LECTURE 9:
Operating Systems

CIS 5100
IS/IT Architectures
Fall 2020

Instructor: Dr. Song Xing

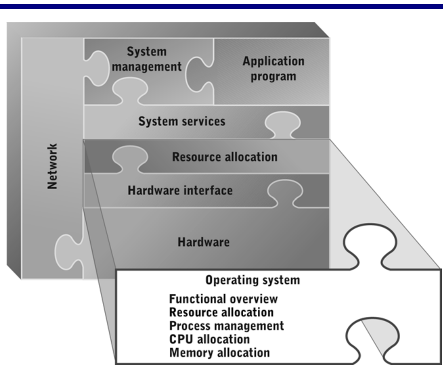
Department of Information Systems
California State University, Los Angeles

Learning Objectives

- Describe the types of the operating system and the services provided.
- Describe how an operating system manages processes and threads
- Compare and contrast alternative CPU scheduling methods
- Explain how an operating system manages memory

2

Lecture Topics



3

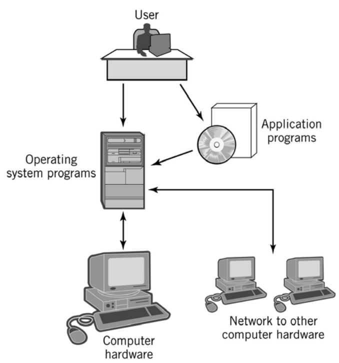
Outline

- OS overview
  - Services provided by OS
  - OS layers
  - Loading OS
- Resource allocation
- Processes management
- CPU allocation
- Memory management

4

CIS510

Integrated Computer Environment



5

CIS510

Definition of an Operating System

An *operating system* is a collection of computer programs that integrate the hardware resources of the computer and make those resources available to a user and the user's programs, in a way that allows the user access to the computer in a productive, timely, and efficient manner.

15-6

CIS510

Operating System (OS) Overview

- OS is the most important component of *system software*.
- Primary purpose: Manages all hardware resources and allocates them to users and applications as needed
  - Manages CPU, memory, processes, secondary storage (files), I/O devices, and users
- Performs many low-level tasks on behalf of users and application programs
- Accesses files and directories, creates and moves windows, and accesses resources over a network

7

CIS510

Basic Services

- Programs that accept commands and requests from a user and a user's program
- Manages, loads, and executes programs
- Manages hardware resources of the computer
- Acts as an interface between the user and the system

8

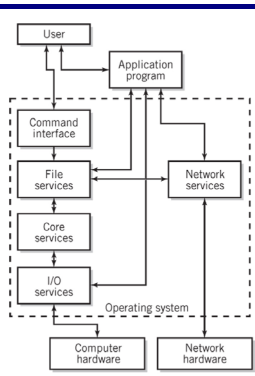
Additional Services

- Provides tools and services for *concurrent processing for multitasking.*
- Provides interfaces for the user and the user's programs
- Provides file support services
- Provides I/O support services
- Provides means of starting the computer
- Handles all interrupt processing
- Provides network services

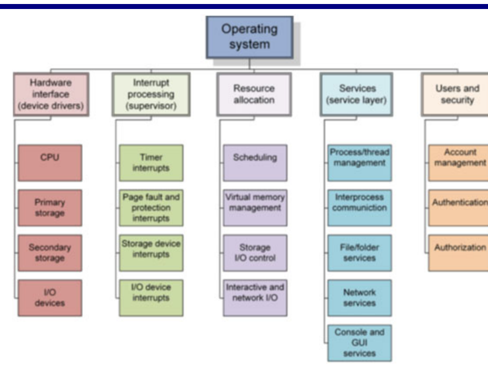
Services Required by Concurrent Processing

- Allocates resources such as memory, CPU time, and I/O devices to programs
- Protects users and programs from each other and provides for inter-program communication
- Provides feedback to the system administrators to permit performance optimization of the computer system

Simplified Diagram of Operating System Services



Operating System Management Functions



Outline

- OS overview
  - Services provided by OS
  - OS layers
    - Command layer
    - Service layer
    - Kernel
  - Loading OS
- Resource allocation
- Processes management
- CPU allocation
- Memory management

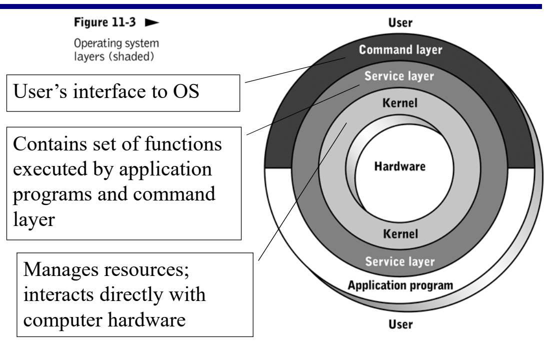
13

Operating System Layers

- Like other complex software, operating systems are organized internally into layers.
- Using layers makes the OS easier to maintain:
  - **Command layer** (shell)
    - *Shell*: user interface and command processor that interacts with the kernel
  - **Service layer**
  - **Kernel**

14

Operating System Layers (cont.)



15

Outline

- OS overview
  - Services provided by OS
  - OS layers
    - Command layer
    - Service layer
    - Kernel
  - Loading OS
- Resource allocation
- Processes management
- CPU allocation
- Memory management

16

Command Layer

CIS510

- **Command layer** – The common user interface elements and set of tools to manage hardware and software, for example:
  - Usually a *graphical user interface (GUI)* though *command languages* are also used.
  - Programs and tools to manage files (e.g., *File Explorer* in Windows 10)
  - Tools to add or update hardware and software (e.g., *Windows update*, Add/remove hardware devices, …)
  - Tools to execute application programs (e.g., double-click on an icon on the desktop or in Explorer)

User Interface Types

CIS510

- The command layer, sometimes called the *shell*, is the user interface to the OS.
  - Through this layer, a user or system administrator can run application and OS utility programs and manage system resources, such as files, folders, and I/O devices.
- User interface types:
  - *CLI* - Command Line Interface
    - A text interface that accepts user input from the keyboard.
  - *GUI* (pronounced *goo-ee*) - Graphical User Interface

