OS-2 Project Documentation

Bounded Buffer Problem

The problem:

There is a buffer of N slots, and each slot is capable of storing one unit of data. There are two processes running, namely, producer and consumer, which are operating on the buffer. A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. In our project, we can find **Multiple Producers** and **Multiple Consumers**.

Solution:

One solution of this problem is to use semaphores. The semaphores which will be used here are:

- mutex, a binary semaphore which is used to acquire and release the lock.

- empty, a counting semaphore whose initial value is the number of slots in the buffer, since initially all slots are empty.

- full, a counting semaphore whose initial value is 0.

At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer

Solution pseudocode:

1-The pseudocode of the producer function :

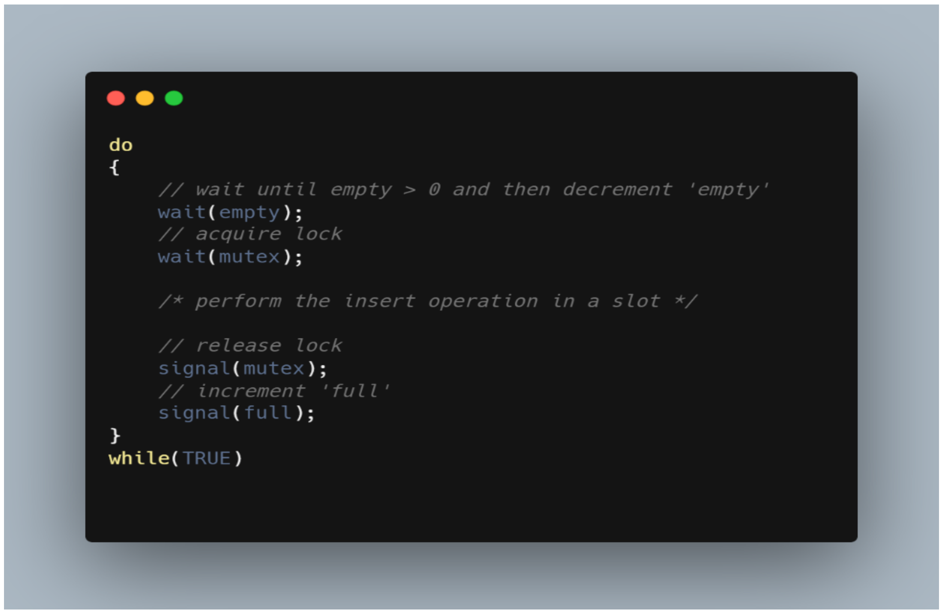
1-The producer first waits until there is at least one empty slot.

2-The Producer decrements the empty semaphore.

3-The producer acquires lock (mutex) on the buffer.

4-After performing the insert operation, the lock (mutex) is released.

5-The producer increments the value of full semaphore by 1.



```
do
{
    // wait until empty > 0 and then decrement 'empty'
    wait(empty);
    // acquire lock
    wait(mutex);

    /* perform the insert operation in a slot */

    // release lock
    signal(mutex);
    // increment 'full'
    signal(full);
}
while(TRUE)
```

2-The pseudocode for the consumer function:

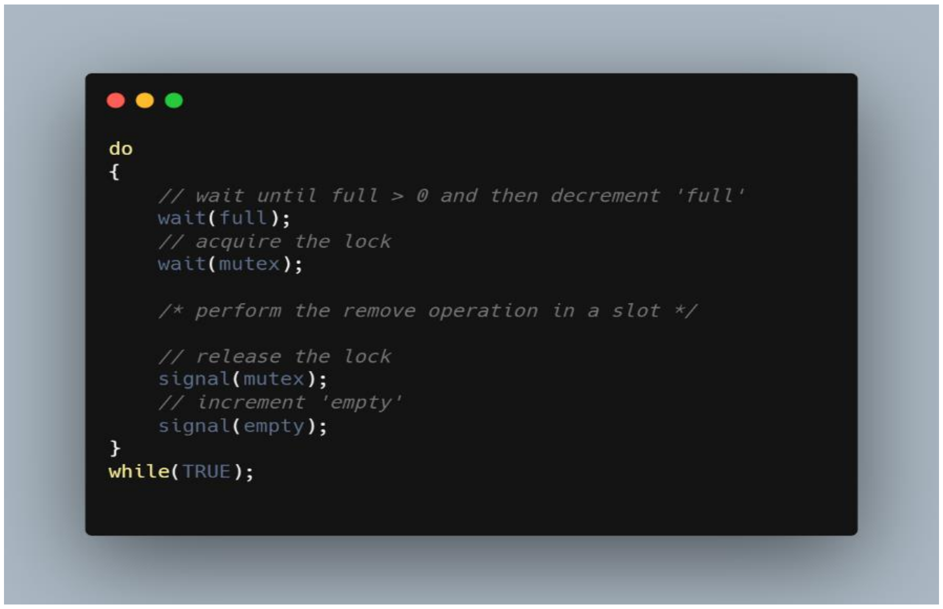
1-The consumer waits until there is at least one full slot in the buffer.

