Take Home Quiz 2 (Section 12)

You may use information from any books or external websites. However, Plagiarism among the individual or extensive use of ChatGPT is not expected and will be heavily penalized.

You have to provide answers in required fields of this form and finally submit it.

Deadline: 3 May, 3 pm (24 hours).

If anyone fails to submit within the deadline using this form he/she will get 0. No make up or alternate will be allowed. Mail submissions will be rejected automatically.

ms.rodsy.tahmid@g.bracu.ac.bd Switch account   Draft saved

* Indicates required question

Email *

✓ Record **ms.rodsy.tahmid@g.bracu.ac.bd** as the email to be included with my response

ID *

20101021

Name *

Ms Rodsy Tahmid

Section *

12

Consider a scenario where two processes each hold a resource and are waiting for a resource held by the other process. **How** would you identify if this situation leads to a deadlock using a resource allocation graph? **Explain your reasoning step by step.**

Ans 1) A set of processes is in a deadlocked state if every process in the set is waiting for an event that can be caused only by another process in the set.

Step-1: Resource Allocation Graph (RAG) is represented by each process as a circle and each resource type as a rectangle.
Then we'll draw directed edges from processes to the resources they are waiting for (request edges) and from resources to the processes that have been allocated those resources (assignment edges).

Step-2: After that, we'll identify the Processes and Resources Involved:
Two Processes: P1, P2
Two Resources: R1, R2
Edges: P1 → R2, R2 → P2, P2 → R1, R1 → P1

Step-3: Then, we'll analyze the Graph for Cycles:
The resource-allocation graph must contain a cycle for deadlock to potentially occur.

Step-4: After that, we'll determine if a Deadlock Exists:
If a cycle exists, a deadlock may occur.
We'll check the conditions of deadlock and see if they are satisfied within the cycle.

Step-5: We'll examine the Cycle for Deadlock Conditions:
we'll have to ensure that each process in the cycle is holding at least one resource while waiting for another resource held by another process in the cycle and then confirm that no process can progress without obtaining the resource held by another process in the cycle.

Step-6: deadlock occurred or not?
If the conditions for deadlock are met within the cycle, then the scenario leads to a deadlock.

In our case or example, it seems that a deadlock did occur since a cycle is present in the RAG.
So this is how we figure out if a situation leads to a deadlock or not using a RAG.

Discuss how a resource allocation graph can be utilized to identify the necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, and circular wait). **Provide** examples to support your explanation.

Ans 2) A resource allocation graph (RAG) can be used to identify deadlocks based on the four necessary conditions: mutual exclusion, hold and wait, no preemption, and circular wait.

Here's how a resource allocation graph helps identify these conditions:

Mutual Exclusion: Resource categories with only one instance (represented by a single dot within the resource box) inherently imply mutual exclusion. Since only one process can hold the resource at a time, there will be an assignment edge from that resource to the process holding it, and no other process can have an edge to that resource dot.

Hold and Wait: This condition can be identified by looking for processes with both assignment edges (holding resources) and request edges (waiting for resources). In RAG, an assignment edge points from a resource (dot) to a process, and a request edge points from a process to a resource box (not a specific instance dot). For example, if process P1 has an assignment edge to resource R1 and a request edge to resource R2, it means P1 is holding R1 and waiting for R2.

No Preemption: No preemption can be identified in the context of the resource allocation graph (RAG) by looking for the absence of any edges being removed. The RAG represents a snapshot of resource allocation, and it doesn't show the system taking back resources from a process.

Circular Wait: This condition is the most easily identifiable using a resource allocation graph (RAG). A cycle formed by request and assignment edges indicates a circular wait. In this cycle, each process in the loop is waiting for a resource held by another process in the same loop. For example, if process P1 has an assignment edge to R1 and a request edge to R2, process P2 has an assignment edge to R2 and a request edge to R3, and process P3 has an assignment edge to R3 and a request edge to R1, then a deadlock exists due to a circular wait.

Examples:

Process P1 holds resource R1 and waits for R2, process P2 holds R2 and waits for R1, creating a cycle. Therefore, a deadlock exists due to circular wait.
Let us imagine a scenario with two printers (resource category) and two processes (P1 and P2). If the resource allocation graph shows P1 holding printer A and requesting printer B, and P2 holding printer B and requesting printer A, this would represent a circular wait and deadlock.

Explain how the resource allocation graph can be employed to detect and resolve deadlocks in a multi-threaded application.

Ans 3) To detect Deadlocks in Multi-threaded Applications, we need Resource Allocation Graphs (RAGs)

A Resource Allocation Graph (RAG) is a useful tool to visualize how resources are allocated to threads in a multi-threaded application and identify potential deadlocks.

RAG consists of vertices and edges.
There are 2 types of edges - assignment edge and request edge.
An Assignment Edge is an arrow pointing from a resource vertex to a thread vertex indicates the resource is currently allocated to that thread (e.g., thread T1 using resource R1).
A Request Edge is an arrow pointing from a thread vertex to a resource vertex indicates the thread is requesting that resource (e.g., thread T2 waiting for resource R2).

