Operating system 2 Project – Cover sheet

Project Title : Bounded Buffer Problem

Group: 6

Discussion time :- 8:50

Instructor: islam gamal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Name(Arabic) | Bounce | Minus | Total Grade | Comment |
| 202000328 | روان اسامه محمد محمود درويش |  |  |  |  |
| 202001032 | هدي الفاتح احمد علي |  |  |  |  |
| 201900111 | احمد هاني طلعت عبد العزيز |  |  |  |  |
| 202000168 | انس سعيد احمد محمد |  |  |  |  |
| 202000962 | مي عاطف السنوسي حسن احمد |  |  |  |  |
| 202000336 | روان محمد رمضان بحيري |  |  |  |  |
| 202000164 | امين شعبان امين محمد |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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) | | |  |  | | |  |

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)

❖Producers must block if the buffer is full.

❖Consumers must block if the buffer is empty.

Pseudocode :

SET shared queue of max size = 4

SET producers , consumers , I

SET production size = 5

GET production number

COMPUTE produced item = (production size \* production number) + I

FOR I = 1 to production size

Increment I

ENDFOR

FOR current production number < production number

IF current size of shared queue != maxsize THEN

Producers will produce in queue

ELSE

Producers will wait

ENDIF

IF current size of shared queue > 0 THEN

Consumers will consume fromm queue

ELSE

Consumers will wait

ENDIF

ENDFOR

What is The 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 :

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 b , c b waits for b to finish his work to start his work. c waits for a to finish its work.

4. deadlock where both processes are waiting to be awakened.

How did solve the Deadlock?

We solve the deadlock here using a shared queue so that both producers and consumers don’t need to wait to have a resource the other one using it while the other is waiting a resource from it ; ( preventing the cicular wait ).

What is starvation?

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Examples of starvation:

Table

Description automatically generated

In the above example, the process P2 is having the highest priority and the process P1 is having the lowest priority. In general, we have a number of processes that are in the ready state for its execution. So, as time passes, if only that processes are coming in the CPU that are having a higher priority than the process P1, then the process P1 will keep on waiting for its turn for CPU allocation and it will never get CPU because all the other processes are having higher priority than P1. This is called Starvation.

2- kitchen have a huge order should be end first it’s take much time and kitchen have another small orders.

How solve the starvation :

We solve the starvation using wait and notify functions that every thread will enter the critical section regardless of it’s priority . all the threads will be notified with the exit of the thread in the critical .

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

Think of MacDonald’s as a burger pipeline : Producer ( cooker ) making burgers, putting them in the box. But if the box fills up, the producer (cooker ) has to stop for a while. The Consumer ( Customer ) come to buy burgers, and take them from the box, but if the box is empty, they have to wait a while. The box has a certain capacity. Similarly, memory buffers have a certain size, the " Bounded Buffer " idea .