

Final Review

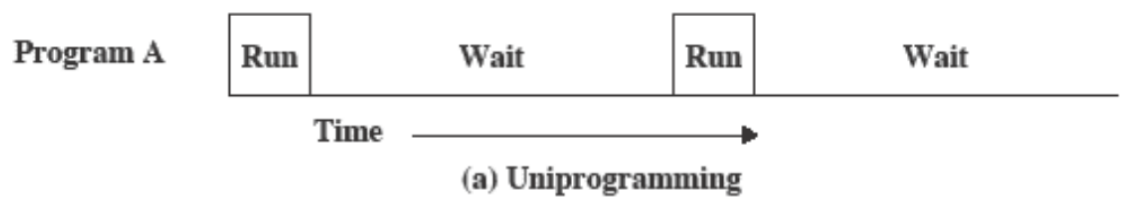
Chapter 1 Overview

- Computer
 - A computer is a machine that can be programmed to carry out sequences of arithmetic or logical operations automatically
 - Software
 - Computer programming instructions
 - Components
 - CPU
 - Data and instruction memory(RAM or DRAM)
 - Permanent storage memory (Hard drive)
 - Input/control devices
 - Information output devices
- Turing Machine
 - Abstract components
 - Finite state control, tape, tape head
 - Finite State Control
 - One of a finite number of states at each instant, and is connected to the tape head components
 - Two states
 - Accept and reject
 - Tape head
 - Scans one of the tape squares of the tape at each instant, and is connected to the finite state control. It can **read and write** symbols from/onto the tape, and it can move **left and right** along the tape
 - Tape
 - Consists of an infinite number of tape squares, each of which can store one of a finite number of tape symbols at each instant. The tape is infinite both to the left and to the right. Every other tape square is initialized to a special blank symbol
- **Halting Problem**
- Instruction Cycle
 - Instructions are fetched from memory one at a time
 - The fetched instruction is then executed by the processor
- Disk Cache
 - A portion of main memory used as a buffer to temporarily hold data for the disk
 - The data are retrieved rapidly from the software cache instead of slowly from disk
- Cache Memory

- Processor speed is faster than memory speed
- Increase the speed of meomory

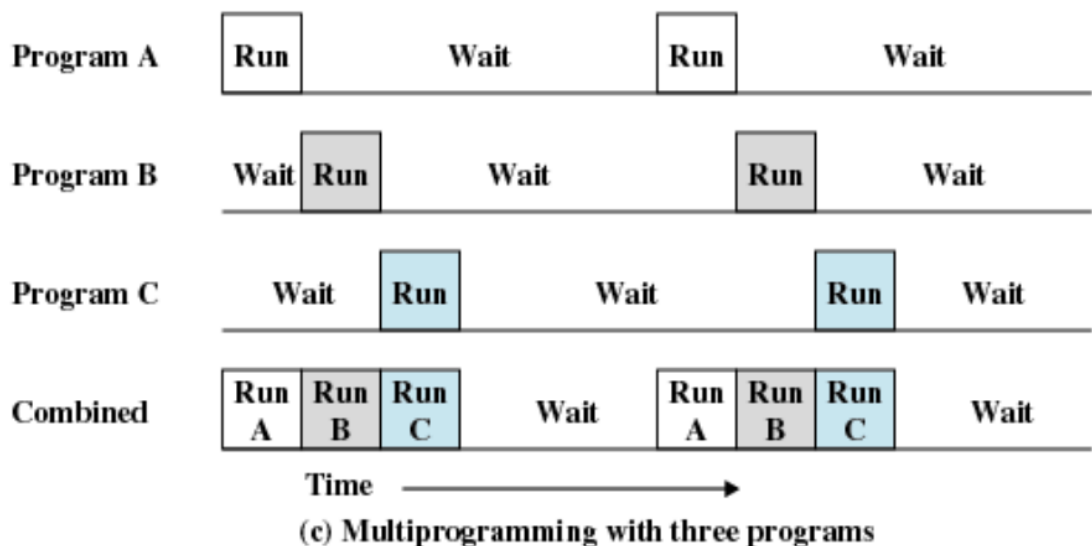
Chapter2

- Operating System
 - A program that controls the execution of application programs. An interface between applications and hardware
 - 3 Objectives
 - Convenient user, Efficient resource management, Ability to evolve
- Uniprogramming
 - Processor must wait for I/O instruction to complete before preceding



◦

- Multiprogramming
 - Running more than one programs at the same time
 - When one job needs to wait for I/O, the processor can switch to the other job

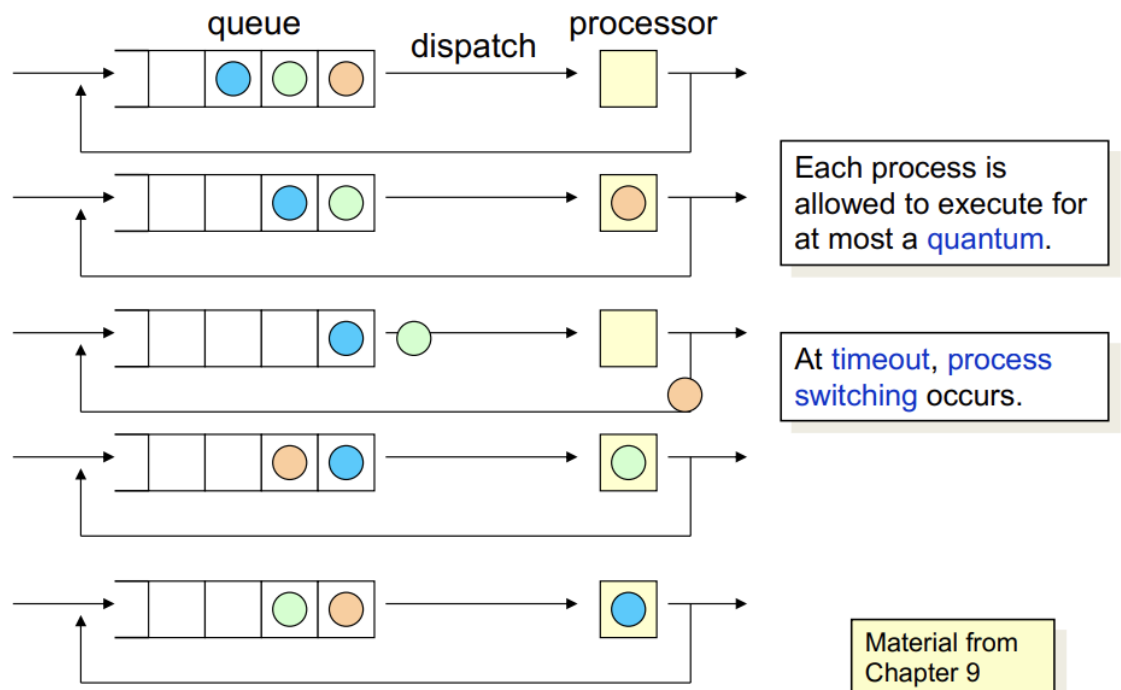


Chapter 3

- Process
 - An instance of a program running on a computer

- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions
- Elements
 - Program code, set of data, A number of attributes describing the state of the process
 - Identifier, State, Priority, Program counter, Memory pointers, Context data, I/O state information, Accounting information
- Process Management
 - Interleave the execution of several processes
 - Allocate resources to processes
 - Support interprocess communication
- Process creation
 - Submission of a batch job
 - User logs on
 - Create to provide a service such as printing
 - Spawned by an existing process
- Process termination
 - Batch job issues Halt instruction
 - User logs off
 - Process executes a service request to terminate
 - Parent may terminate so child processes terminate
 - Operating system intervention
 - Error and fault conditions
- **Dispatcher**
 - A small program that switches the processor from one process to another
- Scheduling

- Round-robin scheduling



- When *Timeout* or *I/O*, or *wait for other events*, a process cannot continue running and must leave the processor, it is **preempted**
- Priority Scheduling
 - Each process is assigned a priority
 - When the OS is going to dispatch a process, it chooses a ready process of the highest priority

