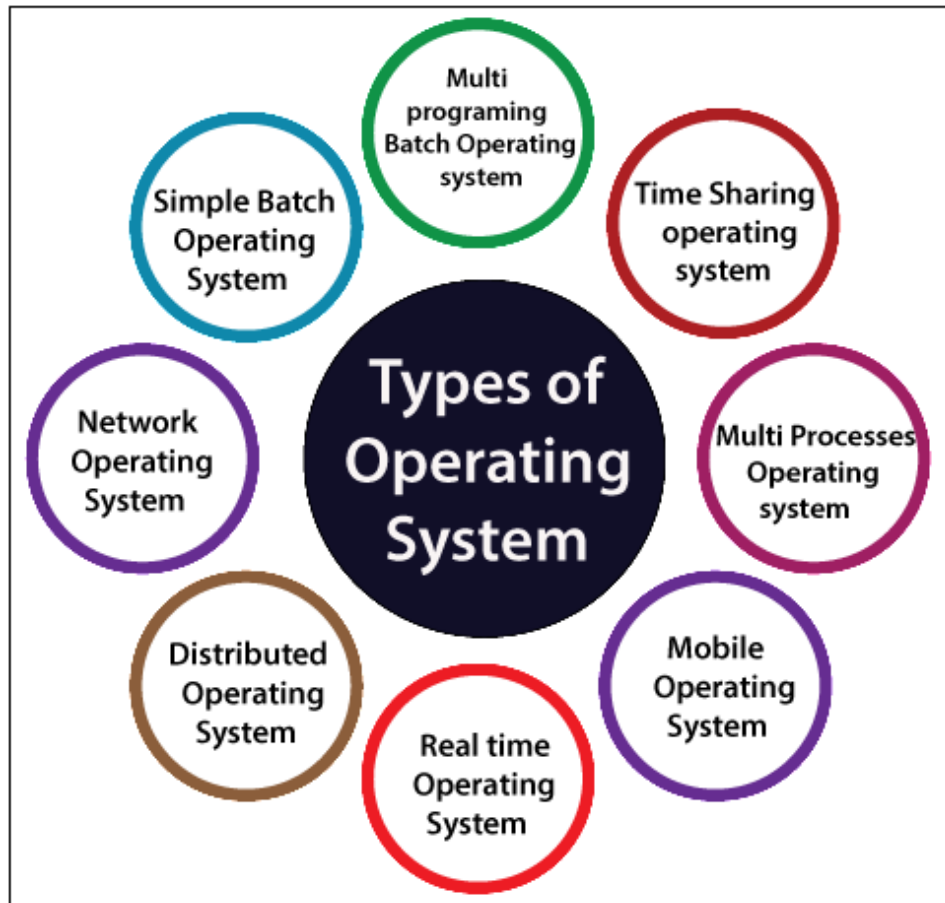








Operating System

Types of OS



What are the different types?

<p>Mac OS is a series of graphical user interface-based operating systems developed by Apple Inc. for their Macintosh</p>		<p>Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution.</p>		<p>Microsoft Windows is a series of graphical interface operating systems developed, marketed, and sold by Microsoft.</p>	
<p>iOS (previously iPhone OS) is a mobile operating system developed and distributed by Apple Inc. Originally unveiled in 2007 for the iPhone, it has been extended to support other Apple devices such as the iPod Touch</p>		<p>Android is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Initially developed by Android, Inc.</p>		<p>BSD/OS had a reputation for reliability in server roles; the renowned Unix programmer and author W. Richard Stevens used it for his own personal web server for this reason.</p>	

Real time Operating system

- Real time operating systems are used as OS in real time system.
- In RTOS tasks are completed in given time constraints.
- RTOS is a multitasking system where multiple tasks run concurrently
 - system shifts from task to task
 - must remember key registers of each task (this is called context of task)



EmbeddedCraft
crafting of intelligent systems

Real-Time Operating System

- ◆ An RTOS is an OS for response time-controlled and event-controlled processes. It is very essential for large scale embedded systems.
- ◆ RTOS occupy little space from 10 KB to 100KB
- ◆ The main task of a RTOS is to manage the resources of the computer such that a particular operation executes in precisely the same amount of time every time it occur.



Renesas automotive dashboard platform.



Soft RTOS...

- In a soft real-time system, it is considered undesirable, but not catastrophic, if deadlines are occasionally missed.
- Also known as “best effort” systems
- Most modern operating systems can serve as the base for a soft real time systems.
- Examples:
 - multimedia transmission and reception,
 - networking, telecom (cellular) networks,
 - web sites and services
 - computer games.

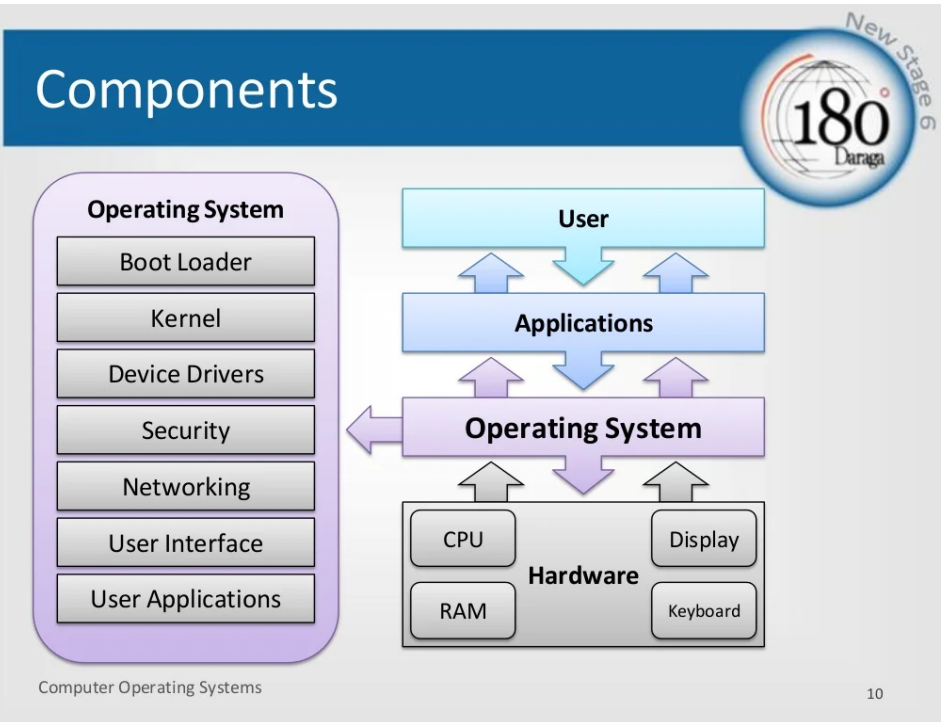


Hard RTOS...

- A hard real-time system has time-critical deadlines that must be met; otherwise a catastrophic system failure can occur.
- Absolutely, positively, first time every time
- Requires formal verification/guarantees of being to always meet its hard deadlines (except for fatal errors).
- Examples:
 - air traffic control
 - vehicle subsystems control
 - Nuclear power plant control



OS Structure

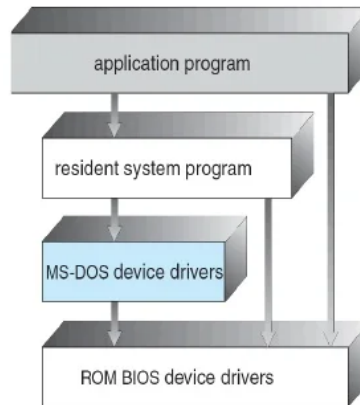


7. Operating-System Structure

- How OS components are interconnected and melded into a kernel.

7.1 Simple Structure

- Started as small, simple, and limited systems and then grown beyond their original scope
- Example : MS-DOS – written to provide the most functionality in the least space
 - Not divided into modules
 - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

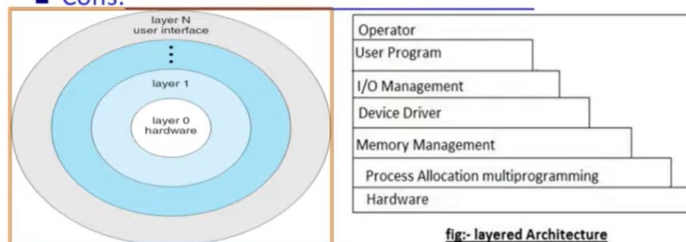


MS-DOS layer structure

Loganathan R, CSE , HKBKCE 15

Layered OS Architecture

- Lower levels independent of upper levels
 - N^{th} layer can only access services provided by $0 \sim (N-1)^{th}$ layer
- Pros: Easier debugging/maintenance
- Cons:





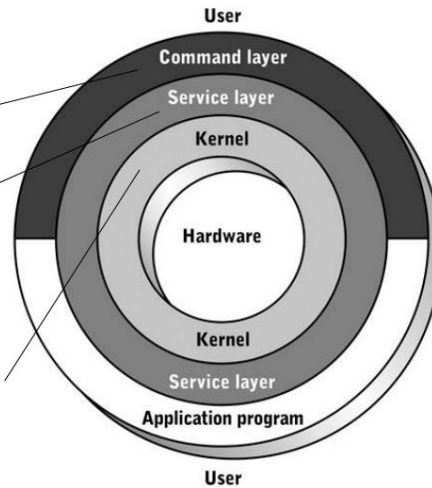
Operating System Layers

Figure 11-3 ▶
Operating system layers (shaded)

User's interface to OS

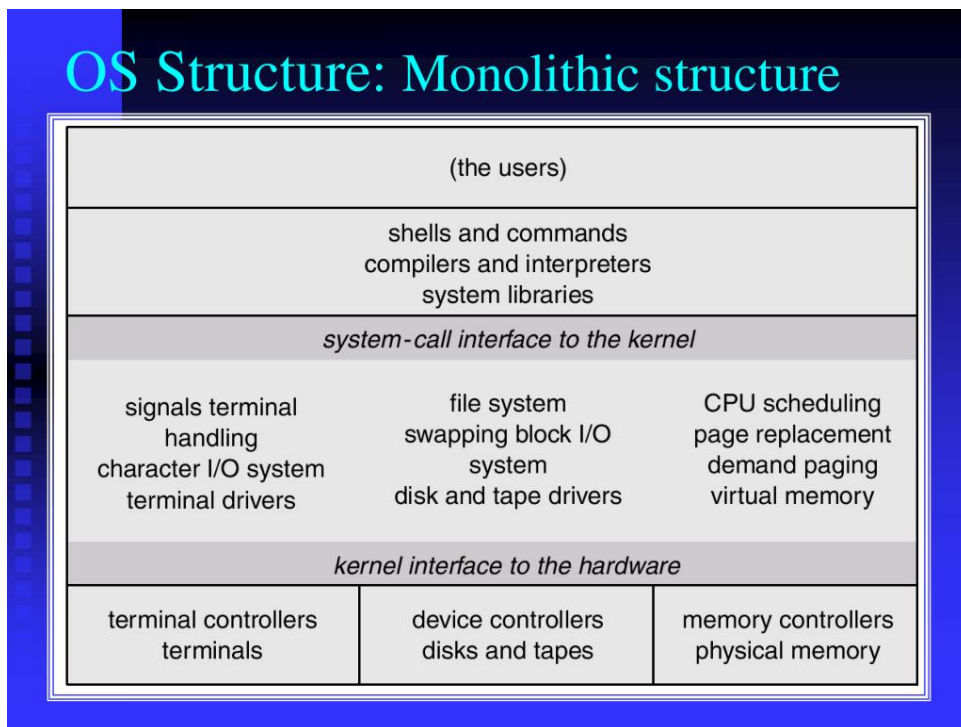
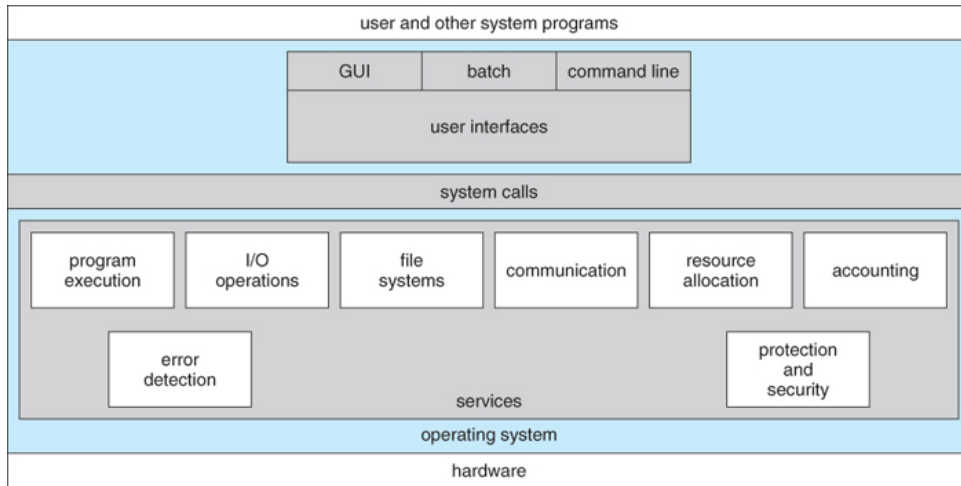
Contains set of functions executed by application programs and command layer

Manages resources; interacts directly with computer hardware



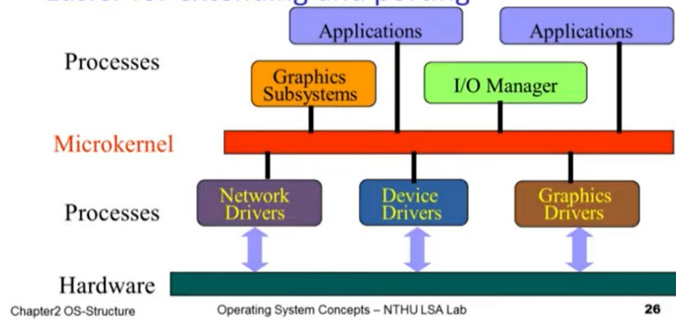
Hardware and Software Architecture

16

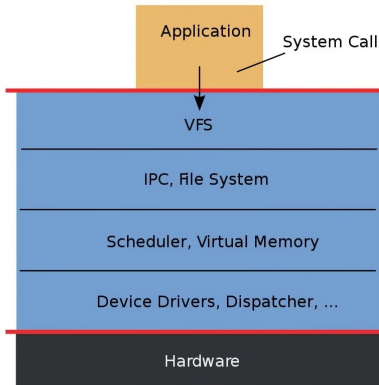


Microkernel OS

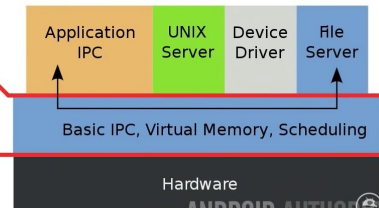
- Moves as much from the kernel into "user" space
- Communication is provided by message passing
- Easier for extending and porting



Monolithic Kernel based Operating System

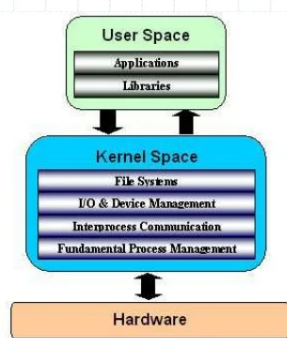


Microkernel based Operating System

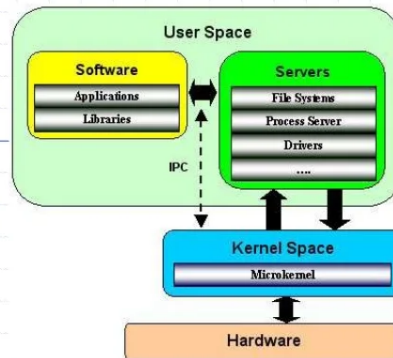


ANDROID AUTHORITY

Monolithic kernel Vs Microkernel



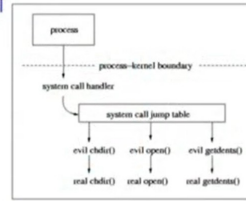
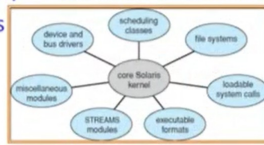
Monolithic Kernel



Microkernel

Modular OS Architecture

- Most modern OS implement **kernel modules**
 - Uses **object-oriented approach**
 - Each core **component is separate**
 - Each talks to the others over **known interfaces**
 - Each is **loadable** as needed within the kernel
- Similar to layers but with more flexible
- E.g., Solaris



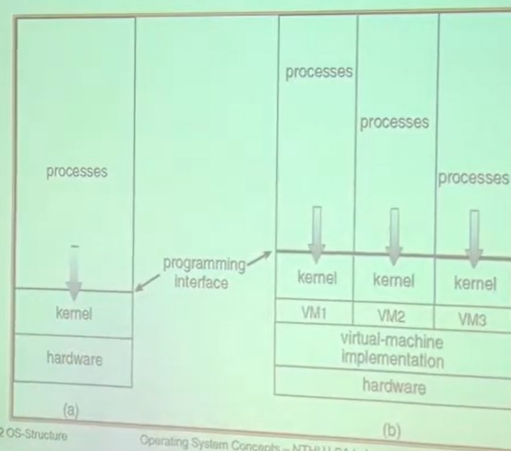
- How to write kernel module
 - http://www.linuxchix.org/content/courses/kernel_hacking/lesson8
 - http://en.wikibooks.org/wiki/The_Linux_Kernel/Modules
 - https://www.thc.org/papers/LKM_HACKING.html

Virtual Machine

Virtual Machine

- A **virtual machine** takes the **layered** approach to its logical conclusion
 - It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface **identical** to the underlying bare hardware
 - Each process is provided with a (virtual) copy of the underlying computer
- Difficult to achieve due to **"critical instruction"**

Virtual Machine



國立清華大學

Usage of Virtual Machine

- provides complete protection of system resources
- a means to solve system compatibility problems
- a perfect vehicle for operating-systems research and development
- A mean to increase resources utilization in cloud computing

Chapter2 OS-Structure Operating System Concepts – NTHU LSA Lab 30

國立清華大學

Vmware (Full Virtualization)

- Run in **user mode** as an application on top of OS
- Virtual machine believe they are running on bare hardware but in fact are running inside a user-level application

Chapter2 OS-Structure Operating System Concepts – NTHU LSA Lab 31

國立清華大學

Para-virtualization: Xen

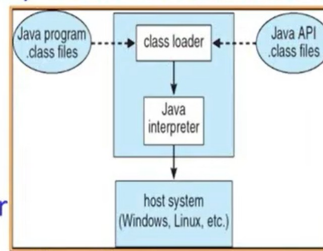
- Presents guest with system **similar but not identical** to the guest's preferred systems (**Guest must be modified**)
- **Hardware rather than OS and its devices are virtualized** (Only one kernel installed)
- Within a **container (zone)** processes thought they are the only processes on the system

- **Solaris 10**: creates a virtual layer between OS and the applications

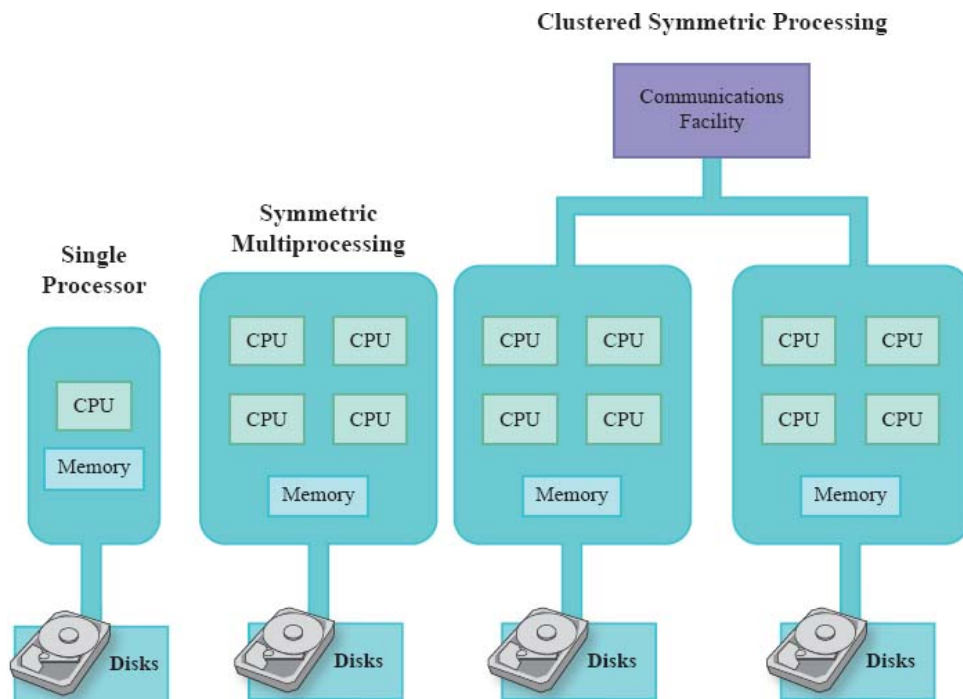
Chapter2 OS-Structure Operating System Concepts – NTHU LSA Lab 32

Java Virtual Machine

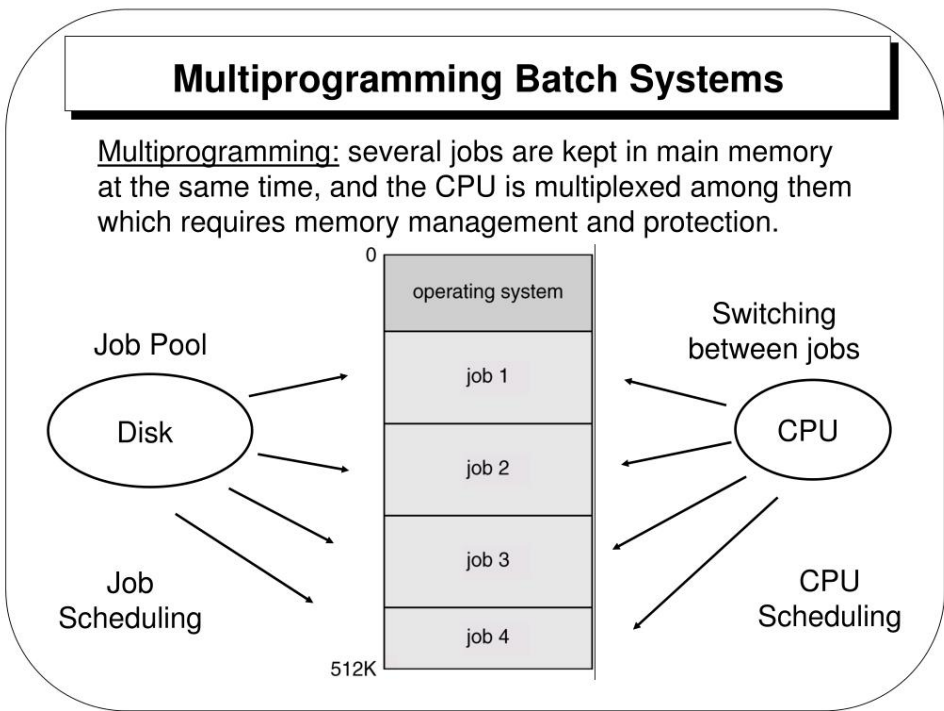
- Compiled Java programs are **platform-neutral bytecodes** executed by a **Java Virtual Machine (JVM)**
- JVM consists of
 - class loader
 - class verifier
 - runtime interpreter
- **Just-In-Time (JIT)** compilers increase performance



- SP vs MP vs Cluster



- Multiprogramming



Operating System Concepts

1.12

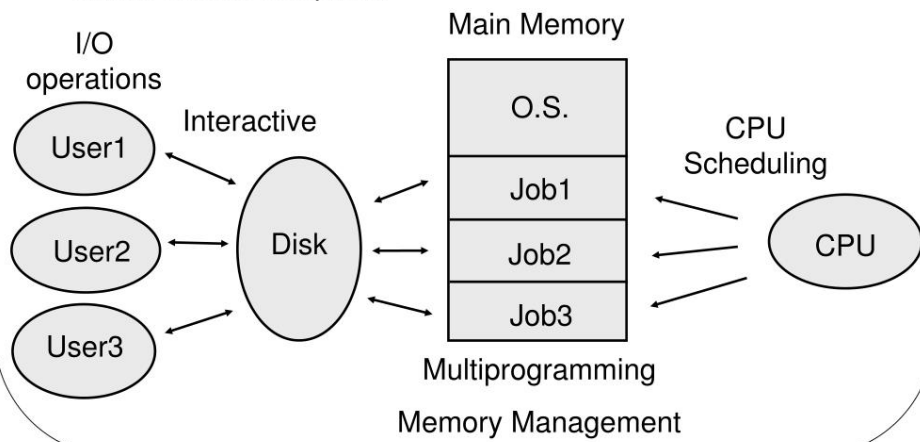
MULTIPROGRAMMING VERSUS MULTITASKING

Multiprogramming	Multitasking
In multiprogramming, multiple processes run concurrently at the same time on a single processor.	Multitasking is when more than one task is executed at a single time utilizing multiple CPUs
It is based on the concept of context switching.	It is based on the concept of time sharing.
Multiple programs reside in the main memory simultaneously to improve CPU utilization so that CPU doesn't sit idle for a long time.	It enables execution of multiple tasks and processes at the same time to increase CPU performance.
It utilizes single CPU for execution of processes.	It utilizes multiple CPUs for task allocation.
It takes more time to execute the processes.	It takes less time to execute the tasks or processes.
The idea is to reduce the CPU idle time for as long as possible.	The idea is to allow multiple processes to run simultaneously via time sharing.

- Time Sharing

Time-Sharing Systems – Interactive Computing

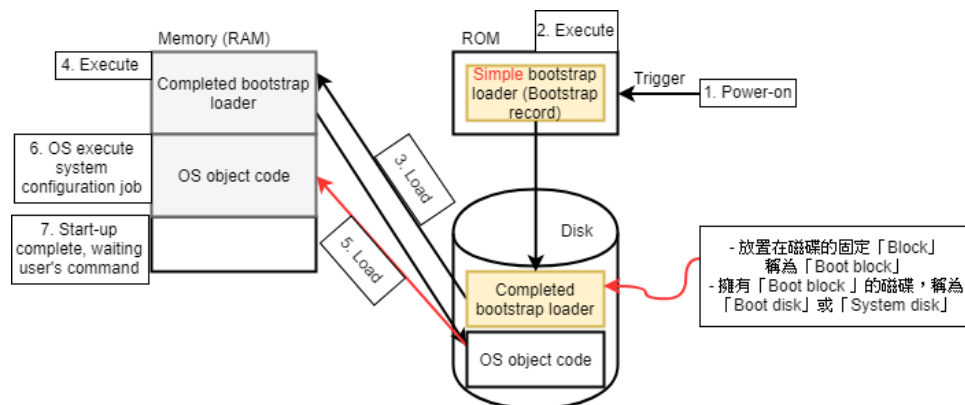
A time-sharing system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.

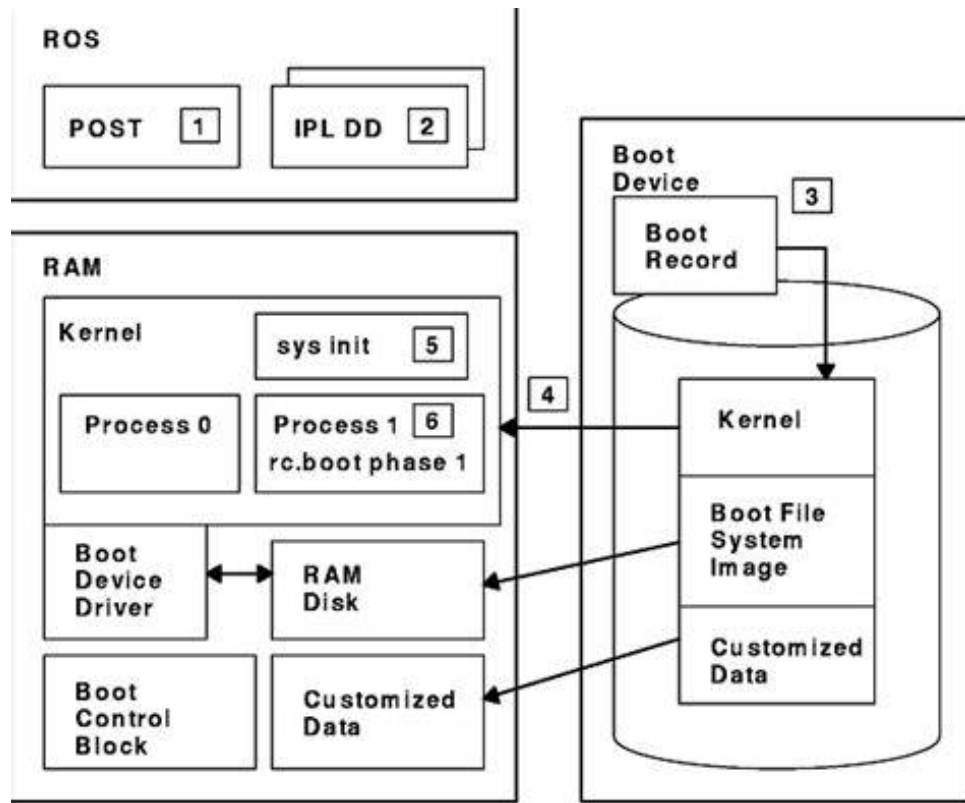


Operating System Concepts

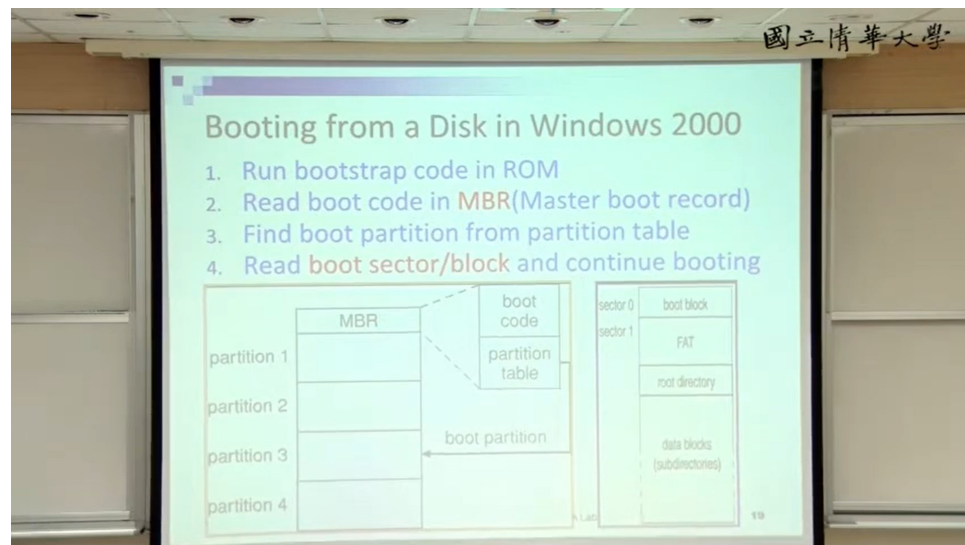
1.15

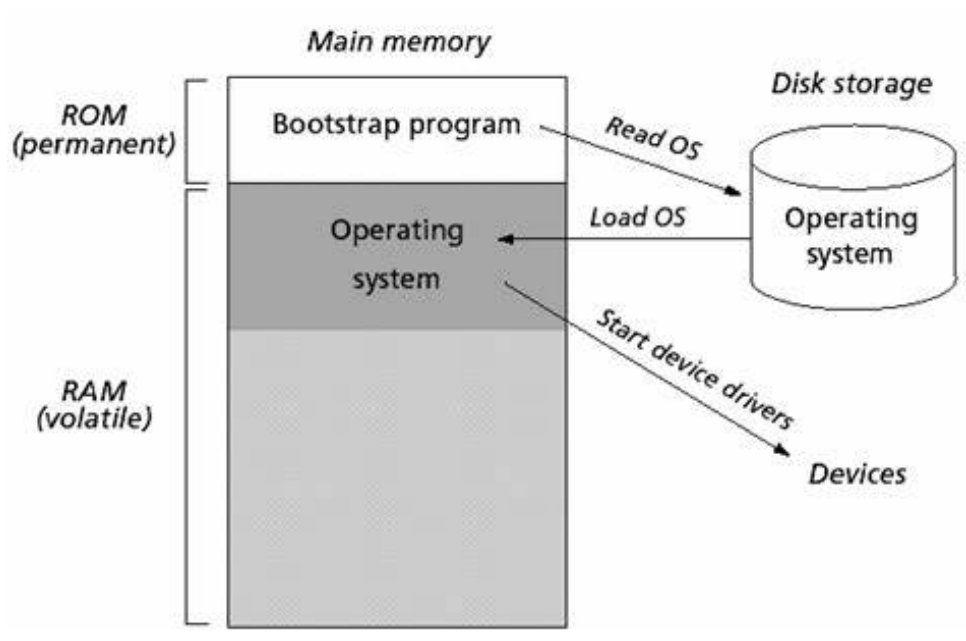
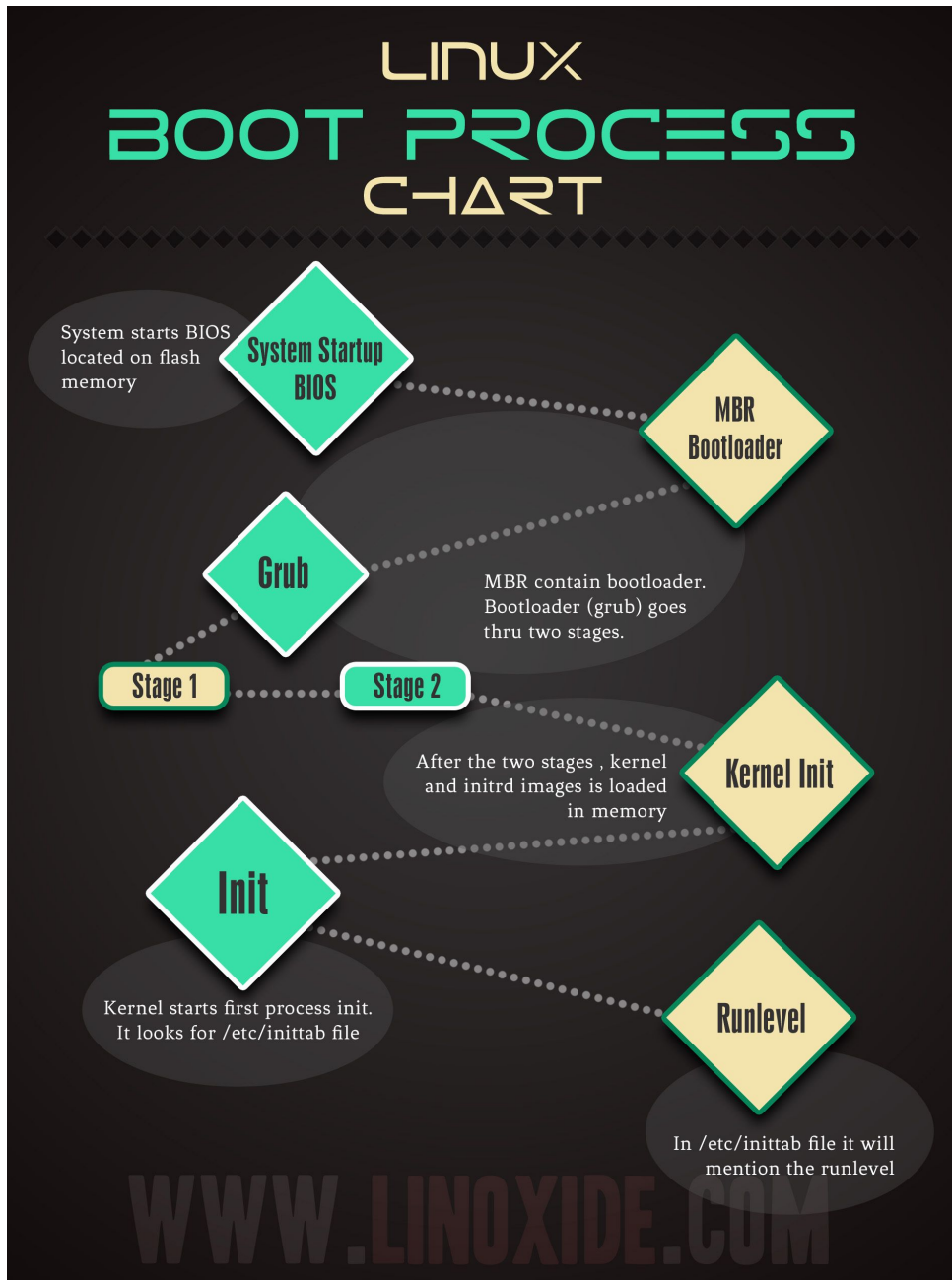
System Boot

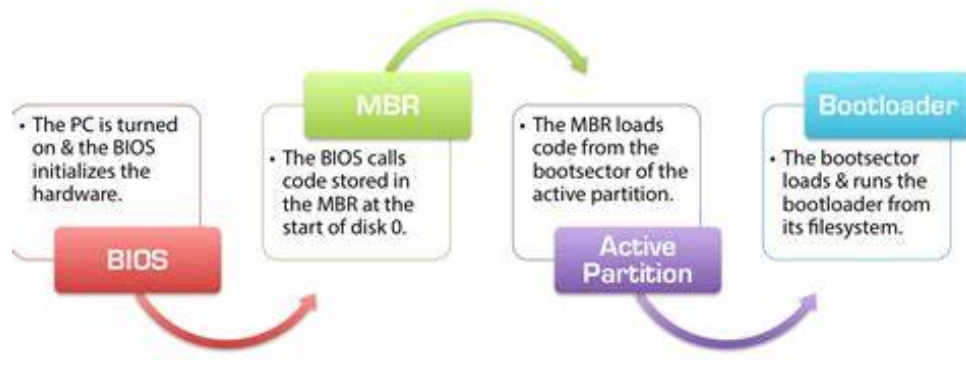




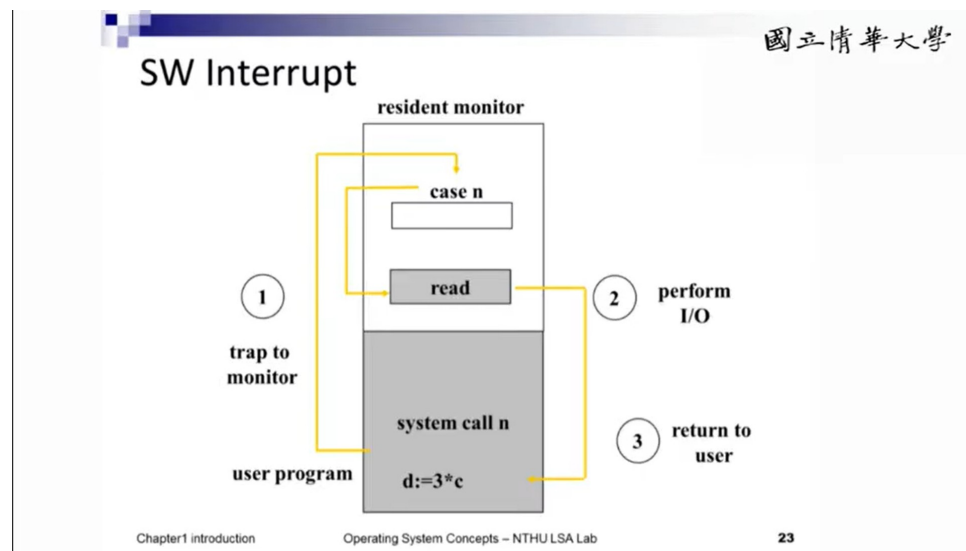
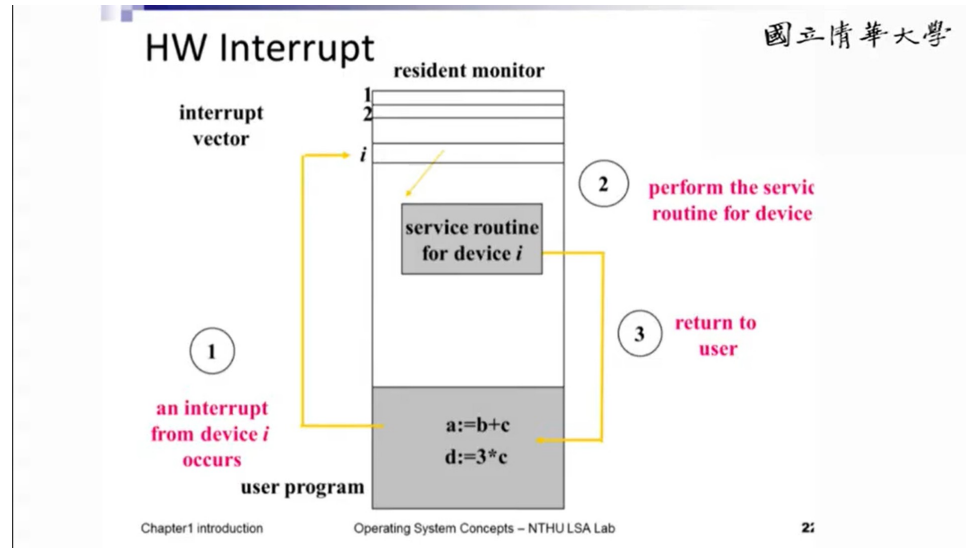
ROS Kernel Init Phase

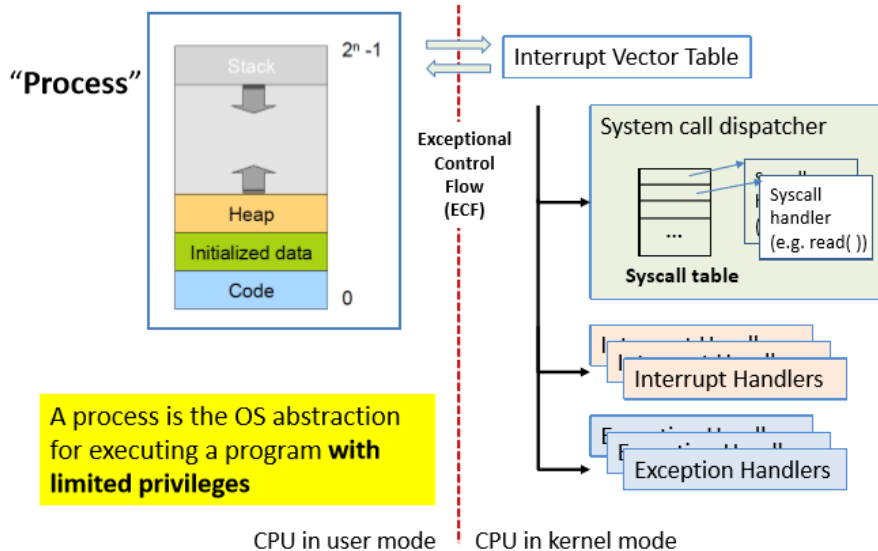






OS - Interrupt Driven

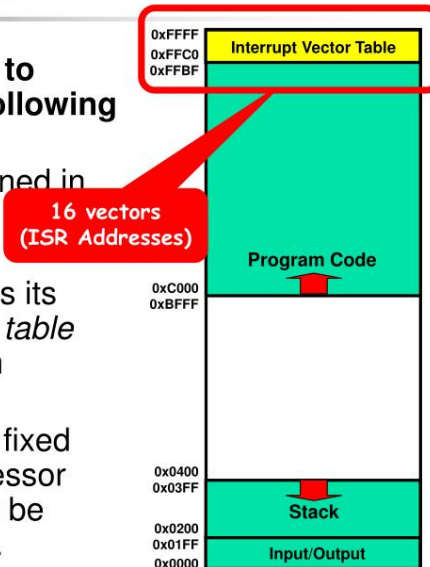




Interrupts

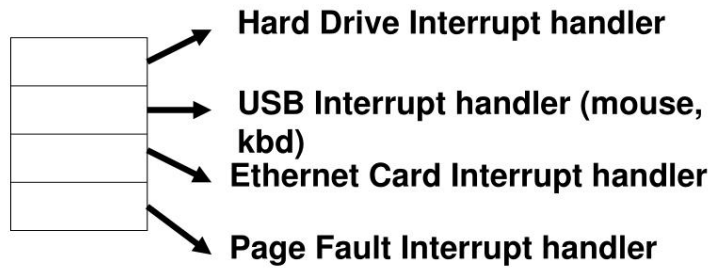
Interrupt Vectors

- The CPU must know where to fetch the next instruction following an interrupt.
- The address of an ISR is defined in an *interrupt vector*.
- The MSP430 uses *vectored interrupts* where each ISR has its own vector stored in a *vector table* located at the end of program memory.
- Note: The *vector table* is at a fixed location (defined by the processor data sheet), but the ISRs can be located anywhere in memory.

