Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
Deadlocks, Deadlocks prevention, detection, and avoidance
Deadlocks Avoidance (Banker’s Algo)

If a process requests a resource that is *currently available*, it may still have to wait.
Deadlocks Avoidance (Banker’s Algo)

Definition

**Banker’s Algorithm.** It is an algorithm that allows the avoidance of deadlocks by determining whether a safe sequence $S$ exists or not.

- Banker’s: The algorithm could be used in banking system to ensure that the bank always satisfies the requests of its customers.
- The algorithm uses four of data structures:
  1. **Available.** A vector of size $m$ (resource types) to express the currently available number of instances for a given resource $R_j$.
     \[
     \text{Available}[j] = k : \text{there are } k \text{ instances of resource type } R_j
     \]
  2. **Max.** A matrix of size $n \times m$ to express the maximum number of resources to be requested by a given process.
     \[
     \text{Max}[i,j] = k : \text{at most } k \text{ instances of resource type } R_j \text{ will be requested by process } P_i.
     \]
Deadlocks Avoidance (Banker’s Algo)

Definition

Banker’s Algorithm. It is an algorithm that allows the avoidance of deadlocks by determining whether a safe sequence $S$ exists or not.

- Banker’s: The algorithm could be used in banking system to ensure that the bank always satisfies the requests of its customers.
- The algorithm uses four data structures:
  3. **Allocation.** A matrix of size $n \times m$ to express the currently held instances of each resource type by a given process.
     \[
     [\text{Allocation}[i, j] = k] : \text{Currently there are } k \text{ instances of resource type } R_j \text{ held by process } P_i.
     \]
  4. **Need.** A matrix of size $n \times m$ to express the remaining number of instances of each resource type to be requested by a given process.
     \[
     [\text{Need}[i, j] = k] : \text{There are } k \text{ instances of resource type } R_j \text{ to be requested by process } P_i \text{ at anytime in the future.}
     \]

\[
\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]
\]
Deadlocks Avoidance (Banker’s Algo)

Resource-Request Algorithm.

Let Request\(_i\) be the request vector for process \(P_i\):

1. If Request\(_i\) \leq\ Need\(_i\), goto 2, else Error.
2. If Request\(_i\) \leq\ Available, goto 3, else \(P_i\) must wait.
3. The system pretends having allocated the requested resources to \(P_i\) by updating the data structures:

\[
\begin{align*}
\text{Available} &= \text{Available} - \text{Request}_i; \\
\text{Allocation}_i &= \text{Allocation}_i + \text{Request}_i; \\
\text{Need}_i &= \text{Need}_i - \text{Request}_i;
\end{align*}
\]

A process \(P_i\) that asks for more than what it has initially declared (i.e., Max\(_i\)) will be terminated by the system.
Deadlocks Avoidance (Banker’s Algo)

Assuming that a given resource request e.g., \( \text{Request}_i \) has been fulfilled, the system is brought to a new state: 

**Banker’s Algorithm.**

1. Let \( \text{Work} \) and \( \text{Finish} \) be vectors of length \( m \) and \( n \) respectively such that \( \text{Work} = \text{Available} \) and \( \forall i: \text{Finish}[i] = \text{false} \).

2. Find an index \( i \):
   - \( \text{Finish}[i] == \text{false} \)
   - \( \text{Need}_i \leq \text{Work} \)

   Else goto 4.

3. \( \text{Work} = \text{Work} + \text{Allocation}_i \); 

   \( \text{Finish}[i] = \text{true} \); 

   goto 2;

4. If \( \forall i: \text{Finish}[i] == \text{true} \) then the system is in safe state.
Deadlocks Avoidance (Banker’s Algo)

Banker’s algorithm example

A system needs $O(m \times n^2)$ operations to execute the banker’s algorithm, where $n$ is the number of processes and $m$ the number of resources.
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
10   5   7

Allocation
R1  R2  R3
P0  0   0   0
P1  0   0   0
P2  0   0   0
P3  0   0   0
P4  0   0   0

Max
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Need
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Process P0
Process P1
Process P2
Process P3
Process P4
Deadlocks Avoidance (Banker’s Algo)

<table>
<thead>
<tr>
<th>Resource 1</th>
<th>Resource 2</th>
<th>Resource 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Available
R1  R2  R3
3  3  2