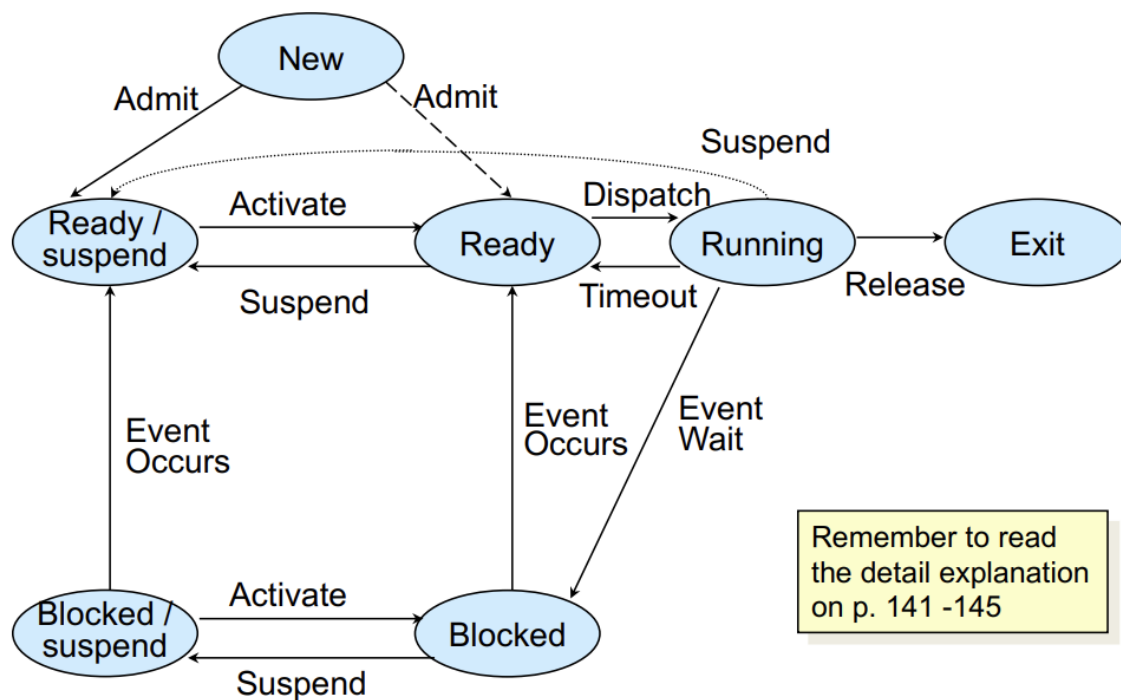
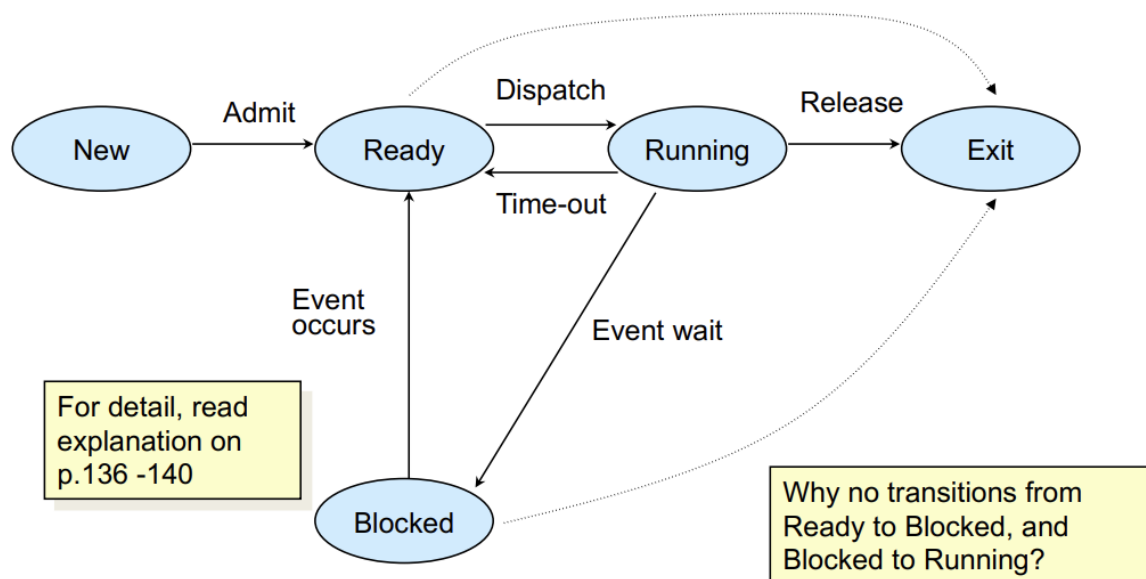
- Suspend Processes

- Stop the execution of a process **temporarily**
- Processor is **faster than I/O** so all processes could be waiting for I/O
- OS may suspend a process that causes a problem
- Interactive user request
- Timing: a process that executes periodically
- Swapping: sometimes OS may swap a blocked process to disk to free up more memory

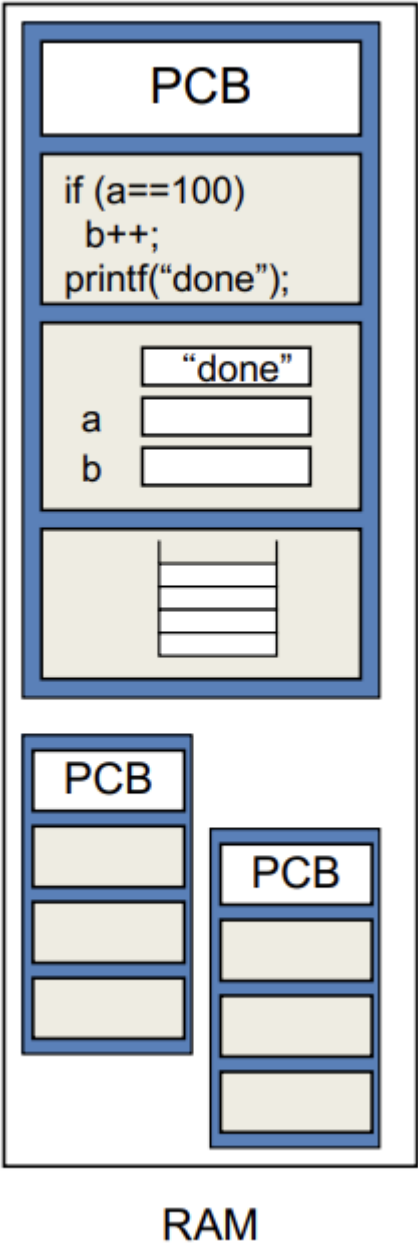
- Five-state Model

- Running - being executed by the processor
- Ready - ready to execute
- Blocked - cannot execute until some event occurs
- New - created, but not yet admitted'

- Exit - released



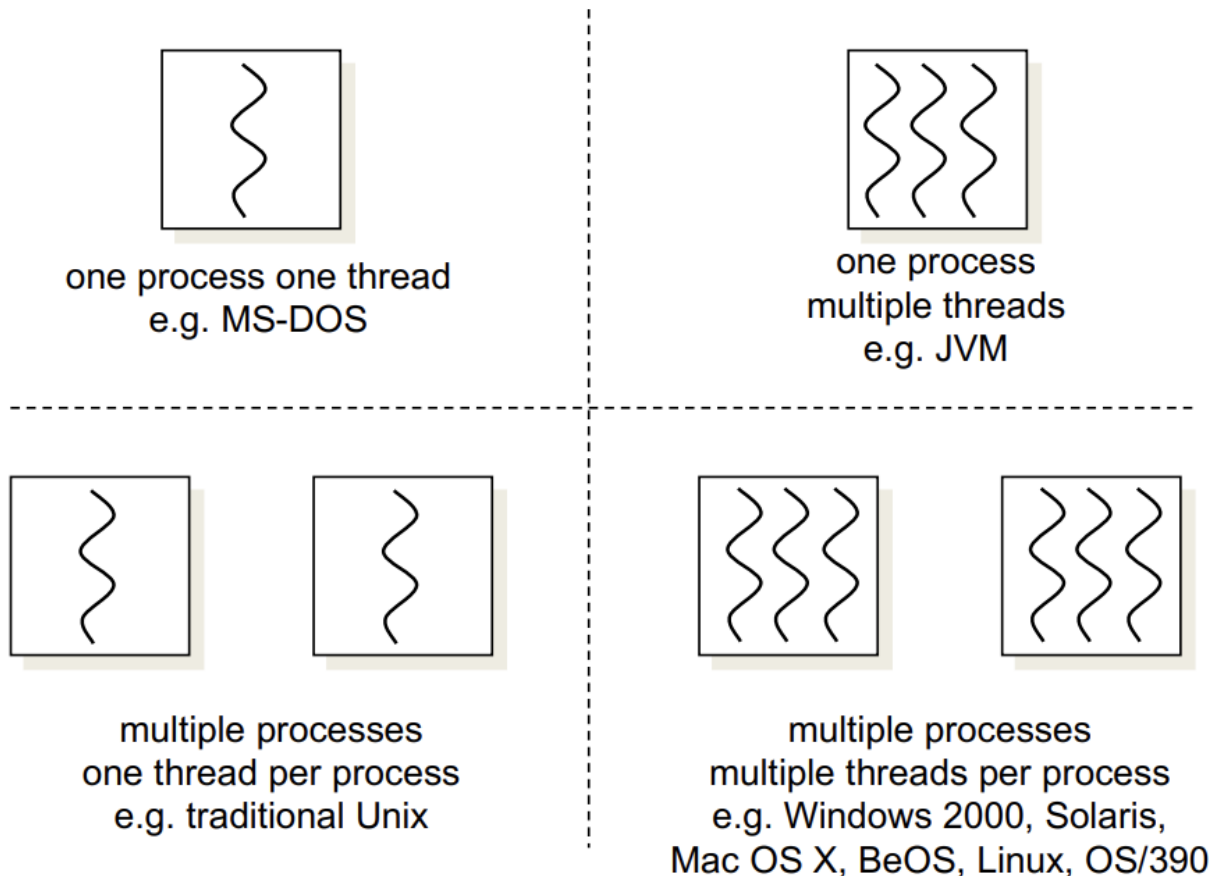
- Process Image
 - Process control bloc(PCB)
 - PCB contains data that the OS needs to control the process
 - Process identification
 - Processor state information
 - Process control information
 - User program
 - User data



