- *Better utilization of computing resources*
- *Automation (convenience and scheduling)*

Chapter 1 — Multi-Programming

- *Performance Requirement*

Chapter 2 — OS and CA for Multi-Programming

- *Sharing Resources between Multiple Processes*
- *Management of Multiple Processes*

Chapter 3 — Process Management for Multi-Programming

- *Sharing CPU between Multiple Processes*
- *Management of Performance Requirements of Processes*

Chapter 4 — CPU Scheduling for Many Processes

- *Shared Data and Variables between Multiple Processes*
- *Race Conditions*

Chapter 5 — Process Synchronization

- *Using Synchronization Tools such as Semaphores*
- *Deadlocks due to Inappropriate Use*

COMPS267F Chapter 6

Deadlock

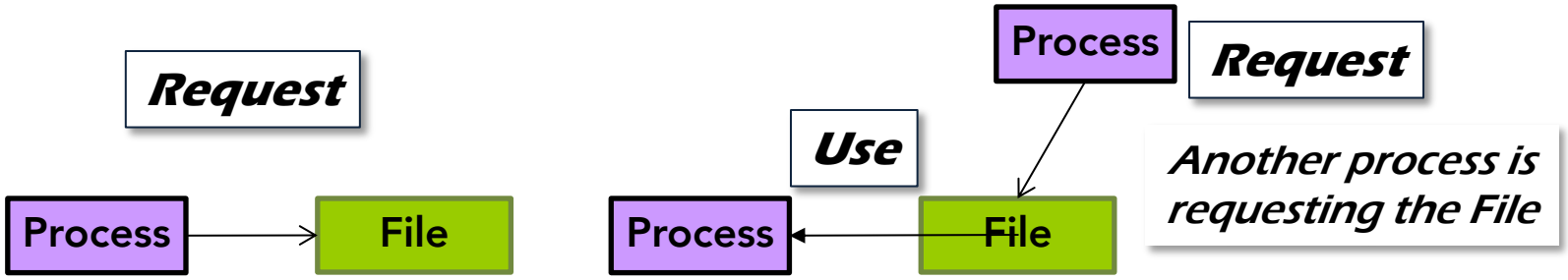
Dr. Andrew Kwok-Fai LUI

Aim of the Chapter

- Discusses the issue of deadlocks in multi-programming systems
- Deadlocks occurs when a set of processes are trapped and unable to make any progress
  - Processes are waiting for resources that never come

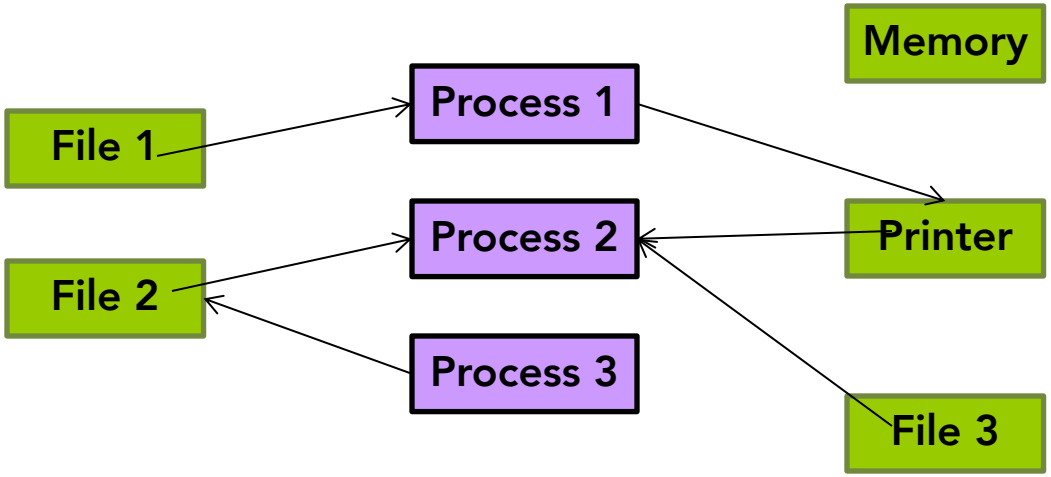
Relation Between Resources and Processes

- A process may be associated with a resource in three operations
  - Request: Before using a resource, a process must request first
    - If the resource is not available, the process must wait
  - Use: The process is hold the resource and using it
  - Release: The process gives up the resource
    - The resource becomes available for other processes



Resource Acquisition and Release

- A process will request, use and release any instance of resource at different time of execution
  - Highly dynamic



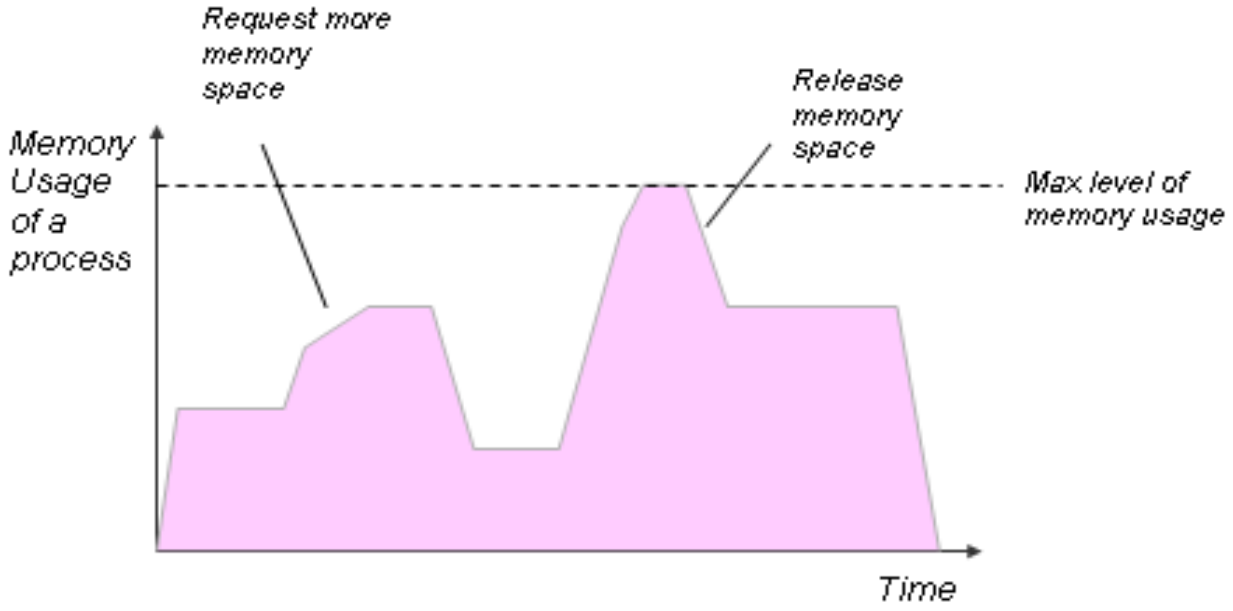
This is one moment in time and will change in the next moment

Relation Between Resources and Processes

- Many resource types are common with programs
  - CPU time, memory, files, printers, tape drives, and hard disks
  - Any of these may be involved in resource competition and deadlocks
- A process may request and hold certain resource instances, and may later release them back to OS