Command Line Interface (CLI)

CIS510

- A set of commands and syntax requirements is called *command language* or *job control language (JCL).*
  - Example: *MS-DOS, IBM MVS JCL, Unix Bourne shell*
- Syntax: `command <operand1> <operand2>` …
  - Operands: specify parameters that define the meaning of the command more precisely.
- Use examples:
  - *Unix/Linux*: `ls cis5100/proj`
    // Consists of command `ls` and operand `cis5100/proj`; Requests a directory listing from the subdirectory with path name `cis5100/proj`
  - *Windows*: `javac Mycode.java //` javac command reads Java source files and compiles them into *bytecode* class files that run on the *Java Virtual Machine (JVM).*

MS-DOS and UNIX Commands Examples

CIS510

MS-DOS/Windows	UNIX/Linux	
dir	ls	List a directory of files or get information about files
copy	cp	Copy a file from one place to another
move	mv	Move a file from one place to another
del or erase	rm	Delete (remove) a file
type	cat	Type a file out to the screen (or redirected to a printer)
mkdir	mkdir	Attach a new subdirectory to the tree at this tree junction
rmdir	rmdir	Delete a subdirectory

CIS510

Windows IPCONFIG Command Result Showing Physical Address

Command Prompt

```
C:\>ipconfig /all

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Dual Band Wireless-AC 8265
   Physical Address. . . . . . . . . : F8-59-71-EB-E1-0F
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::6437:9616:f729:6d2%9(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.4(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Tuesday, September 4, 4:28:34 PM
   Lease Expires . . . . . . . . . . : Thursday, September 6, 8:43:16 AM
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCP Server . . . . . . . . . . . : 192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 133716337
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-21-5F-87-27-F4-FE-CE-14-30-B9
   DNS Servers . . . . . . . . . . . : 192.168.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```


CIS510

Commonly used UNIX commands

Command	Function
date	Display the current date and time
ls -la	Display with details all the files in the current directory
ps -ef	Display details of the current running programs
find *dir filename* -print	Search for *filename* in the directory *dir* and display the path to the filename on finding the file
cat *file*	Display the contents of *file*
cd */d1/d2/d3*	Change the current directory to *d3*, located in */d1/d2*
cp *file1 file2*	Make a copy of *file1*, named *file2*
rm *file*	Remove (delete) *file* (Note that this is a permanent deletion; there is no trash can or recycle bin from which to recover the deleted file.)
mv *file1 file2*	Move (or rename) *file1* to *file2*
mkdir *dir*	Make a new directory named *dir*
rmdir *dir*	Remove the directory named *dir*
who	Display a list of users currently logged on
vi *file*	Use the "visual" editor named vi to edit *file*
lpr *file*	Print *file* using the default printer; lpr actually places *file* in the printer queue. The file is actually printed by lpd (line printer daemon), the UNIX printer service.
grep "*string*" *file*	Search for the string of characters in *string* in the file named *file*
ifconfig	Display the network interface configuration, including the IP address, MAC address, and usage statistics for all NICs in the system
netstat -r	Display the system's TCP/IP network routing table
sort *filename*	Sort alphabetically the contents of *filename*
man *command*	Display the man page entry for "*command*"
chmod *rights file*	Change the access rights (the mode) of *file* to *rights*
chgrp *group file*	Change the group to which the *file* belongs to *group*
telnet *host*	Start a virtual terminal connection to *host* (where *host* may be an IP address or a host name)
ftp *host*	Start an interactive file transfer to (or from) *host* using the FTP protocol (where *host* may be an IP address or a host name)
startx	Start the X Window system
kill *process*	Attempt to stop a running program with the process ID *process*
tail *file*	Display the last 10 lines of *file*
exit	Stop the current running command interpreter. Log off the system if this is the initial command interpreter started when logging is on

CIS510

Use Command Lines

- Advantages
  - More flexible and powerful
  - Faster for experienced users
  - Can combine commands
- Many system administrators use *command lines*
- *GUI* executes commands
  - Responds to mouse clicks
- Command interpreter (*shell*)
  - Accepts keyboard commands and runs them

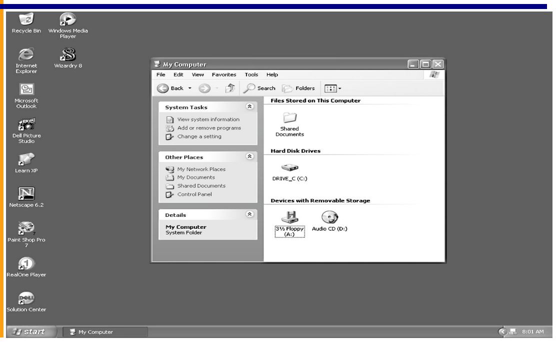
CIS510

GUIs -- Windows Interfaces

- Modern OSs use *Graphical User Interfaces (GUIs)* that enable users to interact with visual representations (*icons*) of programs and other resources and manipulate them with actions, such as touching, clicking, or dragging them.
- *Mouse*-driven and *icon*-based

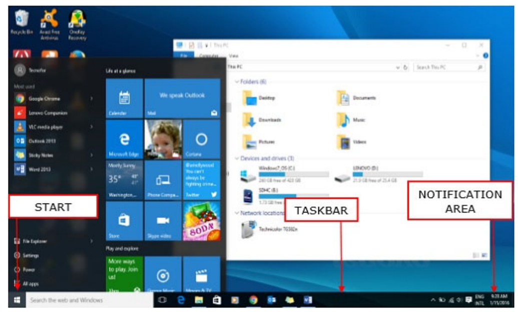
CIS510

GUI Interface – Windows XP



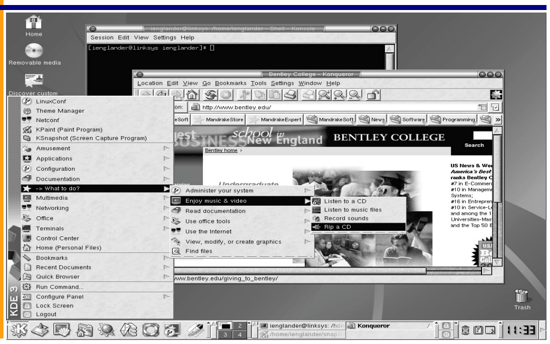
CIS510

GUI Interface – Windows 10



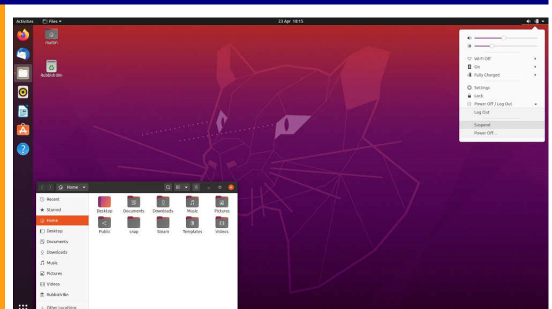
CIS510

GUI Interface – Linux KDE



CIS510

GUI Interface – Linux Ubuntu



GUI vs. CLI

CIS510

GUI

- Advantages
  - Easy to learn and use
  - Little training
  - Amenable to multi-tasking
- Disadvantages
  - Harder to implement
  - More Hardware/Software requirements
  - **Requires lots of memory**
  - Software is complex and difficult to write

CLI

- Advantages
  - More flexible and powerful
  - Faster for experienced users
  - Can combine commands
- Disadvantages
  - More difficult to learn and use

29

Outline

CIS510

OS overview
- Services provided by OS
- OS layers
  - Command layer
  - Service layer
  - Kernel
- Loading OS

Resource allocation
Processes management
