目录

[Chapter 5 1](#_Toc513045451)

[What program data is shared 1](#_Toc513045452)

[Race Condition 2](#_Toc513045453)

[Critical Section: 2](#_Toc513045454)

[Possible solution 3](#_Toc513045455)

[Several failed example 3](#_Toc513045456)

[Peterson’s Alg 5](#_Toc513045457)

[Hardware Atomic solution 5](#_Toc513045458)

[testAndSet(() 6](#_Toc513045459)

[Swap() 6](#_Toc513045460)

[Hardware with bounded waiting 7](#_Toc513045461)

[High level OS sync method 7](#_Toc513045462)

[Lock 7](#_Toc513045463)

[JAVA 9](#_Toc513045464)

[Semaphore 9](#_Toc513045465)

[Classic problem 10](#_Toc513045466)

[Monitor 12](#_Toc513045467)

[Seme and moni’ problem && message passing 13](#_Toc513045468)

[Chapter 7 14](#_Toc513045469)

[A deadlock situation can arise if the following four conditions **hold simultaneously i**n a system: 14](#_Toc513045470)

[Deadlock prevention 15](#_Toc513045471)

[Safe vs. Unsafe States 16](#_Toc513045472)

[Banker’s alogrithm 17](#_Toc513045473)

[Deadlock Detection 17](#_Toc513045474)

[Recovery from Deadlock 18](#_Toc513045475)

[EXAM1 QUES 19](#_Toc513045476)

[FCFS, SJF, RATE MONOTONIC Algorithms … 24](#_Toc513045477)

# Chapter 5

## What program data is shared

Threads

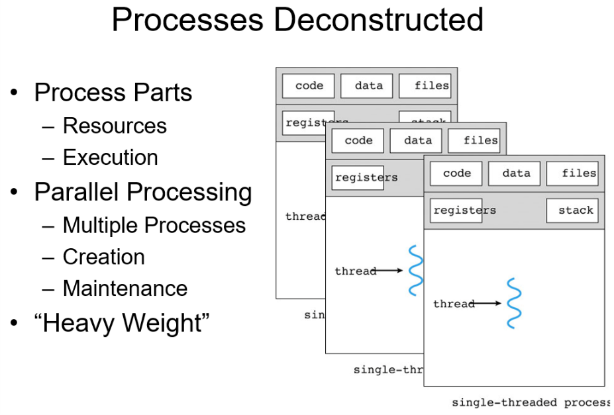
Local variables are not shared (private)

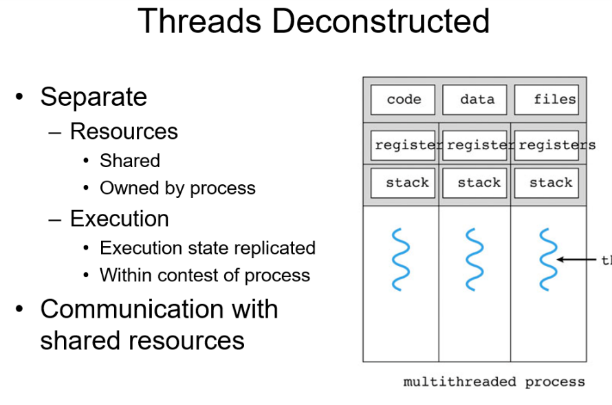
--on its own stack

Global variables and static objects are shared

--Stored in the static data segment, accessible by any threads

Dynamic objects and other heap objects are shared

--Allocated from heap with **malloc/free or** new/delete



## Race Condition

Results depends on order of execution

## Critical Section:

--Part of program where shared resource is accessed

--Deadlock: 2 or more processes waiting on each other to finish before they can finish

--Starvation: One process waits indefinitely for the resources to complete its task, which are repeatedly allocated to other process

Required Properties: 1. Mutual exclusion 2. Progress (thread no in CS cannot block others from entering) 3. Bounded waiting(starvation free)

## Possible solution

Create atomic operations by disabling interrupts!

* Disadvantage

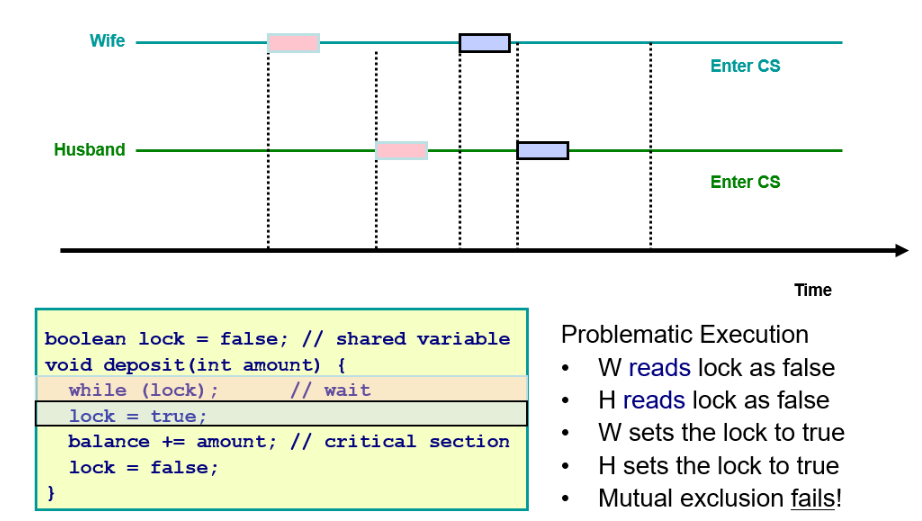
--Unwise to give processes power to turn of interrupts

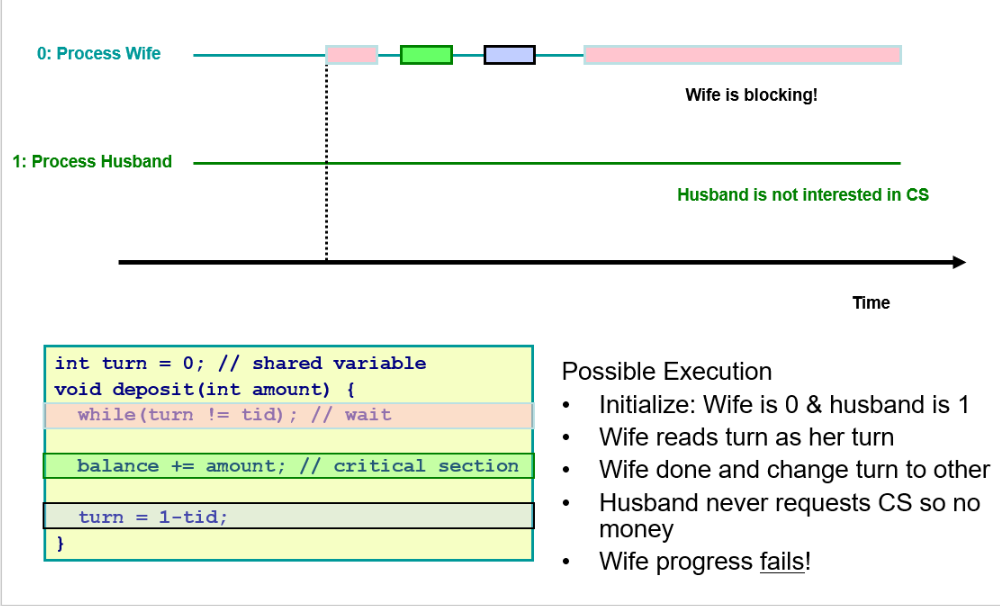
Never turn interrupts on again!