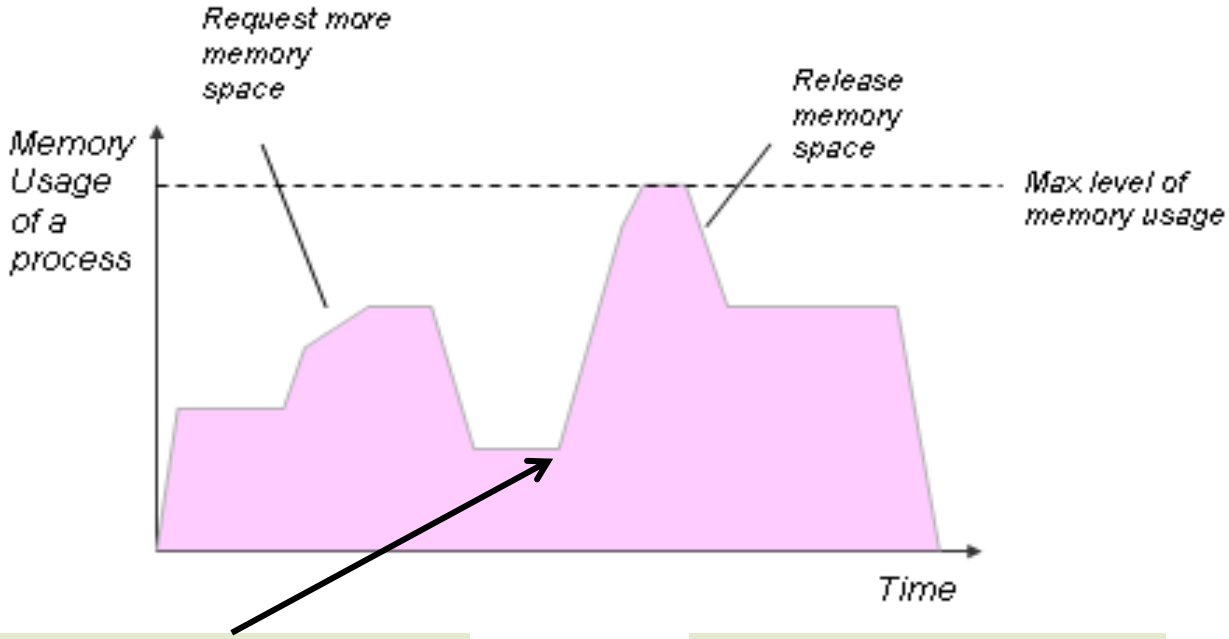
Relation Between Resources and Processes

- A simulated memory usage pattern of a process



Relation Between Resources and Processes

- A simulated memory usage pattern of a process

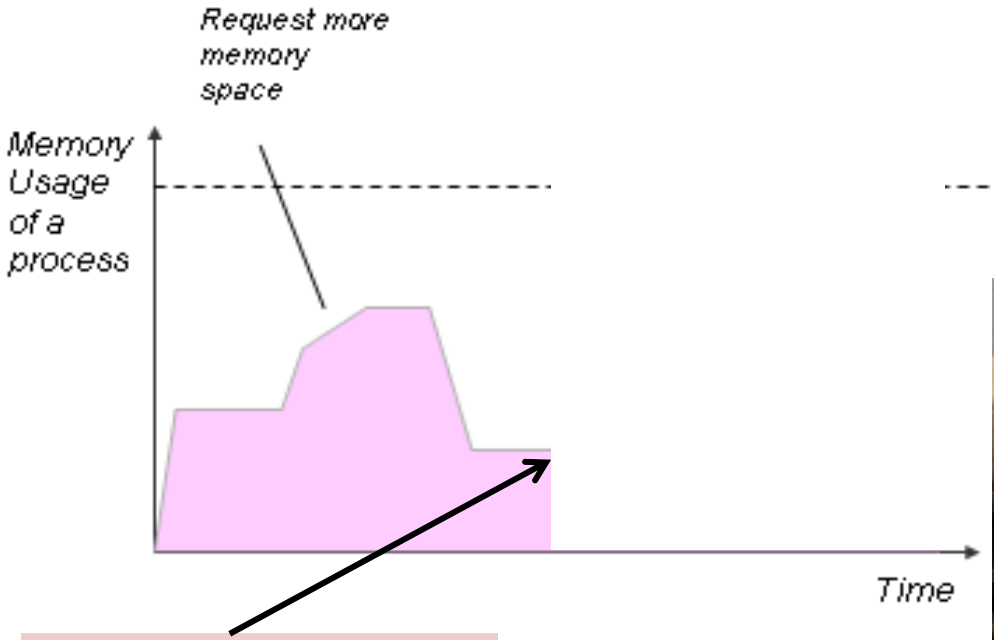


The process may have to wait here if no more memory is available

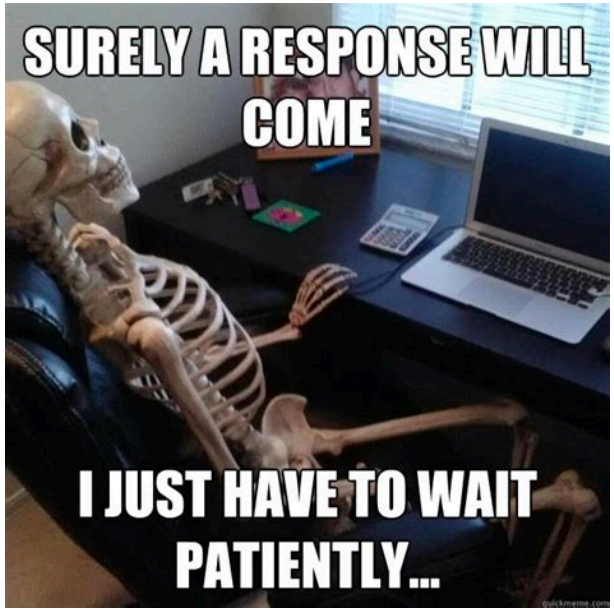
Hopefully free memory will become available in the future

Relation Between Resources and Processes

- A simulated memory usage pattern of a process



It no free memory ever available, the process will wait forever



conditions of deadlock

Example: Deadlock in Multi-Programming

- A program is written to copy data from a tape and sent to a printer
  - One CPU
  - One tape drive
  - One printer

```
00 Request and Lock Tape Drive
01 Request and Lock Printer
02 LOOP
03 Use Tape Drive
04 Use Printer
05 ENDLOOP
06 Release Tape Drive
07 Release Printer
```

Example: Deadlock in Multi-Programming

- Any problem in multi-programming systems?
  - No problem with these two programs

```
00 Request and Lock Tape Drive
01 Request and Lock Printer
02 LOOP
03 Use Tape Drive
04 Use Printer
05 ENDLOOP
06 Release Tape Drive
07 Release Printer
```

```
00 Request and Lock Tape Drive
01 Request and Lock Printer
02 LOOP
03 Use Tape Drive
04 Use Printer
05 ENDLOOP
06 Release Tape Drive
07 Release Printer
```

Example: Deadlock in Multi-Programming

- Is the order of requesting and locking resource acceptable?

```
00 Request and Lock Tape Drive
01 Request and Lock Printer
02 LOOP
03 Use Tape Drive
04 Use Printer
05 ENDLOOP
06 Release Tape Drive
07 Release Printer
```

