Part II
Process Management
Chapter 7: Deadlocks
System Model

- **System resources are utilized in the following way:**
  - **Request:** If a process makes a request to use a system resource which cannot be granted immediately, then the requesting process blocks until it can acquire the resource.
  - **Use:** The process can operate on the resource.
  - **Release:** The process releases the resource.

- **Deadlock:** A set of processes is in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set.
Deadlock: Necessary Conditions

For a deadlock to occur, each of the following four conditions must hold.

- **Mutual Exclusion**: At least one resource must be held in a non-sharable way.
- **Hold and Wait**: A process must be holding a resource and waiting for another.
- **No Preemption**: Resource cannot be preempted.
- **Circular Wait**: A waits for B, B waits for C, C waits for A.
Handling Deadlocks

- **Deadlock Prevention and Avoidance:** Make sure deadlock will never happen.
  - **Prevention:** Ensure one of the four conditions fails.
  - **Avoidance:** The OS needs more information so that it can determine if the current request can be satisfied or delayed.

- **Deadlock:** Allow a system to enter a deadlock situation, detect it, and recover.

- **Ignore Deadlock:** Pretend deadlocks never occur in the system.
Deadlock Prevention: 1/4
Mutual Exclusion

- By ensuring that at least one of the four conditions cannot hold, we can prevent the occurrence of a deadlock.
- **Mutual Exclusion**: Some sharable resources must be accessed exclusively (e.g., printer), which means we cannot deny the mutual exclusion condition.
Deadlock Prevention: 2/4

Hold and Wait

- No process can hold some resources and then request for other resources.
- Two strategies are possible:
  - A process must acquire all resources before it runs.
  - When a process requests for resources, it must hold none (i.e., returning resources before requesting for more).
- Resource utilization may be low, since many resources will be held and unused for a long time.
- Starvation is possible. A process that needs some popular resources may have to wait indefinitely.
Deadlock Prevention: 3/4

No Preemption

- Resources that are being held by the requesting process are preempted. There are two strategies:
  - If a process is holding some resources and requesting for some others that are being held by other processes, the resources of the requesting process are preempted. The preempted resources become available.
  - If the requested resources are not available:
    - If they are being held by processes that are waiting for additional resources, these resources are preempted and given to the requesting process.
    - Otherwise, the requesting process waits until the requested resources become available. While it is waiting, its resources may be preempted.
    - This works only if the state of the process and resources can be saved and restored easily (e.g., CPU & memory).
Deadlock Prevention: 4/4

Circular Waiting

- To break the circular waiting condition, we can order all resource types (e.g., tapes, printers).
- A process can only request resources higher than the resource types it holds.
- Suppose the ordering of tapes, disks, and printers are 1, 4, and 8. If a process holds a disk (4), it can only ask a printer (8) and cannot request a tape (1).
- A process must release some lower order resources to request a lower order resource. To get tapes (1), a process must release its disk (4).
- In this way, no deadlock is possible. Why?
Deadlock Avoidance: 1/5

- Each process provides the maximum number of resources of each type it needs.
- With these information, there are algorithms that can ensure the system will never enter a deadlock state. This is deadlock avoidance.
- A sequence of processes \(<P_1, P_2, \ldots, P_n>\) is a safe sequence if for each process \(P_i\) in the sequence, its resource requests can be satisfied by the remaining resources and the sum of all resources that are being held by \(P_1, P_2, \ldots, P_{i-1}\). This means we can suspend \(P_i\) and run \(P_1, P_2, \ldots, P_{i-1}\) until they complete. Then, \(P_i\) will have all resources to run.
Deadlock Avoidance: 2/5

- A state is *safe* if the system can allocate resources to each process (up to its maximum, of course) in some order and still avoid a deadlock.

- In other word, a state is *safe* if there is a safe sequence. Otherwise, if no safe sequence exists, the system state is *unsafe*.

- An unsafe state is not necessarily a deadlock state. On the other hand, a deadlock state is an unsafe state.
Deadlock Avoidance: 3/5

- A system has 12 tapes and three processes $A$, $B$, $C$. At time $t_0$, we have:

- Then, $<B, A, C>$ is a safe sequence (safe state).
- The system has $12-(5+2+2)=3$ free tapes.
- Since $B$ needs 2 tapes, it can take 2, run, and return 4. After $B$ completes, the system has $(3-2)+4=5$ tapes. $A$ now can take all 5 tapes and run. Finally, $A$ returns 10 tapes for $C$ to take 7 of them.