2-The consumer decrements the full semaphore.

3-The consumer acquires lock (mutex) on the buffer.

4-After performing the removal operation, the lock (mutex) is released.

6-The consumer increments the value of empty semaphore by 1



```
do
{
    // wait until full > 0 and then decrement 'full'
    wait(full);
    // acquire the lock
    wait(mutex);

    /* perform the remove operation in a slot */

    // release the lock
    signal(mutex);
    // increment 'empty'
    signal(empty);
}
while(TRUE);
```

Deadlock

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

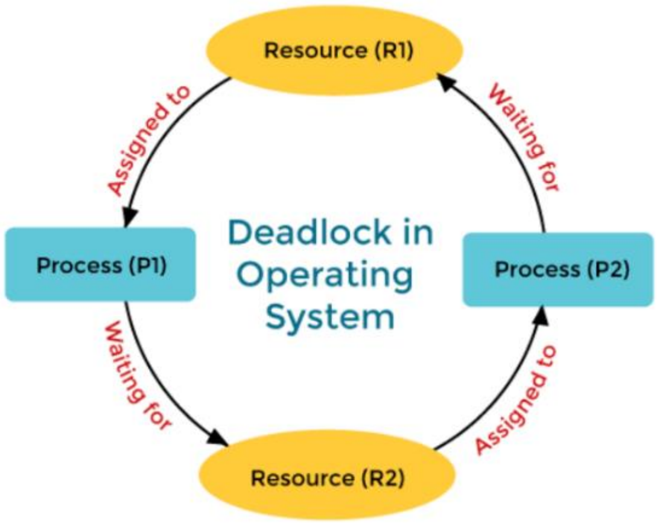
Examples of Deadlock:

- When two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.

- Two cars crossing a single-lane bridge from opposite directions.

- A person going down a ladder while another person is climbing up the ladder.

Similar situations occur in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).



Let's take an instance, Process (P1) is holding Resource (R1) and waiting for the Process (P2) to acquire the Resource (R2), and Process (P2) is waiting for the Resource (R1). Thus, Both the Process (P1) and (P2) is in a deadlock situation.

The following four conditions may occur the condition of deadlock:

1) Mutual Exclusion --- 2) Hold and Wait --- 3) No Preemption --- 4) Circular Wait:

<u>How did we solve Deadlock:</u>

1-Deadlock prevention or avoidance, by zooming into each category individually. We need to ensure that all information about resources which process will need are known to us prior to execution of the process. We use Banker’s algorithm in order to avoid deadlock.

2-Deadlock detection and recovery, by letting the deadlock occur, then do preemption to handle it.

3-Ignore the problem altogether, if the deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and Linux take.

In our bounded buffer problem:

Deadlock occurs if we replace two statements wait(mutex); & wait(full);

Or wait(mutex); & wait(empty);

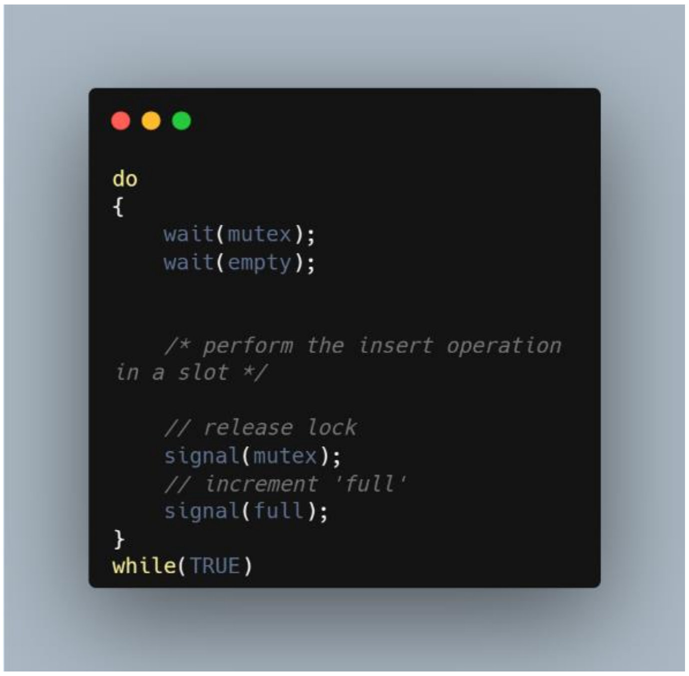
We must be care about sentences arrangement because if we write producer and consumer like above codes, this will cause deadlock

In more detail ….