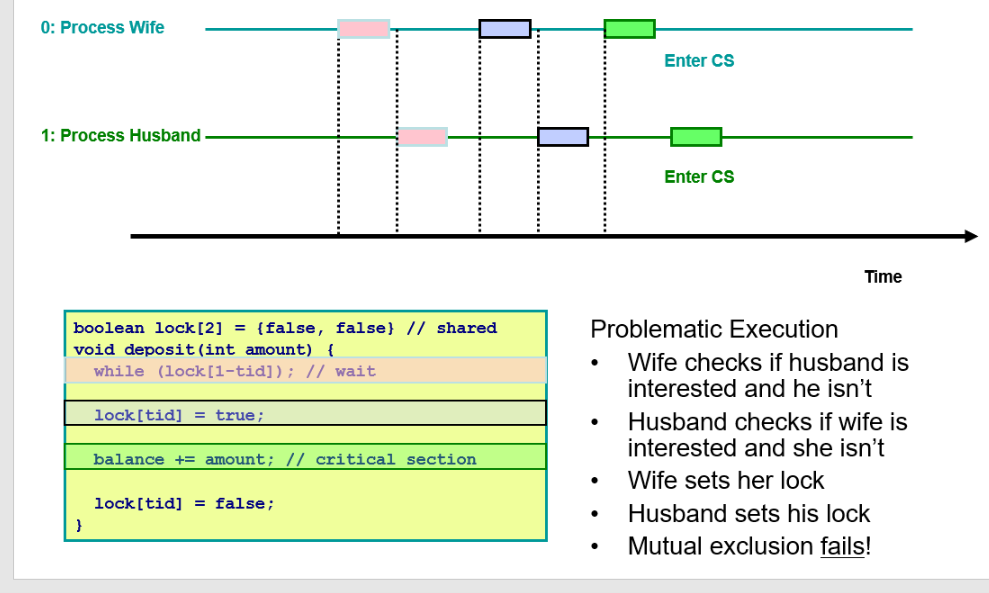
* When to use?

--Good for kernel itself to disable interrupts for a few instructions while it is updating variables or lists

