)Introduction to Operating System (OS

:Course Content

- What is an OS.
- What are its key functions.
- The evaluation of OS.
- What are the popular types of OS.
- Basics of UNIX and Windows.
- Advantages of open source OS like Linux.
- Networks OS.

- User
- Application
- Operating System
- Hardware

?What is an Operating System

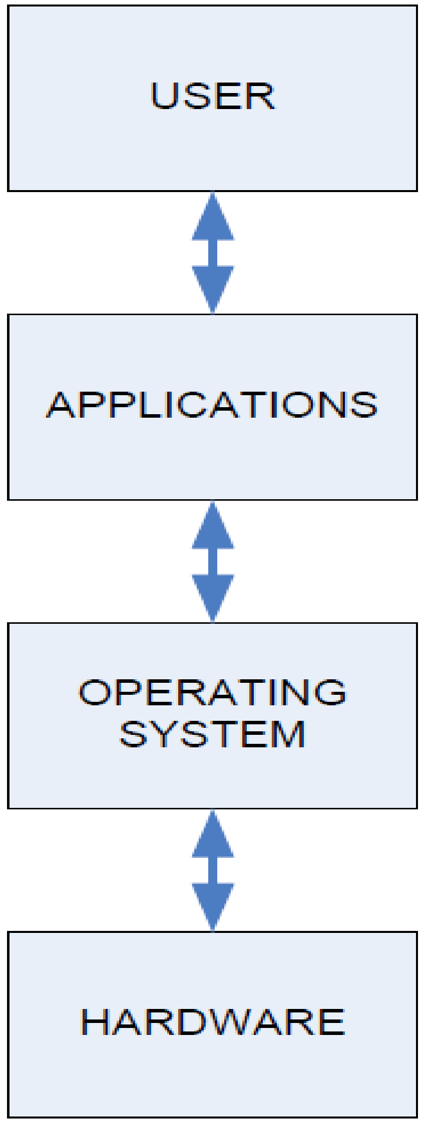
- Computer System = Hardware + Software
- Software = Application Software + System Software(OS)
- An Operating System is a system Software that acts as an intermediary/interface between a user of a computer and the computer hardware.
- Operating system goals:
  - ➢ Execute user programs and make solving user problems easier
  - ➢ Make the computer system convenient to use
  - ➢ Use the computer hardware in an efficient manner

Features of OS

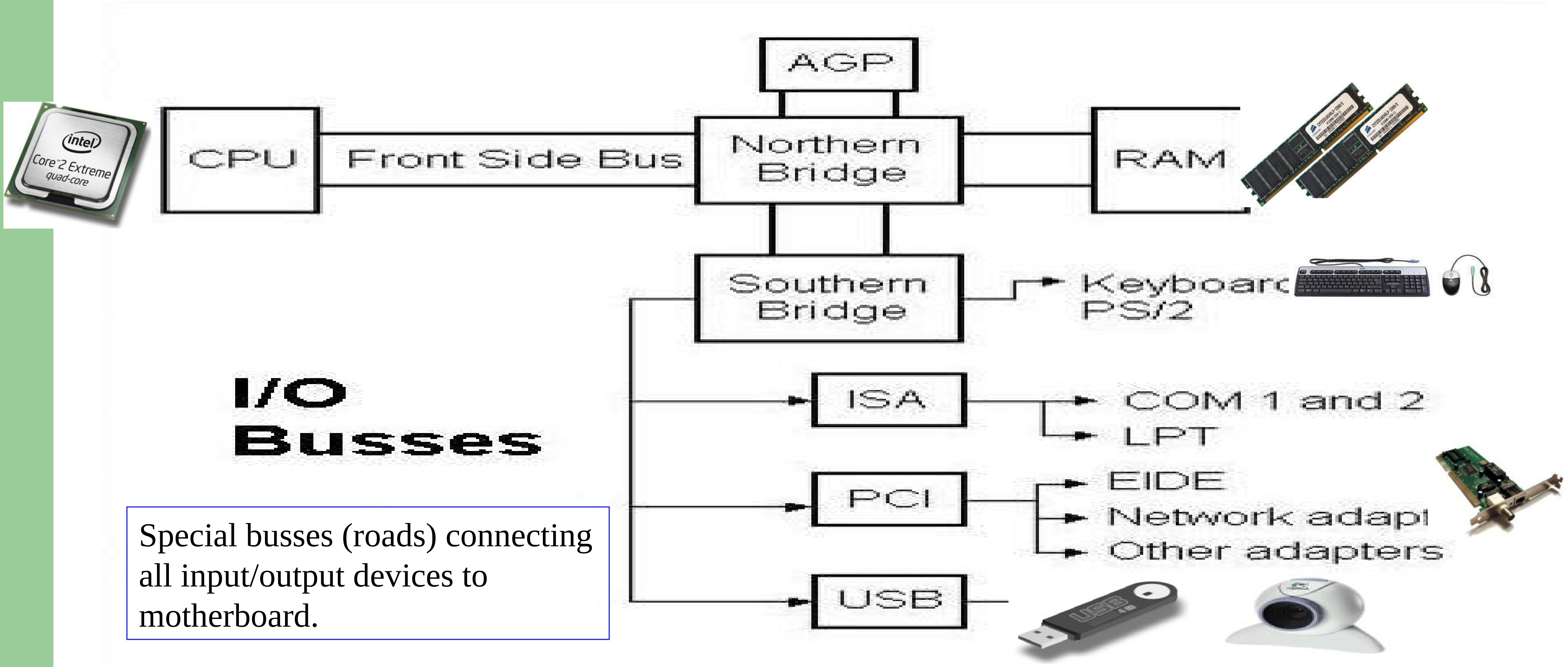
- Convenience
- Efficiency
- Ability to Evolve
- Throughput
- Resource management
- Process management
- Storage management
- Memory management
- Security / Privacy management

The Structure of Computer Systems

- Accessing computer resources is divided into *layers*.
- Each layer is isolated and only interacts directly with the layer below or above it.
- **If we install a new hardware device**
  - ✓ No need to change anything about the user/applications.
  - ✓ However, you do need to make changes to the operating system.
  - ✓ You need to install the device drivers that the operating system will use to control the new device.
- **If we install a new software application**
  - ✓ No need to make any changes to your hardware.
  - ✓ But we need to make sure the application is supported by the operating system
  - ✓ user will need to learn how to use the new application.
- **If we change the operating system**
  - ✓ Need to make sure that both applications and hardware will compatible with the new operating system.



Computer Architecture



Special busses (roads) connecting all input/output devices to motherboard.

CPU – Central Processing Unit

- This is the brain of your computer.
- It performs all of the calculations.
- In order to do its job, the CPU needs commands to perform, and data to work with.
- The instructions and data travel to and from the CPU on the system bus.
- The operating system provides rules for how that information gets back and forth, and how it will be used by the CPU.

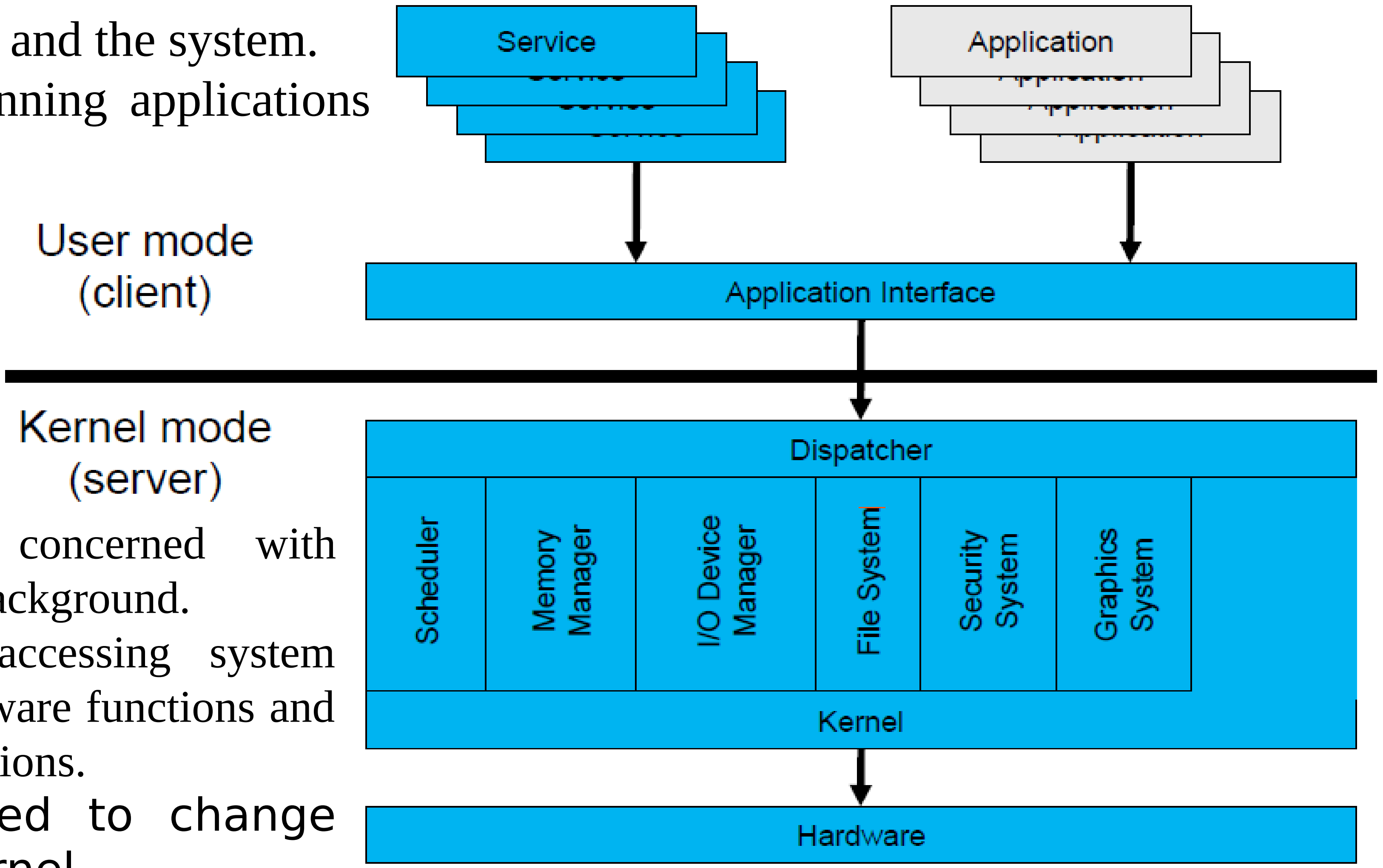
RAM – Random Access Memory

- This is like a desk, or a workspace, where your computer temporarily stores all of the information (data) and instructions (software or program code) that it is currently using.
- Each RAM chip contains millions of address spaces.
- Each address space is the same size, and has its own unique identifying number (address).
- The operating system provides the rules for using these memory spaces, and controls storage and retrieval of information from RAM.
- Device drivers for RAM chips are included with the operating system.

Problem: *If RAM needs an operating system to work, and an operating system needs RAM in order to work, how does your computer activate its RAM to load the operating system?*

Operating System Mode

- ❖ The *User Mode* is concerned with the actual interface between the user and the system.
- ❖ It controls things like running applications and accessing files.



- ❖ The *Kernel Mode* is concerned with everything running in the background.
- ❖ It controls things like accessing system resources, controlling hardware functions and processing program instructions.
- ❖ System calls are used to change mode from User to Kernel.

Kernel

- Kernel is a software code that reside in central core of OS. It has complete control over system.
- When operation system boots, kernel is first part of OS to load in main memory.
- Kernel remains in main memory for entire duration of computer session. The kernel code is usually loaded in to protected area of memory.
- Kernel performs it's task like executing processes and handling interrupts in kernel space.
- User performs it's task in user area of memory.
- This memory separation is made in order to prevent user data and kernel data from interfering with each other.
- Kernel does not interact directly with user, but it interacts using SHELL and other programs and hardware.

…Kernel cont

- Kernel includes:-

  1. Scheduler: It allocates the Kernel’s processing time to various processes.

  2. Supervisor: It grants permission to use computer system resources to each process.

  3. Interrupt handler : It handles all requests from the various hardware devices which compete for kernel services.

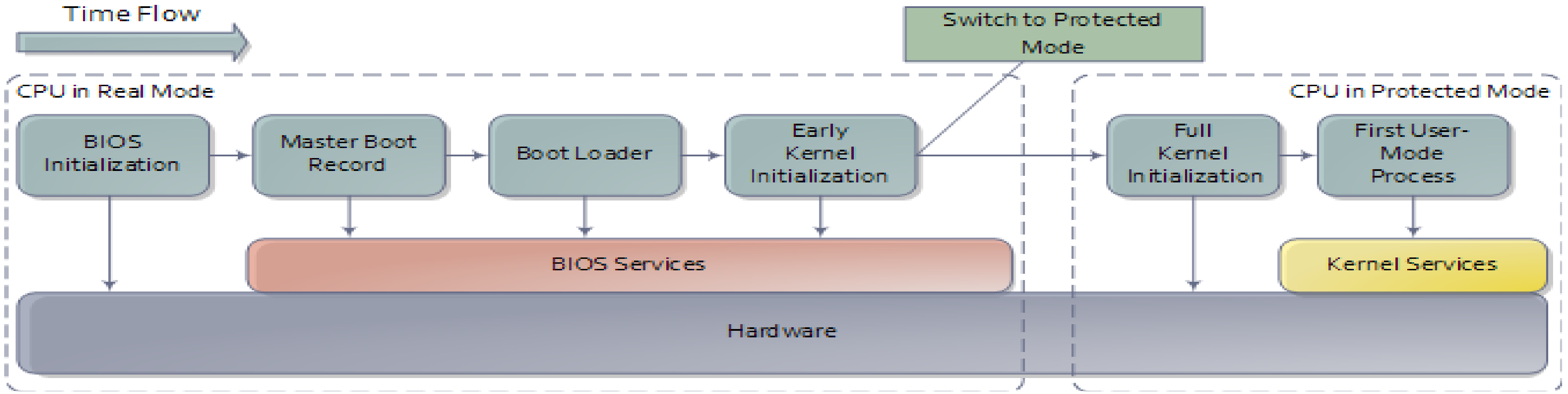
  4. Memory manager : allocates space in memory for all users of kernel service.

- kernel provides services for process management, file management, I/O management, memory management.
- System calls are used to provide these type of services.