CPU allocation
Memory management

30

Service Layer

CIS510

- Service Layer - A large set of utility functions, subroutines, or methods that application programs "call" to perform common tasks such as:
  - File I/O
  - Network interaction
  - Window management
- *Service call*: a *request* to execute a service-layer function is called a *service call*.

Outline

CIS510

OS overview
- Services provided by OS
- OS layers
  - Command layer
  - Service layer
  - Kernel
- Loading OS

Resource allocation
Processes management
CPU allocation
Memory management

32

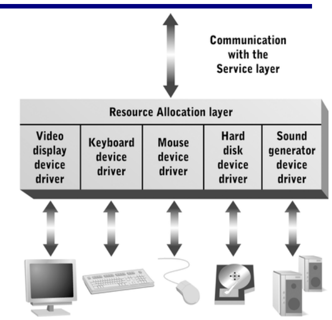
Kernel

- The kernel is the OS portion that manages resources and interacts directly with computer hardware via device drivers and “logical” commands.
- The kernel allocates hardware resources to users and programs and controls access.
- Kernel is the central module of an operating system.
- The kernel is always loaded into memory at *start-up time* and will remain resident as long as the computer is running.
- It contains essential services required by other parts of the operating system and applications.
- It is typically responsible for *memory management*, *process and task management*, and *secondary storage management*.

Components of the Kernel

Figure 11-4 ►
Components of the kernel

Kernel includes a *resource allocation layer* and interface programs called *device drivers* for each hardware device in the computer.

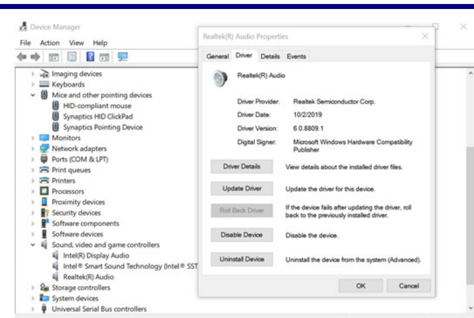


34

Kernel and Device Drivers

- The hardware interface functions of the kernel are performed by a set of *device drivers* – one or more per device.
- Device drivers enable the service layer to “talk” to multiple devices of the same type in the same way
  - For example, all disk drives “look” the same to file I/O functions – a linear address space with basic read/write commands
- *Device drivers can be updated* without changing any other OS or application components
  - This enables OS functions to be extended to new or “improved” hardware

Windows 10 Device Driver Properties



CIS510

Windows 10 Device Driver Properties (cont.)

- On a Windows computer do the following:
  - In the **search** bar located on the left-hand side of your taskbar, next to the **Windows** button, type *Device Manager*
  - Select *Device Manager*
  - Expand the category "Sound, video, and game controllers" (click the + sign)
  - Right-click on the audio device and select *Properties*
  - Examine the contents of the three tabs, especially the one titled *Driver*

CIS510

Outline

OS overview
- Services provided by OS
- OS layers
  - Command layer
  - Service layer
  - Kernel
- Loading OS

Resource allocation

Processes management

CPU allocation

Memory management

CIS510

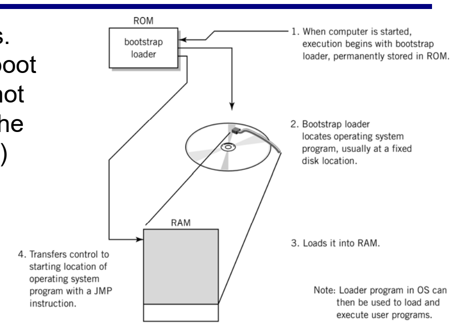
When the Machine Is Powered Up

- Execution begins with *bootstrap loader* stored in ROM (*BIOS* for PC)
  - When the computer is turned on, the CPU passes control to the BIOS program.
  - When *BIOS* boots up (starts up) the computer, it first determines whether all of the attachments (devices) are in place and operational.
  - The *bootstrap loader* looks for OS program in a fixed location (hard disk or diskette drive).
  - OS is loaded into memory.
  - The system control is passed to the *operating system.*
  - Then the loader program in OS is used to load and execute user programs.

CIS510

Bootstrapping a Computer

- Cold vs. warm boot (does not retest the system)



Outline

CIS510

- OS overview
- Resource allocation
  - Single-Tasking Resource Allocation
  - Multitasking resource allocation
- Processes management
- CPU allocation
- Memory management

Resource Allocation

CIS510

- *Resource allocation* is a needed function whenever multiple "things" *compete for* access to *a limited set of resources*, for example:
  - Departments and employees competing for budget
  - Packages competing for space within trucks, planes, and warehouses
  - Vehicles competing for access to roadways and intersections
- Resource allocation is the process of deciding:
  - Who gets to use what resource
  - How much of resource they get
  - When they get it
  - When they have to give it up
- Complexity increases as the number and diversity of resources and competitors rises
  - For example, consider the complexity of budgeting and room scheduling in a 4-room elementary school vs. a university

Resource Allocation Process

CIS510

- Resource allocation: ensure that overall system objectives are achieved efficiently and effectively
- Keeps detailed records of available resources
- Knows which resources are used to satisfy which requests
- Schedules resources based on specific allocation policies to meet present and anticipated demands
- Updates records to reflect resources commitment and release by processes and users

Real and Virtual Resources

CIS510

- **Real resource** – a computer system's physical devices and associated system software that physically exist
  - e.g., my laptop has one eight-core CPU and 16 GB of RAM
- **Virtual resource** – the resources that are apparent to a process or user
  - A virtual resource "looks" real to the user (program) but may or may not physically exist.
  - E.g., My laptop is concurrently executing ten programs - each program "thinks" it has its own CPU.

How is Virtual Resource Possible?

- A single program can be *actively running*, *idle*, or *waiting* – it needs few/no real resources in a *non-running state.*
- Real resources can be *rapidly shifted* among programs as they enter/leave a running state.
- To the extent possible, cheaper resources can be substituted for more expensive ones
  - For example, part of the instructions or data that program #6 "thinks" are stored in RAM are actually on disk --- *virtual memory*
