit and starts execution. This method produces some data and puts it into the blocking queue (buffer) and when the blocking queue is full, the producer asks the user to stop or continue. If the user says continue it will first consume the data which is already in the buffer then again produce data recursively until the last data was produced. If the user wants to stop producing when the buffer is full then the producer stops producing.

Consumer class: this class actually implements the runnable interface which is inbuilt in java and overrides the run method to execute.

In the consumer class at first, we acquire a permit from the semaphore and release the permit after completion. If the permit was not available, it waits until it gets a permit and starts execution.

In this method, we just retrieve the available data from the buffer, print the retrieved data, and remove it from the buffer.

V. APPLICATIONS

In the operating systems, Interfacing with a network device in which the network producing data at a certain rate and placing it on the buffer and the OS consuming the data from the buffer may be at a different rate. Any real-time system relies on a real-time operating system (RTOS) [22].

The network interface card will consist buffer that will be storing the incoming packets from the internet. The hosts will act like a producer producing the packets and the network device driver will consume the packets and then put them in the main memory.

Multi-process servers have lower concurrency processing power and consume more memory on lightweight servers. The generation of multi-threads on a regular basis requires more resources on light servers. As a result, classic concurrent servers based on multi-process/multi-threads can no longer match the demands of high concurrency.

A. *Delivery system using Producer consumer problem*

A hotel/restaurant where food is ordered and delivered to customers is an example of the producer-consumer problem, where the customers are considered as consumers and the hotel/restaurant is considered as the producer. And a delivery agent who delivers the food is considered to act as a buffer. Ideally, if the delivery agents have enough food to carry on them, the hotels should not try to add more food items. The OTP (One Time Password) will not be generated since the hotel will not add any food items to the delivery agent. The customer needs to wait until any delivery agent leaves a space for a new food item to be delivered. When an order is picked up by an agent, a confirmation OTP is delivered to the customer. Moreover, when there are no agents available to deliver the food, the producer is notified that he can either wait or quit the app.

If they opt to wait: After the delivery of previous orders, the restaurants now produce the current order and it is taken by the delivery boys and an OTP is delivered to the customer.

If customers opt to quit: The restaurants stop producing their orders and the boys just deliver their previous orders.

VI. Results

```
enter size of Task queue
5
Produced task : 0  by   pool-1-thread-1
Produced task : 1  by   pool-1-thread-1
Produced task : 2  by   pool-1-thread-1
Produced task : 3  by   pool-1-thread-1
Produced task : 4  by   pool-1-thread-1
Consumed task 0 By  pool-2-thread-1
Consumed task 1 By  pool-2-thread-1
Consumed task 2 By  pool-2-thread-1
Consumed task 3 By  pool-2-thread-1
Consumed task 4 By  pool-2-thread-1
```

Fig. 1. output of producer-consumer problem using thread-pool

All the tasks get produced once, and all the tasks get consumed after producing, as the producer and consumer cannot access the buffer at the same time

When the task queue is full, the producer can't produce any more data. So we must consume it before producing much more tasks. It is designed in such a way that, if we want to continue then the consumer consumes first and makes the task queue empty. The producer starts producing again which is remaining. If we quit the rest process gets terminated and the consumer consumes which are in the task queue.

```
enter size of Task queue
3
Produced task : 0  by   pool-1-thread-1
Produced task : 1  by   pool-1-thread-1
Produced task : 2  by   pool-1-thread-1
The Buffer is full but the producer tries to put data
into the buffer.Do you want to continue or stop? :

Enter 0 for stop and any other num to continue
4
Consuming task 0 By pool-1-thread-1
Consuming task 1 By pool-1-thread-1
Consuming task 2 By pool-1-thread-1
Produced task : 3  by   pool-1-thread-1
Produced task : 4  by   pool-1-thread-1
Consumed task 3 By  pool-2-thread-1
Consumed task 4 By  pool-2-thread-1
```

Fig. 2. output of producer-consumer problem using thread pool when the buffer is full but producer wants to produce data

```
enter size of Task queue
3
Produced task : 0  by   pool-1-thread-1
Produced task : 1  by   pool-1-thread-1
Produced task : 2  by   pool-1-thread-1
The Buffer is full but the producer tries to put data
into the buffer.Do you want to continue or stop? :

Enter 0 for stop and any other num to continue
0
Consumed task 0 By  pool-2-thread-1
Consumed task 1 By  pool-2-thread-1
Consumed task 2 By  pool-2-thread-1
```

Fig. 3. output of producer-consumer problem using thread-pool when buffer is full , producer stops producing data

As the producer stops producing, the data produced already is now consumed.

A. *Delivery System*

```
enter size of Task queue
5
Produced Order : 1  and taken by delivary boy
Produced Order : 2  and taken by delivary boy
Produced Order : 3  and taken by delivary boy
Produced Order : 4  and taken by delivary boy
Produced Order : 5  and taken by delivary boy
Delivered Order to the customer and the otp generated was 8953
Delivered Order to the customer and the otp generated was 9538
Delivered Order to the customer and the otp generated was 5402
Delivered Order to the customer and the otp generated was 3483
Delivered Order to the customer and the otp generated was 6295
```