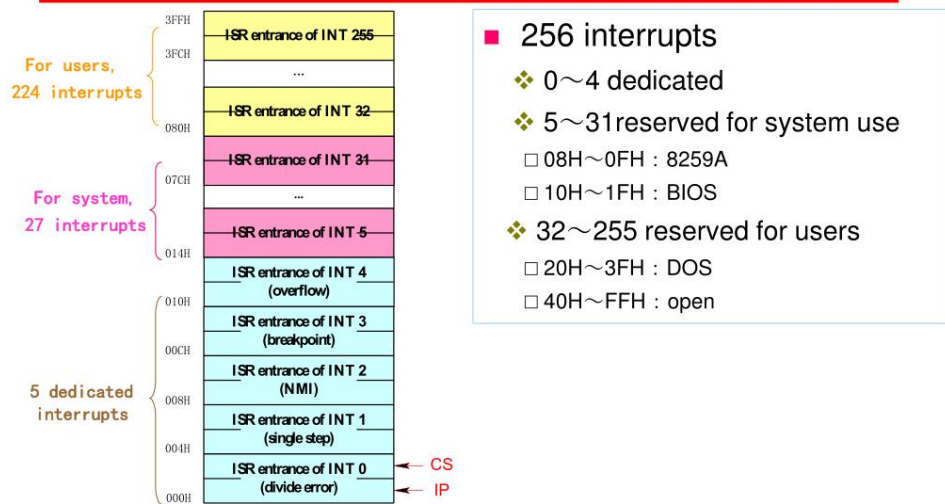


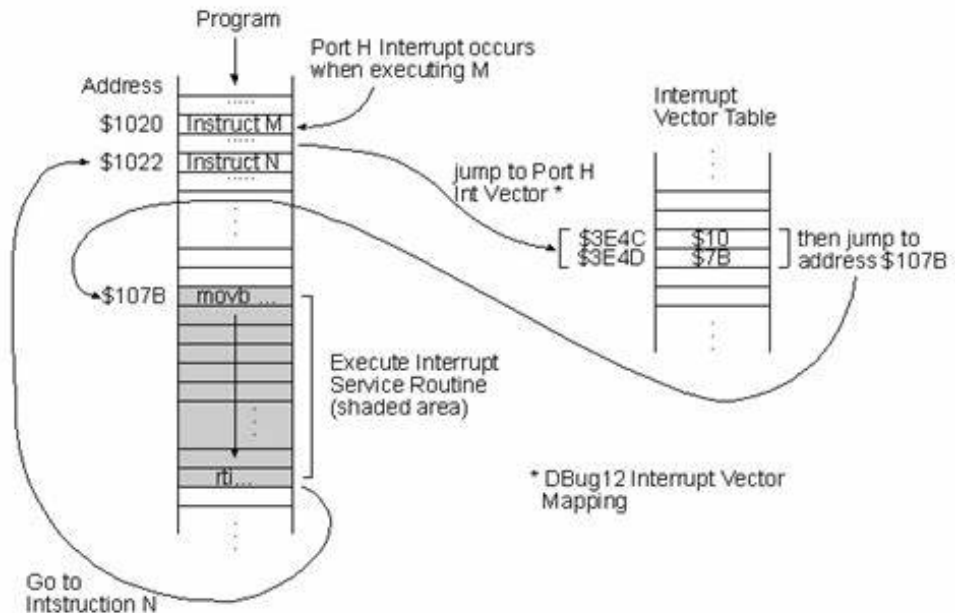
Interrupt Vector

- It is an array of pointers that point to the different interrupt handlers of the different types of interrupts.

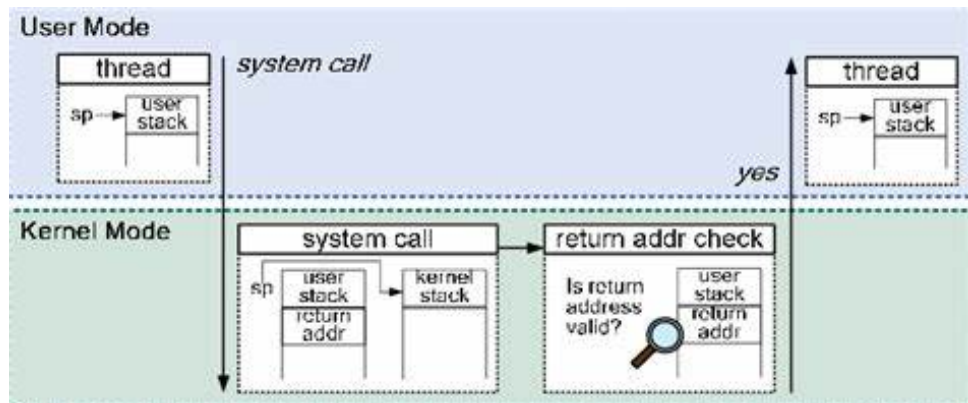
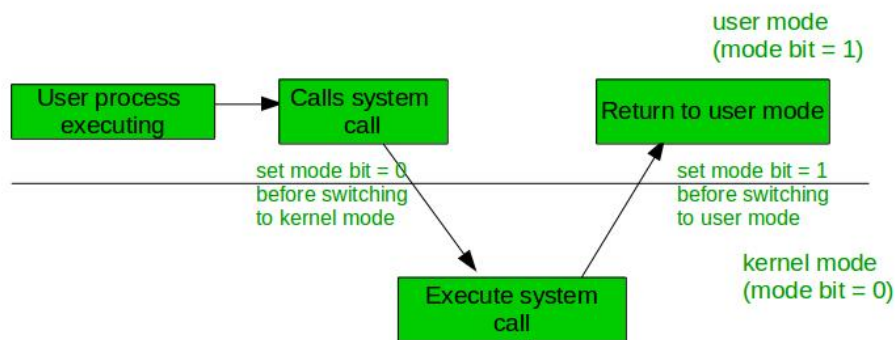


Interrupt Vector Table of 8086/8088





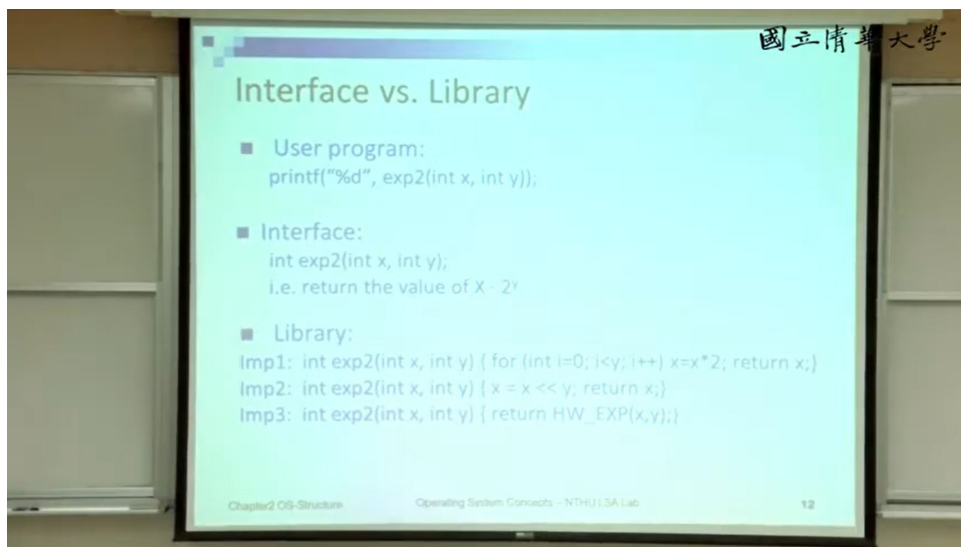
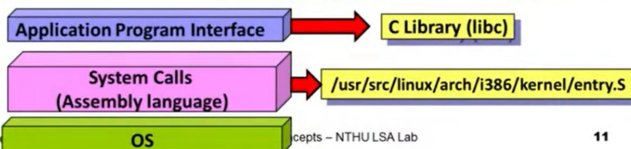
OS - Dual Mode Operation



System Call vs API(Application Programm Interface)

System Calls & API

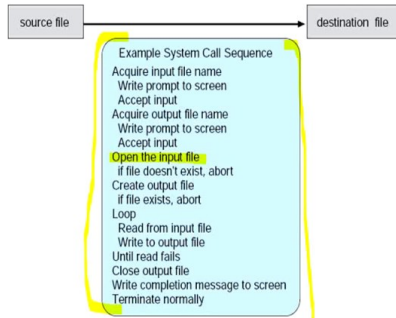
- System calls
 - > The OS interface to a running program
 - > An explicit request to the kernel made via a software interrupt
 - > Generally available as assembly-language instructions
- API: Application Program Interface
 - > Users mostly program against API instead of system call
 - > Commonly implemented by language libraries, e.g., C Library
 - > An API call could involve zero or multiple system call
 - + Both malloc() and free() use system call brk()
 - + Math API functions, such as abs(), don't need to involve system call



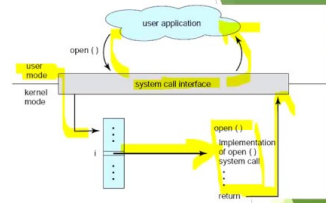
API: Application Program Interface

- Three most common APIs:
 - > Win32 API for Windows
 - + http://en.wikipedia.org/wiki/Windows_API
 - + <http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx>
 - > POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
 - + POSIX → "Portable Operating System Interface for Unix"
 - + <http://en.wikipedia.org/wiki/POSIX>
 - + http://www.unix.org/version4/GS5_APIs.pdf
 - > Java API for the Java virtual machine (JVM)

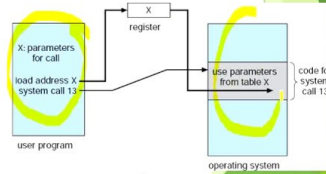
SYSTEM CALLS



Example of how system calls are used.

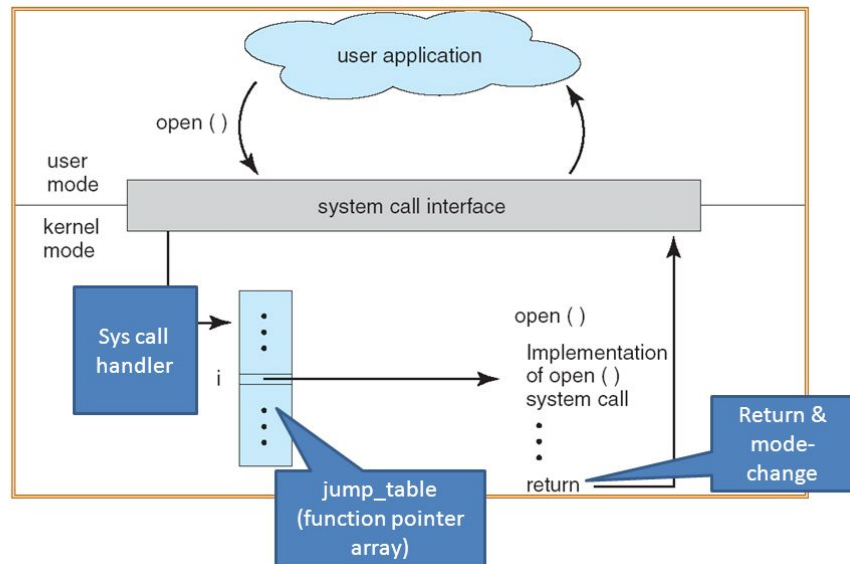


The handling of a user application invoking the open() system call



Passing of parameters as a table.

API – System Call – OS Relationship



Types of System Calls

- s Process control
 - q Load, execute, create process, wait, etc.
 - q Differs between single-tasking and multi-tasking.
- s File management
 - q Create/delete file, open/close, read/write, etc.
- s Device management
 - q Read, write, reposition, attach/detach device, etc.
- s Information maintenance.
 - q Get time/date/process/file, set time/date/process/file, etc.
- s Communications
 - q Send/receive messages, create/delete communication, etc.
 - q Two models for IPC (interprocess communication): messages-passing and shared-memory.



EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

API VERSUS SYSTEM CALL

API	SYSTEM CALL
A set of protocols, routines, functions that programmers use to develop software to facilitate interaction between distinct systems	A programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on
Helps to exchange data between various systems, devices and applications	Allows a program to access services from the kernel of the operating system
	Visit www.PEDIAA.com

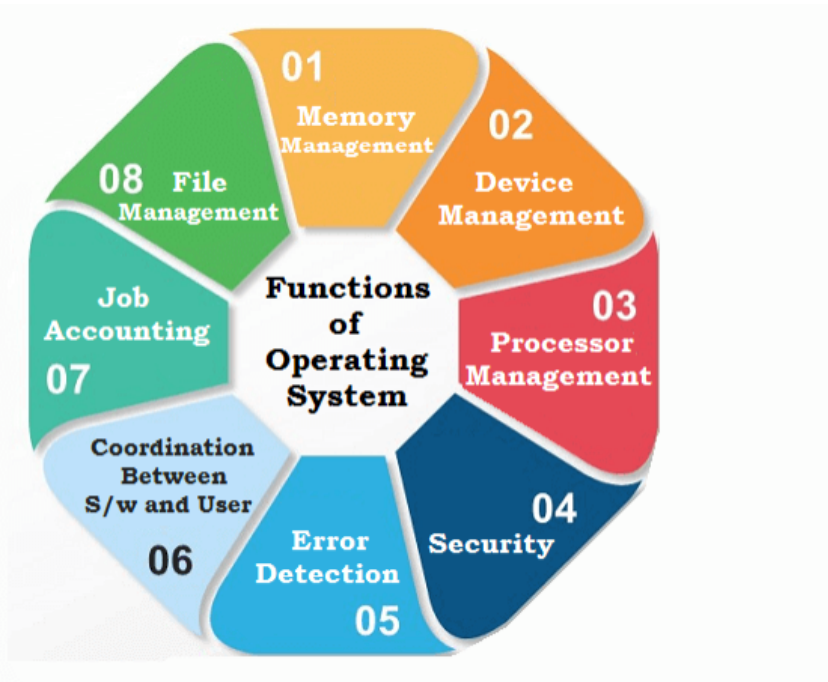
SYSTEM CALL VERSUS LIBRARY CALL

SYSTEM CALL	LIBRARY CALL
A request by the program to the kernel to enter kernel mode to access a resource	A request made by the program to access a function defined in a programming library
The mode changes from user mode to kernel mode	There is no mode switching
Not portable	Portable
Execute slower than library calls	Execute faster than system calls
System calls have more privileges than library calls	Library calls have less privileges than system calls
fork() and exec() are some examples for system calls	fopen(), fclose(), scanf() and printf() are some examples for library calls
	Visit www.PEDIAA.com

OPERATING SYSTEM VERSUS APPLICATION SOFTWARE

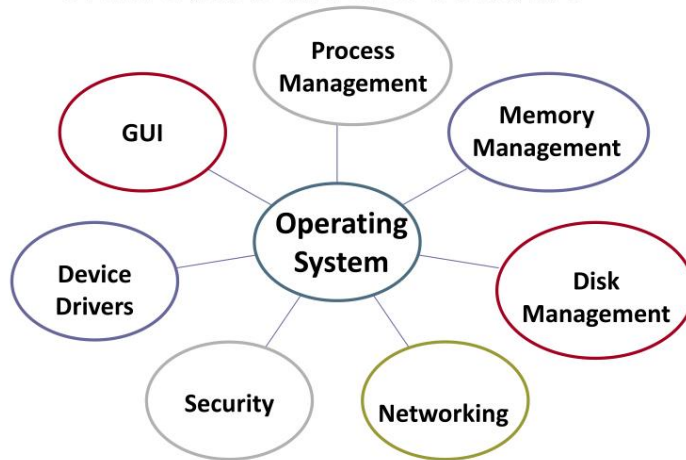
OPERATING SYSTEM	APPLICATION SOFTWARE
A system software that manages computer hardware and software resources and provides common services for computer programs	A software designed to perform a group of coordinated functions, tasks or activities for the benefit of the user
Works as the interface between the user and hardware, performs process management, memory management, task scheduling, hardware device controlling and many more	Performs a single specific task
Developed using C, C++, Assembly languages	Developed using Java, Visual Basic, C, C++
Boots up when the user switches on the computer and runs till he switches off the machine	Runs only when the user requests to run the application
Necessary for the proper functioning of the computer	Cannot be installed without an operating system
Ex: Windows, Unix, Linux, DOS	Ex: Word, Spreadsheet, Presentation, Multimedia tools, Database Management Systems
	Visit www.PEDIAA.com

Functions of OS



The Operating System

- The OS is responsible for all the functions of hardware and also software



User Interface

1. User Interface

- Command Line Interface (CLI)
 - Interaction with text/commands
- Batch Files
 - Interaction with the help of batch files (.bat)
- Graphical User Interface (GUI)
 - User friendly
 - Easier to use : pointing device, menus etc
- Hybrid UI

S

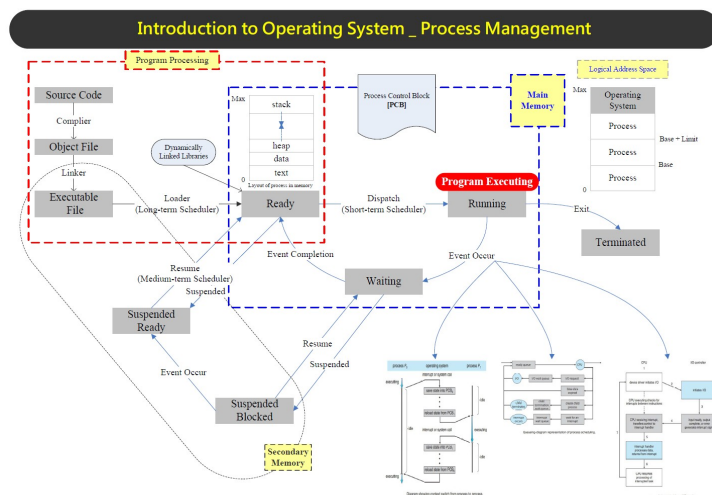


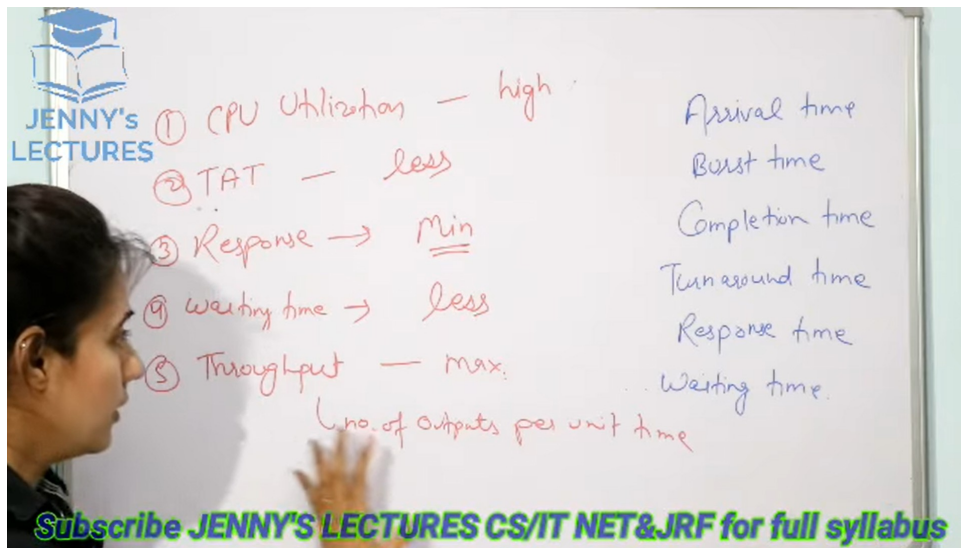
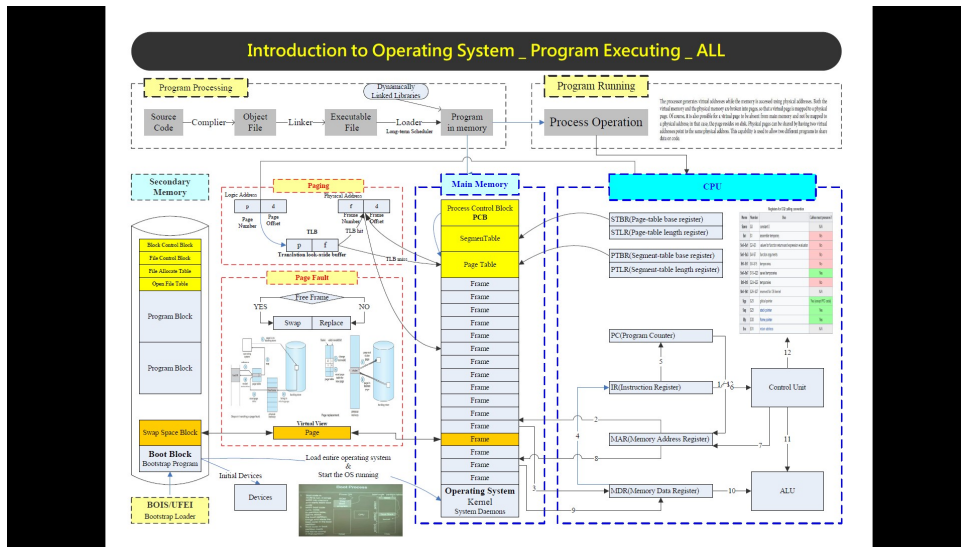
國立清華大學

User Interface

- CLI (Command Line Interface)
 - Fetches a command from user and executes it
 - **Shell: Command-line interpreter (CSHELL, BASH)**
 - ◆ Adjusted according to user behavior and preference
- GUI (Graphic User Interface)
 - Usually mouse, keyboard, and monitor
 - **Icons** represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions

Process Management



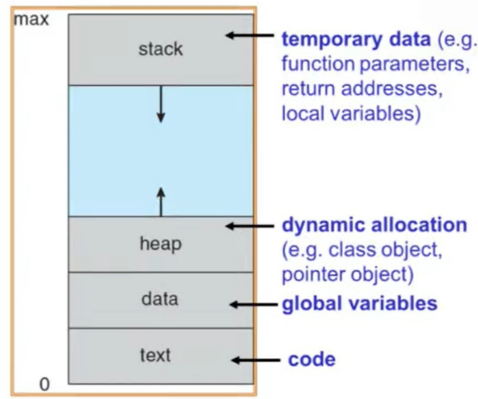


Process Concept

國立清華大學

- An operating system concurrently executes a variety of programs (e.g. Web browser, text editor, etc)
 - Program — passive entity: binary stored in disk
 - Process — active entity: a program in execution in memory
- A process includes:
 - **Code** segment (text section)
 - **Data section** — global variables
 - **Stack** — temporary local variables and functions
 - **Heap** — dynamic allocated variables or classes
 - Current activity (**program counter**, register contents)
 - A set of associated **resources** (e.g. open file handlers)

Process in Memory

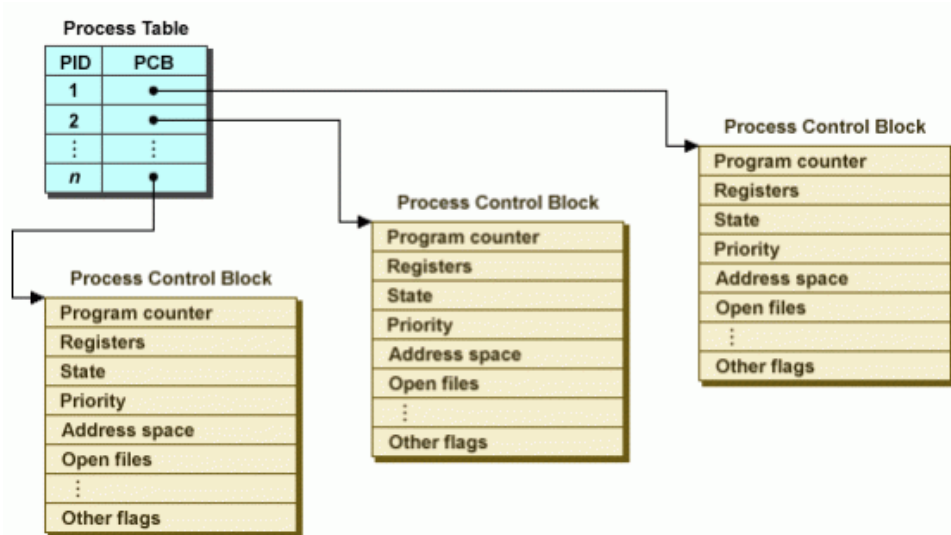


Chapter3 Processes Concept

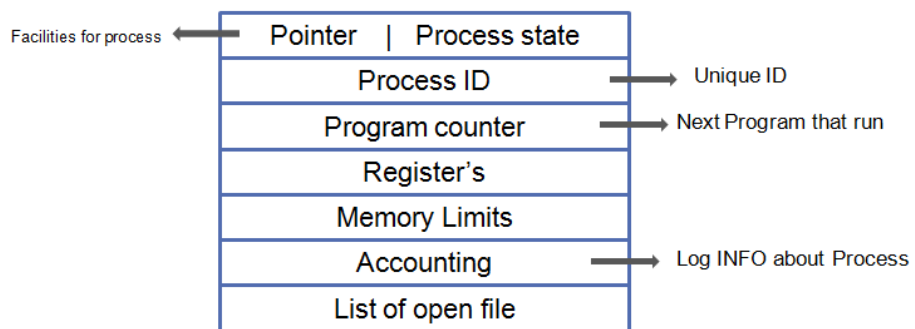
Operating System Concepts – NTHU LSA Lab

5

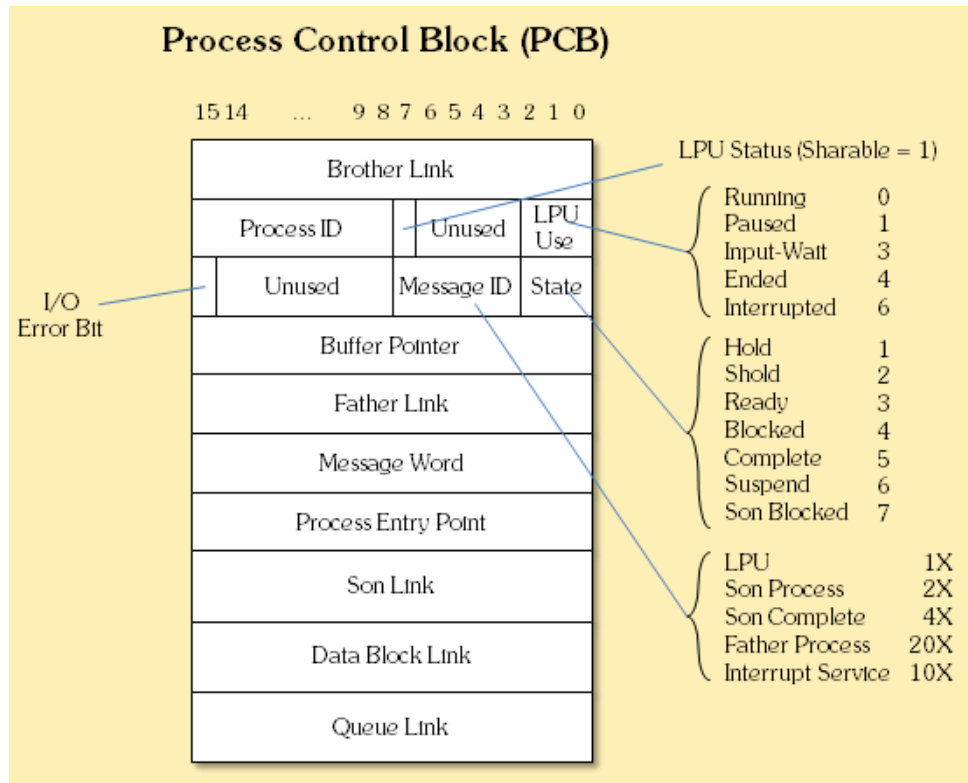
- PCB(Process Control Block)



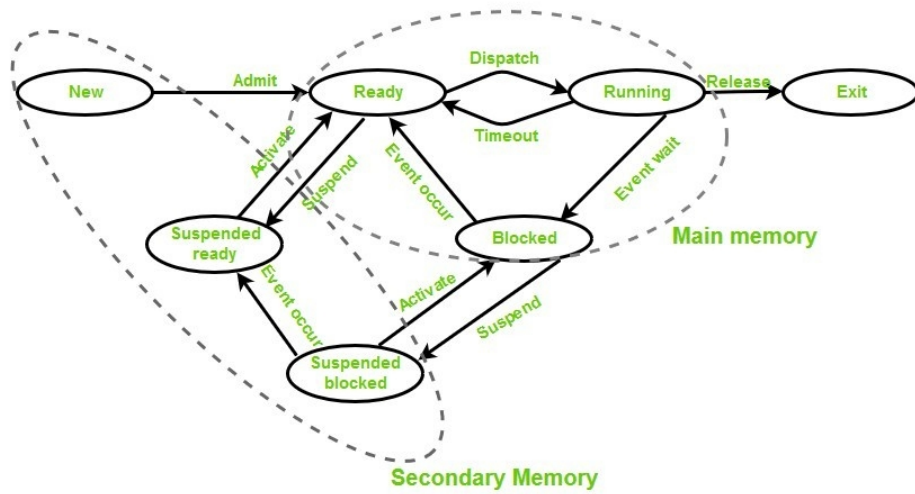
PROCESS CONTROL BLOCK (PCB)



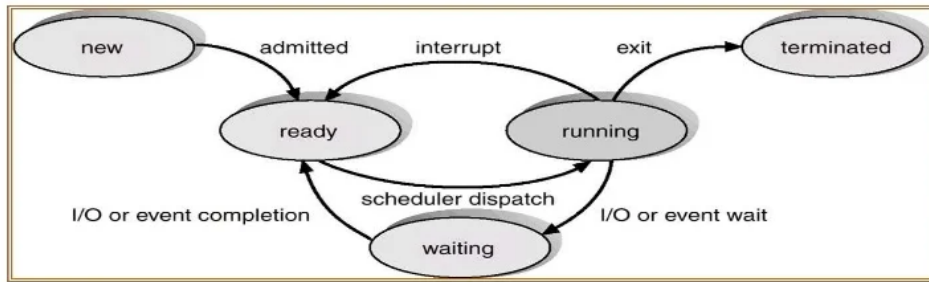
PCB Diagram



Process State Flow



Process Management

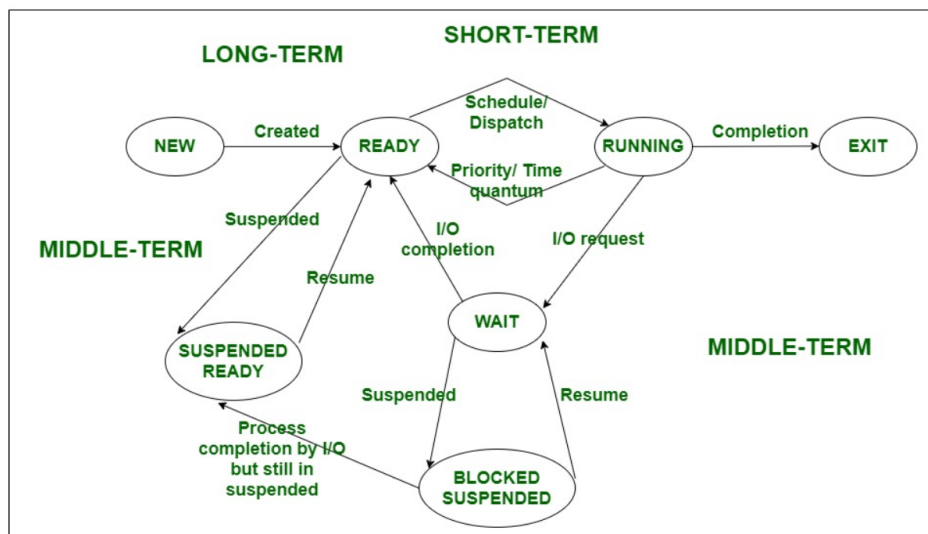


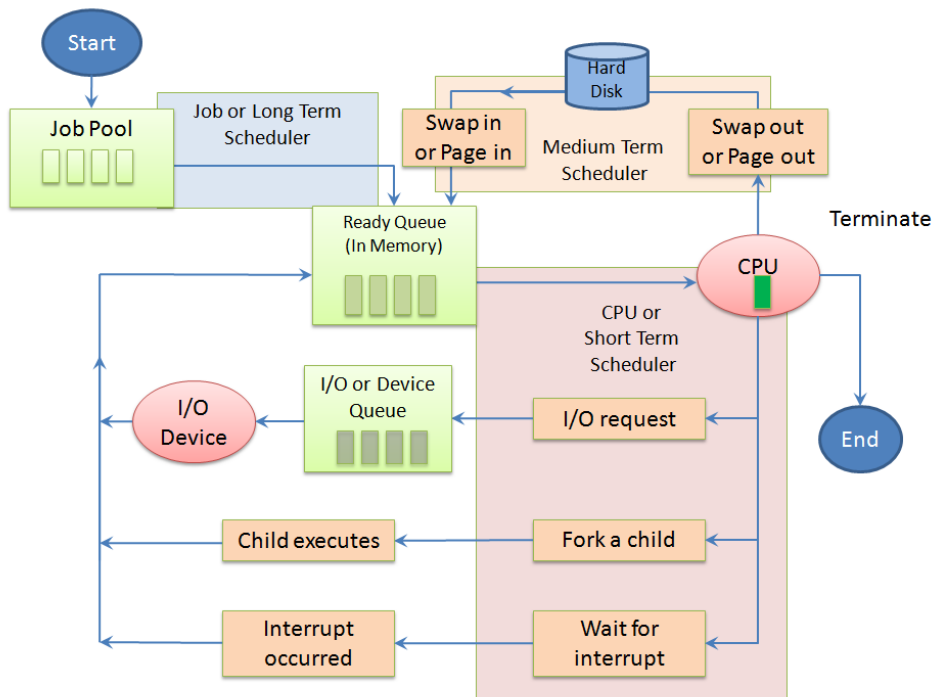
Process States

- New- The process is being created.
- Running- Instructions are being executed.
- Waiting- The process is waiting for some event to occur.
- Ready- The process is waiting to be assigned to a processor.
- Terminated- The process has finished execution.

Process Control Block- contains all information associated with a specific process like process state, program counter, CPU registers and info regarding CPU scheduling algorithms, memory, I/O and accounting.