  - Example: Memory is temporarily substituted for CPU registers while a program is suspended, pending completion of an I/O request. -- *interrupt processing*

Types of Operating Systems

- Single-tasking systems
- Multitasking systems
  - Commonly used
- Distributed systems
  - Processing power is distributed among computers in a cluster or network
- Network server systems
- Embedded systems
  - Designed to control a single piece of equipment, such as the automobile, cellular phone, or microwave oven
- Real-time systems
  - *One or more processes must be able to access the CPU immediately when required*. E.g., control rockets on a space flight, periodic measurements of the temperature in a nuclear reactor

Outline

- OS overview
- Resource allocation
  - ⇨ Single-Tasking Resource Allocation
  - Multitasking resource allocation
- Processes management
- CPU allocation
- Memory management

Single-Tasking Resource Allocation

- If a computer supports only one running application at a time it is said to be **single-tasking.**
- Single-tasking resource allocation is simple since there's little competition for hardware resources
  - Only one application program at a time is loaded into memory and executed.
  - Resource allocation in a single-tasking OS involves only two running programs -- an application and the OS.
  - There are two real-time "competitors" for resources:
    - The OS itself
    - Whatever application program is currently running
- Example: MS-DOS until the early 1990s

Outline

- OS overview
- Resource allocation
  - Single-Tasking Resource Allocation
  - Multitasking resource allocation
- Processes management
- CPU allocation
- Memory management

Multitasking Systems

- OS support for running multiple programs (tasks) at one time is called *multitasking*.
- Multitasking (or *multiprogramming*) operating systems are the norm for general-purpose computers
- Multitasking allows application and system software to be more flexible especially because large programs can be built from *smaller independent modules or processes*.
- Multitasking operating systems must be able to handle multiple programs and users
  - *Multiuser* systems have to be multitasking.

Multitasking Resource Allocation Goals

- A multitasking operating system *manages hardware resources* (CPU, memory, I/O) to achieve the following:
  - Meet the resource needs of processes
  - Prevent processes from interfering with one another
  - Efficiently use hardware and other resources

Achieving Multitasking

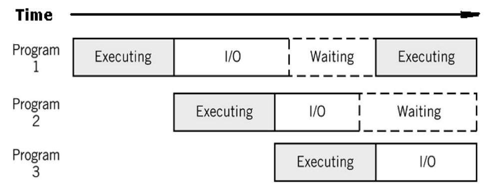
- Multitasking can be achieved by *concurrent processing*.
- The OS acts as a controller to provide concurrent processing.
  - OS makes rapid decisions about which programs receive CPU control and for how long that control is retained.
- Programs share CPUs (called *concurrent* or *interleaved execution*)

Achieving Multitasking (cont.)

- Sharing the CPU during I/O breaks: while one program is waiting for I/O to take place, another program is using the CPU to execute instructions.
- *Time-slicing*: the CPU may be switched rapidly back and forth between different programs.
  - Executing a few instructions from each, using a *periodic clock-generated interrupt*.
  - It was discussed in previous lecture.

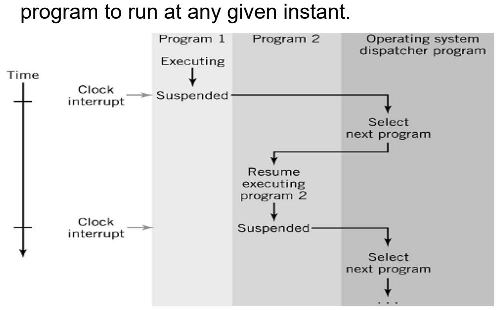
Sharing the CPU During I/O Breaks

- I/O represents a large percentage of a typical program's execution



Time-sharing the CPU (Time-slicing)

- *Dispatching* is the process of selecting which program to run at any given instant.



Outline

- OS overview
- Resource allocation
- Processes management
  - Processes
  - Threads
- CPU allocation
- Memory management

Processes

- A process is the basic unit of work in the OS.
  - A Process is a program together with all the resources that are associated with it as it is executed.
  - I.e., A process contains all of the resources to execute as a stand-alone entity.
  - Program: a file or listing
  - Process: a program being executed
- Managed independently by OS.
- Can request and receive hardware resources and OS services.
- Can be stand-alone or part of a group that cooperates to achieve a common purpose.
- Can communicate with other processes executing on the same computer or on other computers.

57

Process Control Data Structures

- The OS keeps track of each process by creating and updating *a data structure* called a *PCB* for each active process, kept in *a memory area* that is protected from the normal user access.
- The ***PCB** (Process Control Block)*
  - Created when the process is created, updated when the process changes and deleted when the process terminates.
  - Used by OS to perform many functions (e.g., resource allocation, secure resource access, protecting active processes from interference with other active processes)
- PCBs are normally organized into *a larger data structure* (called a linked list, process queue, or process list).

58

Process Control Block (PCB)

- A block of data for each process in the system
- *Contains all relevant information about the process*
  - Location of code in memory, stack pointer value, process ID, priority value and many more
  - PID (process ID): a unique identifier for each process
- Typical process control block on the right →

Process ID
Pointer to parent process
Pointer area to child processes …
Process state
Program counter
Register save area …
Memory pointers
Priority information
Accounting information
Pointers to shared memory areas, shared processes and libraries, files, and other I/O resources

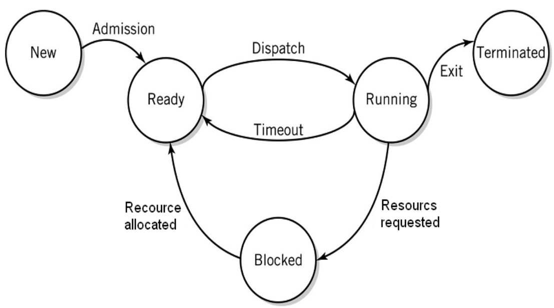
59

Process States

- Three primary process operating states:
  - *Ready state*: a process is waiting for access to a CPU.
