Computer Science 4500
Operating Systems

Module 7
Deadlocks
In This Module…

- Deadlock: Definition and Examples
- Deadlock: Models
- Deadlock: Prevention Algorithms
Deadlock Definition

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.
Deadlock Characteristics

- Deadlocks occur when multiple processes are attempting to make exclusive use of multiple resources.
- The resources that are the source of deadlock conflicts may be hardware resources, software resources, or both.
- Example: Process A gets exclusive access to resource R, B gets access to S. Then A requests S and B requests R... DEADLOCK!
Consumable and Non-Consumable Resources

- We categorize resources involved in deadlocks in several ways.
- First, resources may be consumable or non-consumable. For example:
  - Primary memory is non-consumable, since after its use it is available for other uses.
  - Messages from another process are consumable resources. When they arrive and are processed they disappear.
Resource Preemption

Resources may also be preemptable or non-preemptable. By preemption, we mean as resource can be taken away from a process for a while (to satisfy the needs of another process) and later returned. For example...

- Primary memory is a preemptable resource in virtual memory systems, since it can have its contents saved on secondary storage and later restored.
- A tape drive isn’t normally considered to be a preemptable resource, as it would be very costly to share it.
Typical Use of Non-Preemptable Resources

1. A process first requests exclusive use of the resource. If it is available, the process continues. Otherwise, the process blocks until it is available. If multiple units of the resource are available, any one of these is usually acceptable.

2. The process then uses the resource for an arbitrary time.

3. Finally, the process releases the resource, allowing a process waiting on the resource to continue.
Outcomes for Resource Allocation Requests

- A resource allocation request may have several possible outcomes:
  - The request may be successful, and the process can continue running.
  - The resource may be unavailable, and the process will block.
  - The resource may be unavailable, and the process receives an error indication.

- This last outcome is usually possible only if a process has indicated it does not wish to block if the resource is unavailable.
**Example Outcomes**

Let’s consider the Win32 system call `WaitForSingleObject`. This call has two parameters:

- A *handle* to the object (a semaphore, for example).
- A *timeout* parameter.

If the resource is unavailable, then if

- `timeout` is infinite, the calling thread will block.
- `timeout` = 0, the calling thread will return immediately with an appropriate error indication.
- `timeout` > 0 the thread will wait at most `timeout` for the resource to become available.
How Are Resources Requested?

- Some system calls, like open, explicitly request the use of resources.
- Other resources may be requested with an explicit “request” system call (e.g. the UNIX system call `fcntl` is used to “lock” regions of a file).
Coffman: Four Conditions Necessary for Deadlock

- **Mutual exclusion**: each resource is either assigned to a single process or is unavailable.
- **Hold and wait**: processes holding resources may request and wait for additional resources.
- **No preemption**: resources previously allocated cannot be forcibly removed from a process.
- **Circular wait**: there must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.
**Deadlock Modeling**

Resource Allocation Graphs are directed graphs that can be used to illustrate deadlock conditions:

- Resources are shown as squares labeled with the resource name.
- Processes are shown as circles labeled with the process identification.
- An edge from a resource to a process shows the process “owns” the resource.
- An edge from a process to a resource shows the process is requesting use of the resource.
Process A Holds Resource R
Process B Requests Resource S
Deadlock!
Deadlock Strategies

- Ignore the problem entirely
- Detect and recover
- Prevention
- Avoidance
Ignoring Deadlock

- Tanenbaum calls this the “ostrich algorithm.”
- This approach is appropriate if the cost of a deadlock is small compared to the cost of preventing or avoiding them.
- Many UNIX systems ignore deadlocks.
- Most mission-critical systems are designed with consideration for deadlock.
Detect and Recover

- Detection requires that the system maintain a complete “database” of all resources and owner processes and all pending requests.

- This database must be in a form usable for an algorithm that can detect cycles in graphs.

- For systems with one unit of each resource type, a simple cycle-detection algorithm can be used.
Deadlock Recovery

Techniques for Recovery:

- Preempt resources - not always possible
- Kill a process (releasing its resources) and restart it from the last checkpoint
- Kill a process which can be run again with no adverse side effects (changes)
- Kill a process which breaks the deadlock
Deadlock Prevention

- Impose restrictions on processes so deadlocks are structurally impossible.
- Solution approaches are keyed to the four conditions necessary for deadlock. Ensuring that at least one of the conditions will be satisfied guarantees that deadlock will be impossible.
Prevention: No Mutual Exclusion

- On the surface this doesn’t appear to be viable.
- However some resources are never shared, but owned by a single process that then accepts requests for the use of that resource.
- Example: printer spooling.
Prevention: Hold and Wait

- All processes would need to request all resources before they begin execution.
- Problem 1: this may not be possible, as some resource needs may be determined only at execution time.
- Problem 2: this approach doesn’t lead to efficient use of resources.
- Alternate method: processes must give up resources if a new request would block.
Prevention: No Preemption

- Preempting the use of an assigned resource is either costly or impossible for some resources.
- Examples: preempting a tape drive or printer
- Some resources may also be left with inconsistent states if they are preempted at arbitrary times.
Prevention: Circular Wait

- Assign each resource a unique numeric code.
- Processes must request resources in increasing order of their numeric codes.
- This technique has been used with success in some systems.
- Due to Havender, “Standard Allocation Patterns”
Deadlock Avoidance

Q. How do you avoid having people step in your rose garden?

A1. Nicely ask each person who might accidentally step in it to please not do so.