System Call

- **System call** is the programmatic way in which a computer program/user application requests a service from the kernel of the operating system on which it is executed.
- Application program is just a user-process. Due to security reasons , user applications are not given access to privileged resources(the ones controlled by OS).
- When they need to **do any I/O** or have **some more memory** or **spawn a process** or wait for **signal/interrupt,** it requests operating system to facilitate all these. This **request is made through System Call.**
- System calls are also called **software-interrupts.**

)Starting an Operating System(Booting



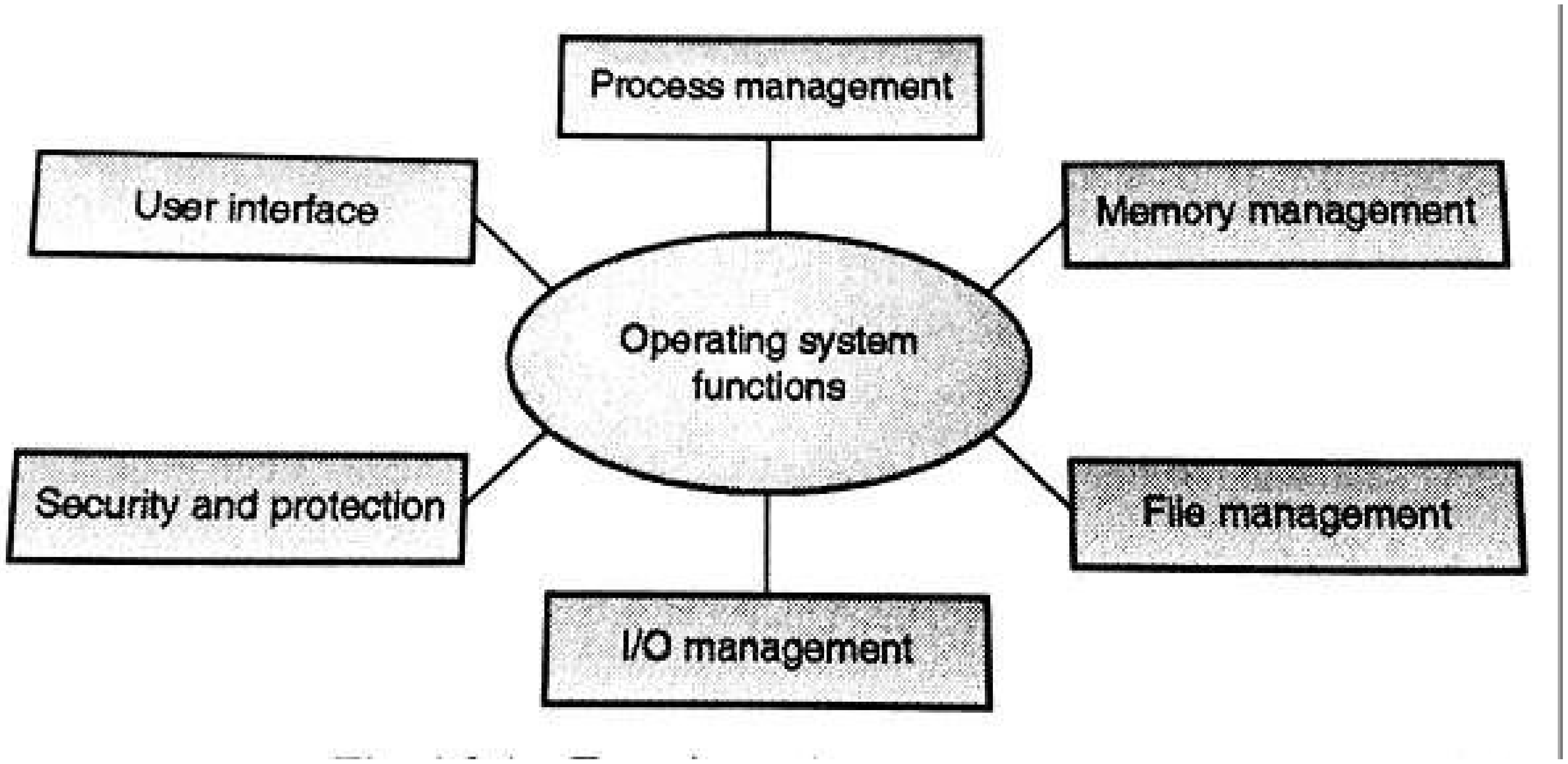
- ✓ Power On Switch sends electricity to the motherboard on a wire called the *Voltage Good line*.
- ✓ If the power supply is good, then the BIOS (Basic Input/Output System) chip takes over.
- ✓ In Real Mode, CPU is only capable of using approximately 1 MB of memory built into the motherboard.
- ✓ The BIOS will do a Power-On Self Test (POST) to make sure that all hardware are working.

- ✓ BIOS will then look for a small sector at the very beginning of your primary hard disk called MBR.
- ✓ The MBR contains a list, or map, of all of the partitions on your computer‘s hard disk (or disks).
- ✓ After the MBR is found the Bootstrap Loader follows basic instructions for starting up the rest of the computer, including the operating system.
- ✓ In Early Kernel Initialization stage, a smaller core of the Kernel is activated.
- ✓ This core includes the device drivers needed to use computer‘s RAM chips.

BIOS

- BIOS firmware was stored in a ROM/EPROM (Erasable Programmable Read-Only Memory) chip known as **firmware** on the PC motherboard.
- BIOS can be accessed during the initial phases of the boot procedure by pressing del, F2 or F10.
- Finally, the firmware code cycles through all storage devices and looks for a boot-loader. (usually located in first sector of a disk which is 512 bytes)
- If the boot-loader is found, then the firmware hands over control of the computer to it.

Functions of Operating System



Process Management. 1

- **A *process* is a program in execution.**
- A process needs certain resources, including CPU time, memory, files, and I/O devices to accomplish its task.
- Simultaneous execution leads to multiple processes. Hence creation, execution and termination of a process are the most basic functionality of an OS
- If processes are dependent, than they may try to share same resources. thus task of process synchronization comes to the picture.
- If processes are independent, than a due care needs to be taken to avoid their overlapping in memory area.
- Based on priority, it is important to allow more important processes to execute first than others.

Memory management. 2

- Memory is a large array of words or bytes, each with its own address.
- It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a **volatile** storage device. When the computer made turn off everything stored in RAM will be erased automatically.
- In addition to the physical RAM installed in your computer, most modern operating systems allow your computer to use a *virtual memory system. Virtual memory allows your computer to use part of a permanent storage device (such as a hard disk) as extra memory.*
- The operating system is responsible for the following activities in connections with memory management:
  - ➢ Keep track of which parts of memory are currently being used and by whom.
  - ➢ Decide which processes to load when memory space becomes available.
  - ➢ Allocate and de-allocate memory space as needed.

File Management. 3

- A file is a collection of related information defined by its creator.
- *File systems provide the conventions for the encoding, storage and management of data on a storage device such as a hard disk.*
  - FAT12 (floppy disks)
  - FAT16 (DOS and older versions of Windows)
  - FAT32 (older versions of Windows)
  - NTFS (newer versions of Windows)
  - EXT3 (Unix/Linux)
  - HFS+ (Max OS X)
- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

Device Management or I/O Management. 4

- *Device controllers are components on the motherboard (or on expansion cards) that act as an interface between the CPU and the actual device.*
- *Device drivers, which are the operating system software components that interact with the devices controllers.*
- A special device (inside CPU) called the Interrupt Controller handles the task of receiving interrupt requests and prioritizes them to be forwarded to the processor.
- Deadlocks can occur when two (or more) processes have control of different I/O resources that are needed by the other processes, and they are unwilling to give up control of the device.
- It performs the following activities for device management.
  - ➢ Keeps tracks of all devices connected to system.
  - ➢ Designates a program responsible for every device known as Input/output controller.
  - ➢ Decides which process gets access to a certain device and for how long.
  - ➢ Allocates devices in an effective and efficient way.
  - ➢ Deallocates devices when they are no longer required.

Security & Protection. 5

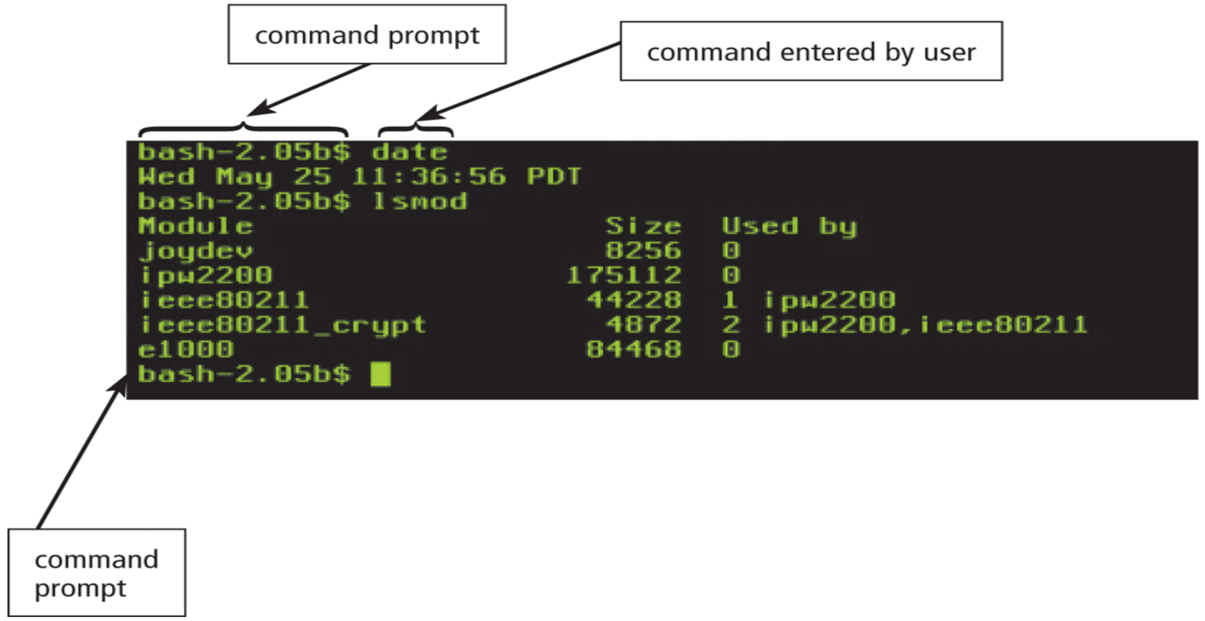
- The operating system uses password protection to protect user data and similar other techniques.
- It also prevents unauthorized access to programs and user data by assigning access right permission to files and directories.
- The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.

User Interface Mechanism. 6

- A **user interface** (**UI**) controls how you enter data and instructions and how information is displayed on the screen
- There are two types of user interfaces
  1. Command Line Interface
  2. Graphical user Interface

Command-line interface. 1

- In a command-line interface, a user types commands represented by short keywords or abbreviations or presses special keys on the keyboard to enter data and instructions



```
bash-2.05b$ date
Wed May 25 11:36:56 PDT
bash-2.05b$ lsmod
Module                  Size  Used by
joydev                  8256  0
ipw2200               175112  0
ieee80211              44228  1 ipw2200
ieee80211_crypt         4872  2 ipw2200,ieee80211
e1000                  84468  0
bash-2.05b$ █
```

Graphical User Interface. 2

- With a graphical user interface (GUI), you interact with menus and visual images

History of Operating System

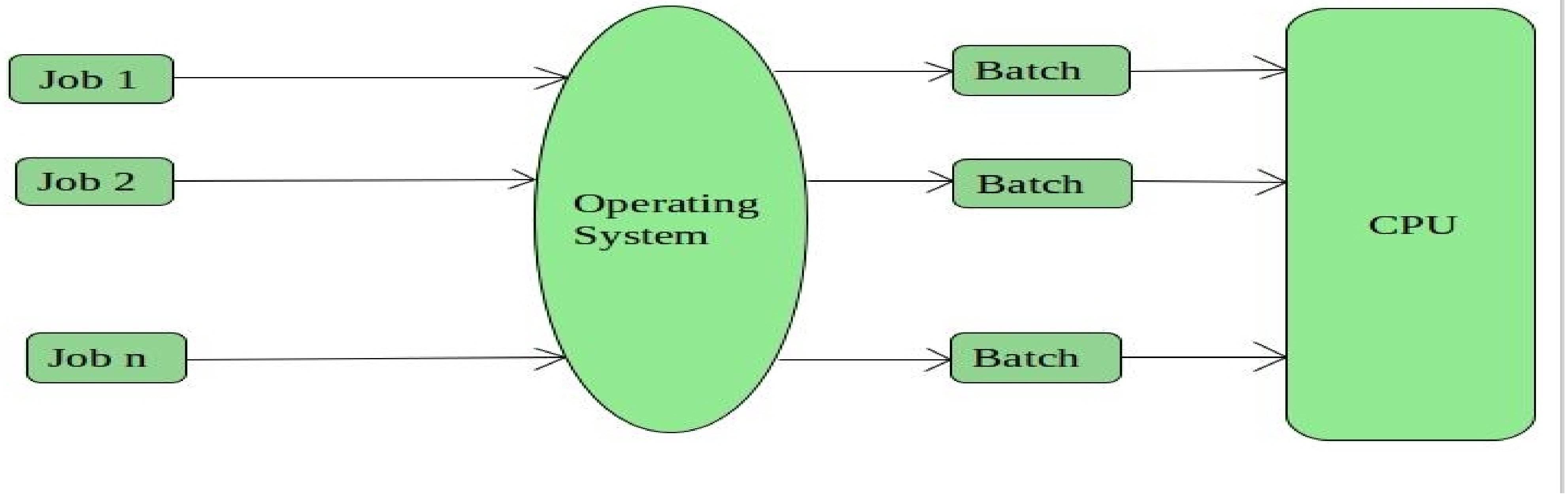
- **The First Generation (1940's to early 1950's)**
  - No Operating System
  - All programming was done in absolute machine language, often by wiring up plug-boards to control the machine’s basic functions.
- **The Second Generation (1955-1965)**
  - First operating system was introduced in the early 1950's.It was called GMOS
  - Created by General Motors for IBM's machine the 701.
  - Single-stream batch processing systems
- **The Third Generation (1965-1980)**
  - Introduction of multiprogramming
  - Development of Minicomputer
- **The Fourth Generation (1980-Present Day)**
  - Development of PCs
  - Birth of Windows/MaC OS

Types of Operating Systems

1. Batch Operating System
2. Multiprogramming Operating System
3. Time-Sharing OS
4. Multiprocessing OS
5. Distributed OS
6. Network OS
7. Real Time OS
8. Embedded OS

Batch Operating System. 1

- The users of this type of operating system does not interact with the computer directly.
- Each user prepares his job on an off-line device like punch cards and submits it to the computer operator
- There is an operator which takes similar jobs having the same requirement and group them into batches.



..Batch Operating System cont. 1

Advantages of Batch Operating System:

- Processors of the batch systems know how long the job would be when it is in queue
- Multiple users can share the batch systems
- The idle time for the batch system is very less
- It is easy to manage large work repeatedly in batch systems

Disadvantages of Batch Operating System:

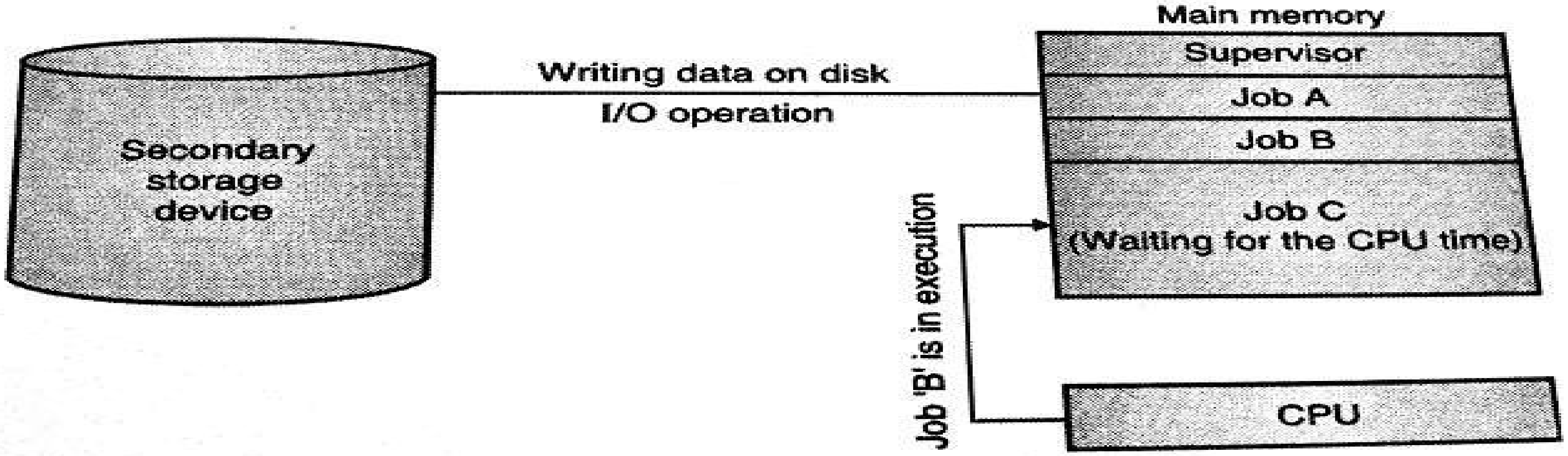
- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometimes costly
- The other jobs will have to wait for an unknown time if any job fails

Examples of Batch based Operating System:

IBM's MVS

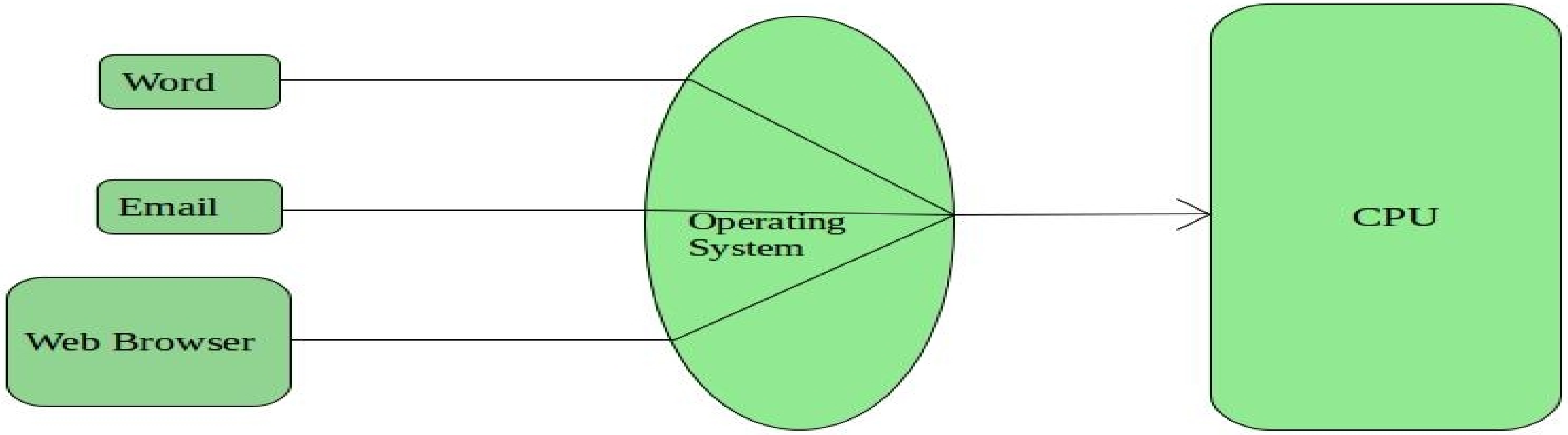
:Multiprogramming Operating System. 2

- This type of OS is used to execute more than one jobs simultaneously by a single processor.
- It increases CPU utilization by organizing jobs so that the CPU always has one job to execute.
- Multiprogramming operating systems use the mechanism of job scheduling and CPU scheduling.