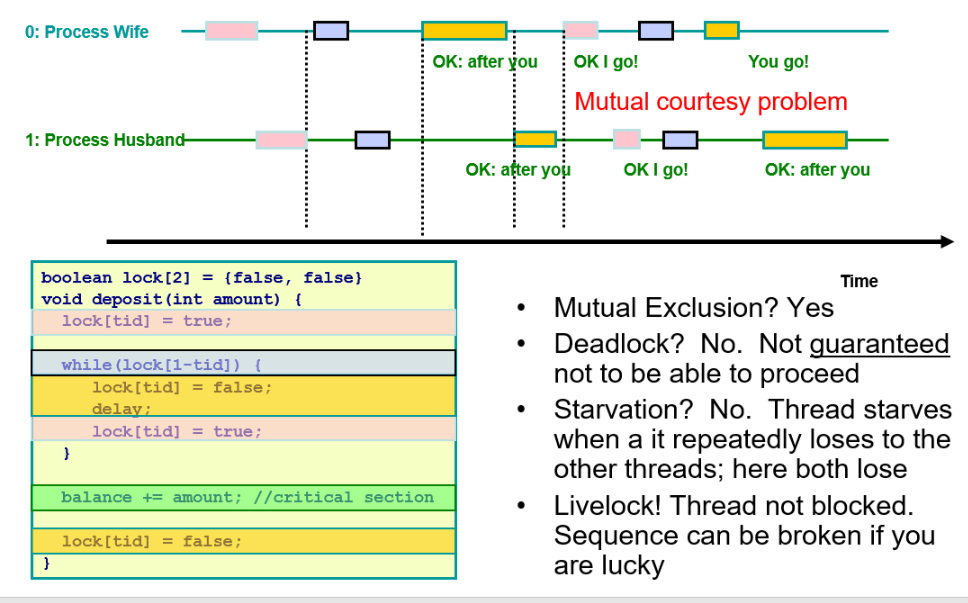
### Several failed example



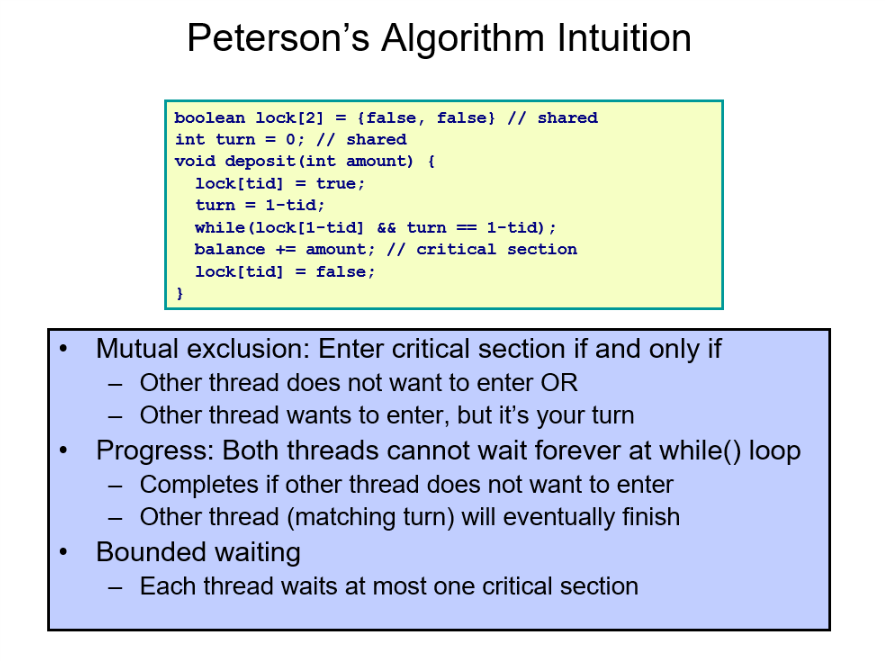








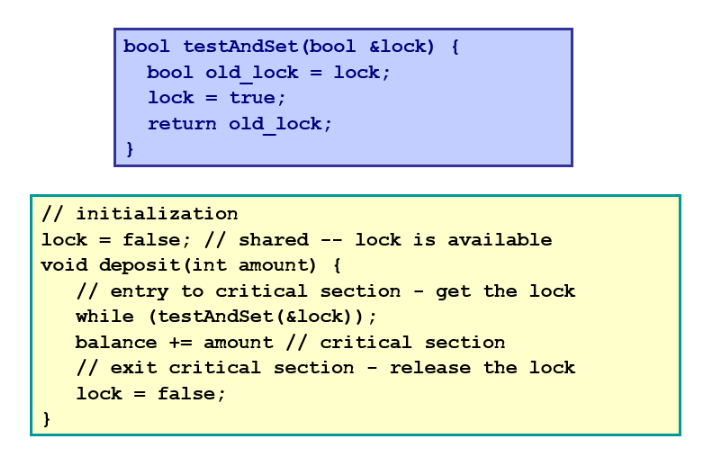
## Peterson’s Alg



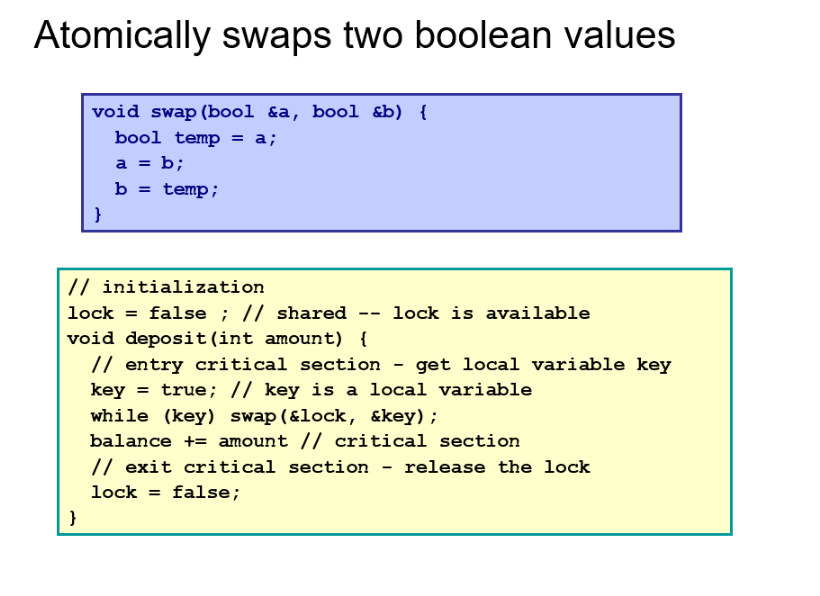
## Hardware Atomic solution

hardware instructions that provide atomic execution of tasks that normally take multiple instructions

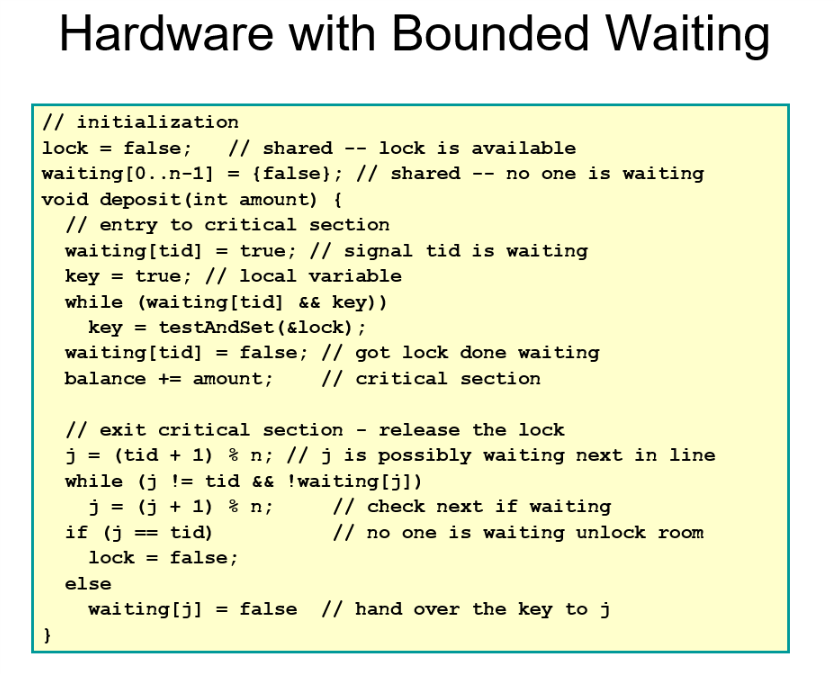
### testAndSet(()