- Scheduling

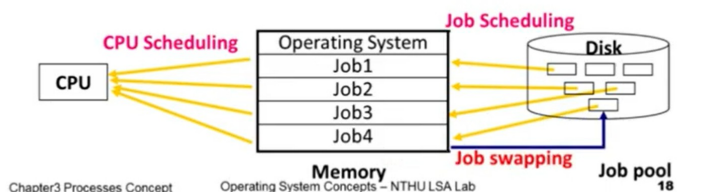




Schedulers

國立清華大學

- Short-term scheduler (CPU scheduler) – selects which process should be executed and allocated CPU (Ready state → Run state)
- Long-term scheduler (job scheduler) – selects which processes should be loaded into memory and brought into the ready queue (New state → Ready state)
- Medium-term scheduler – selects which processes should be swapped in/out memory (Ready state → Wait state)



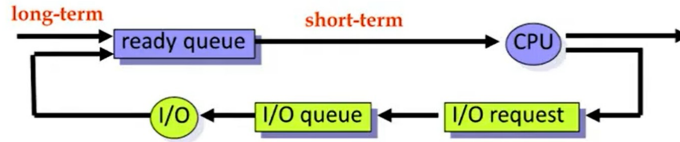
Long-Term Scheduler

國立清華大學

- Control **degree of multiprogramming**
- Execute less frequently (e.g. invoked only when a process leaves the system or **once several minutes**)
- Select a **good mix of CPU-bound & I/O-bound** processes to increase system overall performance
- UNIX/NT: no long-term scheduler
 - Created process placed in memory for short-term scheduler
 - Multiprogramming degree is bounded by hardware limitation (e.g., # of terminals) or on the self-adjusting nature of users

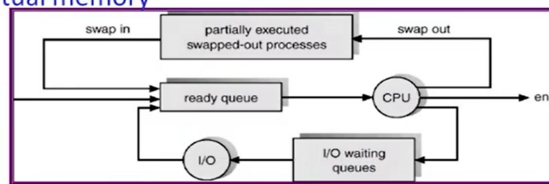
Short-Term Scheduler

- Execute quite frequently (e.g. once per 100ms)
- Must be efficient:
 - if 10 ms for picking a job, 100 ms for such a pick,
 - ➔ overhead = 10 / 110 = 9%

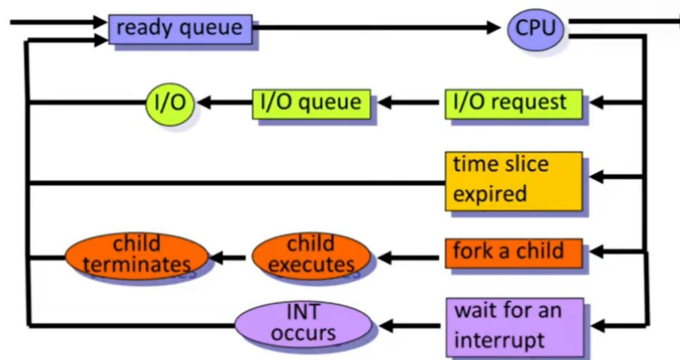


Medium-Term Scheduler

- **swap out**: removing processes from memory to reduce the degree of multiprogramming
- **swap in**: reintroducing swap-out processes into memory
- Purpose: improve process mix , free up memory
- Most modern OS doesn't have medium-term scheduler because having sufficient physical memory or using virtual memory



Process Scheduling Diagram

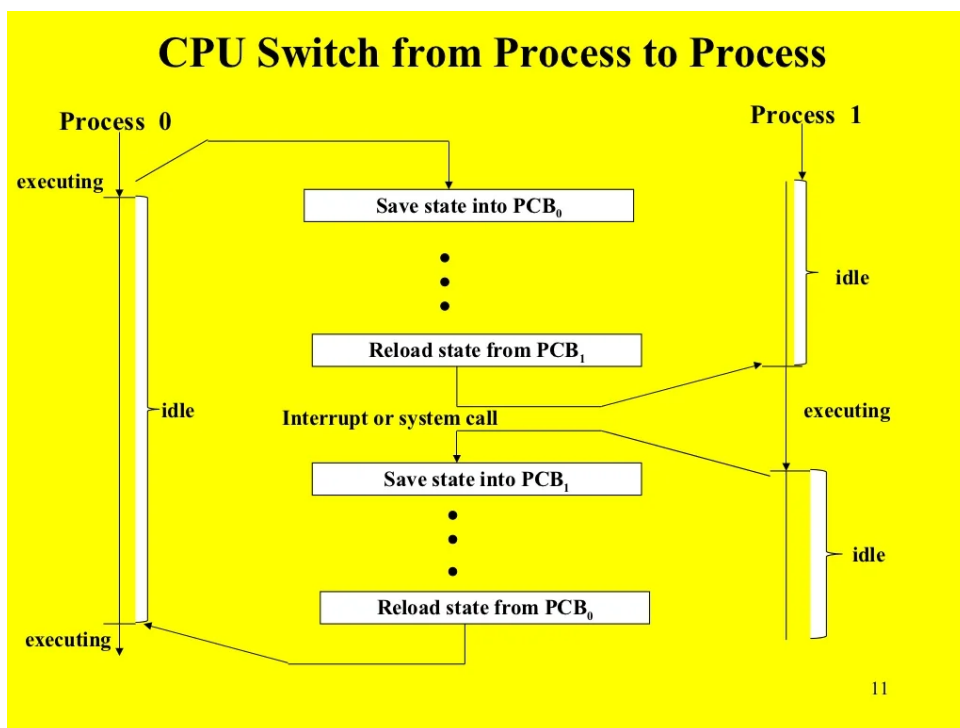


Differentiate between short term and long term scheduler.

{**Note: Any other relevant difference shall be considered**}

Sr. No	Short term scheduler	Long term scheduler
1	It is a CPU scheduler	It is a job scheduler
2	It selects processes from ready queue which are ready to execute and allocates CPU to one of them.	It selects processes from job pool and loads them into memory for execution.
3	Access ready queue and CPU.	Access job pool and ready queue
4	It executes frequently. It executes when CPU is available for allocation.	It executes much less frequently. It executes when memory has space to accommodate new process.
5	Speed is fast	Speed is less than short term scheduler
6	It provides lesser control over degree of multiprogramming	It controls the degree of multiprogramming

- Context Switch

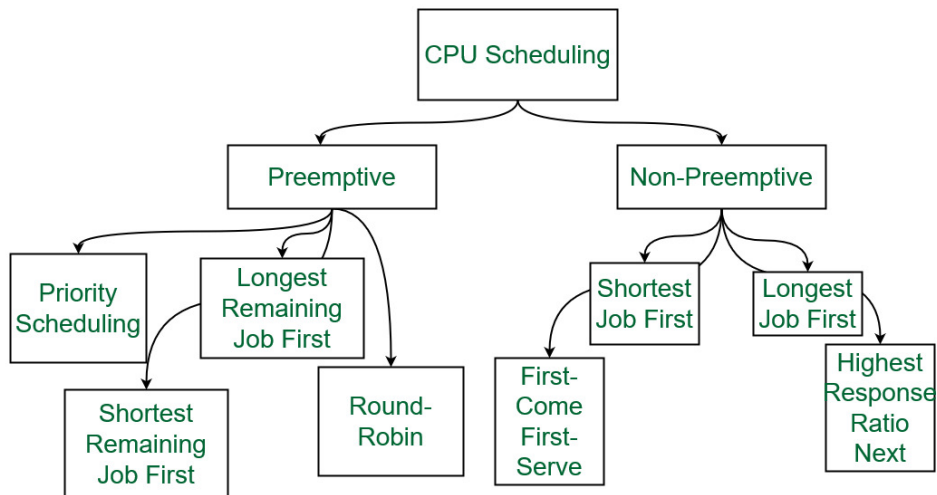
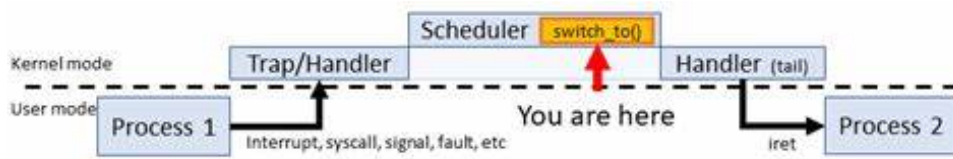
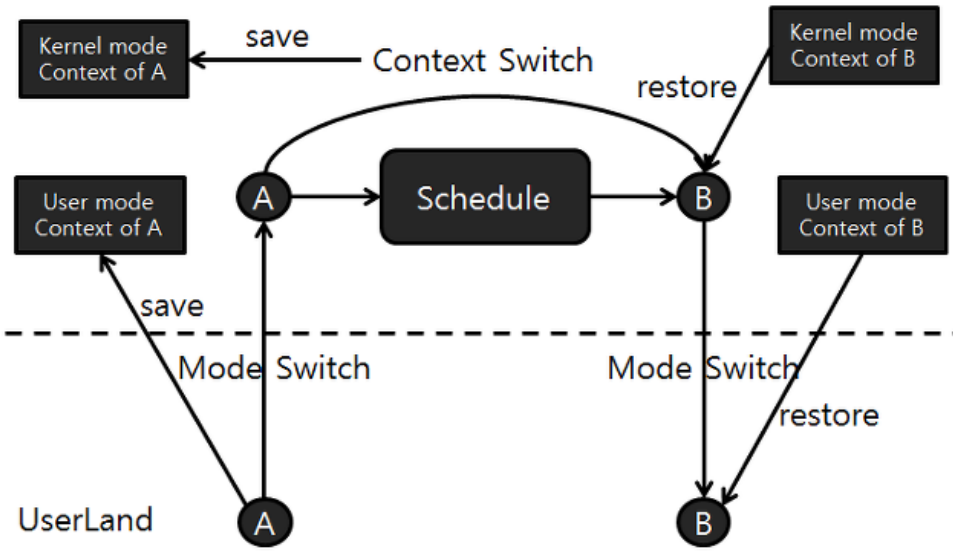


Context Switch

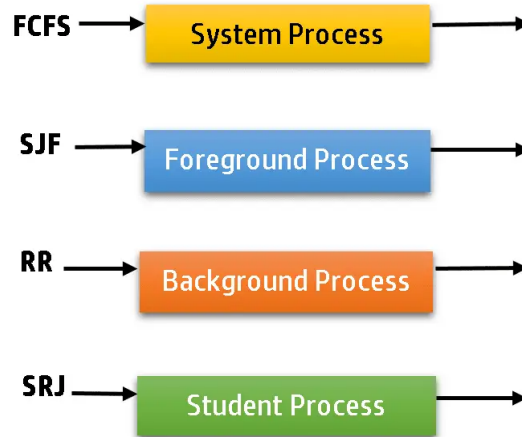
國立清華大學

- **Context Switch:** Kernel saves the state of the old process and loads the saved state for the new process
- Context-switch time is purely **overhead**
- Switch time (about 1~1000 ms) depends on
 - > memory speed
 - > number of registers
 - > existence of special instructions
 - * a single instruction to save/load all registers
 - > hardware support
 - * **multiple sets of registers** (Sun UltraSPARC – a context switch means changing register file pointer)

KernelLand



Highest Priority



Lowest Priority

Created by Notes Jam

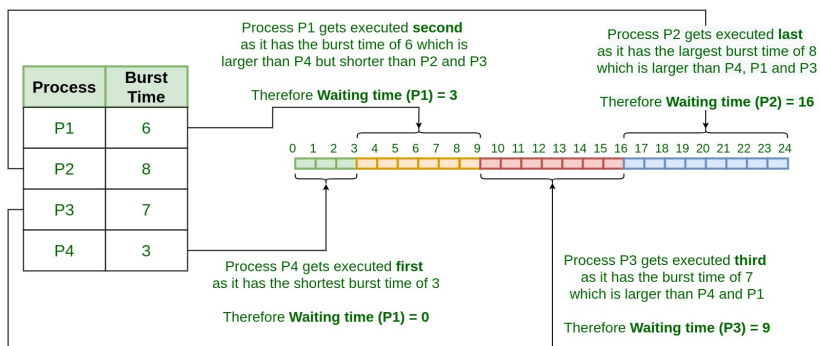
CPU SCHEDULING CRITERIA

- CPU Utilization: Percent of time that the CPU is busy executing a process.
- Throughput: Number of processes executed per unit time.
- Turnaround Time: The interval of time between submission of a process and its completion.
- Waiting Time: The amount of time the process spends in the ready queue waiting for the CPU.
- Response Time: The time between submission of requests and first response to the request.

Round Robin(RR)

- Each process is allotted a time slot(q). After this time has elapsed, the process is pre-empted and added to the end of the ready queue.
- Performance of the round robin algorithm
 - q large \rightarrow FCFS
 - q small $\rightarrow q$ must be greater than the context switch time; otherwise, the overhead is too high

Shortest Job First (SJF) Scheduling Algorithm

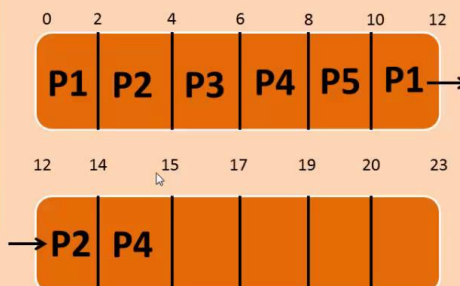


ROUND ROBIN SCHEDULING

PROCESSES	BURST TIME
P1	6 4 2
P2	5 3 1
P3	2
P4	3 1
P5	7 5

Given Time Quantum
2 units

Gantt Chart :



NOTE : Assume that all the Processes arrives at $t = 0$

CPU Scheduling (40 points)

Process	Burst Time	Priority	Arrival Time
P1	12	3	0
P2	6	4	2
P3	4	1	4
P4	18	2	6

Table 1: Process Information.

Consider the processes described in Table 1.

Questions: What is the average waiting time of those processes for each of the following scheduling algorithms? (Draw a Gantt chart for each algorithm.)

- (a) First Come First Serve (FCFS)
- (b) Non-preemptive Shortest Job First (NP-SJF)
- (c) Preemptive Shortest Job First (P-SJF)
- (d) Priority Scheduling
- (e) Round Robin, with the following assumptions:

Assumption (1). The scheduling time quantum is 5 time units.

Assumption (2). If a new process arrives at the same time as the time slice of the executing process expires, the OS puts the executing process in the ready queue, followed by the new process.

Question 3: Consider the following set of processes, their arrival times, length of the CPU burst time given in milliseconds:

Process	Arrival Time	Burst Time	Priority
P1	1	12	3
P2	2	3	1
P3	2	16	4
P4	3	10	2
P5	7	1	1

Let us schedule the execution of these processes using the following scheduling algorithms: First Come First Served (FCFS), Shortest Job First (SJF), Round Robin (RR), and a new type of scheduling algorithm called non-preemptive priority scheduling. The following are the assumptions:

- 1) Larger priority number implies higher priority
- 2) Assume the quantum (aka time slice) of 2 for RR algorithm
- 3) Non-Preemptive means once scheduled, a process cannot be preempted (i.e. it runs to completion).

3a [10 points] What is the response (completion) time of each process for each of the scheduling algorithms. Write the times in the table below.

	FCFS	SJF	Priority	RR
P1				
P2				
P3				
P4				
P5				

3b [10 points] Which of the above methods results in the longest average wait time. Show calculations and explain.

國立清華大學

Approximate Shortest-Job-First (SJF)

- SJF difficulty: no way to know length of the next CPU burst
- **Approximate SJF**: the next burst can be predicted as an **exponential average** of the measured length of previous CPU bursts

$$\tau_{n+1} = \alpha t_n + (1-\alpha) \tau_n \leftarrow \text{history}$$

new one

Commonly,

$$= \alpha t_n + (1-\alpha) \alpha t_{n-1} + (1-\alpha)^2 \alpha t_{n-2} + \dots$$

$$\alpha = 1/2 \rightarrow = \left(\frac{1}{2}\right) t_n + \left(\frac{1}{2}\right)^2 t_{n-1} + \left(\frac{1}{2}\right)^3 t_{n-2} + \dots$$

Chapter5 Process Scheduling Operating System Concepts – NTHU LSA Lab 27

國立清華大學

Priority Scheduling

- A priority number is associated with each process
- The CPU is allocated to the highest priority process
 - Preemptive
 - Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem: starvation (low priority processes never execute)
 - e.g. IBM 7094 shutdown at 1973, a 1967-process never run
- **Solution: aging** (as time progresses increase the priority of processes)
 - e.g. increase priority by 1 every 15 minutes

Chapter5 Process Scheduling Operating System Concepts – NTHU LSA Lab 29

國立清華大學

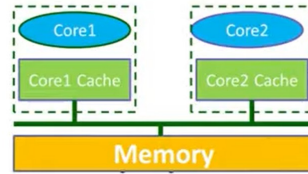
Evaluation Methods

- **Deterministic modeling** – takes a particular predetermined workload and defines the performance of each algorithm for that workload
 - Cannot be generalized
- **Queueing model** – mathematical analysis
- **Simulation** – random-number generator or trace tapes for workload generation
- **Implementation** – the only completely accurate for algorithm evaluation

Chapter5 Process Scheduling Operating System Concepts – NTHU LSA Lab

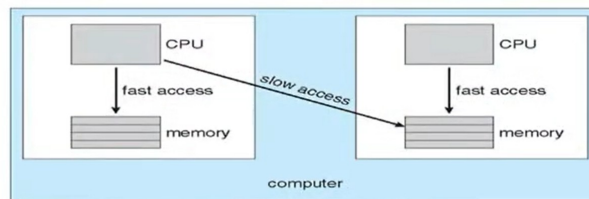
Processor affinity

- **Processor affinity:** a process has an affinity for the processor on which it is currently running
 - A process populates its recent used **data in cache memory of its running processor**
 - **Cache invalidation and repopulation has high cost**
- **Solution**
 - **soft affinity:**
 - ✦ possible to migrate between processors
 - **hard affinity:**
 - ✦ not to migrate to other processor



NUMA and CPU Scheduling

- **NUMA (non-uniform memory access):**
 - Occurs in systems containing combined CPU and memory boards
 - **CPU scheduler and memory-placement works together**
 - A process (assigned affinity to a CPU) can be allocated memory on the board where that CPU resides



Load-balancing

- Keep the workload evenly distributed across all processors
 - Only necessary on systems where each processor has its own private queue of eligible processes to execute
- Two strategies:
 - **Push migration:** move (push) processes from overloaded to idle or less-busy processor
 - **Pull migration:** idle processor pulls a waiting task from a busy processor
 - Often implemented in parallel
- Load balancing often counteracts the benefits of processor affinity

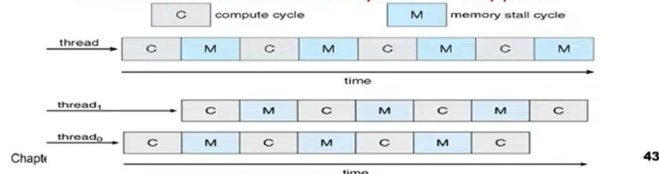
Multi-core Processor Scheduling

Multi-core Processor:

- Faster and consume less power
- **memory stall**: When access memory, it spends a significant amount of time waiting for the data become available. (e.g. cache miss)

Multi-threaded multi-core systems:

- Two (or more) hardware threads are assigned to each core (i.e. **Intel Hyper-threading**)
- Takes advantage of memory stall to make progress on another thread while memory retrieve happens



Chapt

43

Real-Time Scheduling

Real-time does not mean speed, but *keeping deadlines*

Soft real-time requirements:

- Missing the deadline is unwanted, but is not immediately critical
- Examples: multimedia streaming

Hard real-time requirements:

- Missing the deadline results in *a fundamental failure*
- Examples: nuclear power plant controller

Chapter5 Process Scheduling

Operating System Concepts – NTHU LSA Lab

45

Real-Time Scheduling Algorithms

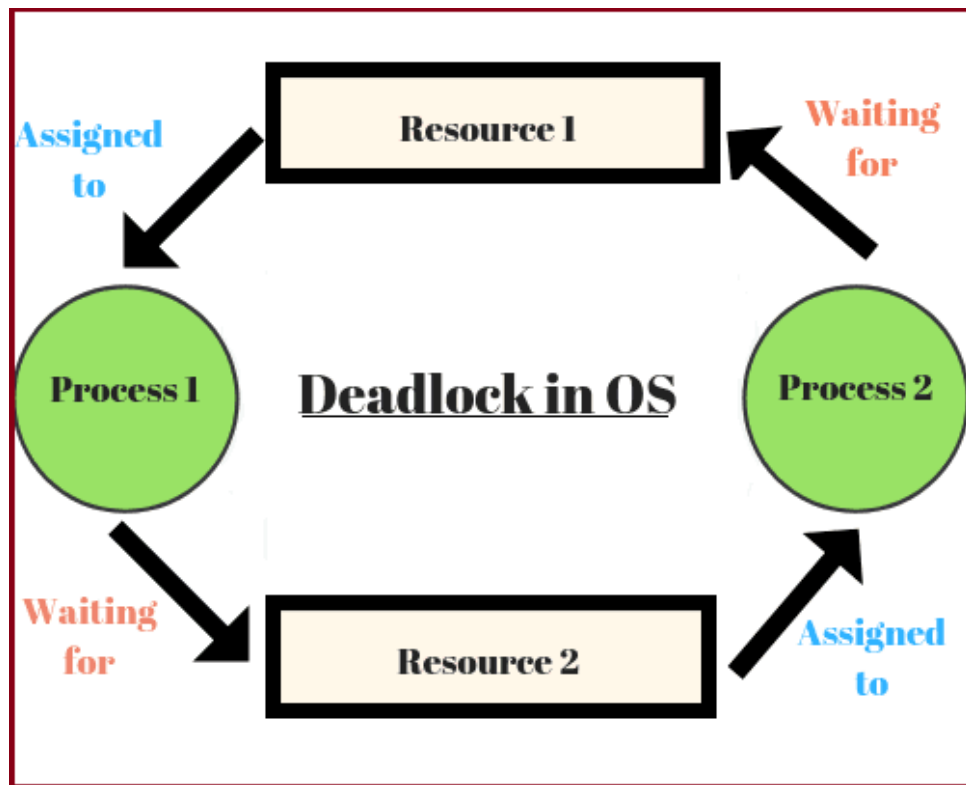
- FCFS scheduling algorithm – Non-RTS
 - $T1 = (0, 4, 10) == (\text{Ready}, \text{Execution}, \text{Period})$
 - $T2 = (1, 2, 4)$
- Rate-Monotonic (RM) algorithm
 - Shorter period \rightarrow higher priority
 - Fixed-priority RTS scheduling algorithm
- Earliest-Deadline-First (EDF) algorithm
 - Earlier deadline \rightarrow higher priority
 - Dynamic priority algorithm

Chapter5 Process Scheduling

Operating System Concepts – NTHU LSA Lab

46

-Deadlock



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .

7.5

- Resource Allocation Graph

Resource-Allocation Graph: Definition

A set of vertices V and a set of edges E .

- V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

The resource-allocation graph is therefore a bipartite directed graph. What would be the graph representing the bridge-crossing example?



Resource Allocation Graph

Set of Vertices
 $V \rightarrow$ Set of Processes $\{P_1, P_2, P_3, \dots, P_n\}$
 $V \rightarrow$ Set of Resources $\{R_1, R_2, R_3, \dots, R_m\}$

Set of Edges
 $E \rightarrow P_i \rightarrow R_j$ (Request Edge)
 $E \rightarrow R_j \rightarrow P_i$ (Assignment Edge)

□ \Rightarrow Resource type
 ○ \Rightarrow Process
 □ with dots \Rightarrow instance of resource type

Subscribe JENNY'S LECTURES CS/IT NET&JRF for full syllabus




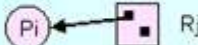
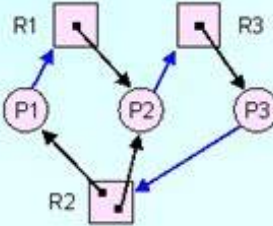
If no cycle in RAG, then no process in the system is deadlocked & if it contains cycle then deadlock may exist

cycle + one instance of resource type \Downarrow Deadlock
 cycle + more than one instance \Downarrow Deadlock may exist

	Allocation			Request			Availability		
	P_1	P_2	P_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	0	1	1	0	0	0	0	0
P_2	1	0	1	0	1	0	0	0	0
P_3	0	1	0	0	0	1	0	0	1

	Allocation		Request		Availability	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	0	1	1	0	0	0
P_2	1	0	0	0	0	0
P_3	1	0	0	1	0	0

Resource Allocation Graph

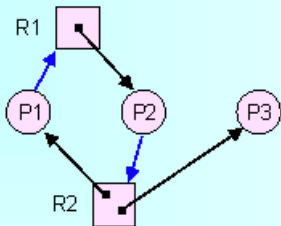
- Process 
- Resource type with 2 instances 
- P_i requests instance of R_j 
- An instance of R_j is allocated to P_i 
- A graph with a deadlock: 

Yair Amir

Fall 00 / Lecture 4

8

Resource Allocation Graph (cont.)

- A graph with a cycle but without a deadlock: 

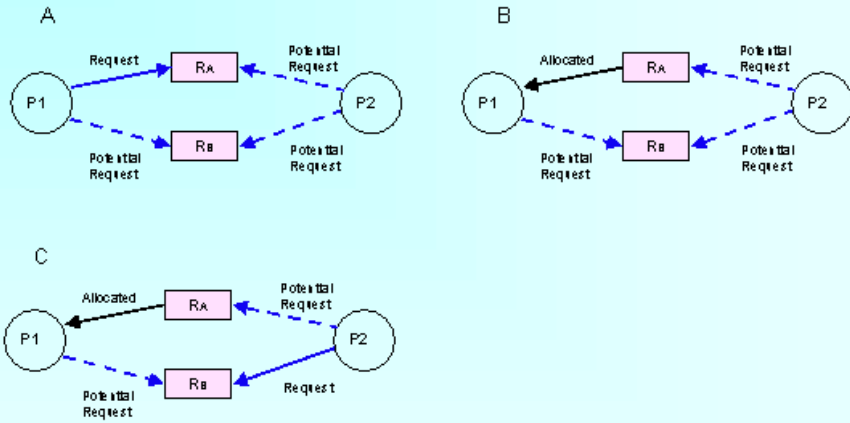
- If there are no cycles then there is no deadlock.
- If there is a cycle:
 - **If there is only one instance per resource type then there is a deadlock.**
 - **If there is more than one instance for some resource type, there may or may not be a deadlock.**

Yair Amir

Fall 00 / Lecture 4

9

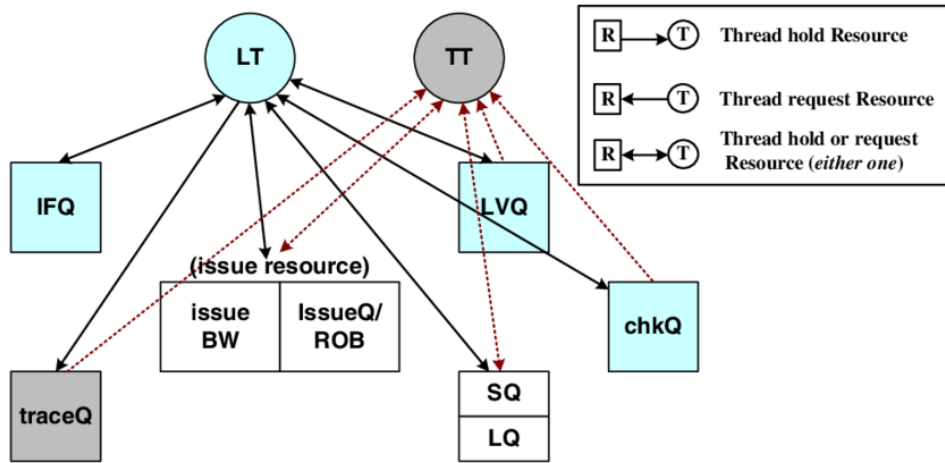
Resource Allocation Graph Algorithm (cont.)



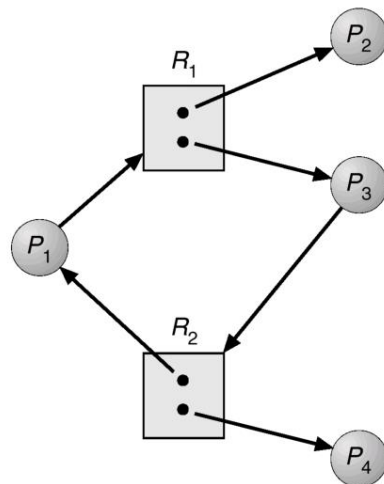
Yair Amir

Fall 00 / Lecture 4

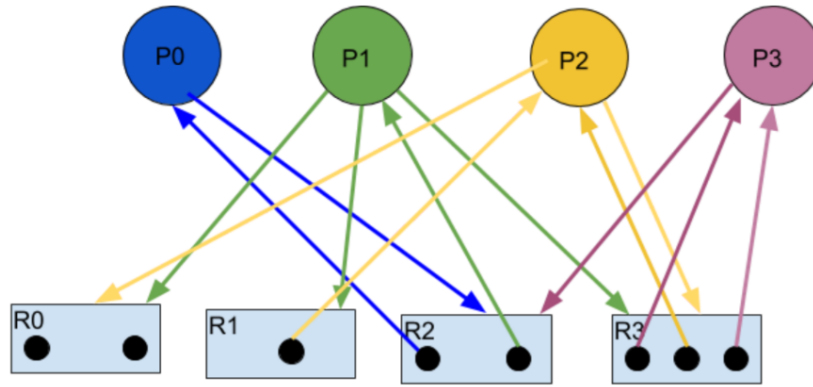
20



Resource Allocation Graph With A Cycle But No Deadlock



- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then deadlock.
 - if several instances per resource type, possibility of deadlock.



[6 pts] Fill in the resource matrix according to the graph above:

	Allocation				Need				Available			
	R0	R1	R2	R3	R0	R1	R2	R3	R0	R1	R2	R3
P0												
P1												
P2												
P3												

Approaches to Deadlock Prevention

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Figure 6-14. Summary of approaches to deadlock prevention.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639

Deadlock Avoidance

- Deadlock avoidance is a technique used to avoid deadlock.
- It requires information about how different processes would request different resources.
- **Safe state:** if deadlock not occur then safe state.
- **Unsafe state:** if deadlock occur then unsafe state.
- The idea of avoiding a deadlock is simply not allow the system to enter an unsafe state the may cause a deadlock.



3.5 Deadlock Detection

- If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.
- In this environment, the system may provide:
 - An algorithm that examines the state of the system to determine whether a deadlock has occurred.
 - An algorithm to recover from the deadlock.
- Single Instance of Each Resource Type.
- Several Instances of a Resource Type.
- When should we invoke the detection algorithm?
- The answer depends on two factors:
 1. How *often* is a deadlock likely to occur?
 2. How *many* processes will be affected by deadlock when it happens?

20

The Difference Between Deadlock Prevention and Deadlock Avoidance

• Deadlock Prevention:



- Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design).
- The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.

• Deadlock Avoidance:



- The system dynamically considers every request and decides whether it is safe to grant it at this point.
- The system requires additional a priori information regarding the overall potential use of each resource for each process.
- Allows more concurrency.

Similar to the difference between a traffic light and a police officer directing traffic.

Yair Amir

Fall 00 / Lecture 4

11

Comparison Deadlock	Starvation
<ul style="list-style-type: none"> • Definition • Other name • Arising conditions • Avoidance/prevention Techniques 	<ul style="list-style-type: none"> • Starvation occurs when a process waits for an indefinite period of time to get the resource it requires. Or Starvation is where low priority processes get blocked, and high priority process proceeds. • Lived lock • Uncontrolled management of resources, Process priorities being strictly enforces Use of random selection, Scarcity of resources
<ul style="list-style-type: none"> • Deadlock occurs when none of the processes in the set is able to move ahead due to occupancy of the required resources by some other process. Or Deadlock is where no process proceeds, and get blocked • Circular waiting • These four conditions arising simultaneously – mutual exclusion, hold and wait, no-preemption and circular wait • Infinite resources, Waiting is not allowed, Sharing is not allowed, Preempt the resources, All Requests made at the starting 	

BetweenMates.com

Handling Deadlocks

國立清華大學

- Ensure the system will **never** enter a **deadlock state**
 - **deadlock prevention**: ensure that at least one of the 4 **necessary conditions** cannot hold
 - **deadlock avoidance**: **dynamically** examines the resource-allocation state before allocation
- Allow to **enter a deadlock state** and **then recover**
 - **deadlock detection**
 - **deadlock recovery**
- **ignore the problem** and pretend that deadlocks never occur in the system
 - used by most operating systems, including UNIX.

Deadlock Recovery

國立清華大學

- Process termination
 - abort all deadlocked processes
 - abort 1 process at a time until the deadlock cycle is eliminated
 - ✦ which process should we abort first?
- Resource preemption
 - select a victim: which one to preempt?
 - rollback: partial rollback or total rollback?
 - starvation: can the same process be preempted always?

Bankers' Algorithm

The banker's algorithm

- A state is safe iff there exists a sequence $\{P_1..P_n\}$ where each P_i is allocated all of its needed resources to be run to completion
 - ◆ i.e.: we can always run all the processes to completion from a safe state
- The **safety algorithm** is the part that determines if a state is safe
- Initialization:
 - ◆ all processes are said to be “unfinished”
 - ◆ set the work vector to the amount resources available: $W(i) = V(i)$ for all i ;

41

Banker's Algorithm

1. Look for a new row in R which is smaller than A . If no such row exists the system will eventually deadlock → not safe.
2. If such a row exists, the process may finish. mark that process (row) as terminate and add all of its resources to A .
3. Repeat Steps 1 and 2 until all rows are marked → safe state
If some are not marked → not safe.

30

Example of Banker's Algorithm

	Allocation A B C
P0	0 1 0
P1	2 0 0
P2	3 0 2
P3	2 1 1
P4	0 0 2

	Need A B C
P0	7 4 3
P1	1 2 2
P2	6 0 0
P3	0 1 1
P4	4 3 1

Available A B C
3 3 2

Try to find a row in $Need_i$ that is \leq Available.

- P1. run completion. Available becomes = $[3\ 3\ 2] + [2\ 0\ 0] = [5\ 3\ 2]$
- P3. run completion. Available becomes = $[5\ 3\ 2] + [2\ 1\ 1] = [7\ 4\ 3]$
- P4. run completion. Available becomes = $[7\ 4\ 3] + [0\ 0\ 2] = [7\ 4\ 5]$
- P2. run completion. Available becomes = $[7\ 4\ 5] + [3\ 0\ 2] = [10\ 4\ 7]$
- P0. run completion. Available becomes = $[10\ 4\ 7] + [0\ 1\ 0] = [10\ 5\ 7]$

We found a sequence of execution: P1, P3, P4, P2, P0. State is safe

40

Banker's Algorithm for a single resource

Has Max	Has Max	Has Max																																				
<table border="1" style="margin: auto;"> <tr><td>A</td><td>0</td><td>6</td></tr> <tr><td>B</td><td>0</td><td>5</td></tr> <tr><td>C</td><td>0</td><td>4</td></tr> <tr><td>D</td><td>0</td><td>7</td></tr> </table>	A	0	6	B	0	5	C	0	4	D	0	7	<table border="1" style="margin: auto;"> <tr><td>A</td><td>1</td><td>6</td></tr> <tr><td>B</td><td>1</td><td>5</td></tr> <tr><td>C</td><td>2</td><td>4</td></tr> <tr><td>D</td><td>4</td><td>7</td></tr> </table>	A	1	6	B	1	5	C	2	4	D	4	7	<table border="1" style="margin: auto;"> <tr><td>A</td><td>1</td><td>6</td></tr> <tr><td>B</td><td>2</td><td>5</td></tr> <tr><td>C</td><td>2</td><td>4</td></tr> <tr><td>D</td><td>4</td><td>7</td></tr> </table>	A	1	6	B	2	5	C	2	4	D	4	7
A	0	6																																				
B	0	5																																				
C	0	4																																				
D	0	7																																				
A	1	6																																				
B	1	5																																				
C	2	4																																				
D	4	7																																				
A	1	6																																				
B	2	5																																				
C	2	4																																				
D	4	7																																				
Free: 10	Free: 2	Free: 1																																				
Any sequence finishes	C,B,A,D finishes	Deadlock (unsafe state)																																				

- Bankers' algorithm: before granting a request, ensure that a sequence exists that will allow all processes to complete
 - Use previous methods to find such a sequence
 - If a sequence exists, allow the requests
 - If there's no such sequence, deny the request
- Can be slow: must be done on each request!

1. Using the banker's algorithm, determine whether the following state is unsafe based on the snapshot of the system below. If the state is safe, provide a safe sequence of execution. Otherwise explain why it is unsafe. Show your calculations for full credit.

[10 points]

	<u>Allocation</u>	<u>Max (Demand)</u>	<u>Available</u> = (1, 0, 0, 2)
	A B C D	A B C D	
P_0	3 0 1 4	5 1 1 7	
P_1	2 2 1 0	3 2 1 1	
P_2	3 1 2 1	3 3 2 1	
P_3	0 5 1 0	4 6 1 2	
P_4	4 2 1 2	6 3 2 5	

Sample question (bankers algorithm)

	Allocation	Max	Need	Available
	A B C D	A B C D	A B C D	3 2 2 1
P0	4 0 0 1	7 0 2 1		
P1	1 1 0 0	1 6 5 0		
P2	1 0 4 5	3 3 4 6		
P3	0 4 2 1	1 5 6 2		
P4	0 3 1 2	2 4 3 2		

Using Bankers algorithm answer the following:

1. How many resources of type A, B, C and D are there?
2. What are the contents of the Need matrix?
3. Is the system in a safe state? Provide reasoning for your answer (show the sequence in which the processes would finish)
4. If a request from process P2 arrives for additional resources of {0, 2, 0, 0}, can the Bankers algorithm grant the request immediately? Provide reasoning for your answer. (14 Marks)

Q1. **Deadlocks.** The Banker's algorithm is used for deadlock avoidance. Consider the state of resource availability and allocation defined by the following matrices.

Claim Matrix

	R1	R2	R3
P1	3	1	4
P2	6	1	3
P3	3	2	2
P4	4	2	2

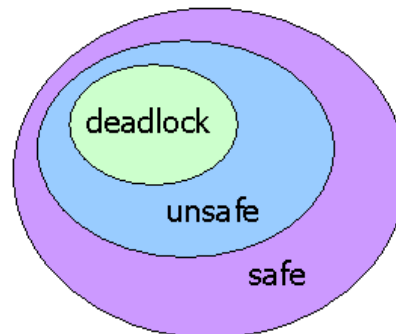
Allocation Matrix

	R1	R2	R3
P1	2	1	1
P2	5	1	1
P3	2	0	1
P4	0	0	2

- (1) Assuming that the total amounts for resources R1, R2, and R3 are 10, 2, and 10, should a new request to the Banker's algorithm by process P3 to acquire one additional resource from R1 and one additional resource from R3 be approved or denied? Explain why or why not.
- (2) Assuming that the total amounts for resources R1, R2, and R3 are 10, 2, and 10, should a new request to the Banker's algorithm by process P4 to acquire one additional resource from R3 be approved or denied? Explain why or why not.
- (3) Assuming that the total amounts for resources R1 and R2 are 10 and 2, what is the minimum amount for resource R3 that would render the above state a safe state under the Banker's algorithm?
- (4) Given your answer for part (3) what are all the possible orderings for the four processes P1, P2, P3, and P4 to complete their execution subject to the Banker's algorithm.
- (5) Assuming that the total amounts for resources R1 and R2 are 10 and 2, what is the minimum amount for resource R3 that would make it possible for the Banker's algorithm to allow process P1 to complete its execution before all other three processes?