Time-Sharing Operating Systems. 3

- Each task is given some time to execute so that all the tasks work smoothly.
- These systems are also known as **Multi-tasking Systems.**
- The task can be from a single user or different users also.
- The time that each task gets to execute is called quantum.
- After this time interval is over OS switches over to the next task.



..Time-Sharing Operating Systems cont. 3

- **Advantages of Time-Sharing OS:**
  - ➢ Each task gets an equal opportunity
  - ➢ Fewer chances of duplication of software
  - ➢ CPU idle time can be reduced
- **Disadvantages of Time-Sharing OS:**
  - ➢ Reliability problem
  - ➢ One must have to take care of the security and integrity of user programs and data
  - ➢ Data communication problem
- **Examples of Time-Sharing Oss**

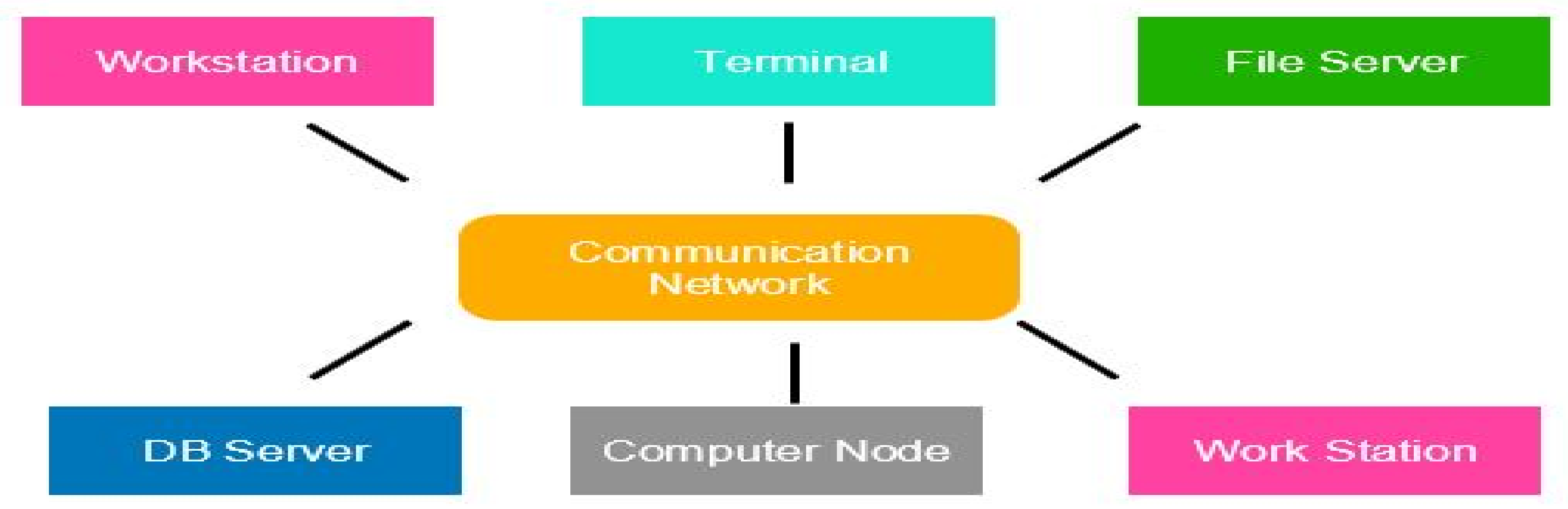
  Multics, Unix, etc.

Multiprocessor operating systems. 4

- *Multiprocessor operating systems are also known as parallel OS or tightly coupled OS.*
- *Such operating systems have more than one processor in close communication that sharing the computer bus, the clock and sometimes memory and peripheral devices.*
- *It executes multiple jobs at the same time and makes the processing faster.*
- *It supports large physical address space and larger virtual address space.*
- *If one processor fails then other processor should retrieve the interrupted process state so execution of process can continue.*
- *Inter-processes communication mechanism is provided and implemented in hardware.*

Distributed Operating System. 5

- Various autonomous interconnected computers communicate with each other using a shared communication network.
- Independent systems possess their own memory unit and CPU.
- These are referred to as **loosely coupled systems.**
- Examples:- Locus, DYSEAC



Network Operating System. 6

- These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions.
- These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network.
- The “ other" computers arc called client computers, and each computer that connects to a network server must be running client software designed to request a specific service.
- popularly known as **tightly coupled systems**.

Network Operating System. 6

Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated into the system
- Server access is possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on a central location for most operations
- Maintenance and updates are required regularly

Examples of Network Operating System are:

Microsoft Windows Server 2003/2008/2012, UNIX, Linux, Mac OS X, Novell NetWare, and BSD, etc.

Real-Time Operating System. 7

- These types of OSs serve real-time systems.
- The time interval required to process and respond to inputs is very small.
- This time interval is called **response time**.
- **Real-time systems** are used when there are time requirements that are very strict like
  - ➢ missile systems,
  - ➢ air traffic control systems,
  - ➢ robots, etc.

Embaded Operating System. 8

- An embedded operating system is one that is built into the circuitry of an electronic device.
- Embedded operating systems are now found in automobiles, bar-code scanners, cell phones, medical equipment, and personal digital assistants.
- The most popular embedded operating systems for consumer products, such as PDAs, include the following:
  - ➢ Windows XP Embedded
  - ➢ Windows CE .NET:- it supports wireless communications, multimedia and Web browsing. It also allows for the use of smaller versions of Microsoft Word, Excel, and Outlook.
  - ➢ Palm OS:- It is the standard operating system for Palm-brand PDAs as well as other proprietary handheld devices.
  - ➢ Symbian:- OS found in " smart" cell phones from Nokia and Sony Ericsson

Popular types of OS

- Desktop Class
  - Windows
  - OS X
  - Unix/Linux
  - Chrome OS
- Server Class
  - Windows Server
  - Mac OS X Server
  - Unix/Linux
- Mobile Class
  - Android
  - iOS
  - Windows Phone

:-Desktop Class Operating Systems

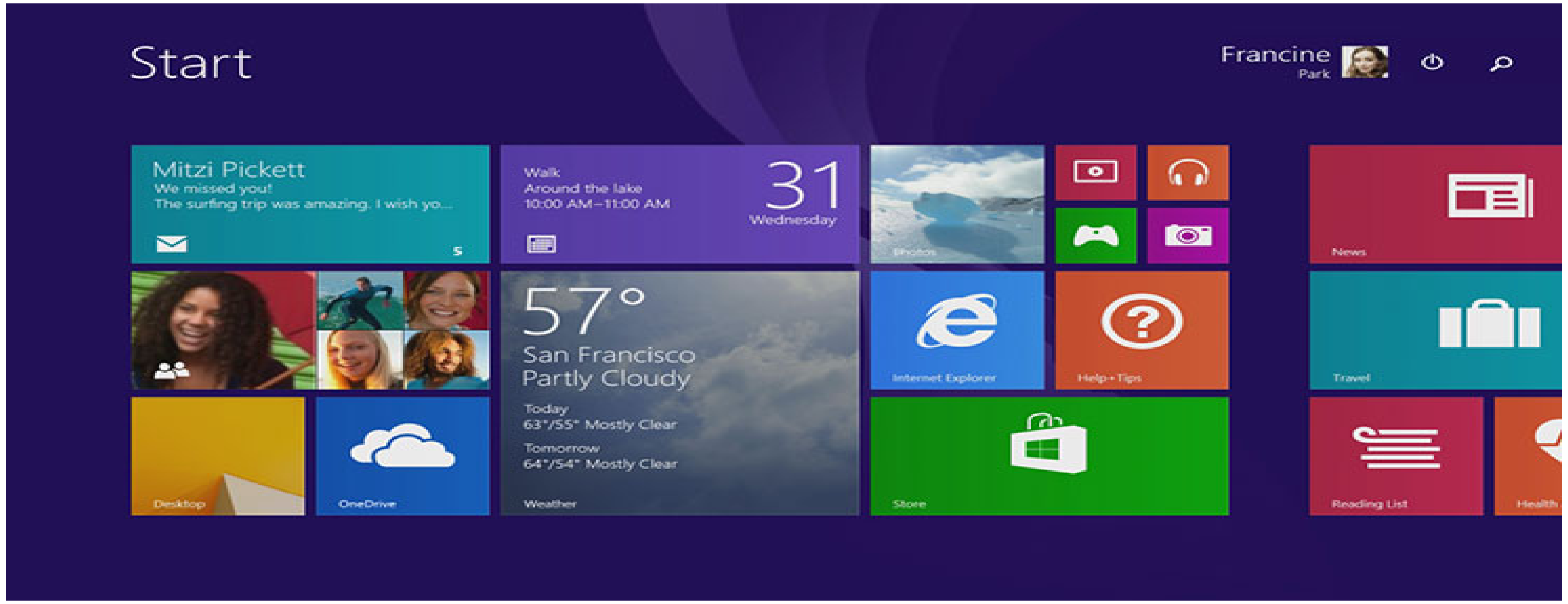
- **Platform:** the hardware required to run a particular operating system
  - Intel platform (IBM-compatible)
    - Windows
    - DOS
    - UNIX
    - Linux
  - Macintosh platform
    - Mac OS
  - iPad and iPhone platform
    - iOS

Ms-DOS

- Single User Single Tasking OS.
- It had no built-in support for networking, and users had to manually install drivers any time they added a new hardware component to their PC.
- DOS supports only 16-bit programs.
- Command line user interface.
- So, why is DOS still in use? Two reasons are its size and simplicity. It does not require much memory or storage space for the system, and it docs not require a powerful computer.

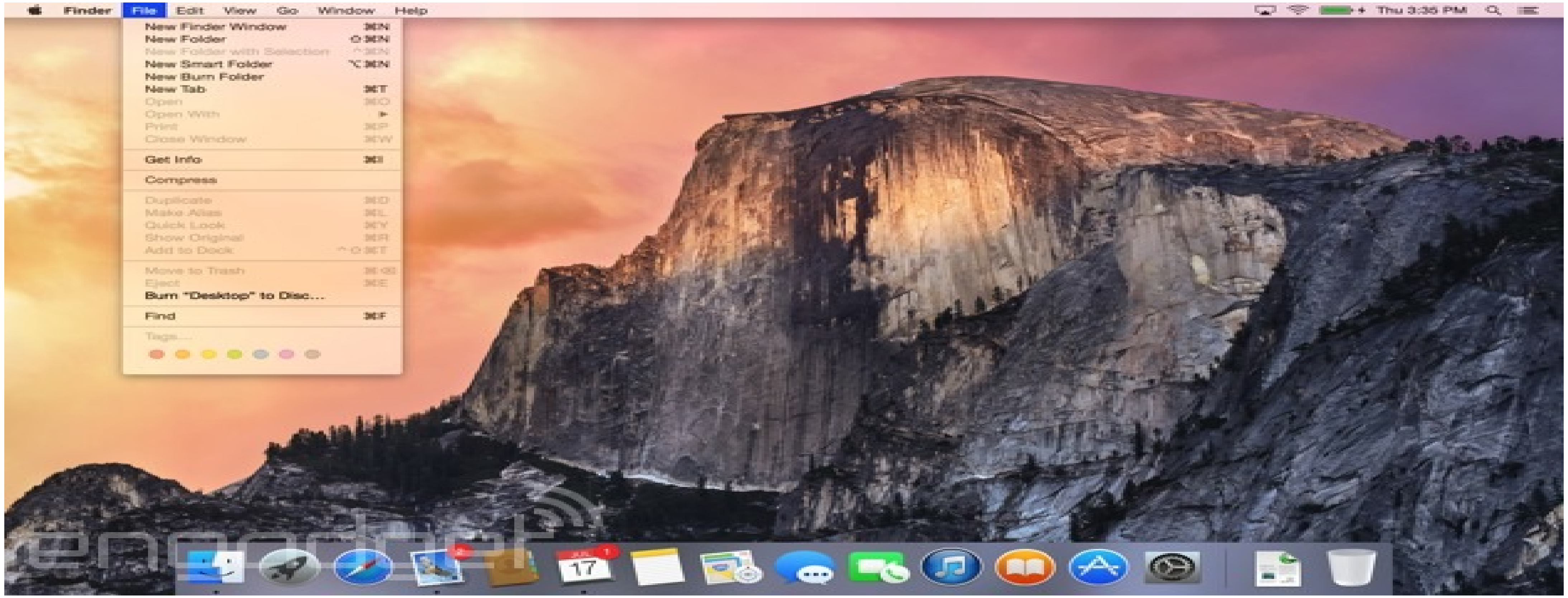
Microsoft Windows

- The graphical Microsoft operating system designed for Intel-platform desktop and notebook computers.
- Best known, greatest selection of applications available.
- Current editions include Windows 7, 8, 8.1 and 10.