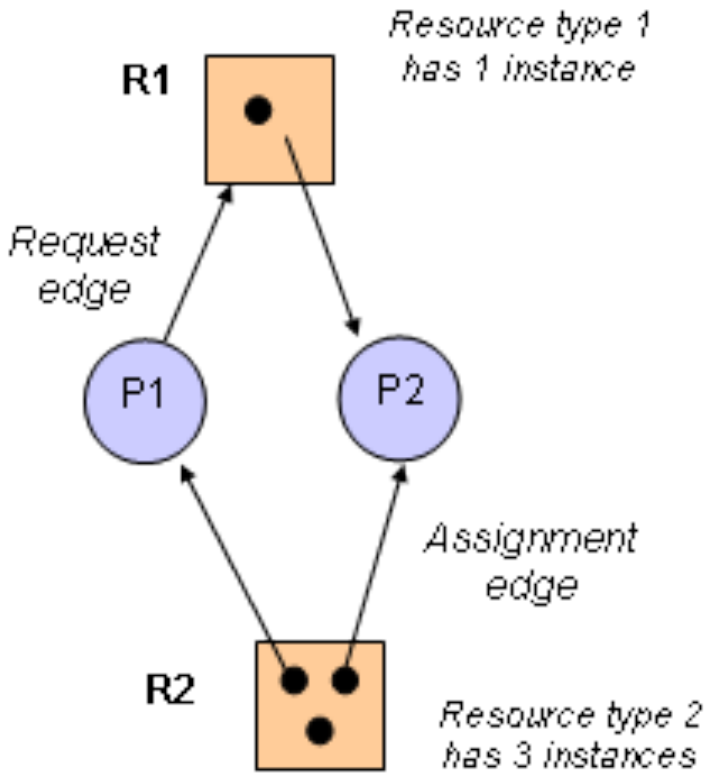
```
00 Request and Lock Printer
01 Request and Lock Tape Drive
02 LOOP
03 Use Tape Drive
04 Use Printer
05 ENDLOOP
06 Release Tape Drive
07 Release Printer
```

Conditions of Deadlock

- The occurrence of deadlock must satisfy four conditions
  - Mutual Exclusion
    - Resource not sharable
  - Hold and wait
    - Allowing a process to hold resources
  - No preemption
    - Process cannot be forced to release resources
  - Circular wait
    - Two processes are requesting a resource instance that is held by another process

resource allocation graphs

Resource Allocation Graph (RAG)



Visual Identification of some deadlock conditions:
-Hold and wait
- Circular wait

Allows visualization of resource request and allocation in relation to processes

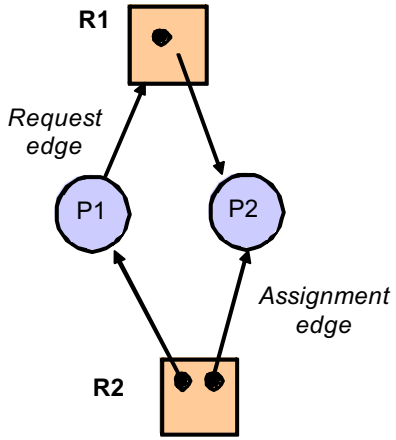
Resource Allocation Graph (RAG)

- Processes are represented by circles
- Resource types are represented by squares
  - Each instance is a dot in the resource square
- Request edge
  - Arrows coming out from a process pointing to a resource type
- Assignment edge
  - Arrows coming out from an instance of resource type

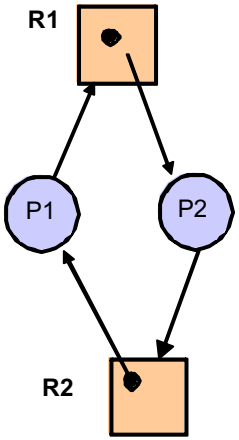
Resource Allocation Graph (RAG)

- No cycles or no hold-and-wait
  - No deadlock
- Contain a cycle between a set of processes
  - Possibility of a deadlock
- If in the cycle a process is requesting for a resource instance with no alternative
  - Deadlock occurs

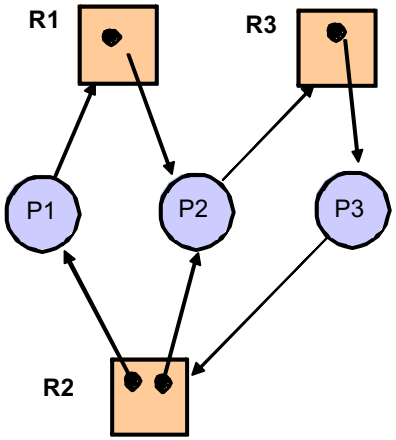
Resource Allocation Graph (RAG)



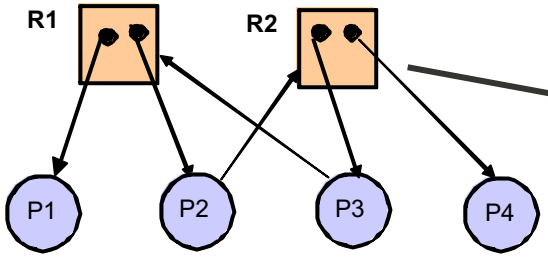
No cycle
No deadlock



A cycle, one instance
Deadlock



A cycle involving R1 and R3, one instance in R1 and R3
Deadlock



A cycle involving R1, R2, and P2, P3
Possibility but no deadlock

We can say that P2 is actually requesting for R2 held by P4 (not P3), and so no actual cycle

Example: A Cake Shop

Example 1: RAG for a Cake Shop Kitchen

A cake shop kitchen has 4 plastic bowls (R1), 3 stirrers (R2), and 5 measuring cups (R3). There are three chefs in the kitchen.

Chef 1 (C1) has got hold of 2 plastic bowls and 1 stirrer.

Chef 2 (C2) has got hold of 1 plastic bowls and 2 measuring cups.

Chef 3 (C3) has got hold of 1 plastic bowl and 1 stirrer and 3 measuring cups.

Chef 1 (C1) is requesting 1 measuring cup.

Draw a resource allocation graph to describe this situation.

Example 1: Resource Allocation Grap

Example 1: RAG for a Cake Shop Kitchen

A cake shop kitchen has 4 plastic bowls (R1), 3 stirrers (R2), and 5 measuring cups (R3). There are three chefs in the kitchen.

Chef 1 (C1) has got hold of 2 plastic bowls and 1 stirrer.

Chef 2 (C2) has got hold of 1 plastic bowls and 2 measuring cups.

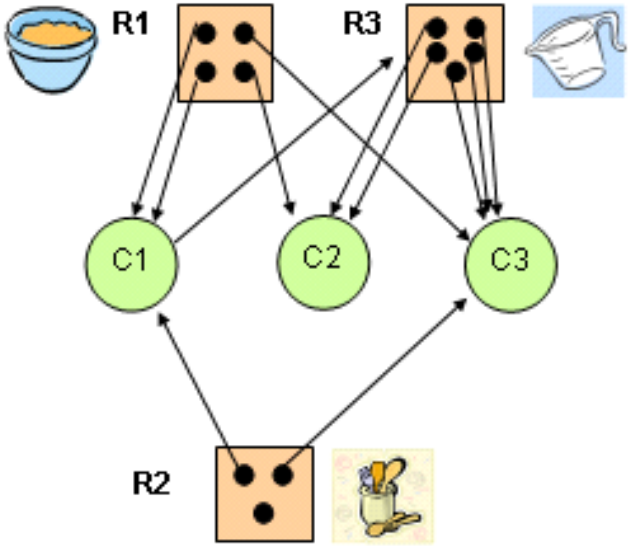
Chef 3 (C3) has got hold of 1 plastic bowl and 1 stirrer and 3 measuring cups.

Chef 1 (C1) is requesting 1 measuring cup.

Draw a resource allocation graph to describe this situation.