- User stack

Chapter4

- Threads and Process



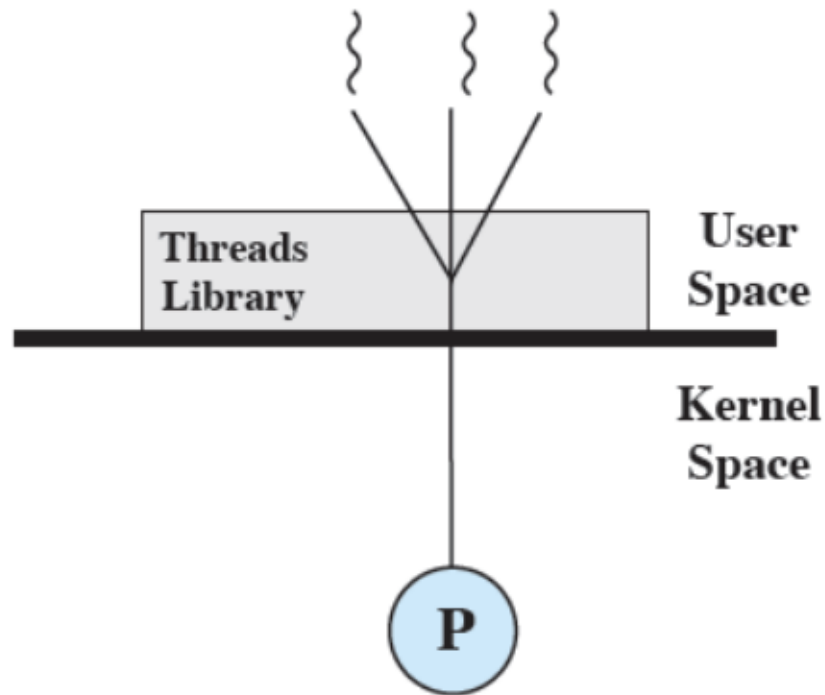
- Multithreaded process model

- In multithreaded process model, the OS keeps a Thread Control Blok(TCB) for each thread

- Thread

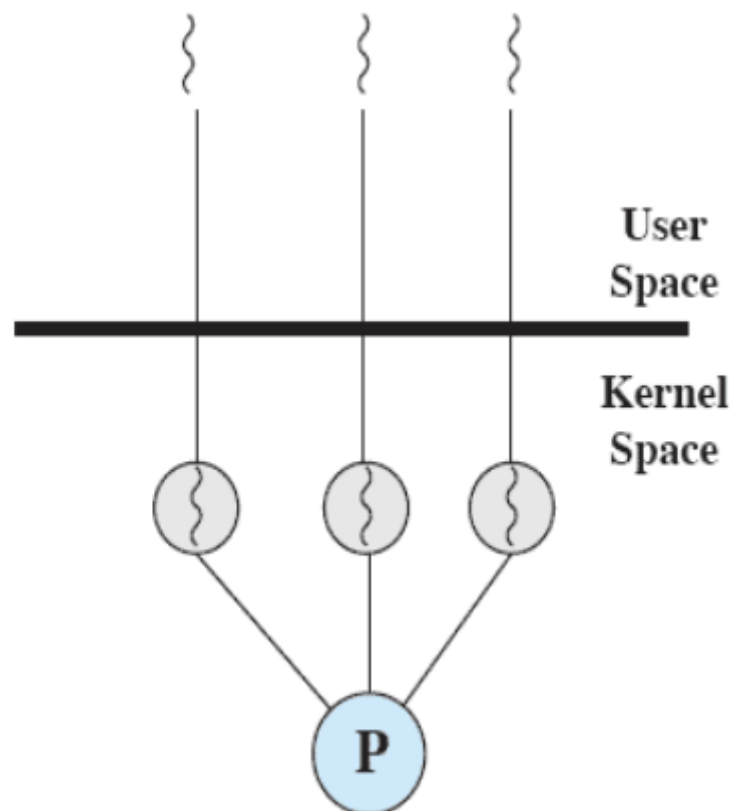
- Execution states
 - Running, Ready, Blocked
- Suspending a process involves suspending all threads of the process
 - All threads share the same address space
- Synchronization
 - It is necessary to synchronize the activities of the various threads
 - Share the same address and affects the other threads
- User-level Threads (ULTs)
 - All thread management is done by the application

- The kernel is not aware of the existence of threads

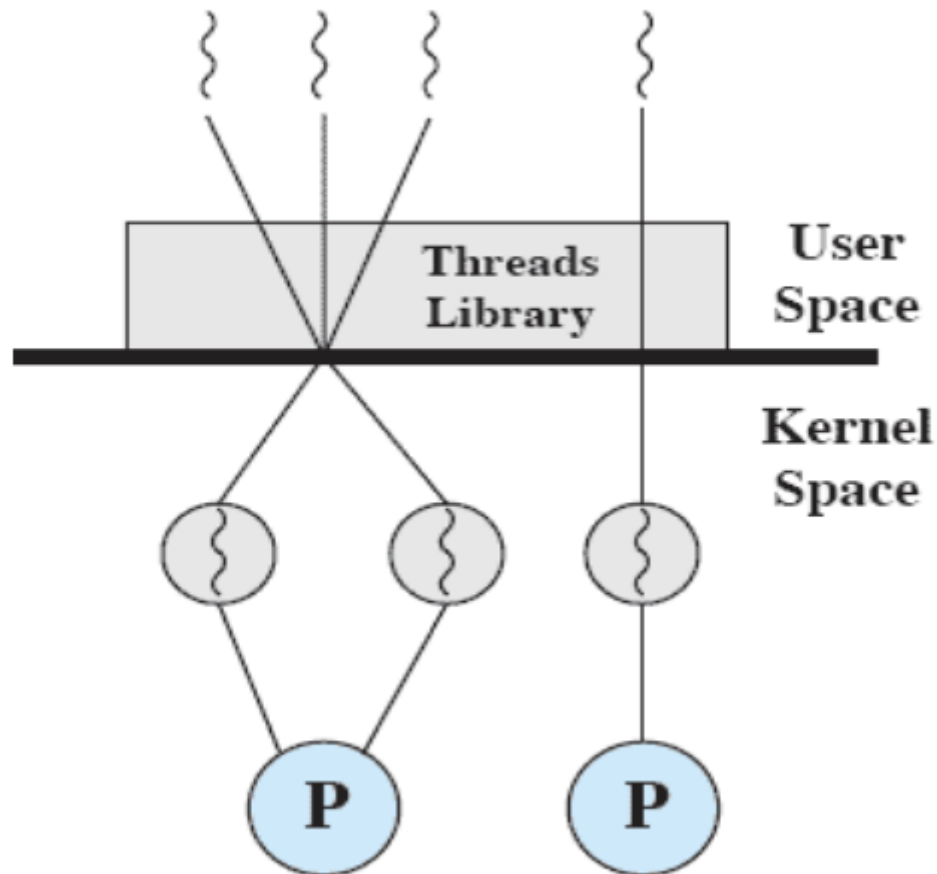


-Kernel-Level Threads (KLTs)