A Single Instance Resource is a box with one dot signifies only one instance of that resource exists (e.g., a printer).
A Multi-Instance Resource is a box with multiple dots signifies there are multiple instances available (e.g., memory units).
By analyzing cycles in a RAG, we can identify potential deadlocks in both Single Instance Resources and Multiple Instance Resources.

In Single Instance Resources, a cycle with each resource having only one instance (single-instance resource) always represents a deadlock. Let us imagine threads T1, T2, and T3 where T1 holds R1, T2 holds R2, and T1 waits for R2 while T2 waits for R1. This creates a circular dependency, and none of the threads can proceed.
In Multi-Instance Resources, a cycle alone isn't sufficient to guarantee a deadlock with multi-instance resources. We need to consider the available resources and thread requests using techniques like Banker's Algorithm to determine if a deadlock exists. In these cases, RAGs can only indicate a potential deadlock, requiring further analysis.
RAG provides a clear visual representation of resource allocation and thread requests in a multi-threaded application; helps in identifying potential deadlocks, especially for smaller systems with single-instance resources.

Unfortunately, RAGs themselves cannot directly resolve deadlocks. However, by identifying potential deadlocks through RAG analysis, developers can implement strategies to prevent or recover from them. Below are some common methods for handling deadlocks:

Deadlock Prevention: Ensure that at least one of the four necessary conditions for deadlocks cannot hold (Mutual Exclusion, Hold and Wait, No Preemption, Circular Wait). This can be complex and might impact resource utilization.

Deadlock Avoidance: The operating system tracks resource allocation and requests, granting or denying requests based on deadlock avoidance algorithms (like Banker's Algorithm) to prevent cycles from forming.

Deadlock Detection and Recovery: Allow deadlocks to occur and then detect them using techniques that analyze the system state (including RAG analysis for single-instance resources). Once detected, recovery involves techniques like preemption (taking resources from a thread) or thread rollback (restarting a thread from a previous state).

Prevention is not possible and deadlock detection & recovery is too complicated. Deadlock Avoidance, especially Banker's Algorithm is mostly preferred.

RAGs are a valuable tool for visualizing resource allocation and potential deadlocks in multi-threaded applications, particularly for single-instance resources. However, they might require additional techniques like Banker's Algorithm for deadlock detection in complex scenarios with multi-instance resources. Deadlock resolution involves strategies like prevention, avoidance, or detection and recovery.

Suppose there are multiple instances of resources available in a system, and processes request and release resources dynamically. **How** would you utilize the concept of a resource allocation graph to determine if a deadlock could occur in such a scenario?

Ans 4) In a system with multiple instances of resources and dynamic resource requests or releases by processes, RAGs alone cannot definitively determine if a deadlock will occur. This is because Cycles Don't Guarantee Deadlock. Unlike single-instance resources where a cycle in the RAG always signifies a deadlock, cycles with multi-instance resources don't necessarily guarantee one. The availability of multiple resource instances creates the possibility of fulfilling requests and breaking the cycle.

However, RAGs can still be helpful in identifying potential deadlocks. RAGs provide a clear picture of how resources are currently allocated and which processes are waiting for specific resources. This visualization can help identify situations where a cycle is forming, even with multiple resource instances.

To definitively determine if a deadlock can occur, RAGs need to be combined with other techniques like Banker's Algorithm and Monitoring Resource Availability. Banker's Algorithm analyzes the allocation and request matrices along with the number of available resources to predict if a safe sequence of resource allocation exists, preventing deadlocks. Another method is continuously tracking the number of available resource instances, this can help identify situations where deadlocks become more likely due to resource depletion. Techniques like Banker's Algorithm and resource availability monitoring are crucial for a more complete picture.

By combining RAG visualization with these additional techniques, we can gain a better understanding of resource usage patterns and take proactive measures to prevent deadlocks.

Discuss the limitations of using a resource allocation graph to manage deadlock in a system.

Ans 5) While RAGs are a valuable tool for visualizing resource allocation and potential deadlocks, they have limitations, especially when managing deadlocks in a system with multiple instances of resources and dynamic resource requests or releases.

Limitations include:-

1. RAGS are not able to detect deadlocks with multi-instance resources. RAGs excel at deadlock detection in scenarios with single-instance resources. A cycle in the RAG with only one instance per resource type always indicates a deadlock. However, with multiple instances, a cycle doesn't guarantee a deadlock. The availability of additional resources creates the possibility of fulfilling requests and breaking the cycle. RAGs alone cannot determine if enough free resources exist to prevent deadlock.

2. As the number of processes and resources increases, RAGs can become visually complex and difficult to interpret. The complexity for larger systems increase. Analyzing intricate cycles and relationships between processes and resources can be challenging.

3. RAGs primarily identify potential deadlocks. They don't provide specific information on how to resolve deadlocks if they occur. We need additional techniques to determine which processes or resources are involved in the deadlock and how to break the cycle.

4. In a dynamic environment where processes frequently request and release resources, constantly updating the RAG can introduce overhead. Maintaining an accurate representation of the system's state can be resource-intensive.

5. RAGs are reactive – they show the current state of resource allocation. They cannot predict future resource requests and potential deadlocks that might arise due to the dynamic behavior of processes.

Submit

Clear form