Lecture 11

April 20, 2023

HW #6 will be posted tonight. Due the day of exam by 11:59 pm.

Final exam: May 4th

3 hours + cheat sheets are allowed including physical textbooks.

What type of questions: similar to midterm longer + more spicy + lecture based + reading assignments

Comprehensive + problem solving + Multiple choice.

Deadlocks

Definition: a deadlock is a situation in which two or more processes have ownership of resources and cannot complete or exit unless each of these processes acquires. Some or all of the one of the other processes resources.

Problem that must be addressed by OS.

OS is problem solver.

Concurrency makes it more complex.

Parallelism is good but the bade side -> harder for OS.

Processes -> OS: oK

Threads -> OS : ok

Threads communication -> Complex

OS Solution: establish Thread

communication protocols such as message passing, pipeline, sockets.....

Process execution + multiple data

1. single Instruction + single data OS: ok --> uniprocessor --> Every day use (example Ms Word)
2. single Instruction + Multiple data OS: ok --> pipeline --> Scientific applications
3. Multiple Instruction + Multiple data OS: ok --> multi-processor --> AI
4. Multiple Instruction + Single data **OS: NO**

"4. Multiple Instruction + Single data", In practice this is hardly needed.

1, 2, 3 are most common ones here.

1 use everyday (example Ms Word)

2 Scientific applications

3 AI

More greedy:

We want threads cooperations => Problem

problem: critical sections

OS said OK (meaning that it will go ahead and do it okay)

solution: Monitors, Semaphores; (Programming Solutions) --> Most powerful

Peterson Algorithms; (User solutions)

Hardware Instructions. (Machine solutions)

More greedy:

Many Processes running in real time, user must not wait

OS: CPU scheduling. FIFS, RR, SJF ...

Synchronizations: Monitors, ...

Parallelism + Limited resources: These two is a recipe of a disaster -> Deadlocks

OS has to come up with the solution. You can do it in the present tense, or future, or past.

- present: Deadlock prevention
- future: Deadlock Avoidance
- past: Deadlock Detection

Deadlock prevention, deadlock avoidance could be very conservative or more pessimistic, meaning they will prevent a process to execute even though it may not result in a deadlock.

Recall it is all about managing resources. The second most vital resource after CPU is **Memory**.

-> Parallelism + limited memory

How to manage memory

Partition, paging, segmentation.

+ the sky is the limit we pretend to have unlimited RAM => virtual Memory

We have reviewed: is the objective of this courses.

OS challenges + Solutions.

Deadlocks

Four conditions must exist at the same time in order for a deadlock to occur.

1. Mutual exclusion must be enforced
2. Hold and wait
3. No preemption
4. circular wait

If you break any one of these and deadlock is not going to happen.

1 must have, 2 if you break it's inefficient.

Basically try to break 3, 4.

Deadlock Avoidance

The OS makes a decision dynamically whether the current resources allocation request will if granted potentially lead to a deadlock.

What does it mean?

Process initiation denial. Do not start a process if its demands might lead to a deadlock.

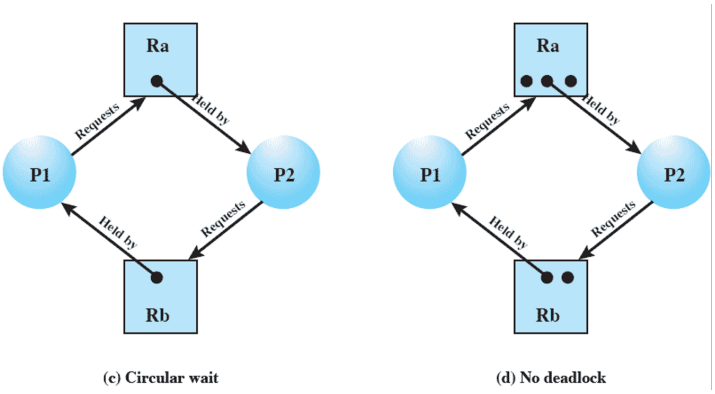
Alternatively, a process p0.

p0 is asking the OS to grant it an instance of resource R0.

OS might deny granting this request because it may lead to a deadlock.

Conditions for possible Deadlock

- Mutual exclusion
  - Only one process may use a resource at a time
- Hold-and-wait
  - A process may hold allocated resources while awaiting assignment of others
- No pre-emption
  - No resource can be forcibly removed form a process holding it.
- Circular wait
  - A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.



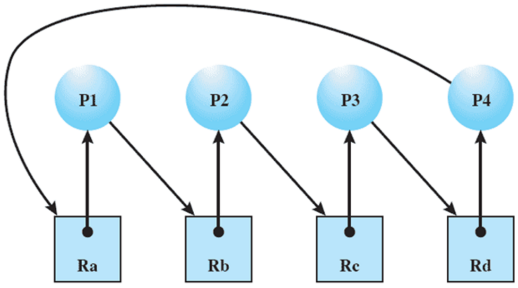


Figure 6.6 Resource Allocation Graph for Figure 6.1b

No body wants to let go, then we will be in a deadlock situation.