### Swap()



### Hardware with bounded waiting



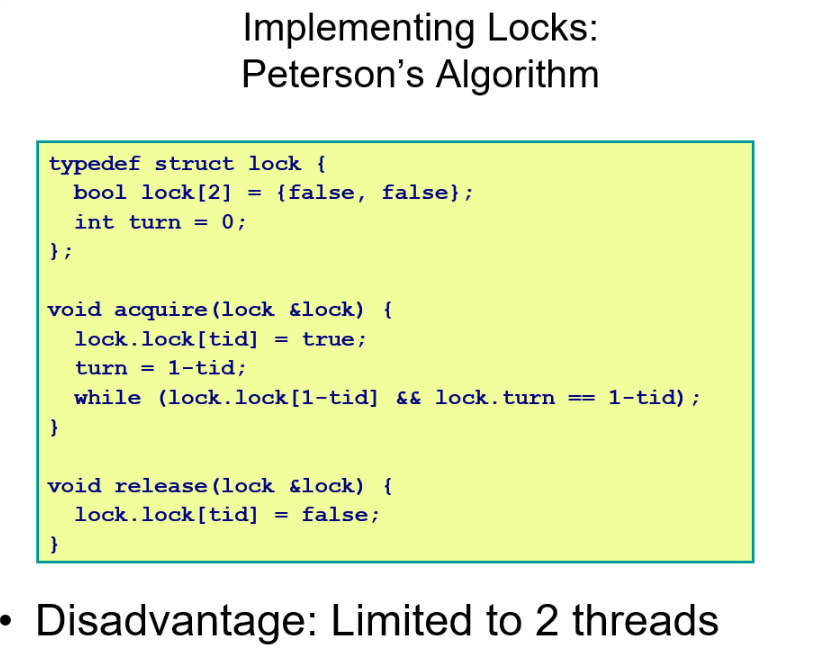


## High level OS sync method

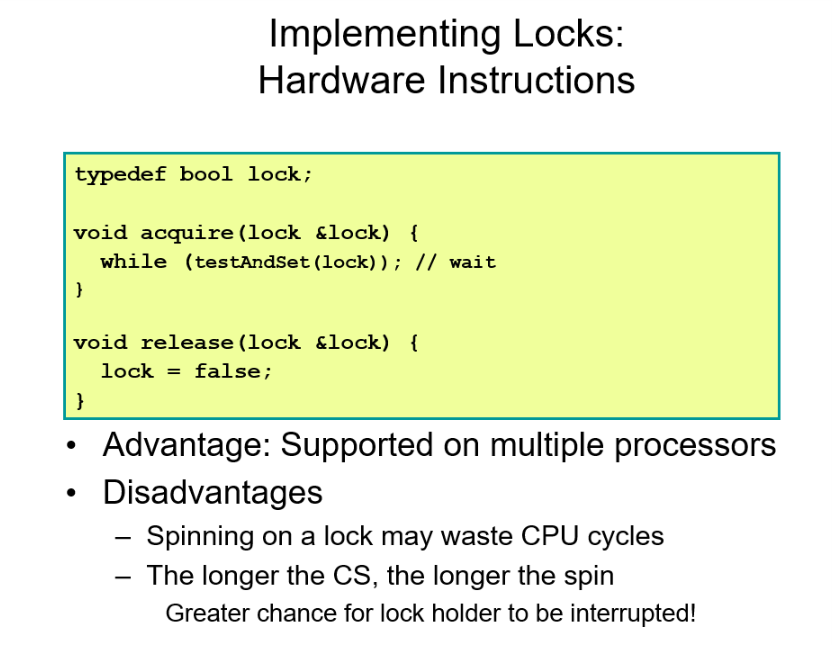
### Lock

Acquire and release





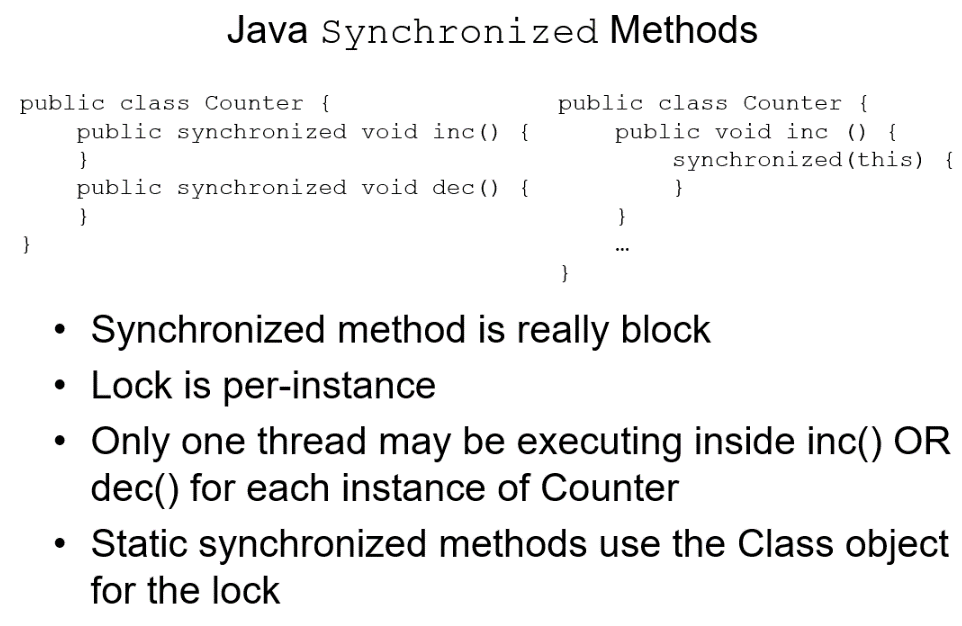






### JAVA

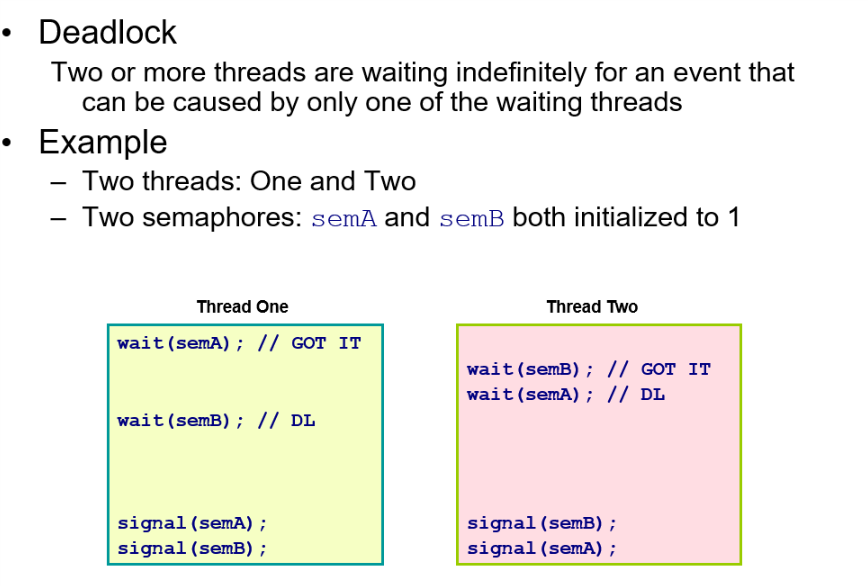
#### Synchronized



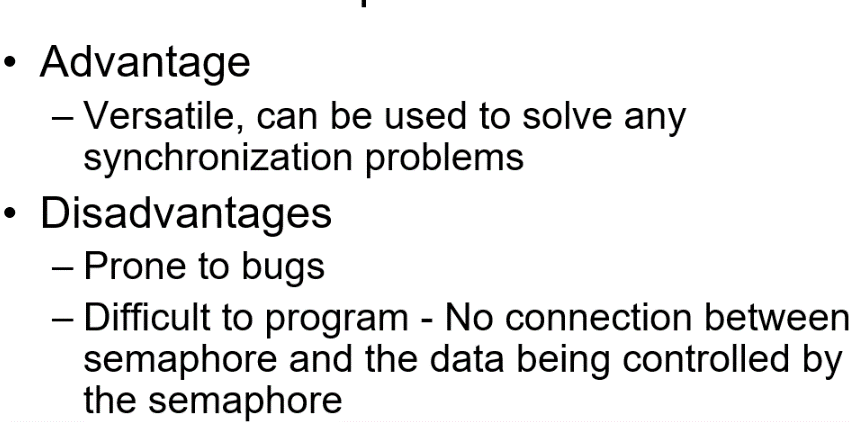


#### Volatile???

### Semaphore

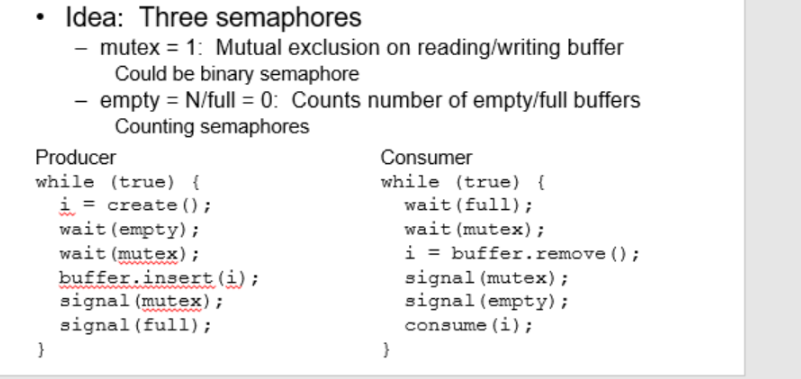


#### Pro and con

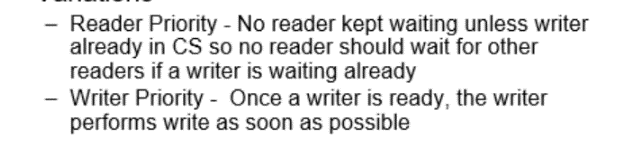


### Classic problem

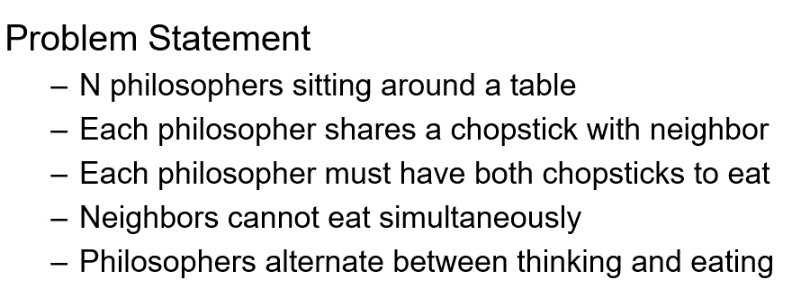
#### Producer/Consumer

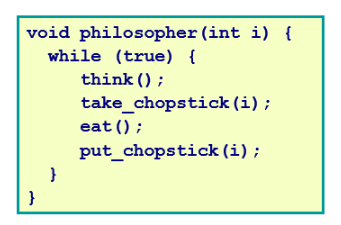


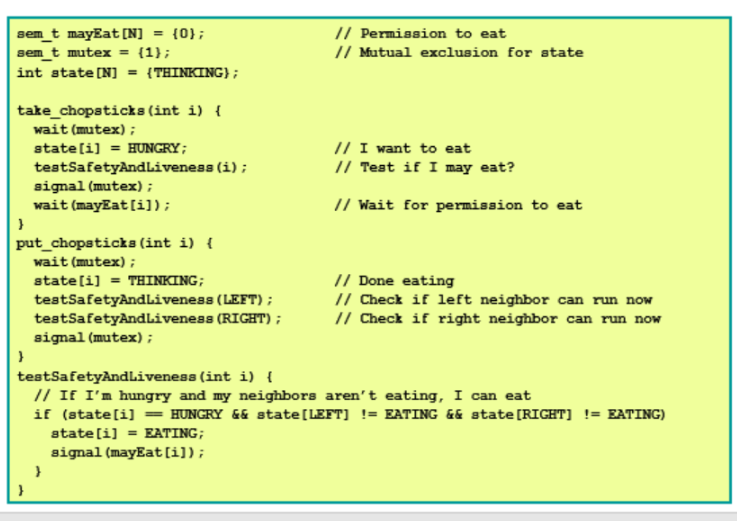
#### Reader writer



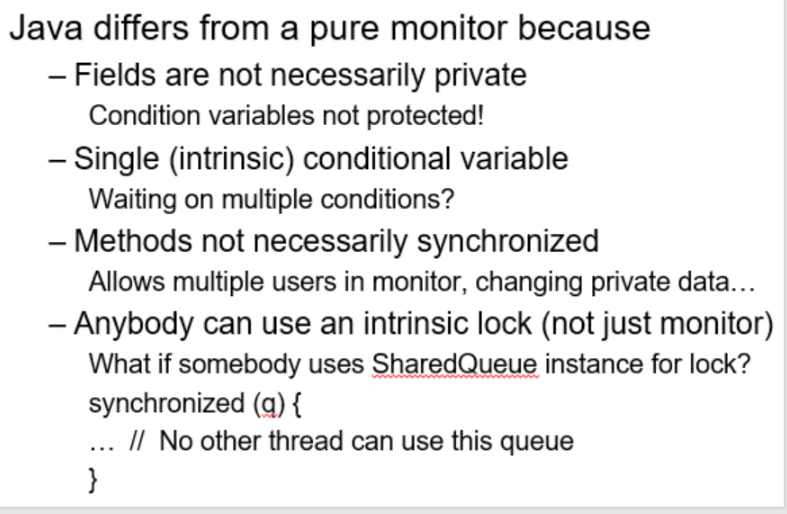
#### Dinning :

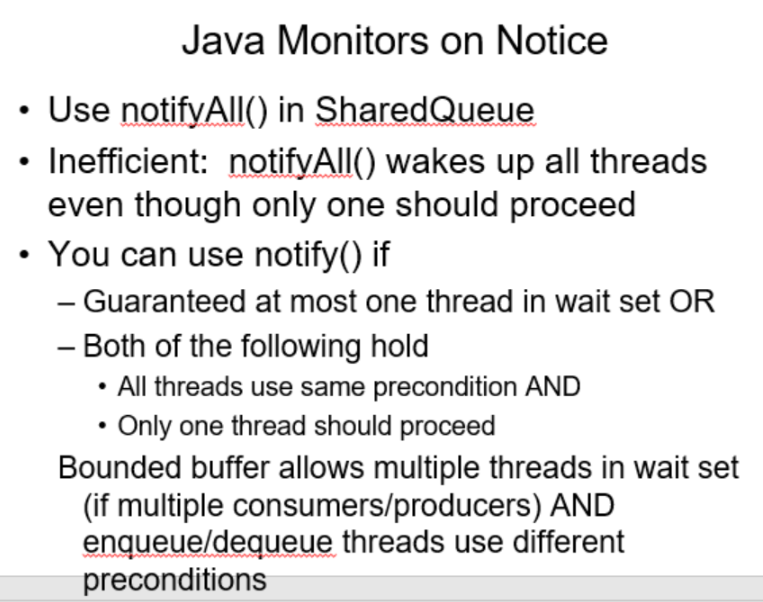


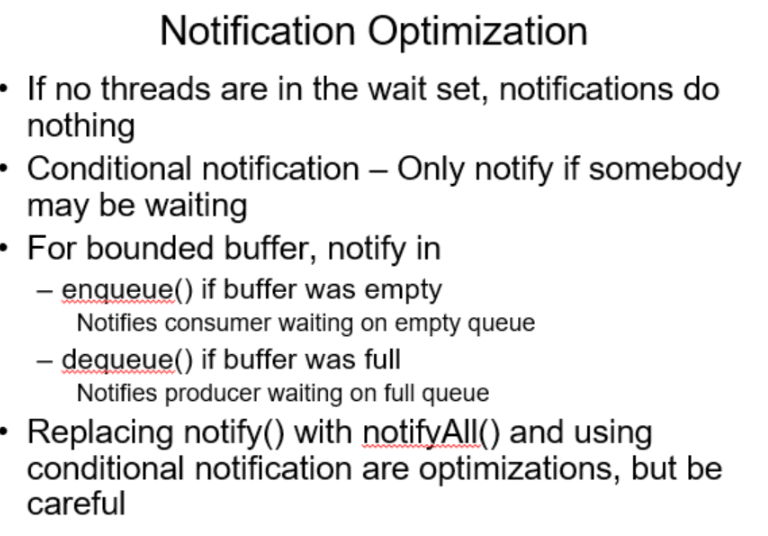




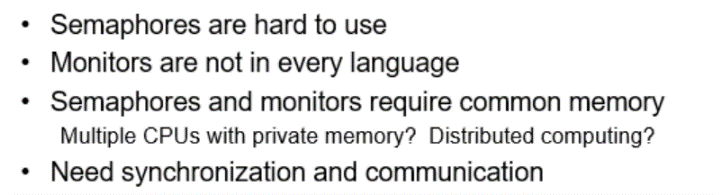
### Monitor

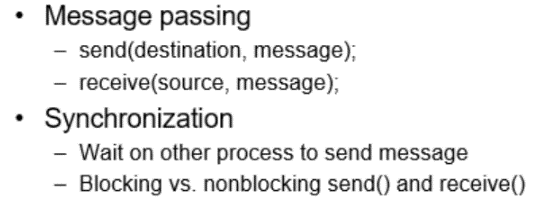






### Seme and moni’ problem && message passing





# Chapter 7

## A deadlock situation can arise if the following four conditions **hold simultaneously i**n a system:

1. Mutual exclusion – Only one process may use a resource at a time
2. Hold and wait – A process may hold allocated resources while awaiting assignment of others
3. No preemption – Previously granted resources may not be taken away
4. Circular wait – A chain of processes exists, such that each process holds at least one resource needed by the next process in the chain



## Deadlock prevention

* + Avoid at least one of 4 necessary conditions for deadlock

Mutual exclusion, Hold & wait, No preemption, Circular wait

Deadlock Prevention

--eliminate Mutex

Make resource sharable, ex read-only files

Problem: some resources are nonsharable

-Eliminate Hold and Wait: Starvation??