Answer:

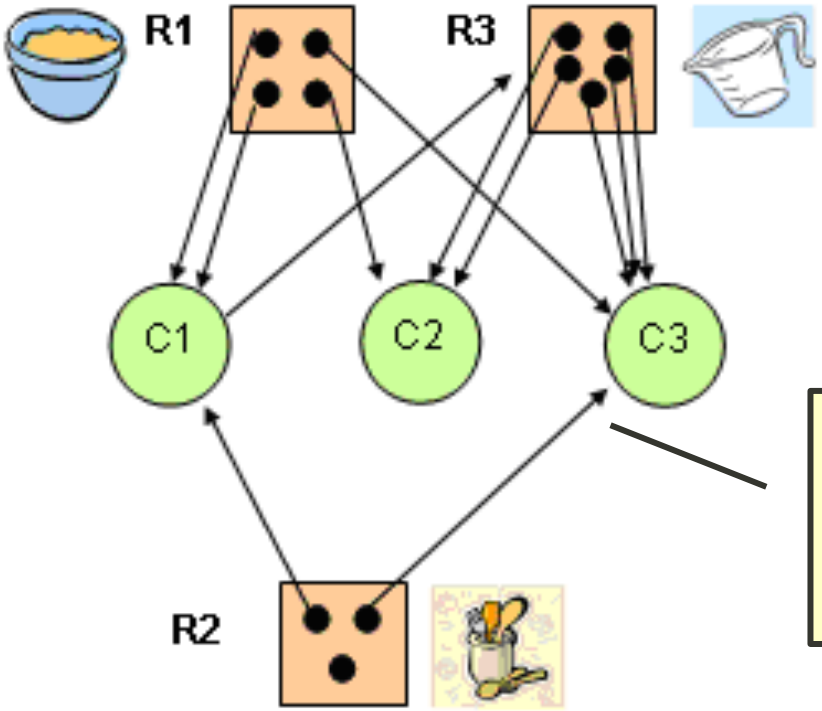
The RAG is shown in the following.



Example 1: Deadlock

Example 2: Cake Shop Kitchen Deadlock

Using the answer in Example 1 to evaluate if a deadlock has occurred in the kitchen



Only C1 is hold-and-wait.
No cycle involving C1
No deadlock.

Example 2: Deadlock

After a change of management, a cake shop kitchen has 3 plastic bowls (R1), 2 stirrers (R2), and 1 measuring cup (R3). There are three chefs in the kitchen.

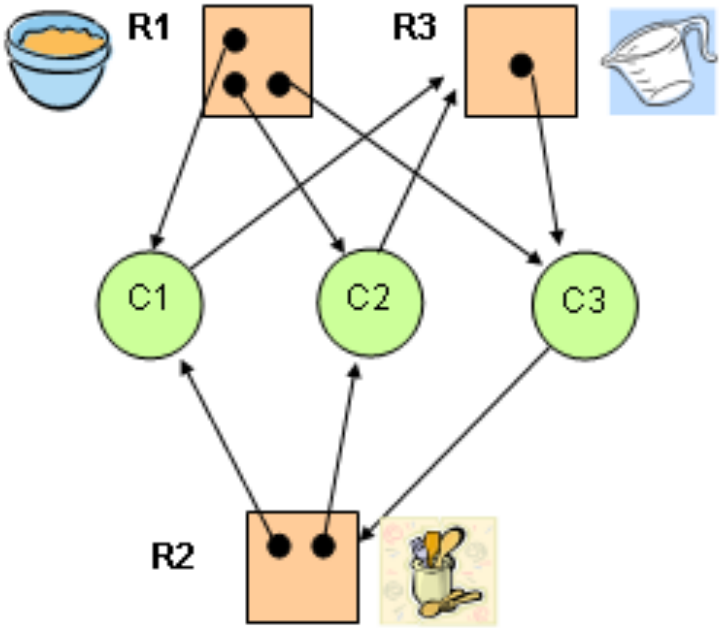
- Chef 1 (C1) has got hold of 1 plastic bowl and 1 stirrer. C1 is requesting 1 measuring cup.
- Chef 2 (C2) has got hold of 1 plastic bowl and 1 stirred. C2 requesting 1 measuring cup.
- Chef 3 (C3) has got hold of 1 plastic bowl and 1 measuring cup. C3 requesting 1 stirrer.

Draw the RAG of the current resource allocation situation. Evaluate if a deadlock occurs.

Please download Errata for Page 92 to 94

Example 2: Deadlock

The RAG is shown below.



- C1, C2 and C3 are hold-and-wait.
- A cycle seems to be found in C1/C2 – R3 – C3 – R2.
- There are no more instances available in R3.
- There is no alternative instance of R2, as one is held by C1 and one is held by C2
- Deadlock occurs between C1, C2, and C3

Example 2A: Deadlock

Let's consider a very similar scenario as Example 2 above.

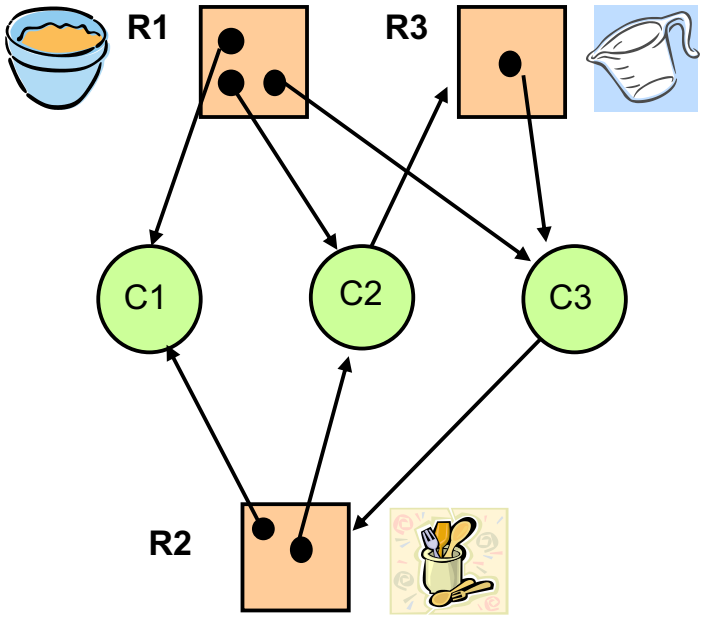
The cake shop kitchen now has 3 plastic bowls (R1), 2 stirrers (R2), and 1 measuring cup (R3). Their allocation situation is

- Chef 1 (C1) has got hold of 1 plastic bowl and 1 stirrer.
- Chef 2 (C2) has got hold of 1 plastic bowl and 1 stirred. C2 requesting 1 measuring cup.
- Chef 3 (C3) has got hold of 1 plastic bowl and 1 measuring cup. C3 requesting 1 stirrer.

Draw the RAG of the current resource allocation situation. Evaluate if a deadlock occurs.

Example 2A: Deadlock

The RAG is shown below.



