Operating system 2 Project – Cover sheet

Project Title: Bounded Buffer Problem Group: 9

Discussion time : 9:20 AM Instructor: Islam Gamal

ID	Name(Arabic)	Bounce	Minus	Total Grade	Comment
201900430	عبدالرحمن محمد احمد سليمان				
201900829	مصطفي محمد اسماعيل المتولي				
201900631	محمد اسماعيل منصور عبدالله				
201900832	مصطفي محمود عبدالحميد عبدالتواب				
201900846	منار اشرف عطيه محمد				
201900780	محمود فوزي محمود عبدالغفار				
201900324	زياد شعبان محمود عبدالفتاح				

Critrial		Grade	Team Grade	Comment
Documentation	Solution pseudocode	1		
	Examples of Deadlock	1		
	How did solve deadlock	1		
	Examples of starvation	1		
	How did solve starvation	1		
	Explanation for real world application and how did apply the problem	1		
GitHub	Upload project files	2		
	Submitted before discussion time (shared GitHub project link with TA and Dr)	1		
	Only one contribution	-1		
Implementation	Run correctly (correct output)	5		
	Run but with incorrect output	-3		
	Not run at all (error and exceptions)	-8		
	Free from Deadlock	3		
	Free from deadlock in some cases and not free in other cases	-2		
	Free from Starvation	2		
	Free from Starvation in some cases and not free in other cases	-1		
	Apply problem to real world application	6		
Total	Total grade for Team	25		
	Total Team Grade(after adjustment)	25		
Bounce	Multithreading GUI Based Java Swing	+5		
	Multithreading GUI Based Java Swing(adjustment)			
	Multithreading GUI Based JavaFX	+10		
	Multithreading GUI Based JavaFX(adjustment)			
	Bounce Graphic and animation	+5		
Total with Bounce	Total Team Grade			
	Total Team Grade(after adjustment)			

Documentation of OS2

Introduction

- the producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem.
- The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue .
- The producer's job is to generate data, put it into the buffer, and start again.At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.
- The buffer will have a max value. (The maximum amount of data it can store)

Problem : To make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.
The previously mentioned problems are solved by the team using semaphores and a FIFO queue in order to describe the behaviors of the producer and consumer
Both the producer and consumer will have their own semaphores and separate threads

Solution : The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

What is The Deadlock ?:

A deadlock in OS is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process.

Examples Of Deadlock

1- You will apply for a new job but new job require of some jobs you worked
2- All trains are stopped, waiting for another to go, though none of them move
3- two employees a,b
   a waits for b to finish his work to start his work
   b waits for a to finish its work
4- deadlock where both processes are waiting to be awakened

What is starvation? :

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.

Examples of starvation:

1- Imagine an airport with a single check-in counter and two queues, one for business class and one for economy class. As you may know the business class has the priority over the economy class.
2- kitchen have a huge order should be end first it's take much time and kitchen have another small orders

Algorithm to Solve Deadlock And Starvation :

- Empty = 0
- Full = N(Max size)
- Mutex= 1

Producer:

```
do {
    wait (full);
    wait(mutex);
    //add item
    signal(mutex);
    signal(empty);
}while(1);
```

Consumer:

```
do {
    wait (empty);
    wait(mutex);
    //consume item
    signal(mutex);
    signal(full);
}while(1);
```

- ## Solution Pseudocode

    1. Make a producer class as (Cooker) to add in buffer
    2. Make a consumer class as (Customer) to remove from buffer
    3. Make a buffer class as (Kitchen)
    4. Make a 3 semaphore Mutex = 1,empty=BufferSize ,full=0
    5. Check if item will add less than buffer size
    6. Make function insert and do mutex , empty semaphore acquire
    7. Add item in buffer
    8. do mutex , full semaphore release
    9. Make function remove and do mutex , full semaphore acquire
    10. Check if buffer not empty
    11. remove item from buffer
    12. do mutex , empty semaphore release
    13. Make a multithreads for baker and customer
    14. Start threads

Explanation for real world application and how did apply the problem:

Think of MacDonald's as a burger pipeline: cook making burgers, putting them in the bin. But if the bin fills up, the cook has to stop for a while. The consumers come to buy burgers, and take them from the bin, but if the bin is empty, they have to wait a while. The bin has a certain capacity. Similarly, memory buffers have a certain size, the "bounded buffer" idea