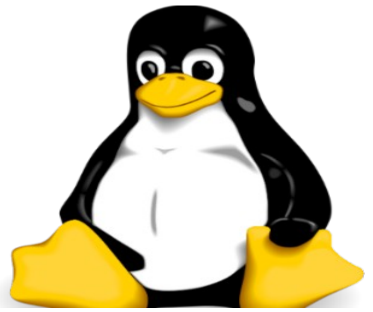


Mac OS

- User-friendly, runs on Mac hardware. Many applications available.
- Current editions include: Sierra, High Sierra, Mojave, Catalina & Big Sur—Version XI(Released in Nov 2020)



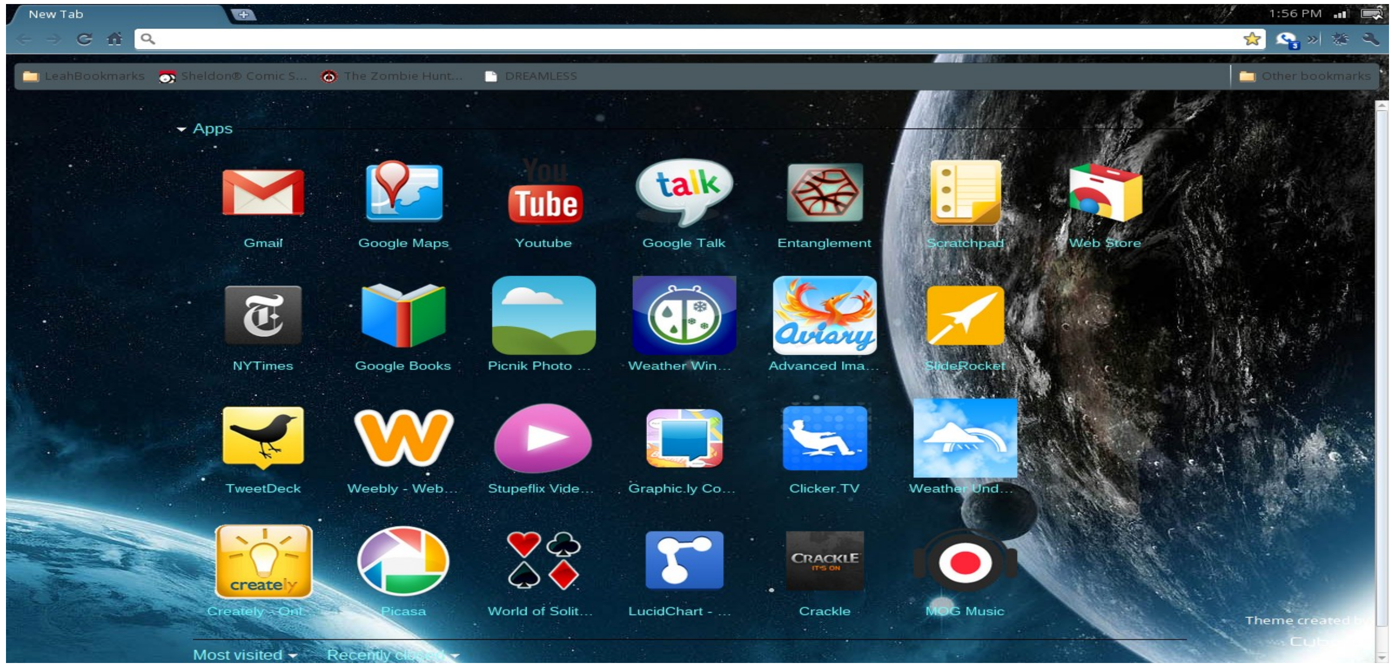
Linux



- **Linux:** An open-source, cross-platform operating system that runs on desktops, notebooks, tablets, and smartphones.
  - The name *Linux* is a combination *Linus* (the first name of the first developer) and *UNIX (another operating system*.
- Users are free to modify the code, improve it, and redistribute it,
- Developers are not allowed to charge money for the Linux kernel itself (the main part of the operating system), but they can charge money for **distributions** (**distros** for short).

Google Chrome OS

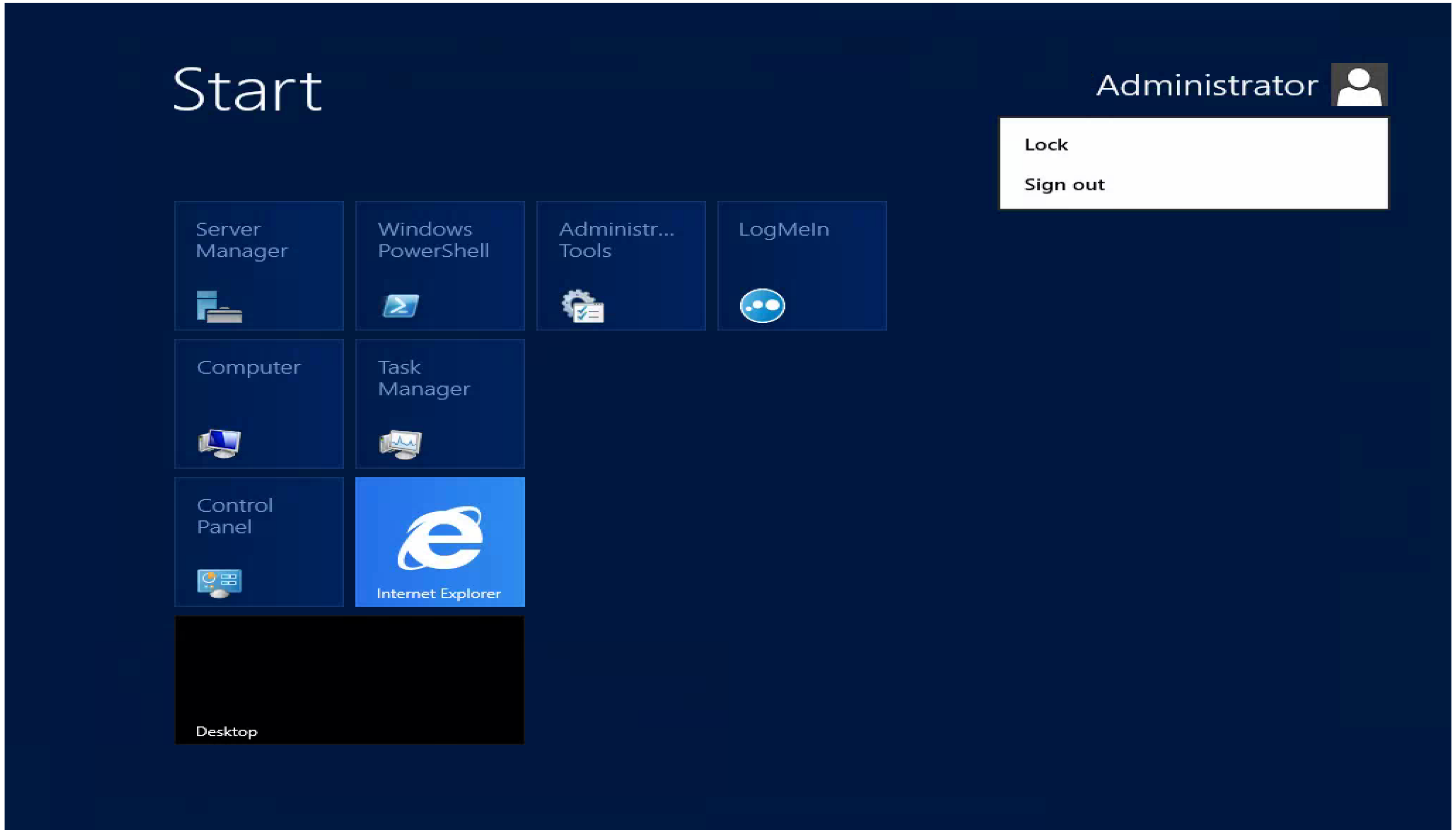
- **Chrome OS**. Is a popular thin client operating system.
- **Thin client** A computer with minimal hardware, designed for a specific task. For example, a thin web client is designed for using the Internet.



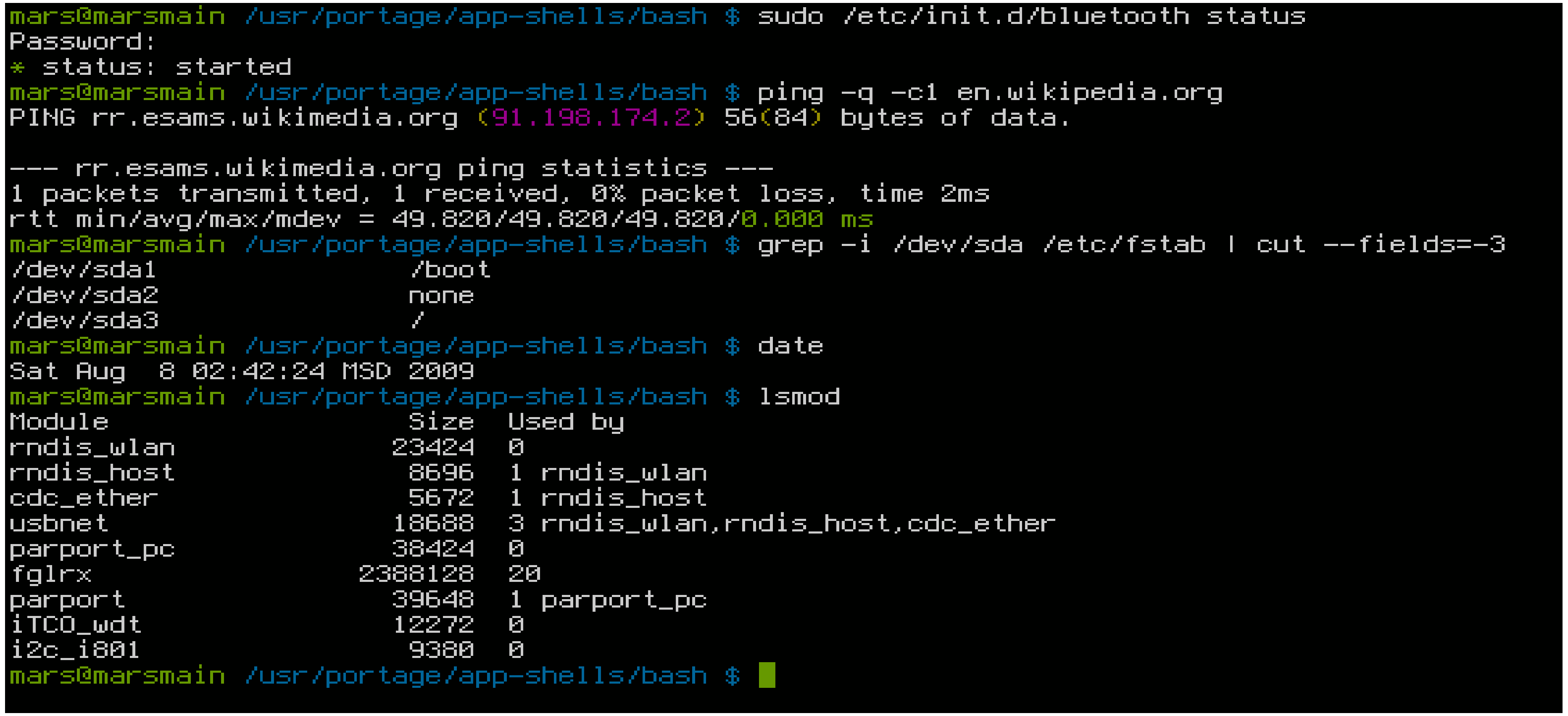
Server Operating Systems

- Windows Server
  - Familiar GUI interface for those experienced with Windows
- UNIX
  - Very mature server capabilities, time-tested, large user community, stable
- Linux
  - Free, customizable, many free services and utilities available

Windows Server



UNIX



```
mars@marsmain /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
mars@marsmain /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esams.wikimedia.org (91.198.174.2) 56(84) bytes of data.

--- rr.esams.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.820/49.820/49.820/0.000 ms
mars@marsmain /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /boot
/dev/sda2               none
/dev/sda3               /
mars@marsmain /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
mars@marsmain /usr/portage/app-shells/bash $ lsmod
Module                  Size  Used by
rndis_wlan             23424  0
rndis_host              8696  1 rndis_wlan
cdc_ether               5672  1 rndis_host
usbnet                 18688  3 rndis_wlan,rndis_host,cdc_ether
parport_pc             38424  0
fglrx                2388128  20
parport                39648  1 parport_pc
iTCO_wdt               12272  0
i2c_i801                9380  0
mars@marsmain /usr/portage/app-shells/bash $ 
```

Tablet and Phone Operating Systems

- **System-on-chip (SoC):** An operating system that comes preinstalled on a chip on a portable device such as a smartphone.
- Popular SoC operating systems:
  - iOS: for iPad, iPhone
  - Android: for a variety of tablets and phones
- Downloadable applications (apps) from an App store, for example:
  - Apple App Store
  - Google Play Store

ANDROID APP ON Google play

Available on the App Store

iOS on the iPhone and iPad

- The Apple-created operating system for Apple tablets and phones.
- The current stable version, iOS 14, was released to the public on September 16, 2020.

"Remind me about this when I get home"
tap to edit
OK, I'll remind you when you arrive at home:
Reminder
11 Hot Beach Escapes - Jetsetter
Arriving: Home
Remove
Search
SIRI SUGGESTIONS
Show More
Nisha
David
Erin
Steve
Twitter
KAYAK
Instagram
OpenTable
NEARBY
Food
Drink
Shopping
Fun
NEWS
NASA Curiosity Rover Captures Sunset On Mars
Healthy diet may improve memory, says study - CNN.com
"Show me photos from Utah last August"
tap to edit

Android

- Android, a popular OS for smartphones and tablets, is based on Linux Kernel.
  - Developed by Google
- Current versions include:
  - Android 8 Oreo
  - Android 9 Pie
  - Android 10
  - Android 11 (released on Sep, 2020)

Advantage of Linux Operating System

1. Open Source

As it is open-source, its source code is easily available.

Anyone having programming knowledge can customize the operating system.

One can contribute, modify, distribute, and enhance the code for any purpose.

2. Security

The Linux security feature is the main reason that it is the most favourable option for developers.

It is not completely safe, but it is less vulnerable than others.

Each application needs to authorize by the admin user.

Linux systems do not require any antivirus program.

3. Free

Certainly, the biggest advantage of the Linux system is that it is free to use.

We can easily download it, and there is no need to buy the license for it.

It is distributed under GPL (General Public License).

Comparatively, we have to pay a huge amount for the license of the other OS

Advantage of Linux Operating System

4. Lightweight

The requirements for running Linux are much less than other operating system

In Linux, the memory footprint and disk space are also lower.

Generally, most of the Linux distributions required as little as 128MB of RAM around the same amount for disk space.

5. Stability

Linux is more stable than other operating systems.

Linux does not require to reboot the system to maintain performance levels.

It rarely hangs up or slow down. It has big up-times.

Advantage of Linux Operating System

6. Performance

Linux system provides high performance over different networks.

It is capable of handling a large number of users simultaneously.

7. Flexibility

Linux operating system is very flexible.

It can be used for desktop applications, embedded systems, and server applications too.

It also provides various restriction options for specific computers.

We can install only necessary components for a system.

8. Software Updates

In Linux, the software updates are in user control.

We can select the required updates.

There a large number of system updates are available.

These updates are much faster than other operating systems.

So, the system updates can be installed easily without facing any issue.

Advantage of Linux Operating System

9. Distributions/ Distros

There are many Linux distributions available in the market.

It provides various options and flavors of Linux to the users.

We can choose any distros according to our needs.

Some popular distros are **Ubuntu, Fedora, Debian, Linux Mint, Arch Linux,**

For the beginners, Ubuntu and Linux Mint would be useful.

Debian and Fedora would be good choices for proficient programmers.

10. Live CD/USB

Almost all Linux distributions have **a Live CD/USB** option.

It allows us to try or run the Linux operating system without installing it.

11. Graphical User Interface

Linux is a command-line based OS but it provides an interactive user interface like Windows.

Advantage of Linux Operating System

12. Suitable for programmers

It supports almost all of the most used programming languages such as C/C++, Java, Python, Ruby, and more.

Further, it offers a vast range of useful applications for development.

The programmers prefer the Linux terminal over the Windows command line.

The package manager on Linux system helps programmers to understand how things are done.

Bash scripting is also a functional feature for the programmers.

It also provides support for SSH, which helps in managing the servers quickly.

13. Community Support

Linux provides large community support.

We can find support from various sources.

There are many forums available on the web to assist users.

Further, developers from the various open source communities are ready to help us.

Advantage of Linux Operating System

14. Privacy

Linux always takes care of user privacy as it never takes much private data from the user. Comparatively, other operating systems ask for the user's private data.

15. Networking

Linux facilitates with powerful support for networking. The client-server systems can be easily set to a Linux system. It provides various command-line tools such as ssh, ip, mail, telnet, and more for connectivity with the other systems and servers. Tasks such as network backup are much faster than others.

16. Compatibility

Linux is compatible with a large number of file formats as it supports almost all file formats.

17. Installation

Linux installation process takes less time than other operating systems such as Windows. Further, its installation process is much easy as it requires less user input. It does not require much more system configuration even it can be easily installed on old machines having less configuration.