  - *Running state*: the process retains control of the CPU until it terminates normally or an interrupt occurs.
  - *Blocked state*: the process is suspended while an *interrupt* is being processed; Waiting for some event to occur (completion of service request or correction of an error condition)

60

Process State Diagram



Interrupt Processing

- CPU automatically suspends currently executing process, pushes current register values onto the stack, and transfers control to OS master interrupt handler.
- Suspended process's state remains on the stack until interrupt processing is completed (Process is put into a blocked state)
- Once interrupt has been processed, OS can leave suspended process in blocked state, move it to *ready state*, or return it to *running state*.

Outline

- OS overview
- Resource allocation
- Processes management
  - Processes
  - Threads
- CPU allocation
- Memory management

Threads

- Processes can subdivide themselves into more easily managed subunits called *threads*.
- A process has at least one thread, but a thread is not necessarily a process.
- A thread is a portion of a process that *can be scheduled and executed independently*.
- Each thread consists of a program counter, a register set, and a stack space, but
  - *shares* all resources allocated to its parent process including primary storage, files and I/O devices.
- Advantage: reduce OS overhead for resource allocation and process management.

Threads (Cont.)

CIS510

- The OS keeps track of thread-specific information in a *thread control block (TCB)* .
  - Each *PCB* contains pointers to its related TCBs.
- Threads can execute *concurrently on a single processor* or *simultaneously on multiple processors*.
- Like processes, threads can be created and destroyed and can be in *ready*, *running*, and *blocked* states.

Thread States

CIS510

- Ready
  - Waiting for access to the CPU
- Running
  - Retains control of CPU until the thread or its parent process terminates normally or an interrupt occurs
- Blocked
  - Waiting for some interrupt event to occur (completion of service request or correction of an error condition)

Interrupt Examples

CIS510

- *Interrupts* can occur for a variety of reasons, including the following
  - Executing a service call, such as a file I/O request
  - A hardware-generated interrupt indicating an error, such as overflow, or a critical condition, such as a power failure alarm from an uninterruptible power supply.
  - An interrupt generated by a peripheral device, such as a NIC when a packet arrives.

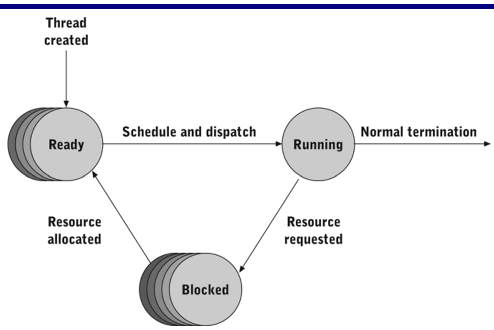
Thread Waiting for Resources using Interrupt Processing

CIS510

- Thread can be blocked waiting for resources.
- Thread is put into a *wait state* (*blocked state*) and its state stored on the stack.
- Some *interrupt handler* will process the blocked thread's request.
- When the resource has been allocated, thread is moved from blocked state to ready or running state.
- The thread remains in the blocked state until the request is satisfied or a time out occurs.

CIS510

Thread Movement Between States



69

CIS510

Process vs. Thread

- A process is a unit of execution that contains all of the resources to execute as a stand-alone entity.
- A thread is usually a *subset of a process*, and is the *smallest unit* of executable code that can be scheduled on its own.
- A process has at least one thread, but a thread is not necessarily a process.
- Processes are sometimes considered '*heavy-weight*' while threads are considered '*light-weight*', referring to the amount of resources allocated to each type.
- Processes have unique address spaces; threads within a process share the address space of the process.

70

CIS510

Types of Operating Environments

- A single process, non-threaded (SPNT) OS runs one process at a time.
  - Example: *Microsoft*'s DOS
- A single process, multi-threaded (SPMT) OS runs only one process at a time, but supplies an interface that allows for multiple threads to execute in that process.
  - Example: *General Software*'s Embedded DOS
- A multi-process, non-threaded (MPNT) OS have many processes with a single thread of execution.
  - Example: OS in mini-computers
- Unix/Linux and Windows are examples of *multi-process, multi-threaded (MPMT)* environments.
  - macOS is the Unix-based graphical operating system

71

CIS510

Outline

- OS overview
- Resource allocation
- Processes management
- ⇨ CPU allocation
  - Preemptive scheduling
  - Non-preemptive scheduling
  - Real-time scheduling
- Memory management

72

CPU Allocation

- A multitasking OS can execute dozens, hundreds, or thousands of threads in the same time frame.
- Most computers have only one or few CPUs, threads usually share CPUs (concurrent or interleaved execution).
- OS makes rapid decisions about which threads receive CPU control and for how long that control is retained.
- *CPU time allocation*
  - Provides mechanism for the acceptance of threads into the system and for the actual *allocation of CPU time* to execute those threads.
  - Meets the objective to optimize use of computer system resources by allowing multiple threads to *execute concurrently*.

Concurrent Thread Execution on a Single CPU

	Time slice 1	Time slice 2	Time slice 3	Time slice 4	Time slice 5	Time slice 6	Time slice 7
Thread 1	Running	Idle	Idle	Idle	Idle	Running	Idle
Thread 2	Idle	Running	Idle	Running	Idle	Idle	Idle
Thread 3	Idle	Idle	Running	Idle	Running	Idle	Running

FIGURE 11.7 Concurrent (interleaved) thread execution on a single CPU

74

Scheduling

- Decision-making process used by OS to determine which *ready thread* moves to the *running state*.
- The portion of the operating system that makes scheduling decisions is called the *scheduler*.

75

Scheduling Objectives

