

OS Documentation

Producer consumer



December 19, 2022

1. Moatasem mohamed RAGAB: 202000928
2. HANA MOHAMED ZEIN:202001039
3. AYA RAAFT MTWALY:202000184
4. EMAD MOSTAFA SERAG:202000580
5. MOHAMED IBRAHIM RAGAB:201901006
6. MOHAMED ALI MOSLH:202000804
7. YASMEEN HAMDY ABDO:201900958

OS Documentation

Solution pseudocode:

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

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: Generally

* when two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.In simple words, we can say that if two or more processes are waiting for some events to happen, which never happens, then, that is called deadlock and those processes are in deadlock state.

Diagram

Description automatically generated

Here, P1 and P2 are two processes. R1 and R2 are two resources.

Note: R –> P means process (P) has taken the resource (R). For example, R2 –> P2 means resource R2 has been allocated to process P2

Explanation

* there are two resources R1 and R2 and two processes P1 and P2.  
  Let us say R1 is allocated to P1 or P1 has taken R1. P1 is requesting for R2 but R2 is held by P2. Now, P2 is requesting for R1. So, we can say that it is kind of deadlock, or it is a deadlock situation.

Note – There are 4 necessary conditions for deadlock.

* when two trains approach each other at a crossing, both shall come to a full stop, and neither shall start up again until the other has gone.

Deadlocks occur in Producer consumer problem:

When we switch between wait(mutex) and wait(full), This is what causes:

1. **Mutual Exclusion**

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

1. **Hold and Wait**

A process waits for some resources while holding another resource at the same time.

1. **No preemption**

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

1. **Circular Wait**

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

**Which lead to deadlock occurs.**

we will show you the result when we switch between two semaphores in both producer and consumer:

Text

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

Output:

Graphical user interface, text, application, Word

Description automatically generated

Deadlocks occur in thread which id=20 threads will loop forever.

How did solve deadlock?

Generally, there are three ways to solve deadlock:

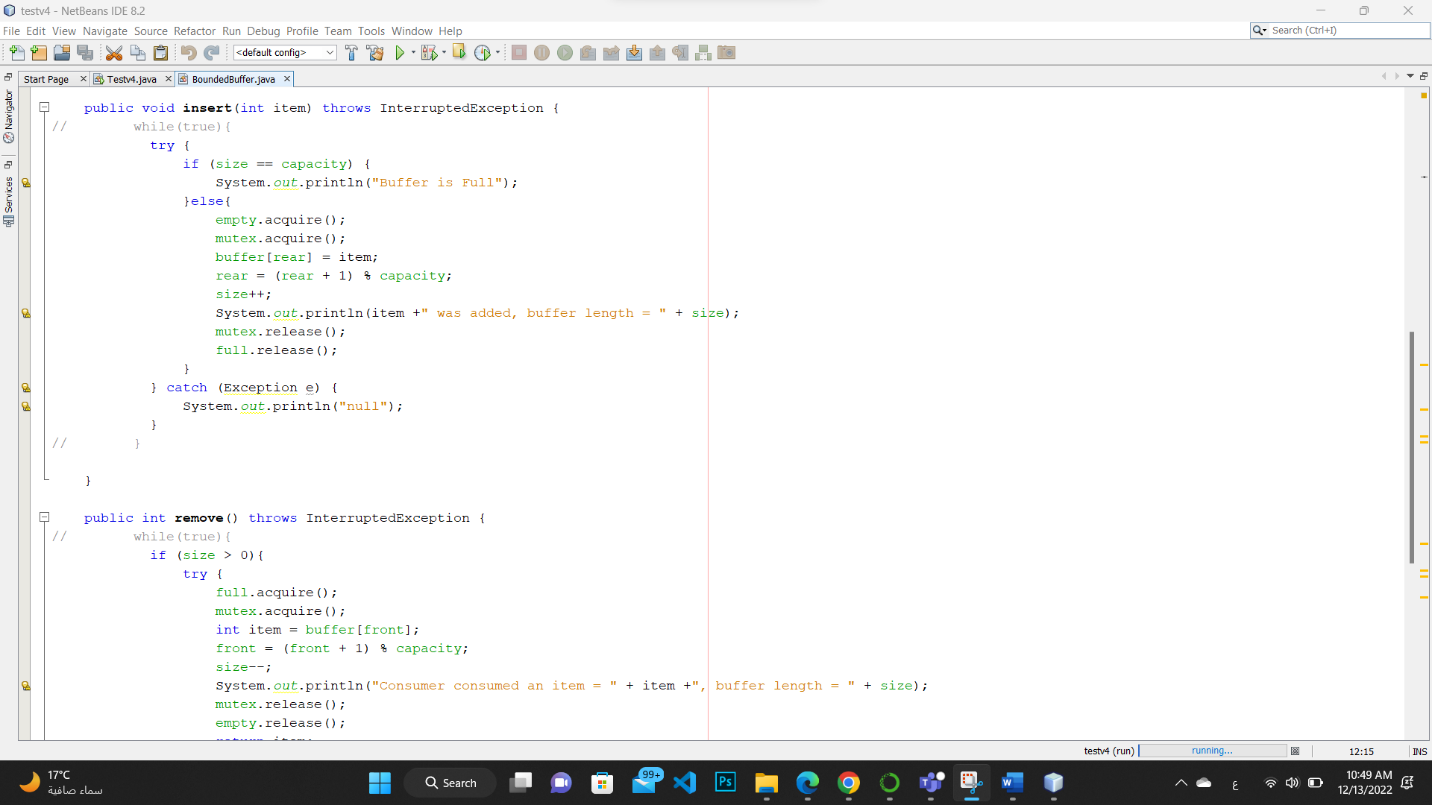
1. Ensure that the system will never enter a deadlock state:
   1. Deadlock prevention
   2. Deadlock avoidance
2. Allow the system to enter a deadlock state and then recover
3. Ignore the problem and pretend that deadlocks never occur in the system