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1  R2  R3</td>
<td>R1  R2  R3</td>
<td>R1  R2  R3</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>P0 0 1 0</td>
<td>P0 7 5 3</td>
<td>P0 7 4 3</td>
</tr>
<tr>
<td>P1 2 0 0</td>
<td>P1 3 2 2</td>
<td>P1 1 2 2</td>
</tr>
<tr>
<td>P2 3 0 2</td>
<td>P2 9 0 2</td>
<td>P2 6 0 0</td>
</tr>
<tr>
<td>P3 2 1 1</td>
<td>P3 2 2 2</td>
<td>P3 0 1 1</td>
</tr>
<tr>
<td>P4 0 0 2</td>
<td>P4 4 3 3</td>
<td>P4 4 3 1</td>
</tr>
</tbody>
</table>

Processes:
- Process P0
- Process P1
- Process P2
- Process P3
- Process P4
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
3    3    2

Allocation
R1  R2  R3
P0  0   1   0
P1  2   0   0
P2  3   0   2
P3  2   1   1
P4  0   0   2

Max
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Need
R1  R2  R3
P0  7   4   3
P1  1   2   2
P2  6   0   0
P3  0   1   1
P4  4   3   1

We claim that the system is currently in safe state.
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
3   3   2

Allocation
R1  R2  R3
P0  0   1   0
P1  2   0   0
P2  3   0   2
P3  2   1   1
P4  0   0   2

Max
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Need
R1  R2  R3
P0  7   4   3
P1  1   2   2
P2  6   0   0
P3  0   1   1
P4  4   3   1

Suppose now: P1 issues a request for resources Request \_1 = (1, 0, 2)
# Deadlocks Avoidance (Banker’s Algo)

### Available Resources

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### Allocation of Resources

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Max and Need Resources

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Suppose now: P1 issue a request for resource Request_1 = (1, 0, 2)

Check: \[ \text{Request}_1 \leq \text{Available} : (1, 0, 2) \leq (3, 3, 2) \]
Deadlocks Avoidance (Banker’s Algo)

Recall: Resource-Request Algorithm.

Let Request\(_i\) be the request vector for process \(P_i\):

1. If Request\(_i\) \(\leq\) Need\(_i\), goto 2, else Error.
2. If Request\(_i\) \(\leq\) Available, goto 3, else \(P_i\) must wait.
3. The system pretends having allocated the requested resources to \(P_i\) by updating the data structures:

\[
\begin{align*}
\text{Available} &= \text{Available} - \text{Request}_i; \\
\text{Allocation}_i &= \text{Allocation}_i + \text{Request}_i; \\
\text{Need}_i &= \text{Need}_i - \text{Request}_i;
\end{align*}
\]

This algorithm allows to determine whether a request can be safely granted or not.
**Deadlocks Avoidance (Banker’s Algo)**

### Available Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

### Allocation

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Max Resources

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### Need Resources

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Suppose now: P1 issues a request for resources Request_1 = (1, 0, 2)
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
2    3    0

Allocation
R1  R2  R3
P0  0   1   0
P1  3   0   2
P2  3   0   2
P3  2   1   1
P4  0   0   2

Max
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Need
R1  R2  R3
P0  7   4   3
P1  0   2   0
P2  6   0   0
P3  0   1   1
P4  4   3   1

The system has reached a new state
## Deadlocks Avoidance (Banker’s Algo)

<table>
<thead>
<tr>
<th>Resource 1</th>
<th>Resource 2</th>
<th>Resource 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

**Available**
- R1: 2
- R2: 3
- R3: 0

**Allocation**

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Max**

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Need**

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**The system has reached a new state**

**Safe or not?**
Recall: Banker’s Algorithm.

1. Let Work and Finish be vectors of length m and n respectively such that Work=Available and \( \forall i: \) Finish\[i\]=false.

2. Find an index \( i \):
   - Finish\[i\] == false
   - Need\(i\) \( \leq \) Work

   Else goto 4.

3. Work = Work + Allocation;

   Finish\[i\]=true;

   goto 2;

4. If \( \forall i: \) Finish\[i\]=true then the system is in safe state.
Deadlocks Avoidance (Banker’s Algo)

<table>
<thead>
<tr>
<th>Available</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 R2 R3</td>
<td>R1 R2 R3</td>
<td>R1 R2 R3</td>
</tr>
<tr>
<td>P0 0 1 0</td>
<td>P0 7 5 3</td>
<td>P0 7 4 3</td>
</tr>
<tr>
<td>P1 3 0 2</td>
<td>P1 3 2 2</td>
<td>P1 0 2 0</td>
</tr>
<tr>
<td>P2 3 0 2</td>
<td>P2 9 0 2</td>
<td>P2 6 0 0</td>
</tr>
<tr>
<td>P3 2 1 1</td>
<td>P3 2 2 2</td>
<td>P3 0 1 1</td>
</tr>
<tr>
<td>P4 0 0 2</td>
<td>P4 4 3 3</td>
<td>P4 4 3 1</td>
</tr>
</tbody>
</table>