Advantage of Linux Operating System

18. Multiple Desktop Support

Linux system provides multiple desktop environment support for its enhanced use. The desktop environment option can be selected during installation. We can select any desktop environment such as **GNOME (GNU Network Object Model Environment)** or **KDE (K Desktop Environment)** as both have their specific environment.

19. Multitasking

It is a multitasking operating system as it can run multiple tasks simultaneously without affecting the system speed.

20. Heavily Documented for beginners

There are many command-line options that provide documentation on commands, libraries, standards such as manual pages and info pages. Also, there are plenty of documents available on the internet in different formats, such as Linux tutorials, Linux documentation project, Serverfault, and more. To help the beginners, several communities are available such as **Ask Ubuntu**, Reddit, and **StackOverflow.**

Thank you!

Unit 3

CPU Scheduling

CPU Scheduling

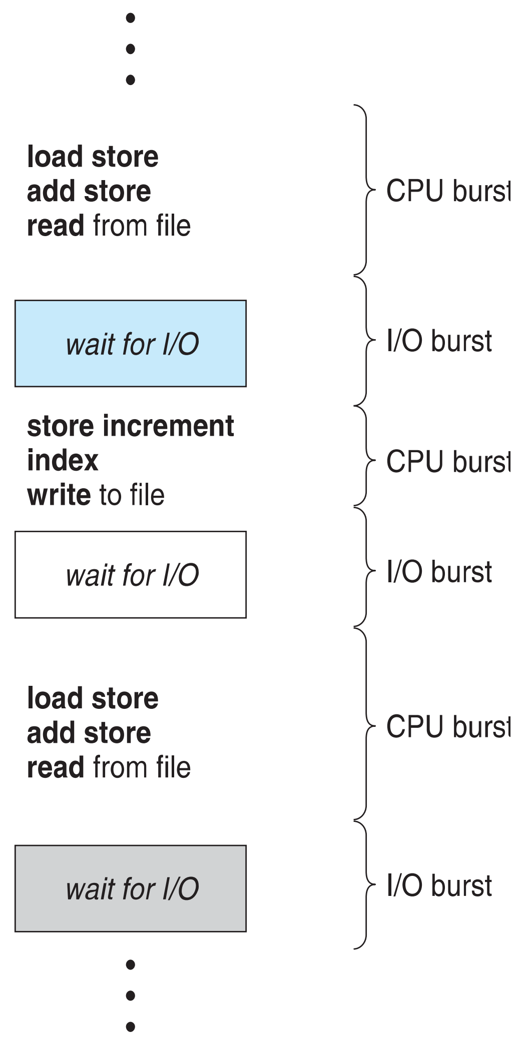
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

Objectives

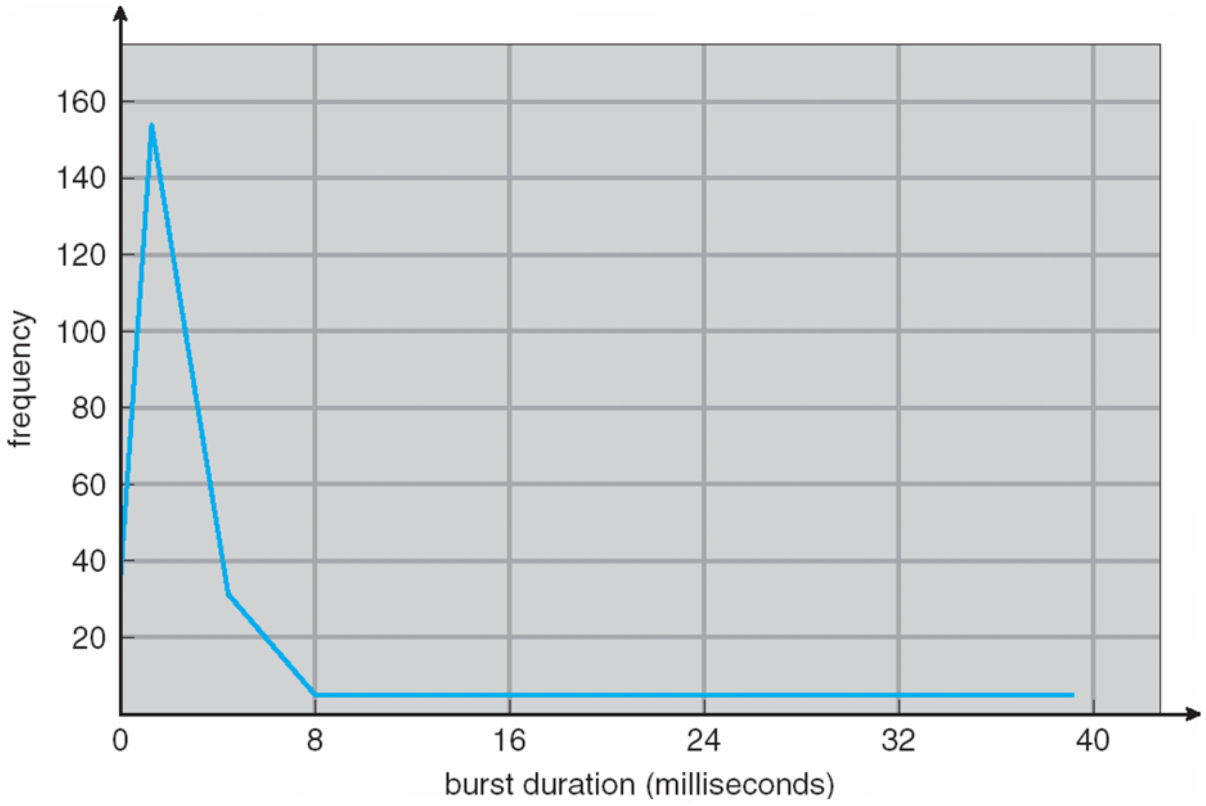
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



Histogram of CPU-burst Times



CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

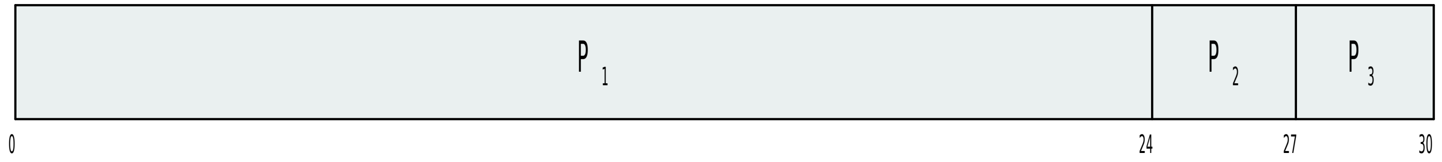
Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



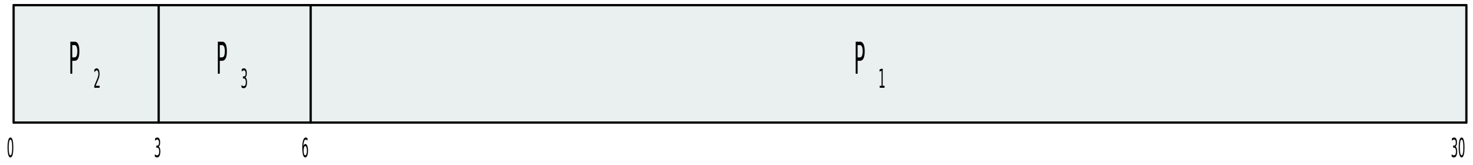
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

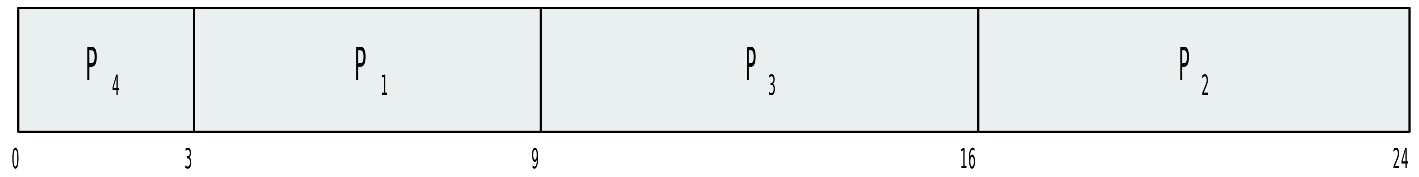
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Determining Length of Next CPU Burst

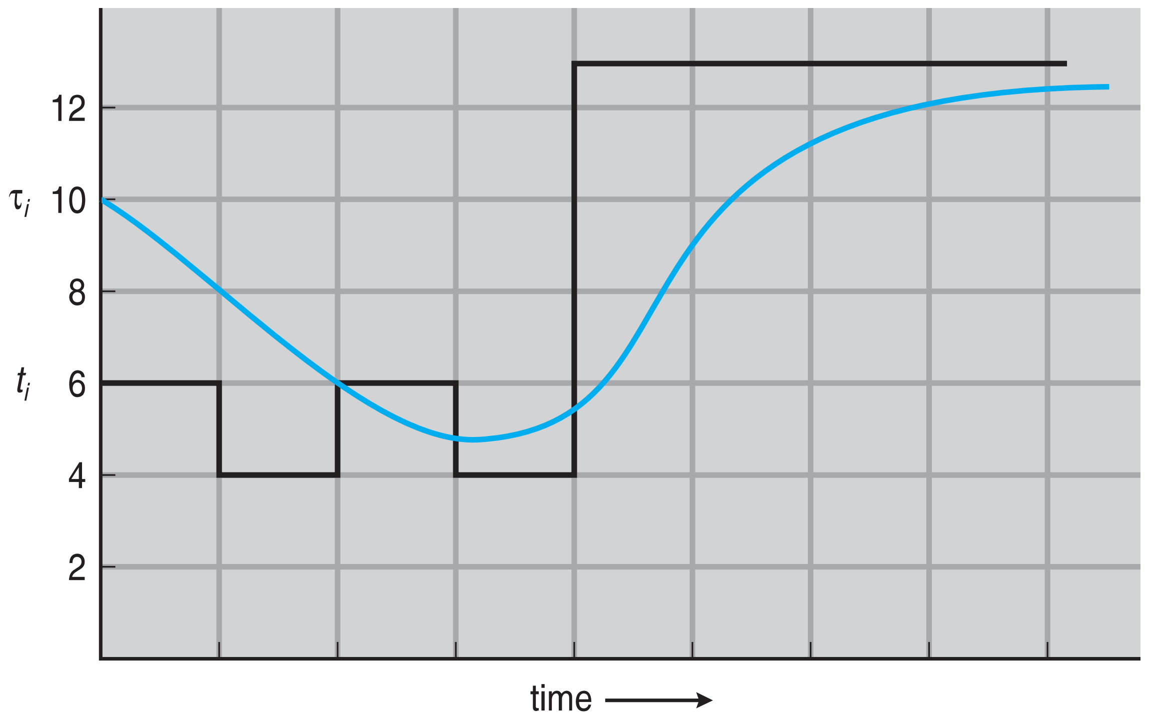
- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst

- Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define: $\tau_{n=1} = \alpha\, t_n + (1- \alpha)\tau_n.$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} &= \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots \\ &\quad + (1 - \alpha)^j \alpha\, t_{n-j} + \ldots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

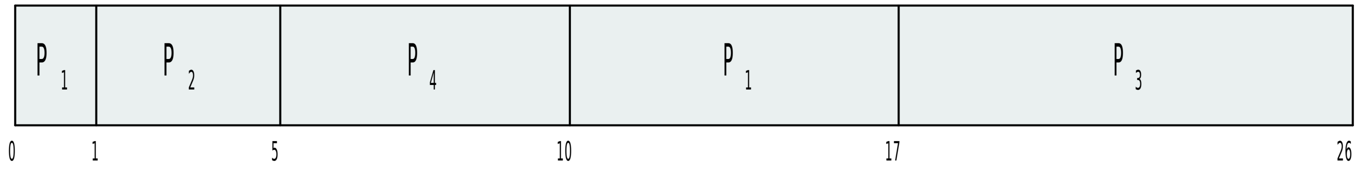
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

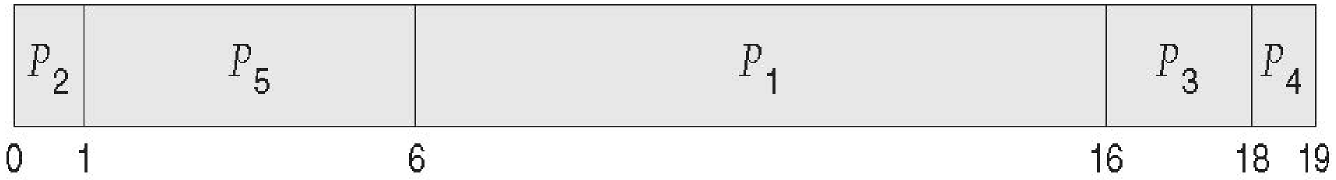
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

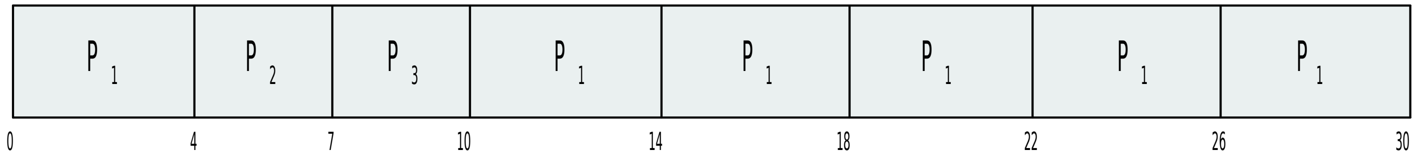
Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

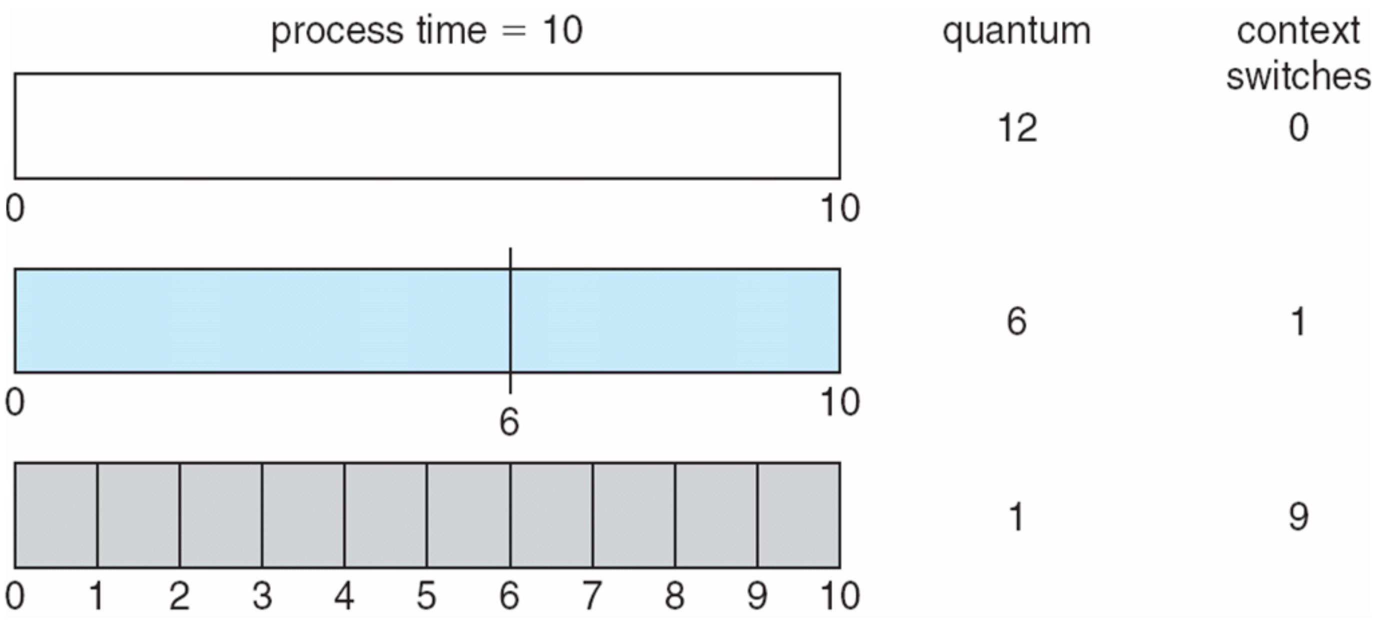
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

