ELEC 377 – Operating Systems

Week 5 – Class 2

Today

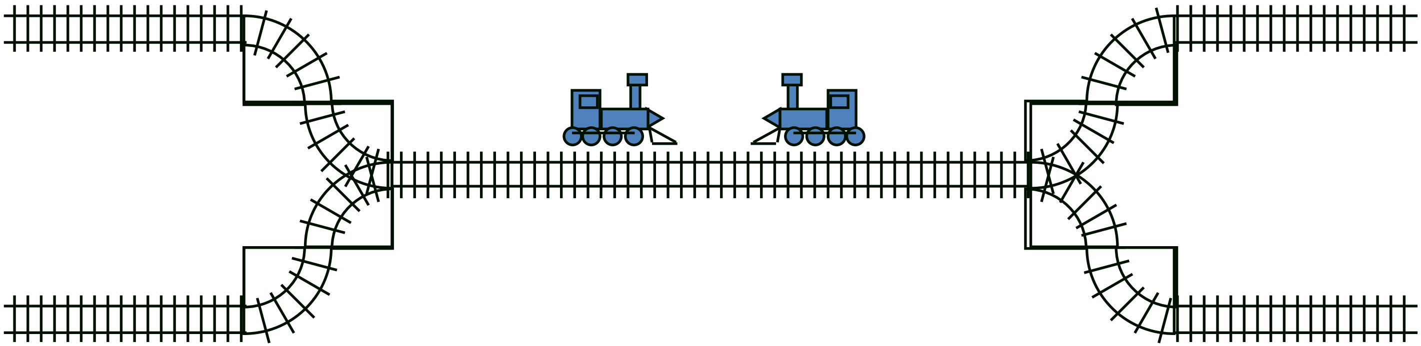
- Deadlock
- ◊ Characterization
- ◊ Prevention
- ◊ Avoidance
- ◊ Recovery

Admin

- Final Exam
  - Friday December 7th – 9AM
- Quiz #1, get from me after class
- Quiz #2, Oct 16

What is Deadlock?

- A set of process, each holding a resource that another process in the set needs



- Common track is a resource
- Starvation
- rollback?

System Model

- Resource Types $R_1, R_2, \ldots, R_n$
  - ◊ Each resource has a number of instances (W_i)
  - ◊ Resource instances are indistinguishable – doesn't matter which one you get.

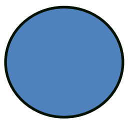
- Process resource protocol
  - ◊ request
  - ◊ use
  - ◊ release

Deadlock Conditions

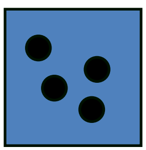
- four conditions necessary for deadlock:
- ◊ **mutual exclusion**: only a limited number (usually one) process at a time can use a resource
- ◊ **hold and wait**: a process has (at least) one resource and is waiting for another
- ◊ **no preemption**: we can't take a resource away from a process
- ◊ **circular wait**: P_0 waits for a resource held by P_1, which waits for a resource held by P_2, … P_n, which waits for a resource held by P_0