--No Preemption:

* 1. Preempt requestor resource
  2. Preempt holder resource

Problem: Can’t take away a lock without causing problems

--Eliminate the possibility of a circular wait:

1.only one resource at a time

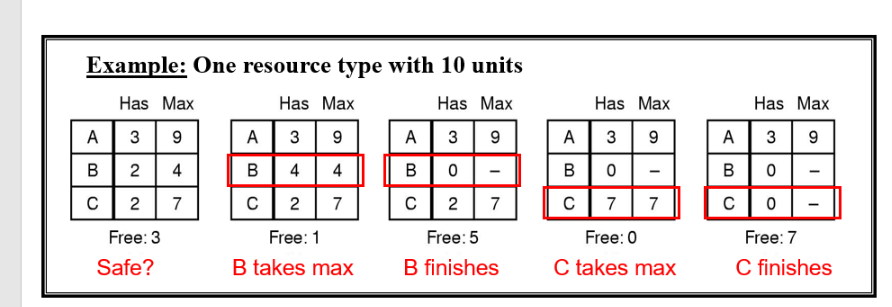
Very limited

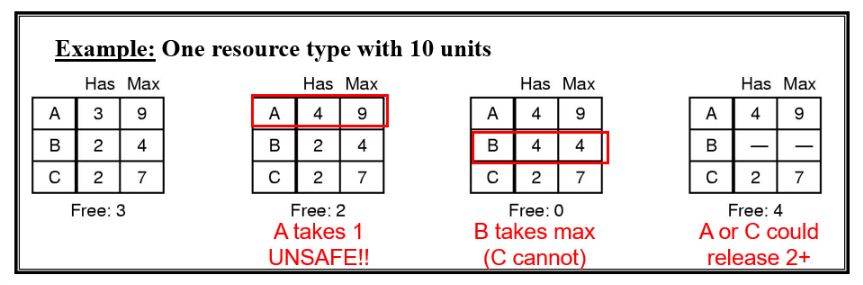
2.impose global ordering for acquiring resources

Ex. Printer = 1, disk = 2, tape = 3 Processes must acquire resource in rank order

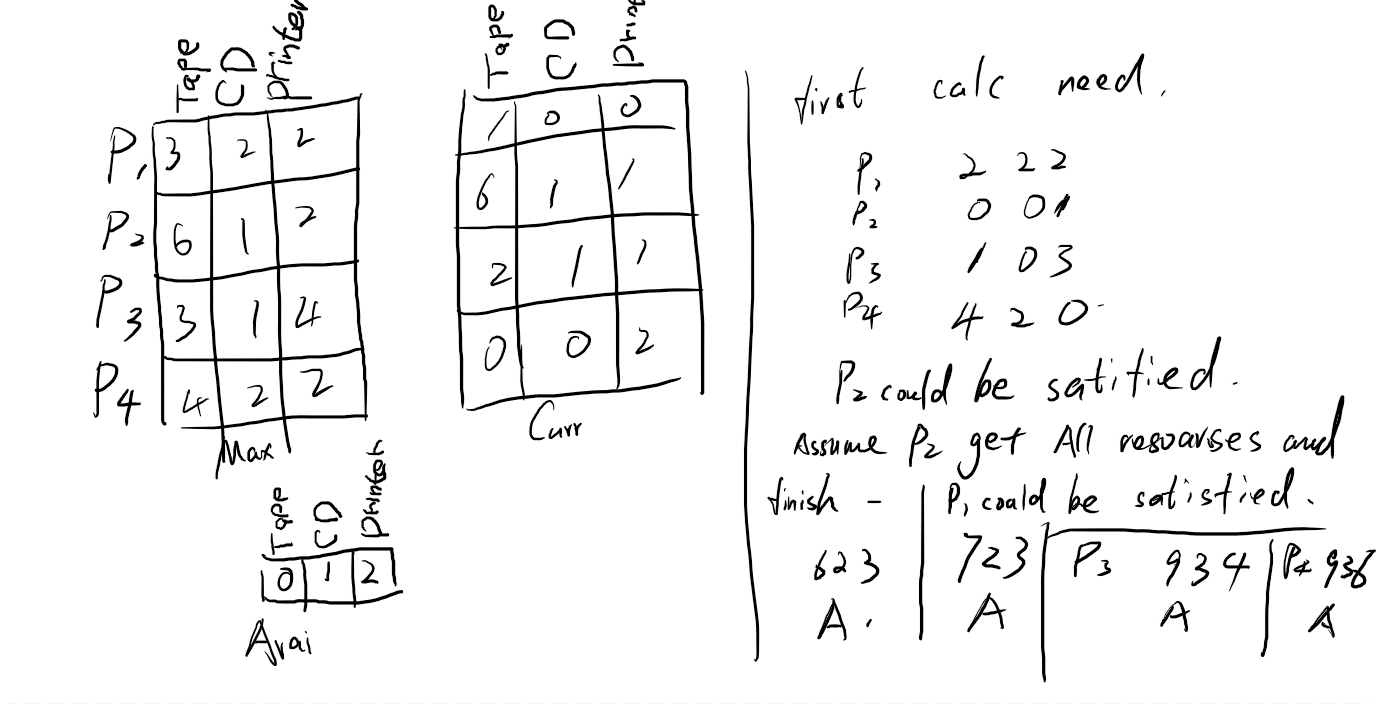
Problem: Inefficient (e.g., Need to read from tape to disk in order to print but must get printer first even if tape copy is long). No order satisfies everybody

## Safe vs. Unsafe States

A safe state avoids deadlock ( banker’s could find a way out, like in the first part)

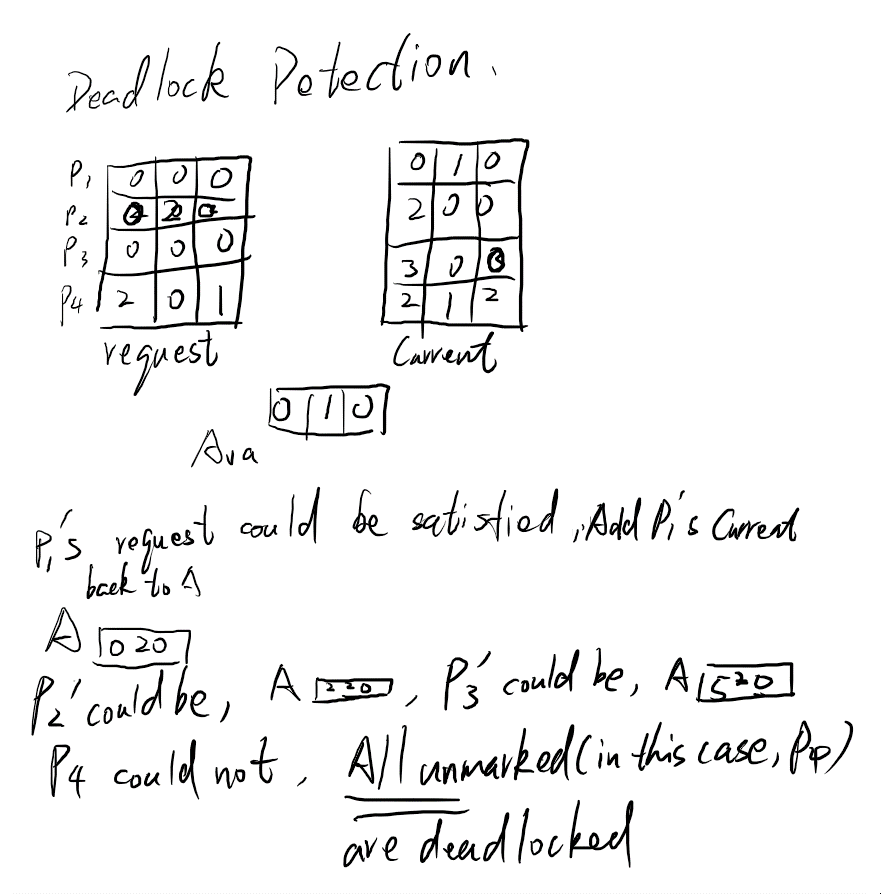
An unsafe is not necessarily deadlocked but may lead to a deadlocked state (banker’s couldn’t find a way out, like in the second part)