<table>
<thead>
<tr>
<th></th>
<th>Max needs</th>
<th>Current holding</th>
<th>Will need</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$B$</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$C$</td>
<td>9</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>
Deadlock Avoidance: 4/5

- A system has 12 tapes and three processes \(A\), \(B\), \(C\). At time \(t_1\), \(C\) has one more tape:

<table>
<thead>
<tr>
<th></th>
<th>Max needs</th>
<th>Current holding</th>
<th>Will need</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>(B)</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(C)</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

- The system has 12-(5+2+3)=2 free tapes.
- At this point, only \(B\) can take these 2 and run. It returns 4, making 4 free tapes available.
- But, none of \(A\) and \(C\) can run, and a deadlock occurs.
- The problem is due to granting \(C\) one more tape.
Deadlock Avoidance: 5/5

- A *deadlock avoidance algorithm* ensures that the system is always in a safe state. Therefore, no deadlock can occur.
- Resource requests are granted only if in doing so the system is still in a safe state.
- Consequently, resource utilization may be *lower* than those systems without using a deadlock avoidance algorithm.
Banker’s Algorithm: 1/2

- The system has \( m \) resource types and \( n \) processes.
- Each process must declare its maximum needs.
- The following arrays are used:

  - \( \text{Available}[1..m] \): one entry for each resource. \( \text{Available}[i]=k \) means resource type \( i \) has \( k \) units available.
  - \( \text{Max}[1..n,1..m] \): maximum demand of each process. \( \text{Max}[i,j]=k \) means process \( i \) needs \( k \) units of resource \( j \).
  - \( \text{Allocation}[1..n,1..m] \): resources allocated to each process. \( \text{Allocation}[i,j]=k \) means process \( i \) is currently allocated \( k \) units of resource \( j \).
  - \( \text{Need}[1..n,1..m] \): the remaining resource need of each process. \( \text{Need}[i,j]=k \) means process \( i \) needs \( k \) more units of resource \( j \).
Banker’s Algorithm: 2/2

We will use $A[i, \ast]$ to indicate the $i$-th row of matrix $A$.

Given two arrays $A[1..m]$ and $B[1..m]$, $A \leq B$ if $A[i] \leq B[i]$ for all $i$. Given two matrices $A[1..n,1..m]$ and $B[1..n,1..m]$, $A[i, \ast] \preceq B[i, \ast]$ if $A[i,j] \leq B[i,j]$ for all $j$.

When a resource request is made by process $i$, this algorithm calls the Resource-Request algorithm to determine if the request can be granted. The Resource-Request algorithm calls the Safety Algorithm to determine if a state is safe.
Safety Algorithm

1. Let $Work[1..m]$ and $Finish[1..n]$ be two working arrays.

2. $Work := Available$ and $Finish[i]=FALSE$ for all $i$

3. Find an $i$ such that both
   - $Finish[i] = FALSE$ \hspace{1cm} // process $i$ is not yet done
   - $Need[i,\ast] \leq Work$ \hspace{1cm} // its need can be satisfied

   If no such $i$ exists, go to Step 5

4. $Work = Work + Allocation[i,\ast]$ \hspace{1cm} // run it and reclaim
   $Finish[i] = TRUE$ \hspace{1cm} // process $i$ completes
   go to Step 3

5. If $Finish[i] = TRUE$ for all $i$, the system is in a safe state.
Resource-Request Algorithm

1. Let $\text{Request}[1..n,1..m]$ be the request matrix. $\text{Request}[i,j]=k$ means process $i$ requests $k$ units of resource $j$.

2. If $\text{Request}[i,\ast] \leq \text{Need}[i,\ast]$, go to Step 3. Otherwise, it is an error.

3. If $\text{Request}[i,\ast] \leq \text{Available}$, go to Step 4. Otherwise, process $i$ waits.

4. Do the following:
   
   $\text{Available} = \text{Available} - \text{Request}[i,\ast]$
   
   $\text{Allocation}[i,\ast] = \text{Allocation}[i,\ast] + \text{Request}[i,\ast]$
   
   $\text{Need}[i,\ast] = \text{Need}[i,\ast] - \text{Request}[i,\ast]$

   If the result is a safe state (Safety Algorithm), the request is granted. Otherwise, process $i$ waits and the resource-allocation tables are restored back to the original.
Consider a system of 5 processes $A$, $B$, $C$, $D$ and $E$, and 3 resource types ($X=10$, $Y=5$, $Z=7$). At time $t_0$, we have

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need=Max-Alloc</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$Y$</td>
<td>$Z$</td>
<td>$X$</td>
</tr>
<tr>
<td>$A$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$B$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$C$</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$D$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$E$</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

A safe sequence is $<B,D,E,C,A>$. Since $B$’s $[1,2,2] \leq \text{Avail}$’s $[3,3,2]$, $B$ runs. Then, $\text{Avail}=[2,0,0]+[3,3,2]=[5,3,2]$. $D$ runs next. After this, $\text{Avail}=[5,3,2]+[2,1,1]=[7,4,3]$. $E$ runs next. $\text{Avail}=[7,4,3]+[0,0,2]=[7,4,5]$. Since $C$’s $[6,0,0] \leq \text{Avail}=[7,4,5]$, $C$ runs. After this, $\text{Avail}=[7,4,5]+[3,0,2]=[10,4,7]$ and $A$ runs. There are other safe sequences: $<D,E,B,A,C>$, $<D,B,A,E,C>$, ...
Example: 2/4

- Now suppose process $B$ asks for 1 $X$ and 2 $Z$s. More precisely, $\text{Request}_B = [1,0,2]$. Is the system still in a safe state if this request is granted?