- Maximize throughput
  - Maximize the number of jobs completed in a given time period
- Minimize turnaround time
  - Minimize the time between submission of a job and its completion
- Maximize CPU utilization
  - Keep the CPU busy
- Maximize resource allocation
  - Maximize the use of all resources by *balancing processes* that require heavy CPU time with those emphasizing I/O

76

Scheduling Objectives (Cont.)

- Promote graceful degradation
  - As the system load becomes heavy, it should degrade gradually in performance.
- Provide minimal and consistent response time
  - An algorithm that allows a large variation in the response time may not be considered acceptable to users.
- Prevent starvation
  - *Starvation* is a situation that occurs when a process is never given the CPU time that it needs to execute.

77

Typical Scheduling Methods

- Non-preemptive scheduling
  - Program voluntarily gives up control
- Preemptive scheduling
  - Uses clock interrupt for multitasking
- Real-time scheduling
  - Actual selection of process(es) that will be executed at any given time

78

Outline

OS overview
Resource allocation
Processes management
CPU allocation
- ⇨ Preemptive scheduling
- Non-preemptive scheduling
- Real-time scheduling

Memory management

79

Preemptive Scheduling

- *Preemptive* systems will limit the time that the thread remains in the running state to a fixed length of time corresponding to one or more quanta.
- A running thread controls the CPU by controlling the content of the *instruction pointer*.
- In preemptive scheduling, a thread can be removed involuntarily from the running state.
  - CPU control of a thread is lost whenever an **interrupt** is received.
  - CPU then transfers control to the OS.
  - The portion of the OS that receives control is called the *supervisor*.

80

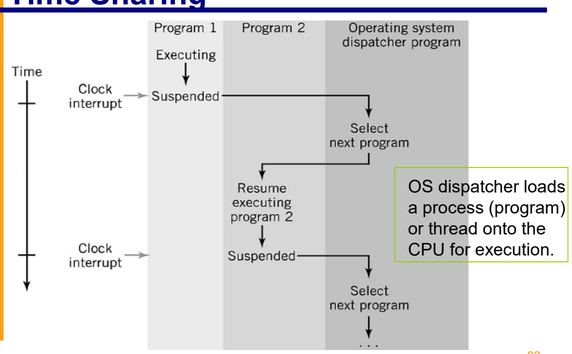
Functions of the Supervisor and the Scheduler

- Functions of the *supervisor*
  - Calls appropriate interrupt handler
  - Transfers control to the scheduler
- Functions of the *scheduler*
  - Updates status of any process or thread affected by last interrupt
  - Decides which thread to dispatch to the CPU
  - Updates thread control information and the stack to reflect the scheduling decision
  - Dispatches selected thread

Timer Interrupts

- Generated at regular intervals by CPU to give scheduler an opportunity to suspend currently executing thread
- Not a “real” interrupt; *no interrupt handler to call*; supervisor passes control to the *scheduler*.
- Important CPU hardware feature for multitasking OSs
  - Guarantee no thread can hold CPU for long period
- It is used in *round robin* scheduler.

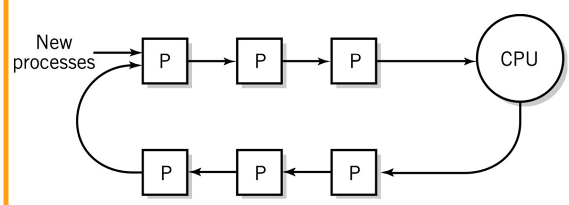
Using a Timer Interrupt for CPU Time Sharing



Round Robin

- The simplest *preemptive algorithm*.
- Round robin gives each process *a quantum of CPU time*.
- If the process is not completed within its quantum, it is forced to return to the ready state to await another turn.
- It is inherently fair and maximizes the throughput since *shorter jobs get processed quickly*.

Round Robin (cont.)



Round Robin (cont.)

- A variation on *round robin* that is used by some UNIX systems calculates a *dynamic priority* based on the ratio of required CPU time to the total time that the process has been in the system.
- The smallest ratio is treated as the highest priority and is assigned the CPU next.
- Both **Windows** and **Linux** also use such a *dynamic priority scheduling* algorithm as their primary criterion for dispatch selection.
  - The algorithms on both systems adjust priority based on their use of resources.

Outline

OS overview
Resource allocation
Processes management
CPU allocation
- Preemptive scheduling
- Non-preemptive scheduling
- Real-time scheduling

Memory management

Non-preemptive Scheduling

- *Non-preemptive* systems will allow a running process (thread) to continue running until it is completed or blocked.
- Types of non-preemptive scheduling:
  - First come first served (FCFS)
  - Priority scheduling
  - Shortest time remaining (STR)

First-Come, First Served (FCFS)

- Or *First in, First out (FIFO)*
- The simplest possible dispatch algorithm of non-preemptive scheduling
- Processes will be executed as they arrive, in order.
- Unfair to short processes and I/O based processes.

89

Priority Scheduling

- Uses a set of priority levels and assigns a level to each process or thread.
- Job with the highest priority is selected.
- If multiple jobs have the highest priority then the dispatcher selects among them using FIFO.

90

Shortest Remaining Time

- Or *Shortest Job First (SJF)*
- Chooses the next process to be dispatched based on the expected amount of CPU time needed to complete the process.
- Maximize throughput by selecting jobs that require only a small amount of CPU time.
- Longer jobs can be starved as short jobs will be pushed ahead of longer jobs.
  - Hence, when SJF is implemented, it generally includes a *dynamic priority factor* that raises the priority of jobs as they wait, until they reach a priority where they will be processed next regardless of length.

91

Outline

OS overview
Resource allocation
Processes management
CPU allocation
- Preemptive scheduling
- Non-preemptive scheduling
- Real-time scheduling

Memory management

92

Real-Time Scheduling

- Guarantees the minimum amount of CPU time to a thread if the thread makes an *explicit real-time scheduling request* when it is created.
- Guarantees a thread enough resources to complete its function within a specified time.
- Often used in transaction processing, data acquisition, and automated process control

Outline

OS overview
Resource allocation
Processes management
CPU allocation
Memory management