The Banker's Algorithm

- Idea: know what each process *might* ask for
- Only make allocations that leave the system in a *safe* state
- Inefficient



Resource allocation state space

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P ₁	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P ₂	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P ₃	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P ₄	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
Total	2	9	10	12					3	14	12	12				

Need Matrix?
 Is system in safe state?
 if yes then find safe sequence

Total: - 3 14 12 12

$Need_i = Allocation_i - Max_i$

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P ₁	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P ₂	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P ₃	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P ₄	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
Total									3	14	12	12				

Need Matrix? ✓
 Is system in safe state? Yes ✓
 if yes then find safe sequence ✓

Total: - 3 14 12 12

$Need_i = Allocation_i - Max_i$

Sequence: - P₀ P₂ P₃ P₄ P₁

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P ₁	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P ₂	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P ₃	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P ₄	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
Total									3	14	12	12				

otherwise goto step 4
 Step 4:- if flag[i]=0 for all i
 then system is in safe state
 otherwise unsafe state

P₀ P₂ P₃ P₄ P₁

n: no. of processes
 m: no. of resources

also: Input:- Processes

- any 2 out of 3 (Max Need, Allocation)
- available or total no. of resources

flag[i]=0 for i=0 to (n-1) & find Need[n][m] = Max[n][m] - allocation[n][m]

find a process P_i such that: - flag[i]=0 & Need_i <= Available

if such i exists then
 flag[i]=1, available = available + Allocate_i
 goto step 1

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	2	0	0	1	4	2	1	2	3	3	2	1	2	2	1	1
P ₁	3	1	2	1	5	2	5	2					2	1	3	1
P ₂	2	1	0	3	2	3	1	6					0	2	1	3
P ₃	1	3	1	2	1	4	2	4					0	1	1	2
P ₄	1	4	3	2	3	6	6	5					2	2	3	3
	9	9	6	9												

Resource-Request algo:-

Step 1:- if Request_i ≤ Need_i; then go to step 2
 otherwise error

Step 2:- if Request_i ≤ Available then go to step 3
 otherwise P_i will wait

Step 3:- System pretend as if request has been granted by modifying the state as follows:-

- Available -= Request_i
- Allocation_i += Request_i
- Need_i -= Request_i

① Need Matrix? Yes

② Is system in Safe state? if Yes find safe sequence

③ if request from P₁ arrives for (1,1,0,0) can request be immediately granted? Yes

④ if request from P₄ arrives for (0,0,2,0) can it be immediately granted? No

If modified resource-allocation state is safe then request granted otherwise P_i will wait & old allocation state is restored

Deadlock Detection & Recovery

- Allow the system to enter into deadlocked state
- Deadlock detection algorithms (2 types)

Recovery techniques

- Single instance (Wait-for graph)
 - (Detect cycle)
- Multiple instances (Banker's algo)

Subscribe JENNY'S LECTURES CS/IT NET&JRF for full syllabus

Deadlock Recovery

① Optimistic approach (Preemption of Resources & Processes)

- Pre-empt some resources from process & give these resources to other processes until the deadlock cycle is broken

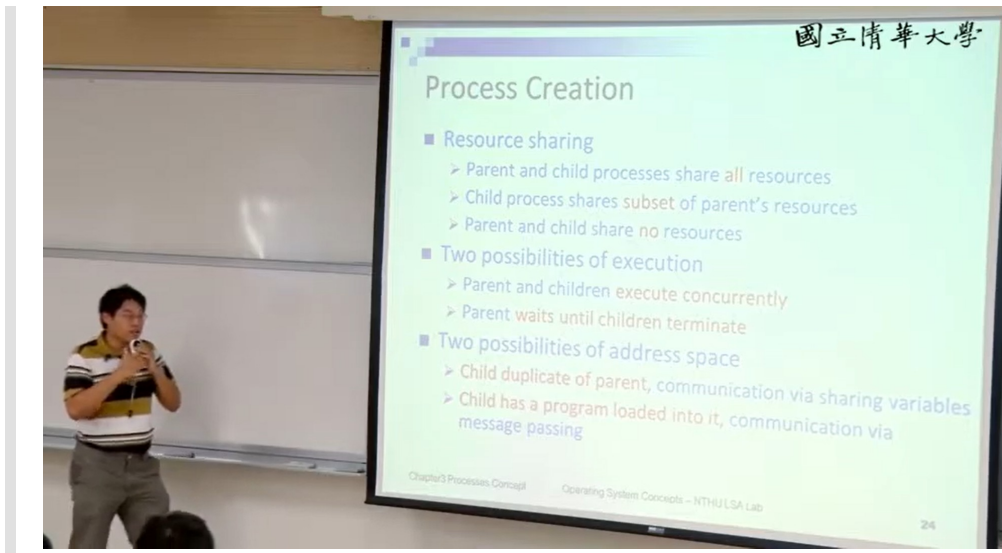
selecting a victim

- Rollback → Priority of process
- Starvation → How long the process has computed?
- Starvation → how much longer a process will compute before completion
- Starvation → How many & what type of resources process has used
- Starvation → How many resources the process needs to complete its execution etc.

② Pessimistic Approach (Process Termination)

- Abort all deadlocked processes *costly*
- Abort one process at a time and decide next to abort after deadlock detection
- overhead of calling detection algo again & again

Process Create



國立清華大學

UNIX/Linux Process Creation

- **fork** system call
 - Create a new (child) process
 - The new process **duplicates the address space** of its parent
 - Child & Parent **execute concurrently** after fork
 - Child: return value of fork is 0
 - Parent: return value of fork is PID of the child process
- **execlp** system call
 - Load a new binary file into memory – **destroying the old code**
- **wait** system call
 - The parent waits for **one of its child processes** to complete

Chapter3 Processes Concept Operating System Concepts - NTHU LSA Lab 25

國立清華大學

UNIX/Linux Process Creation

- **Memory space of fork():**
 - Old implementation: A's child is an **exact copy** of parent
 - Current implementation: use **copy-on-write** technique to store **differences in A's child address space**

free memory
B
free memory
A
kernel

Originally

free memory
A's child
B
free memory
A
kernel

After A does an **fork**

free memory
C
B
free memory
A
kernel

After the child does an **execlp**

Chapter3 Processes Concept Operating System Concepts - NTHU LSA Lab 26

國立清華大學

UNIX/Linux Example

```

#include <stdio.h>
void main ( )
{
    int A;
    /* fork another process */
    A = fork( );

    if (A == 0) { /* child process */
        printf("this is from child process\n");
        execlp("/bin/ls", "ls", NULL);
    } else { /* parent process */
        printf("this is from parent process\n");
        int pid = wait(&status);
        printf("Child %d completes", pid);
    }
    printf("process ends %d\n", A);
}

```

Output:
 this is from child process
 this is from parent process
 a.out hello.c readme.txt
 Child 32185 completes
 process ends 32185

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 27

國立清華大學

Example Quiz:

■ How many processes are created?

```

#include <stdio.h>
#include <unistd.h>
int main() {
    for (int i=0; i<3; i++){
        fork();
    }
    return 0;
}

```

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 28

國立清華大學

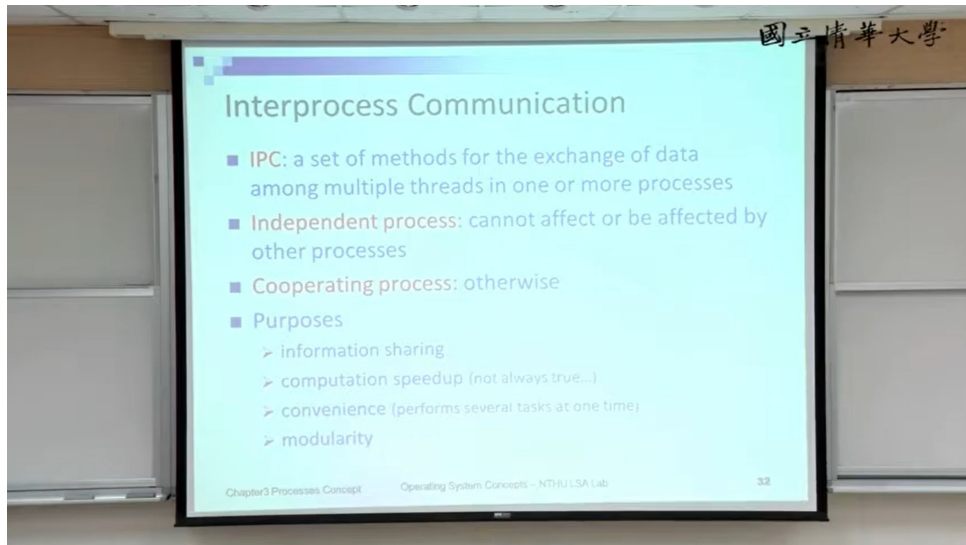
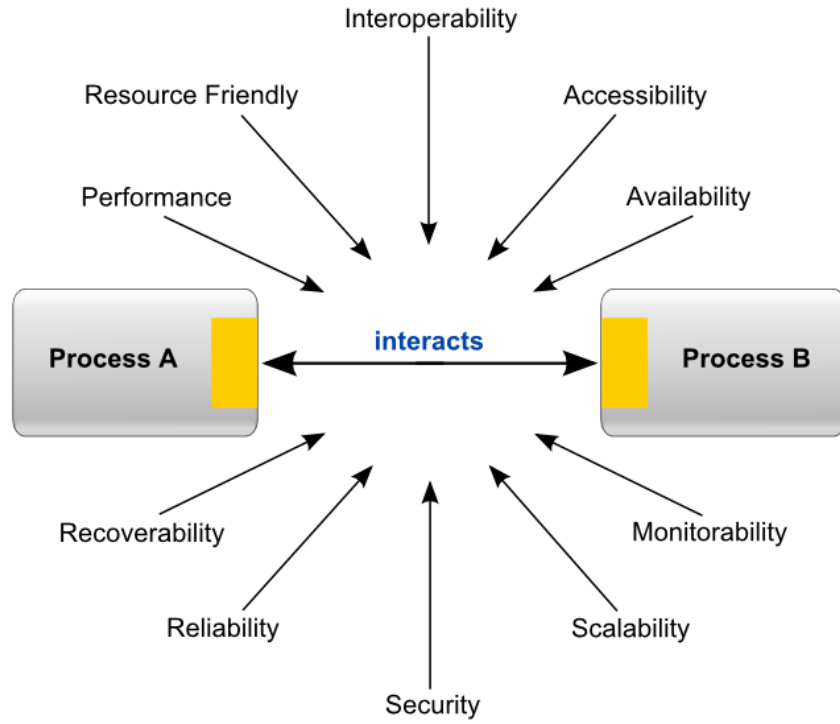
Process Termination

- Terminate when the last statement is executed or **exit()** is called
 - > All resources of the process, including physical & virtual memory, open files, I/O buffers, are deallocated by the OS
- Parent may terminate execution of children processes by specifying its PID (**abort**)
 - > Child has exceeded allocated resources
 - > Task assigned to child is no longer required
- Cascading termination:
 - > killing (exiting) parent → killing (exiting) all its children

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 29

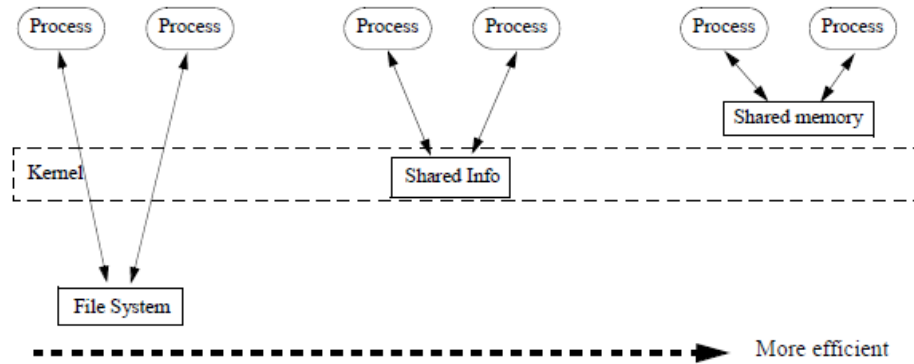
IPC(Interprocess Communication)

Interprocess Communication



IPC Mechanisms

IPC classified by implementation mechanisms.



Examples:

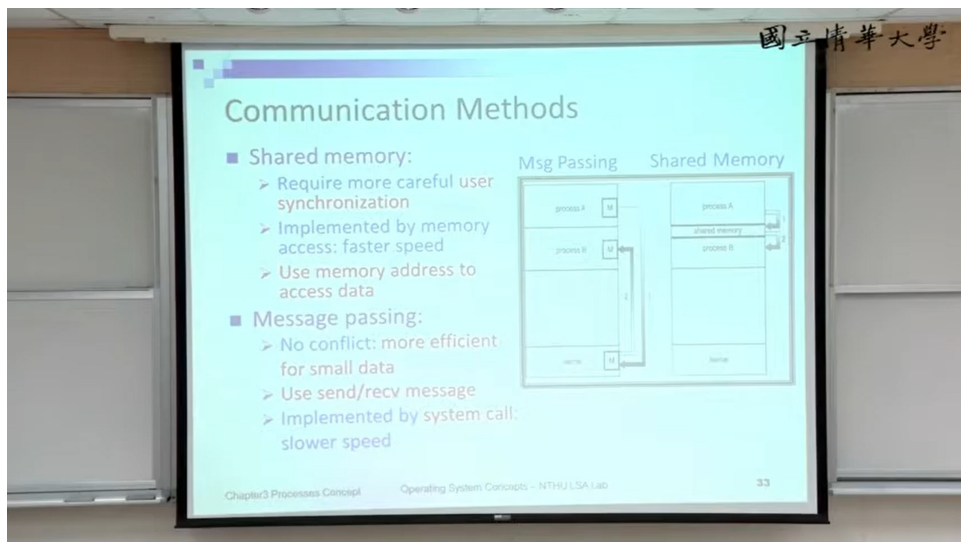
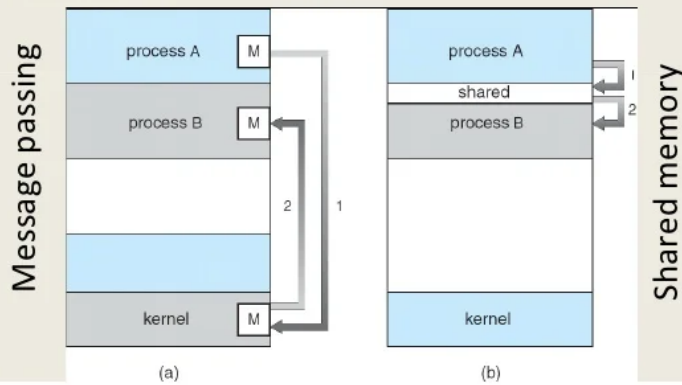
files

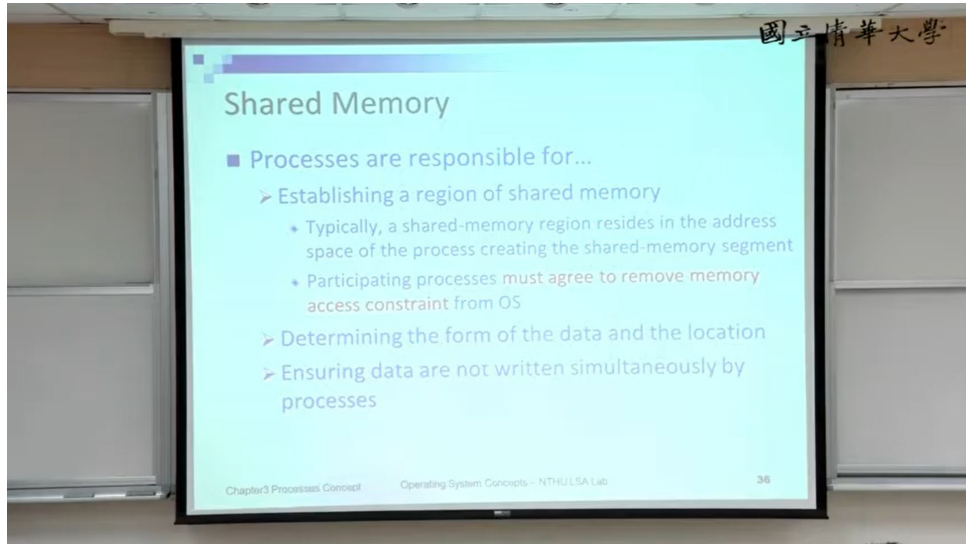
semaphore, socket,
pipe, message queue,
signal

shared memory

4. Interprocess Communication Contd...

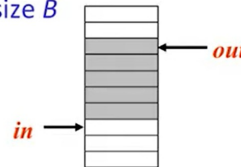
- Two fundamental models of Interprocess communication
- **Shared memory**
 - a region of memory that is shared by cooperating processes is established then exchange information takes place by reading and writing data to the shared region
- **Message passing**
 - communication takes place by means of messages exchanged between the cooperating processes





Consumer & Producer Problem

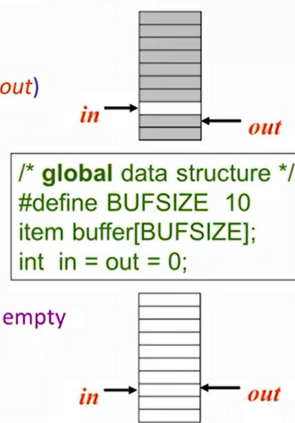
- **Producer** process produces information that is consumed by a **Consumer** process
- Buffer as a circular array with size B
 - next free: in
 - first available: out
 - empty: $in = out$
 - full: $(in+1) \% B = out$

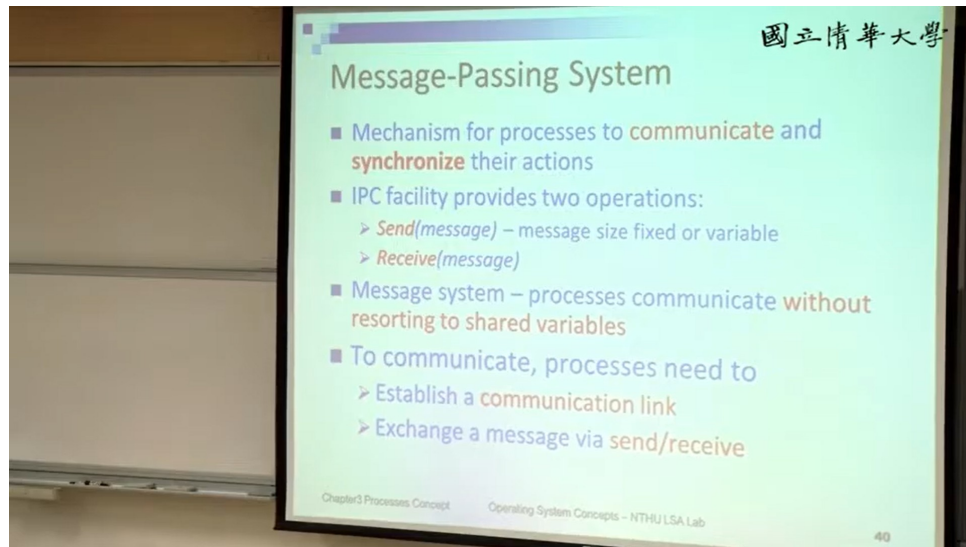


Shared-Memory Solution

```

/*producer*/
while (1) {
    while (((in + 1) % BUFFER_SIZE) == out)
        ; //wait if buffer is full
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
/*consumer*/
while (1) {
    while (in == out); //wait if buffer is empty
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
    
```

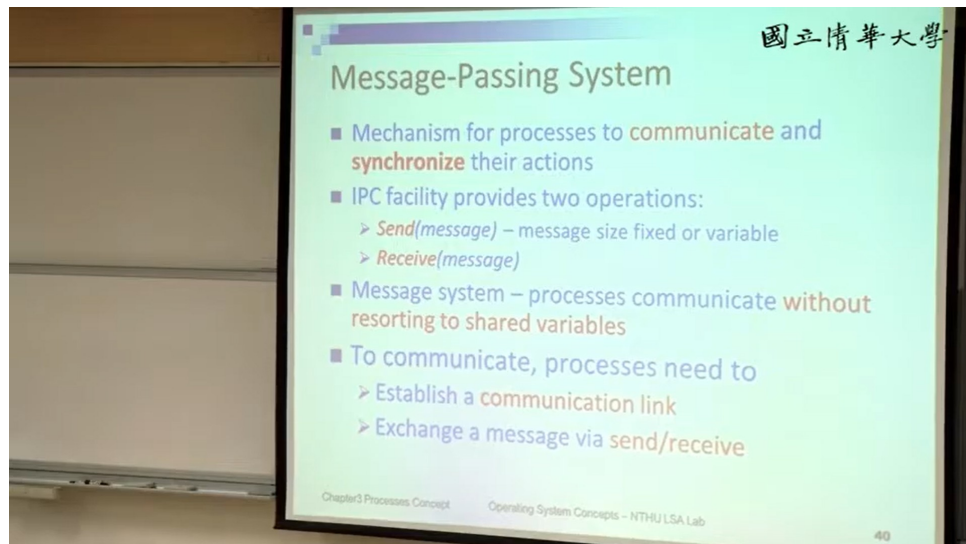




國立清華大學

Message-Passing System

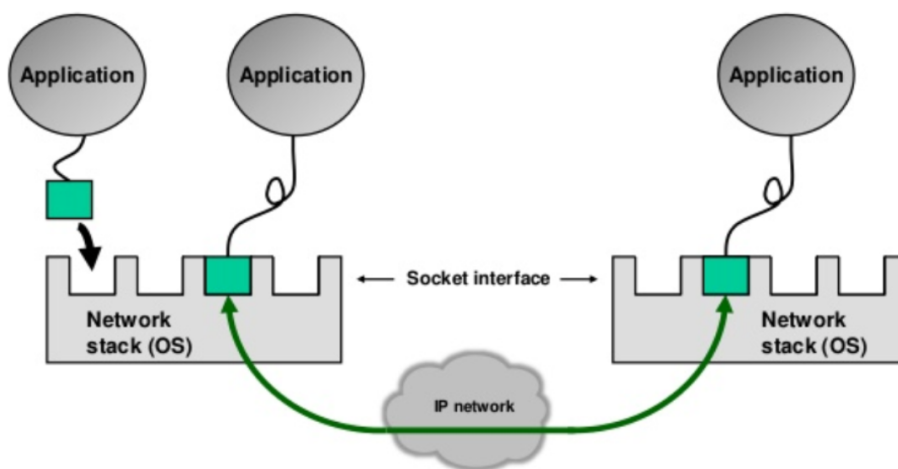
- Implementation of communication link
 - physical (e.g., shared memory, HW bus, or network)
 - logical (e.g., **logical properties**)
 - ✦ **Direct or indirect communication**
 - ✦ Symmetric or asymmetric communication
 - ✦ **Blocking or non-blocking**
 - ✦ Automatic or explicit buffering
 - ✦ Send by copy or send by reference
 - ✦ Fixed-sized or variable-sized messages



Communication Methods

- Sockets:
 - > A network connection identified by IP & port
 - > Exchange unstructured stream of bytes
- Remote Procedure Calls:
 - > Cause a procedure to execute in another address space
 - > Parameters and return values are passed by message

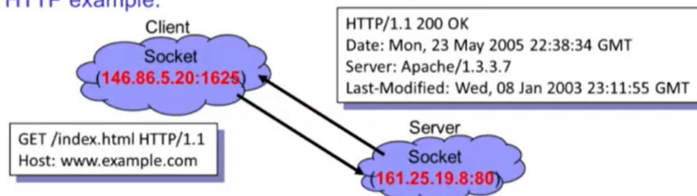
Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 34



Sockets

- Considered as a low-level form of communication **unstructured stream of bytes** to be exchanged
- **Data parsing responsibility falls upon the server and the client applications**

HTTP example:



國立清華大學

Sockets

- A socket is identified by a concatenation of IP address and port number
- Communication consists between a pair of sockets
- Use 127.0.0.1 to refer itself

```

Client (146.86.5.20)
  socket()
  connect()
  write()
  read()
  close()

Server (161.25.19.8)
  socket()
  bind()
  listen()
  accept()
  read()
  write()
  close()

Well-known port 161.25.19.8:80
Assign port 146.86.5.20:1625
Block until client requests
Data req.
Data reply
  
```

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 48

國立清華大學

Remote Procedure Calls: RPC

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
 - allows programs to call procedures located on other machines (and other processes)
- Stubs – client-side proxy for the actual procedure on the server

```

client
  val = server.someMethod(A,B)
  stub

remote object
  boolean someMethod (Object x, Object y)
  {
    implementation of someMethod
  }
  skeleton

A, B, someMethod
boolean return value
  
```

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 50

國立清華大學

Client and Server Stubs

Client stub:

- Packs parameters into a message (i.e. parameter marshaling)
- Calls OS to send directly to the server
- Waits for result-return from the server

```

Client
  Call remote procedure
  Request
  Wait for result
  Return from call

Server
  Call local procedure and return results
  Reply
  
```

Server stub:

- Receives a call from a client
- Unpacks the parameters
- Calls the corresponding procedure
- Returns results to the caller

Chapter3 Processes Concept Operating System Concepts – NTHU LSA Lab 51

Threads

Thread Control Block

- **Thread Control Block (TCB)** is a data structure in the operating system kernel which contains thread-specific information needed to manage it. The TCB is "the manifestation of a thread in an operating system".
 - An example of information contained within a TCB is:
 - Stack pointer: Points to thread's stack in the process
 - Program counter
 - State of the thread (running, ready, waiting, start, done)
 - Thread's register values
 - Pointer to the Process control block (PCB) of the process that the thread lives on
- The Thread Control Block acts as a library of information about the threads in a system. Specific information is stored in the thread control block highlighting important information about each thread.

國立清華大學

Threads

- A.k.a **lightweight process**: basic unit of CPU utilization
- All threads belonging to the same process share
 - code section, data section, and OS resources (e.g. open files and signals)
- But each thread has its own
 - thread ID, program counter, register set, and a stack

The diagram illustrates the difference between single-threaded and multithreaded processes. On the left, a 'single-threaded' process is shown with a single thread pointing to a shared set of resources: code, data, files, registers, and stack. On the right, a 'multithreaded' process is shown with three threads, each having its own registers and stack, but sharing the code, data, and files sections. A red box highlights the shared code, data, and files sections in the multithreaded process.

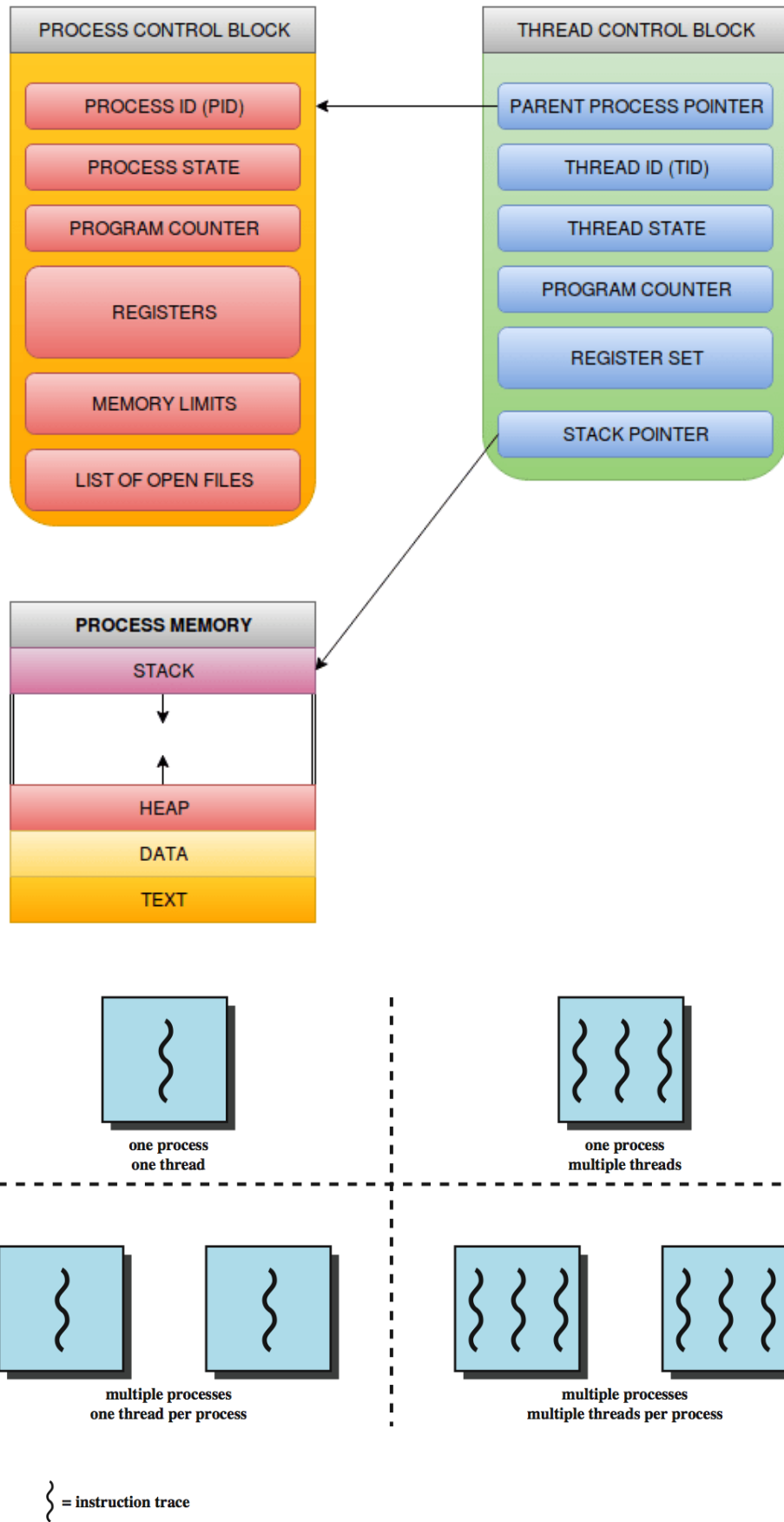
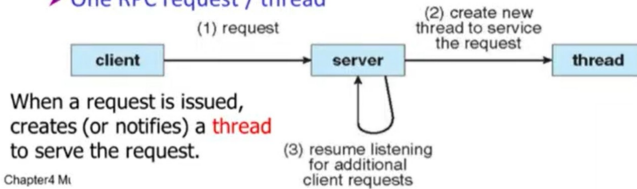


Figure 4.1 Threads and Processes [ANDE97]

Thread	Process
It is lightweight entity.	It is heavy weight entity
If a thread ends working process keep working.	Process may keep working and if a process terminates all its threads will terminate also.
Communication b/w threads happens via memory.	Communication b/w Process happens via OS.
The Creation of thread and context switching is inexpensive	The creation of the process is expensive.

Motivation

- **Example: a web browser**
 - One thread displays contents while the other thread receives data from network
- **Example: a web server**
 - One request / process: poor performance
 - One request / thread: better performance as code and resource sharing
- **Example: RPC server**
 - One RPC request / thread



Chapter4 Mt

4

Benefits of Multithreading

- **Responsiveness:** allow a program to continue running even if part of it is blocked or is performing a lengthy operation
- **Resource sharing:** several different threads of activity all within **the same address space**
- **Utilization of MP arch.:** Several thread may be **running in parallel** on different processors
- **Economy:** Allocating memory and resources for process creation is costly. In Solaris, creating a process is about **30 times slower than is creating a thread**, and **context switching is about five times slower**. A register set switch is still required, but **no memory-management related work is needed**

Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

5

Challenges in Multicore Programming

- **Dividing activities:** divide program into concurrent tasks
- **Data splitting:** divide data accessed and manipulated by the tasks
- **Data dependency:** synchronize data access
- **Balance:** evenly distribute tasks to cores
- **Testing and debugging**

Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

8

User vs. Kernel Threads

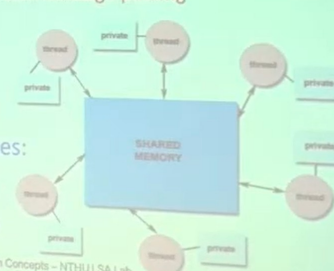
- User threads – thread management done by **user-level threads library**
 - POSIX Pthreads
 - Win32 threads
 - Java threads
- Kernel threads – supported by the **kernel (OS) directly**
 - Windows 2000 (NT)
 - Solaris
 - Linux
 - Tru64 UNIX

User vs. Kernel Threads

- User threads
 - **Thread library** provides support for thread creation, scheduling, and deletion
 - Generally **fast** to create and manage
 - **If the kernel is single-threaded, a user-thread blocks → entire process blocks** even if other threads are ready to run
- Kernel threads
 - The **kernel** performs thread creation, scheduling, etc.
 - Generally **slower** to create and manage
 - If a thread is blocked, the kernel can schedule another thread for execution

Shared-Memory Programming

- **Definition:** Processes communicate or work together with each other through a **shared memory space** which can be accessed by all processes
 - Faster & more efficient than message passing
- Many issues as well:
 - Synchronization
 - Deadlock
 - Cache coherence
- Programming techniques:
 - Parallelizing compiler
 - Unix processes
 - Threads (**Pthread**, Java)

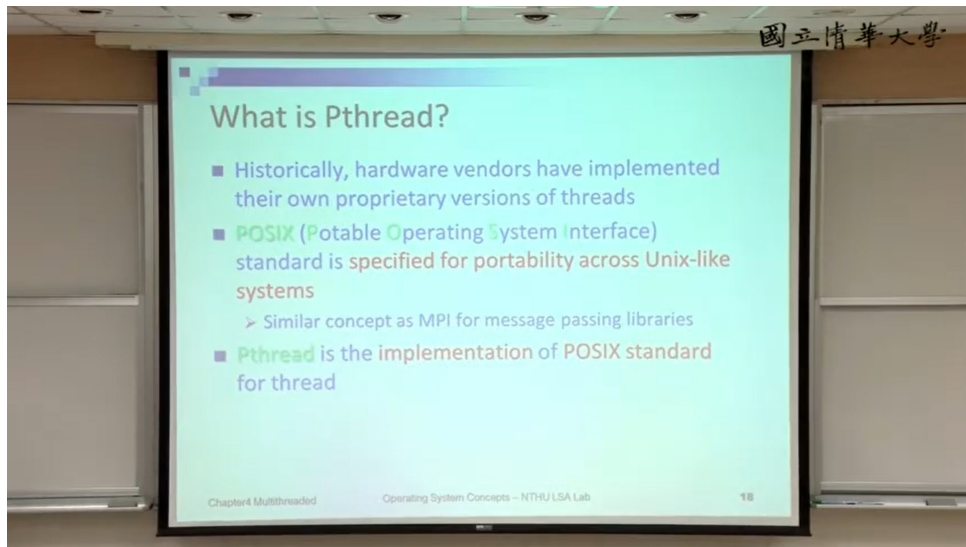


The diagram illustrates shared-memory programming. It features a central blue box labeled 'SHARED MEMORY'. Surrounding this central box are several smaller circles, each representing a process. Each process has its own 'private' memory space, indicated by a smaller circle labeled 'private' connected to the process circle. Arrows point from each process's private memory space to the central shared memory space, showing that all processes can access the same shared memory.

Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

17

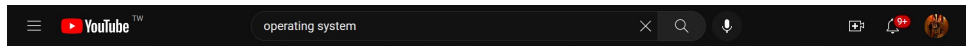
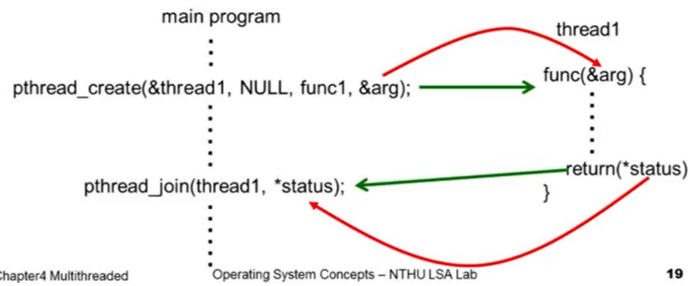


What is Pthread?

