ESPINA, Allain Daniel S.

3ITF

Asst. Prof. Bernard G. Sanidad, DIT, DBA

September 11, 2023

HOMEWORK 1: DEADLOCK

1. Describe the difference between deadlock, race, and starvation.

A **Deadlock** occurs whenever two or more processes end up waiting for specific resources that may still be utilized by some other process, therefore, blocking all of the said processes and preventing them from executing as a result. For example, two students attend a class in which they will have to answer a quiz. However, Student 1 only brought one piece of paper and forgot to bring his own pen. On the other hand, Student 2 only brought a pen and forgot to bring any paper for the quiz. This results to both of them not being able to accomplish anything as they both need a pen and a paper to be able to take the quiz. Similarly, two processes which currently hold a resource that the other one needs cannot execute their respective tasks.

Meanwhile, a **Race** is an instance that happens when a system executes two or more processes concurrently instead of sequentially when the outcome of their execution depends on the order in which they are successfully accomplished. A Race can be related to a situation wherein one store clerk tries to accommodate several customers all at once, which may lead to mix-ups when it comes to customer requests and concerns.

Finally, a **Starvation** is a situation wherein processes cannot proceed as there are currently no available resources that they must utilize for the execution of their assigned task, just like whenever you need to repair something, but you have no tools to do so. Although a Starvation and a Deadlock may seem similar at first glance, the main difference between the two is that processes in a Deadlock keep on waiting for each other to be finished for them to proceed while a Starvation occurs because low priority processes are being blocked because of the continuous execution of higher priority processes. Both instances of Starvations and Deadlocks may also be a result of a Race that occurred within a system.

2. Identify several causes of system deadlock and livestock.

A Deadlock, characterized by the stalling of two or more processes waiting for the other processes to release their required resources, can only occur when one of these four conditions are met: (1) **Mutual Exclusion**, (2) **Hold and Wait**, (3) **No Preemption**, and (4) **Circular Wait**.

The first condition for a Deadlock to occur, **Mutual Exclusion**, is an instance when a resource required by more than one process is non-shareable or can only be used by a single process, one at a time. Second is the concept of **Hold and Wait**, wherein processes hold particular resources at a time while waiting for other processes to release another resource it also needs to execute its task. The third requirement is **No Preemption**, which simply means that no resource can be acquired by another process unless it is voluntarily released by the process which is currently utilizing it. Last but not the least is the occurrence of a **Circular Wait**, wherein a set of processes wait for each other to release their required resource in a circular manner, much like how a circular queue in data structures work. For reference, let us say that there are three processes, process 1, 2, and 3. process 2 depends on the completion of process 1, process 3 depends on the completion of process 2, and process 1 depends on the completion of process 3.

3. Differentiate deadlock prevention and deadlock avoidance.

A **Deadlock Prevention**, as the name implies, prevents any of the four required conditions for a deadlock to occur to happen at all, maintaining the smooth flow of the entire system. However, doing so may be costly as preventing these four requirements from occurring may require different corresponding services.

On the other hand, **Deadlock Avoidance** works similarly but not entirely the same as Deadlock Prevention. In Deadlock Avoidance, the process is a bit more complicated as the system must know almost everything about the processes and resources of the entire system for it to decide whether to let one process to proceed or not, in order to prevent a Deadlock from happening. However, implementing Deadlock Avoidance will trade off the system's performance in exchange for the reliability and validity of the required data.

4. Describe how to detect and recover from deadlock and starvation.

Deadlock Detection involves identifying situations wherein two or more processes are halted because they are all waiting for other processes to release their required resources. A system can identify Deadlocks by utilizing system information, similar to the Deadlock Avoidance Process, for it to then be able to initiate the corresponding procedures to resolve the instance of the Deadlock by implementing varying Deadlock Recovery procedures. Deadlock Detection makes use of different methods to detect the occurrence of a Deadlock, one of which is the ***Resource Allocation Graph (RAG)*** that illustrates the resources allocated to the corresponding processes at a time, which is used to detect cycles within a system. Another common method is the use of ***Wait-For Graphs (WFG)*** which illustrates the relationships or dependencies of each process.