- C1 is not hold-and-wait (so not considered anymore)
- C2 and C3 are hold-and-wait.
- A cycle seems to be found in C2 – R3 – C3 – R2.
- There are no more instances available in R3.
- But there is an alternative instance of R2.
- Chefs C3 can be waiting for R2 held by C1 rather than C2.
- The cycle is not truly there.
- A deadlock is possible but no deadlock now.

overview of deadlock handling

Handling of Deadlocks

- Deadlock may be handled in three ways
  - Use a procedure to prevent or avoid deadlocks
  - Devise method to detect a deadlock state and recover from it
  - Ignore the deadlock problem

Handling of Deadlocks

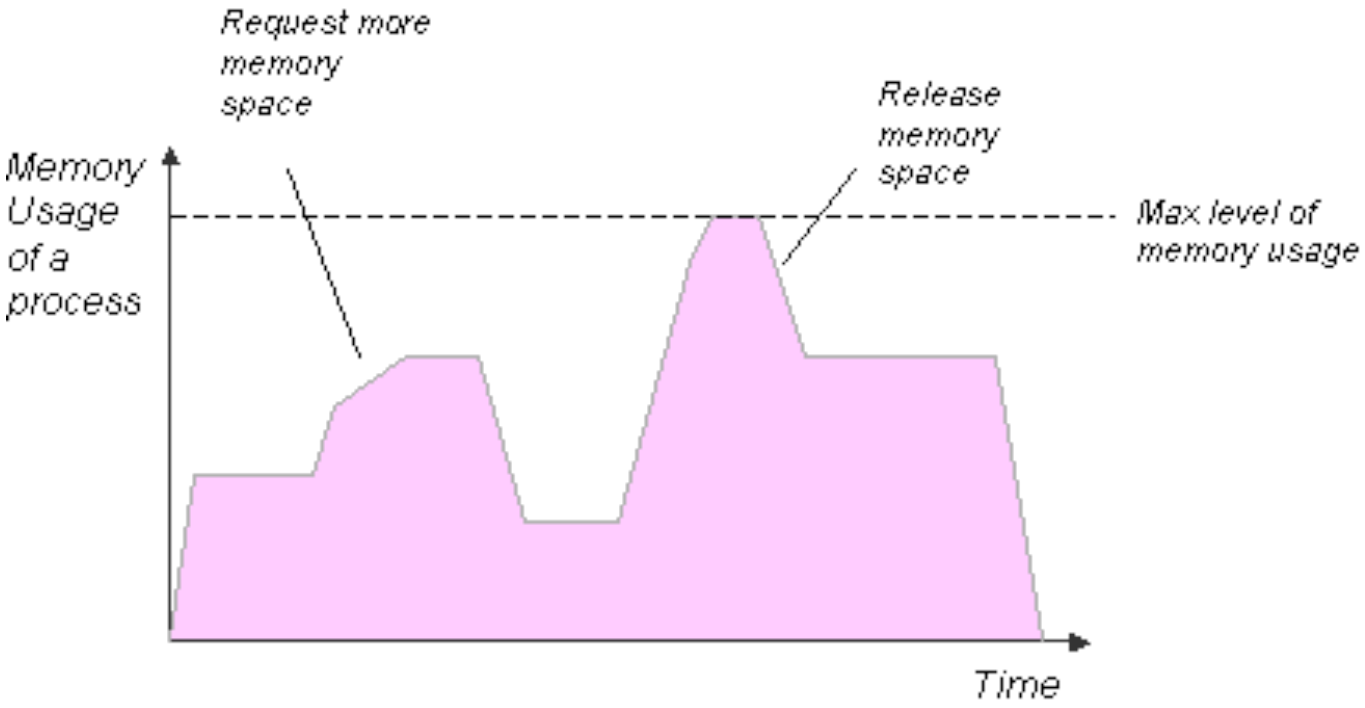
- Deadlock prevention prevents the possibility of a system entering into deadlock
  - Prevent any one of the four necessary conditions of deadlock to happen.
  - Reduce utilization level of system resources
    - Resource allocation tendency is conservative
    - Resource not allocated if a possibility of deadlock exists

Handling of Deadlocks

- Deadlock avoidance is more efficient
  - Make use of additional information about the resource requirements of the processes
  - Determine a sequence of resource allocation so that deadlocks can be avoided

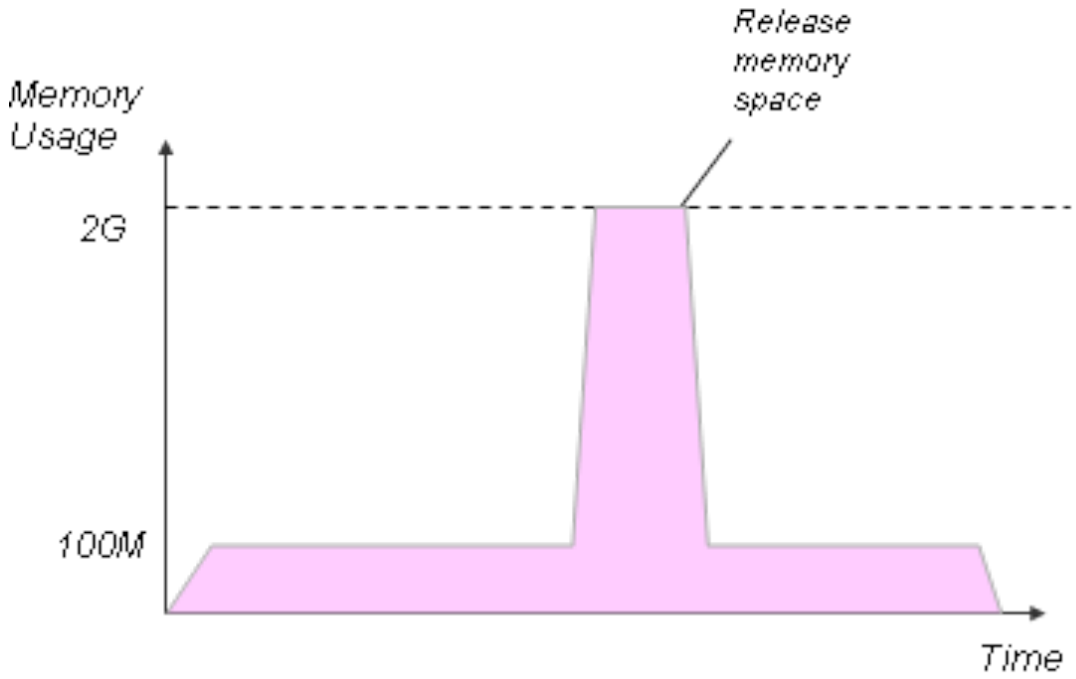
Resource Usage Pattern

- An example of memory usage of a process



Handling of Deadlocks

- Deadlock avoidance tries to maximize resource utilization
  - Most process resource requirement is not always at the maximum



Handling of Deadlocks

- A system can choose not to implement deadlock prevention or deadlock avoidance
  - Allowed to a deadlock situation
  - Deadlock detection and deadlock recovery to return to normal

Handling of Deadlocks

- A system may choose to not implement any of the above
  - Reasonable if deadlock happens infrequently
  - Manually recovered

deadlock prevention

Handling of Deadlocks

- Prevent any of the four deadlock conditions from happening
  - Mutual exclusion
    - Some resources are inherently non-shareable
    - An example is a file that can be modified
  - Cannot do much about it
    - Physical nature

Handling of Deadlocks

- Prevent any of the four deadlock conditions
  - Hold and wait
  - A process does not hold resources while waiting for more
    - Get all needed resource before execute.
    - If failed, release all resources and try again
    - A process can incrementally request resources: release all N resource instances and request N + 1 resource instances
  - Easy to implement
    - Low utilization levels for resources
  - Some processes requiring large amount of resources may starve

Handling of Deadlocks

- Prevent any of the four deadlock conditions
  - No pre-emption
  - Pre-emption is the action to force a process to give up all the currently held resources
  - Difficult if not impossible
    - Sequential processing devices such as tape drives are difficult to pre-empt
    - Pay for preemption