- Historically, hardware vendors have implemented their own proprietary versions of threads
- **POSIX** (Potable Operating System Interface) standard is specified for portability across Unix-like systems
 - > Similar concept as MPI for message passing libraries
- **Pthread** is the implementation of POSIX standard for thread

Pthread Creation

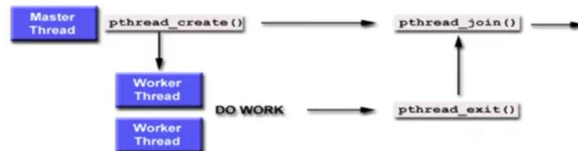
- `pthread_create(thread,attr,routine,arg)`
 - > **thread**: An **unique identifier** (token) for the new thread
 - > **attr**: It is used to set **thread attributes**. NULL for the default value:
 - > **routine**: The routine that the thread will execute once it is created
 - > **arg**: A **single argument** that may be passed to **routine**



Pthread Joining & Detaching

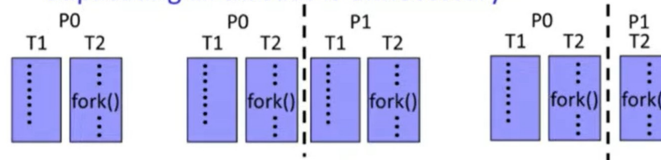
- `pthread_join(threadId, status)`
 - > **Blocks** until the specified **threadId** thread terminates
 - > One way to **accomplish synchronization** between threads
 - > Example: to create a pthread barrier


```
for (int i=0; i<n; i++) pthread_join(thread[i], NULL);
```
- `pthread_detach(threadId)`
 - > Once a thread is **detached**, it can **never** be joined
 - > Detach a thread could **free some system resources**



Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?
 - Some UNIX system support two versions of **fork()**
- **execlp()** works the same; **replace the entire process**
 - If **exec()** is called immediately after forking, then duplicating all threads is unnecessary



Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

26

Signal Handling

- Signals (**synchronous** or **asynchronous**) are used in UN systems to notify a process that an event has occurred
 - Synchronous: illegal memory access
 - Asynchronous: <control-C>
- A **signal handler** is used to process signals
 1. Signal is generated by particular event
 2. Signal is delivered to a process
 3. Signal is handled
- Options
 - Deliver the signal to the thread to which the signal applies
 - Deliver the signal to every thread in the process
 - Deliver the signal to certain threads in the process
 - Assign a specific thread to receive all signals for the process

Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

28

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages
 - Usually slightly **faster to service a request** with an existing thread **than create a new thread**
 - Allows the number of threads in the application(s) to be **bound to the size of the pool**
- **# of threads:** # of CPUs, expected # of requests, amount of physical memory

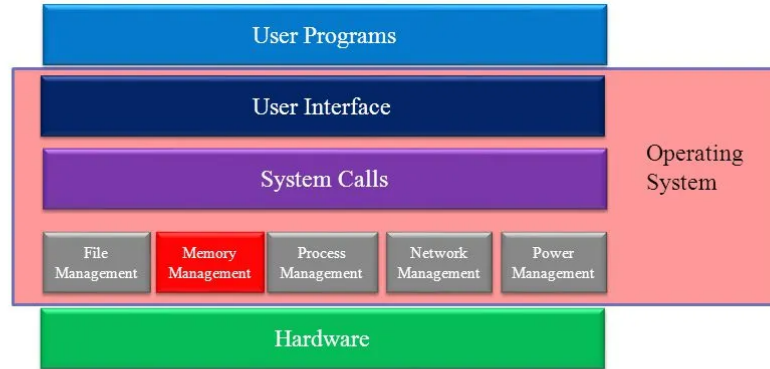
Chapter4 Multithreaded

Operating System Concepts – NTHU LSA Lab

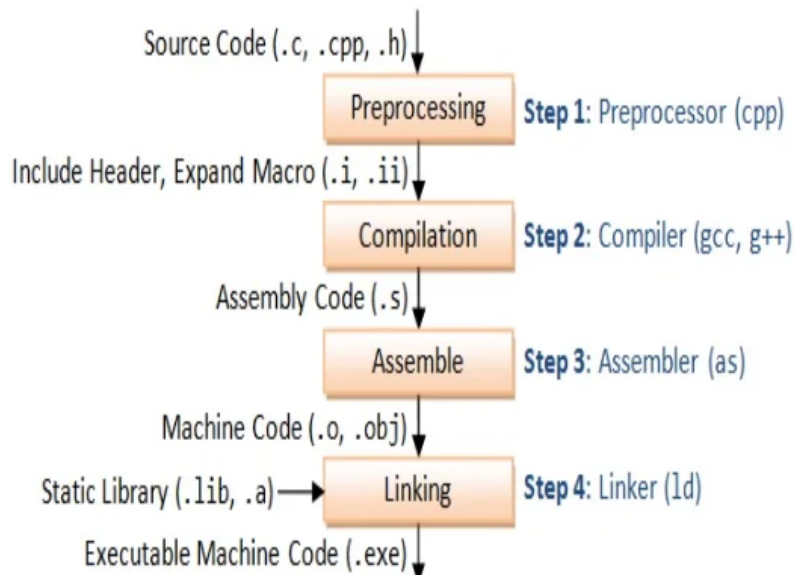
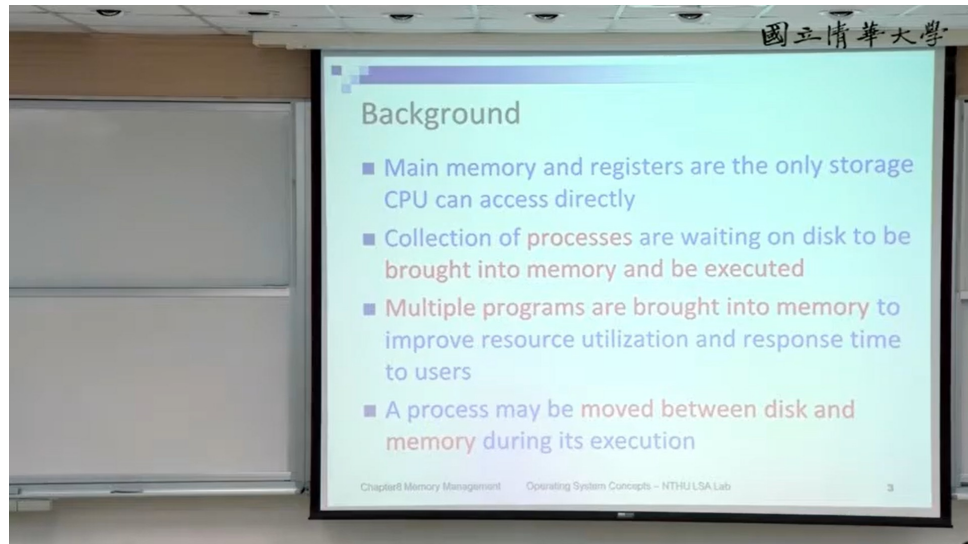
29

Memory Management

Memory Management in OS

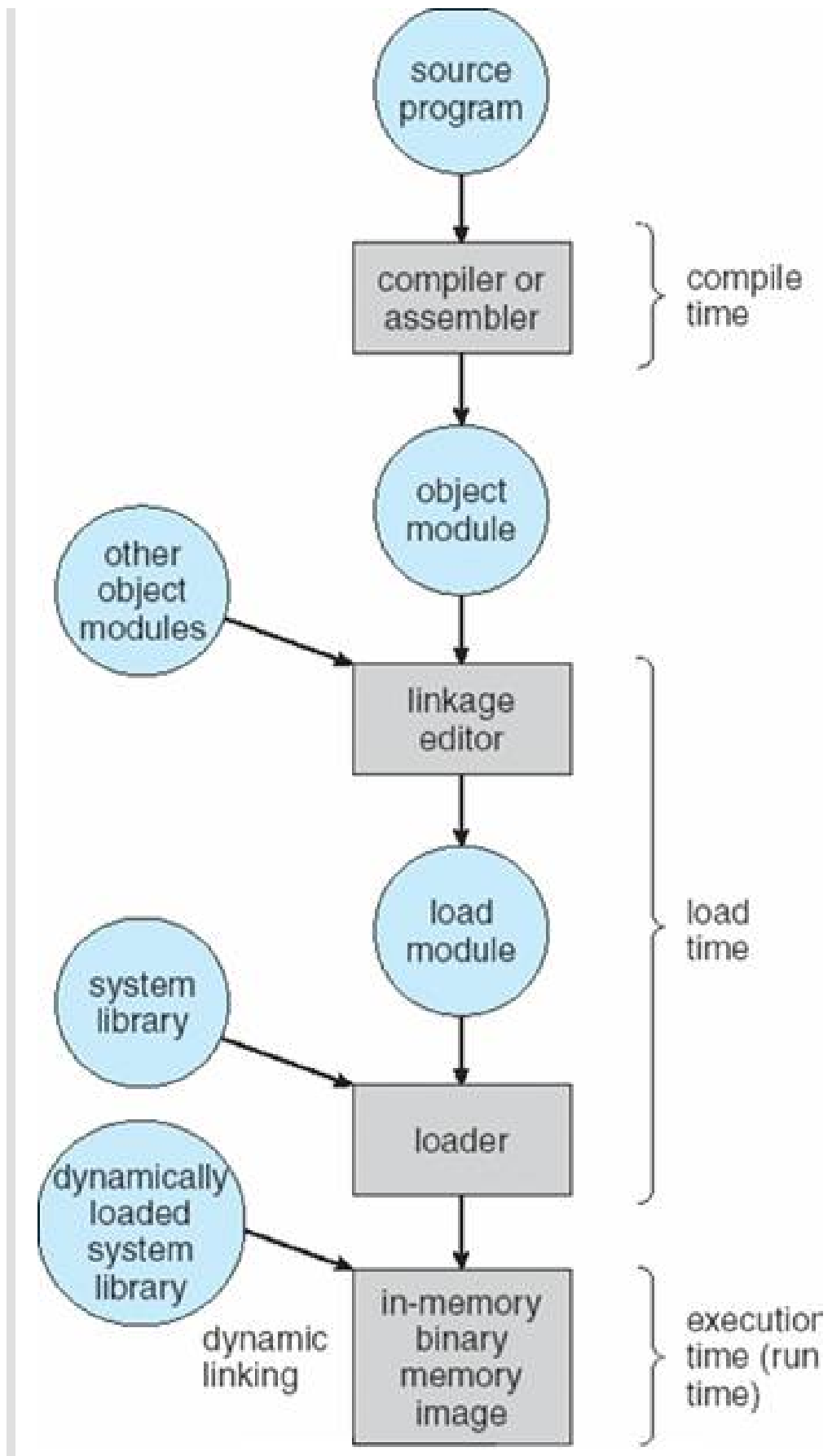


5



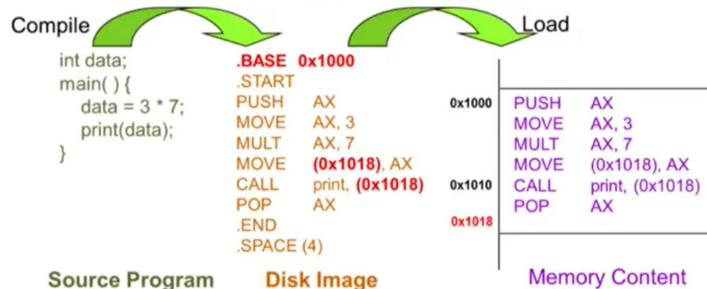
09-06-2015

3



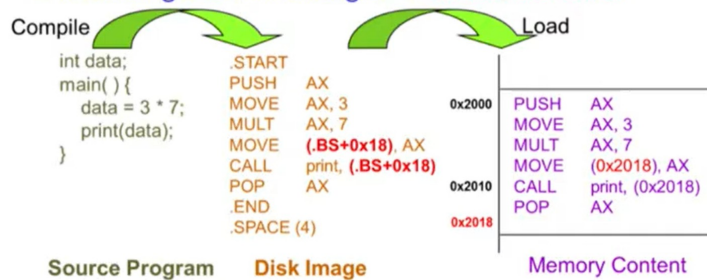
Address Binding – Compile Time

- Program is written as symbolic code
- Compiler translates symbolic code into *absolute code*
- If starting location changes → **recompile**
- Example: MS-DOS .COM format binary



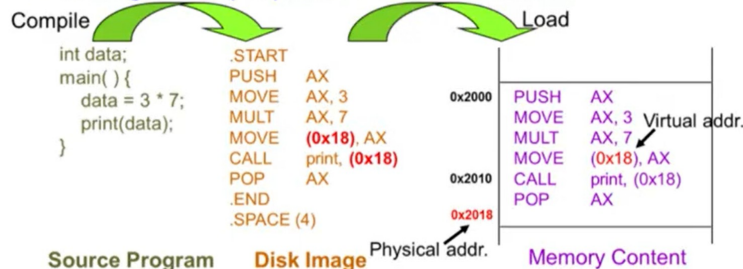
Address Binding – Load Time

- Compiler translates symbolic code into *relocatable code*
- **Relocatable code:**
 > Machine language that can be run from any memory location
- If starting location changes → **reload the code**



Address Binding – Execution Time

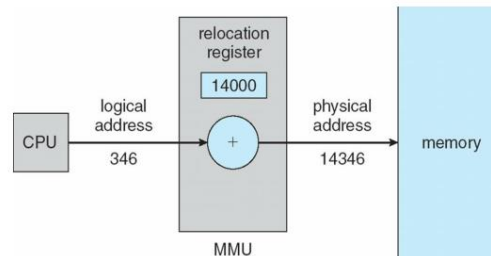
- Compiler translates symbolic code into *logical-address (i.e. virtual-address) code*
- Special hardware (i.e. MMU) is needed for this scheme
- Most general-purpose OS use this method





Memory-Management Unit (MMU)

- Hardware device that maps **virtual** to **physical** address
- In MMU scheme, the value in the relocation register is **added to every address generated by a user process** at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses



Dynamic relocation using a relocation register



國立清華大學

Logical vs. Physical Address

- Logical address – generated by CPU
 - > a.k.a. virtual address
- Physical address – seen by the memory module
- compile-time & load-time address binding
 - > logical addr = physical addr
- Execution-time address binding
 - > logical addr ≠ physical addr
- The user program deals with logical addresses; it never sees the real physical addresses

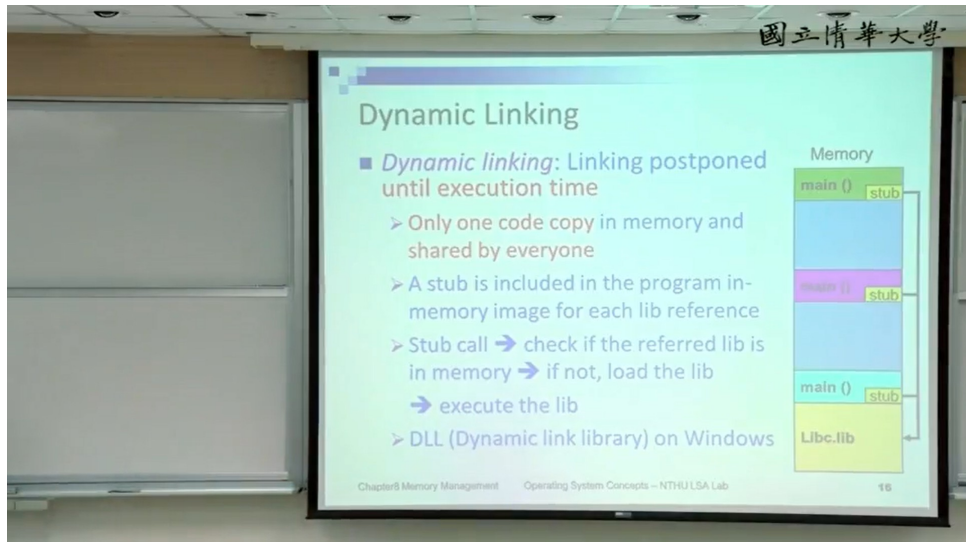
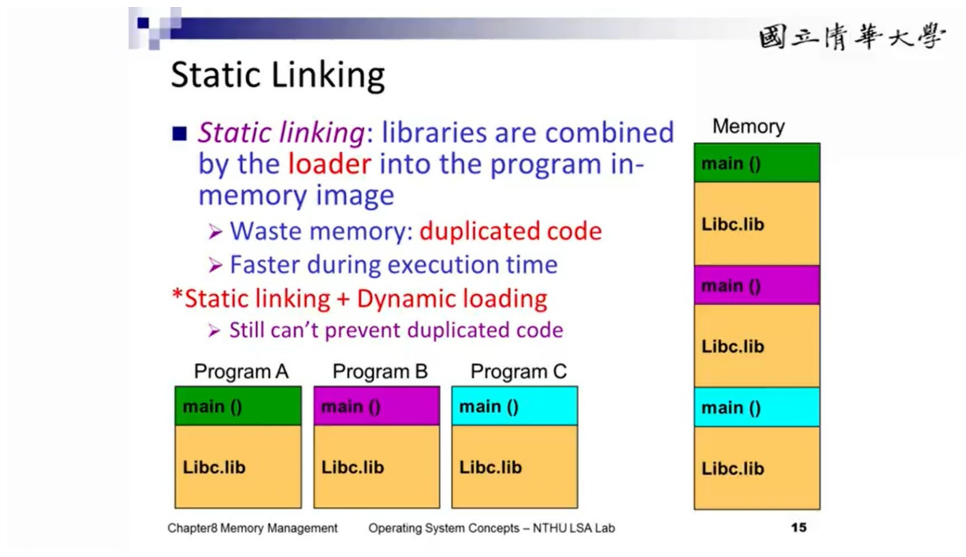
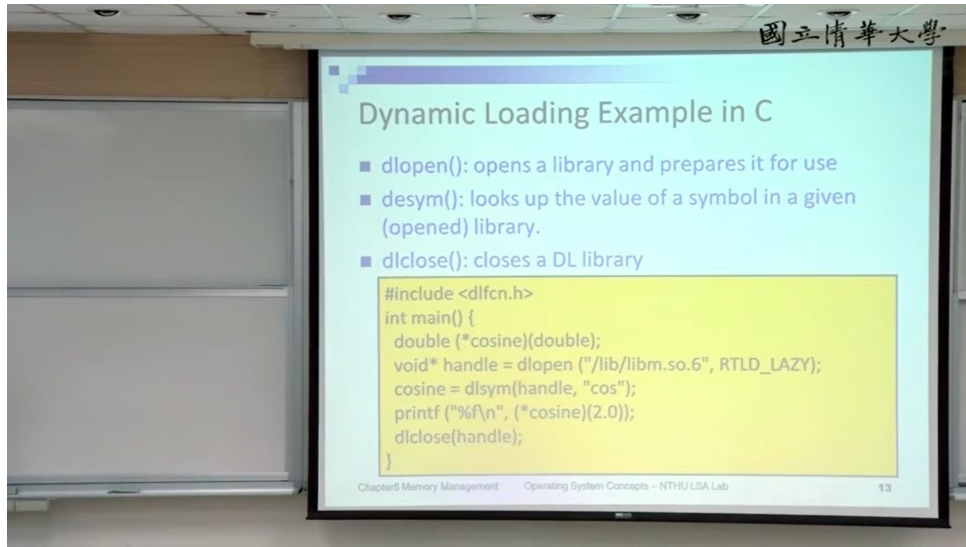
Chapter 8 Memory Management Operating System Concepts – NTHU LSA Lab 10

國立清華大學

Dynamic Loading

- The entire program must be in memory for it to execute?
- No, we can use dynamic-loading
 - > A routine is loaded into memory when it is called
- Better memory-space utilization
 - > unused routine is never loaded
 - > Particularly useful when large amounts of code are infrequently used (e.g., error handling code)
- No special support from OS is required implemented through program (library, API calls)

Chapter 8 Memory Management Operating System Concepts – NTHU LSA Lab 12



Fragmentation

國立清華大學

Fragmentation

- **External fragmentation**
 - Total free memory space is big enough to satisfy a request, but is not contiguous
 - Occur in variable-size allocation
- **Internal fragmentation**
 - Memory that is internal to a partition but is not being used
 - Occur in fixed-partition allocation
- **Solution: compaction**
 - Shuffle the memory contents to place all free memory together in one large block at execution time
 - Only if binding is done at execution time

External

Internal

Compaction

Chapter 8 Memory Management Operating System Concepts – NTHU LSA Lab

Paging

國立清華大學

Paging Concept

- **Method:**
 - Divide physical memory into fixed-sized blocks called **frames**
 - Divide logical address space into blocks of the same size called **pages**
 - To run a program of n pages, need to find n free frames and load the program
 - **keep track of free frames**
 - Set up a **page table** to translate logical to physical addresses
- **Benefit:**
 - Allow the physical-address space of a process to be **noncontiguous**
 - Avoid external fragmentation
 - Limited internal fragmentation
 - Provide **shared memory/pages**

Chapter 8 Memory Management Operating System Concepts – NTHU LSA Lab

國立清華大學

Address Translation Scheme

- Logical address is divided into two parts:
 - **Page number (p)**
 - ✦ used as an index into a page table which contains base address of each page in physical memory
 - ✦ N bits means a process can allocate at most 2^N pages
 - ➔ $2^N \times$ page size memory size
 - **Page offset (d)**
 - ✦ combined with base address to define the physical memory address that is sent to the memory unit
 - ✦ N bits means the **page size** is 2^N
- **Physical addr = page base addr + page offset**

Chapter 8 Memory Management Operating System Concepts – NTHU LSA Lab

MEMORY MANAGEMENT

PAGING

Permits a program's memory to be physically noncontiguous so it can be allocated from wherever available. This avoids fragmentation and compaction.

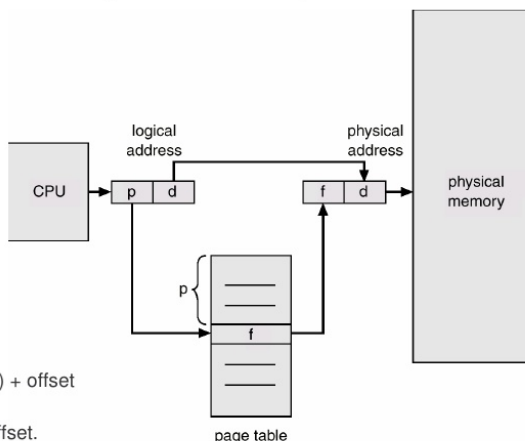
Frames = physical blocks
Pages = logical blocks

Size of frames/pages is defined by hardware (power of 2 to ease calculations)

HARDWARE

An address is determined by:

- page number (index into table) + offset
- > mapping into --->
- base address (from table) + offset.



8: Memory Management

Address Translation Architecture

國立清華大學

- If Page size is 1KB(2^{10}) & Page 2 maps to frame 5
- Given 13 bits logical address: (p=2,d=20), what is physical addr.?
- $5 * (1KB) + 20 = 1,010,000,000,000 + 0,000,010,100 = 1,010,000,010,100$

The diagram shows the MMU (Memory Management Unit) architecture. The CPU provides a logical address (p, d) to the MMU. The MMU uses the page number 'p' to find the corresponding frame number 'f' in the page table. The MMU then outputs a physical address (f, d) to the physical memory.

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 33

Address Translation

國立清華大學

- Total number of pages does not need to be the same as the total number of frames
 - Total # pages determines the logical memory size of a process
 - Total # frames depending on the size of physical memory
- E.g.: Given 32 bits logical address, 36 bits physical address and 4KB page size, what does it mean?
 - Page table size: $2^{32} / 2^{12} = 2^{20}$ entries
 - Max program memory: $2^{32} = 4GB$
 - Total physical memory size: $2^{36} = 64GB$
 - Number of bits for page number: 2^{20} pages \rightarrow 20bits
 - Number of bits for frame number: 2^{24} frames \rightarrow 24bits
 - Number of bits for page offset: 4KB page size = 2^{12} bytes \rightarrow 12

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 34

國立清華大學

Paging Summary

- Paging helps separate user's view of memory and the actual physical memory
- User view's memory: one single contiguous space
 - Actually, user's memory is scatter out in physical memory
- OS maintains a copy of the **page table** for each process
- OS maintains a **frame table** for managing physical memory
 - One entry for each physical frame
 - Indicate whether a frame is free or allocated
 - If allocated, to which page of which process or processes

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 37

國立清華大學

Implementation of Page Table

- Page table is kept in memory
- **Page-table base register (PTBR)**
 - The **physical memory address** of the page table
 - The PTBR value is stored in **PCB (Process Control Block)**
 - Changing the value of PTBR during **Context-switch**
- With PTBR, each memory reference results in **2 memory reads**
 - One for the page table and one for the real address
- The 2-access problem can be solved by
 - **Translation Look-aside Buffers (TLB)** (HW) which is implemented by **Associative memory** (HW)

國立清華大學

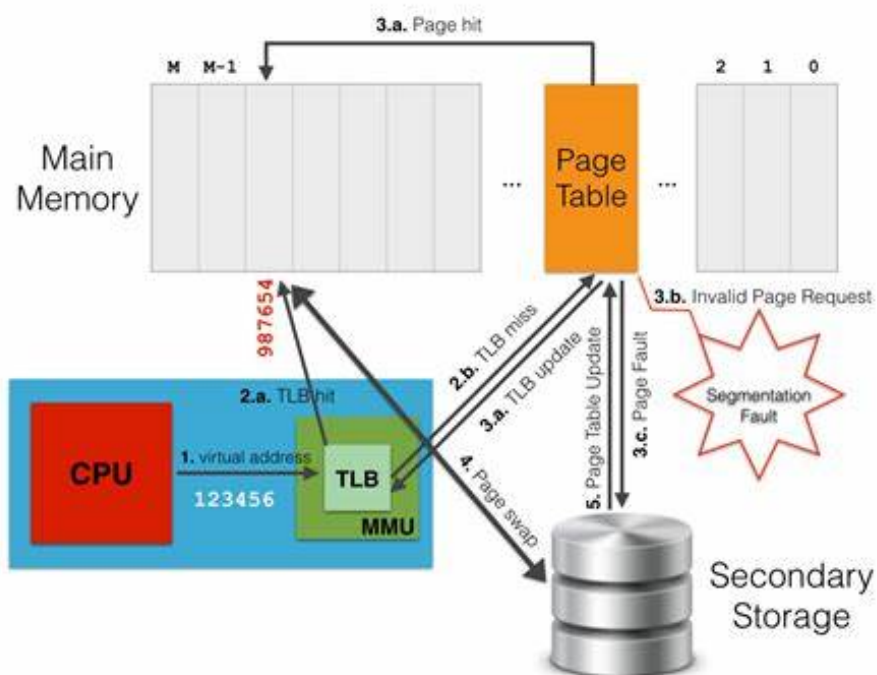
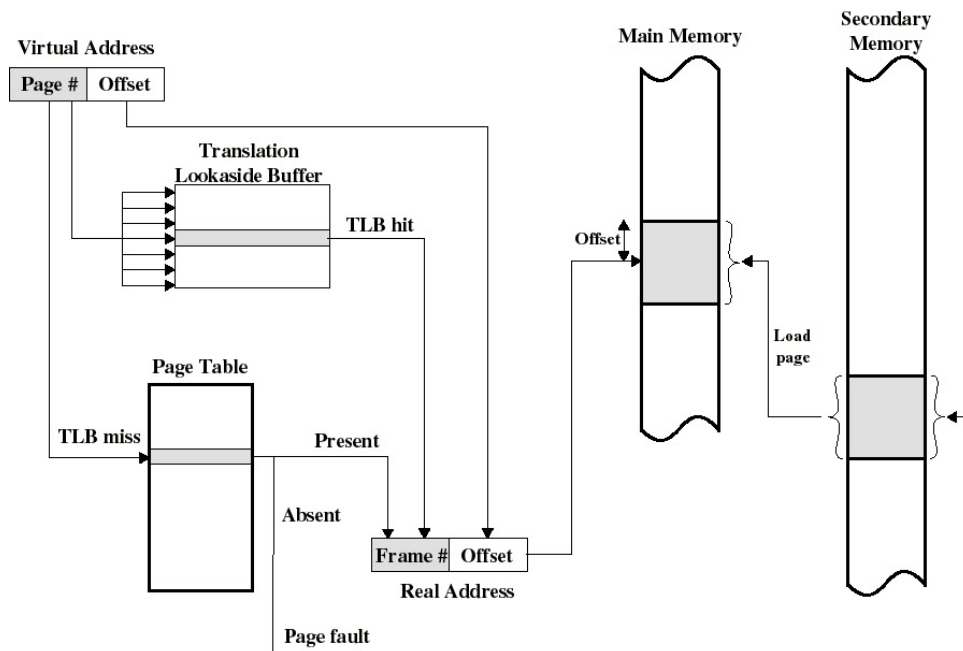
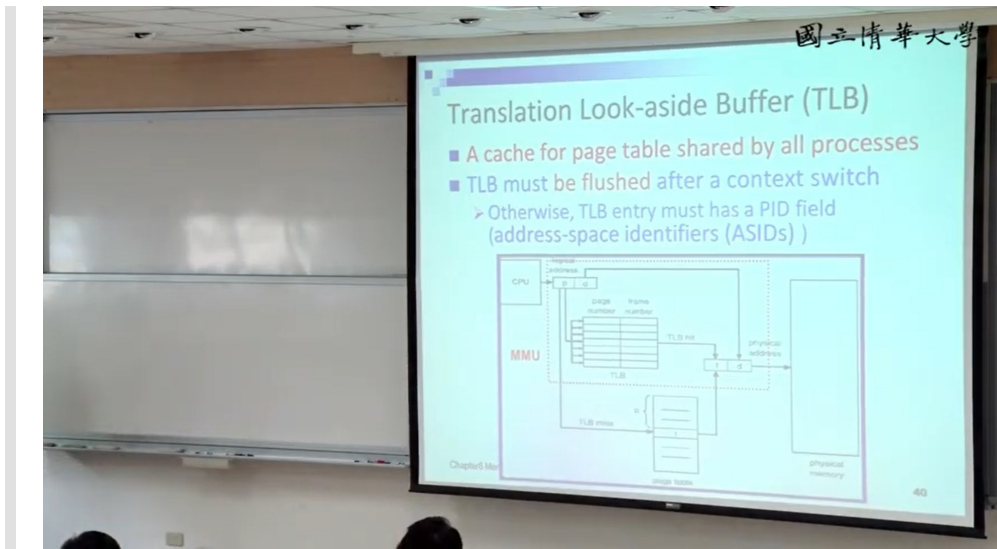
Associative Memory

- All memory entries can be accessed at the same time
 - Each entry corresponds to an associative register
- But number of entries are limited
 - Typical number of entries: 64 ~ 1024
 - Associative memory – parallel search

Page #	Frame #

Address translation (A' , A'')

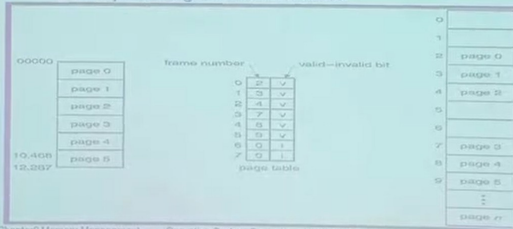
- If A' is in associative register, get frame # out.
- Otherwise get frame # from page table in memory



Valid-Invalid Bit Example

■ Potential issues:

- Un-used page entry cause memory waste → use page table length register (PTLR)
- Process memory may NOT be on the boundary of a page → memory limit register is still needed



Shared Pages

- Paging allows processes share common code, which must be reentrant
- Reentrant code (pure code)
 - It never change during execution
 - text editors, compilers, web servers, etc
- Only one copy of the shared code needs to be kept in physical memory
- Two (several) virtual addresses are mapped to one physical address
- Process keeps a copy of its own private data and code

Page Table Memory Structure

- Page table could be huge and difficult to be loaded
 - 4GB (2^{32}) logical address space with 4KB (2^{12}) page
 - 1 million (2^{20}) page table entry
 - Assume each entry need 4 bytes (32bits)
 - Total size=4MB
 - Need to break it into several smaller page tables, better within a single page size (i.e. 4KB)
 - Or reduce the total size of page table
- Solutions:
 - Hierarchical Paging
 - Hash Page Tables
 - Inverted Page Table

Hashed Page Table Address Translation

logical address: p | d

physical address: r | d

hash function

hash table: [q | s | l | p | r | i | ...]

physical memory

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 54

Inverted Page Table

- Maintains **NO** page table for each process
- Maintains a *frame table* for the whole memory
 - One entry for each real frame of memory
- Each entry in the frame table has
 - (PID, Page Number)
- Eliminate the memory needed for page tables but increase memory access time
 - Each access needs to search the whole frame table
 - Solution: use hashing for the frame table
- Hard to support shared page/memory

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 56

Segmentation

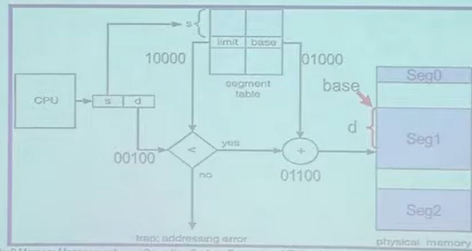
- Memory-management scheme that supports **user view of memory**
- A program is a collection of segments. A segment is a logical unit such as:
 - main program
 - function, object
 - local/global variables,
 - stack, symbol table,
 - arrays, etc...

logical address space

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 60

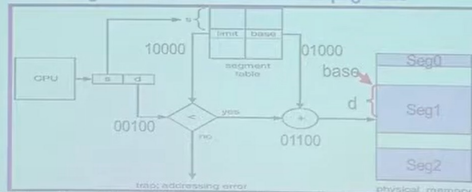
Segmentation Hardware

- Limit register is used to check offset length
- MMU allocate memory by assigning an appropriate base address for each segment
 - > Physical address cannot overlap between segments



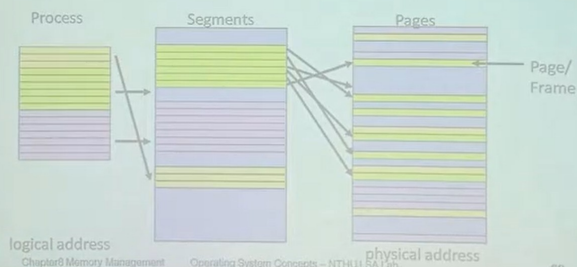
Address Translation Comparison

- Segment
 - > Table entry: (segment base addr., limit)
 - > Segment base addr. can be arbitrary
 - > The length of "offset" is the same as the physical memory size
- Page:
 - > Table entry: (frame base addr.)
 - > Frame base addr. = frame number * page size
 - > The length of "offset" is the same as page size



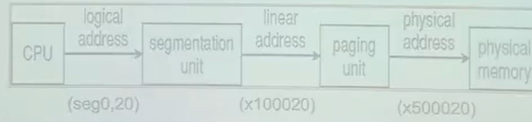
Basic Concept

- Apply segmentation in logical address space
- Apply paging in physical address space



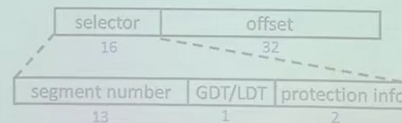
Address Translation

- CPU generates logical address
 - > Given to segmentation unit
 - produces linear addresses
 - > Linear address given to paging unit
 - generates physical address in main memory
- Segmentation and paging units form equivalent of MMU



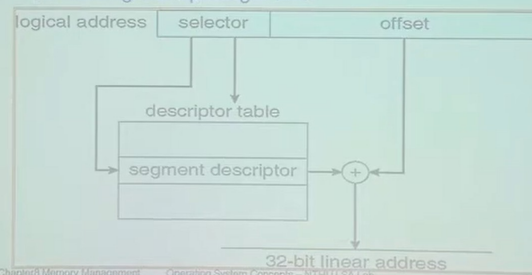
Example: The Intel Pentium

- Logical-address space is divided into 2 partitions:
 - > 1st: $8K(2^{13})$ segments (private), local descriptor table (LDT)
 - > 2nd: $8K(2^{13})$ segments (shared), global descriptor table (GDT)
- Logical address:
 - > max # of segments per process = $2^{14} = 16K$
 - > size of a segment $\leq 2^{32} = 4GB$



Intel Pentium Segmentation

- Segment descriptor
 - > Segment base address and length
 - > Access right and privileged level



國立清華大學

Intel Pentium Paging (Two-Level)

- Page size can be either 4KB or 4MB
- Each page directory entry has a flag for indication

Chapter8 Memory Management Operating System Concepts – NTHU LSA Lab 73

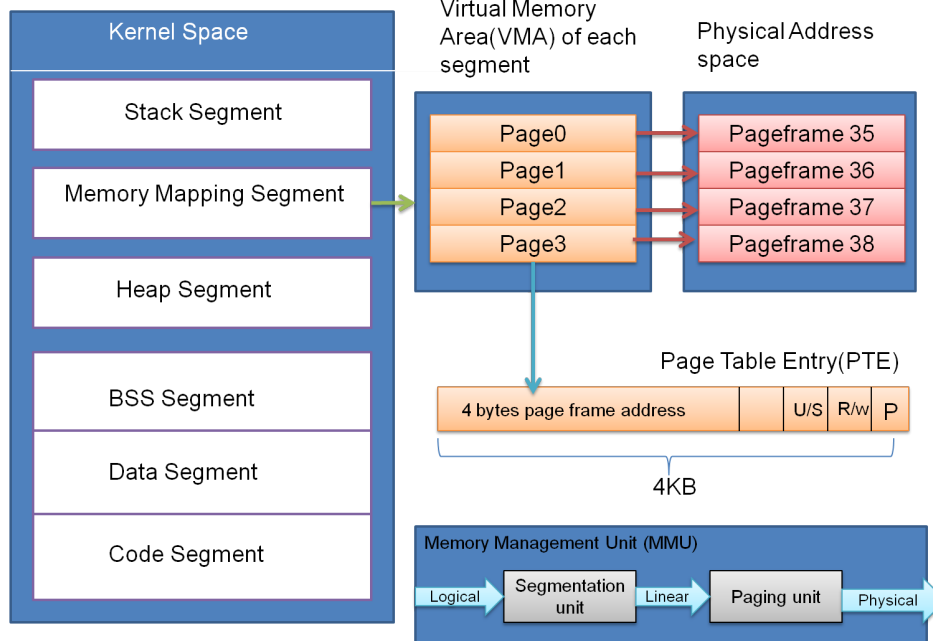
國立清華大學

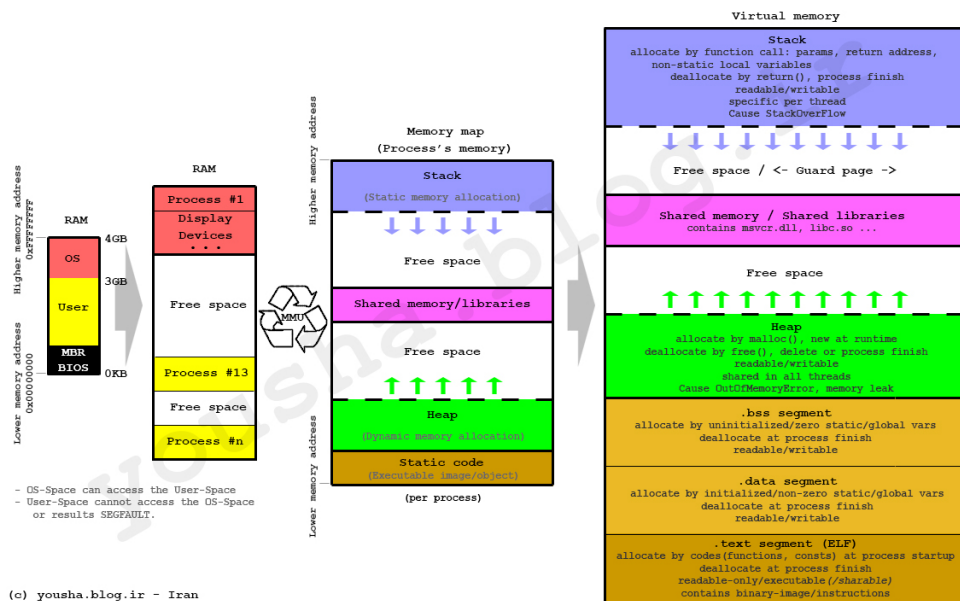
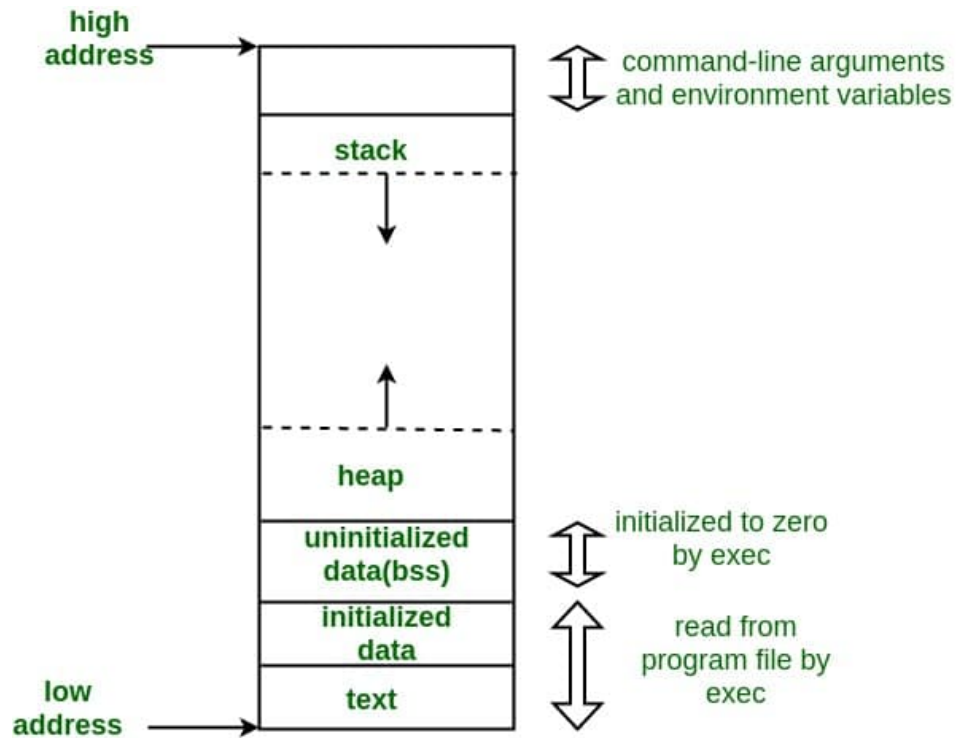
Example Question

- Let the physical mem size is 512B, the page size is 32B and the logical address of a program can have 8 segments. Given a 12 bits hexadecimal logical address "448", translate the addr. With below page and segment tables.
- linear addr:010111110, phy addr:001011110

