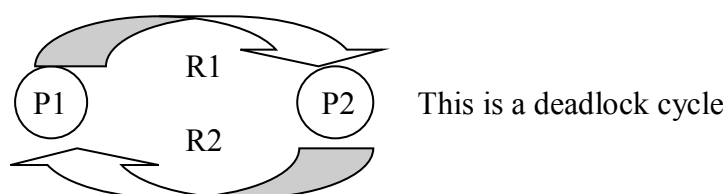


1) Deadlock Management

a. What is a deadlock?

Deadlock is when in a set of processes one of this process is waiting a response from one of the process which is waiting for the same reason. For example, we have two processes. The first one needs a response from the second one to awake and give a response to the second process to awake him... it is a blocked situation. Both of the processes are going to wait the other one for a response which will never come.

This unexpected situation can be schematized as below:



To have a deadlock situation we must have the 4 conditions below:

- Mutual exclusion condition, when a resource cannot be shared at a time by other processes.
- Hold and wait condition, when a process is holding resources and waits some resources from other process to acquire.
- No pre-emptive condition, when a process must wait the end of its allocated time.
- Circular wait condition, when each process is waiting for resources held by other process which are in the same situation.

b. How can we handle deadlock?

There are four ways to avoid deadlock situation:

b.1) The Ostrich algorithm

to stick your head in the sand and pretend that there is no problem

This strategy consists on ignoring problems due to the hypothesis that there will be rare. Moreover, it will cost more to solve the problem than ignore it.

b.2) Deadlock detection and recovery

At first, we need to find if there is a deadlock situation. For this, we will check if there is a cycle in the resource graph. For each node we will do a depth first search. If we come back on a node already traversed there is a cycle.

Then we will recovery by using one of the following methods:

- Through pre-emptive: it may can free a resource

- Rollback: We make a checkpoint periodically to see which resource is needed in case of deadlock. Then we take the resource from the process which having it. Then we restart this process.
- Killing processes: if it is possible, we will kill a process which can be rerun and will not damage anything.

b.3) Deadlock avoidance

It avoids situations which can lead to deadlock. For this, it is necessary to have safe states. There is safe state if the state is not in deadlock state or if there is some sequence by which all requests can be satisfied.

Then, we just need to make transitions between safe states.

To realise it we can use Banker's Algorithm. When a request is made, it checks if after the request this request is satisfied. There is one or several sequence of moves that can satisfy all the requests. Thus the new state is safe. If so, satisfy the request, else make the request wait.

b.4) Deadlock prevention

The difference from avoidance is that in such a system, it was designed to have no deadlocks. It is always guaranty that at least 1 of the 4deadlocks conditions is never satisfied.

To do it we can:

- Attack Mutex condition: never grant exclusive access. but this may not be possible for several resources.
- Attacking pre-emption: not a good idea, too dangerous for the processes
- Attacking hold and wait condition: make a process hold at the most 1 resource at a time. Then make all the requests at the beginning. If it works it is running if it is not, nothing happen.
- Attacking circular wait: order all the resources to be sure that there are no cycles in the resource graph.

2) Conclusion

Thus, the deadlock situations can happen in almost of operating systems. That is why we developed different way to solve this problem. Some are using more resources than others or may be able to lead to others problems.

3) Sources

<http://www.scribd.com/doc/27201125/deadlock-management>

<http://www.wiziq.com/tutorial/481-Operating-Systems-OS>

http://www1bpt.bridgeport.edu/sed/projects/cs503/Spring_2001/kode/os/deadlock.htm#avoidance