OS Lab First Assignment

Saba Madadi

26 Azar 1403

Overview

In this exercise, which is the first lab assignment for Operating Systems, the goal is to implement producer and consumer functions in the producer-consumer algorithm using threads and a fixed-size buffer and semaphores for synchronization under various conditions. In this exercise, two tasks were implemented with different numbers of producers and consumers, as well as varying processing times. In the subsequent two tasks, the issue of deadlock in semaphores was examined, and a solution for a specific problem was implemented.

1. Producer-consumer algorithm using threads that meets the following conditions:

- 1 producer
- 5 consumers
- Buffer size = 30

Covering different scenarios based on sleep times, including:

- The buffer is empty, and the thread waits for production.

I created a limited buffer with a size of 30. I also used a boolean variable, initially set to false, to stop the program after a certain time. For the producer function, each item I produced had a unique ID. The producer continued generating items until the program stopped. Before producing each item, I checked the buffer to see if it was full. If the buffer was full, I waited for 1 second and checked the buffer again. For the consumer function, I continued consuming items until the program stopped. First, I checked if the buffer was empty. If the buffer was not empty, I consumed an item. If the buffer was empty, I waited for 1 second before checking again. I set up a timer to stop all threads when it expired.
In this setup, I had 1 producer and 5 consumers running with a fixed buffer size of 30. My goal was to keep the buffer empty and have consumers waiting for items to be produced. The timing was arranged so that if a consumer found the buffer empty, it would wait for an item to be produced. Whenever an item was produced, it was consumed immediately by a consumer, ensuring that the buffer remained empty and threads were kept waiting for production.

Producer: (1.0, 3.0,) Consumer: (i, 1.0, 3.0,) Timer: (20,)

A part of the output, Full output is available in code:

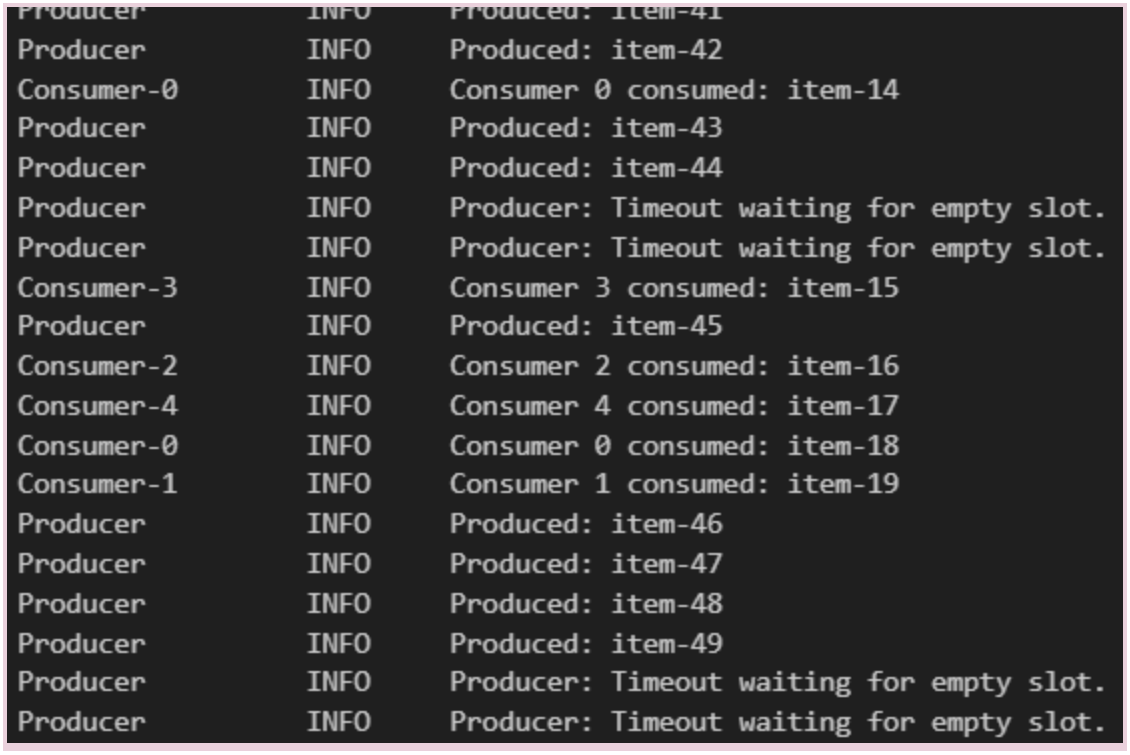
```
Consumer-0      INFO    Consumer 0: Timeout waiting for items.
Producer        INFO    Produced: item-3
Consumer-4      INFO    Consumer 4 consumed: item-3
Consumer-2      INFO    Consumer 2: Timeout waiting for items.
Consumer-3      INFO    Consumer 3: Timeout waiting for items.
Consumer-1      INFO    Consumer 1: Timeout waiting for items.
Consumer-4      INFO    Consumer 4: Timeout waiting for items.
Producer        INFO    Produced: item-4
Consumer-2      INFO    Consumer 2 consumed: item-4
Consumer-0      INFO    Consumer 0: Timeout waiting for items.
Consumer-3      INFO    Consumer 3: Timeout waiting for items.
Producer        INFO    Produced: item-5
Consumer-1      INFO    Consumer 1 consumed: item-5
Consumer-4      INFO    Consumer 4: Timeout waiting for items.
Producer        INFO    Produced: item-6
Consumer-2      INFO    Consumer 2 consumed: item-6
Consumer-3      INFO    Consumer 3: Timeout waiting for items.
Consumer-0      INFO    Consumer 0: Timeout waiting for items.
Consumer-1      INFO    Consumer 1: Timeout waiting for items.
Producer        INFO    Produced: item-7
Consumer-4      INFO    Consumer 4 consumed: item-7
Consumer-3      INFO    Consumer 3: Timeout waiting for items.
Consumer-0      INFO    Consumer 0: Timeout waiting for items.
Producer        INFO    Produced: item-8
Consumer-2      INFO    Consumer 2 consumed: item-8
```