Resource Allocation Graph

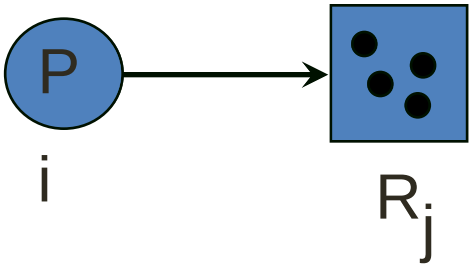
- Process



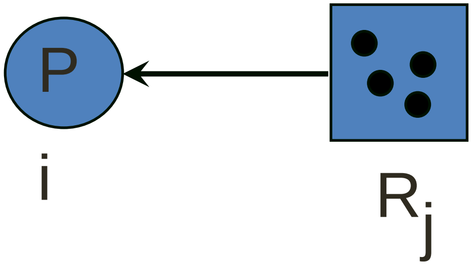
- Resource Type
◊ 4 instances



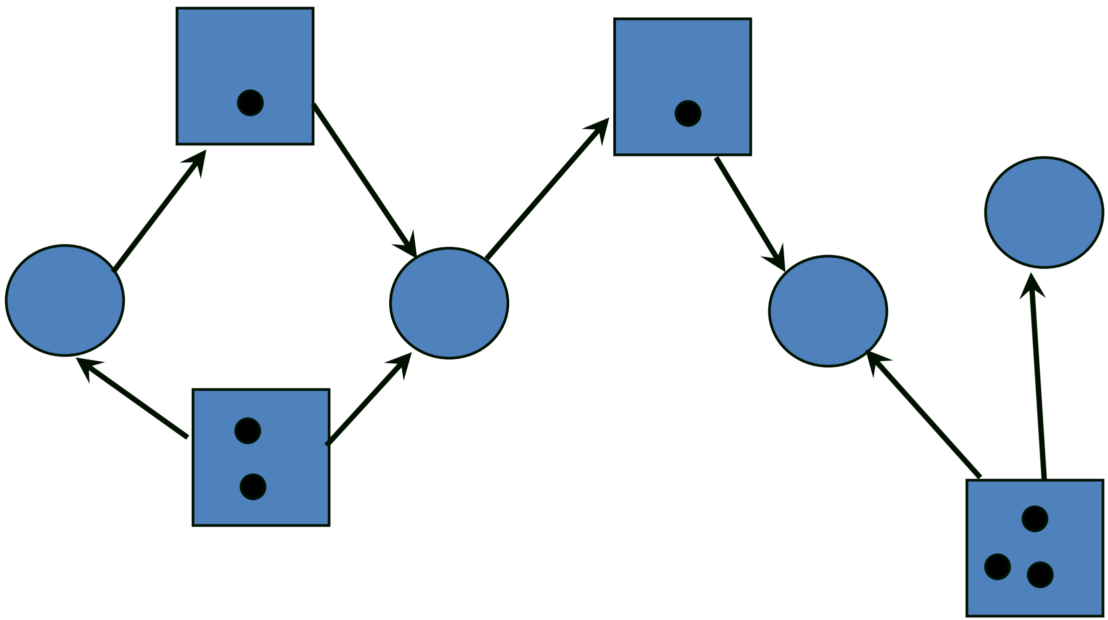
- P_i requests an instance of R_j



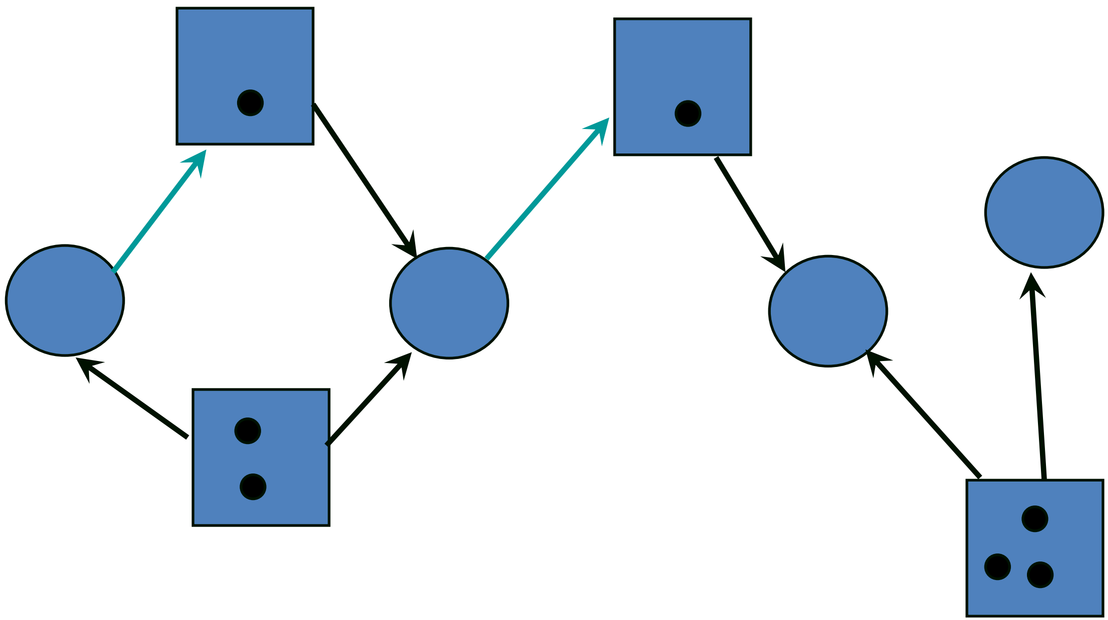
- P_i holds an instance of R_j



Resource Allocation Graph Example

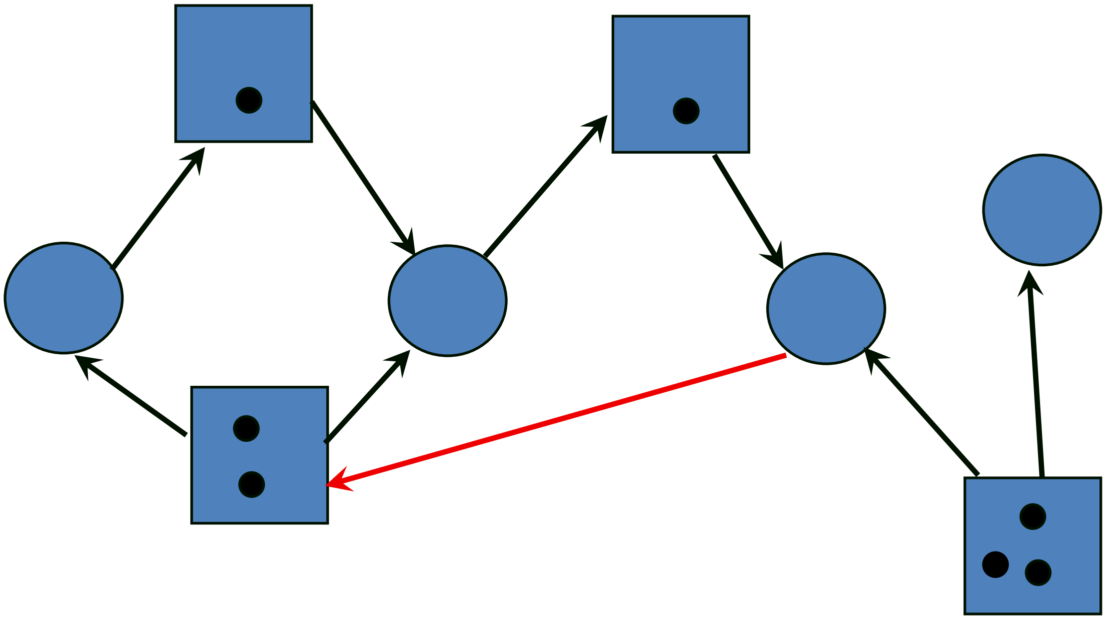


Resource Allocation Graph Example

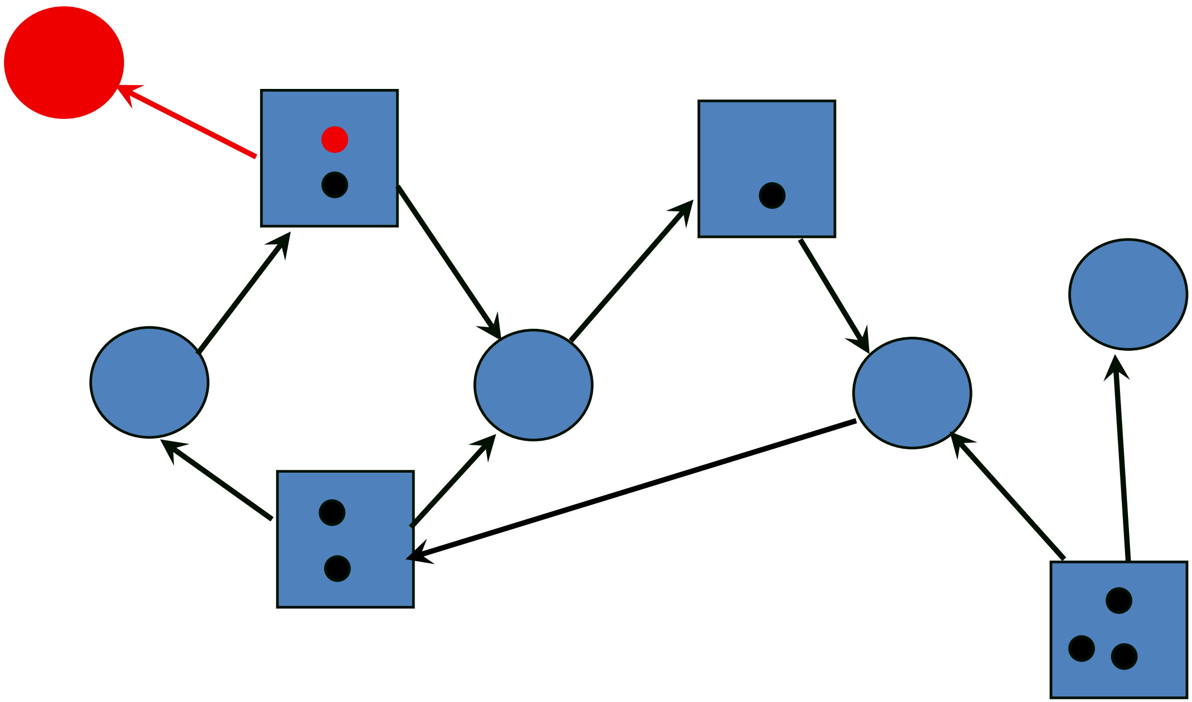


Resource
Requests

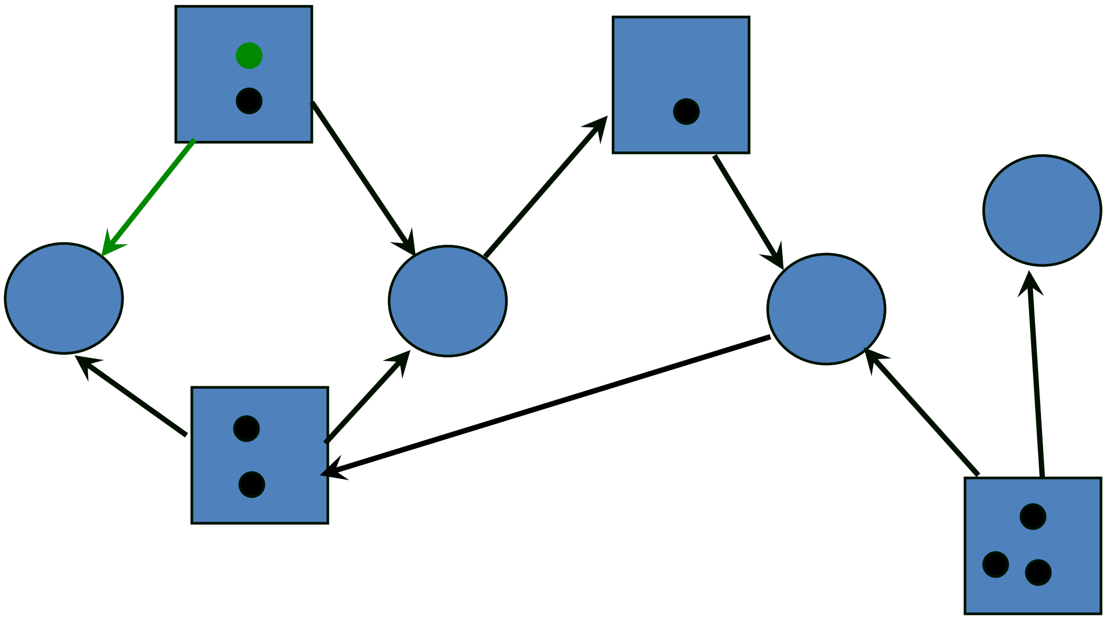
Deadlock Example



No Deadlock



No Deadlock



Deadlock Basics

- No cycle –> no deadlock

- Cycle
- ◊ one instance per resource type –> deadlock
- ◊ more than one instance per resource type?
  - might be a deadlock
  - also might not be a deadlock!!

What do we do??

- Prevention