Chapter9 Memory Management

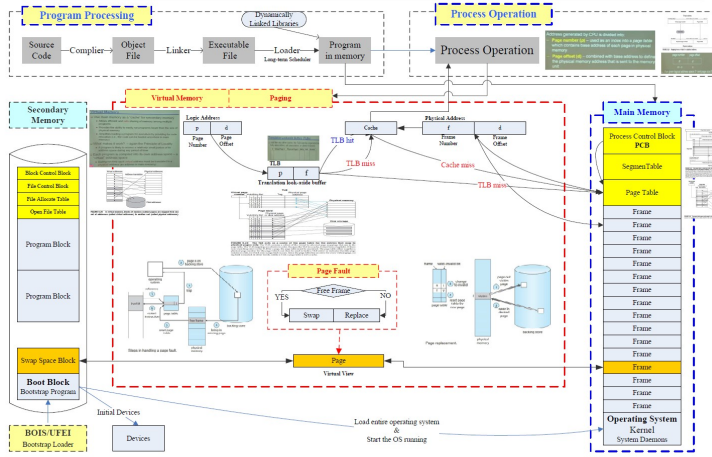
Process Virtual Memory Layout





Virtual Memory

Introduction to Operating System _ Virtual Memory _ ALL

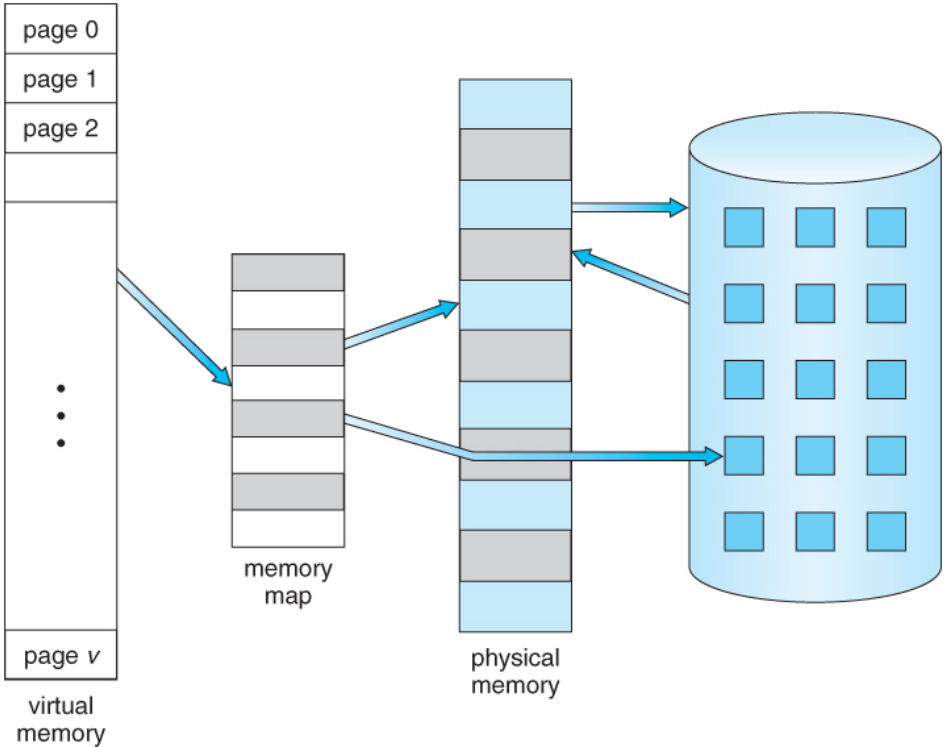


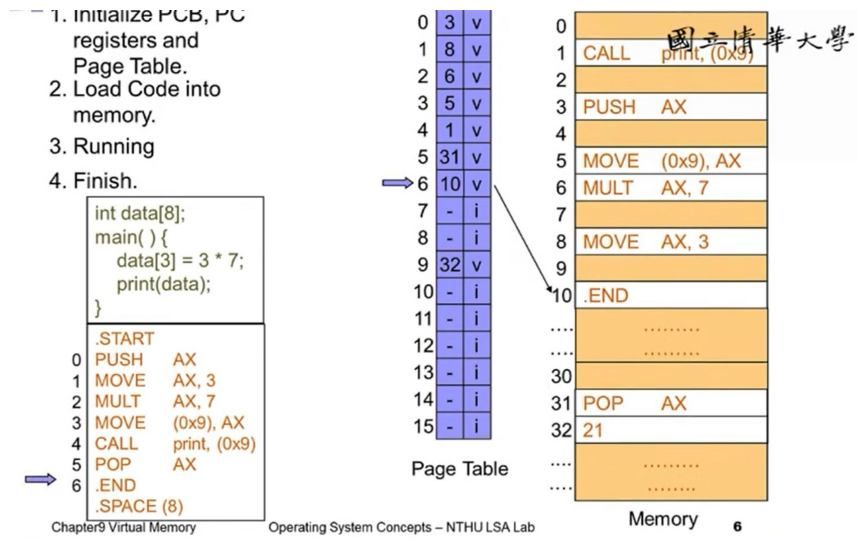
國立清華大學

Background

- **Virtual memory** – separation of user logical memory from physical memory
 - > To run an extremely **large process**
 - ◆ Logical address space can be much larger than physical address space
 - > To increase **CPU/resources utilization**
 - ◆ higher degree of multiprogramming degree
 - > To **simplify programming tasks**
 - ◆ Free programmer from memory limitation
 - > To run programs **faster**
 - ◆ less I/O would be needed to load or swap
- Virtual memory can be implemented via
 - > **Demand paging**
 - > Demand segmentation: more complicated due to variable sizes

Chapter9 Virtual Memory Operating System Concepts – NTHU LSA Lab 4





Demand Paging

國立清華大學

Demand Paging

- A page rather than the whole process is brought into memory only when it is needed
 - Less I/O needed → Faster response
 - Less memory needed → More users
- Page is needed when there is a reference to the page
 - Invalid reference → abort
 - Not-in-memory → bring to memory via paging
- **pure demand paging**
 - Start a process with no page
 - Never bring a page into memory until it is required

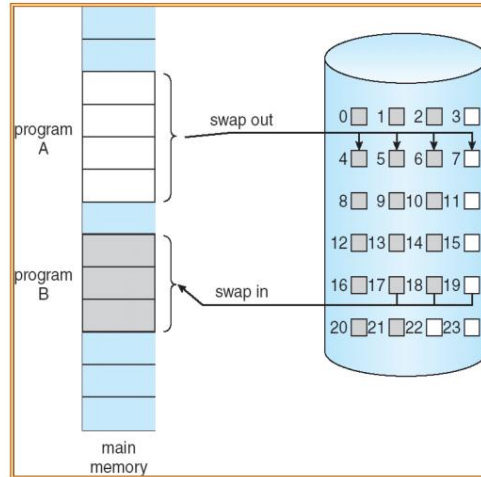
Demand Paging

國立清華大學

- A **swapper** (midterm scheduler) manipulates the entire **process**, whereas a **pager** is concerned with the individual pages of a process
- Hardware support
 - **Page Table: a valid-invalid bit**
 - ◆ 1 → page in memory
 - ◆ 0 → page not in memory
 - ◆ Initially, all such bits are set to 0
 - Secondary memory (swap space, **backing store**):
Usually, a high-speed disk (swap device) is use

Demand Paging

- **Demand paging:** pages are only loaded into memory when they are demanded during execution
 - Less I/O needed
 - Less memory needed
 - Higher degree of multiprogramming
 - Faster response
- **Pager (lazy swapper)** never swaps a page into memory unless that page will be needed.
- **An extreme case: Pure demand paging** starts a process with no pages in memory ...



Transfer of a Paged Memory to Contiguous Disk Space

Demand Paging Mechanisms

- PTE helps us implement demand paging
 - Valid \Rightarrow Page in memory, PTE points at physical page
 - Not Valid \Rightarrow Page not in memory; use info in PTE to find it on disk when necessary
- Suppose user references page with invalid PTE?
 - Memory Management Unit (MMU) traps to OS
 - » Resulting trap is a “Page Fault”
 - What does OS do on a Page Fault?:
 - » Choose an old page to replace
 - » If old page modified (“D=1”), write contents back to disk
 - » Change its PTE and any cached TLB to be invalid
 - » Load new page into memory from disk
 - » Update page table entry, invalidate TLB for new entry
 - » Continue thread from original faulting location
 - TLB for new page will be loaded when thread continued!
 - While pulling pages off disk for one process, OS runs another process from ready queue
 - » Suspended process sits on wait queue

Cache

Demand Paging Performance



Performance of Demand Paging

- Demand paging can significantly affect the performance of a computer system. Let's compute the effective access time for a demand-paged memory.
 - For most computer systems, the memory-access time (m_a) ranges from 10 to 200 nanoseconds.
 - As long as we have no page faults, the effective access time is equal to the memory access time.
 - If, however a page fault occurs, we must first read the relevant page from disk and then access the desired word.
- Page Fault Rate $0 \leq p \leq 1.0$
 - if $p = 0$ no page faults
 - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

$$\text{EAT} = (1 - p) \times \text{memory access} + p (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})$$



Performance of Demand Paging (Cont.)

- Three major activities
 - Service the interrupt – careful coding means just several hundred instructions needed
 - Read the page – lots of time
 - Restart the process – again just a small amount of time
- Page Fault Rate $0 \leq p \leq 1$
 - if $p = 0$ no page faults
 - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

$$\text{EAT} = (1 - p) \times \text{memory access} + p (\text{page fault overhead} + \text{swap page out} + \text{swap page in})$$





Demand Paging Performance Example

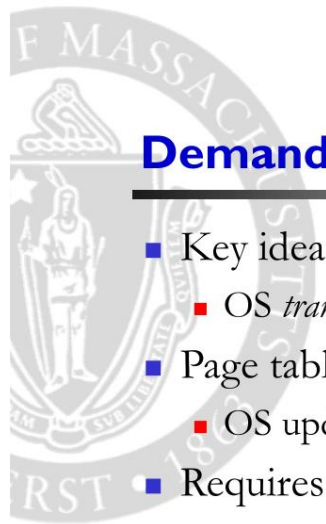
- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- $EAT = (1 - p) \times 200 + p (8 \text{ milliseconds})$
 $= (1 - p) \times 200 + p \times 8,000,000$
 $= 200 + p \times 7,999,800$
- If one access out of 1,000 causes a page fault, then
 $EAT = 8.2 \text{ microseconds.}$
 This is a slowdown by a factor of 40!!



Performance of Demand Paging

• Stages in Demand Paging

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
 1. Wait in a queue for this device until the read request is serviced
 2. Wait for the device seek and/or latency time
 3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction



Demand-Paged Virtual Memory

- Key idea: use **RAM** as cache for disk
 - OS *transparently* moves pages
- Page table: page on disk, in memory
 - OS updates whenever pages change state
- Requires locality:
 - **Working set** (pages referenced recently) must fit in memory
 - If not: **thrashing** (nothing but disk traffic)



Page Fault

Page Fault

國立清華大學

- First reference to a page will trap to OS
 - ➔ **page-fault trap**
- 1. OS looks at the internal table (in PCB) to decide
 - Invalid reference ➔ abort
 - Just not in memory ➔ continue
- 2. Get an empty frame
- 3. Swap the page from disk (swap space) into the frame
- 4. Reset page table, valid-invalid bit = 1
- 5. **Restart instruction**

Page Fault

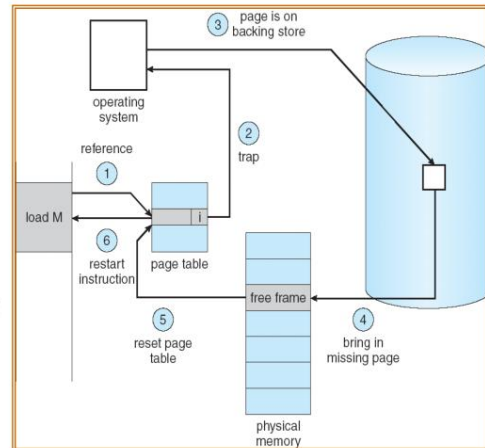
A reference to a page with valid bit set to 0 will trap to OS => page fault

OS looks at PCB to decide

- invalid reference => abort
- just no in memory
 - . Get free frame
 - . Swap into frame
 - . Reset tables

What if there is no free frame?

- evict a victim page in memory



Page Fault Handling – a different perspective

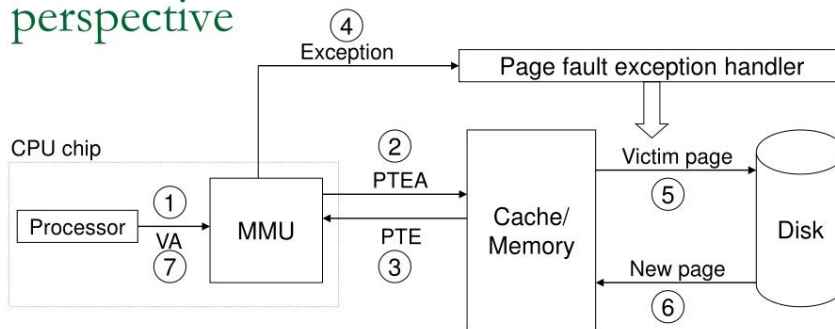


Fig. 10.14 (Bryant)

Process & Virtual Memory

國立清華大學

- **Demand Paging:** only bring in the page containing the first instruction
- **Copy-on-Write:** the parent and the child process share the same frames initially, and frame-copy when a page is written
- **Memory-Mapped File:** map a file into the virtual address space to bypass file system calls (e.g., read(), write())

The photograph shows a presentation slide titled "Copy-on-Write" from NTHU. The slide content is as follows:

Copy-on-Write

- Allow both the parent and the child process to share the same frames in memory
- If either process modifies a frame, only then a frame is copied
- COW allows efficient process creation (e.g., fork())
- Free frames are allocated from a pool of zeroed-out frames (security reason)
 - The content of a frame is erased to 0

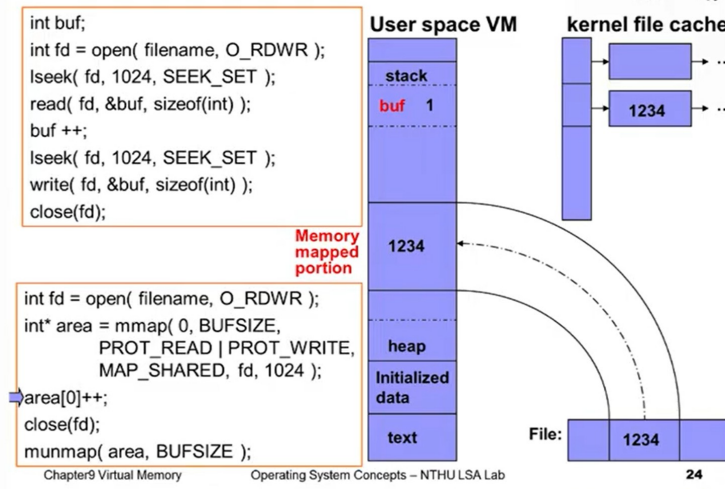
Chapter9 Virtual Memory Operating System Concepts – NTHU LSA Lab 19

The photograph shows a presentation slide titled "Memory-Mapped Files" from NTHU. The slide content is as follows:

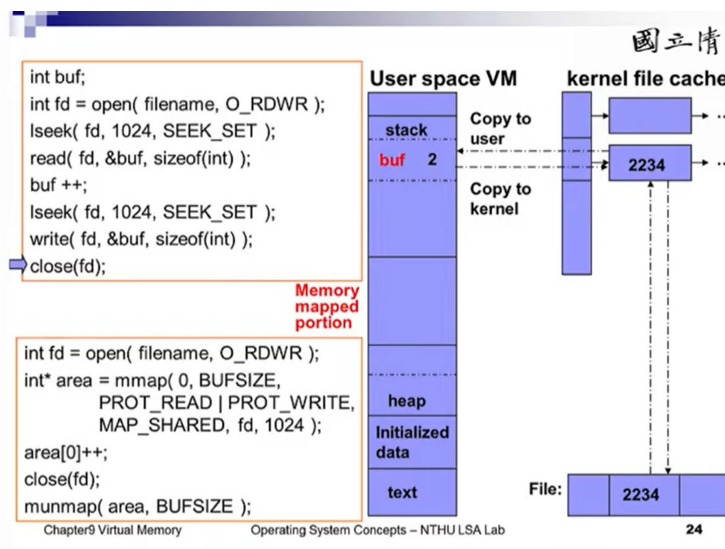
Memory-Mapped Files

- Approach:
 - MMF allows file I/O to be treated as routine memory access by mapping a disk block to a memory frame
 - A file is initially read using demand paging. Subsequent reads/writes to/from the file are treated as ordinary memory accesses
- Benefit:
 - Faster file access by using memory access rather than read() and write() system calls
 - Allows several processes to map the SAME file allowing the pages in memory to be SHARED
- Concerns:
 - Security(access control), data lost, more programming efforts

Chapter9 Virtual Memory Operating System Concepts – NTHU LSA Lab 22



File System Mechanism



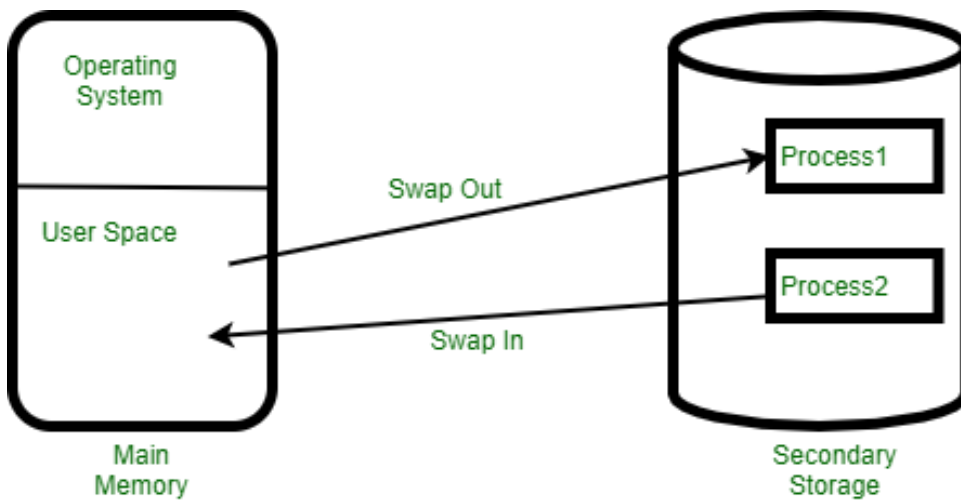
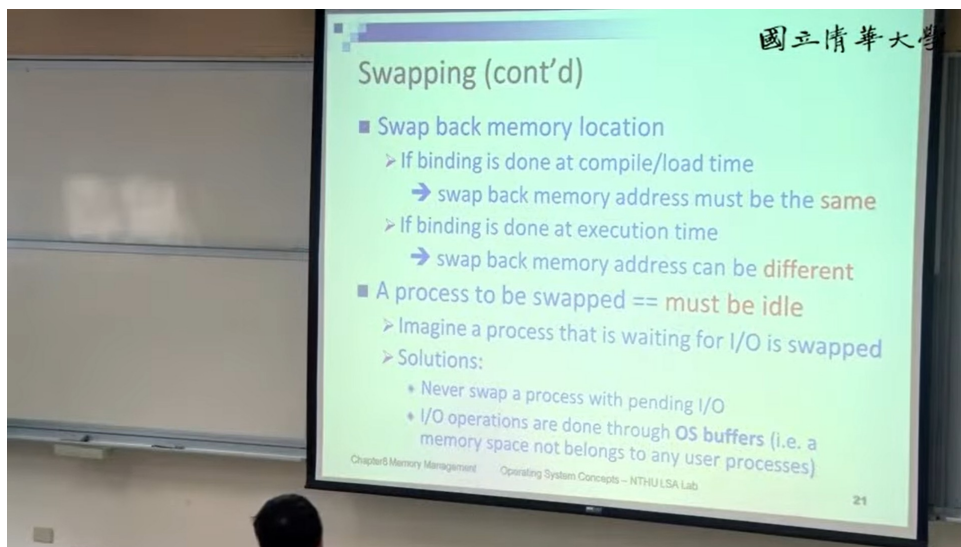
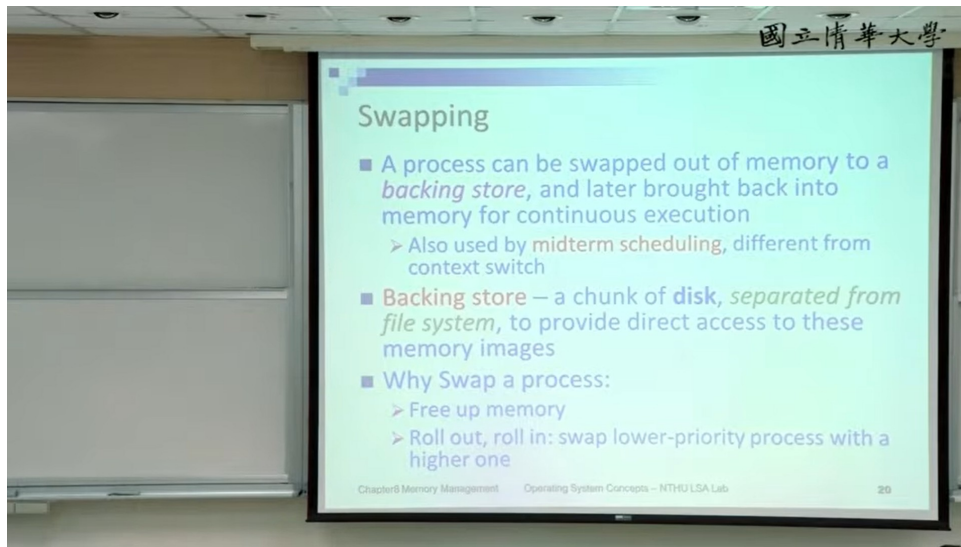
Example 1

Assume that a program has just referenced an address in virtual memory. Describe a scenario in which each of the following can occur. (If no such scenario can occur, explain why.)

- TLB miss with no page fault → Page is in memory but there is no page-no entry in TLB
- TLB miss and page fault → Page not in memory & no page-no entry in TLB
- TLB hit and no page fault → Page is in memory & page-no entry is in TLB
- TLB hit and page fault → Not possible. TLB contains most accessed page table entries, so if it's a TLB hit that means page is there in memory.

Dr. Ragini Karwayun

SWAP



Page Replacement

Page Replacement Concept

- When a page fault occurs with no free frame
 - swap out a process, freeing all its frames, or
 - page replacement: find one not currently used and free it
 - ◆ Use **dirty bit** to reduce overhead of page transfers – only modified pages are written to disk
- Solve two major problems for demand paging
 - frame-allocation algorithm:
 - ◆ Determine **how many frames** to be allocated to a process
 - page-replacement algorithm:
 - ◆ select **which frame** to be replaced

Page replacement algorithm

- * Page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated.
- * Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.

Need for Page Replacement

- Until there is not enough RAM to store all the data needed
- The process of obtaining an empty page frame does not involve removing another page from RAM.
- But If all page frames are non-empty, obtaining an empty page frame requires choosing a page frame containing data to empty.
- Efficient paging systems must determine the page frame to empty by choosing one that is least likely to be needed within a short time
- *Limited physical memory --> limited number of frame --> limited number of frame allocated to a process.*
- Thus we need various Page Replacement Algorithms

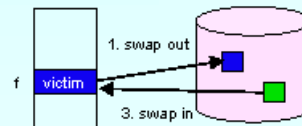
Page Replacement

- What happens if there is no free frame to allocate?
- Here comes page replacement - find some page in memory that is "not that used" and swap it out. A free frame was just created.
- Note that the same page may be brought into memory several times over the life of a process.
- Use a dirty bit to reduce overhead of page transfers - only modified pages need to be written to disk, unmodified pages can be discarded.
- Page replacement completes the separation between logical memory and physical memory. Large virtual memory can be provided on a smaller physical memory.

2. Modify to invalid

	i
f	v

4. Modify to f, valid



Yair Amir

Fall 00 / Lecture 6

12

Lecture Slides By Adil Aslam

FIFO Example

• Reference String is: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	7	7	2	2	2	4	4	4	0	0	0
	0	0	0	3	3	3	2	2	2	1	1
		1	1	1	0	0	0	3	3	3	2

1 is Present in table so hit the page

Page Fault : 1+1+1+1+1+1+1+1+1+1+1+1

Check the oldest page and replaced it. If it is not present in table

Belady's Anomaly

For FIFO algorithm, as the following counter-example shows, increasing m from 3 to 4 increases faults

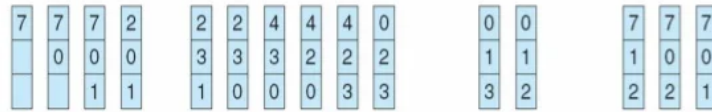
w	1 2 3 4 1 2 5 1 2 3 4 5	
m=3	1 2 3 4 1 2 5 5 5 3 4 4	9 page
	1 2 3 4 1 2 2 2 5 3 3	faults
	1 2 3 4 1 1 1 2 5 5	
m=4	1 2 3 4 4 4 5 1 2 3 4 5	10 page
	1 2 3 3 3 4 5 1 2 3 4	faults
	1 2 2 2 3 4 5 1 2 3	
	1 1 1 2 3 4 5 1 2	

36

FIFO Page Replacement

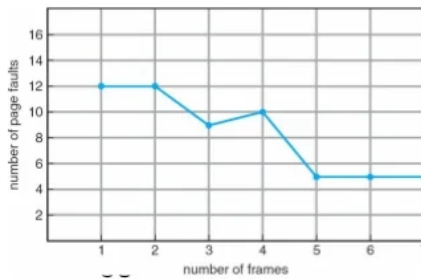
reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



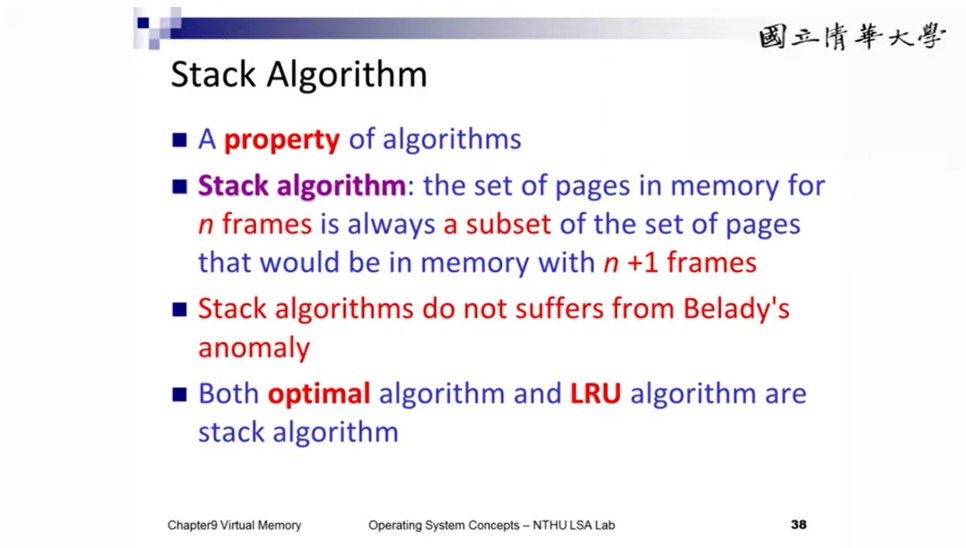
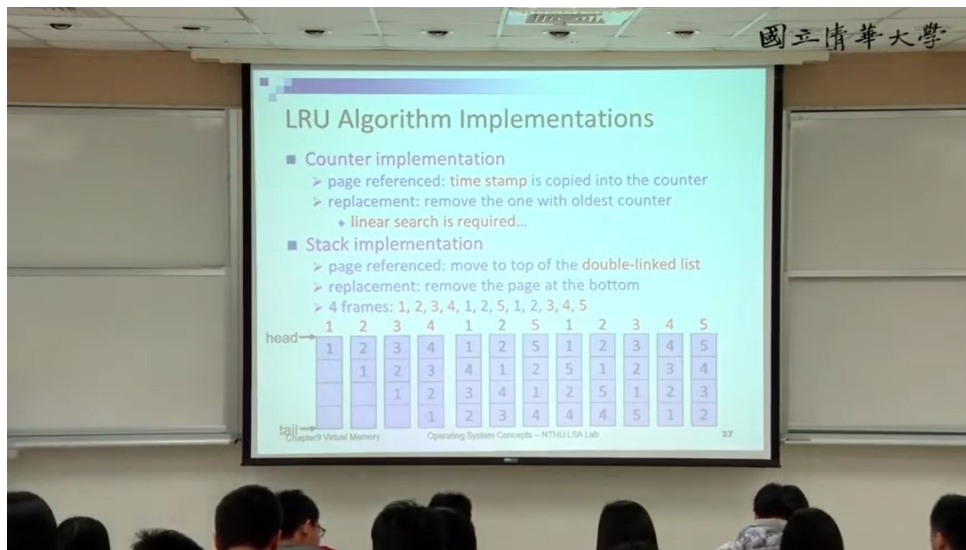
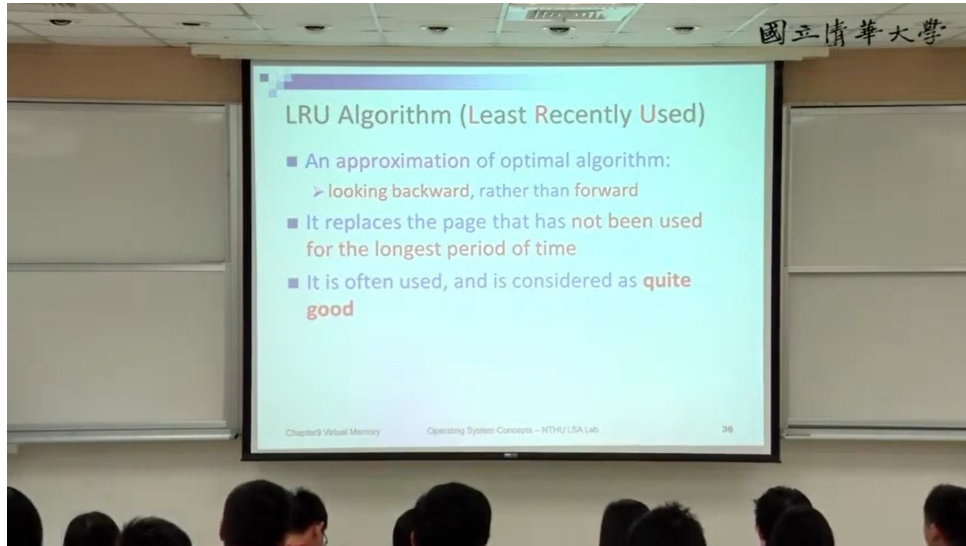
page frames

FIFO Illustrating Belady's Anomaly



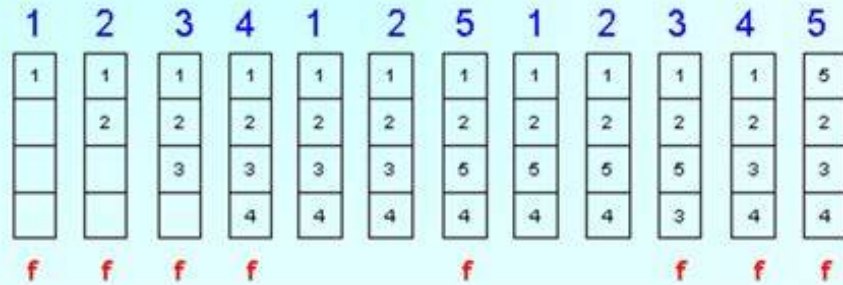
Unit-4

45



Least Recently Used (LRU)

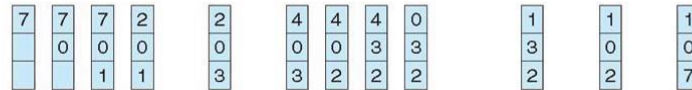
- Well, OPT is not possible...
- **Replace the page that has not been used for the longest period of time.**



LEAST RECENTLY USED (LRU) ALGORITHM

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



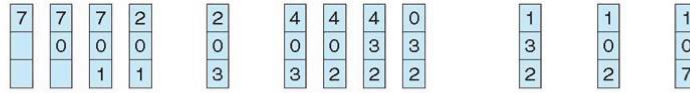
page frames

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

LEAST RECENTLY USED (LRU) ALGORITHM

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1	1	1				
	0	0	0		0		0	0	3	3			3	0	0				
		1	1		3		3	2	2	2			2	2	7				

2

Problem-05: Optimal Page Replacement Algorithms

Optimal is a type of cache algorithm used to manage memory within a computer. OS replaces the page that will not be used for the longest period of time in future.

A system uses 4-page frames for storing process pages in main memory. It uses the **Optimal** page replacement policy. Assume that the first 4 frames have references 6,7,3,4. What is the total number of page faults that will occur while processing the page reference string given below;

6,7,1,2,3,4,1,2,4,3

5,3,2,5,6,7,1,2,3,3

so calculate the hit ratio and miss ratio.

frames	6	7	3	4	6	7	1	2	3	4	1	2	4	3	5	3	2	5	6	7	1	2	3	3
1	6	6	6	6	6	6	6	2	2	2	2	2	2	2										
2		7	7	7	7	7	1	1	1	1	1	1	1	1										
3			3	3	3	3	3	3	3	3	3	3	3	3										
4				4	4	4	4	4	4	4	4	4	4	4										
PF	x	x	x	x	/	/	x	x	/	/	/	/	/	/										

= miss =

I

: hit =

Problem 3 (20 pts) Memory Management

a. (8 pts) Given the following page replacement algorithms: FIFO, LRU and Belady answer each of the following questions. Must Explain each answer for full credit.

- 1 Which algorithm that is applicable and hardest to implement in practice?
2. Which algorithm that is applicable and easiest to implement in practice.
- 3.. Which algorithm might behave worse when given more physical frames?
4. . Which algorithm is not applicable as it is?

b. (7 pts) Consider a paging system with a virtual memory size 4 Gbytes. Let the page size be 4 Kbytes, and assume the physical memory size be 64 Mbytes. Each entry in the page table has one Modify bit, and two protection bits.

1. What is the maximum memory needed to store the one level page table in bits.?
2. If two level page tables were to be used how many secondary page tables we will have if the master page table is stored in one frame? Assume each entry in the master page table 4 bytes.

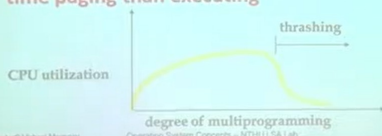
c. (5 pts) Name one advantage and one disadvantage to two level page table compared to 1 level page table then explain which one is the best to implement in practice and why?

Trashing

國立清華大學

Definition of Thrashing

- If a process does not have “enough” **frames**
 - the process does not have # frames it needs to support pages in active use
 - ➔ Very high paging activity
- A process is **thrashing** if it is **spending more time paging than executing**



CPU utilization

degree of multiprogramming

thrashing

Chapter9 Virtual Memory Operating System Concepts – NTHU LSA Lab 47

國立清華大學

Thrashing

- Performance problem caused by thrashing (Assume global replacement is used)
 - processes queued for I/O to swap (page fault)
 - ➔ low CPU utilization
 - ➔ OS increases the degree of multiprogramming
 - ➔ new processes take frames from old processes
 - ➔ more page faults and thus more I/O
 - ➔ CPU utilization drops even further
- To prevent thrashing, must provide enough frames for each process:
 - **Working-set model, Page-fault frequency**

Chapter9 Virtual Memory Operating System Concepts – NTHU LSA Lab 48

國立清華大學

Working-Set Model

- **Locality**: a set of pages that are actively used together
- Locality model: as a process executes, it moves from locality to locality
 - program structure (subroutine, loop, stack)
 - data structure (array, table)
- **Working-set model** (based on locality model)
 - working-set **window**: a parameter Δ (delta)
 - working set: set of pages in most recent Δ page references (**an approximation locality**)

Chapter9 Virtual Memory Operating System Concepts – NTHU LSA Lab 49

Working-Set Example

■ If $\Delta = 10$:

page reference table

... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 4 1 3 2 3 4 4 4 3 4 4 ...

$WS(t_1) = \{1, 2, 5, 6, 7\}$ $WS(t_2) = \{3, 4\}$

Chapter 9 Virtual Memory Operating System Concepts – NTHU LSA Lab 50

Working-Set Model

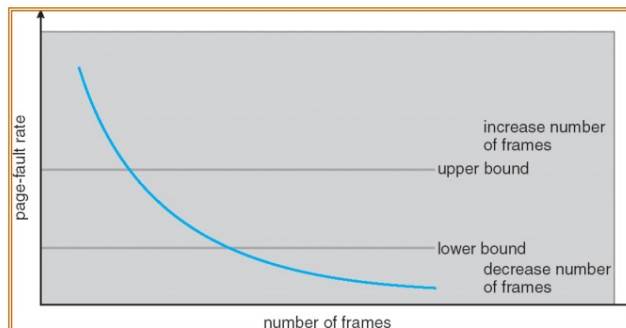
- Prevent thrashing using the working-set size
 - > WSS_i : working-set size for process i
 - > $D = \sum WSS_i$ (total demand frames)
 - > If $D > m$ (available frames) \Rightarrow thrashing
 - > The OS monitors the WSS_i of each process and allocates to the process enough frames
 - if $D \ll m$, increase degree of MP
 - if $D > m$, suspend a process
- ⊙ 1. prevent thrashing while keeping the degree of multiprogramming as high as possible
- ⊙ 2. optimize CPU utilization
- ⊙ : too expensive for tracking

Chapter 9 Virtual Memory Operating System Concepts – NTHU LSA Lab 51



Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
 - If actual rate too low, process loses frame
 - If actual rate too high, process gains frame



Thrashing Diagram

- Why does paging work?
 - Locality model
 - Process migrates from one locality to another.
 - Localities may overlap.
- Why does thrashing occur?
 - Σ size of locality > total **allocated** memory size
- **High degree of MP results in allocation for a process to get too small and perhaps lose its "working set" – see below**

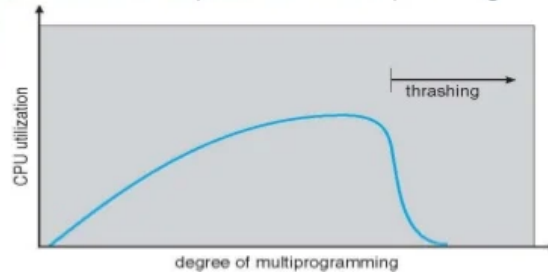
Applied Operating System Concepts

10.41

Silberschatz, Galvin, and Gagne ©1999

Cause of Thrashing

- All processes now queue up for the paging device, the ready queue empties. => CPU utilization decreases.
- CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
- Result: more page faults, longer queue for the paging device , CPU utilization drops further, ...
- Thrashing has occurred, and system throughput plunges, the pagefault rate increases tremendously. As a result, the effective memory-access time increases.
- No work is getting done, because the processes are spending all their time in paging.
- To prevent Thrashing, we must provide a process with as many frames as it needs.
- how many frames ?