## Banker’s alogrithm



## Deadlock Detection

An algorithm to detect a cycle in a graph requires O(n2) operations???, where n is the number of vertices in the graph

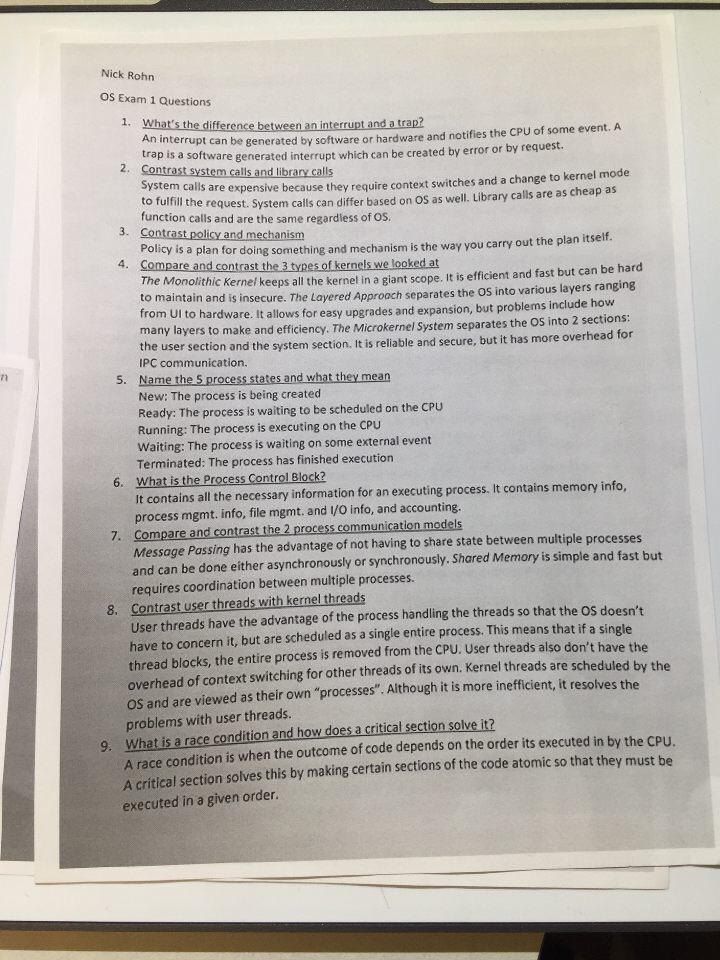
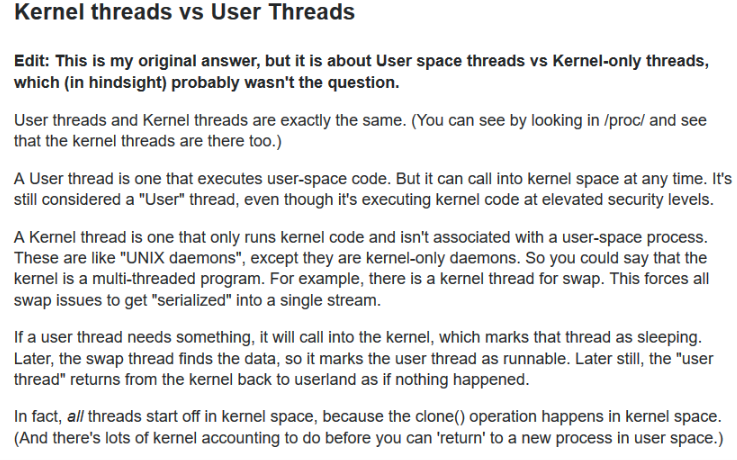
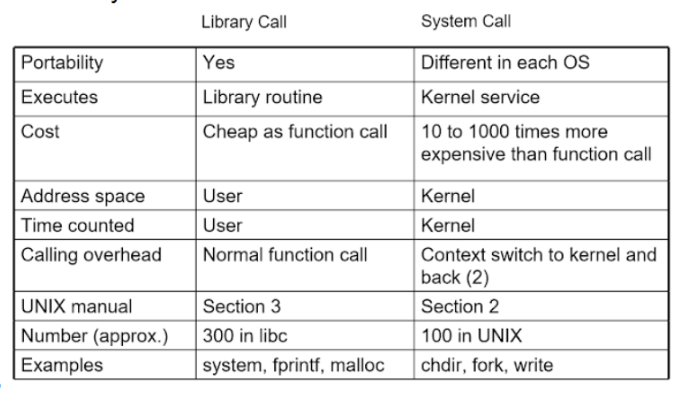
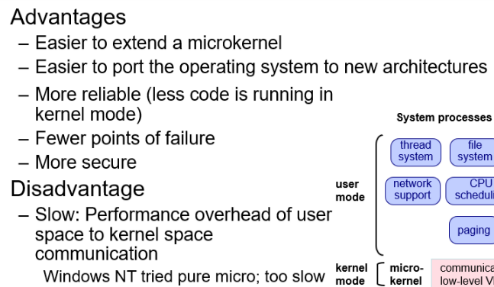