- The buffer is full, and the producer waits for consumption.

Here we have 1 producer and 5 consumers with a fixed buffer size of 30. The goal is for the buffer to be full and for the producer to wait for the consumers to take items.
We will adjust the timings so that the producer produces several items and puts them into the buffer. The consumers will consume these items at a slower rate. This will cause the buffer to fill up. When the producer wants to produce a new item, it will find the buffer full and will have to wait. So, buffer is full, and producer is waiting for consumers to take items.

```
Producer: (0.1, 0.3,)        Consumer: (i, 3.0, 4.0,)        Timer: (20,)
```

A part of the output, Full output is available in code:



```
Producer        INFO    Produced: item-42
Consumer-0      INFO    Consumer 0 consumed: item-14
Producer        INFO    Produced: item-43
Producer        INFO    Produced: item-44
Producer        INFO    Producer: Timeout waiting for empty slot.
Producer        INFO    Producer: Timeout waiting for empty slot.
Consumer-3      INFO    Consumer 3 consumed: item-15
Producer        INFO    Produced: item-45
Consumer-2      INFO    Consumer 2 consumed: item-16
Consumer-4      INFO    Consumer 4 consumed: item-17
Consumer-0      INFO    Consumer 0 consumed: item-18
Consumer-1      INFO    Consumer 1 consumed: item-19
Producer        INFO    Produced: item-46
Producer        INFO    Produced: item-47
Producer        INFO    Produced: item-48
Producer        INFO    Produced: item-49
Producer        INFO    Producer: Timeout waiting for empty slot.
Producer        INFO    Producer: Timeout waiting for empty slot.
```

- Both production and consumption occur simultaneously.