- Thread management is done by the kernel
- No thread mangement is done by the application



-Combined



- Benifits
 - Less time to create/switch/terminate
 - Easier communication
- Summary
 - Process
 - Have a virtual address space which holds the process image
 - Protected access to memory, files, and I/O resources
 - Thread
 - 3 execution states
 - execution stack
 - Thread context is saved and restored in thread switching
 - Has access to the memory and resources of its process

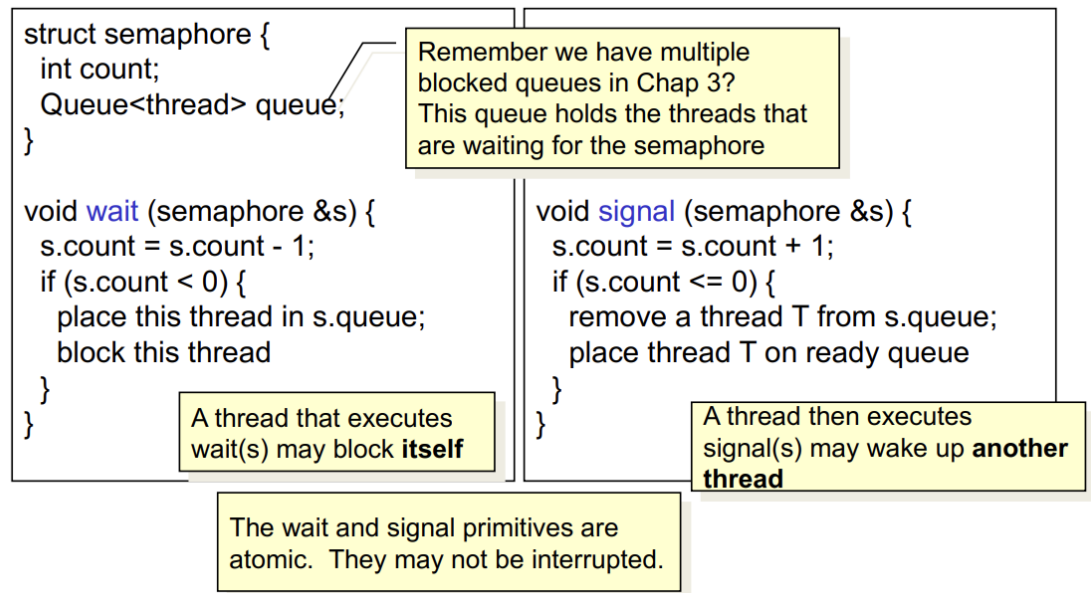
Chapter5

- Some defination
 - Multithreading - multiple threads within a process
 - Multiprogramming - multiple processes in a uniprocessor system
 - Multiprocessing - multiple processes within a multiprocessor
- Concurrency Key Terms

- Atomic Operation
 - It is considered as undividable operation, which ensure safe shared resource
- **Critical Section**
 - Critical section is a section of code within a process that requires to shared resources and which may not be executed while another process in a corresponding section of code
 - The programmer marks the critical sections of the program with some special code. Theses code must ensure that ar most one process/thread can be running inside the critical sections
 - 3 approaches to critical section
 - Software approach, Hardware support, OS support
- **Deadlock**
 - Permant blocking of a set of processes that compete for system resources
 - 4 necessary conditions: Mutual exclusion, Hold and Wait, No Preemption, Circular Wait
- Livelock
 - Two or more processes continuously change their states in response to changes in the other process without doing any useful work
- Mutual Exclusion
 - Only one thread/process is allowed to be in the critical section
- Race Condition -竞态条件 (Race Condition) 是指在多线程或多进程环境下，对共享资源的访问顺序不确定或不可预测，从而导致程序的行为不一致或不正确的现象
- Starvation
 - A process can never obtain access to resources it needs
- **Hardware Support**
 - Used internally in OS and device drivers to protect shared kernel data structures
 - 2 methods
 - Interrupt disabling, Spinlock
 - Problems of Interrupt Disabling
 - 通过禁用中断，可以确保某些代码段的执行不会被中断打断，从而提供了一种临时的保护机制
 - Limits the processor's ability to interleave programs
 - Does not work in Multiprocessor
 - disbaling interrupts on one processor does not prohibit other processors to enter CS
- **OS support**
 - The OS provides special functions (API) to protect critical sections and support synchronization
- Busy waiting -忙等待 (Busy Waiting) 是一种在计算机编程中使用的技术，它指的是一个进程或线程在没有其他有用工作可做时，通过循环检查条件来等待某个事件的发生。在忙等待期间，进程或线程会持续地检查特定的条件，直到条件满足为止。
- Problems of busy waiting
 - Consumes processor time
 - May cause starvation
 - May cause deadlock

• Semaphore

- A special integer value with 3 operations
 - initialize to a non-negative value
 - wait -decrements the value. If the value < 0 , the thread executing **wait** is blocked
 - signal -increments the value. If the value < 0 , then a thread blocked by a **wait** operation is unblocked



- How to protect CS
 - Create a shared semaphore S, initialize to 1 enclose the CS in each thread by wait(s) and signal(s)

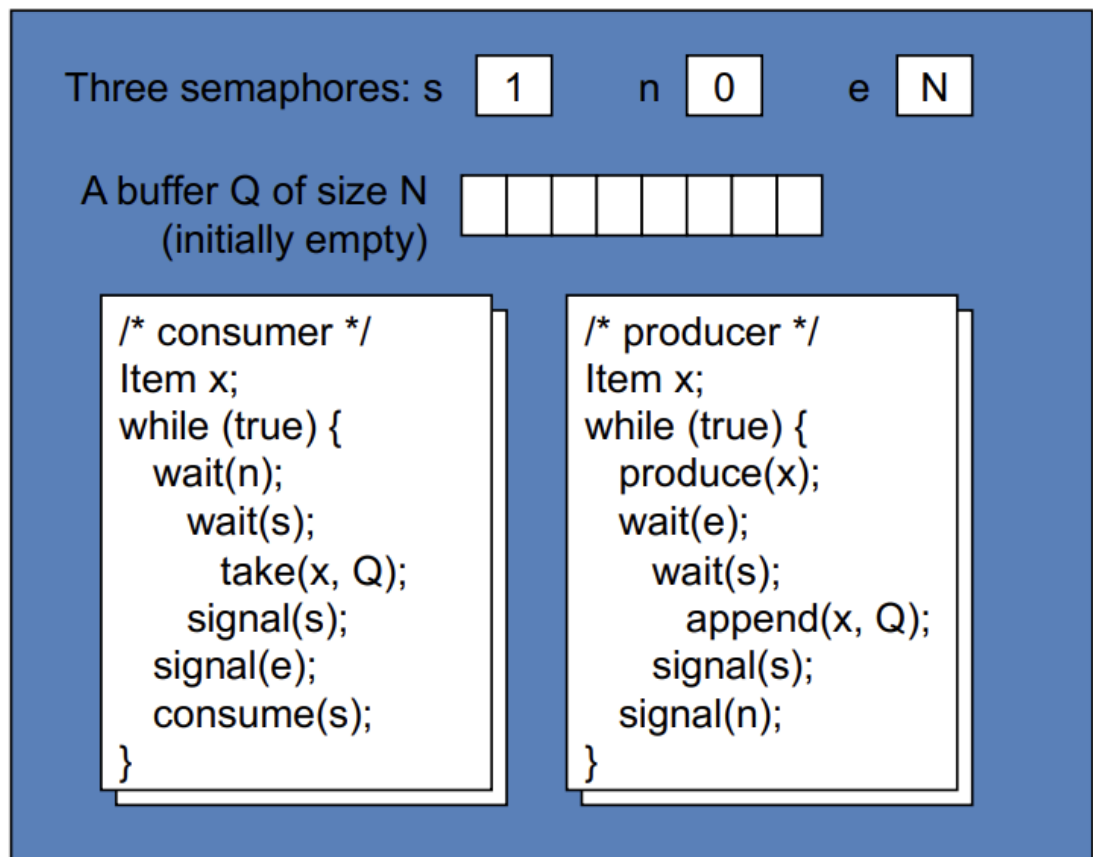
An observation: at any time,

- $s = 1$: No threads is inside the CS
- $s = 0$: a thread is inside the CS. Nobody is waiting
- $s < 0$: a thread is inside the CS. $|s|$ is the number of threads blocked in queue s.queue