Introduction to Operating Systems

Working Sets and Page Fault Rates

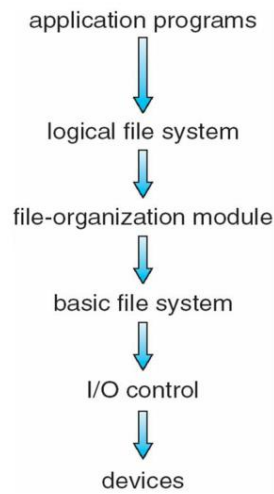
- Memory has locality property
- When the process moves to a new WS, the PF rate rises toward a peak

Chapter 9 Virtual Memory Operating System Concepts – NTHU, USA Lab 54

File Management



Layered File System

Operating System Concepts – 8th Edition

11.5

Silberschatz, Galvin and Gagne ©2009



File System Layers

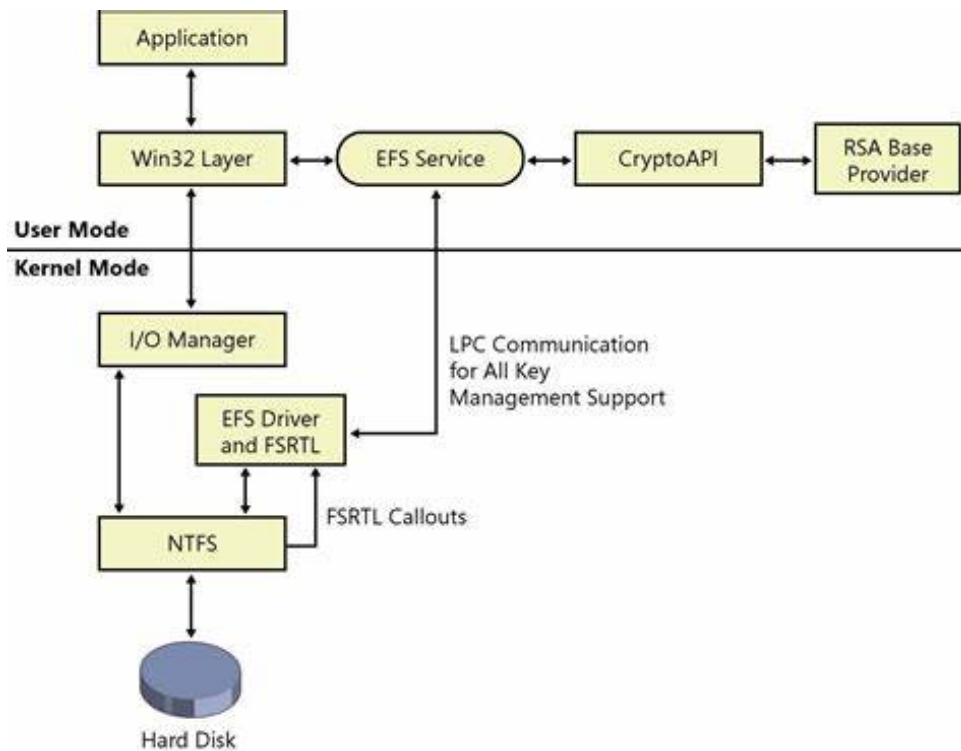
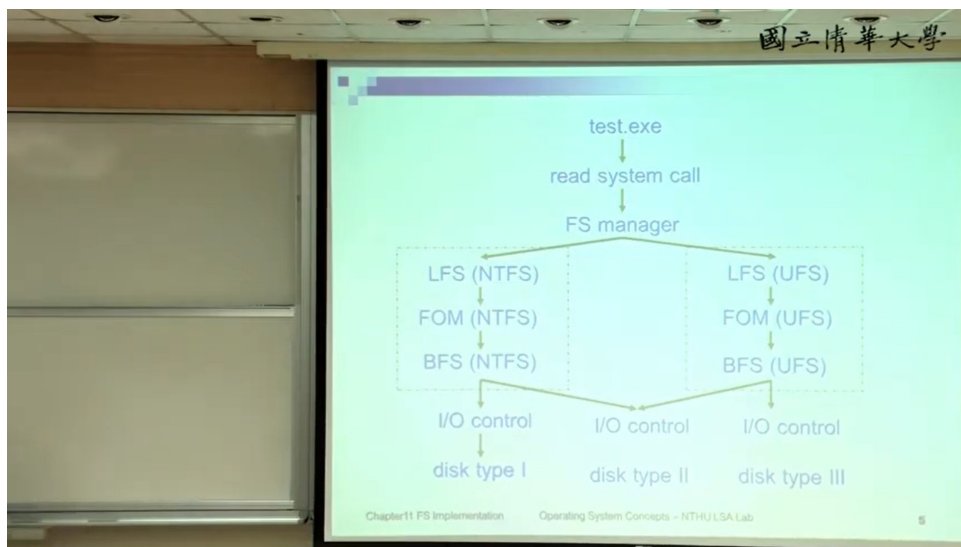
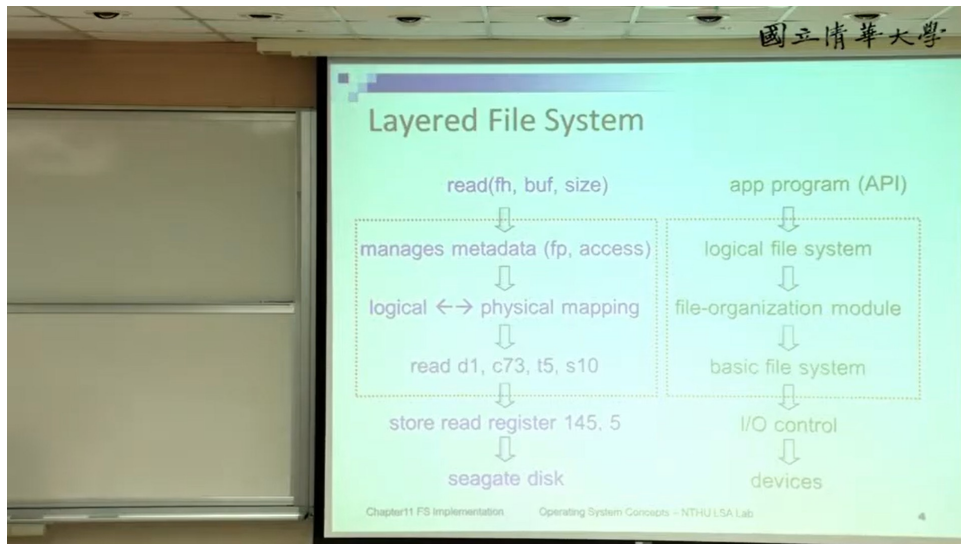
- **I/O control** manages I/O devices at the I/O control layer
 - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **Basic file system:** given command like "retrieve block 123" translates to device driver
 - Also manages memory buffers and caches (allocation, freeing, replacement)
 - Buffers hold data in transit
 - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
 - Translates logical block # to physical block #
 - Manages free space, disk allocation
- **Logical file system** manages metadata information
 - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in Unix)
 - Directory management
 - Protection

Operating System Concepts – 8th Edition

11.5

Silberschatz, Galvin and Gagne ©2009





File Attributes

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

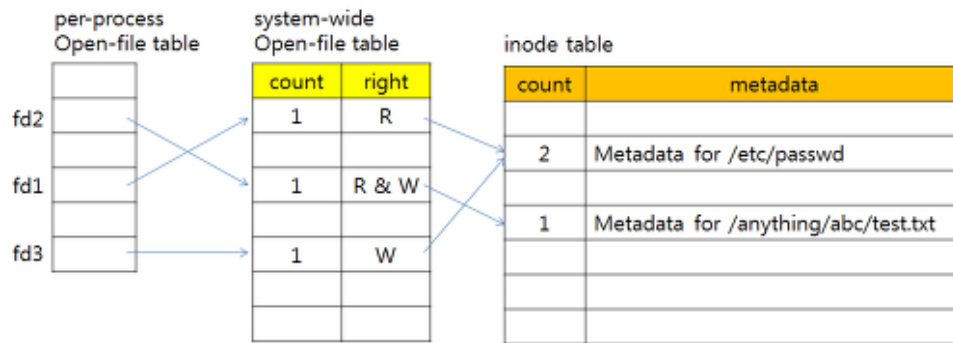
- ▶ **Name** – only information kept in human-readable form
- ▶ **Identifier** – unique tag (number) identifies file within file system
- ▶ **Type** – needed for systems that support different types
- ▶ **Location** – pointer to file location on device


- ▶ **Size** – current file size
- ▶ **Protection** – controls who can do reading, writing, executing
- ▶ **Time, date, and user identification** – data for protection, security, and usage monitoring

- ▶ Information about files are kept in the **directory structure**, which is maintained on the disk. Typically, a directory entry consists of the file's name and its unique identifier.

SHASHI KS

File Operations

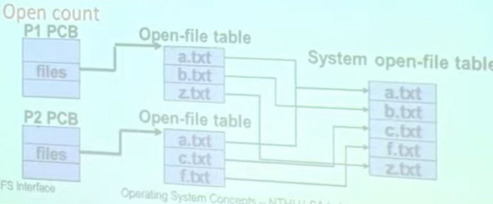




Open-File Tables

國立清華大學

- **Per-process table**
 - ▶ Tracking all files opened by this process
 - ▶ Current **file pointer** for each opened file
 - ▶ Access rights and accounting information
- **System-wide table**
 - ▶ Each entry in the per-process table points to this table
 - ▶ **Process-independent** information such as disk location, access dates, file size
 - ▶ Open count



Chapter10 FS Interface Operating System Concepts – NTHU LSA Lab

File operation	Declaration & Description
<p>fopen() - To open a file</p>	<p>Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=fopen ("filename", "mode"); Where, fp - file pointer to the data type "FILE". filename - the actual file name with full path of the file. mode - refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.</p>
<p>fclose() - To close a file</p>	<p>Declaration: int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. fclose (fp);</p>
<p>fgets() - To read a file</p>	<p>Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. fgets (buffer, size, fp); where, buffer - buffer to put the data in. size - size of the buffer fp - file pointer</p>
<p>fprintf() - To write into a file</p>	<p>Declaration: int fprintf(FILE *fp, const char *format, ...); fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);</p>

File operation	Declaration & Description
<p>fopen() - To open a file</p>	<p>Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=fopen ("filename", "mode"); Where, fp - file pointer to the data type "FILE". filename - the actual file name with full path of the file. mode - refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.</p>
<p>fclose() - To close a file</p>	<p>Declaration: int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. fclose (fp);</p>
<p>fgets() - To read a file</p>	<p>Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. fgets (buffer, size, fp); where, buffer - buffer to put the data in. size - size of the buffer fp - file pointer</p>
<p>fprintf() - To write into a file</p>	<p>Declaration: int fprintf(FILE *fp, const char *format, ...); fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);</p>

File operation	Declaration & Description
fopen() - To open a file	<p>Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist.</p> <pre>FILE *fp; fp=fopen ("filename", "'mode");</pre> <p>Where, fp - file pointer to the data type "FILE". filename - the actual file name with full path of the file. mode - refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.</p>
fclose() - To close a file	<p>Declaration: int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below.</p> <pre>fclose (fp);</pre>
fgets() - To read a file	<p>Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below.</p> <pre>fgets (buffer, size, fp);</pre> <p>where, buffer - buffer to put the data in. size - size of the buffer fp - file pointer</p>
fprintf() - To write into a file	<p>Declaration: int fprintf(FILE *fp, const char *format, ...); fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);</p>

File Type

File type	Usual extension	Function
Executable	exe,com,bin	Read to run machine language program
Object	obj,o	Compiled,machine language not linked
Source code	C,java,pas,asm,a	Source code in various languages
Batch	bat,sh	Commands to the command interpreter
Text	txt,doc	Textual data,documents
Word processor	Wp,tex,rrf,doc	Various word processor formats
Archive	arc,zip,tar	Related files grouped into one file compressed

File Access

Access Methods

Sequential access

- > Read/write next (block)
- > Reset: repositioning the file pointer to the beginning
- > Skip/rewind *n* records



Access Methods

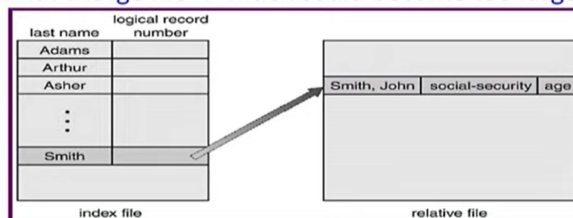
Direct (relative) access

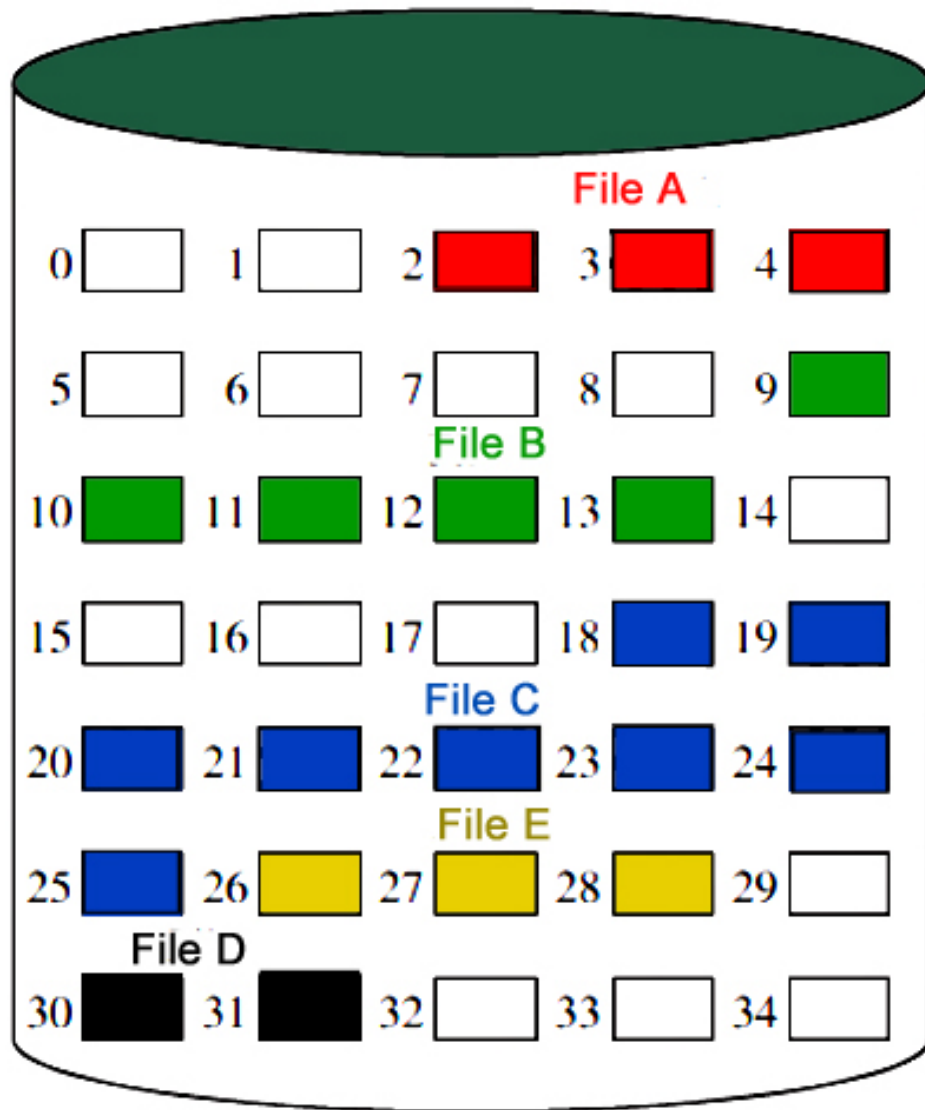
- > Access an element at an arbitrary position in a sequence
- > File operations include the **block #** as parameter
- > Often use **random access** to refer the access pattern from direct access

sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp+1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp+1;</i>

Index Access Methods

- Index: contains pointers to **blocks of a file**
- To find a record in a file:
 - > search the index file → find the pointer
 - > use the pointer to directly access the record
- With a large file → index could become too large

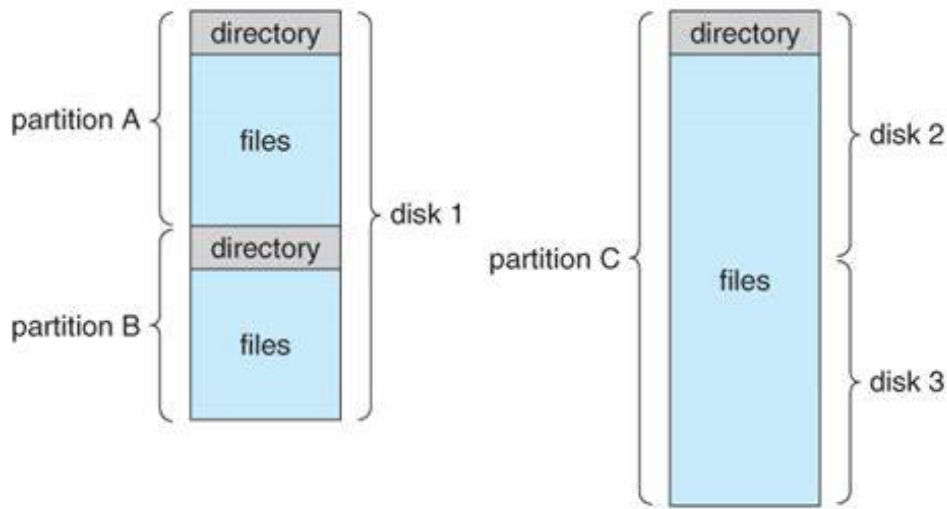




File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

File System Organization



國立清華大學

Tree-Structured Directory

- **Absolute path:** starting from the root
- **Relative path:** starting from a directory

Chapter10 FS Interface Operating System Concepts - NTHU/LSA Lab 19

國立清華大學

Acyclic-Graph Directory

- Use links to share files or directories
 - > UNIX-like: **symbolic link** (ln -s /spell/count /dict/count)
- A file can have multiple **absolute paths**
- When does a file actually get deleted?
 - > deleting the link but not the file
 - > deleting the file but leaves the link → **dangling pointer**

Chapter10 FS 20

File System Mounting and Sharing and Protection

國立清華大學

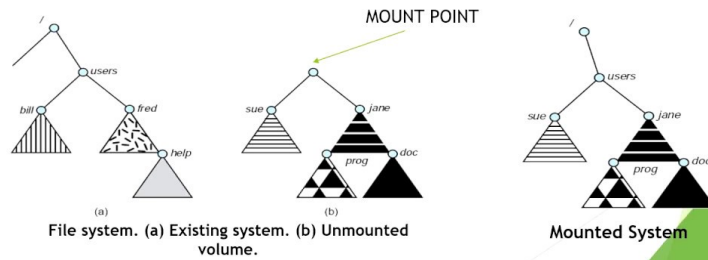
File System Mounting

- A file system must be mounted before it can be accessed
- **Mount point:** the root path that a FS will be mounted to
- Mount timing:
 - boot time
 - automatically at run-time
 - manually at run-time

Chapter 10 FS Interface
Operating System Concepts - NTHU USA Lab
25

FILE SYSTEM MOUNTING

- ▶ As a file must be *opened* before it is used, a file system must be *mounted* before it can be available to processes on the system.
- ▶ The operating system is given the name of the device and the **mount point**—the location within the file structure where the file system is to be attached. Typically, a mount point is an empty directory.



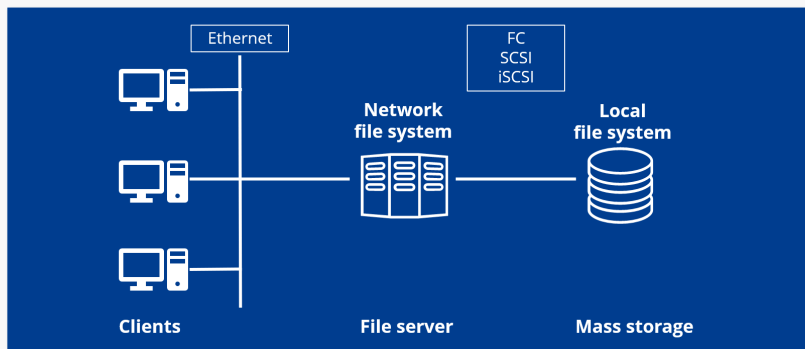
ITFT COLLEGE - CHANDIGARH
www.iff.edu.in | 1800 180 2424

File Sharing



- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a **network**
- Network File System (NFS) is a common distributed file-sharing method

File server



IONOS

國立清華大學

File Sharing on Multiple Users

- Each user: (userID, groupID)
 - > ID is associated with every ops/process/thread the user issues
- Each file has 3 sets of attributes
 - > owner, group, others
- Owner attributes describe the privileges for the owner of the file
 - > same for group/others attributes
 - > group/others attributes are set by owner or root

The diagram shows a large blue oval representing the set of all users. Inside, there are three yellow ovals representing 'group1', 'group2', and 'group3'. A red dot labeled 'Owner' is located within 'group1'. A red dot labeled 'Other users' is located outside all groups. A red dot labeled 'Group user' is located within 'group2'.

Chapter10 FS Interface Operating System Concepts – NTHU LSA Lab 28

國立清華大學

Access-Control List

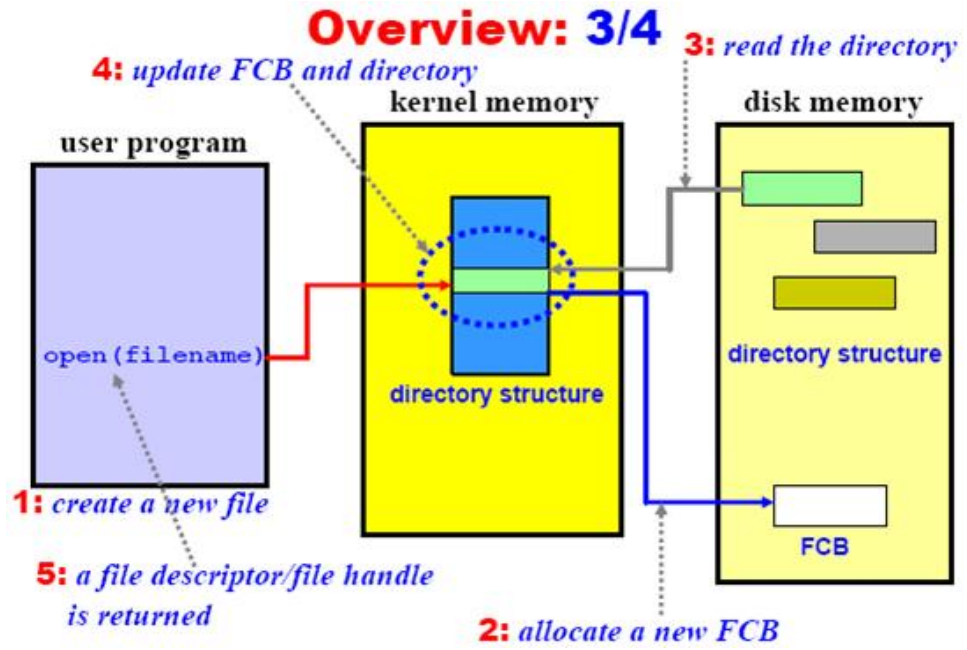
- We can create an **access-control list (ACL)** for each use
 - > check requested file access against ACL
 - > problem: unlimited # of users
- 3 classes of users → 3 ACL (**RWX**) for each file
 - > owner (e.g. 7 = RWX = 111)
 - > group (e.g. 6 = RWX = 110)
 - > public (others) (e.g. 4 = RWX = 100)

```

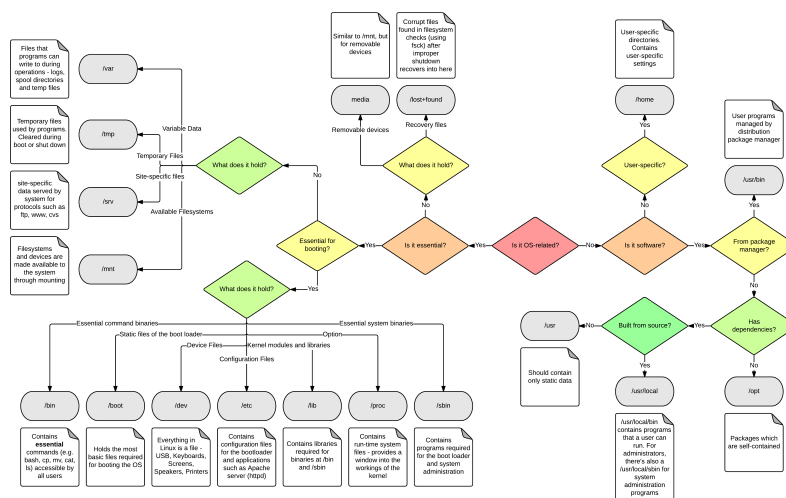
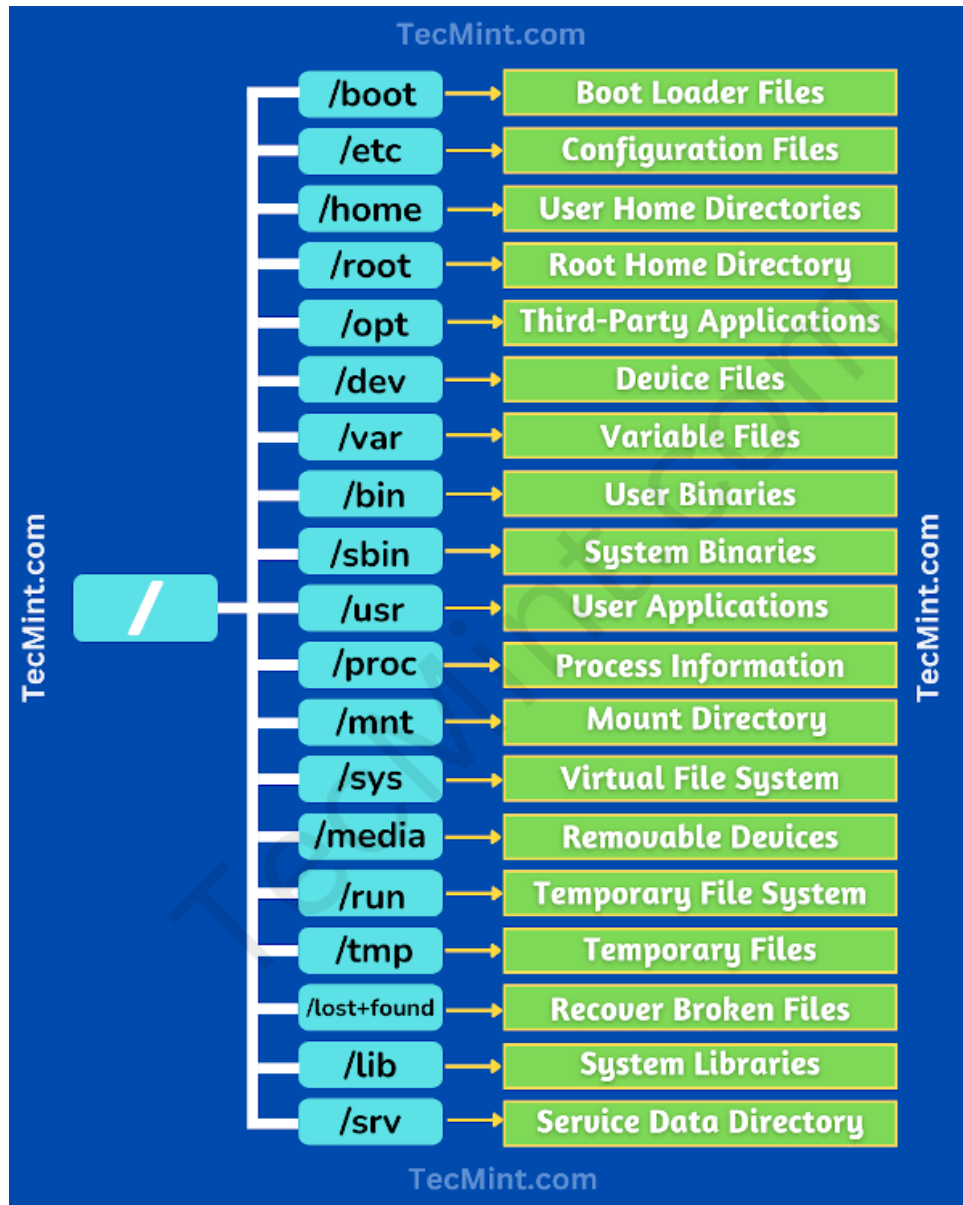
-rw-rw-r-- 1 pbg staff 31200 Sep 3 08:30 intro.ps
drwx----- 5 pbg staff 512 Jul 8 09:33 private/
drwxrwxr-x 2 pbg staff 512 Jul 8 09:35 doc/
drwxrwx--- 2 pbg student 512 Aug 3 14:13 student-proj/
-rw-r--r-- 1 pbg staff 9423 Feb 24 2003 program.c
-rwxr-xr-x 1 pbg staff 20471 Feb 24 2003 program
drwx--x--x 4 pbg faculty 512 Jul 31 10:31 lib/
drwx----- 3 pbg staff 1024 Aug 29 06:52 mail/
drwxrwxrwx 3 pbg staff 512 Jul 8 09:35 test/
    
```

chmod 664 intro.p

Chapter10 FS Interface Operating System Concepts – NTHU LSA Lab 29



File Structure in Linux



Understanding The Linux File Permissions

Column: 1 2 3 4

- rwx rwx rwx

Directory/File/Link
Information

User Rights

Group Rights

Others Rights

While the first column defines a directory, file or link, the next 3 columns (2, 3, 4) define the permissions for the User, Group and Others (everyone else) groups.

Linux Permissions Made Easy

-	user	group	everyone	
-	rwx	rwx	rwx	
	4 2 1	4 2 1	4 2 1	← decimal notification
	1 1 1	1 1 1	1 1 1	← binary notification
	7	7	7	

decimal notification: add each number to obtain the value (4 + 2 + 1 = 7)
binary notification: convert it to decimal then you should have the value (r-x = 101 base 2 = 5 base 10)

File type Permission classes

User Group Other

d rwx rwx rwx

Read

Write

Execute

rwx

- rwx rw- r--

Read, write and execute permissions for all other users

Read, write and execute permissions for members of the group owning the file

Read, write and execute permissions for the owner of the file

File type: "-" means a file. "d" means a directory.

Read, write and execute permissions for all other users

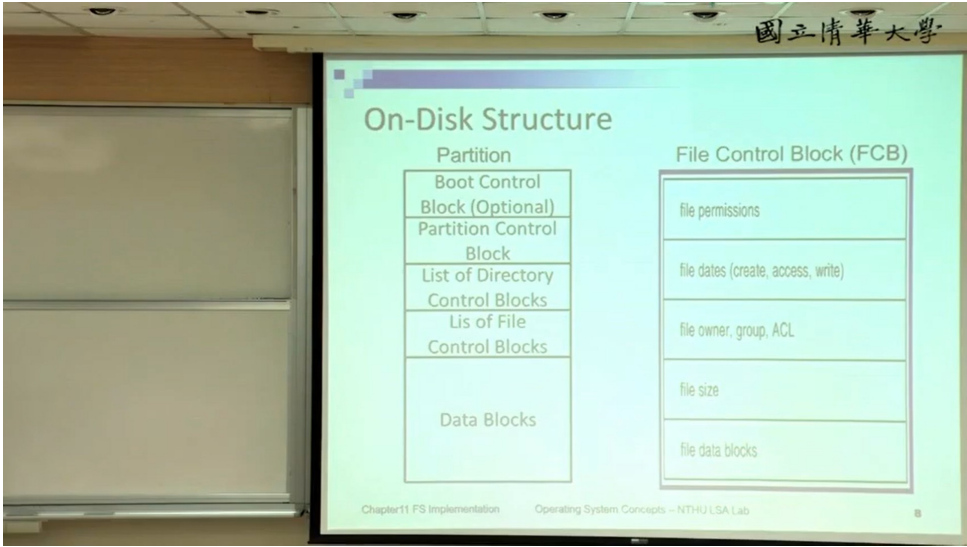
Read, write and execute permissions for members of the group owning the file

Read, write and execute permissions for the owner of the file

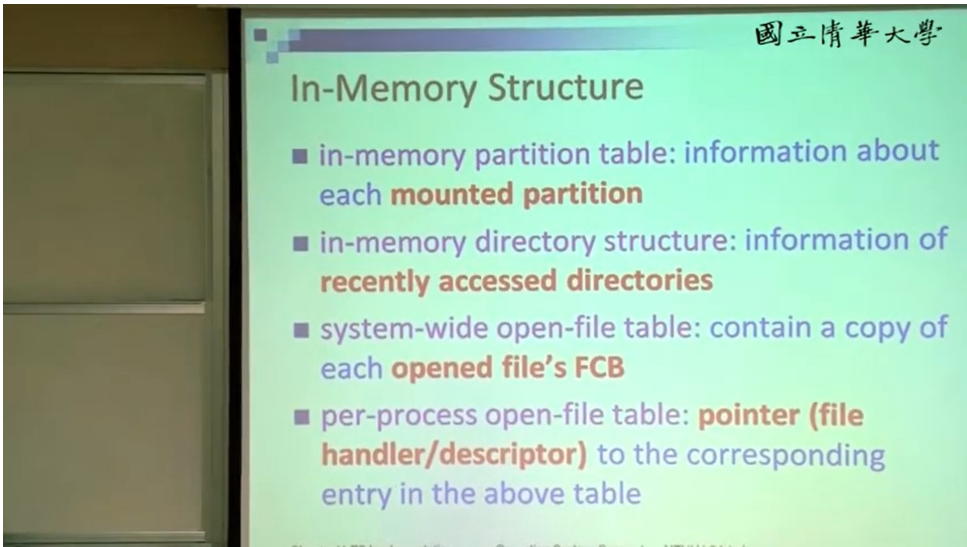
On-Disk Structure

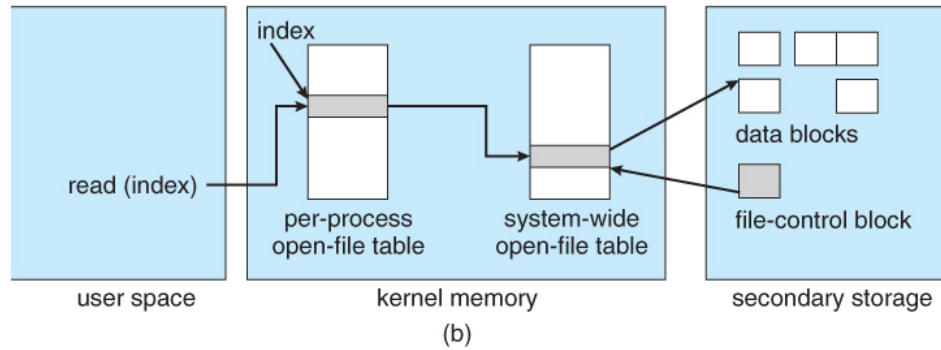
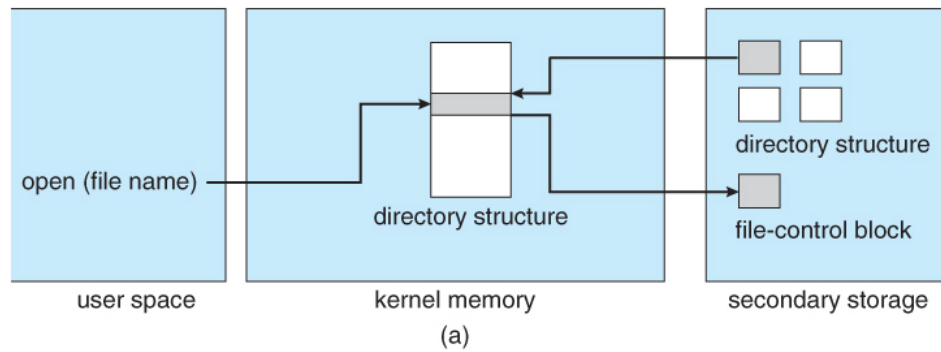
On-Disk Structure

- **Boot control block (per partition):** information needed to boot an OS from that partition
 - > typical the **first block of the partition (empty means no OS)**
 - > UFS (Unix File Sys.): **boot block**, NTFS: partition boot sector
- **Partition control block (per partition):** partition details
 - > details: # of blocks, block size, free-block-list, **free FCB pointers**, etc
 - > UFS: **superblock**, NTFS: Master File Table
- **File control block (per file):** details regarding a file
 - > details: permissions, size, **location of data blocks**
 - > UFS: **inode**, NTFS: stored in MFT (relational database)
- **Directory structure (per file system):** organize files



In-Memory Structure



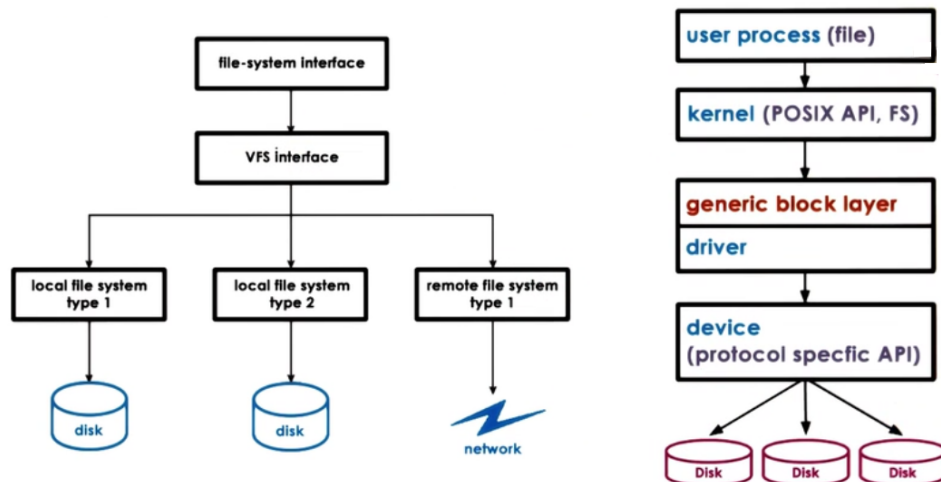


File Creation Procedure

國立清華大學

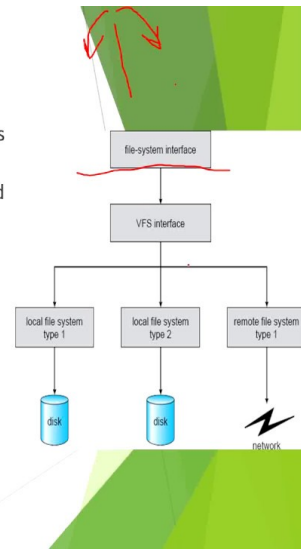
1. OS allocates a new **FCB**
2. Update **directory structure**
 1. OS reads in the corresponding **directory structure into memory**
 2. **Updates the dir structure with the new file name and the FCB**
 3. (After file being closed), OS **writes back the directory structure back to disk**
3. The file appears in user's dir command

Virtual File System



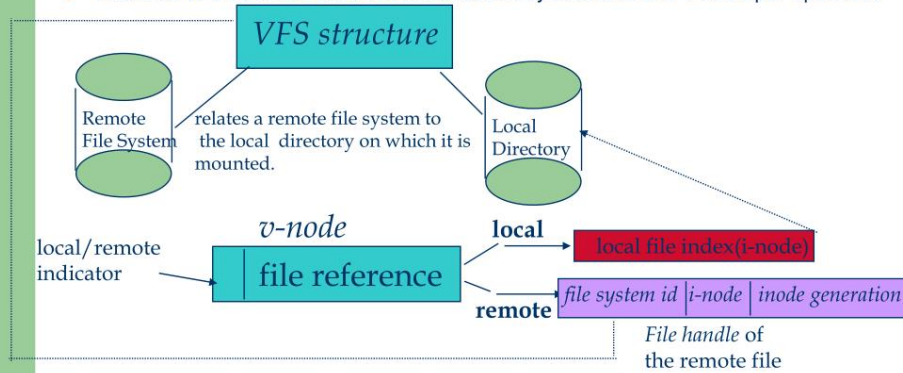
VIRTUAL FILE SYSTEMS

- ▶ General method of implementing multiple types of file systems is to write directory and file routines for each type.
- ▶ But, most operating systems, including UNIX, use object-oriented techniques to simplify, organize, and modularize the implementation.
- ▶ The first layer is the file-system interface, based on the `open()`, `read()`, `write()`, and `close()` calls and on **file descriptors**.
- ▶ The second layer, the virtual file system (VFS) layer, serves two important functions:
 1. It separates file-system-generic operations from their implementation by defining a clean VFS interface.
 2. The VFS provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a **vnode** (unique id for network wide files).