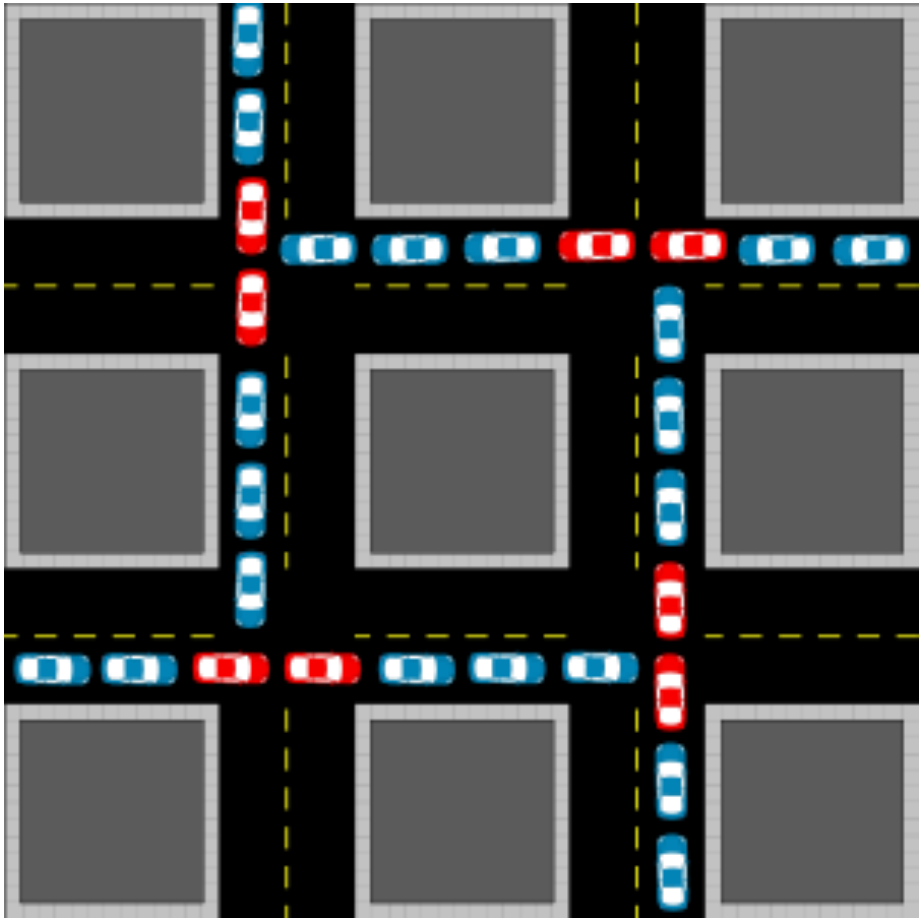
Handling of Deadlocks

- Prevent any of the four deadlock conditions
  - Circular Wait
  - Avoid the circular dependency.
    - Impose a total ordering of the resource request
  - Each resource type available in a system is assigned a unique rank an order. For example, hard disk is 6 and printer is 12.
  - A process requesting resources only in increasing order.
  - If several instances of a particular resource type are needed, a single request for all of them should be made.

Handling of Deadlocks

- Prevent any of the four deadlock conditions
  - Circular Wait
  - Avoid the circular dependency.
    - Impose a total ordering of the resource request

Handling of Deadlocks



Handling of Deadlocks

Handling of Deadlocks

Evaluation of the Solutions

- Deadlock prevention solutions are too restrictive
  - Low utilization of resources
  - Causing the processes to release held resources frequently
  - Adding to system overhead

Example: Deadlock Prevention

Example 4: Cake Shop Kitchen Deadlock Prevention

Consider how to add deadlock prevention measures to prevent deadlock in the kitchen.

Example: Deadlock Prevention

Example 4: Cake Shop Kitchen Deadlock Prevention

Consider how to add deadlock prevention measures to prevent deadlock in the kitchen.

Answer:

Each of the four deadlock conditions is discussed.

Mutual Exclusion. The resource instance like a bowl cannot be shared between two chefs at the same time. Nothing could be done with this condition.

Hold and Wait. Rules could be imposed on the chefs that if they cannot obtain all the needed resource instances at one go, they should release them. They must not hold onto them. A disadvantage of this solution is low utilization of the resources. Now a chef cannot start working with just some of the required resource instance. Previously, for example, the chef could use a measuring cup and a bowl to start mixing up ingredients while waiting for the stirrer.

No Pre-emption. No chef would want to release the resource instance held while waiting for other needed ones. A kitchen manage could be installed to look after the allocation of resource. If there is a potential of a deadlock, the manager could order a chef to release all held resources.

Circular Wait. Rules could be imposed on the order of requesting resource types. Chefs must request in this order: bowl, measuring cups, and then stirrer.

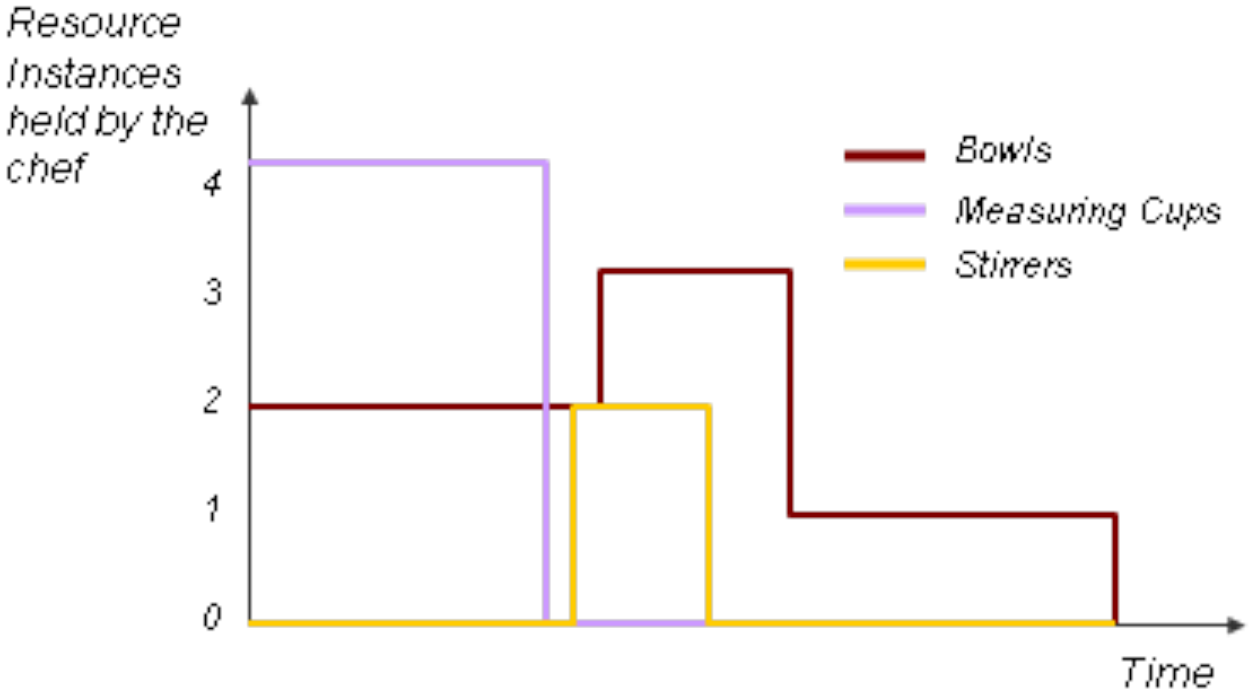
deadlock avoidance

Deadlock Avoidance

- Usage level of resource can change significantly during its lifetime
  - Maximum level: required to complete the execution
  - Current level: process can make progress
    - Most processes should be able to perform work

Example: Deadlock Avoidance

- A chef requires 3 bowls, 2 stirrers, and 4 measuring cups to complete the work of baking a cake.
  - Making progress with only 2 bowls and 4 measuring cups



Deadlock Avoidance

- Deadlock avoidance improves resource utilization
  - Exploiting the time-varying nature of resource requirement
  - Max resource is not needed for most period of time
  - Possible to have a sequence of resource allocation and de-allocation to satisfy resource requirement

Example: Deadlock Avoidance

Example 5: New Management Style for Cake Shop Kitchen

The existing utensils such as bowls and measuring cups have aged. New ones are to be bought. There is a new management installed in the cake shop kitchen in order to reduce the cost of infrastructure. The chefs are asked the maximum instances of bowls, stirrers and measuring cups needed for completing their jobs.