- Strong semaphores
 - the process that has been blocked the longest is release form the queue first
 - The order in which processes are removed from the queue is not specified

• Producer/Consumer Problem

- 2 process/threads communicate indirectly through the shared buffer
 - **producer** produces an item, appends it to shared buffer, and repeats
 - **consumer** takes an item from the shared buffer, consumes it, and repeats
 - protect the **shared buffer**, and prevent buffer underflow/overflow
- Producer has to wait for empty slot(semaphore e)
 - 这边的e表示可使用的位置
- Consumer has to wait for available item(semaphore n)
 - 这边的n表示被占用的位置



- Monitor
 - Monitor is a software module
 - Local data variables are accessible only by the monitor
 - Process enters monitor by invoking one of its procedures / methods
- Readers/Writers Problem
 - Readers read the value of shared data
 - Writers modify shared data
 - When a writer is writing, no readers should read

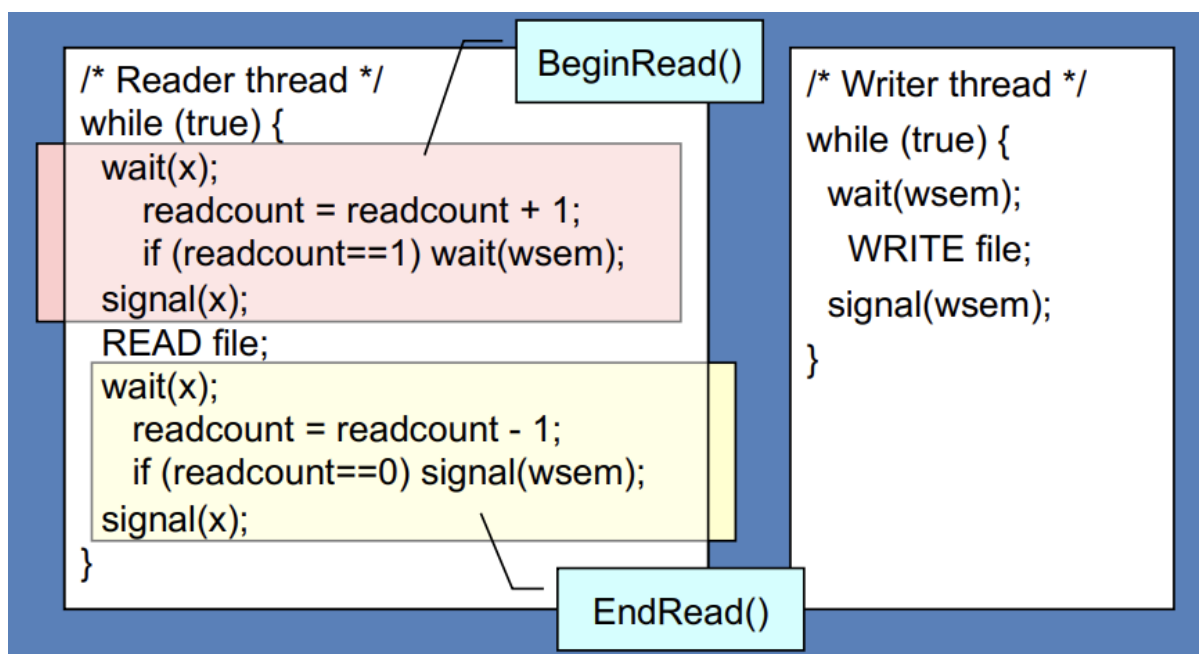
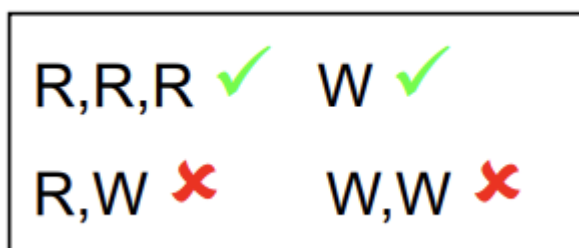
- When a reader is reading, no writers should write

1. 读者优先解决方案:

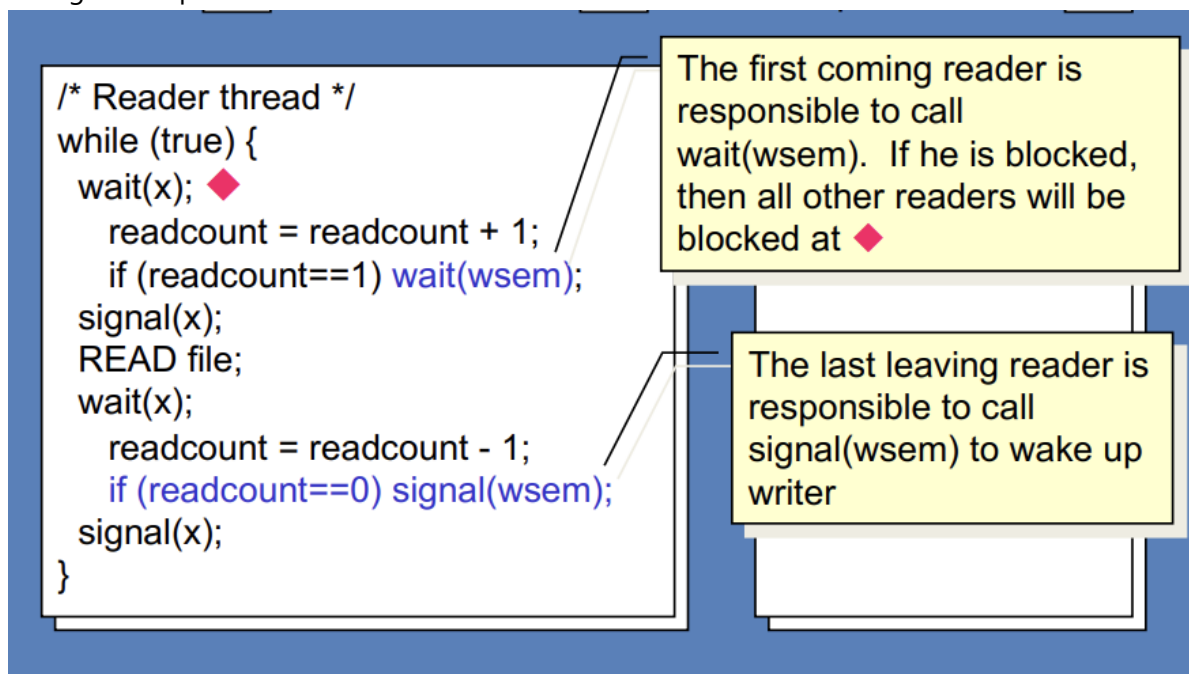
- 读者优先的解决方案允许多个读者同时访问共享资源，但当有写者等待时，读者必须等待。
- 使用信号量和计数器来实现读者/写者问题的解决方案。具体而言，使用一个互斥信号量（mutex）来保护对计数器和共享资源的访问，一个计数器来记录当前读者的数量。
- 当有读者访问资源时，读者计数器递增；当读者访问结束时，读者计数器递减。当读者计数器为1（即当前仅有一个读者）时，写者需要等待；当读者计数器归零时，写者被唤醒。
- 写者在访问资源之前获取互斥信号量，确保独占访问资源。

2. 写者优先解决方案:

- 写者优先的解决方案允许写者尽快地访问共享资源，而当有写者等待时，读者必须等待。
- 使用互斥锁和条件变量来实现读者/写者问题的解决方案。互斥锁用于保护对共享资源的访问，条件变量用于读者和写者之间的等待和唤醒。
- 当有写者等待时，读者需要等待条件变量，直到写者完成操作并发出信号。写者在访问和修改资源之前，需要获取互斥锁。