Memory Management

- Memory management is the point at which the operating system and hardware meet and it is concerned with managing the main memory and disk drives.
- When computers first appeared, an address generated by the computer corresponded to the location of an operand in physical memory. Even today, 8-bit microprocessors do not use memory management.
- Today, the *logical address* generated by high-performance computers in PCs and workstations is not the *physical address* of the operand accessed in memory.

Virtual Memory – Basic Ideas

- *Virtual memory (VM)* increases the apparent amount of memory by using far less expensive hard disk space.
- Provides for process separation
- Demand paging
  - *Pages* reside on hard disk and are brought into memory as needed
- *Page table*
  - Keeps track of what is in memory and what is still out on hard disk

CIS510

VM Terminology

- The *processes* are divided into **pages** – fixed size chunks, e.g., 4KB
- The memory is also divided into the page-sized chunks called the **page frames** (or **frames**) – same size as the process pages
- For each process, OS creates a ***page table*** residing *in memory* which stores the information about all pages of a single process.
- The OS memory manager software maintains the page tables for each program (process).

CIS510

Pages and Frames

	Process	**Memory**
Unit	Page	Page Frame
Address	Logical	Physical
Size	2 to 4KB	2 to 4KB
Amount	# of bits in instruction word	Installed memory

CIS510

Portion of a Process's Page Table

Page number	Memory status	Frame number	Modification status
0	In memory	214	no
1	On disk	101	n/a
2	On disk	44	n/a
3	In memory	110	yes
4	On disk	252	n/a

CIS510

Virtual Memory Management

- The only portion of a process that must be in memory at any point during execution are the *next instruction* to be fetched and *any operands* stored in memory.
- Only a few bytes of any process must reside in memory at any one time.
- OS minimizes the amount of process code and data stored in memory at one time, which frees large quantities of memory for use by other processes and substantially increases the number of processes that can execute concurrently.
- During process execution, one or more pages are allocated to frames in memory, and the rest are still held in secondary storage (auxiliary storage).

Virtual Memory Management (cont.)

- As pages in secondary storage are needed for current processing, the OS copies them into page frames in memory.
- If necessary, pages currently in memory are written to secondary storage to make room for other pages being loaded.

Handling Page Fault

- When the program is loaded, an exact, page-by-page ***image*** of the program is also stored in a known auxiliary storage location (called a *backing store* or *swap space*, *swap file*).
- The auxiliary storage area is usually found on disk or SSD.
- When an instruction or data reference is on a page that does not have a corresponding frame in memory, the CPU hardware causes a special type of ***interrupt*** called a *page fault* or a *page fault trap*.

Handling Page Fault (cont.)

- When a *page fault interrupt* occurs, the OS memory manager software answers the interrupt, selects a memory frame in which to place the required page.
- It then loads the required page from its program image in the *backing store* (disk or SSD) into the selected memory frame.
- If every memory frame is already in use by other pages, the manager software must pick a page in memory to be replaced.

Handling Page Fault (cont.)

- If the page being replaced has been altered, it must first be stored back into its own image, before the new required page can be loaded.
  - This is a requirement, because the page may have to be reloaded again later.
- That way, the *backing store* always contains the latest version of the program and data as the program is executed.

CIS510

Handling Page Fault (cont.)

- The process of page replacement is known as *page swapping*.
- Most OSs perform page swapping only when it is required as a result of a *page fault*. This procedure is called *demand paging*.
- Since the *virtual memory mapping* assures that any program page can be loaded anywhere into memory, there is no need to be concerned about allocating particular locations in memory. Any free frame will do.

105

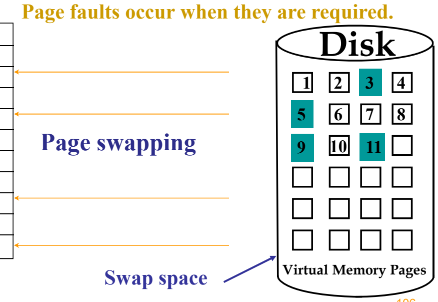
CIS510

Page Swapping

Pages (3, 5, 9, 11) are not in main memory. Page faults occur when they are required.

Page	Frame
1	6
2	4
3	
4	8
5	
6	10
7	1
8	2
9	
10	7
11	

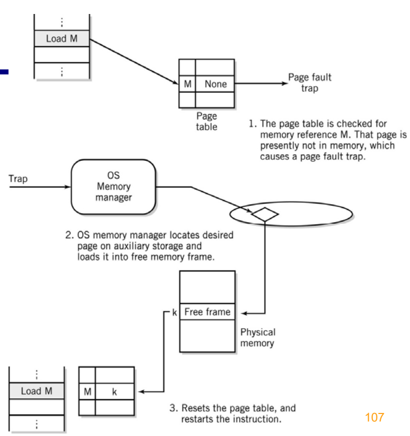
Page Table



106

CIS510

Steps in Handling a Page Fault



107

CIS510

Memory Address

- The physical memory address is divided into two parts: a *frame number* and the *offset* (because it represents the offset from the beginning of the frame).
  - The first address in a frame is 0.
- The instruction and data memory address references in a program is *logical* or *virtual* memory references.
- The logical (virtual) address is also separated into its *page number* and an *offset*.
- Since each page fits exactly into a frame, the offset of a particular address from the beginning of a page is also exactly the same as the offset from the beginning of the frame where the page is physically loaded.

108

Dynamic Address Translation

CIS510

- The *dynamic address translation (DAT)* automatically and invisibly translates every individual address in a program (the *logical* or *virtual* address) to a different corresponding physical location (the *physical* address) in memory.
- A lookup in the *program's page table* locates the entry in the table for the page number, then translates, or maps, the virtual memory reference into a physical memory location consisting of the corresponding frame number and the same offset.