Dealing with Deadlock

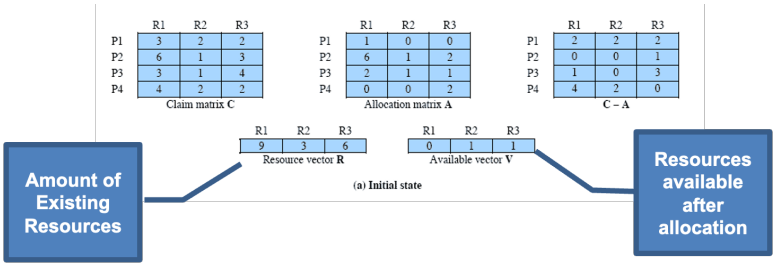
- Three general approaches exist for dealing with deadlock.
  - Prevent deadlock
  - Avoid deadlock
  - Detect deadlock

Safe State

Final Exam

Determination of Safe State

- **A system consisting of four processes and three resources.**
- **Allocations are made to processors**
- **Is this a safe state?**



C: is what each process is asking for

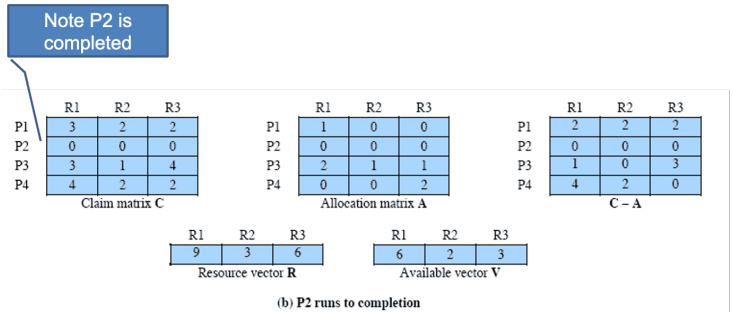
A: is what OS just give you a few for whatever reason.

C-A: is what remaining for these processes to complete to finish to exit and each one of them is requesting certain type of resources. It doesn't mean that it is going to grant it.

R: the total number of available resources.

V: the left of the resources.

After p2 runs to completion



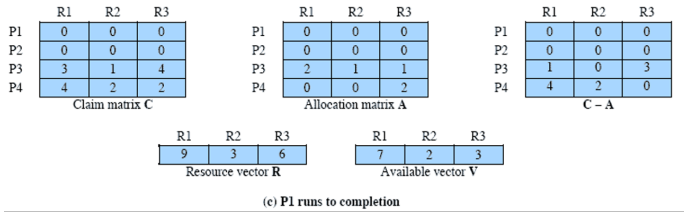
(b) P2 runs to completion

Available vector V 6, 2, 3 can satisfy any one of these needs, I can satisfy p1 is requests, I can satisfy p3 requests and I can satisfy p4 requests because I have enough. It doesn't matter which one I pick up next. I am guaranteed that everybody is going to run into completion.

Eventually once everything is done, the available vector V will be 9,3,6.

This is what we call safe State.

After P1 completes

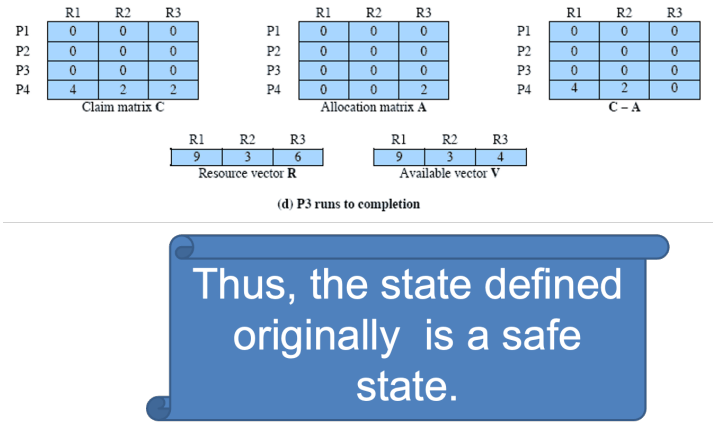


(c) P1 runs to completion

P3 Completes, resulting in the state in the following figure

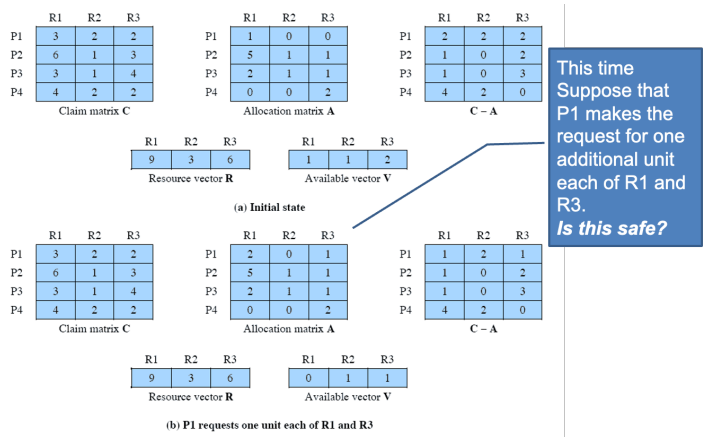
Finally, we can complete P4. At this point, all of the processes have been run to completion.

