11 Deadlock

Deadlock is one area where there is a strong theory, but it's almost completely ignored in practice. Reason: solutions are expensive and/or require predicting the future.

Deadlock example: semaphores. Two processes: one does P(x) followed by P(y), the other does the reverse.

Deadlock: a situation where each of a collection of processes is waiting for something from other processes in the collection. Since all are waiting, none can provide any of the things being waited for.

These are relatively simple-minded cases. Things may be much more complicated:

- In general, don't know in advance how many resources a process will need. If only we could predict the future....
- Deadlock can occur over separate resources, as in semaphore example, or over pieces of a single resource, as in memory, or even over totally separate classes of resources (tape drives and memory). Deadlock can occur over anything involving waiting, for example messages in a pipe system. Hard for OS to control.

In general, there are four conditions for deadlock:

- Mutually exclusive access: resources cannot be shared.
- No preemption. Once given, a resource cannot be taken away.
- Hold and wait: processes can hold resources while they are waiting for more resources.
- There is a circularity in the graph of who has what and who wants what. This graph shows processes as circles, resources as squares, arrows from process to resource waited for, from resource to owning process.

Solutions to the deadlock problem fall into two general categories:

- Detection : determine when the system is deadlocked and then take drastic action. Requires termination of one or more processes in order to release their resources. Usually this isn't practical.
- Prevention : organize the system so that it is impossible for deadlock ever to occur. May lead to less efficient resource utilization in order to guarantee no deadlocks.

Deadlock prevention: must find a way to eliminate one of the four necessary conditions for deadlock:

- Don't allow exclusive access. This is probably not reasonable for many applications.
- Create enough resources so that there's always plenty for all.
- Don't allow waiting. (phone company solution) This punts the problem back to the user.
- Allow preemption. E.g. pre-empt student's thesis disk space?
- Make process ask for everything at once. Either get them all or wait for them all. Tricky to implement: must be able to wait on many things without locking anything. Painful for process: may be difficult to predict, so must make very wasteful use of resources. This requires the process to predict the future.
- Banker's algorithm.
  1. state maximum resource needs in advance
  2. allocate reources when needed; wait when granting request would lead to deadlock
- Make *ordered* or *hierarchical* requests. E.g. ask for all S's, then all T's, etc. All processes must follow the same ordering scheme. Of course, for this you have to know in advance what is needed.

In general, prevention of deadlock is expensive and/or inefficient. Detection is also expensive and recovery is seldom possible (what if process has things in a weird state?).