Aidan Kiser

28 March 2024

COMP-3500

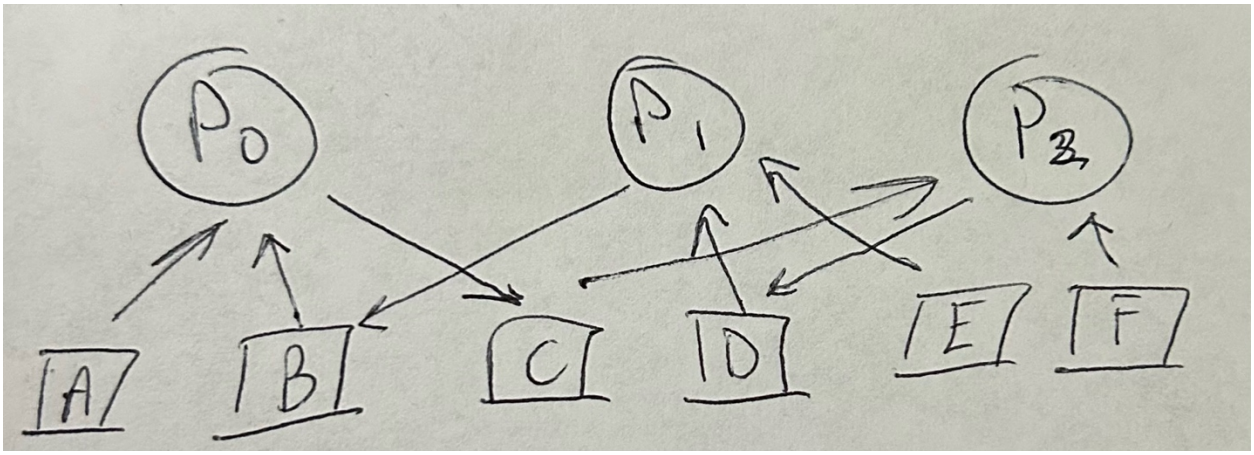
Project 4

1. (a). A deadlock is possible if there is a cycle, meaning that each process is holding at least one resource and waiting for another that is held by another process in the cycle. A deadlock would occur in these events:

1. P0 request A: Resource A is free, assigned to P0
2. P1 request D: Resource D is free, assigned to P1
3. P2 request C: Resource C is free, assigned to P2
4. P0 request B: Resource B is free, assigned to P0
5. P1 request E: Resource E is free, assigned to P1
6. P2 request F: Resource F is free, assigned to P2
7. P0 request C: Process P0 waits for resource C to be free
8. P1 request B: Process P1 waits for resource B to be free
9. P2 request D: Process P2 waits for resource D to be free

This forms a cycle of processes each waiting for the next to release a resource, which is a classic deadlock condition.

Below is the resource allocation graph:



(b). Order of get requests that do not cause a deadlock:

Process P0:

1. get(B)
2. get(C)
3. get(A)

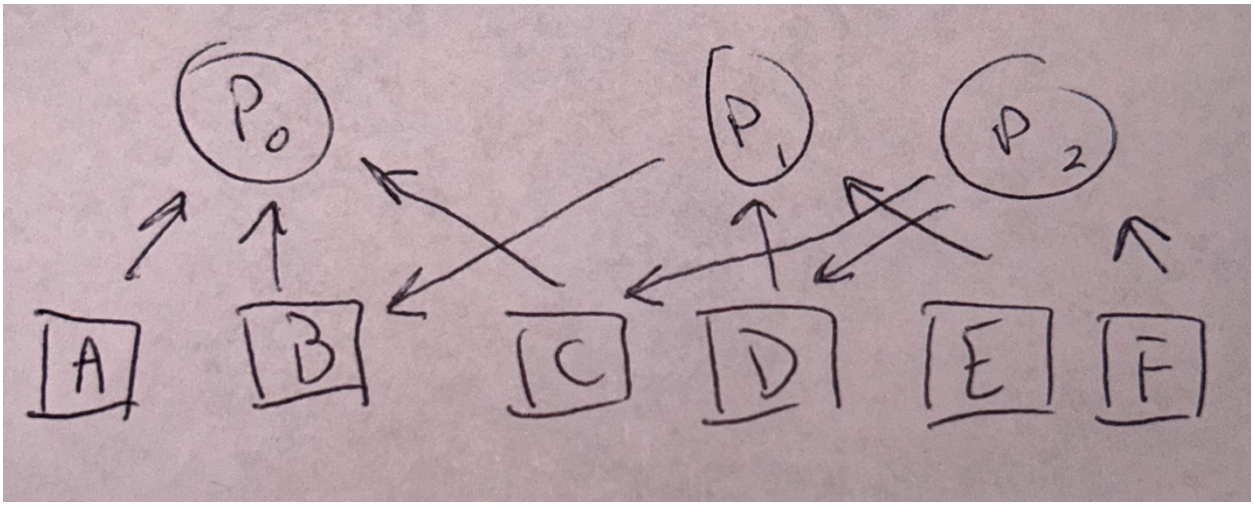
Process P1:

1. get(D)
2. get(E)
3. get(B)