Deadlock Recovery, on the other hand, occurs shortly right after a Deadlock has been detected by the system. There are various ways for a system to recover from the occurrence of a Deadlock, some of which are (1) ***Process Termination***, (2) ***Resource Preemption***, (3) ***Priority Inversion***, (4) ***Rollback***, and (5) ***Concurrency Control***. The first method is called ***Process Termination***, a straightforward solution to a Deadlock as it simply terminates or "kills off" the processes involved in the Deadlock, one by one. Second is ***Resource Preemption***, wherein the system instead identifies the resources involved. Once identified, these resources will then be assigned to other processes first while the processes these were taken from will be suspended for the meantime until the resources are available once again, making this method slightly more complicated. Third is ***Priority Inversion***, wherein the processes who currently holds the required resources are given higher priority as opposed to those who are still in waiting. One major drawback of this method is that the system may then instead experience a Starvation. Fourth on the list is the ***Rollback*** method, wherein the system "restores" itself to a state wherein the Deadlock has not yet occurred, making use of system logs to determine which version of itself should it revert back to. Lastly is the ***Concurrency Control*** method which utilizes several mechanisms to ensure that processes can only access a specific resource at a time, ensuring that all processes will not interfere with the other as they proceed with the execution of their task or as they wait for a resource to be released.

5. Describe the concept of race and how to prevent it.

Operating System (OS) **Races**, as mentioned in the first item, is an event that occurs whenever a system executes two or more processes simultaneously instead of in a specific order, when the outcome of their execution depends on the order in which they are completed. An example of which is when you and a family member waters your plants at home. Hypothetically, both of you water the plants at different times but are not aware that the other one is doing the same thing as well. As a result, the plant may not grow as you expect it to because it receives two much water, which is the same thing that occurs to a system during an OS Race.

In order to prevent OS Races, several methods can be employed. The use of ***Locks and Semaphores***, which are synchronization features that allows different processes to access a shareable resource anytime. Preparing for the possibility of ***Concurrency*** during the ***Design*** process of an application is a good practice to ensure your application's capability to prevent and handle such problems in the future. The process of ***Message Passing*** can also help you prevent Races as this ensures that only one process can access a system message at the same time, whenever they try to communicate. The use of ***Thread-Safe Libraries***, and uninterrupted ***Atomic Operations*** for shared data manipulation, are also considered as good practice. Opting to refrain from using ***Global Variables*** also prevents the occurrence of an OS Race as it prohibits certain processes from accessing and/or modifying it simultaneously.

References:

Agrawal, D. (2022, February 24). *Deadlock in os*. Scaler Topics. https://www.scaler.com/topics/operating-system/deadlock-in-os/

Deadlock detection and recovery. Online Courses and eBooks Library. (n.d.). https://www.tutorialspoint.com/deadlock-detection-and-recovery

GeeksforGeeks. (2023a, March 18). *Deadlock detection and recovery*. GeeksforGeeks. https://www.geeksforgeeks.org/deadlock-detection-recovery/

GeeksforGeeks. (2023b, July 19). *Introduction of deadlock in operating system*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/

Mansi. (2022, November 9). *What is starvation in OS (operating systems)?*. Scaler Topics. https://www.scaler.com/topics/starvation-in-os-operating-system/

Mundada, M. (2022, March 11). *Race condition in os*. Scaler Topics. https://www.scaler.com/topics/race-condition-in-os/

Nilesh, K. (2023, July 1). *What is Race Condition in OS?*. Coding ninjas studio. https://www.codingninjas.com/studio/library/what-is-race-condition-in-os

What is a race condition?. Baeldung on Computer Science. (2023, May 5). https://www.baeldung.com/cs/race-conditions