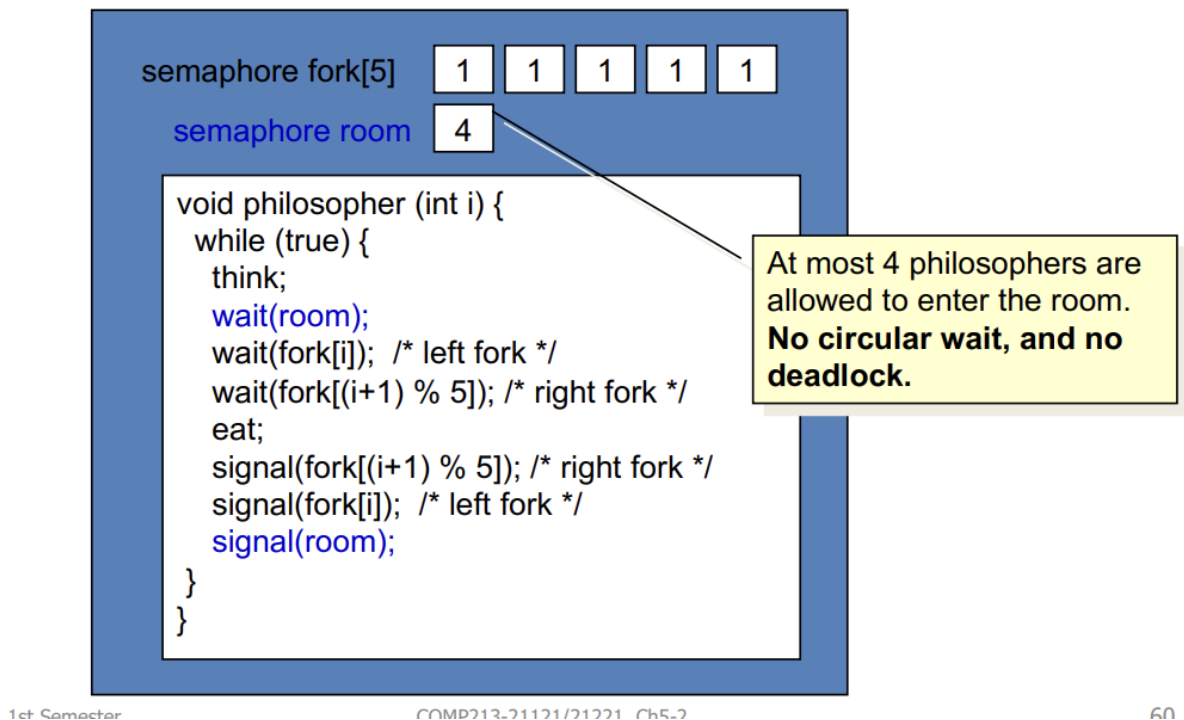


A single semaphore wsem is used to ensure mutual exclusion of both READs and WRITEs



- Dining Philosophers Problem
 - Deadlock occurs when there is a circular wait

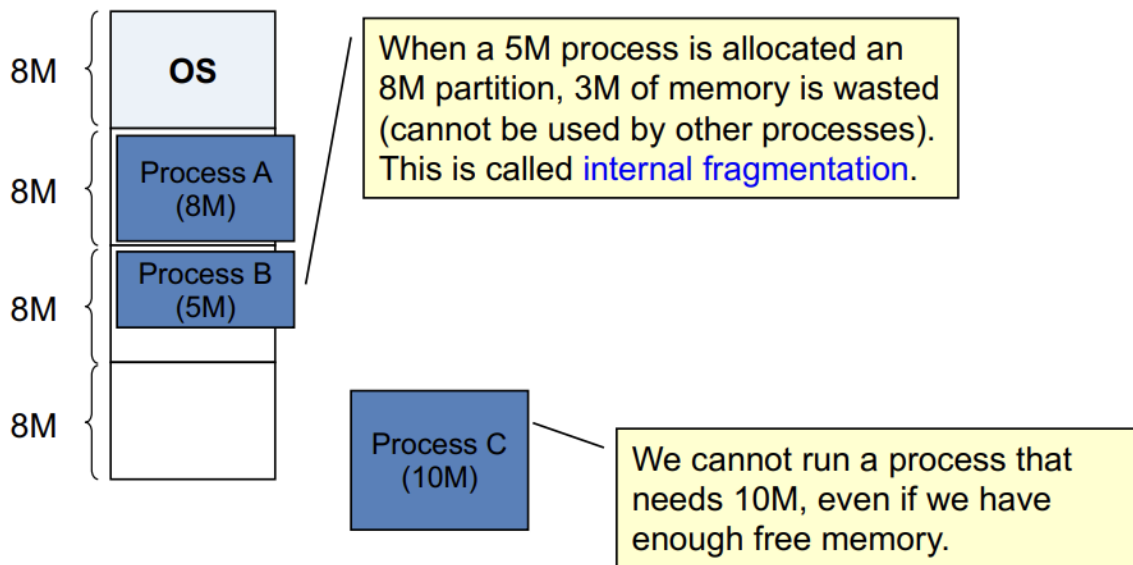
- To solve the dining problem



Chapter 7

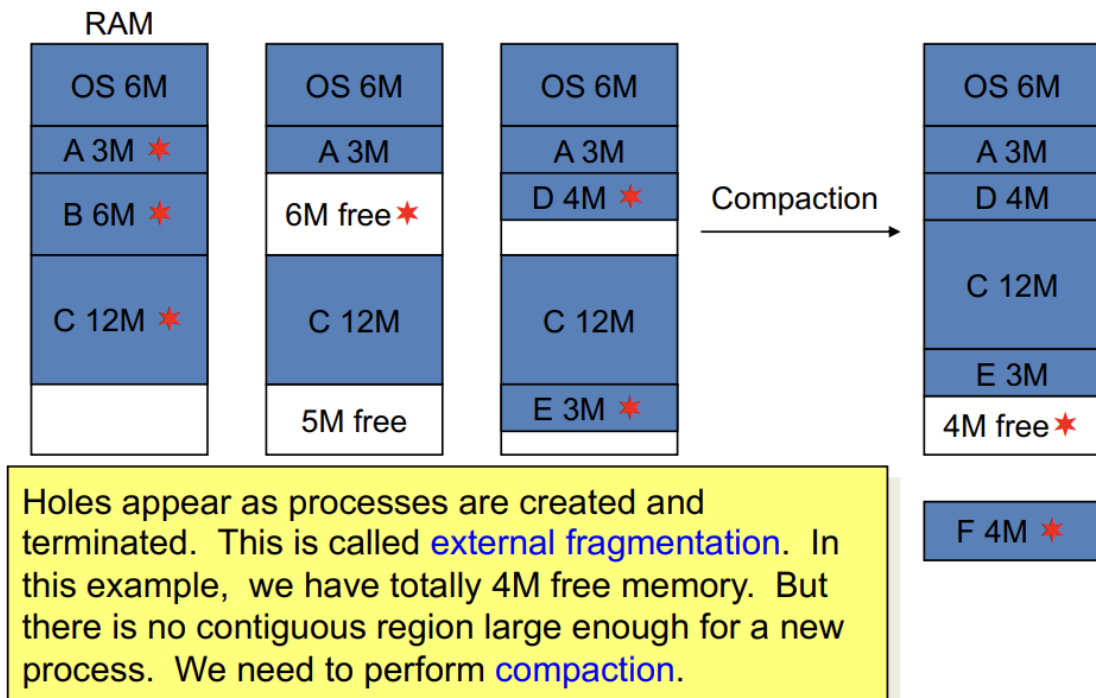
- Subdivide memory to accommodate multiple processes
- Memory Management Requirements
 - Relocation
 - When a program is compiled, the address of the code and data are fixed
 - Protection
 - Other processes cannot reference memory of a process without permission
 - Sharing
 - Allow several processes to access the same portion of memory
 - several processes running the same program
 - several processes sharing a common library
 - shared memory for data transfer between processes
 - Logical organization
 - programs are written in modules
 - Physical Organization
 - memory available may be insufficient so we use secondary memory to simulate primary memory
- Fragmentation
 - Fixed Partitioning
 - Partition available memory into regions with fixed boundary

- Internal fragmentation



- Dynamic Partitioning
 - Partitions are of variable length and number
 - Use external fragmentation
 - **Process is allocated exactly as much memory as required**
- External Fragmentation