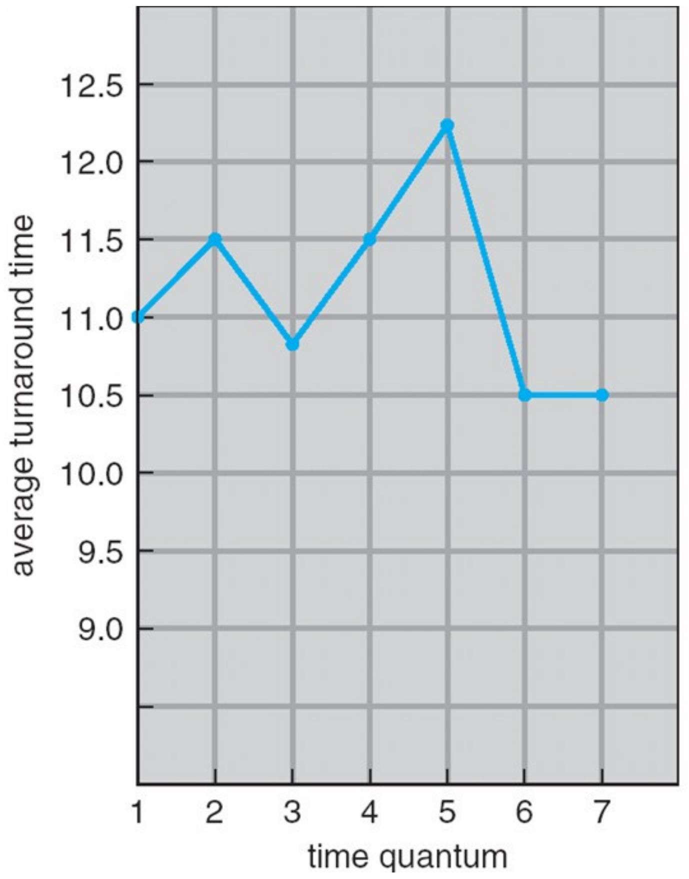


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

Unit 3
Processes

Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication
- Examples of IPC Systems
- Communication in Client-Server Systems

Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To explore interprocess communication using shared memory and message passing
- To describe communication in client-server systems

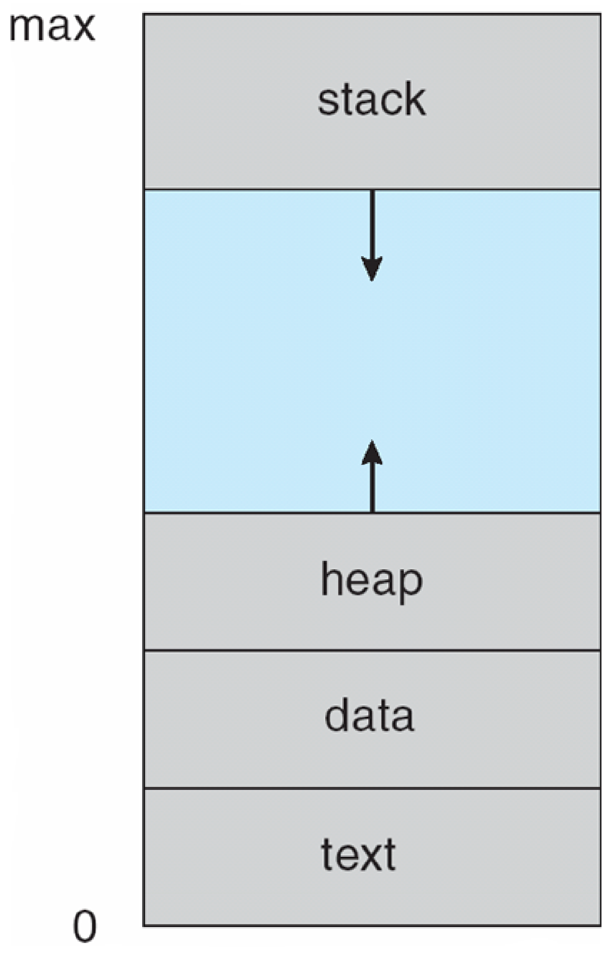
Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

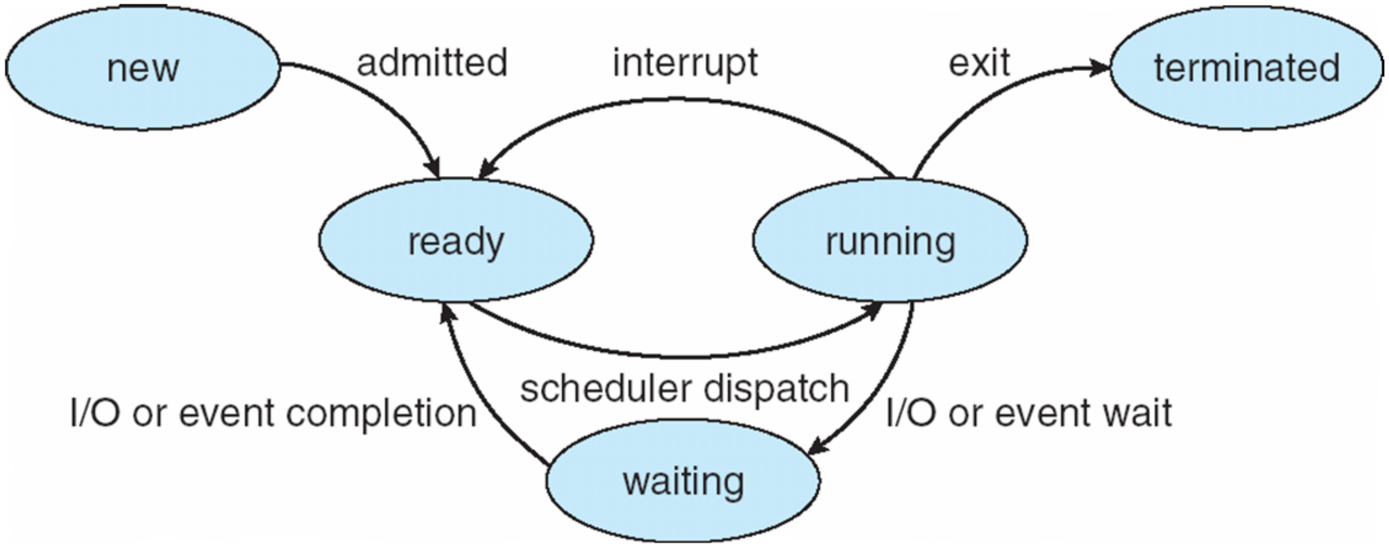
Process in Memory



Process State

- As a process executes, it changes **state**
  - **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

Diagram of Process State

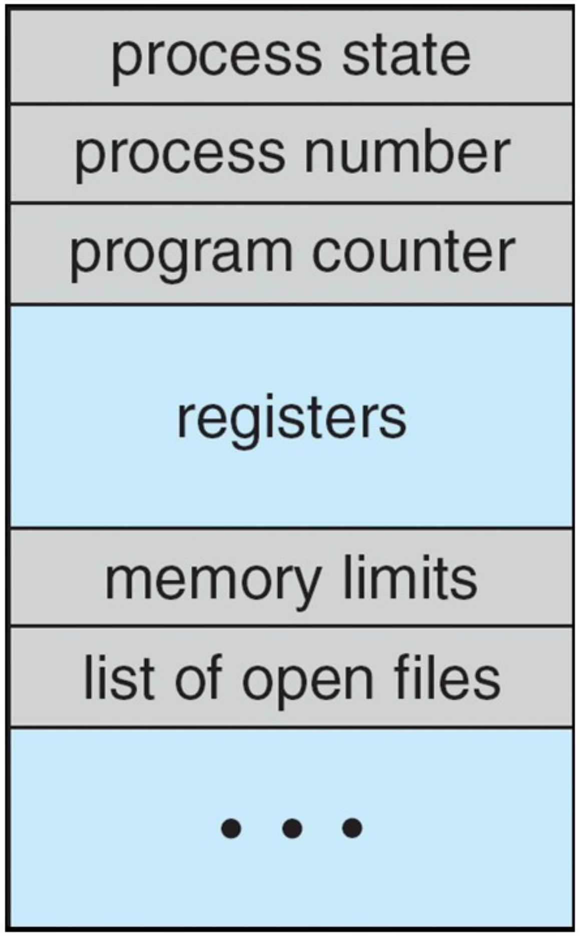


Process Control Block (PCB)

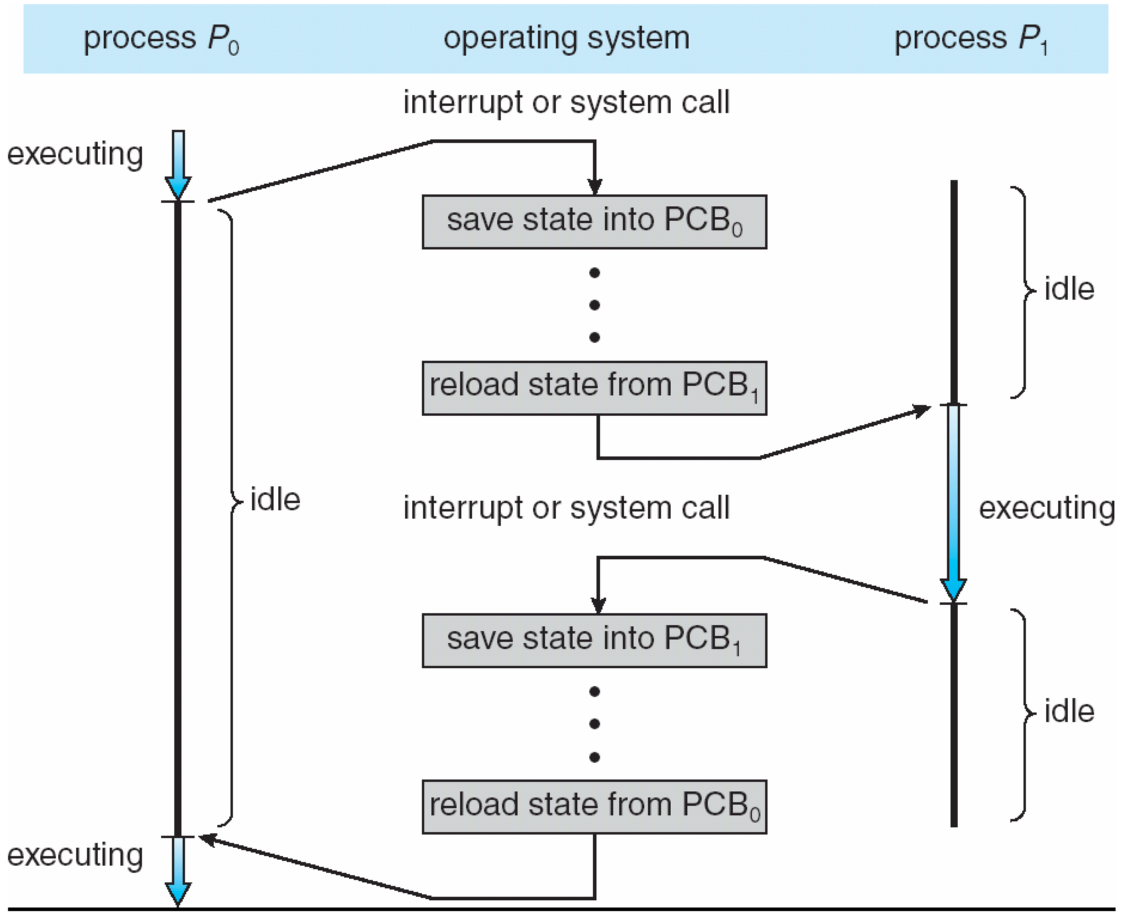
Information associated with each process

(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



CPU Switch From Process to Process



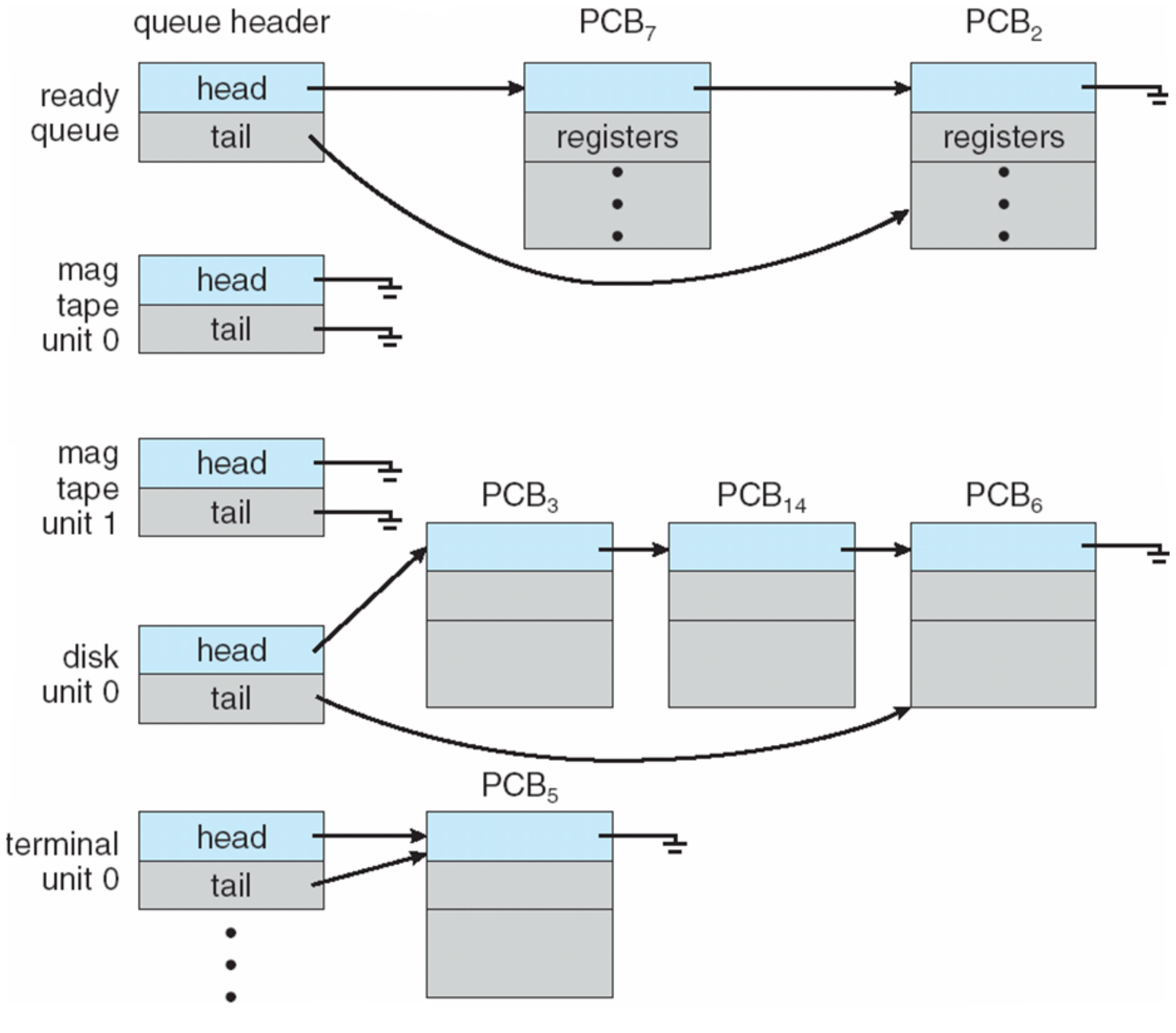
Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple program counters in PCB
- See next chapter

