Deadlock II

Tom Kelliher, CS 311

Apr. 2, 2012

1 Administrivia

Announcements

Exam in one week.

Assignment

From Last Time

Deadlock I.

Outline

1. Deadlock detection and recovery.
2. Deadlock avoidance.
3. Summary.

Coming Up

Linux kernel modules.

2 Deadlock Detection and Recovery

1. Permit deadlocks to occur, subsequently recover from them.
2. Periodically run deadlock detection routines.
3. Choose victim processes.

2.1 Deadlock Detection

Algorithmic deadlock detection:

1. n processes.
2. m resource *types*.
3. Available — vector of length m.
4. Allocation — matrix of size n by m. Rows: processes; columns: resources.
5. Request — matrix of size n by m.
6. Finish — vector of length n.
7. Work — vector of length m.

The algorithm:

```
work = available;

for (i = 0; i < n; ++i)
   finish[i] = false;

while there is an i such that finish[i] == false
and request[i] <= work
{
   finish[i] = true;
   work = work + allocation[i];
}
```

1. Running time?

2. How often should this be run? What are the tradeoffs?

3. Any processes for which `finish` is false are deadlocked.

Using a resource allocation graph to detect deadlock (5 processes, 3 resource types):

$$\text{Available} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$\text{Allocation} = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

$$\text{Request} = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Retry with this request matrix:

$$\text{Request} = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

2.2 Deadlock Recovery

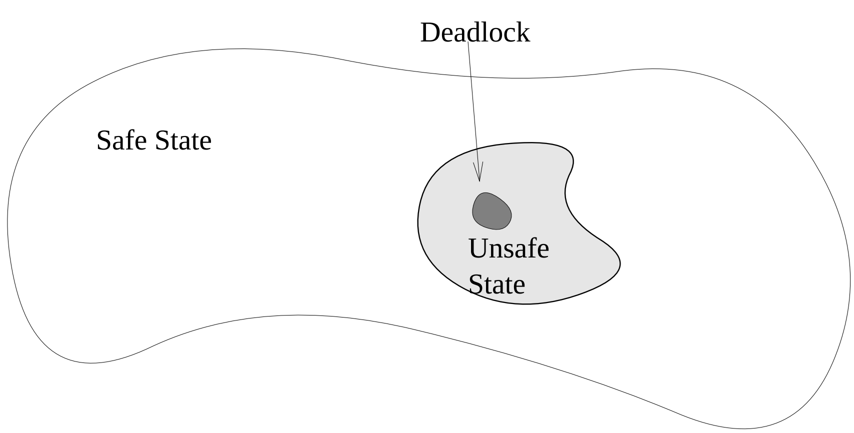
Some recovery mechanisms:

1. Terminate *all* deadlocked processes. What about threads (say only some of the threads in a task are deadlocked)?

2. Terminate processes one at a time.
    (a) How do you choose the victim?
    (b) When do you stop terminating victims?
3. "Rolling back" a process and preempting resources.
    (a) Process termination is drastic and messy.
    (b) Checkpoints.
    (c) How much state has to be checkpointed?
    (d) Starvation.

3 Deadlock Avoidance

Always maintain the system in a safe state:



Grant a resource request only if resulting state is safe.

Safe state:

1. There exists a sequence in which all resource requests can be satisfied.
2. The operating system is in control of the resource situation.

Unsafe state:

1. The operating system is no longer in control of the resource situation.
2. Processes are in control and can cause a deadlock.

Banker's algorithm used to maintain safe state.

Additional matrix required: Claim — maximum number of each resource type needed by each process.

```
For each resource request which can be satisfied
{
   simulate:
      satisfy the request;
      update state matrices;
      turn all remaining claims into requests;
      test for deadlock;

   if simulation resulted in deadlock
      defer the request;
   else
      grant the request;
}
```

3.1 Example

Assume a maximum-claim serially reusable system with four processes and three resource types. The claim matrix is given by

$$C = \begin{bmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 5 & 7 & 13 \\ 1 & 1 & 6 \end{bmatrix},$$

where $C_{i,j}$ denotes the maximum claim of process i for resource j. The total units of each resource type are given by the vector (5 8 16). The allocation of resources is given by the matrix

$$A = \begin{bmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{bmatrix},$$

where $A_{i,j}$ denotes the number of units of resource j that are allocated to process i.

1. Determine if the current state of the system is safe.
2. Determine if granting a request by process 1 for 1 unit of resource 1 can be safely granted.
3. Determine if granting a request by process 3 for 6 units of resource 3 can be safely granted.

4 Summary

1. Some resources are easy to keep out of deadlock: CPU cycles, memory.
2. No one "silver bullet" — must combine mechanisms.
3. Must tradeoff between cost of protection and cost of deadlock.