If we write wait(mutex); then wait(full); , by this consumer will acquire the lock and enter the critical section without checking if there are any items to consume and we suppose there is no item to be consumed . by this we have the lock and still in the critical section, but we cannot do our operation and consumer is waiting producer to insert any item.

on the other side, if we write wait(mutex); then wait(empty); by this producer will acquire the lock and enter the critical section without checking if there is any slot to produce item in it and we suppose that buffer is full , in this case producer can not insert any item and it is waiting the consumer to consume items .

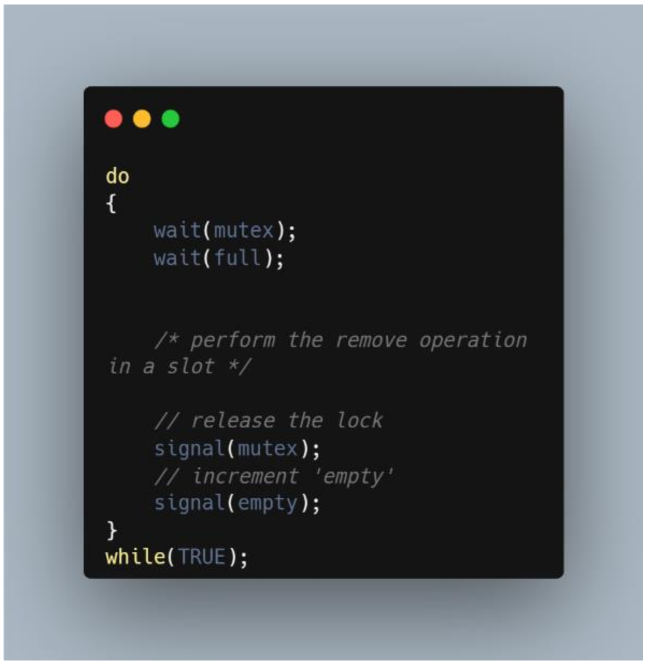
Now we can see that both are waiting for each other.



```
do
{
    wait(mutex);
    wait(empty);


    /* perform the insert operation
in a slot */

    // release lock
    signal(mutex);
    // increment 'full'
    signal(full);
}
while(TRUE)
```



```
do
{
    wait(mutex);
    wait(full);


    /* perform the remove operation
in a slot */

    // release the lock
    signal(mutex);
    // increment 'empty'
    signal(empty);
}
while(TRUE);
```

Starvation

Starvation happens when a low priority program requests a system resource but cannot run because a higher priority program has been employing that resource for a long time. When a process is ready to start executing, it waits for the CPU to allocate the necessary resources. However, because other processes continue to block the required resources, the process must wait indefinitely.

How did we solve Starvation:

1.The resource allocation priority scheme should contain concepts such as aging, in which the priority of a process increases the longer it waits. It prevents starvation.

2.An independent manager may be used for the allocation of resources. This resource manager distributes resources properly and tries to prevent starvation.

3.Random process selection for resource allocation or processor allocation should be avoided since it promotes starvation.

In our bounded buffer problem

The Bounded-Buffer problem has several variations, all involving priorities.

-The simplest one, referred to as the first producers & Consumers problem, requires that no producers will be kept waiting unless a consumer has already obtained permission to use the shared object. In other words, no producer should wait for other producers to finish simply because a consumer is waiting. (Priority for producers)