The system has reached a new state
The state is safe because there exists a safe sequence <P1, P3, P4, P0, P2>
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
2   3   0

Allocation
R1  R2  R3
P0  0   1   0
P1  3   0   2
P2  3   0   2
P3  2   1   1
P4  0   0   2

Max
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Need
R1  R2  R3
P0  7   4   3
P1  0   2   0
P2  6   0   0
P3  0   1   1
P4  4   3   1

Hence, we can immediately grant the request of process P1

Process P0
Process P1
Process P2
Process P3
Process P4
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
2    3    0

Allocation
R1  R2  R3
P0  0   1   0
P1  3   0   2
P2  3   0   2
P3  2   1   1
P4  0   0   2

Max
R1  R2  R3
P0  7   5   3
P1  3   2   2
P2  9   0   2
P3  2   2   2
P4  4   3   3

Need
R1  R2  R3
P0  7   4   3
P1  0   2   0
P2  6   0   0
P3  0   1   1
P4  4   3   1

Now if process P4 requests the system
Request_4 = (3, 3, 0)
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
2  3  0

Allocation
R1  R2  R3
P0  0  1  0
P1  3  0  2
P2  3  0  2
P3  2  1  1
P4  0  0  2

Max
R1  R2  R3
P0  7  5  3
P1  3  2  2
P2  9  0  2
P3  2  2  2
P4  4  3  3

Need
R1  R2  R3
P0  7  4  3
P1  0  2  0
P2  6  0  0
P3  0  1  1
P4  4  3  1

Now if process P4 requests the system
Request = (3, 3, 0)
Will not be granted
Deadlocks Avoidance (Banker’s Algo)

**Available**

R1  R2  R3  
2    3    0

<table>
<thead>
<tr>
<th>Allocation</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Max</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>R2</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>R3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Need</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>R2</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>R3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Now if process P4 requests the system
Request P4 = (3, 3, 0)

Because there is not enough resources available
Deadlocks Avoidance (Banker’s Algo)

<table>
<thead>
<tr>
<th>Resource 1</th>
<th>Resource 2</th>
<th>Resource 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Available
R1  R2  R3
2    3    0

Allocation
R1  R2  R3
P0  0  1  0
P1  3  0  2
P2  3  0  2
P3  2  1  1
P4  0  0  2

Max
R1  R2  R3
P0  7  5  3
P1  3  2  2
P2  9  0  2
P3  2  2  2
P4  4  3  3

Need
R1  R2  R3
P0  7  4  3
P1  0  2  0
P2  6  0  0
P3  0  1  1
P4  4  3  1

Now if process P0 requests the system
Request_0 = (0, 2, 0)
Deadlocks Avoidance (Banker’s Algo)

Available
R1  R2  R3
2    3    0

Allocation
<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Max
<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Need
<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Now if process P0 requests the system
Request \_0 = (0, 2, 0)

Will not be granted
Deadlocks Avoidance (Banker’s Algo)

Available
R1 R2 R3
2 3 0

Allocation
R1 R2 R3 R0
P0 0 1 0
P1 3 0 2
P2 3 0 2
P3 2 1 1
P4 0 0 2

Max
R1 R2 R3
P0 7 5 3
P1 3 2 2
P2 9 0 2
P3 2 2 2
P4 4 3 3

Need
R1 R2 R3
P0 7 4 3
P1 0 2 0
P2 6 0 0
P3 0 1 1
P4 4 3 1

Now if process P0 requests the system
Request_0 = (0, 2, 0)

Because there will be no safe sequence
Deadlocks Avoidance (Banker’s Algo)

**Summery.** The banker’s algorithm:

- Simulate “granting the request”.
- In the new system state, tries to determine the existence of an ordering of processes which execution does not end up in a deadlock.
- That ordering $P_1, P_2, ..., P_n$, called safe sequence, is such that:
  - Process $P_1$ can have all possible requests satisfied (up to max).
  - Process $P_1$ runs till completion and releases the resources.
  - Process $P_2$ can have all possible requests satisfied (up to max).
  - Process $P_2$ runs till completion and releases the resources.
  - ...
  - Process $P_n$ runs till completion and releases the resources.
- If such sequence exists, the system is in a safe state.
- If the system is safe, the OS grants the request, else it make the process wait.
End.