  - ensure one of the 4 conditions never happens
- Avoidance:
  - Extra information before allocating an available resource
- Recovery:
  - enter deadlock state and recover
- Ignore
  - hope it never happens
  - handle it manually
  - Most interactive operating systems use this approach

Prevention

- Mutual Exclusion
  - Some resources are shareable (some are not)
  - Can add spooler or other device driver in some cases
  – Unfortunately, this is often the least flexible condition

Prevention

- Prevent Hold and Wait

◊ When requesting a resource, cannot already have another resource

◊ If need more than one resource at a time, then must request them all at the same time

◊ After using one or more resources, then must release them before requesting new resources

◊ Efficiency??

  – Resource utilization lower

    - have to hold resources longer

    - over commit resources

    - might need resource, so take resource

  – Starvation??

Prevention

- Relax Preemption

◊ Take away resources when needed

◊ If holding several resources and ask for more that are not available, lose the ones you have

◊ Wait for entire set to become available

◊ Other possibility is to preempt another process that is waiting for the requested resources

◊ rollback??

◊ restart of transaction??

Prevention

- Prevent Circular wait

P1	P2	P1	P2
wait(A)	wait(B)	wait(A)	wait(A)
wait(B)	wait(A)	wait(B)	wait(B)

- Request in same order – no circular requests

- Impose order on all resource requests

◊ Based on typical order for the given system

◊ Optimal for some (most?) processes, suboptimal for others.

Avoidance

- Information up front
◊ Processes declare maximum resources needed

- Dynamically check current resource allocation to make sure cannot be a circular-wait condition

- *Resource allocation state*
◊ Number of available and allocated resources and the *a priori* known maximum resources

Safe State

- System is safe if there is some order we can allocate the resources and not produce a deadlock
  - ◊ might not be the order that the processes actually request the resources
  - ◊ Safe order means that someone may have to wait
- <P1, P2, …, Pn> is safe if Pi can satisfy the maximum resources with available (free) resources and the resources owned by previous processes.
  - ◊ P1 max must be satisfied only with free resources
  - ◊ P2 max must be satisfied with free + P1
  - ◊ P3 max gets available + P1 + P2
  - ◊ If not, wait until a previous process finishes.