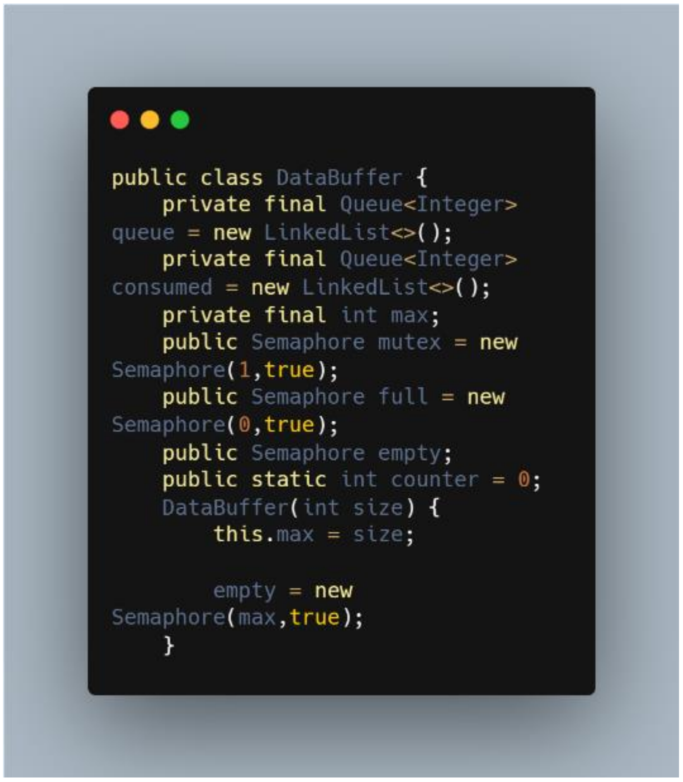
In this case, Consumer may starve.

-The second producers & Consumers problem ,if a consumer is waiting to access the object, no new producers may start producing. (Priority for consumer)

In this case, Producer may starve.

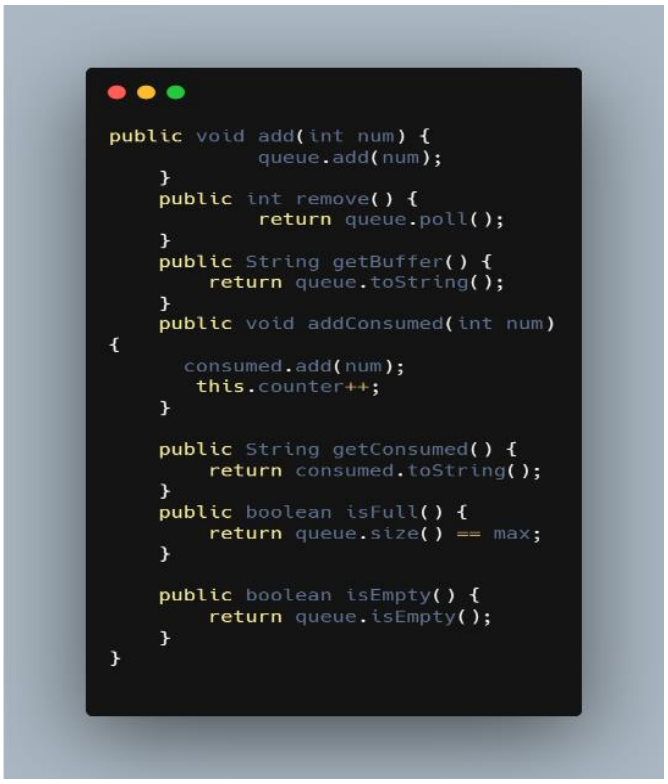
Solution:

We can solve it by using FIFO Queue, “Thread that arrives first is served first”



```
public class DataBuffer {
    private final Queue<Integer>
queue = new LinkedList<>();
    private final Queue<Integer>
consumed = new LinkedList<>();
    private final int max;
    public Semaphore mutex = new
Semaphore(1,true);
    public Semaphore full = new
Semaphore(0,true);
    public Semaphore empty;
    public static int counter = 0;
    DataBuffer(int size) {
        this.max = size;

        empty = new
Semaphore(max,true);
    }
```



```
public void add(int num) {
            queue.add(num);
    }
    public int remove() {
            return queue.poll();
    }
    public String getBuffer() {
        return queue.toString();
    }
    public void addConsumed(int num)
{
      consumed.add(num);
       this.counter++;
    }

    public String getConsumed() {
        return consumed.toString();
    }
    public boolean isFull() {
        return queue.size() == max;
    }

    public boolean isEmpty() {
        return queue.isEmpty();
    }
}
```

Real world application: In our real-world application, we suppose that we have a Factory , an Inventory, and a Retailer. The Factory is our Producer (produces different kinds of items), The Inventory is our Buffer, in which we store items produced by the Factory. The Retailer is our Consumer who takes an item from the Inventory.