A2. Put up a fence some distance from the roses.

It is the second approach that is taken by deadlock avoidance: avoid getting into situations that might result in deadlock.
Resource Trajectories - I

- Draw a plot with one axis for each process.
- Each axis has increasing process “progress” (i.e. time) extending from the origin.
- In the appropriate regions on each axis draw orthogonal regions in which various resources are used.
- Intersecting regions with the same unit of the same resource are “forbidden.”
Resource Trajectories - II

Since progress in each process must be in one direction only, reaching a region in which any process progress is blocked by forbidden regions spells t-r-o-u-b-l-e!

Such regions are called unsafe.

Regions where at least one process can make progress are called safe.

Unsafe states are not deadlocked states.
Resource Trajectories - III

Process B

S & R

S

Process A

R

R & S

Deadlock

Unsafe
Dijkstra’s Banker’s Algorithm

- Originally expressed in terms of florins.
- Idea: Customers express their maximum need for money in advance. The banker has enough cash to satisfy each request, but not necessarily all requests at the same time. Thus some customers may have to wait for others to repay their loans before getting their cash.
- Problem: how can the banker make safe loans?
The Banker’s Algorithm

- Build a matrix that shows allocation to customers, and their maximum “claim.” We also keep track of the available cash.
- When a customer asks for more cash, the algorithm simulates making the loan, and then determines if all customers could eventually complete. If they can, then the “real” loan is given out to the customer. If not, the customer has to wait for the cash.
A Safe State (10 available units)

<table>
<thead>
<tr>
<th>Who</th>
<th>Allocated</th>
<th>Max. Claim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Barbara</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Bill</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Paula</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

With 10 available units, anyone can ask for and receive their maximum claim, after which they will be required to return it. All other users can then obtain their maximum claims and return them.
Another Safe State (2 available units)

<table>
<thead>
<tr>
<th>Who</th>
<th>Allocated</th>
<th>Max. Claim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Barbara</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Bill</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Paula</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

Bill can get 2 units and then repay his total of 4 units. Then either Barbara or Paula could get their maximum claim (and repay). Finally, Andy could get his maximum claim. Since all users can obtain their maximum claims, this is a safe state.
An Unsafe State (1 available unit)

<table>
<thead>
<tr>
<th>Who</th>
<th>Allocated</th>
<th>Max. Claim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Barbara</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Bill</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Paula</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

With this allocation of resources, there is no process that is guaranteed to be able to obtain its maximum claim. Of course, we don’t know for sure that any process will make such a claim. Nevertheless, this is an unsafe state.
Banker’s Algorithm for Multiple Resources

- Generalization of the single resource algorithm
- Keep two matrices, each having rows labeled with process names and columns labeled with resource names
- One matrix records assigned resources
- The other matrix records resources still needed
- We also need to know total units of all resources in the system.
Arrays for Multiple Resources

- E: a vector listing units of existing resources
- A: a vector listing available units of resources
- C: a matrix (processes by resources) listing current allocations of resources to processes
- R: a matrix (processes by resources) listing remaining resources that may be requested.

\[ \sum C_{ij} + A_j = E_j \]
The Algorithm’s Steps

1. Look for a row with unmet needs less than or equal to the available resources. If no such row exists, system will deadlock.

2. Assume the process for the row selected in 1 gets the needed resources and eventually finishes. Mark it complete and release its resources to the available vector.

3. Repeat steps 1 and 2 until all processes are marked complete or deadlock occurs.
### Resources assigned (C Matrix)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Resources still needed (R Matrix)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Summary vectors

- Existing resources (E): 6 3 4 2
- Possessed resources: 5 3 2 2
- Available resources (A): 1 0 2 0
Examples of the Banker’s Algorithm

- B requests 1 unit of the third resource; this request leads to a safe state.
- Then E requests 1 unit of the third resource; this request would result in deadlock, so E must wait.
**Two-phase locking**

- Frequently used in database systems with multiple concurrent users
- Process tries to lock all required records, one at a time.
- If successful, the record updates are done and all records are unlocked.
- If unsuccessful, existing locks are released and the sequence of locks is tried again.
Next…

- In the next module we’ll begin our study of input/output.
- We will examine input/output hardware characteristics in general, and look at a few devices in some detail.
- We’ll then study the way input/output devices are managed by the operating system and made available to application programs.