Process Scheduling

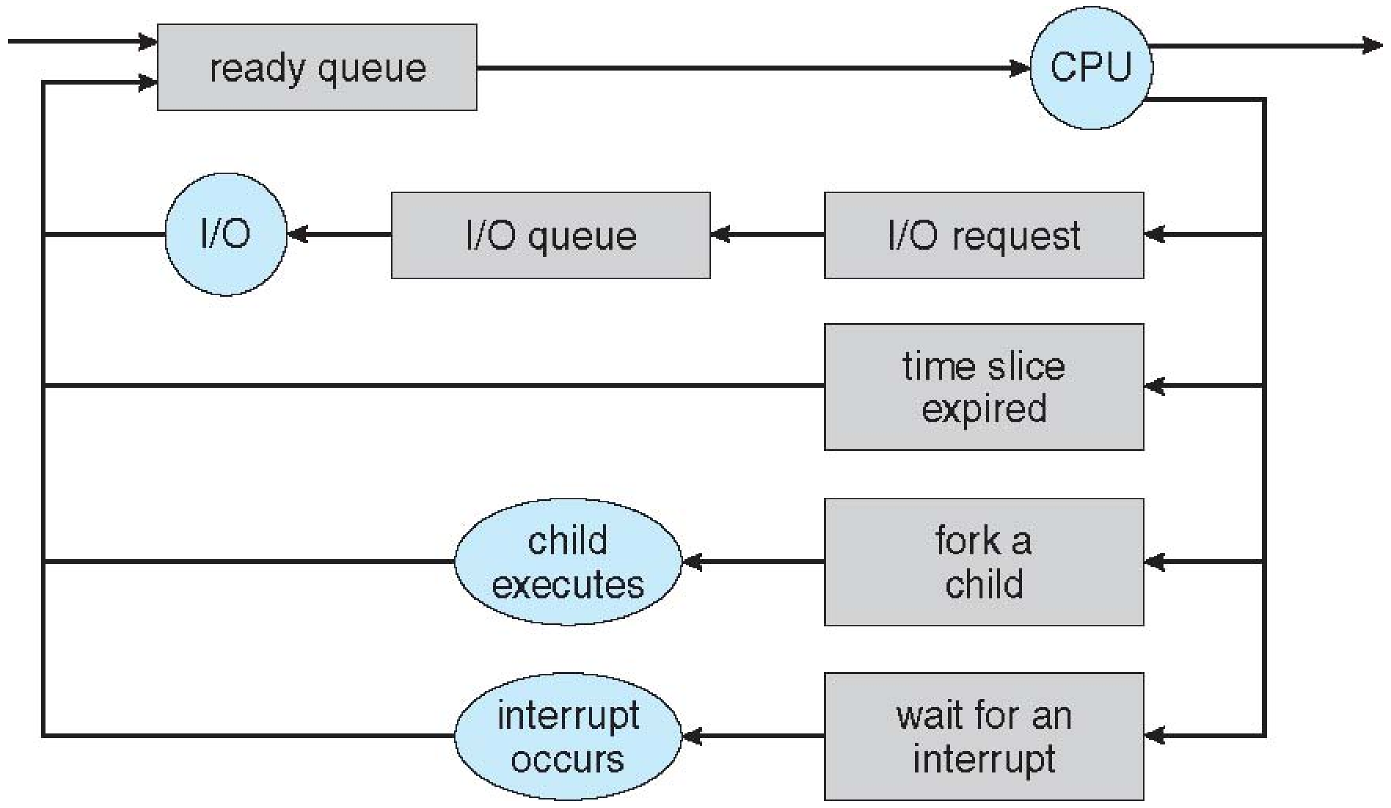
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
    - **Job queue** – set of all processes in the system
    - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
    - **Device queues** – set of processes waiting for an I/O device
    - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows

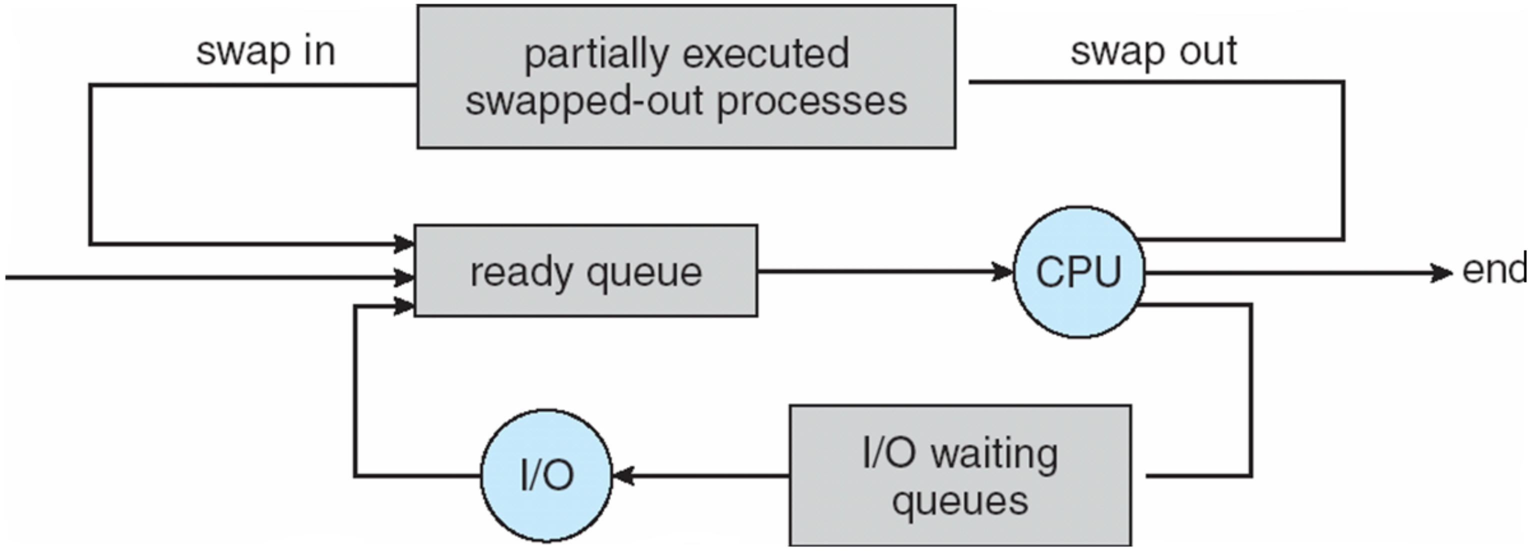


Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



Multitasking in Mobile Systems

- Some mobile systems (e.g., early version of iOS) allow only one process to run, others suspended
- Due to screen real estate, user interface limits iOS provides for a
  - Single **foreground** process- controlled via user interface
  - Multiple **background** processes– in memory, running, but not on the display, and with limits
  - Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback
- Android runs foreground and background, with fewer limits
  - Background process uses a **service** to perform tasks
  - Service can keep running even if background process is suspended
  - Service has no user interface, small memory use

Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Operations on Processes

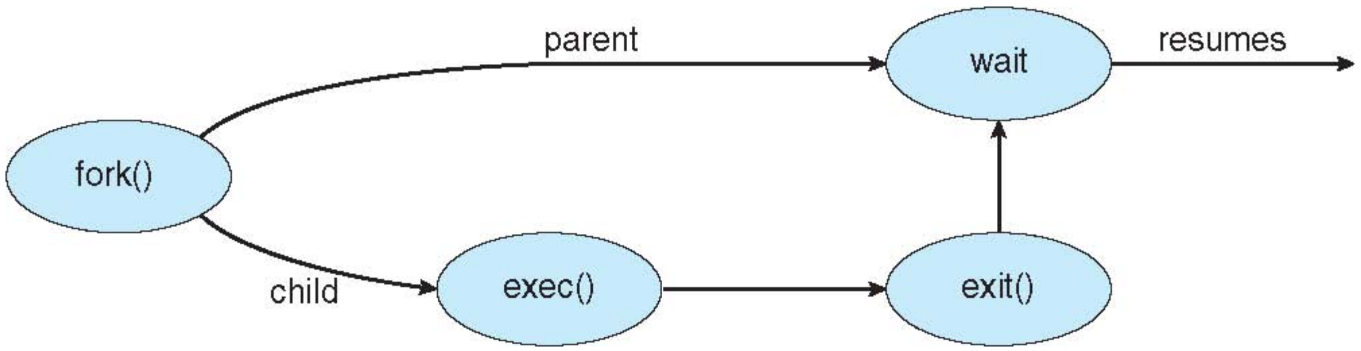
- System must provide mechanisms for:
  - process creation,
  - process termination,
  - and so on as detailed next

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **`fork()`** system call creates new process
  - **`exec()`** system call used after a **`fork()`** to replace the process' memory space with a new program



Process Termination

- Process executes last statement and then asks the operating system to delete it using the **`exit()`** system call.
  - Returns status data from child to parent (via **`wait()`**)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Process Termination

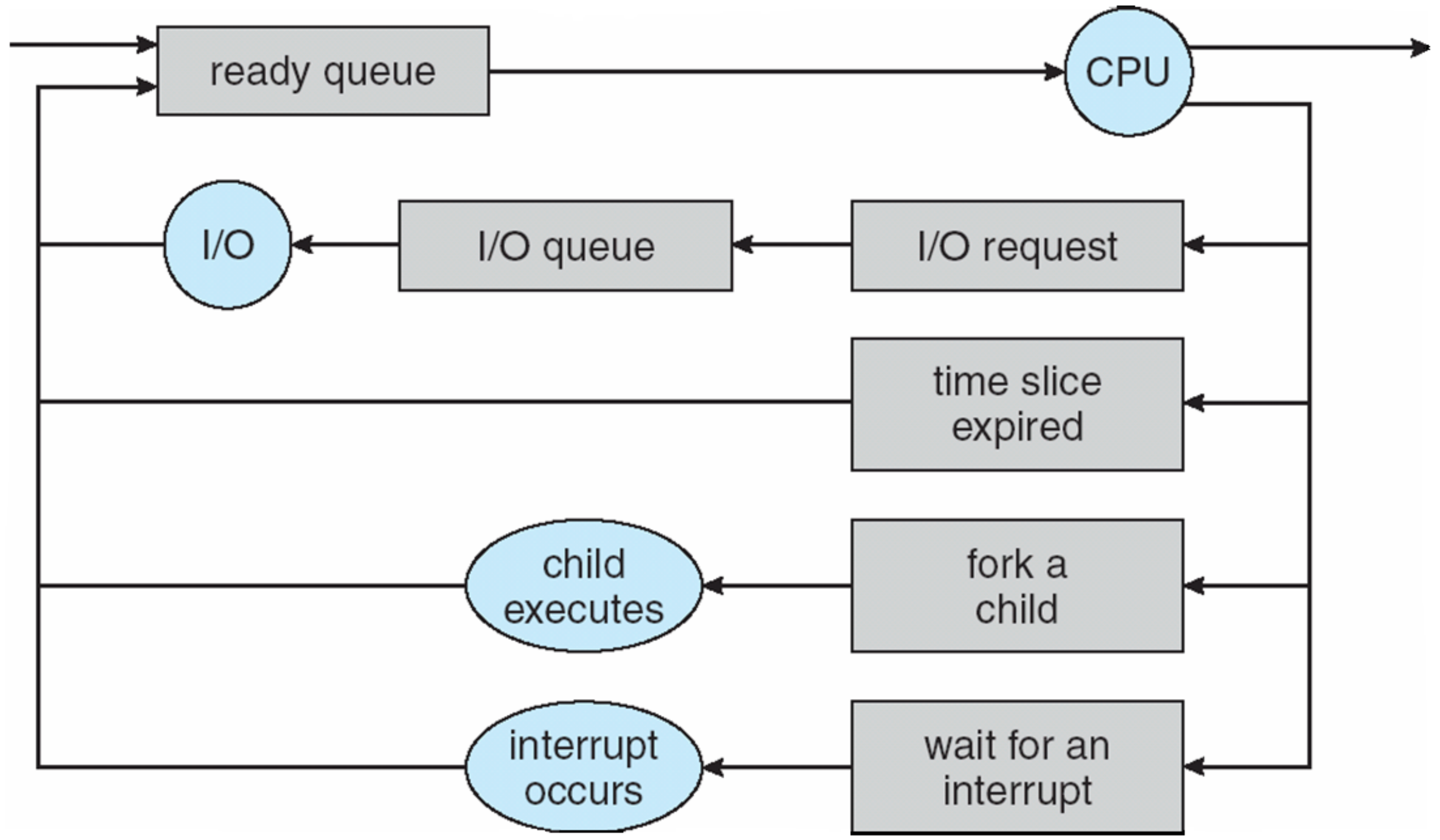
- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the **`wait()`**system call**.** The call returns status information and the pid of the terminated process

```
        pid = wait(&status);
```

- If no parent waiting (did not invoke **`wait()`**) process is a **zombie**
- If parent terminated without invoking **`wait`** , process is an **orphan**

