Operating system 2 Project Documentation

Bounded Buffer Problem:

Also known as the Producer-Consumer problem. In this problem, there is a buffer of n slots, and each buffer can store one unit of data. There are multiple Producers and multiple Consumers. The producers try to insert data and the consumers try to remove data

This problem occurs when many threads of execution try to access the same shared resources at a time. There are N producers to produce data and K consumers to consume data to shared resources, so in this documentation we will solve this problem.

Pseudocode:

1- Make a producer class
2- Make a Consumer class
3- Define two attributes Queue and the size.
4- In function produce( ) check if the queue is full
    a. Wait( );
    b. If not full:
        i. Add item (Ticket)
        ii. NotifyAll( );
5- In function consume ( ) check if the queue is empty
    a. Wait( );
    b. If not empty
        i. Consume item (Ticket)
        ii. notifyAll( );
6- In any case of failure will catch and print the reason of the failure.
7- Create objects from producer and consumer
8- Create object from Thread
9- Start Thread

Consumer class

```
    public void run() {
        try {
            consume();
        } catch (InterruptedException ex) {
            System.out.println("interupted at consumer");
            ex.printStackTrace();
        }
    }

    private void consume() throws InterruptedException {
        synchronized (queue) {
            while (queue.isEmpty()) {
                System.out.println("underflow");
                System.out.println(Thread.currentThread().getName() + " is waiting , size: " + queue.size());
                queue.wait();
            }
            Thread.sleep(500);
            int i = (Integer) queue.remove(0);
            System.out.println("consumed: " + i);
            queue.notifyAll();
        }
    }
}
```

Producer class

```
@Override
public void run() {
    int count = 1;
    try {
        produce(count++);
    } catch (InterruptedException ex) {
        System.out.println("interupted at producer");
        ex.printStackTrace();
    }
}

private void produce(int i) throws InterruptedException {
    synchronized (queue) {
        while (queue.size() == size) {
            System.out.println("overflow");
            System.out.println(Thread.currentThread().getName() + " is waiting,size: " + queue.size());
            queue.wait();
        }
        Thread.sleep(500);
        queue.add(i);
        System.out.println("Produced: " + i);
        queue.notifyAll();
    }
}
```

Example when the deadlock can occur:

When we request to enter and lock the critical section, a deadlock may occur if we do not open the critical section, any producer or consumer will request to enter the critical section, will find that the critical section is busy but there is no one in the critical section.

- A real-world example would be traffic, which is going only in one direction.

How to solve deadlock:

In wait and notify after we exit from the critical section then notify release the critical section to allow anyone to enter to it.

Example when the starvation can occur:

we have producers and consumers so if there is any producer who wants to get into the critical section, he must ask to produce and then lock the critical section to prevent any other consumer or producer to write so if we lock the producing room no one can produce from this resource but after this producer ends. Any producer else can produce to it but, the consumer can't consume.

- Cinema theater with a single check-in counter

How to solve starvation:

In wait and notify we have the turn that gives the producer or the consumer the right to access the critical section, so that any producer or consumer can access the critical section without starvation happened.

Real world application:

In the real world each producer or consumer has a turn that gives them the right to access the critical section and do whatever they want to do and no one can enter the critical section if anyone else inside it.

The real-world application is example for cinema theater.

The producer represents the add ticket function.

- If the tickets are full, we can't add more tickets until somebody books a ticket.

The consumer represents the book ticket function

- If there are no tickets left, no one can book a ticket until we add more tickets.