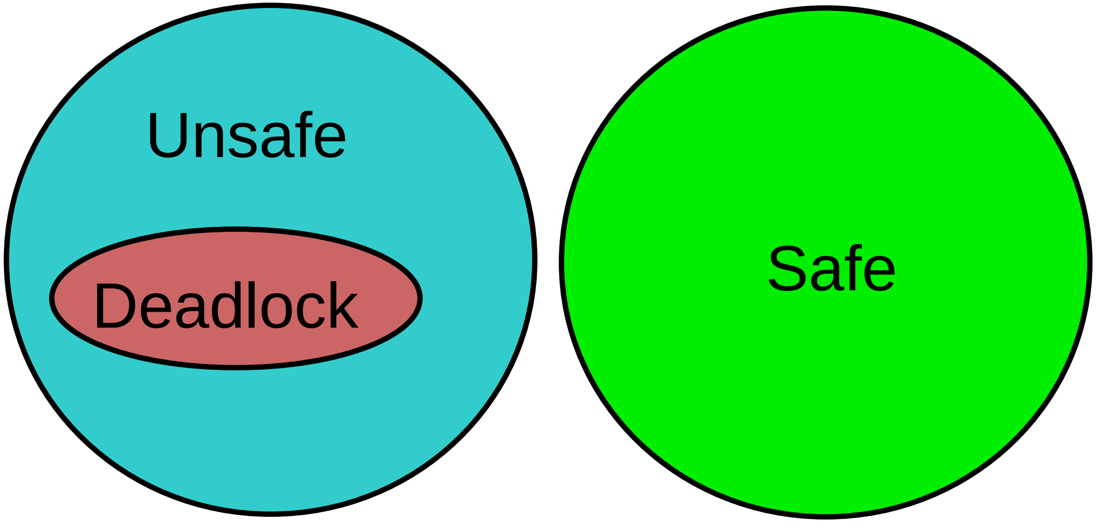
Safe State

- Safe state - no deadlock

- unsafe - possibility of deadlock

- Stay in safe state
◊ easier to calculate than deadlock

Safe State



Deadlock Detection

- Allow system to deadlock

- run a detection algorithm occasionally
  - ◊ Maintain a “waitfor” graph
  - ◊ look for cycles

- Recovery scheme

Flat tire - change the tire

Deadlock Detection

- How often do we run the deadlock detection algorithm?
  - ◊ how often do deadlocks occur?
  - ◊ how many processes do we have to rollback?

- If we wait to long, the graph may have many cycles, and we can't rollback only the process that created the mess

- Algorithm is expensive
  - ◊ if run to often, waste too many cycles

Deadlock Recovery

- Terminate Processes?

◊ all deadlocked processes

◊ one at a time until deadlock is resolved

- Run deadlock algorithm each time
- How do we choose?
    1) process priority
    2) compute time (past and future)
    3) resource usage
    4) resources needed
    5) type of process (interactive, batch)

Deadlock Recovery

- Resource Preemption

◊ take away resources from other processes
- same questions as for termination

◊ process must be rolled back

◊ starvation - is one process always chosen as the victim?

- Different than prevention case

◊ In prevention case, we just said resources are pre-emptible and build into system. Here we use it only as a last resort.

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2	9

Total = 12, Free = 3

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2	9

Total = 12, Free = 3

< P1 , P0 , P2 >

P1 (2) + 3 = 5 >= P1Max(4)

5 + P0(5) = 10 >= P0Max(10)

5 + P2(2) = 7 not >= P2Max(9) (wait for prev. proc. (P0) to finish)

10 + P2(2) = 12 >= P2Max(9)

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2+1	9

Total = 12, Free = 3-1 = 2 ----> No longer safe

< P1 , P0 or P2?, P2 or P0 >

P1 (2) + 2 = 4 >= P1Max(4)

4 + P0(5) = 9 not>= P0Max(10)

4 + P2(3) = 7 not >= P2Max(9)

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2	7

Total = 12, Free = 3

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2	7

Total = 12, Free = 3

P0(5) + 3 = 8 not >= P0Max(10)

P1(2) + 3 = 5 >= P1Max(4)

P2(2) + 3 = 5 not >= P2Max(7)

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2	7

Total = 12, Free = 3

< P1 , P0 , P2 >

P1 (2) + 3 = 5 >= P1Max(4)

5 + P0(5) = 10 >= P0Max(10)

5 + P2(2) = 7 >= P2Max(7)

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2+1	7

Total = 12, Free = 3-1 = 2 ----> safe?