In this situation, both production and consumption happen, creating a balance.
Producer produces items and puts them into buffer, while consumers take items from

buffer. This prevents buffer from becoming too full or too empty because, producer keeps producing, and consumers consume at a balanced rate.

```
Producer: (0.2, 0.5,)        Consumer: (i, 1.5, 3.0,)        Timer: (20,)
```

A part of the output, Full output is available in code:

```
Producer        INFO    Produced: item-37
Consumer-3      INFO    Consumer 3 consumed: item-28
Consumer-2      INFO    Consumer 2 consumed: item-29
Producer        INFO    Produced: item-38
Consumer-4      INFO    Consumer 4 consumed: item-30
Producer        INFO    Produced: item-39
Producer        INFO    Produced: item-40
Consumer-0      INFO    Consumer 0 consumed: item-31
Producer        INFO    Produced: item-41
Producer        INFO    Produced: item-42
Consumer-1      INFO    Consumer 1 consumed: item-32
Producer        INFO    Produced: item-43
Consumer-2      INFO    Consumer 2 consumed: item-33
Consumer-3      INFO    Consumer 3 consumed: item-34
Producer        INFO    Produced: item-44
Producer        INFO    Produced: item-45
Producer        INFO    Produced: item-46
Consumer-4      INFO    Consumer 4 consumed: item-35
Producer        INFO    Produced: item-47
Consumer-0      INFO    Consumer 0 consumed: item-36
Producer        INFO    Produced: item-48
Producer        INFO    Produced: item-49
Consumer-1      INFO    Consumer 1 consumed: item-37
Consumer-2      INFO    Consumer 2 consumed: item-38
Producer        INFO    Produced: item-50
Consumer-4      INFO    Consumer 4 consumed: item-39
Consumer-3      INFO    Consumer 3 consumed: item-40
Producer        INFO    Produced: item-51
Producer        INFO    Produced: item-52
```

2. Producer-consumer algorithm using threads that meets the following conditions:

- 3 producers
- 5 consumers

- Buffer size = 30

Covering different scenarios based on sleep times, including:

In this section, managing timings is more challenging because we have 3 producers:

- The buffer is empty, and the thread waits for production.

A producer-consumer scenario using threads, where producers create items and acquired semaphore while consumers take items from are rileasing semaphores. The semaphore has a limited size, causing producers to wait if it's full and consumers to wait if it's empty. A timer thread is included to stop all threads after 20 seconds. Finally, the program ensures that all threads finish before terminating, indicating the end of the process.

We set the timing so that initially, if a consumer encounters an empty buffer, it will wait. After that, any items produced by the producers will be consumed by the consumers until the buffer is empty. The difference from the previous part is the number of producers.

Producer: (i, 1.0, 2.0,) Consumer: (i, 1.0, 0.5,) Timer: (20,)

A part of the output, Full output is available in code:

```
Consumer-0      INFO    Consumer 0: Timeout waiting for items.
Consumer-2      INFO    Consumer 2: Timeout waiting for items.
Consumer-4      INFO    Consumer 4: Timeout waiting for items.
Consumer-3      INFO    Consumer 3: Timeout waiting for items.
Producer-2      INFO    Produced: item-9
Consumer-1      INFO    Consumer 1 consumed: item-9
Producer-1      INFO    Produced: item-10
Consumer-0      INFO    Consumer 0 consumed: item-10
Consumer-2      INFO    Consumer 2: Timeout waiting for items.
Consumer-4      INFO    Consumer 4: Timeout waiting for items.
Producer-0      INFO    Produced: item-11
Consumer-1      INFO    Consumer 1 consumed: item-11
Consumer-3      INFO    Consumer 3: Timeout waiting for items.
Consumer-0      INFO    Consumer 0: Timeout waiting for items.
Producer-2      INFO    Produced: item-10
Consumer-2      INFO    Consumer 2 consumed: item-10
Consumer-4      INFO    Consumer 4: Timeout waiting for items.
```

- The buffer is full, and the producer waits for consumption.