In producer consumer we solve deadlock uses:

Deadlock Prevention:

* By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.
* Invalidate one of the four necessary conditions for deadlock:

circular-wait condition — is considered a practical solution by invalidating one of the necessary conditions: One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each thread requests resources in increasing order of enumeration.



What is starvation in operating system?

**Starvation** or indefinite blocking is a phenomenon associated with the Priority scheduling algorithms. A process that is present in the ready state and has low priority keeps waiting for the CPU allocation because some other process with higher priority comes with due respect time. Higher-priority processes can prevent a low-priority process from getting the CPU.

**What Causes Starvation in OS**

Here are a few reasons why starvation in OS occurs:

* In starvation, a process with low priority might wait forever if the process with higher priority uses a processor constantly.
* Because the low-priority processes are not interacting with resources the deadlock does not occur but there are chances of starvation as the low-priority processes are kept in a wait state.
* Hence starvation is precisely a **fail-safe method**, that is it prevents deadlock temporarily, but it affects the system in general.
* The important cause of starvation might be that there are not enough resources to provide for every resource.
* If a process selection is random then there can be a possibility that a process might have to wait for a long duration.
* Starvation can also occur when a resource is never provided to a process for execution due to faulty allocation of resources.

Examples of starvation:

Table

Description automatically generated

the above image process has higher priority than other processes getting CPU earlier. We can think of a scenario in which only one process has very low priority (for example, 127), and we are giving other processes high priority. This can lead to indefinitely waiting for the process for CPU, which is having low priority, which leads to **Starvation.**

Starvation example in Producer consumer occur:

**Graphical user interface, application

Description automatically generated**

**When P: empty, Q: empty, R: full, S: full: results in starvation. Its producer at P condition blocks the process that there are no items to give, and R conditions blocks that there is no item to consume because it is already full it means something blocking high-priority processes keep executing and low priority processes get blocked for an indefinite time.**

**Output of starvation in producer consumer**

**Graphical user interface, text, application, Word

Description automatically generated**

**Starvation occurs when thread which id =13 full the buffer and prevented consumer to consume from buffer.**

How did solve starvation?

Here are the following ways in which the starvation situation in OS can be handled:

* The allocation of resources by CPU should be taken care of by a freelance manager to ensure that there is an even distribution of resources.
* Random choice of process method should be avoided due to which starvation occurs.
* The **aging criteria of processes** should be taken into consideration while resource allocation to avoid starvation.
* **Scheduling algorithm** with priority queue can also be used to handle starvation.
* If the random technique is to be used, then use it with a priority queue to handle starvation.
* **Multilevel feedback queue** can also be used for avoiding starvation in the operating system.

**We used to solve starvation in Producer consumer Queue linked list.**

And we use semaphore in a right way.

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

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

Let’s say in our Example, the producer is Mr. Simpson whose job is to bake Pizza and consumer is Joey Tribune who consumes Pizza at the same time. Both share common store tables to interact with each other.

The problem is to make sure that the Simpson won't try to add more pizza-on-store tables if it's full. He needs to wait until Joey leave store.

1. It simplifies development process. you can Code Producer and Consumer independently and concurrently, they just need to know shared object.
2. Producer doesn't need to know about who consumer or how many consumers is are there. Same is true with Consumer.
3. Producer and Consumer can work with different speed. In fact, by monitoring consumer speed, one can introduce more consumer for better utilization.