- This operation is implemented in hardware by processor's *memory management unit (MMU).*

Dynamic Address Translation (cont.)

CIS510

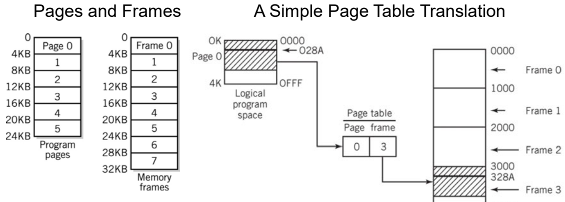
- Every memory reference in a *fetch-execute cycle* goes through the same translation process.
- The address that would normally be sent to the *memory address register (MAR)* is now mapped through the *page table* and then sent to the MAR.

Page Table Translation Example 1

CIS510

Pages and Frames

A Simple Page Table Translation



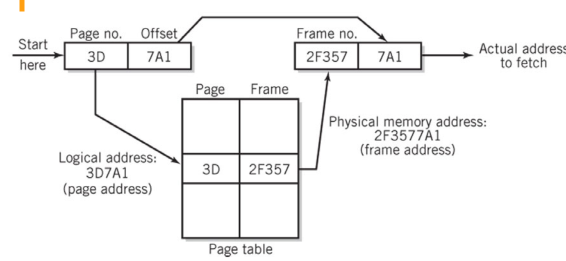
Each program has its collection of pages.

The total number of pages can exceed the number of frames (physical memory).

Logical address 028A is translated into the physical address 328A. 28A is the offset in this example.

Page Table Translation Example 2

CIS510



Note: the offset in a program page is also exactly the same as the offset in the memory frame.

CIS510

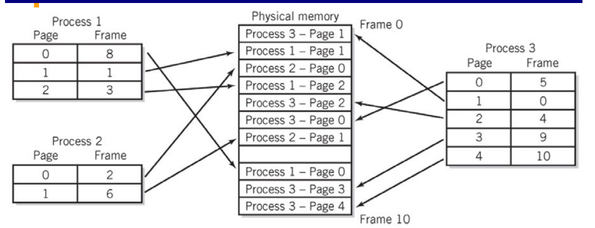
Address Mapping for Multiple Processes in Multitasking System

- With virtual storage, *each process in a multitasking system has its own virtual memory, and its own page table.*
- Physical memory is shared among the different processes.
- Since all the pages are of the same size, any frame may be placed anywhere in memory.
  - *The pages selected do not have to be contiguous to be placed in memory.*
- The ability to load any page into any frame solves the problem of finding enough contiguous memory space in which to load programs of different sizes.

113

CIS510

Mapping for Three Processes



Each process's page table points to a different area of physical memory. There is *no conflict* between different processes that use the same virtual addresses.

114

CIS510

How are Memory Frames Managed and Assigned to Pages?

- Physical memory is shared among all of the active processes in a system.
- Since each process has its own page table, it is not practical to identify available memory frames by accumulating data from all of the tables.
- Rather, there must be *a single resource* that identifies the entire pool of available memory frames from which the memory manager may draw, when required.
- It can be done by using the *inverted page table*.

115

CIS510

Inverted Page Table

- *Inverted Page Table* lists every memory frame with its associated process and page.
- The table represents what is in physical memory at every instant.
- Any frame without an associated page entry is available for allocation.

Frame	Process #	Page	
0	3	1	
1	1	1	
2	2	0	
3	1	2	
4	3	2	
5	3	0	
6	2	1	
7			Free page frame
8	1	0	
9	3	3	
10	3	4	

18-116

Page Replacement Algorithms

- Algorithms to manage swapping
  - FIFO – First-In, First-Out
  - LRU – Least Recently Used
  - NUR – Not Used Recently

117

FIFO – First-In, First-Out

- The simplest realistic page replacement algorithm.
- *The oldest page remaining in the page table is selected for replacement.*
- Not a good page replacement algorithm:
  - A page that has been in memory for a long period of time is probably there because it is heavily used.
  - *Belady's Anomaly* – when increasing number of page frames results in more page faults.
    - This phenomenon is commonly experienced when using the FIFO page replacement algorithm.

118

LRU – Least Recently Used

- Replace the page in the memory that has not been used for the longest time, on the assumption that the page probably will not be needed again.
- The algorithm performs *fairly well*, but requires a considerable amount of overhead.
  - To implement it, the page tables must record the time every time the page is referenced.
  - Then when page replacement is required, every page must be checked to find the page with the oldest recorded time.
  - If the number of pages is large, this can take a considerable amount of time.

119

NUR – Not Used Recently

- Add two additional bits for each entry in the page tables: *whole reference bit* and *dirty bit*.
- *Whole reference bit :* One bit is set (changed to 1) whenever the page is referenced (used).
- *Dirty bit :* The other bit is set (changed to 1) whenever that data on the page is modified, that is written to.

120

NUR – Not Used Recently (cont.)

- It is *a commonly used algorithm*.
- The memory manager software will attempt to find a page with both bits set to 0 to be replaced in memory.
  - This is the page that has not been used for a while and not been modified. So it is necessary only to write the new page over it.
- The second choice will be a page whole dirty bit is set, but whole reference bit is unset.
- The third choice will be a page that has been referenced, but not modified.
- Finally, *least desirable* will be a page that has been recently referenced and modified.

121

Thrashing

- A condition that can arise when a system is heavily loaded is called *thrashing*.
- Thrashing occurs when every frame of memory is in use, and programs are allocated just enough pages to meet their minimum requirement.
- A *page fault* occurs in a program, and the page is replaced by another page that will itself be needed for replacement almost immediately.
- Too many page faults affect system performance.

122

Virtual Memory Tradeoffs

Disadvantages

- SWAP file takes up space on disk.
- Paging takes up resources of the CPU.

Advantages

- Programs share memory space.
- More programs run at the same time.
- Programs run even if they cannot fit into memory all at once.
- Process separation

123

Virtual Memory vs. Caching

- Cache *speeds* up memory access.
- Virtual memory increases amount of *perceived storage.*
  - Independence from the configuration and capacity of the memory system
  - Low cost per bit compared to main memory

124