Operating Systems

Bankers Algorithm

Banker's Algorithm (Deadlock Avoidance Algorithm)

- Multiple instances of each resource type.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Need the following data structures, where **n** is the number of processes in the system and **m** is the number of resource types.

- **`Available:`** Array of length **m,** indicates the number of available resources of each type.
- **`Max:`** **n** by **m** matrix, defines maximum demand of each process for each type.
- **`Allocation:`** **n** by **m** matrix, defines number of assigned resources to each process for each type.
- **`Need:`** **n** by **m** matrix, defines number of remaining resources of each type needed by each process.

```
Need [i,j] = Max[i,j] – Allocation [i,j]
```

Banker's Safety Algorithm

Algorithm for finding out whether or not a system is in a safe state

1. Let **Work** and **Finish** be vectors of length m and n, respectively. Initialize
   `Work = Available` and
   `Finish[i] = false` for i = 0, 1, ..., n − 1.
2. Find an index `i` such that both of the following meets,
   `Finish[i] == false`
   $\texttt{Need}_i \leq \texttt{Work}$
   if no such `i` exists, go to step 4.
3. $\texttt{Work} = \texttt{Work} + \texttt{Allocation}_i$
   `Finish[i] = true`
   Go to step 2.
4. If `Finish[i] == true` for all *i*, then the system is in a safe state.

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ Work then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

finish:	false	false	false	false	false

work:	3	3	2

i = 0

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	false	false	false	false	false

work:	3	3	2

`i = 0`
`(7,4,3) <= (3,3,2) == false`

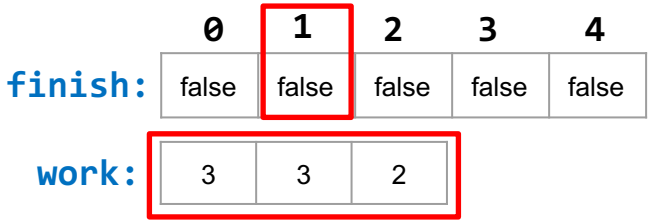
Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.




```
i = 1
(1,2,2) <= (3,3,2) == true

work = (3,3,2) + (2,0,0)
Finish[1] = true
```

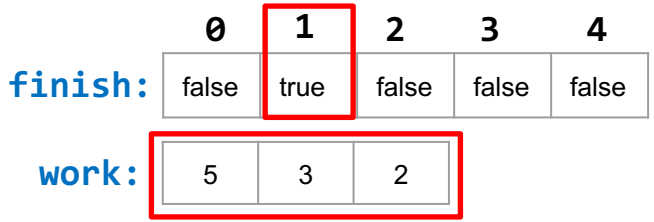
Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.




```
i = 1
(1,2,2) <= (3,3,2) == true
```

< P_1 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$ } Step 3
`Finish[i] = true`
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	false	true	false	false	false

work:	5	3	2

i = 2
(6,0,0) <= (5,3,2) == false

< P_1 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	false	true	false	false	false

work:	5	3	2

i = 3
(2,1,1) <= (5,3,2) == true

work = (5,3,2) + (2,1,1)
Finish[3] = true

< P_1 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	false	true	false	true	false

work:	7	4	3

i = 3
(2,1,1) <= (5,3,2) == true

< P_1, P_3 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

< P_1, P_3 >

	0	1	2	3	4
finish:	false	true	false	true	false

work:	7	4	3

i = 4
(0,0,2) <= (7,4,3) == true

work = (7,4,3) + (0,0,2)
Finish[4] = true

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	false	true	false	true	true

work:	7	4	5

```
i = 4
(0,0,2) <= (7,4,3) == true
```

< P_1, P_3 , P_4 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if finish[i] == false & $Need_i \leq Work$ then
    Work = Work + $Allocation_i$
    Finish[i] = true
} Step 3
Step 4. if finish[i] == true for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	false	true	false	true	true

work:	7	4	5

i = 0
(7,4,3) <= (7,4,5) == true

work = (7,4,5) + (0,1,0)
Finish[0] = true

< P_1, P_3 , P_4 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$
`Finish[i] = true`
} Step 3
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	true	true	false	true	true

work:	7	5	5

i = 0
(7,4,3) <= (7,4,5) == true

< P_1, P_3 , P_4 , P_0 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Step 2. iterate through i with i = 0 initially
if `finish[i] == false & Need`$_i$ ≤ `Work` then
`Work = Work + Allocation`$_i$ } Step 3
`Finish[i] = true`
Step 4. if `finish[i] == true` for all *i*, then the system is in a safe state.

	0	1	2	3	4
finish:	true	true	false	true	true

work:	7	5	5

```
i = 2
(6,0,0) <= (7,5,5) == true

work = (7,5,5) + (3,0,2)
Finish[2] = true
```

< P_1, P_3 , P_4 , P_0 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

	0	1	2	3	4
finish:	true	true	true	true	true

work:	10	5	7

Step 2. iterate through i with i = 0 initially
if finish[i] == false & $Need_i \leq Work$ then
    $Work = Work + Allocation_i$
    Finish[i] = true
} **Step 3**
Step 4. if finish[i] == true for all *i*, then the system is in a safe state.

< P_1, P_3 , P_4 , P_0 , P_2 >

Banker's Algorithm Example

- 5 processes P0 through P4;
- 3 resource types A (10 instances), B (5 instances), and C (7 instances)
- at time T0, the following snapshot of the system:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

	0	1	2	3	4
finish:	true	true	true	true	true

work:	10	5	7

Step 2. iterate through i with i = 0 initially
if finish[i] == false & $Need_i \leq Work$ then
    $Work = Work + Allocation_i$
    Finish[i] = true
} Step 3

Step 4. if finish[i] == true for all *i*, then the system is in a safe state.

The system is in a safe state since the sequence < P_1, P_3, P_4, P_0, P_2 > satisfies safety criteria

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i

If `Request[i][j] = k` then process P_i wants k instances of resource type R_j

1. If $Request_i <= Need_i$ go to step 2.
   Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i <= Available$, go to step 3.
   Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

```
Available = Available - Requesti;
Allocationi = Allocationi + Requesti;
Needi = Needi – Requesti;
```

4. Check Bankers Safety Algorithm, if this new state is safe.
   - ➔ If safe the resources are allocated to P_i.
   - ➔ If unsafe P_i must wait, and the old resource-allocation state is restored

What if P_1 Request (1,0,2) more ?

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- To decide whether this request can be immediately granted,
  - we first check that **$Request_1 \leq Need_1$** — that is, $(1,0,2) \leq (1,2,2)$? => TRUE
  - Then we check that **$Request_1 \leq Available$** — that is, $(1,0,2) \leq (3,3,2)$? => TRUE
  - If both is true,

    $Available = (3,3,2) - (1,0,2) = (2,3,0)$

    $Allocation_1 = (2,0,0) + (1,0,2) = (3,0,2)$

    $Need_1 = (1,2,2) - (1,0,2) = (0,2,0)$
- So, we arrive at the following new state

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Is the system in safe state?
Yes. Because we find a safe sequence: <P_1, P_3, P_4, P_0, P_2>.
Grant the request of P_1.

What about request of (3 3 0) resources for P4?
What about request of (0 2 0) resources for P0?