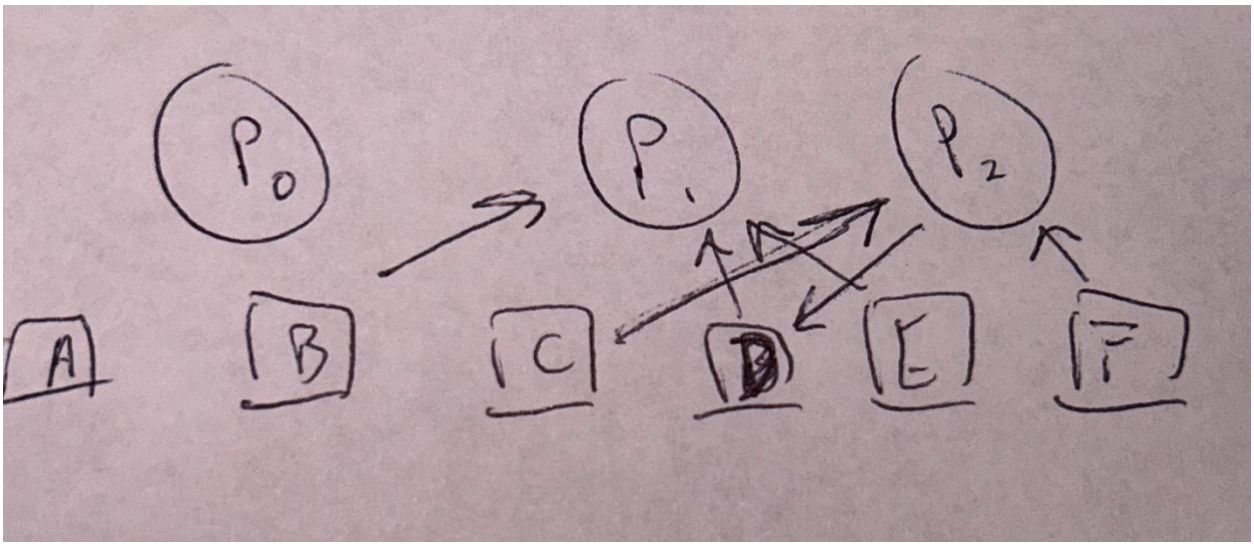
Process P2:

1. get(F)
2. get(D)
3. get(C)

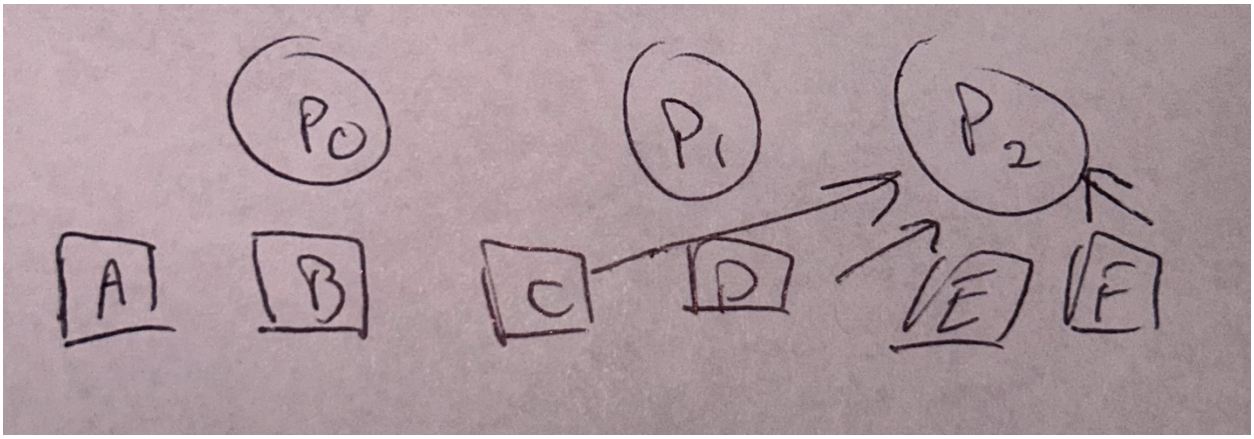
Below is the resource allocation graph:



Process P0 gets all its resource requirements fulfilled and hence will enter its critical section. After finishing execution in its critical section, process P0 will release all the resources held by it.



Resource B is free and process P1 is requesting it, B will be assigned to process P1. Similarly resource C will be assigned to process P1. P1 now holds all required resources and will enter its critical section.



The resource allocation graph after the changes would show a chain but no cycles, which means that no deadlock can occur.

2.

Yes, these two processes can enter a deadlock. This can happen if they acquire the semaphores in a way that each process is holding one semaphore and is waiting for the other, which is being held by the other process. The semaphores S and R are used to ensure mutual exclusion and to coordinate the execution between the two processes. Since both semaphores are initialized to 1, they can be taken by a process, and any other process trying to take the same semaphore will block until the semaphore is released.

Here's an example of an execution sequence where foo and bar would become deadlocked:

1. foo executes semWait(S), taking semaphore S (which goes to 0).
2. bar executes semWait(R), taking semaphore R (which goes to 0).
3. foo tries to execute semWait(R) but blocks because bar holds R.
4. bar tries to execute semWait(S) but blocks because foo holds S.

3.

Deadlock prevention, avoidance, and detection are strategies to manage deadlocks in a multi-process system. Deadlock Prevention is about designing the system in a way that the conditions for a deadlock are eliminated, ensuring deadlocks cannot occur. Deadlock Avoidance requires system knowledge about future allocations and ensures that a system only enters states where no deadlocks can arise, usually through careful resource allocation. Deadlock Detection lets deadlocks happen but includes mechanisms to identify and resolve them, typically by aborting processes or rolling back transactions. Prevention is the most restrictive and can lead to underutilization of resources, while avoidance is less restrictive but needs advance knowledge of resource requests, and detection handles deadlocks reactively but can disrupt process execution.

4.

The system can be kept deadlock-free by ensuring that a process can only acquire a second resource if there are at least three resources available. This rule guarantees that there will always be at least one resource available, preventing a situation where a circular wait could occur, as each process will be able to obtain at least one resource. Thus, by this strategic allocation, the system avoids deadlocks and allows all processes to eventually complete their tasks.