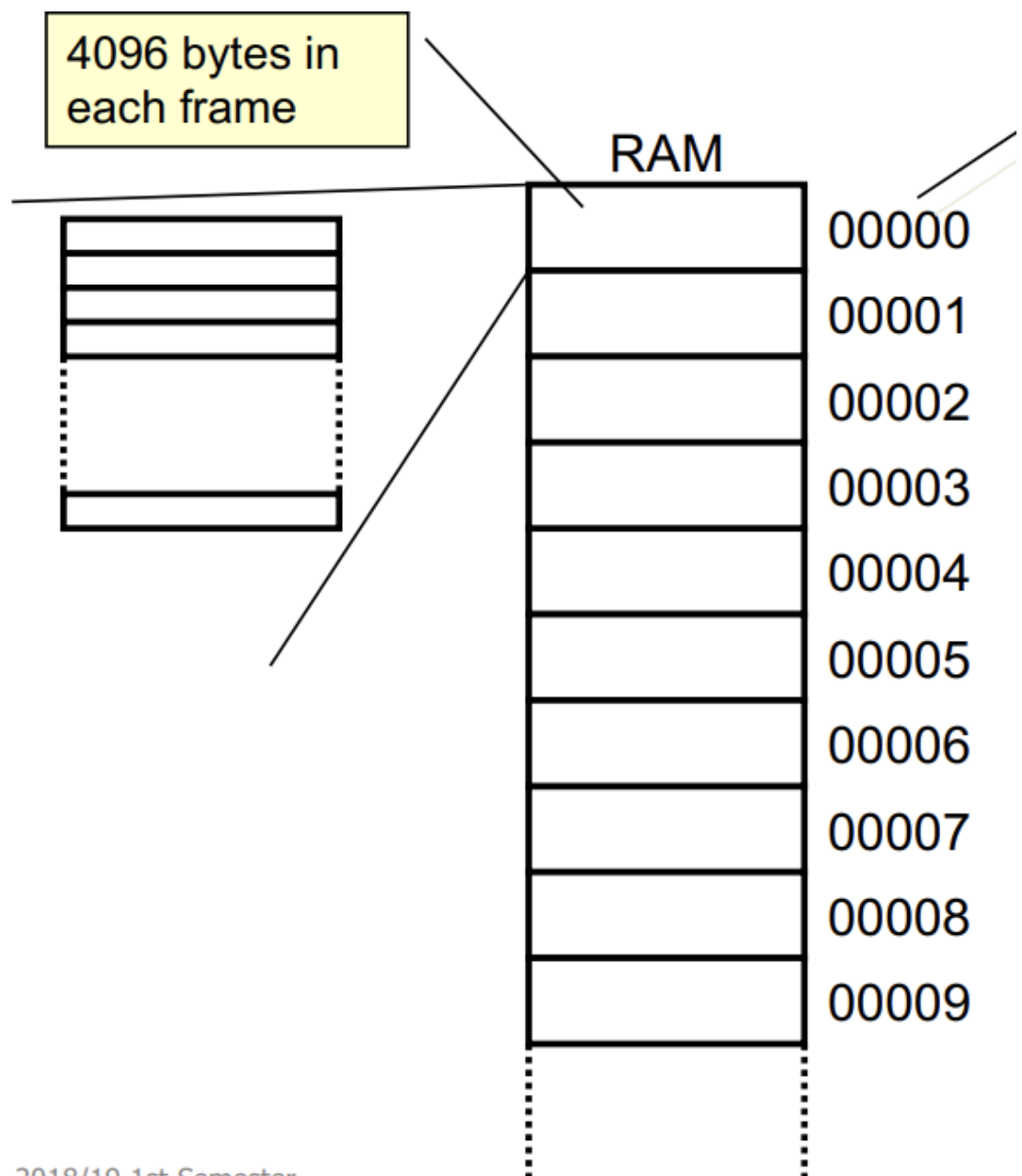
External Fragmentation



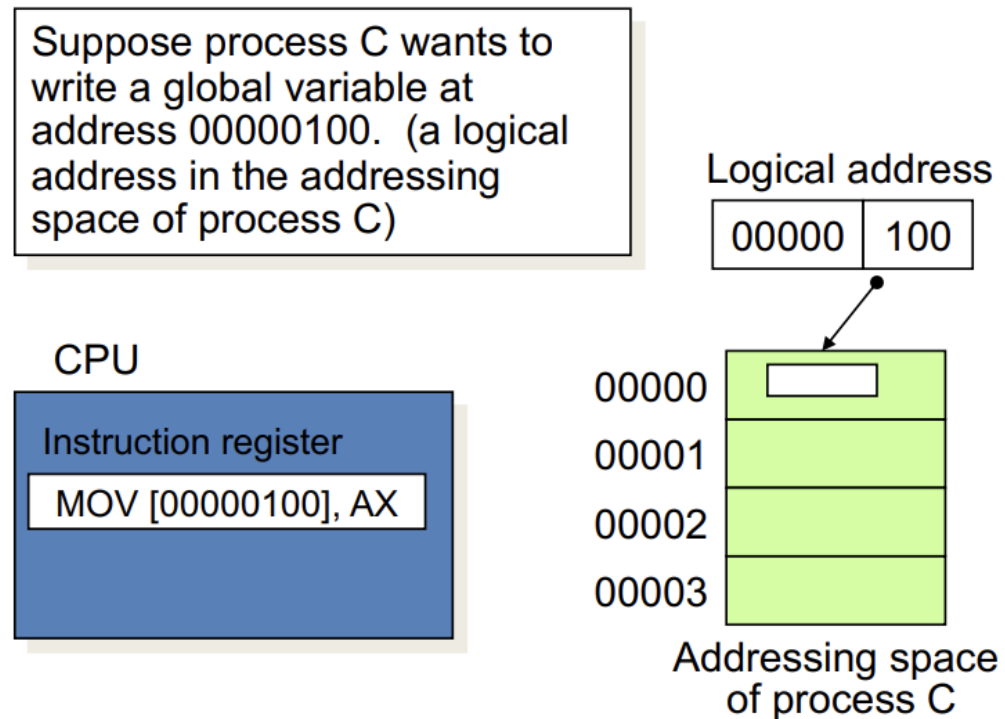
- Placement Algorithm
 - First-fit

- Starts scanning memory from the beginning and chooses the first available block that is large enough
 - Next-fit
 - Starts scanning memory from the location of the last placement and chooses the next available block that is large enough
 - Best-fit
 - chooses the block that is closest in size to the request
 - worst performer
- Physical memory / RAM
 - Each byte in the RAM is identified by a 32 bit physical address
 - Frame
 - The RAM is divided into chunks of equal size. They are called frames

r i d



- Each process has its own 'memory' called addressing space. It is an **illusion** created by the hardware and the OS
- The addressing space of each process is divided into chunks of equal size called pages
 - The pages in the addressing space of each process are mapped to the frames in the physical memory
 - The OS maintains a data structure called page table to store the mapping between the pages and frames



- **Logical address is the address as seen by a process. The process does not know where in the RAM the byte actually resides in**
 - **Physical address is the address actually used by the CPU to read the physical memory. It is the actual address of the byte in RAM**
- Segments
 - The addressing space of a process is divided into parts of unequal length called segments.
 - The segments are mapped to the RAM through a segment table. It marks the base address and length of the segment

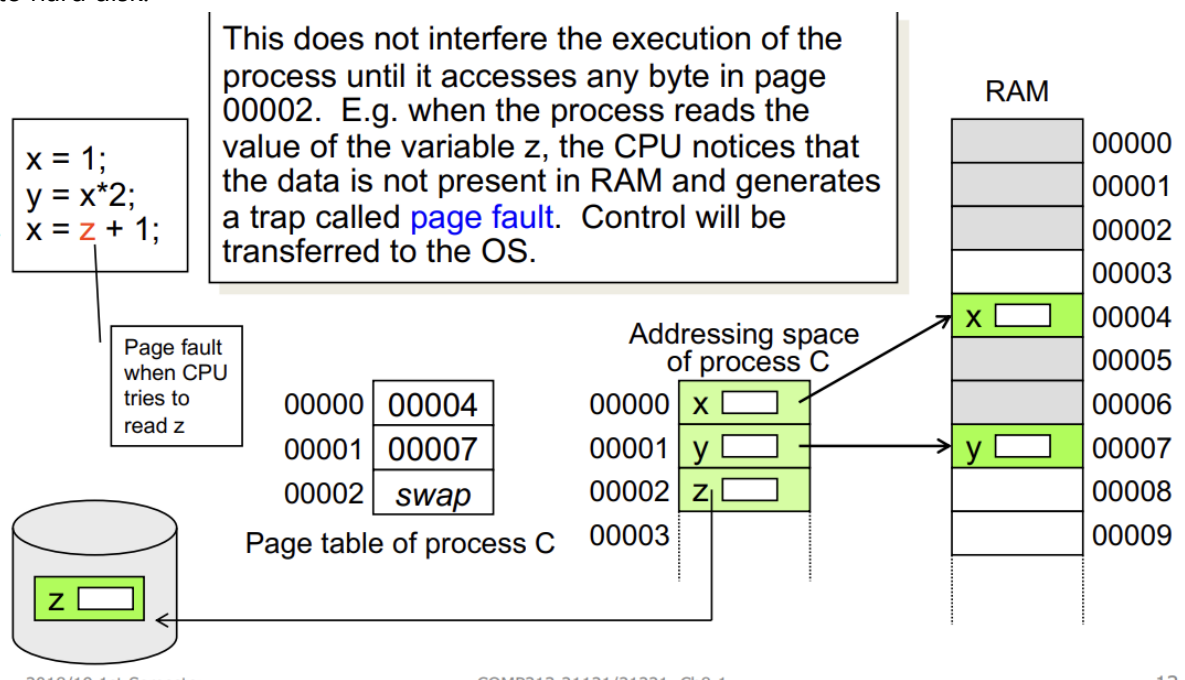
- Combined Paging and Segmentation

Combined Paging and Segmentation

- Paging is transparent to the programmer
- Paging eliminates external fragmentation
- Segmentation is visible to the programmer
- Segmentation allows for growing data structures, modularity, and support for sharing and protection
- Each segment is broken into fixed-size pages

Chapter 8

- Virtual memory
 - "Simulate more RAM using hard disk space"
 - The program/data used by a process do not have to be present in the RAM all the time. Pages not currently used are swapped out to disk
- How the vm works?
 - When the OS find there is not enough memory in RAM, some unnecessary frame may be moved to hard disk.