Here we have 3 producers and 5 consumers with a fixed buffer size of 30. We want for buffer to be full and for producers to wait for consumers to take items. We will adjust the timings so that the producers produce several items and put them into the buffer. The consumers will consume these items at a slower rate. This will cause the buffer to fill up. When the producers want to produce a new item, they will find the buffer full and will have to wait. So, buffer is full, and producers are waiting for consumers to take items.

```
Producer: (i, 0.1, 0.3,)        Consumer: (i, 3.0, 4.0,)        Timer: (20,)
```

A part of the output, Full output is available in code:

```
Producer-0      INFO    Producer: Timeout waiting for empty slot.
Producer-1      INFO    Producer: Timeout waiting for empty slot.
Consumer-1      INFO    Consumer 1 consumed: item-6
Producer-2      INFO    Produced: item-18
Consumer-0      INFO    Consumer 0 consumed: item-7
Producer-0      INFO    Produced: item-17
Consumer-3      INFO    Consumer 3 consumed: item-7
Producer-1      INFO    Produced: item-16
Consumer-2      INFO    Consumer 2 consumed: item-8
Producer-2      INFO    Produced: item-19
Producer-0      INFO    Producer: Timeout waiting for empty slot.
Producer-1      INFO    Producer: Timeout waiting for empty slot.
Producer-2      INFO    Producer: Timeout waiting for empty slot.
Producer-0      INFO    Producer: Timeout waiting for empty slot.
```

- Both production and consumption occur simultaneously.

In this situation, we have 3 producers and 5 consumers, creating a balance between production and consumption. Each producer makes items and puts them into the buffer, while the consumers take items from the buffer. This process helps keep buffer from getting too full or too empty because producers continuously produce, and consumers consume at a balanced rate.

```
Producer: (i, 0.2, 0.4,)        Consumer: (i, 0.45, 0.6,)       Timer: (20,)
```

A part of the output, Full output is available in code:

```
Producer-1          INFO     Produced: item-61
Producer-2          INFO     Produced: item-60
Consumer-0          INFO     Consumer 0 consumed: item-58
Producer-0          INFO     Produced: item-62
Producer-1          INFO     Produced: item-62
Consumer-3          INFO     Consumer 3 consumed: item-57
Consumer-2          INFO     Consumer 2 consumed: item-58
Producer-2          INFO     Produced: item-61
Consumer-1          INFO     Consumer 1 consumed: item-59
Consumer-4          INFO     Consumer 4 consumed: item-59
Producer-0          INFO     Produced: item-63
Consumer-0          INFO     Consumer 0 consumed: item-58
Producer-1          INFO     Produced: item-63
Producer-2          INFO     Produced: item-62
Consumer-3          INFO     Consumer 3 consumed: item-60
Producer-1          INFO     Produced: item-64
Consumer-2          INFO     Consumer 2 consumed: item-60
Producer-0          INFO     Produced: item-64
Consumer-1          INFO     Consumer 1 consumed: item-59
```

3. Creating a Deadlock Using Semaphores

A scenario that leads to a deadlock using two threads, thread1 and thread2.

First, thread1 acquires a semaphore (sem1) and prints a message. It then waits for 1 second and tries to acquire another semaphore (sem2). Meanwhile, thread2 acquires sem2 and prints its message, then waits for 1 second before trying to acquire sem1.

Thread1 has sem1 and is waiting for sem2, while thread2 has sem2 and is waiting for sem1. Because of this, neither thread can continue. It is a deadlock, where both threads are stuck waiting for each other. After 5 seconds, deadlock occurred.

```
Thread 1 acquired sem1
Thread 2 acquired sem2
Thread 2 waiting for sem1...
Thread 1 waiting for sem2...
Deadlock occurred.
```

4. Ensuring Sequential Execution of Two Processes (P1 Before P2)

We use threads to run two functions, P1 and P2, in a specific order. P1 starts first and prints a message, which shows P1 is waiting 2 seconds and P2 is waiting. Then, P1 waits for 2 seconds to simulate some work before indicating it has finished. After completing its task, P1 releases a semaphore, signaling that P2 can proceed. P2 starts and immediately waits for this semaphore, meaning it cannot continue until P1 finishes. Once P1 finishes and releases the semaphore, P2 can proceed and print its message. The main part starts both threads, ensuring that P1 finishes before P2. Finally, it confirms that P1 completed its work before P2, as intended.

```
P2 waiting for P1 to finish...
2 seconds sleep to ensure that P2 is not doing anything before P1
P1 is running first...
P1 finished, releasing semaphore for P2.
P2 is now running after P1.
P1 finished before P2, as required.
```