Thus, the state is a safe state.



Final Exam, you might be given something like this and you'll be asked is it the safe state or not.

Unsafe State



Exam: I'll give you three different values of the vector v and ask you which one of these will result in a safe state or which one of these will result in an unsafe state.

Question: Will OS granted?

Avoidance: NO

Policy for granting/denying resources when applying Deadlock avoidance. Algorithm.

If a process say p1 is asking for a resource R1, OS **pretends** that such a request is granted and determines if such granting will result in deadlock. To do the OS examines the other processes say p2, p3, p4 ... and determine the units of R1 that each of these processes still needs. If after granting p1 resource R1, OS finds out that the available units of R1 are not enough to satisfy any of the other processes requests. Then p1 is denied R1.

This is called avoidance.

Deadlock Detection

- Deadlock prevention strategies are very conservative;
  - limit access to resources and impose restrictions on processes.
- Deadlock detection strategies do the opposite
  - Resource requests are granted whenever possible.
  - Regularly check for deadlock

HOW?

A Common Detection Algorithm

- Use a Allocation matrix and Available vector as previous
- Also use a request matrix Q
  - Where Q_{ij} indicates that an amount of resource j is requested by process i
- First 'un-mark' all processes that are not deadlocked
  - Initially that is all processes

Deadlock Detection:

policy: grant resources whenever requested different from Deadlock avoidance, where try to foresee the future state before granting resources.

Deadlock Detection: grant resources + frequently check for deadlocks.

How?

Algorithm:

1. The allocation matrix A and the available vector V are used as before.
2. Define a new matrix Q to represent the amount of resources of type j requested by process i

   Initially all processes are unmark

Do the following:

Detection Algorithm:

1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vector W to equal available vector.
3. Find an index i such that process i is currently unmarked and the ith row of Q is less than or equal to W.
   - i.e. $Q_{ik} \leq W_k$ for $1 \leq k \leq m$.
   - If no such row is found, terminate

4. If such a row is found
    - mark process i and add the corresponding row of the allocation matrix to W.
    - i.e. set $W_k = W_k + A_{ik}$ for $1 \leq k \leq m$.

Return to step 3.

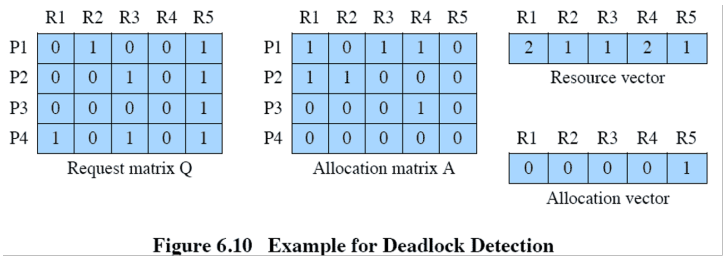
- A deadlock exists if and only if there are unmarked processes at the end
- Each unmarked process is deadlocked.

The strategy in this algorithm is to find a process whose resource requests can be satisfied with the available resources, and then assume that those resources are granted and that the process runs to completion and releases all of its resources. The algorithm then looks for another process to satisfy.

Note that this algorithm does not guarantee to prevent deadlock;

- That will depend on the order in which future requests are granted.
- All that it does is determine if deadlock currently exists.

Example



	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request matrix Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation matrix A

R1	R2	R3	R4	R5
2	1	1	2	1

Resource vector

R1	R2	R3	R4	R5
0	0	0	0	1

Allocation vector

Figure 6.10 Example for Deadlock Detection

1. Mark P4, because P4 has no allocated resources.
2. Set $W = (0, 0, 0, 0, 1)$
3. The request of process P3 is less than or equal to W, so mark P3 and set W = W + (0, 0, 0, 1, 0) = (0, 0, 0, 1,1).
4. No other unmarked process has a row in Q that is less than or equal to W.

   Therefore, terminate the algorithm.

The algorithm concludes with P1 and P2 unmarked, indicating that these processes are deadlocked.

Recovery Strategies Once Deadlock Detected

- Abort all deadlocked processes
- Back up each deadlocked process to some previously defined checkpoint, and restart all process
    - Risk or deadlock recurring
- Successively abort deadlocked processes until deadlock no longer exists.
- Successively preempt resources until deadlock no longer exists.