## Recovery from Deadlock

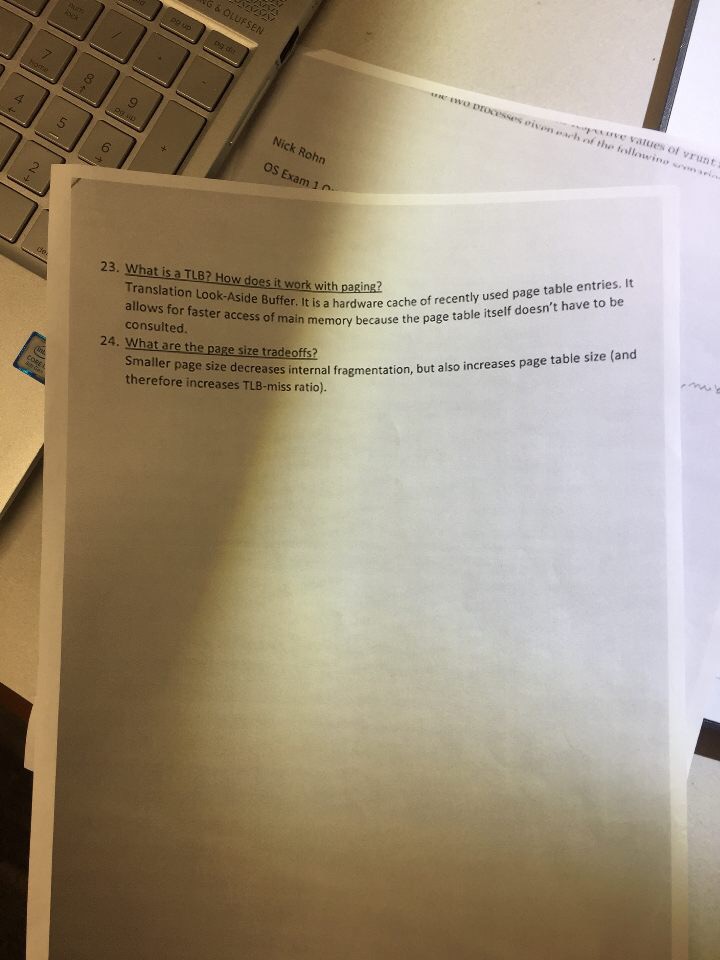
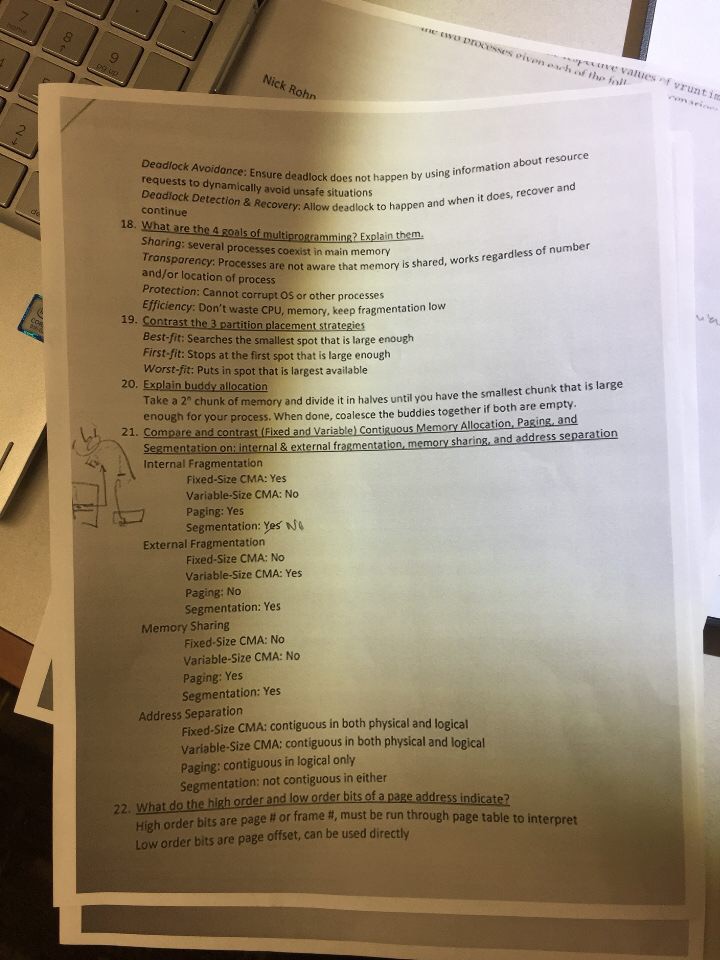
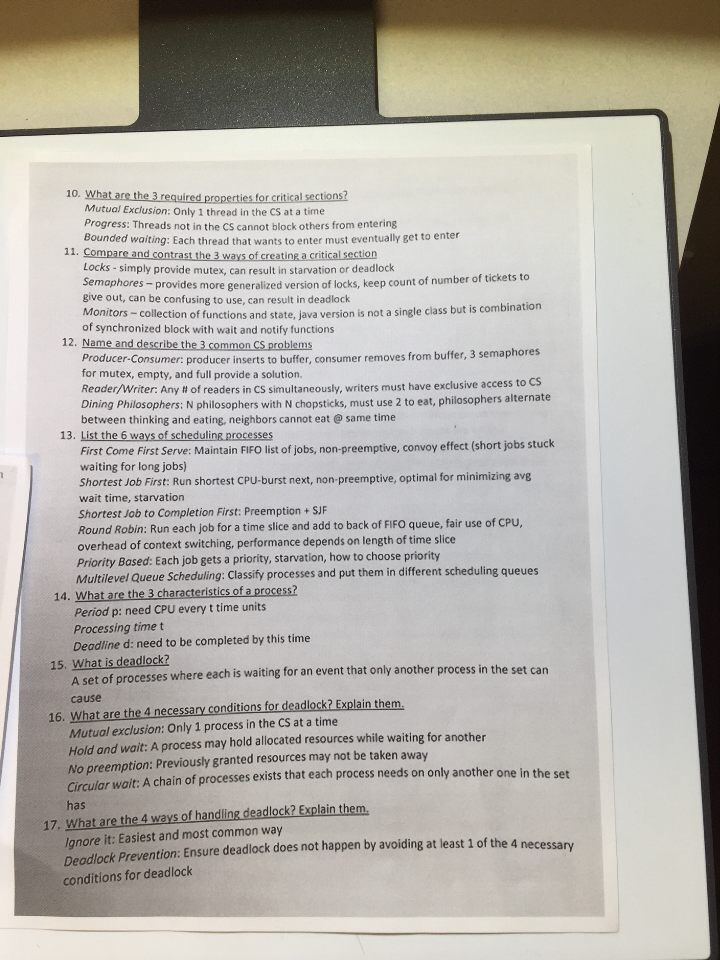
Process termination

Resource preemption

# EXAM1 QUES



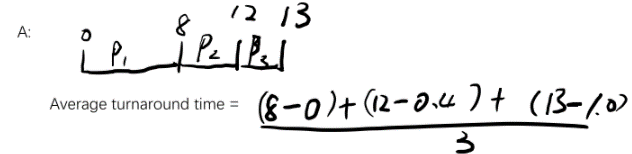
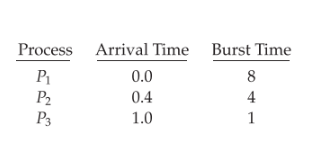


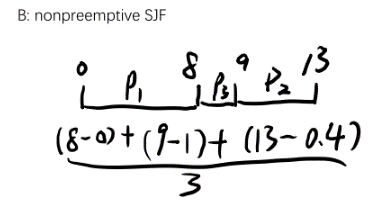
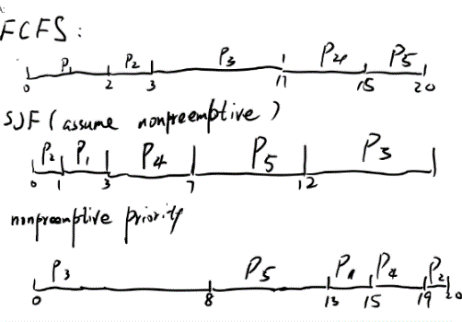




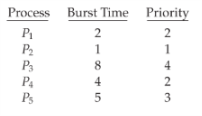
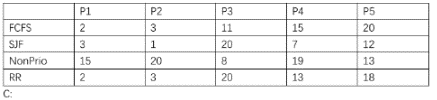
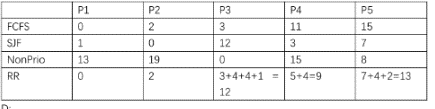
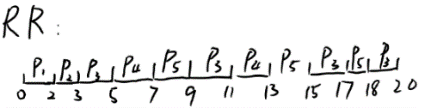
# FCFS, SJF, RATE MONOTONIC Algorithms …

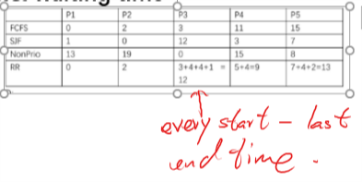
**Turnaround time:**



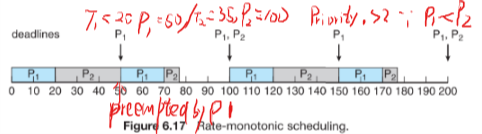


**Example2**

 **turnaround time && waiting time**

 **Response time is time(first response) is time(arrive)**

**Rate Monotonic**



**earliest deadline first**



**When to use notifyall or notify (ppt 2 sentences)**

**Chapter 8 尤其重要,HW**

**以及exam2 doc过**