Fig. 4. output of Delivery System using producer-consumer problem

Since the producer and consumer cannot access the buffer at the same time, the order gets placed first and then gets delivered along with the OTP which gets generated for the corresponding order when delivered.

```
enter size of Task queue
3
Produced Order : 1  and taken by delivary boy
Produced Order : 2  and taken by delivary boy
Produced Order : 3  and taken by delivary boy
Delivary boys are not available currently.Do you want to wait or stop? :
Enter 0 for stop and any other num to continue
1
Delivered Order to the customer and the otp generated was 7236
Delivered Order to the customer and the otp generated was 9743
Delivered Order to the customer and the otp generated was 2580
Produced Order : 4  and taken by delivary boy
Produced Order : 5  and taken by delivary boy
Delivered Order to the customer and the otp generated was 9770
Delivered Order to the customer and the otp generated was 6035
```

Fig. 5. output of Delivery System when there is no delivery man free (task queue is full)

when all the delivery men are busy then it asks whether to wait or quit. If he (customer) waits then all the order gets delivered with OTP which makes all the delivery boys free (task queue is empty) and then the remaining order gets produced and gets delivered with OTP. If we quit the remaining order gets terminated.

```
enter size of Task queue
3
Produced Order : 1  and taken by delivary boy
Produced Order : 2  and taken by delivary boy
Produced Order : 3  and taken by delivary boy
Delivary boys are not available currently.Do you want to wait or stop?
Enter 0 for stop and any other num to continue
0
Delivered Order to the customer and the otp generated was 8364
Delivered Order to the customer and the otp generated was 9711
Delivered Order to the customer and the otp generated was 8465
```

Fig. 6. output of Delivery System when customer want to exit if a delivery man is not available

The producer stops producing their order and the boys just deliver their previous orders and the remaining orders get terminated.

VII. Conclusion

we have proposed the thread-pool-based solution to the producer-consumer problem. Using this thread pool we have saved a lot of time as there is no need to create threads for every task. The different ways to find the optimal thread-pool size is discussed. Semaphores in this model prevented multiple processes from simultaneously accessing the same buffer, allowing the model to run efficiently. We have successfully implemented a delivery system that is similar to the producer-consumer problem and after completion of every successful delivery, OTP is sent to the customer.

References

[1] "Tongyoo, t., v. chutchavong, and o. sangaroon. "object-oriented design message control multi-threaded execution." 2006 sice-icase international joint conference. ieee, 2006."

[2] "Dijkstra; 2000; ewd1303 my recollections of operating system design."

[3] "Wang, liangzhou, and chaobin wang. "producer-consumer model based thread pool design." journal of physics: Conference series. vol. 1616. no. 1. iop publishing, 2020."

[4] "Zhang, yang, jingjun zhang, and dongwen zhang. "implementing and testing producer-consumer problem using aspect-oriented programming." 2009 fifth international conference on information assurance and security. vol. 2. ieee, 2009."

[5] "Mehmood, syed nasir, et al. "implementation and experimentation of producer-consumer synchronization problem." international journal of computer applications 975.8887 (2011): 32-37."

[6] "Tanenbaum, andrew s. "processes and threads." modern operating systems (2008)."

[7] "Luo, ying, and xianping wang. "the study of the classic producer-consumer problem in a series of it courses." proceedings of the 21st annual conference on information technology education. 2020."

[8] "Arvind, s., and v. anantha narayanan. "an overview of security in coap: attack and analysis." 2019 5th international conference on advanced computing communication systems (icaccs). ieee, 2019."

[9] "Palatty, jashin joseph, srinivas phani chandra edireswarapu, and p. sivraj. "performance analysis of freertos based video capture system." 2019 3rd international conference on electronics, communication and aerospace technology (iceca). ieee, 2019."

[10] "Saha, aishika. "application of threads."."

[11] "Megalingam, rajesh kannan, and avinash hegde kota. "optimal approach to speech recognition with ros." 2021 6th international conference on communication and electronics systems (icces). ieee, 2021."

[12] "Sedlacek, peter, marek kmec, and patrik rusnak. "software visualization application for threads synchronization handling in operating systems." 2020 18th international conference on emerging elearning technologies and applications (iceta). ieee, 2020."

[13] "Anusuya, r., and prabhaker mateti. "android dashboard for past and present processes." 2018 international conference on advances in computing, communications and informatics (icacci). ieee, 2018."

[14] "Lakshmi, r. vidya, et al. "jpermit: usable and secure registration of guest-phones into enterprise voip network." 2010 international conference on advances in computer engineering. ieee, 2010."

[15] "Narayanan, ajay, et al. "real-time detection of distracted drivers using a deep neural network and multi-threading." advances in artificial intelligence and data engineering. springer, singapore, 2021. 89-100."

[16] "Ling, yibei, tracy mullen, and xiaola lin. "analysis of optimal thread pool size." acm sigops operating systems review 34.2 (2000): 42-55."

[17] "Xu, dongping. performance study and dynamic optimization design for thread pool systems. no. is-t 2359. ames lab., ames, ia (united states), 2004."

[18] "Kang, donghyun, et al. "prediction-based dynamic thread pool scheme for efficient resource usage." 2008 ieee 8th international conference on computer and information technology workshops. ieee, 2008."

[19] "Kim, ji hoon, et al. "prediction-based dynamic thread pool management of agent platform for ubiquitous computing." international conference on ubiquitous intelligence and computing. springer, berlin, heidelberg, 2007."

[20] baeldung, "A guide to the java executorservice," Dec. 2021.

[21] "Java executorservice," Dec. 2021.

[22] "Rajesh, m., and b. sreevidya. "vulnerability analysis of real-time operating systems for wireless sensor networks." advanced computing and intelligent engineering. springer, singapore, 2020. 449-460."