1. **List three examples of deadlocks that are not related to a computer system environment.**

* A person wishes to exit through the door, and another wishes to enter through the door. They won't be able to get through together.
* On a crossroads of two roads, there are four cars, each of which wants to travel in one way.
* A knife and a fork are included. One person is holding a fork, while the other is holding a knife. To eat a steak, you'll need both.
* Two cars crossing a single lane bridge from opposite directions.
* A person going down a ladder while another person is climbing up the ladder.
* Two trains traveling toward each other on the same track.

1. **Is it possible to have a deadlock involving only one single process? Explain your answer.**

A deadlock situation can only arise if the following four conditions hold simultaneously in a system:

* Mutual Exclusion
* Hold and Wait
* No Preemption
* Circular-wait

It is impossible to have circular-wait when there is only one single-threaded process. There is no second process to form a circle with the first one. One process cannot hold a resource, yet be waiting for another resource that it is holding.

So it is not possible to have a deadlock involving only one process.

(OR)

It is not **possible to have a deadlock involving only one single process**. The **deadlock** involves a circular “hold-and-wait” condition between two or more **processes**, so “**one**” **process** cannot hold a resource, yet be waiting for another resource that it is holding.

1. **Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock free.**

Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

|  |  |  |
| --- | --- | --- |
| **P1** | **P2** | **P3** |
| R | R | R |
| R |  |  |

Yes, system is deadlock free. As soon as P1 completes, the resources can be used by either P2 or P3.

1. **Consider a system consisting of m resources of the same type being shared by n processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold: a. The maximum need of each process is between one resource and m resources. b. The sum of all maximum needs is less than m + n.**

In a system containing m resources of the same type being shared by n processes, a deadlock can occur when multiple processes are waiting for resources that are held by other processes.

To ensure that the system is deadlock free, two conditions must hold:

(i) The maximum need of each process is between 1 and m resources.

* This means that no process can request more than m resources at any given time.
* This ensures that there will always be enough resources available to satisfy the needs of all processes.

(ii) The sum of all maximum needs is less than m+n.

* This means that the total number of resources needed by all processes is less than the total number of resources available (m) plus the total number of processes (n).
* This ensures that there will always be some resources available for each process to request, even if all processes request their maximum number of resources at the same time.
* Together, these two conditions guarantee that there will never be a situation in which all processes are waiting for resources that are held by other processes, thus avoiding deadlock.

1. **What is a safe state? Give the use of safe state in deadlock avoidance.**

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

System is in safe state if there exists a sequence of ALL the processes in the systems such that for each Pi , the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj , with j < I

That is: λ

* If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished
* When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate
* When Pi terminates, Pi +1 can obtain its needed resources, and so on