TTIT62 Real-time Process Control

Session topic: Banker's algorithm

Simin Nadjm-Tehrani

Real-time Systems Laboratory

Department of Computer and Information Science
Linköping university

- Allocate multiple resources as and when processes ask for it, but only:
  - up to a predefined max value for each process and resource
  - provided that remaining resources together with potential future releases are enough for future allocations (up to the max value)

Implementation

For n processes and m resources we need following data structures:

$$\boxed{\mathit{Max}\text{: } n \times m \text{ matrix}}$$

Max[i,j] = *k* means that process *i* requires max *k* elements of resource type *j*

$\boxed{\texttt{Allocation:}\ n \times m\ \texttt{matrix}}$

$\texttt{Allocation[i,j]} = k$ means that process i has already been allocated k elements of resource type j

$\boxed{\texttt{Available:}\ m\ \texttt{vector}}$

$\texttt{Available[i]} = k$ means that k elements of resource type i are available for allocation

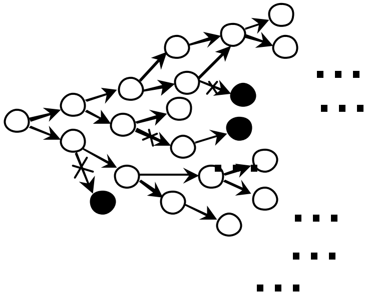
$Request_i$: m vector

process i:s request for resources

Notation:

$Allocation_i$: the *i-th* row in the *Allocation* matrix

State: instantiations of *Allocation*



Banker's algorithm

Input:

Matrix *Max*, vector *Available*,

a given state, and

$Request_i$ for some process i

Output:

Yes + new state, or

No + unchanged state

($Request_i$ can not be allocated now)

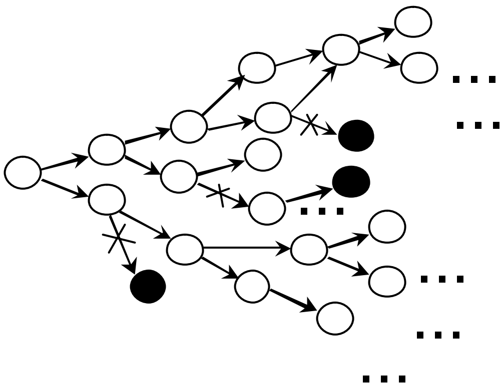
Algorithm:

1. Need := Max;
2. Check if $Request_i \leq Available$ if not, return "No".
3. Pretend that resources in $Request_i$ are to be allocated, compute new state.

$$Allocation_i := Allocation_i + Request_i$$
$$Need_i := Need_i - Request_i$$
$$Available := Available - Request_i$$

4. Test if the new state is
is deadlock-avoiding, in which case return ”Yes”.

Otherwise, return ”No” -
roll back to the old state.



Testing for deadlock-avoidance

Start with a given *Allocation* and check if it is deadlock-avoiding According to the 3-step algorithm below.

Finish: *n* vector with Boolean values (initially false)

Work : *m* vector denotes the changing resource set as the processes become ready and release resources (initially *Work := Available*)

1. Check if there is some process *i* for which $Finish_i$ = *false* and for which $Need_i \leq Work$. If there is no such process *i*, go to step 3.

2. Free the resources that i has used to get finished:

$Work := Work + Allocation_i$

$Finish_i := true$

continue from step 1.

3. If $Finish_i = true$ for all i then the initial state is deadlock-avoiding, otherwise it is not.