< P1 , P0 , P2 >

P1 (2) + 2 = 4 >= P1Max(4)

4 + P0(5) = 9 not>= P0Max(10)

Safe State – Examples

Process	Current	Max
P0	5	10
P1	2	4
P2	2+1	7

Total = 12, Free = 3-1 = 2 ----> safe!!!!

< P1 , P2 , P0 >

P1 (2) + 2 = 4 >= P1Max(4)

4 + P0(5) = 9 not>= P0Max(10)

4 + P2(3) = 7 >= P2Max(7)

Safe State

- As described, only works on one resource type
- have to define the order for multiple resource types simultaneously.
- Find order of processes so that cascading sum holds in parallel for each resource type
  ◊ Bankers Algorithm

Bankers Algorithm

- Allocation Algorithm

1 Compare request to available

- not available, cannot allocate (sleep)

2 Compare request to max

- violates max request, terminate process

3 Create temporary new state as if resource were allocated

- do not allocate resources, just pretend to

4 Run safety algorithm on new state.

- If not safe, put process to sleep until another process releases resources
- if safe, allocate resources

Bankers Algorithm

	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1
Total Allocated	7 2 5		

Avail	3 3 2
#Res	10 5 7

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	f
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	f
P4	0 0 2	4 3 3	4 3 1	f
Total Allocated	7 2 5			

Avail	3 3 2
#Res	10 5 7
Work	3 3 2

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	f
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	f
P4	0 0 2	4 3 3	4 3 1	f
Total Allocated	7 2 5			

Avail	3 3 2
#Res	10 5 7
Work	3 3 2

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	f
P4	0 0 2	4 3 3	4 3 1	f
Total Allocated	7 2 5			
Avail	3 3 2			
#Res	10 5 7			
Work	5 3 2	<P1		

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	f
P4	0 0 2	4 3 3	4 3 1	f
Total Allocated	7 2 5			
Avail	3 3 2			
#Res	10 5 7			
Work	5 3 2	<P1		

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	f
Total Allocated	7 2 5			

Avail	3 3 2	
#Res	10 5 7	
Work	7 4 3	<P1, P3

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	f
Total Allocated	7 2 5			
Avail	3 3 2			
#Res	10 5 7			
Work	7 4 3	<P1, P3		

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	t
Total Allocated	7 2 5			

Avail	3 3 2	
#Res	10 5 7	
Work	7 4 5	<P1, P3, P4

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	f
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	t
Total Allocated	7 2 5			
Avail	3 3 2			
#Res	10 5 7			
Work	7 4 5	<P1, P3, P4		

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	t
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	t
Total Allocated	7 2 5			

Avail	3 3 2	
#Res	10 5 7	
Work	10 4 7	<P1, P3, P4, P2

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	f
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	t
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	t
Total Allocated	7 2 5			

Avail	3 3 2	
#Res	10 5 7	
Work	10 4 7	<P1, P3, P4, P2

Bankers Algorithm - Safety Algorithm

	Allocation	Max	Need	Finished
P0	0 1 0	7 5 3	7 4 3	t
P1	2 0 0	3 2 2	1 2 2	t
P2	3 0 2	9 0 2	6 0 0	t
P3	2 1 1	2 2 2	0 1 1	t
P4	0 0 2	4 3 3	4 3 1	t
Total Allocated	7 2 5			

Avail	3 3 2	
#Res	10 5 7	
Work	10 5 7	<P1, P3, P4, P2, P0> SAFE

Bankers Algorithm

- Allocation Algorithm

1 Compare request to available
  - not available, cannot allocate (sleep)

2 Compare request to max
  - violates max request, terminate process

3 Create temporary new state as if resource were allocated
  - do not allocate resources, just pretend to

4 Run safety algorithm on new state.
  - If not safe, put process to sleep until another process releases resources
  - if safe, allocate resources