

Concurrent Architectures: Bankers Algorithm

Alistair Rendell

Also, see wikipedia page

Bankers Algorithm

- Handles deadlock scenarios when there are multiple instances for different resource types
- Key aspects:

Quantity	Dim	Meaning
n	1	No of processes
m	1	No of resource types
alloc	[n,m]	Resources allocated to process
free	[m]	Unallocated resources
claim	[n,m]	Resources to be required by process
request	[n,m]	Resources requested by process

Example 1

	R1	R2	R3	R4	R5	
R=	2	1	1	2	1	
Alloc(Req)	1 0	0 1	1 0	1 0	0 1	P1
	1 0	1 0	0 1	0 0	0 1	P2
	0 0	0 0	0 0	1 0	0 1	P3
	0 1	0 0	0 1	0 0	0 1	P4
Free=	0	0	0	0	1	

Allocate resource to P3 →

Ignore P4 as it has no resources →

Alloc(Req)	1 0	0 1	1 0	1 0	0 1	P1
	1 0	1 0	0 1	0 0	0 1	P2
	0 0	0 0	0 0	1 0	1 0	P3
						P4
Free=	0	0	0	0	0	

Neither P1 or P2 can proceed
DEADLOCKED

P2 resources returned →

Alloc(Req)	1 0	0 1	1 0	1 0	0 1	P1
	1 0	1 0	0 1	0 0	0 1	P2
						P3
						P4
Free=	0	0	0	1	1	

Example 2

	R1	R2	R3	
R=	1	2	1	
Alloc(Req)	0 1	1 0	0 0	P1
	1 0	1 0	0 1	P2
	0 0	0 1	1 0	P3
Free=	0	0	0	

No process can proceed
DEADLOCKED

Remove P3 allocated resource

	R1	R2	R3	
Free Before =	0	0	1	
Alloc(Req)	0 1	1 0	0 0	P1
	1 0	1 0	0 1	P2
	0 0	0 1	0 1	P3
Free Before =	0	0	0	
Allocate P2 →	0 1	1 0	0 0	P1
	1 0	1 0	1 0	P2
	0 0	0 1	0 1	P3
Free Before =	1	1	1	
Alloc(Req)	0 1	1 0	0 0	P1
	0 0	0 1	0 1	P3
Free Before =	0	1	1	
Allocate P1 →	1 0	1 0	0 0	P1
	0 0	0 1	0 1	P3
Free Before =	1	2	1	
Alloc(Req)	0 0	0 1	0 1	P3
Free Before =	1	1	0	
Allocate P1 →	0 0	1 0	1 0	P3

Terminates => No Deadlock