Virtual File System layer (VFS)

- Has one VFS structure for each mounted file system and one v-node per open file.



Virtual File System

國立清華大學


- Four main object types defined by Linux VFS:
 - > inode → an individual file
 - > file object → an open file
 - > superblock object → an entire file system
 - > dentry object → an individual directory entry
- VFS defines a set of operations that must be implemented (e.g. for file object)
 - > `int open(...)` → open a file
 - > `ssize_t read()` → read from a file

Directory Implementation 國立清華大學

- Linear lists
 - list of file names with pointers to data blocks
 - easy to program but poor performance
 - ◆ insertion, deletion, searching
- Hash table – linear list w/ hash data structure
 - constant time for searching
 - linked list for collisions on a hash entry
 - hash table usually has fixed # of entries

Chapter11 FS Implementation Operating System Concepts – NTHU LSA Lab 14

Allocation Methods

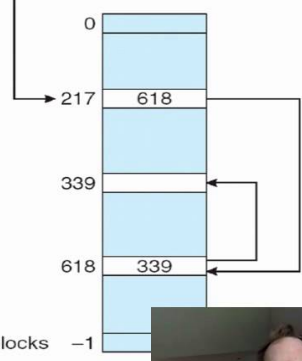


File-Allocation Table (FAT)

directory entry

test	...	217
name		start block

- First portion of blocks of each partition are used to store the FAT.
- As many entries in the FAT as there are data blocks in the partition.
- The FAT stores the pointers that would be stored in each block in a pure linked scheme
- Not as fragile as linked scheme, provided the FAT is protected/backed-up
- Was used in MS-DOS

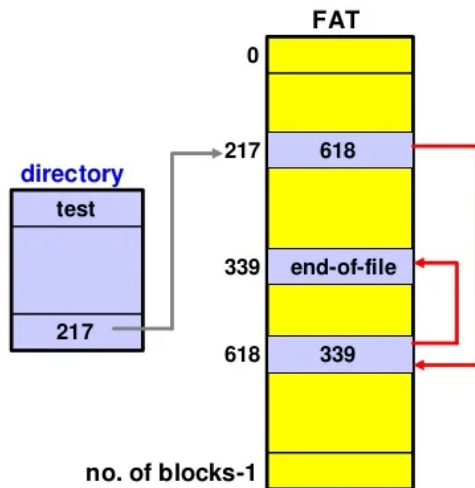


no. of disk blocks

Operating System Concepts – 9th Edition 12.25 Silberschatz, Galvin



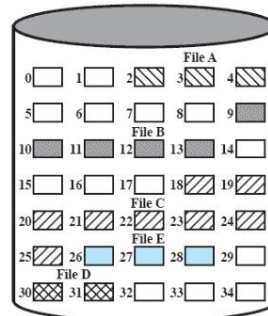
File Allocation Table (FAT)



- This is a variation of the linked allocation by pulling all pointers into a table, the **file allocation table (FAT)**.
- Large no. of disk seeks.
- Can do direct access.
- FAT needs space.
- The left diagram shows file **test** has its first block at **217**, followed by **618, 339** (end of file).
- What if FAT is damaged? We all know it well!

Contiguous File Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation
- Preallocation strategy using variable-size portions
- Is the best from the point of view of the individual sequential file

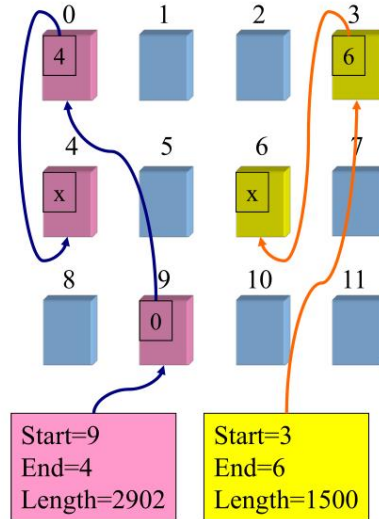


File Allocation Table		
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.9 Contiguous File Allocation

Linked allocation

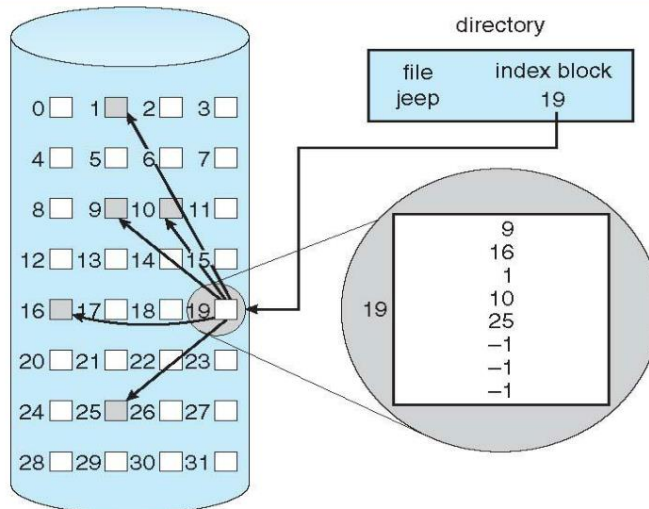
- File is a linked list of disk blocks
 - Blocks may be **scattered** around the disk drive
 - Block contains **both pointer to next block and data**
 - Files may be as long as needed
- New blocks are **allocated as needed**
 - Linked into list of blocks in file
 - Removed from list (bitmap) of free blocks



32



Example of Indexed Allocation



Operating System Concepts – 8th Edition

11.22

Silberschatz, Galvin and Gagne ©2009

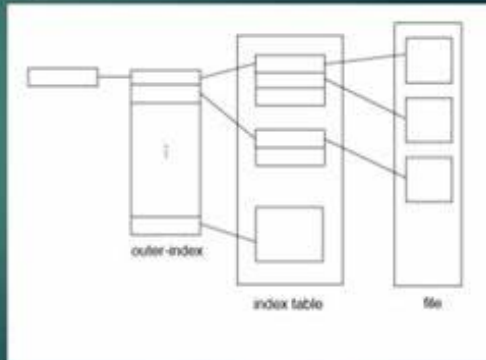


Indexed Allocation – Mapping (Cont.)

18

□ Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)

$LA / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$
 Q_1 = displacement into outer-index.
 R_1 is used as follows:
 $R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$
 Q_2 = displacement into block of index table
 R_2 displacement into block of file:



Free Space

- Free-space list: records all free disk blocks
- Scheme
 - Bit vector
 - Linked list (same as linked allocation)
 - Grouping (same as linked index allocation)
 - Counting (same as contiguous allocation)
- File systems usually manage free space in the same way as a file

國立清華大學

Bit vector

- Bit Vector (bitmap): one bit for each block
 - e.g. 001111100111111111001110011000000.....
- ⊕: simplicity, efficient
(HW support bit-manipulation instruction)
- ⊖: bitmap must be cached for good performance
 - A 1-TB(4KB block) disk needs 32MB bitmap

0 1 2 ... n-1

bit[i] = $\begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$

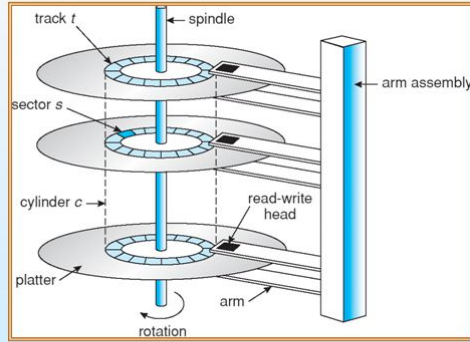
Chapter11 FS Implementation Operating System Concepts – NTHU LSA Lab 31

Storage Management

Disk Structure



Disk Structure



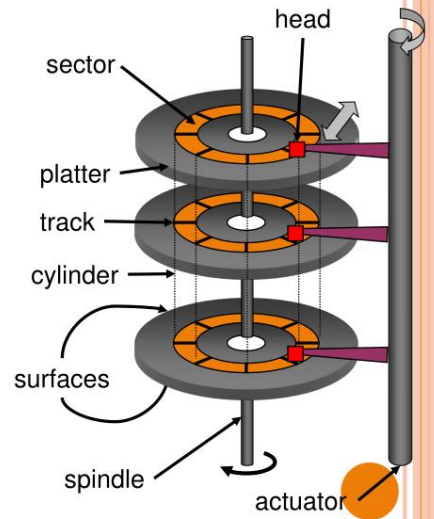
Moving-head disk mechanism

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**
 - The logical block is the smallest unit of transfer, usually 512 bytes
- The array of logical blocks is mapped into the **sectors** of the disk sequentially
 - Sector 0 is the first sector of the first **track** on the outermost **cylinder**
 - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

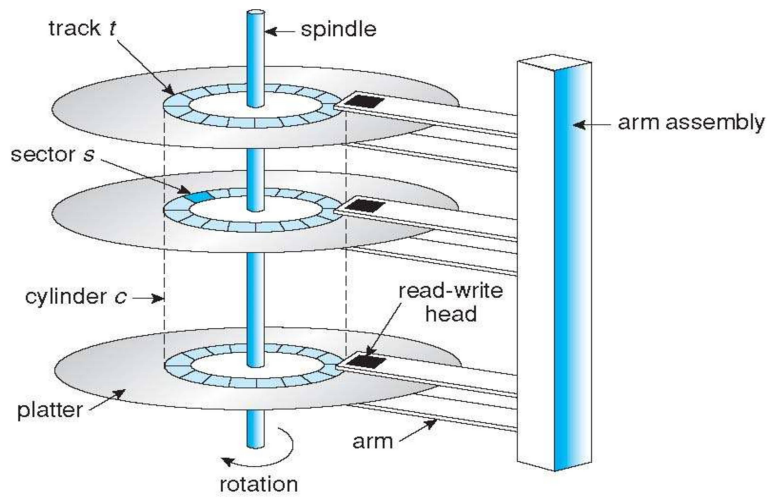


DISK DRIVE STRUCTURE

- Data stored on surfaces
 - Up to two surfaces per platter
 - One or more platters per disk
- Data in concentric tracks
 - Tracks broken into sectors
 - 256B-1KB per sector
 - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
 - Actuator moves heads
 - Heads move in unison



Moving-head Disk Mechanism



12.5

Sectors per Track

國立清華大學

- **Constant linear velocity (CLV)**
 - density of bits per track is uniform
 - more sectors on a track in outer cylinders
 - keeping same data rate
 - ➔ increase rotation speed in inner cylinders
 - applications: CD-ROM and DVD-ROM
- **Constant angular velocity (CAV)**
 - keep same rotation speed
 - larger bit density on inner tracks
 - keep same data rate
 - applications: hard disks

Chapter12 Mass Storage System Operating System Concepts – NTHU LSA Lab

4

Disk IO

國立清華大學

- Disk drive attached to a computer by an I/O bus
 - EIDE, ATA, SATA (Serial ATA), USB, SCSI, etc
 - I/O bus is controlled by controller
 - ◆ Host controller (computer end)
 - ◆ Disk controller (built into disk drive)

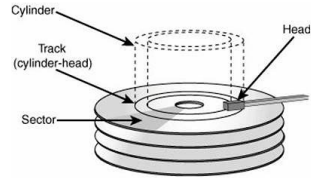


Chapter12 Mass Storage System Operating System Concepts – NTHU LSA Lab

5

Operating System

Disk Structure



- **Track**
- **Sector**
- **Cylinder**
- **Spindle**
- **Read/ Write Head (Arm assembly)**
- **Seek Time**
- **Rotational Latency**
- **RPM**
- **Transfer Time**

Presented by :Parul Vaghamshi

Disk Scheduling Algorithm



Selecting a Disk-Scheduling Algorithm

- Simulation results
 - low load --> SCAN
 - medium to heavy load --> C-SCAN
- influenced by the file-allocation method
 - contiguously allocated file
 - a linked or indexed file
- the location of directories and index blocks
 - placing the directories halfway between the inner and outer edge of the disk
 - placing the directories at either end

SunMoon University

19

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

16

Disk Scheduling Algorithms

Selection according to requestor		
RSS	Random scheduling	For analysis & simulation
FIFO	First in first out	Fairest of them all
Priority	Priority by process	No disk optimization
LIFO	Last in first out	Max locality & resource
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	NsS w/N=queue at beginning of SCAN cycle	Load sensitive

Example

Trace the policies FIFO, SSTF, SCAN, C-SCAN and FSCAN for the following disk requests. Each I/O request on a track takes 5 time units. At time 0, the disk starts reading track 10, and the read/write head was moving to the larger track number direction .

Time	0	1	2	3	6	7
Request to access track ..	10	19	3	14	12	9

	Track access order	Average seek length
FIFO	10,19,3,14,12,9	$(9+16+11+2+3)/5 = 8.2$
SSTF	10,14,12,9,3,19	$(4+2+3+6+16)/5 = 6.2$
SCAN	10,14,19,12,9,3	$(4+5+7+3+6)/5 = 5$
C-SCAN	10,14,19,3,9,12	$(4+5+16+6+3)/5 = 6.8$
FSCAN	10,14,19,3,9,12	$(4+5+16+6+3)/5 = 6.8$



Disk Scheduling Algorithms

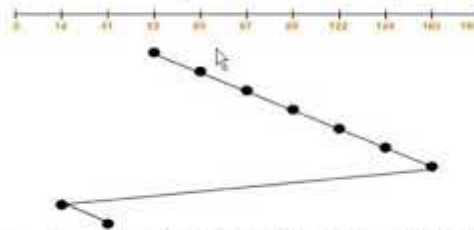
Table 11.2 Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	Average seek length	27.5	Average seek length	27.8	Average seek length	35.8

Problem - C-LOOK

Disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53 and moving towards the end. The cylinders are numbered from 0 to 199. The total head movement in number of cylinders incurred while servicing these requests is _

Solution-



Order of head movements = 53 → 65 → 67 → 98 → 122 → 124 → 183 → 14 → 41

Total head movements incurred while servicing these requests

$$= |53 - 65| + |65 - 67| + |67 - 98| + |98 - 122| + |122 - 124| + |124 - 183| + |183 - 14| + |14 - 41| = 12 + 2 + 31 + 24 + 2 + 59 + 169 + 27 = 326$$

Verification: = $|183 - 53| + |183 - 14| + |41 - 14| = 130 + 169 + 27 = 326$

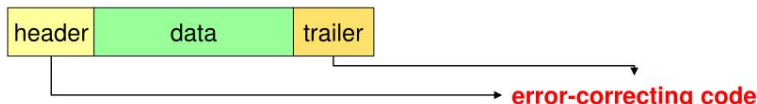
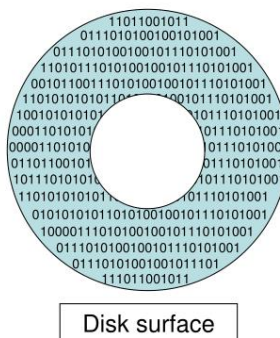
Disk Formatting

Disk Formatting

A brand new magnetic disk is a blank slate. There is no such thing as sectors. Tracks exist only as abstractions: we know they are actually created by how disk heads move over the disk surface (step motors).

Before a disk unit can be used as we've discussed, it is necessary to divide each track into sectors, what is known as *low-level formatting*.

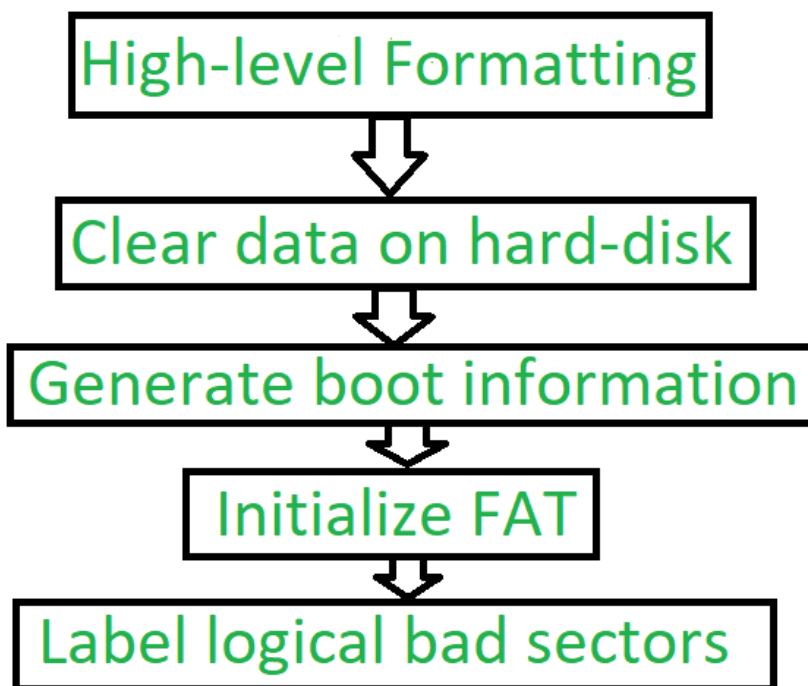
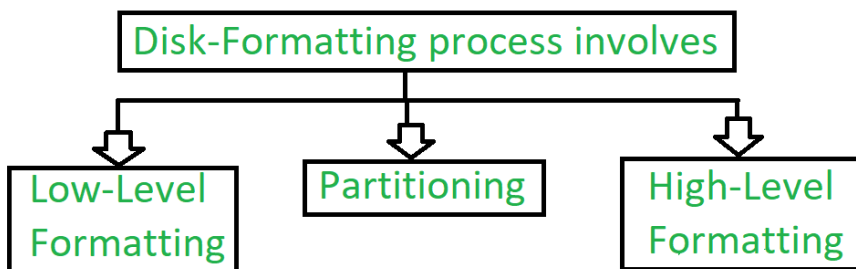
This formatting operation fills the disk with a special data structure for each sector:



04/21/2004

CSCI 315 Operating Systems Design

2



Swap-Space Management

Swap-Space Management

- **Swap-space — Virtual memory uses disk space as an extension of main memory**
- **Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition**
- **Swap-space management**
 - Allocate swap space when process starts; holds text segment (the program) and data segment
 - Kernel uses **swap maps** to track swap-space use



Swap-Space Management

- Swap-space
 - Virtual memory uses disk space as an extension of main memory
- Swap-space can be carved out of the normal file system, or,
 - more commonly, it can be in a separate disk partition
- Swap-space management
 - BSD allocates swap space when process starts;
 - holds *text segment* (the program) and *data segment*
 - Kernel uses *swap maps* to track swap-space use
 - Solaris 2 allocates swap space only when a page is forced out of physical memory,
 - not when the virtual memory page is first created.

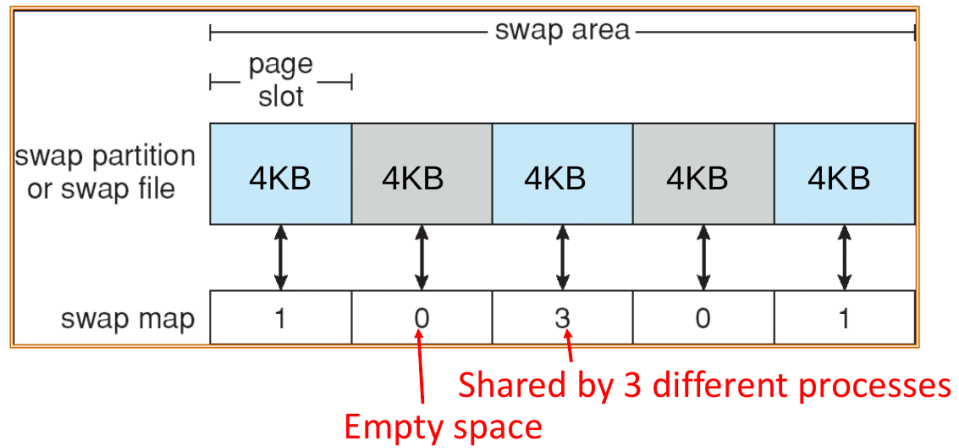


國立清華大學

Swap Space Allocation

- 1st version: copy entire process between contiguous disk regions and memory
- 2nd version: copy pages to swap space
 - Solaris 1:
 - **text segments** read from file system, thrown away when pageout
 - Only **anonymous memory** (stack, heap, etc) store in swap space
 - Solaris 2:
 - swap-space allocation only when **pageout** rather than **virtual memory creation time**

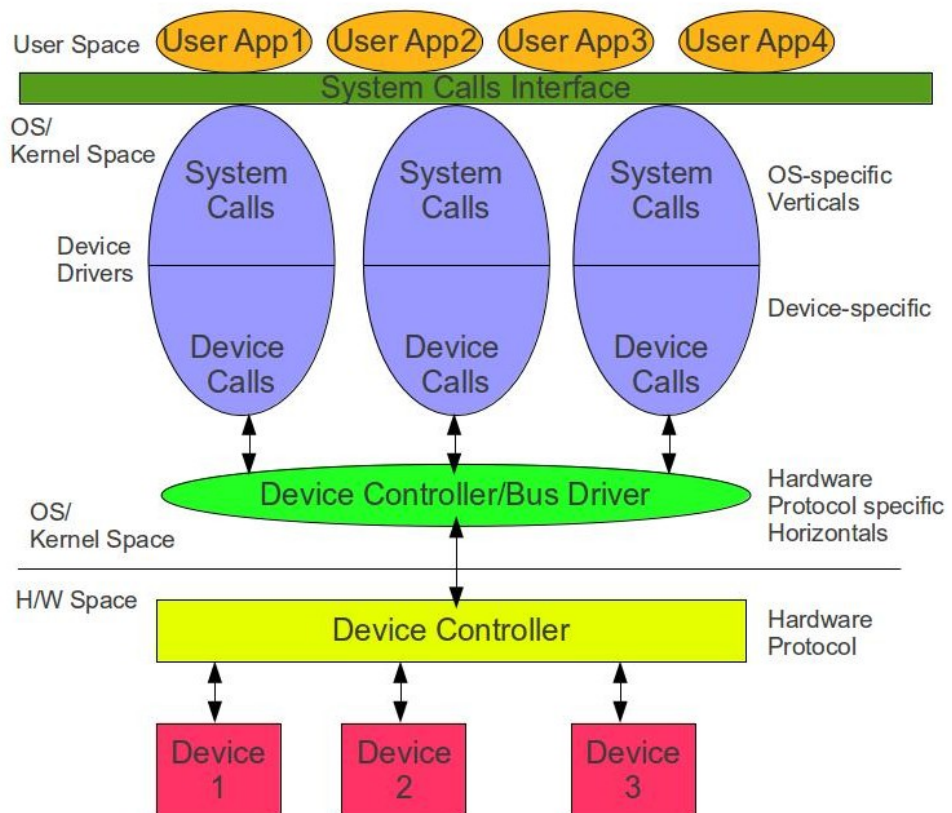
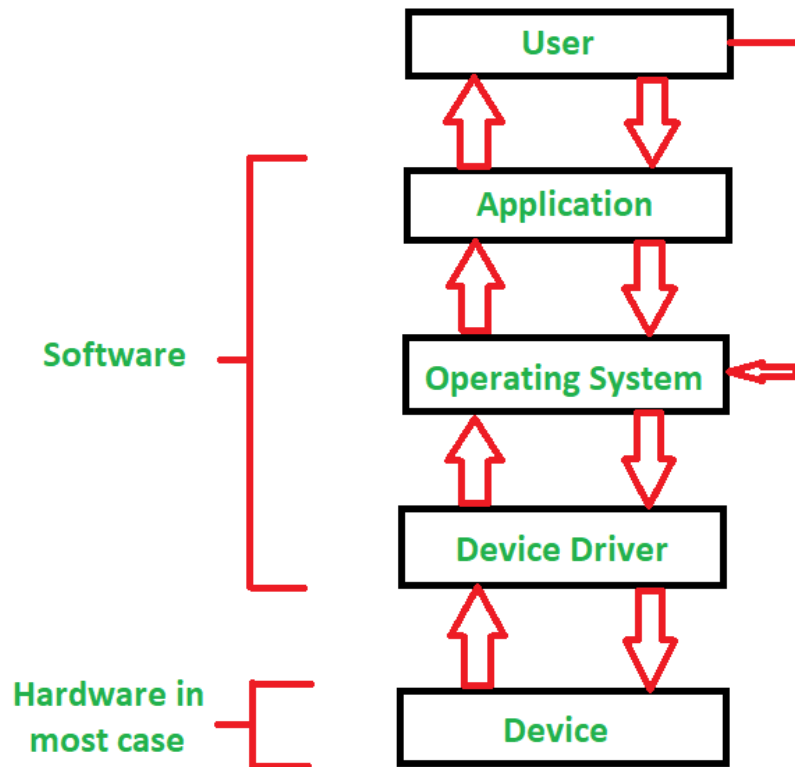
Chapter 12: Mass Storage System Operating System Concepts – NTHU, USA Lab 22

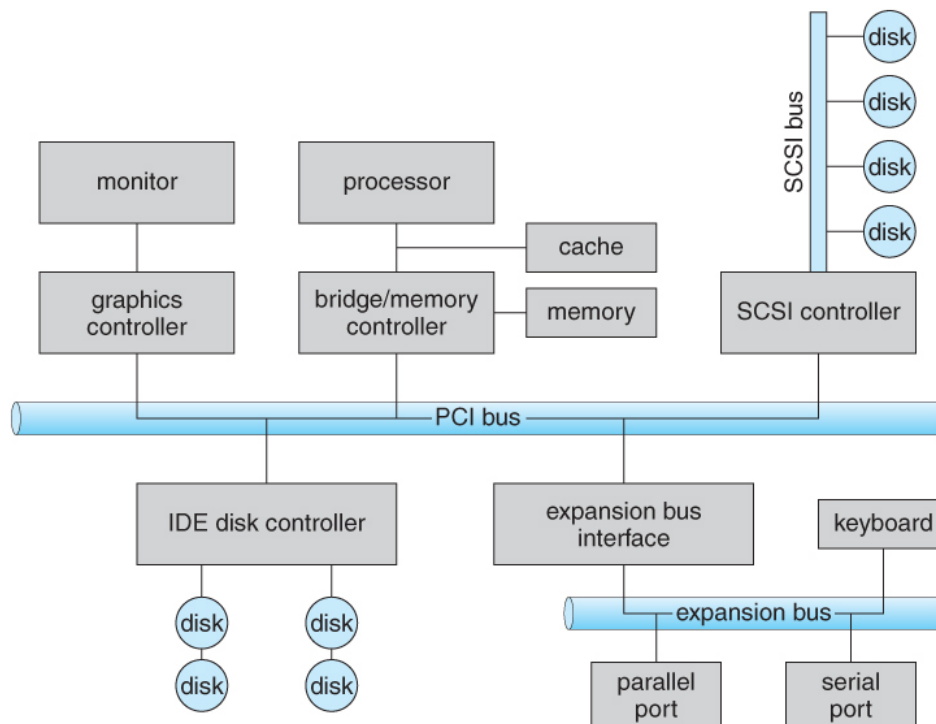


I/O System Mangement

I/O Hardware

- Computers support a wide variety of I/O devices, but common concepts apply to all:
 - **Port**: connection point for a device
 - **Bus**: set of wires that connects to many devices, with a protocol specifying how information can be transmitted over the wires
 - **Controller**: a chip (or part of a chip) that operates a port, a bus, or a device
- How can the processor communicate with a device?
 - Special instructions allow the processor to transfer data and commands to registers on the device controller
 - The device controller may support **memory-mapped I/O**:
 - The same address bus is used for both memory and device access.
 - The same CPU instructions can access memory locations or devices.





國立清華大學

I/O Methods Categorization

- Depending on how to address a device:
 - **Port-mapped I/O**
 - ✦ Use different address space from memory
 - ✦ Access by special I/O instruction (e.g. **IN**, **OUT**)
 - **Memory-mapped I/O**
 - ✦ Reserve specific memory space for device
 - ✦ Access by standard data-transfer instruction (e.g. **MOV**)
 - ☺ More efficient for large memory I/O (e.g. graphic card)
 - ☹ Vulnerable to accidental modification, error

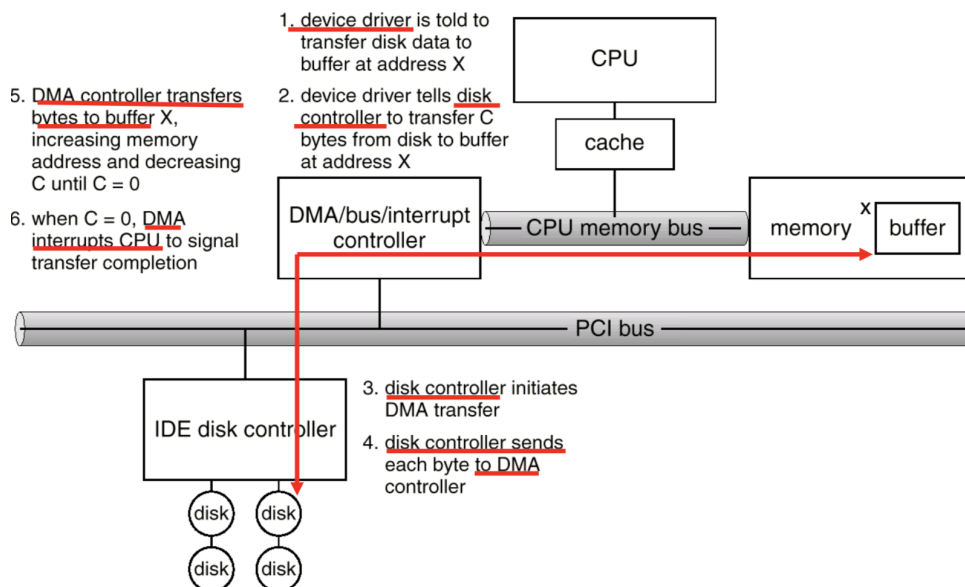
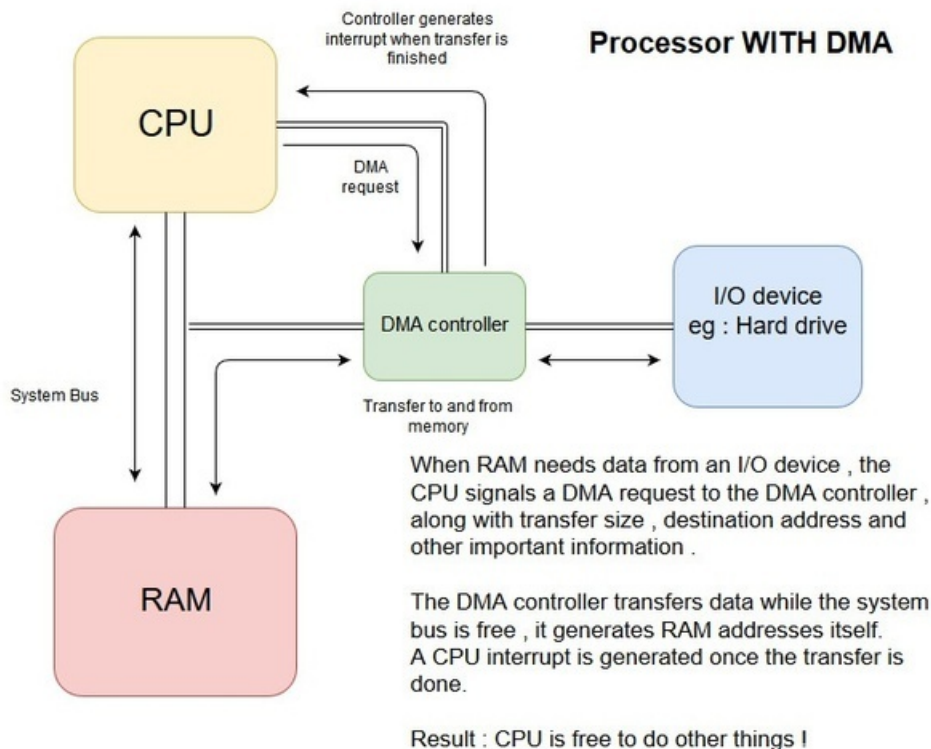
國立清華大學

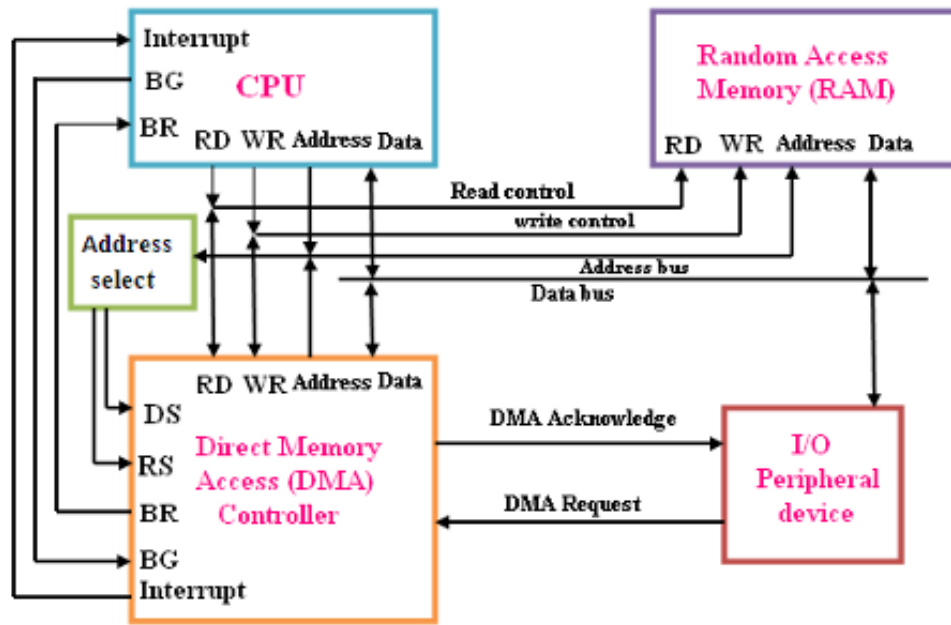
I/O Methods Categorization

- Depending on how to interact with a device:
 - **Poll** (busy-waiting): processor periodically check status register of a device
 - **Interrupt**: device notify processor of its completion
- Depending on who to control the transfer:
 - **Programmed I/O**: transfer controlled by CPU
 - **Direct memory access (DMA) I/O**: controlled by **DMA controller** (a special purpose controller)
 - ✦ Design for large data transfer
 - ✦ Commonly used with memory-mapped I/O and interrupt I/O method

Chapter 13 I/O Systems Operating System Concepts - NTHU, USA Lab 10

DMA(Direct Memroy Access)





I/O Structure

國立清華大學

A Kernel I/O Structure

- **Device drivers:** a uniform device-access **interface** to the I/O subsystem; hide the differences among device controllers from the I/O sub-system of OS

Chapter13 I/O Systems Operating System Concepts – NTHU USA Lab 24

國立清華大學

Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated shareable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read/write	CD-ROM graphics controller disk

Chapter13 I/O Systems Operating System Concepts – NTHU USA Lab 25


OS Security

Precautions for OS Security

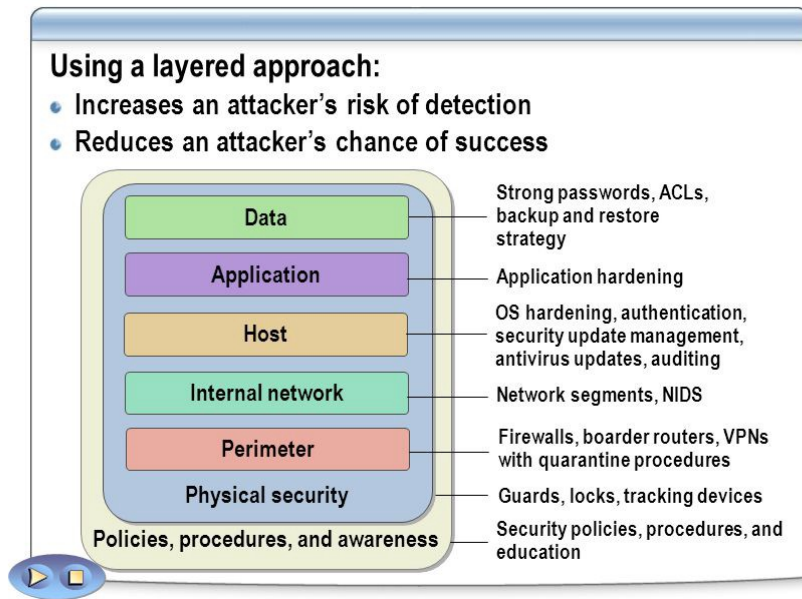
To Secure Our System and to avoid the Security breaches we must take some precautions in our OS.

So we have to ensure of the Shown Security Components:-

- Bios Security
- User Accounts Security
- Data Security
- Antivirus Security
- Firewall



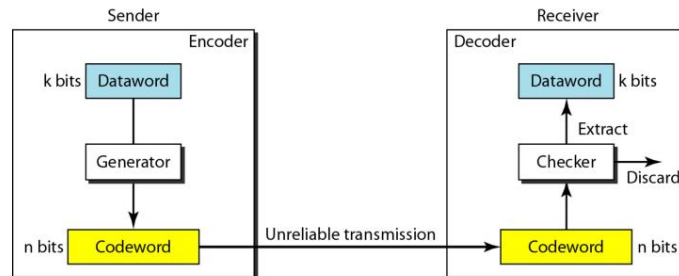
Understanding Defense-in-Depth



Error Detection and Correction

Error Detection

- A receiver can detect a change if the original codeword if
 - The receiver has a list of valid codewords, and
 - The original codeword has changed to an invalid one.



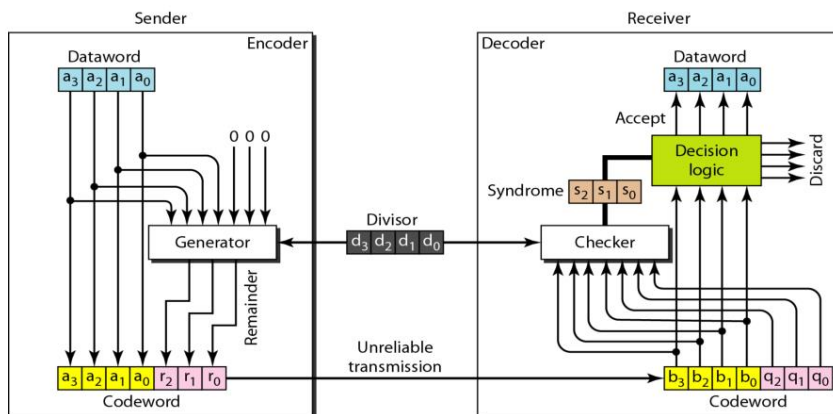
10.12

BCS Error Detection & Correction

- **Error Detection**
 - Check if any error has occurred
 - Don't care the number of errors
 - Don't care the positions of errors
- **Error Correction**
 - Need to know the number of errors
 - Need to know the positions of errors
 - More complex
 - The number of errors and the size of the message are important factors in error correction

BCS Error Detection & Correction

Figure 1.6 CRC encoder and decoder



1. Resource Allocation

- In case of multiple users accessing same resource
- In case of multiple processes accessing same resource
- Example:
 - Multiple users may be accessing same resource (say printer), then OS allocates the printer based on some algo (like FIFO for ex)
 - CPU Scheduling algos (FIFO, SJF etc)

S

2. Accounting

- Statistics related to the resource usage by the users – how much resource each user consumes, what all resources a user uses etc.
- Can be used for billing purposes
- Can also help in reconfiguring system to improve computing services

S

3. Protection and Security

- Protection – access to system should be controlled
 - Multiuser systems should not allow an unauthorized user to access content
 - No interference between 2 processes
- Security – from external world
 - To access a system, there must be some kind of authorization (like a password)



S

-- Memo End --