Illustrate how deadlock can occur.

Multiple processes can have access to the same resource. If the processes are not properly synced together they can end up in a deadlock, a situation where they are waiting for each other to release this shared resource.

Define the four necessary conditions tha characterize deadlock.

The four necessary conditions are:

1. **Mutual exclusion:** At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time. (non-shareable resource)
2. **Hold and wait:** A process must be holding at least one resource and waiting for resources currently held by other processes. (resource holding)
3. **No preemption:** A resource can be released only voluntarily by the process holding it, after that process has completed its task. (no preemption)
4. **Circular wait:** A set $\{P_0, P_1, ..., P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2, ..., P_{n-1}$ is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

We can still apply a general assumption: *if we give a process all the requested resources, the process will finally give all resources back*

Detect a deadlock situation in a resource allocation graph.

See picture example in the slides: 7-15

Detect a deadlock situation using the matrix-based deadlock detection algorithm.

Given the following vectors and matrices:

$$E = \begin{pmatrix} 3 & 2 & 3 & 1 \end{pmatrix} A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix} C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} R = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix}$$

We can use the safety algorithm to detect whether or not a deadlock exists. The safety algorithm is as follows:

- Find and grant a request of instances that A can provide
- Mark that process as finished and add its resources to A
- Repeat until all processes are finished or there are no more requests that can be granted
- If all processes are finished, then there is no deadlock. Otherwise there is a deadlock.

For this example we could grant P_3 its resources, mark it as finished and have $A = \begin{pmatrix} 2 & 2 & 2 & 0 \end{pmatrix}$ resources for the next request. Then we could grant either of P_1 or P_2 request as they are both asking for an equal or less amount of resources than exist in A. Therefore there is no deadlock.

Evaluate the four different approaches for handling deadlocks.

There are 4 different approaches for handling deadlocks:

- **Ignore the problem:** Easy to implement, but not a great solution.
- **Detection and recovery:** Allow system to enter deadlock state, detect it and then recover from it.
- **Dynamic avoidance:** Prevent deadlock by careful resource allocation. Reject resource requests that may lead to deadlock.
- **Prevention:** Ensure that one of the four necessary conditions for deadlock cannot occur.

Apply the Safety algorithm to obtain safe schedules (if they exist)

See the example of the Safety algorithm in the previous question.

Apply the Banker's algorithm for deadlock avoidance.

The Banker's algorithm makes sure we don't grant requests of resources that will lead to a deadlock. We do this by granting a request and checking if that state is safe. If so, we grant the request, otherwise we deny it.

Evaluate approaches for recovering from deadlock.

There are three approaches for recovering from deadlock:

- **Human intervention:** The system will inform the operator that a deadlock has occurred. The operator will then decide which process(es) to terminate.
- **Process termination:** Abort all deadlocked processes. Abort one process at a time until the deadlock cycle is eliminated.
- **Resource preemption:** Select a victim. Rollback and restart the victim process so that a requesting process can have its resources.