Chef	Max (Bowls, Stirrers, Measuring Cups)
C1	1 2 1
C2	3 1 0
C3	0 1 1

Calculate many bowls, stirrers, and measuring cups should be bought.

Example: Deadlock Avoidance

Example 5: New Management Style for Cake Shop Kitchen

The existing utensils such as bowls and measuring cups have aged. New ones are to be bought. There is a new management installed in the cake shop kitchen in order to reduce the cost of infrastructure. The chefs are asked the maximum instances of bowls, stirrers and measuring cups needed for completing their jobs.

Chef	Max (Bowls, Stirrers, Measuring Cups)
C1	1 2 1
C2	3 1 0
C3	0 1 1

Calculate many bowls, stirrers, and measuring cups should be bought.

Answer:

The expensive solution is simply a sum of the requirements of all chefs.

Bowls = 1 + 3 + 0 = 4

Stirrers = 2 + 1 + 1 = 4

Measuring Cups = 1 + 0 + 1 = 2

Example: Deadlock Avoidance

Example 6: Cost Saving Measure of Cake Shop Kitchen

The new management considers buying 4 bowls, 4 stirrers, and 2 measuring cups cost too much. The management consulted you and check if 3 bowls, 2 stirrers, and 1 cup are sufficient to keep the kitchen running.

Chef	Max (Bowls, Stirrers, Measuring Cups)
C1	1 2 1
C2	3 1 0
C3	0 1 1

Example: Deadlock Avoidance

Example 6: Cost Saving Measure of Cake Shop Kitchen

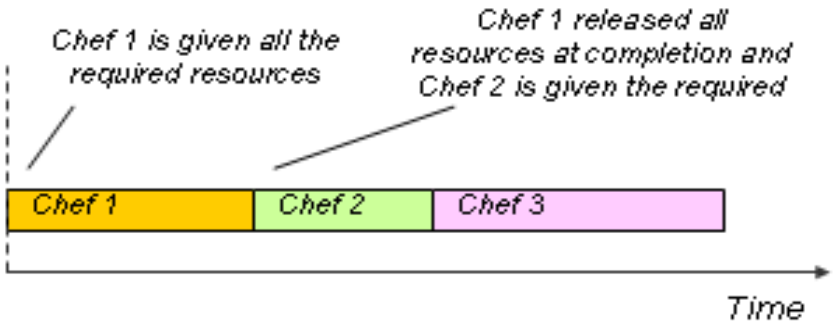
The new management considers buying 4 bowls, 4 stirrers, and 2 measuring cups cost too much. The management consulted you and check if 3 bowls, 2 stirrers, and 1 cup are sufficient to keep the kitchen running.

Chef	Max (Bowls, Stirrers, Measuring Cups)
C1	1 2 1
C2	3 1 0
C3	0 1 1

Answer:

Yes. It can be done if the chefs use the resources in the sequence of C1 – C2 – C3. Chef 1 will use 1 bowl, 2 stirrers, and 1 measuring cup first. Then Chef 2 started working after Chef 1 has completed. The resources originally allocated to Chef 1 were released for Chef 2 to use. The Chef 3 could start working after Chef 2 completed the work.

The drawback of this approach is the chefs cannot work at the same time and reduced the output of the kitchen.



Example: Deadlock Avoidance

Example 7: Cost Saving Yet More Efficient

In Example 6, the solution of having the chefs working sequentially is not efficient. Is there a better method?

Chef	Max (Bowls, Stirrers, Measuring Cups)
C1	1 2 1
C2	3 1 0
C3	0 1 1

Example: Deadlock Avoidance

Example 7: Cost Saving Yet More Efficient

In Example 6, the solution of having the chefs working sequentially is not efficient. Is there a better method?

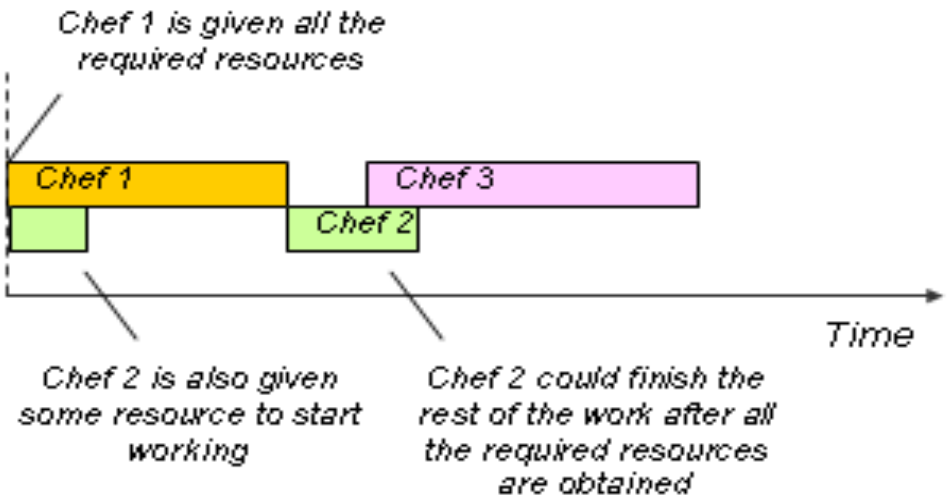
Chef	Max (Bowls, Stirrers, Measuring Cups)
C1	1 2 1
C2	3 1 0
C3	0 1 1

Answer:

Yes. The kitchen has 3 bowls, 2 stirrers, and 1 cup.

At time = 0, C1 is given 1 bowl, 2 stirrer and 1 cup to start working, and C2 is given 2 bowls to start working. C2 is washing strawberries and the bowls can allow C2 to do some of the required work. The work of the three chefs could complete in a shorter time.

Example: Deadlock Avoidance



Safe and Unsafe States

- Deadlock avoidance avoids deadlock by defining states.
  - Safe state: at least a way to satisfy all pending requests by scheduling the processes in a sequence
  - Unsafe state: no sequence found that fulfil the resource requirements of all processes.
    - Not a deadlock state
    - An unsafe state can potentially become a deadlock state
  - Deadlock state

Safe and Unsafe States

The Key: Always Remain in the Safe State

Example: Safe and Unsafe States

Example 8: A Safe State

Evaluate if the current state of resource allocation is in a safe state. There is only one resource type.

Process	Allocation	Max
P1	2	9
P2	2	4
P3	2	7

Available free resource instances = 3

Example: Safe and Unsafe States

Example 8: A Safe State

Answer:

If it is in a safe state, there exists a sequence of resource allocation that can satisfy all pending requests. First we calculate how many resource instances are needed of each process.

Process	Allocation	Max	Need
P1	2	9	7
P2	2	4	2
P3	2	7	5

The following is a sequence that allows all processes to obtain the required resources.

Allocate 2 to P2, P2 has 4, P2 completed execution, P2 releases 4 back to system.

Available resource instances = 1 + 4 = 5

Allocate 5 to P3, P3 has 7, P3 completed execution, P3 releases 7 back to system.

Available resource instances = 0 + 7 = 7

Allocate 7 to P1, P1 has 9, P1 completed execution, P1 releases 9 back to system.

Available resource instances = 0 + 9 = 9

The system is safe.