- Since $\text{Request}_B = [1,0,2] \leq \text{Available} = [3,3,2]$, this request may be granted as long as the system is safe.

- If this request is actually granted, we have the following:

<table>
<thead>
<tr>
<th></th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$[3,0,2] = [2,0,0] + [1,0,2]$  $[0,2,0] = [1,2,2] - [1,0,2]$  $[2,3,0] = [3,3,2] - [1,0,2]$
Example: 3/4

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need=Max-Alloc</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Y Z</td>
<td>X Y Z</td>
<td>X Y Z</td>
<td>X Y Z</td>
</tr>
<tr>
<td>A</td>
<td>0 1 0</td>
<td>7 5 3</td>
<td>7 4 3</td>
</tr>
<tr>
<td>B</td>
<td>3 0 2</td>
<td>3 2 2</td>
<td>0 2 0</td>
</tr>
<tr>
<td>C</td>
<td>3 0 2</td>
<td>9 0 2</td>
<td>6 0 0</td>
</tr>
<tr>
<td>D</td>
<td>2 1 1</td>
<td>2 2 2</td>
<td>0 1 1</td>
</tr>
<tr>
<td>E</td>
<td>0 0 2</td>
<td>4 3 3</td>
<td>4 3 1</td>
</tr>
</tbody>
</table>

Is the system in a safe state after this allocation?

Yes, because the safety algorithm will provide a safe sequence <B,D,E,A,C>. Verify it by yourself.

Therefore, B’s request of [1,0,2] can safely be made.
Example: 4/4

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need=Max-Alloc</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- After this allocation, E’s request $\text{Request}_E=[3,3,0]$ cannot be granted since $\text{Request}_E=[3,3,0] \leq [2,3,0]$ is false.
- A’s request $\text{Request}_A=[0,2,0]$ cannot be granted because the system will be unsafe.
- If $\text{Request}_A=[0,2,0]$ is granted, $\text{Available}=[2,1,0]$.
- None of the five processes can finish and the system is unsafe.
Deadlock Detection

- If a system does not use a deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. Thus, we need
  - An algorithm that can examine the system state to determine if a deadlock has occurred. This is a *deadlock detection* algorithm.
  - An algorithm that can help recover from a deadlock. This is a *recovery* algorithm.
- A deadlock detection algorithm does not have to know the maximum need \( \text{Max} \) and the current need \( \text{Need} \). It uses only \( \text{Available} \), \( \text{Allocation} \) and \( \text{Request} \).
Deadlock Detection Algorithm

1. Let $Work[1..m]$ and $Finish[1..n]$ be two working arrays.
2. $Work := Available$ and $Finish[i] = FALSE$ for all $i$
3. Find an $i$ such that both
   - $Finish[i] = FALSE$  // process $i$ is not yet done
   - $Request[i, *] \leq Work$  // its request can be satisfied
   If no such $i$ exists, go to Step 5
4. $Work = Work + Allocation[i, *]$  // run it and reclaim
   $Finish[i] = TRUE$  // process $i$ completes
   go to Step 3
5. If $Finish[i] = TRUE$ for all $i$, the system is in a safe state. If $Finish[i] = FALSE$, then process $P_i$ is deadlocked.

*Use Request here rather than Need in the safety algorithm*
Example: 1/2

Suppose maximum available resource is [7,2,6] and the current state of resource allocation is shown above.

Is the system deadlocked? No. We can run A first, making Available=[0,1,0].

Then, we run C, making Available=[3,1,3]. This is followed by D, making Available=[5,2,4], and followed by B and E.
Example: **2/2**

<table>
<thead>
<tr>
<th></th>
<th><strong>Allocation</strong></th>
<th><strong>Request</strong></th>
<th><strong>Available</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>X</strong></td>
<td><strong>Y</strong></td>
<td><strong>Z</strong></td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- Suppose **C** requests for one more resource **Z**.
- Now, **A** can run, making **Available** = [0,1,0].
- However, none of **B**, **C**, **D** and **E** can run. Therefore, **B**, **C**, **D** and **E** are deadlocked!
The Use of a Detection Algorithm

Frequency

- If deadlocks occur frequently, then the detection algorithm should be invoked frequently.
- Once per hour or whenever CPU utilization becomes low (i.e., below 40%). Low CPU utilization means more processes are waiting.
How to Recover: 1/3

- When a detection algorithm determines a deadlock has occurred, the algorithm may inform the system administrator to deal with it. Of, allow the system to *recover* from a deadlock.

- There are two options.
  - Process Termination
  - Resource Preemption

- These two options are not mutually exclusive.
Recovery: Process Termination: 2/3

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated

Problems:
- Aborting a process may not be easy. What if a process is updating or printing a large file? The system must find some way to maintain the state of the file and printer before they can be reused.
- The termination may be determined by the priority/importance of a process.
Selecting a victim: which resources and which processes are to be preempted?

Rollback: If we preempt a resource from a process, what should be done with that process?
- Total Rollback: abort the process and restart it
- Partial Rollback: rollback the process only as far as necessary to break the deadlock.

Starvation: We cannot always pick the same process as a victim. Some limit must be set.