Deadlock Prevention

- Mutual Exclusion
  - Must be supported by the operating system
- Hold and Wait
  - Require a process request all of its required resources at one time

Deadlock Prevention

- No Preemption
  - Process must release resource and request again
  - Operating system may preempt a process to require it releases its resources
- Circular Wait
  - Define a linear ordering of resource types
Deadlock Avoidance

- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process request

Two Approaches to Deadlock Avoidance

- Do not start a process if its demands might lead to deadlock
- Do not grant an incremental resource request to a process if this allocation might lead to deadlock
Resource Allocation Denial

- **Banker’s algorithm**
- **State of the system:** the current allocation of resources to processes
- **Safe state:** there is at least one sequence that does not result in deadlock
- **Unsafe state:** a state that is not safe

Determination of a Safe State

Initial State

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

Claim matrix C

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</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>

Allocation matrix A

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

Resource vector R

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Available vector V

(a) Initial state
Determination of a Safe State
P2 Runs to Completion

Determination of a Safe State
P1 Runs to Completion
Determination of a Safe State
P3 Runs to Completion

\[
\begin{array}{c|c|c}
R1 & R2 & R3 \\
\hline
P1 & 0 & 0 \\
P2 & 0 & 0 \\
P3 & 0 & 0 \\
P4 & 4 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c}
R1 & R2 & R3 \\
\hline
P1 & 0 & 0 \\
P2 & 0 & 0 \\
P3 & 0 & 0 \\
P4 & 0 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c}
R1 & R2 & R3 \\
\hline
P1 & 0 & 0 \\
P2 & 0 & 0 \\
P3 & 0 & 0 \\
P4 & 4 & 2 \\
\end{array}
\]

Claim matrix C
Allocation matrix A
C - A

\[
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
9 & 3 & 6 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
9 & 3 & 4 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
2 & 0 & 2 \\
\end{array}
\]

(d) P3 runs to completion

Determination of an Unsafe State

\[
\begin{array}{c|c|c}
R1 & R2 & R3 \\
\hline
P1 & 3 & 2 \\
P2 & 1 & 3 \\
P3 & 3 & 1 \\
P4 & 4 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c}
R1 & R2 & R3 \\
\hline
P1 & 1 & 0 \\
P2 & 5 & 1 \\
P3 & 2 & 1 \\
P4 & 0 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
P1 & 2 & 2 & 2 \\
P2 & 1 & 0 & 2 \\
P3 & 1 & 0 & 3 \\
P4 & 4 & 2 & 0 \\
\end{array}
\]

Claim matrix C
Allocation matrix A
C - A

\[
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
9 & 3 & 6 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
1 & 1 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
R1 & R2 & R3 \\
\hline
2 & 2 & 2 \\
\end{array}
\]

(a) Initial state
Determination of an Unsafe State

![Diagram showing matrices and vectors](image)

(b) P1 requests one unit each of R1 and R3

Deadlock Avoidance Logic

(a) global data structures

```c
struct state {
  int resource[n];
  int available[n];
  int claim[n];
  int alloc[n];
}
```

(b) resource alloc algorithm

```c
if (alloc[i] + request[*] > claim[i]) /* total request > claim*/
  < error >
else if (request[*] > available[*])
  < suspend process >
else
  < simulate alloc >
  < define new state by:
  alloc[i] = alloc[i] + request[*];
  available[*] = available[*] - request[*];
  if (safe (newstate))
    < carry out allocation >
  else
    < restore original state >;
  < suspend process >;
```
Deadlock Avoidance Logic

```java
boolean safe (state S) {
    int currentavail[m];
    process rest(<number of processes>);
    currentavail = available;
    rest = all processes;
    possible = true;
    while (possible) {
        <find a process Pk in rest such that
        claim [k,*] - alloc [k,*] <= currentavail;>
        if [found] /* simulate execution of Pk */
            currentavail = currentavail + alloc [k,*];
            rest = rest -(Pk);
        else
            possible = false;
    } return(rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

Deadlock Avoidance

- Maximum resource requirement must be stated in advance
- Processes under consideration must be independent; no synchronization requirements
- There must be a fixed number of resources to allocate
- No process may exit while holding resources