- When process want to access the frame

- A page fault occurs when a reference memory address is not present in RAM

... The CPU generates a page fault trap, and control is transferred to the OS. The OS

- block the process (other processes can run)
- find the page in the swap file in hard disk
- find an unused frame in RAM
- issue a read request to the hard disk

... When the page loading is completed,

- an I/O interrupt occurs
- the OS updates the page table of process that caused the page fault
- The OS wakes up the process

- The state changes in Page Fault Handling is Running - blocked - ready
- Efficiency
 - Since the hard disk I/O is slow, it causes *thrashing*
 - Thrashing is a state in which the system spends most of its time swapping pieces rather than excuting instructions
 - To avoid thrashing, the OS tries to guess (based on recent history) which pieces are least likely to be used in the near future.
 - Principle of Locality
 - 它指的是程序在任何给定时间倾向于访问其地址空间或内存的相对较小部分的观察结果。
 - Temporal locality – tendency to access data that are accessed recently
 - Spatial locality – tendency to access data near data that are recently accessed
- Hardware support
 - Page Table Entry

- Page table is an array of page table entries

Each page table entry contains

- frame number – where this page maps to
- present bit – the page is in RAM?
- modify bit – any byte modified in the page since last loaded?
- other control bits...

- **The entire page table may take up too much memory, we can stored it in virtual memory**

- Present Bit
 - In address translation, the CPU first check the present bit of the page
 - If $p = 1$, continues, if $p = 0$, page fault occurs.
- Modify bit
- Page Table Pointer
 - A register that holds the starting address of the page table of the running process

1. 最近最少使用 (Least Recently Used, LRU) : 根据数据最近被访问的时间来进行替换。当需要替换时, 选择最久未被访问的数据块进行替换。
2. 先进先出 (First-In-First-Out, FIFO) : 按照数据最早进入缓存的顺序进行替换。当需要替换时, 选择最先进入缓存的数据块进行替换。
3. 最不经常使用 (Least Frequently Used, LFU) : 根据数据被访问的频率来进行替换。当需要替换时, 选择被访问次数最少的数据块进行替换。
4. 随机替换 (Random Replacement) : 随机选择一个数据块进行替换, 不考虑其访问时间或频率。

- Replacement policy

- Selection of a page in memory to be replaced when a new page is brought in

- Optimal Policy (Impossible)

最佳替换策略 (Optimal Policy) 是一种理论上的替换策略，它假设在缓存或页面置换过程中，你知道未来所有访问的情况，并根据这些信息最佳替换决策。

最佳替换策略的原理是选择要被替换的数据块是未来最长时间不会再被访问到的数据块。通过这种方式，最佳替换策略可以最大程度地减少缓存未命中的次数，从而达到最佳的性能。

- Replaced the page for which the time to the next reference is the longest
- Results in the fewest number of page faults
- LRU (Difficult)
 - Replaces the page that has not been referenced for the longest time
- FIFO
 - Replaces the page that has been in memory the longest
- Clock
 - Needs an additional bit U in PTE
 - when a page is first loaded in memory, OS sets U=1
 - when the page is referenced, CPU sets U=1

时钟替换策略的工作方式如下：

1. 初始化：将所有缓存块的访问位设置为0，并选择一个起始位置。
2. 替换过程：当需要替换一个缓存块时，从当前位置开始，按照顺时针方向访问每个缓存块。
 - a. 如果访问位为0，则将其替换出去，并将新的数据块放入该位置，并将访问位设置为1。
 - b. 如果访问位为1，则将其保留不替换，并将访问位设置为0。
3. 更新当前位置：每次访问都会更新当前位置，使其指向下一个缓存块。

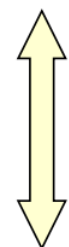
时钟替换策略相对于LRU（最近最少使用）策略来说，实现更简单且计算开销更低。它适用于硬件实现，特别是在处理器的高速缓存中。然而，时钟替换策略可能会导致一些冷数据一直被保留在缓存中，而热数据可能被替换出去的问题，因为访问位只记录了最近一次访问的情况，而没有考虑到访问的频率。

■ Disadvantage

- Categories of frames using U, M

U=0, M=0	Not accessed recently, not modified
U=0, M=1	Not accessed recently, modified
U=1, M=0	Accessed recently, not modified
U=1, M=1	Accessed recently, modified

Most favorable to replace



Should be expensive to replace

- Resident set
 - Portion of process that is in main memory
 - Fixed-allocation, Variable-allocation

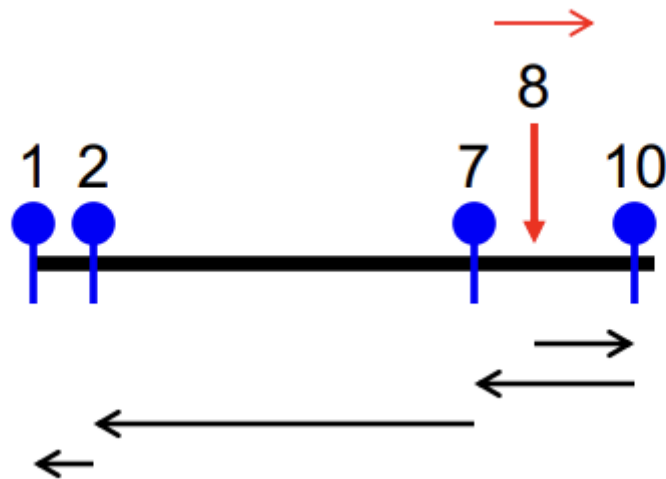
Chapter11

- Perfroming I/O
 - Programmed I/O -Process is busy-waiting for the operation to complete
 - Interrupt-driven I/O
 - I/O command is issued
 - Processor continues executing instructions
 - I/O module sends an interrupt when done
 - Direcy Memory Access
 - DMA module controls exchange of data between main memory and the I/O device

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		DMA

- I/O buffering
 - One buffer
 - Double buffer

- two buffers are used
- Circular buffer
 - More than two buffers are used
- Disk Shceduling
 - Access time
 - Total = seek time + rotational dalay + data transfer time
 - Seek time - time required to move the disk arm to the required track
 - Rotational delay - time required to rotate the disk to the required sector
 - Data transfer time - time to read/write data from/to the disk
 - FIFO
 - Process requests in the order that the requests are made
 - SSTF
 - select the disk I/O that requieres the least movement of the disk arm from its current position
 - minimum seek time, but may cause starvation
 - SCAN
 - Arm moves in one direction only, satisfying all outstanding requests until there is not more requests **in that direction**. Then reversed



- C-SCAN
 - restrict scanning to one direction only
 - When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again
- FSCAN

SCAN算法的工作流程如下：

1. 将磁盘上的所有磁道划分为两个方向：正向和反向。可以将正向视为磁头移动方向的一端，而反向视为磁头移动方向的另一端。
2. 当有新的I/O请求到达时，该请求会被插入到相应的扫描队列中，根据磁道号的顺序将请求放入正向或反向队列中。
3. 当前正在进行的I/O请求完成后，磁头会继续在当前方向上移动，直到扫描队列中的所有请求都被处理完毕。
4. 如果当前方向上没有更多的请求需要处理，则磁头会立即改变方向，并开始处理另一个方向上的请求。
5. 这个过程会一直重复，直到所有的I/O请求都被处理完毕。