Banker's Algorithm

- The name of this algorithm is very illustrative
  - Bankers need to do when they allocate their resources
- Example:
  - A bank with $10,000 in deposit. Three customers each would like to borrow $4,000.
  - Possible for the bank to satisfy all three customers
  - When and how much each customer needs
  - If all three draw their approved credit line at the same time then the bank is in trouble

Banker's Algorithm

- Example:
  - The total sum needed by the client is less than $10,000 at the same time the bank will do fine
  - Most clients do repay their loans on time
- Generally the banker's algorithm allocates M types of resources, each with a certain number of instances, to N number of processes

Banker's Algorithm

- The banker's algorithm requires the following data structures
  - Available vector
    - Number of available resource instances of each type
  - Max matrix
    - Maximum demand of each type of resource by each process
  - Allocation matrix
    - Currently allocated instances of resource to each process
  - Need matrix
    - Remaining amount of instances of each type of resources required by each process

Banker's Algorithm

- The banker's algorithm requires the following data structures
  - Available vector
    - Number of available resource instances of each type
  - Max matrix
    - Maximum demand of each type of resource by each process
  - Allocation matrix
    - Currently allocated instances of resource to each process
  - Need matrix
    - Remaining amount of instances of each type of resources required by each process

Banker's Algorithm

- Knowing two out of Max, Allocation, and Need, can work out the remaining one

  Need = Max – Allocation

Banker's Algorithm: Resource Allocation Table

Available vector is (1 0 2 0)

Process	*Allocation*	*Need*	*Max*
P1	3 0 1 1	1 1 0 0	4 1 1 1
P2	0 1 0 0	0 1 1 2	0 2 1 2
P3	1 1 1 0	3 1 0 0	4 2 1 0
P4	1 1 0 1	0 0 1 0	1 1 1 1
P5	0 0 0 0	2 1 1 0	2 1 1 0

Banker's Algorithm

- From the resource allocation table
  - Four types of resources
  - Allocation column: number of instances of each type of resources allocated to a process
  - Need column: resource instances required but not allocated
  - Max column: maximum level of resource instances required by each process so that the execution can complete
  - Available vector: free resource instances currently available

Banker's Algorithm

- Steps in the algorithm:
  - Look for a row in Need whose unmet resources are all smaller than or equal to Available
    - If no such row exists, the system is in unsafe state
  - Allocate the resources to the process chosen above (and modify the Allocation matrix) and assume it can finish
    - Mark the process as done and add all its resources to Available
  - Continue the above steps until all processes are marked finished, in which case the original state is safe

Example: Banker's Algorithm

Example 9: Banker's Algorithm

Evaluate if the above resource allocation status is in a safe state with banker's algorithm.

Answer:

Actions	Available
	1 0 2 0
Allocate 0010 to P4. P4 has 1111. P4 completed and release resources back to the system	1 0 2 0 – 0 0 1 0 + 1 1 1 1 = 2 1 2 1
Allocate 1100 to P1. P1 has 4111. P1 completed and release resource back to the system	2 1 2 1 – 1 1 0 0 + 4 1 1 1 = 5 1 3 2
Allocate 3100 to P3. P3 has 4210. P3 completed and release resource back to the system	5 1 3 2 – 3 1 0 0 + 4 2 1 0 = 6 2 4 2
Allocate 0112 to P2. P2 has 0212. P2 completed and release resource back to the system	6 2 4 2 – 0 1 1 2 + 0 2 1 2 = 6 3 4 2
Allocate 2110 to P5. P5 has 2110. P5 completed and release resource back to the system	6 3 4 2 – 2 1 1 0 + 2 1 1 0 = 6 3 4 2

The system is in a safe state. There exists a sequence of resource allocation that can satisfy the resource requirements of all processes.

Example: Banker's Algorithm

Example 9: Banker's Algorithm

Consider the following resource allocation state again, if P3 asks for the allocation of 1000 (1 instance of Resource #1). Should this request be granted?

Example: Banker's Algorithm

Example 9: Banker's Algorithm

Consider the following resource allocation state again, if P3 asks for the allocation of 1000 (1 instance of Resource #1). Should this request be granted?

Answer:

Check if the system remains a safe state if the request is granted. If the request is granted, the resource allocation state is changed as below. P3 received one instance of Resource #1 and so the Allocation becomes 2110 and the Need becomes 2100.

Available vector is (**0** 0 2 0)

Process	*Allocation*	*Need*	*Max*
P1	3 0 1 1	1 1 0 0	4 1 1 1
P2	0 1 0 0	0 1 1 2	0 2 1 2
P3	**2** 1 1 0	**2** 1 0 0	4 2 1 0
P4	1 1 0 1	0 0 1 0	1 1 1 1
P5	0 0 0 0	2 1 1 0	2 1 1 0

Example: Banker's Algorithm

Example 9: Banker's Algorithm

Available vector is (**0** 0 2 0)

Process	*Allocation*	*Need*	*Max*
P1	3 0 1 1	1 1 0 0	4 1 1 1
P2	0 1 0 0	0 1 1 2	0 2 1 2
P3	**2** 1 1 0	**2** 1 0 0	4 2 1 0
P4	1 1 0 1	0 0 1 0	1 1 1 1
P5	0 0 0 0	2 1 1 0	2 1 1 0

The banker's algorithm is applied below.

Actions	**Available**
	0 0 2 0
Allocate 0010 to P4. P4 has 1111. P4 completed and release resources back to the system	0 0 2 0 – 0 0 1 0 + 1 1 1 1 = 1 1 2 1
Allocate 1100 to P1. P1 has 4111. P1 completed and release resource back to the system	1 1 2 1 – 1 1 0 0 + 4 1 1 1 = 4 1 3 2
Allocate 2100 to P3. P3 has 4210. P3 completed and release resource back to the system	4 1 3 2 – 2 1 0 0 + 4 2 1 0 = 6 2 4 2
Allocate 0112 to P2. P2 has 0212. P2 completed and release resource back to the system	6 2 4 2 – 0 1 1 2 + 0 2 1 2 = 6 3 4 2
Allocate 2110 to P5. P5 has 2110. P5 completed and release resource back to the system	6 3 4 2 – 2 1 1 0 + 2 1 1 0 = 6 3 4 2

It is in a safe state. The request from P3 can be granted.

deadlock detection and recovery

Deadlock Detection

- A system can fall into a deadlock state if no deadlock prevention or avoidance algorithms
  - Need deadlock detection ability
  - A remedy of deadlock recovery

Deadlock Detection

- A deadlock detection algorithm is very similar to a banker's algorithm
  - Available vector: number of available resource instances of each type
  - Allocation matrix: currently allocated instances of each type of resource to each process
  - Request matrix: remaining amount of instances of each type of resources requested by each process

Deadlock Detection

- The following algorithm can determine system is in a deadlock state
  - Look for a row (i.e. Process) in Request whose unmet resources are all smaller than or equal to Available
  - If no such row exists, the system is in deadlock, and all unfinished processes are deadlocked

Deadlock Detection

- Deadlock detection algorithm is costly to run.
  - How often is to execute the algorithm is important

Deadlock Recovery

- Recovery is needed if no deadlock avoidance nor prevention
  - Assume that we can detect deadlocks
  - Process termination
  - Process pre-emption

Deadlock Recovery

- Process Termination
  - Simple and effective.
  - One of the processes in the deadlock cycle is terminated
  - Allocated resource instances are released
  - Repeat until the deadlock is resolved
- Terminating a process is difficult and expensive.
  - Example: aborting a printing job of a long document
  - Choosing the suitable process to be terminated is sometimes a matter of guessing

Deadlock Recovery

- Process Pre-emption
  - Pre-empt a process
  - Typically the system will restart the preempted process from some safe state later
  - Not easy to select a suitable process, stop it, rollback and restart