Operating Systems

CPU Scheduling

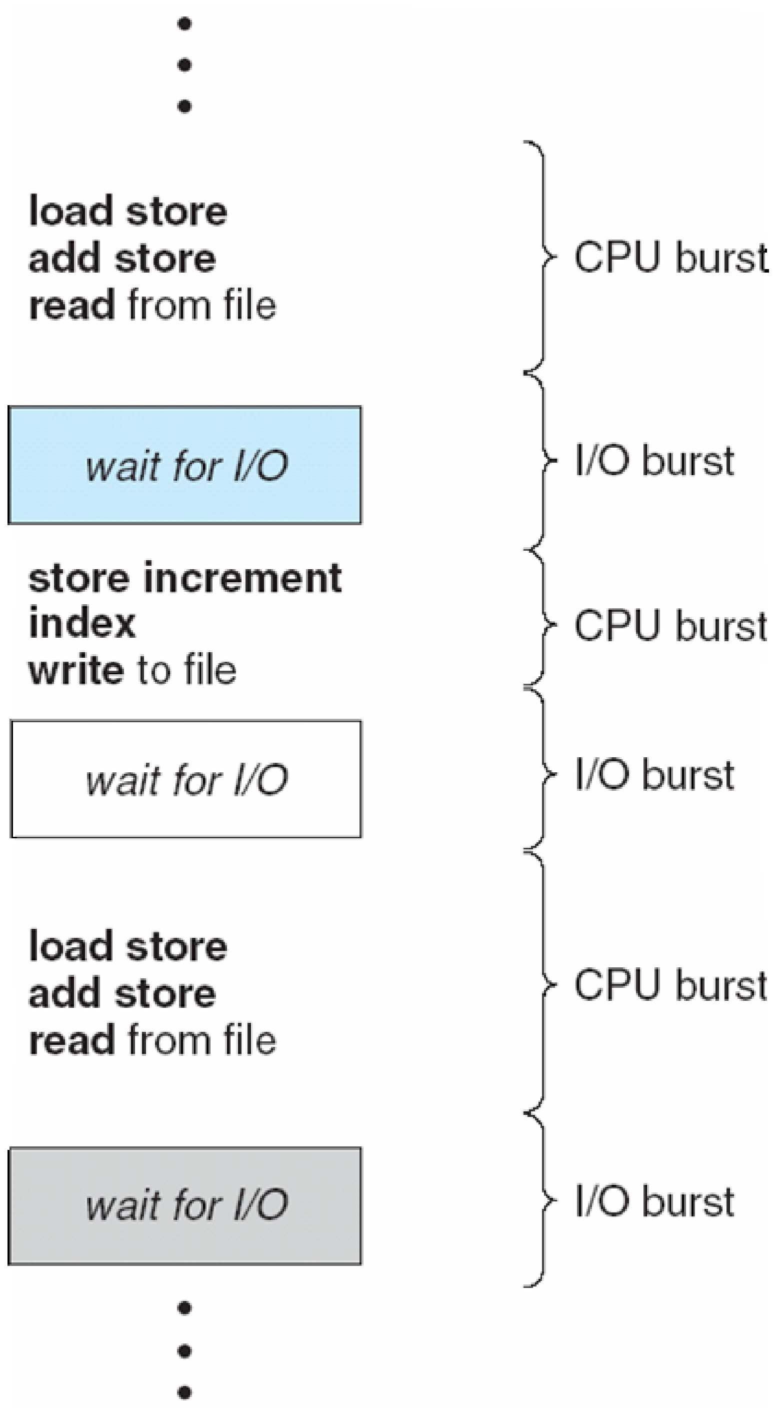


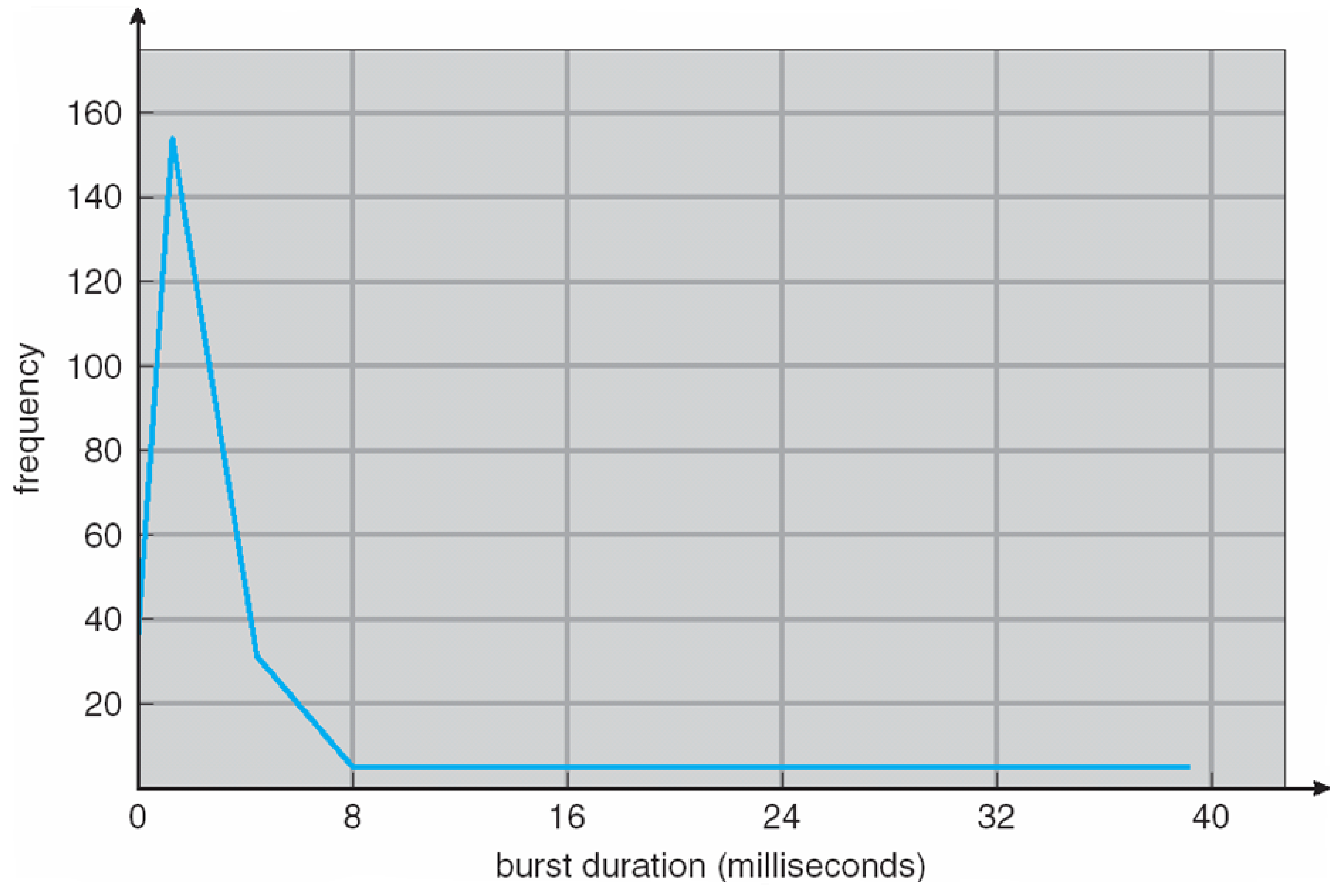
- How is the OS to decide which of several tasks to take off a queue?
- Scheduling: deciding which threads are given access to resources from moment to moment.

Assumptions about Scheduling

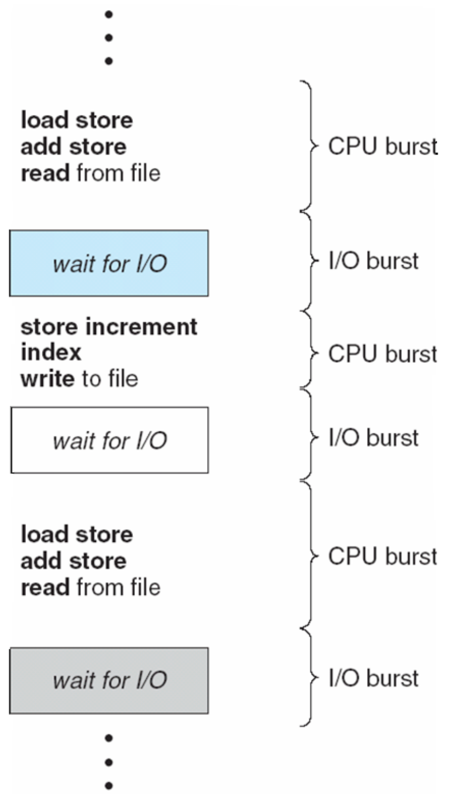
- CPU scheduling big area of research in early ‘70s
- Many implicit assumptions for CPU scheduling:
  - One program per user
  - One thread per program
  - Programs are independent
- These are unrealistic but simplify the problem
- Does “fair” mean fairness among users or programs?
  - If I run one compilation job and you run five, do you get five times as much CPU?
    - Often times, yes!
- Goal: dole out CPU time to optimize some desired parameters of the system.
  - What parameters?

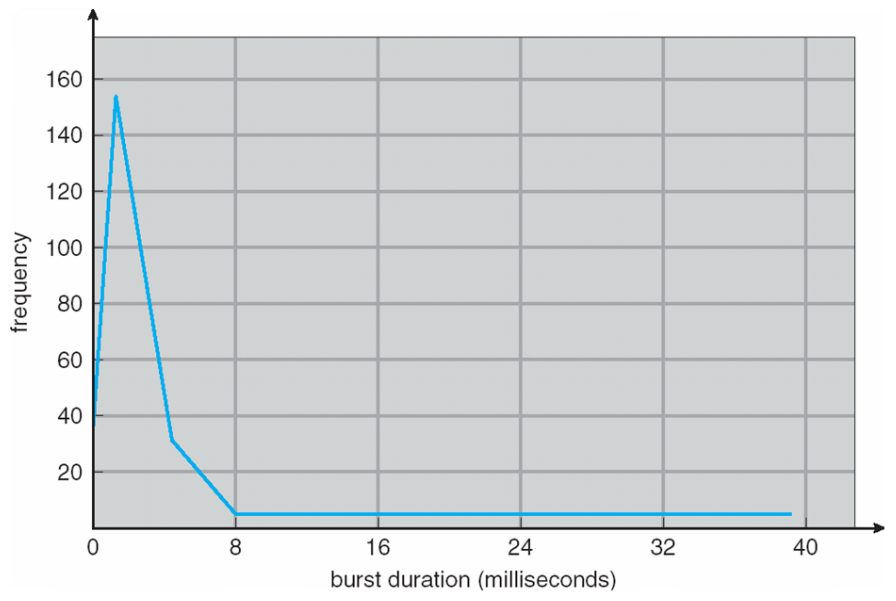
Assumption: CPU Bursts





Assumption: CPU Bursts





- Execution model: programs alternate between bursts of CPU and I/O
  - Program typically uses the CPU for some period of time, then does I/O, then uses CPU again
  - Each scheduling decision is about which job to give to the CPU for use by its next CPU burst
  - With timeslicing, thread may be forced to give up CPU before finishing current CPU burst.

What is Important in a Scheduling Algorithm?

What is Important in a Scheduling Algorithm?

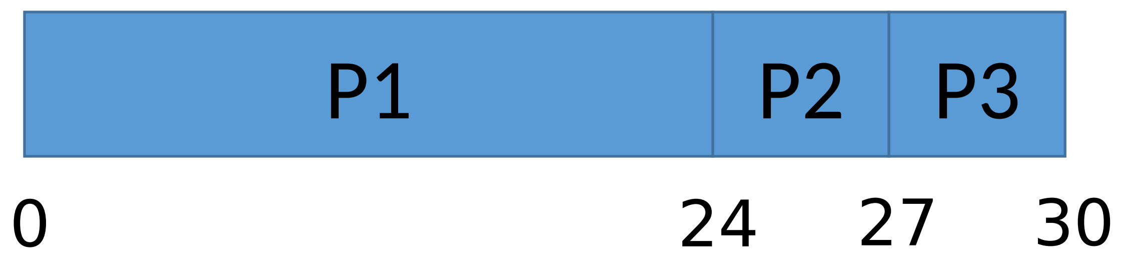
- Minimize Response Time
  - Elapsed time to do an operation (job)
  - Response time is what the user sees
    - Time to echo keystroke in editor
    - Time to compile a program
    - Real-time Tasks: Must meet deadlines imposed by World
- Maximize Throughput
  - Jobs per second
  - Throughput related to response time, but not identical
    - Minimizing response time will lead to more context switching than if you maximized only throughput
  - Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)
- Fairness
  - Share CPU among users in some equitable way
  - Not just minimizing average response time

Scheduling Algorithms: First-Come, First-Served (FCFS)

- “Run until Done:” FIFO algorithm
- In the beginning, this meant one program runs non-preemtively until it is finished (including any blocking for I/O operations)
- Now, FCFS means that a process keeps the CPU until one or more threads block
- Example: Three processes arrive in order P1, P2, P3.
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3
- Draw the Gantt Chart and compute Average Waiting Time and Average Completion Time.

Scheduling Algorithms: First-Come, First-Served (FCFS)

- Example: Three processes arrive in order P1, P2, P3.
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3
- Waiting Time
  - P1: 0
  - P2: 24
  - P3: 27
- Completion Time:
  - P1: 24
  - P2: 27
  - P3: 30
- Average Waiting Time: (0+24+27)/3 = 17
- Average Completion Time: (24+27+30)/3 = 27

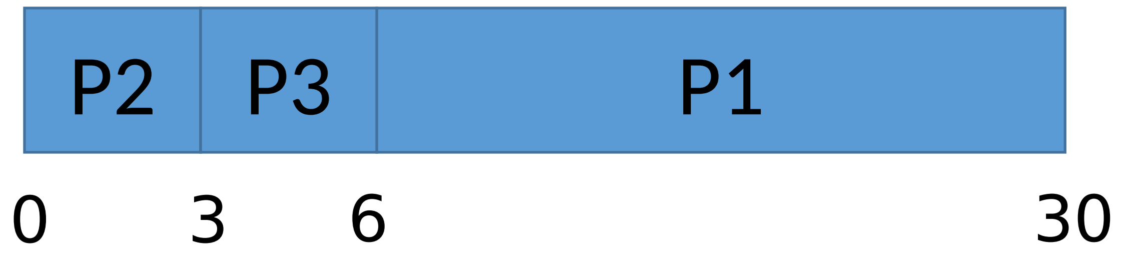


Scheduling Algorithms: First-Come, First-Served (FCFS)

- What if their order had been P2, P3, P1?
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3

Scheduling Algorithms: First-Come, First-Served (FCFS)

- What if their order had been P2, P3, P1?
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3



- Waiting Time
  - P1: 0
  - P2: 3
  - P3: 6
- Completion Time:
  - P1: 3
  - P2: 6
  - P3: 30
- Average Waiting Time: (0+3+6)/3 = 3 (compared to 17)
- Average Completion Time: (3+6+30)/3 = 13 (compared to 27)

Scheduling Algorithms: First-Come, First-Served (FCFS)

- Average Waiting Time: (0+3+6)/3 = 3 (compared to 17)
- Average Completion Time: (3+6+30)/3 = 13 (compared to 27)
- FIFO Pros and Cons:
  - Simple (+)
  - Short jobs get stuck behind long ones (-)
    - If all you’re buying is milk, doesn’t it always seem like you are stuck behind a cart full of many items
  - Performance is highly dependent on the order in which jobs arrive (-)

How Can We Improve on This?

Round Robin (RR) Scheduling

- FCFS Scheme: Potentially bad for short jobs!
  - Depends on submit order
  - If you are first in line at the supermarket with milk, you don’t care who is behind you; on the other hand…
- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets 1/N of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?

Round Robin (RR) Scheduling

- FCFS Scheme: Potentially bad for short jobs!
  - Depends on submit order
  - If you are first in line at the supermarket with milk, you don't care who is behind you; on the other hand…
- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets 1/N of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?
      - No process waits more than (n-1)q time units

Round Robin (RR) Scheduling

- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets 1/N of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?
      - No process waits more than (n-1)q time units
- Performance Depends on Size of Q
  - Small Q => interleaved
  - Large Q is like…
  - Q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

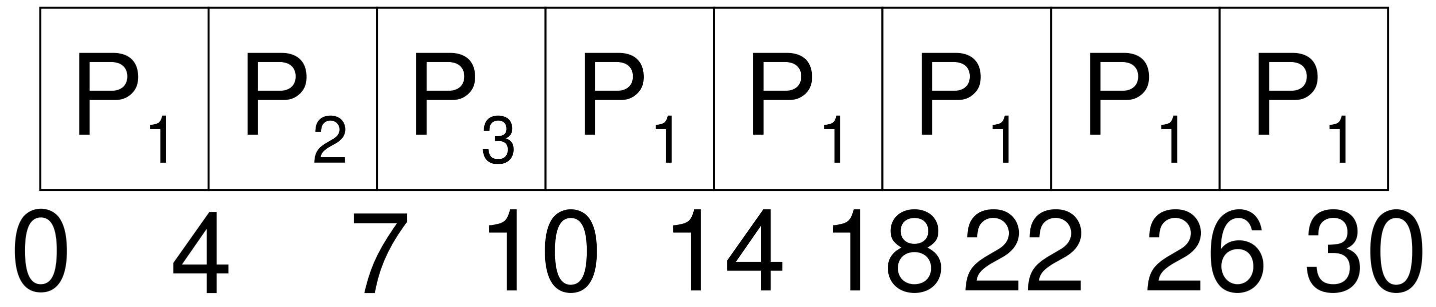
Round Robin (RR) Scheduling

- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets 1/N of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?
      - No process waits more than (n-1)q time units
- Performance Depends on Size of Q
  - Small Q => interleaved
  - Large Q is like FCFS
  - Q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

Example of RR with Time Quantum = 4

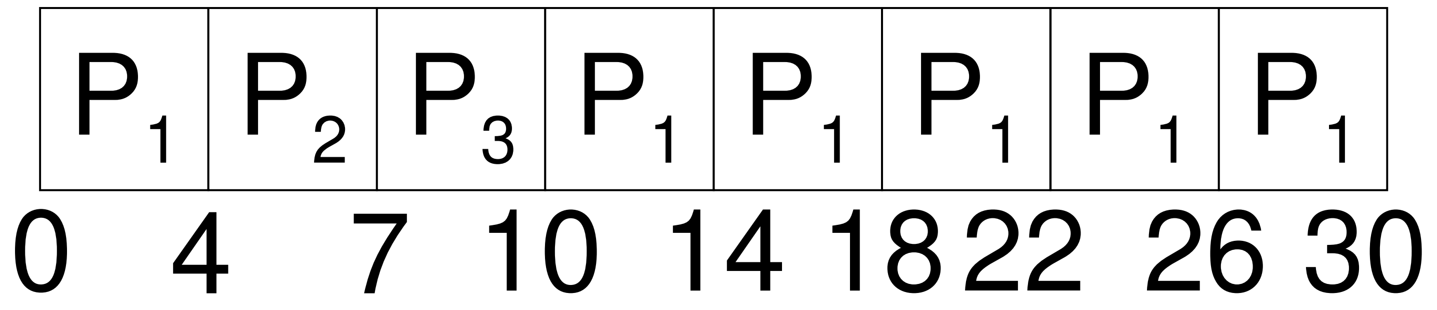
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



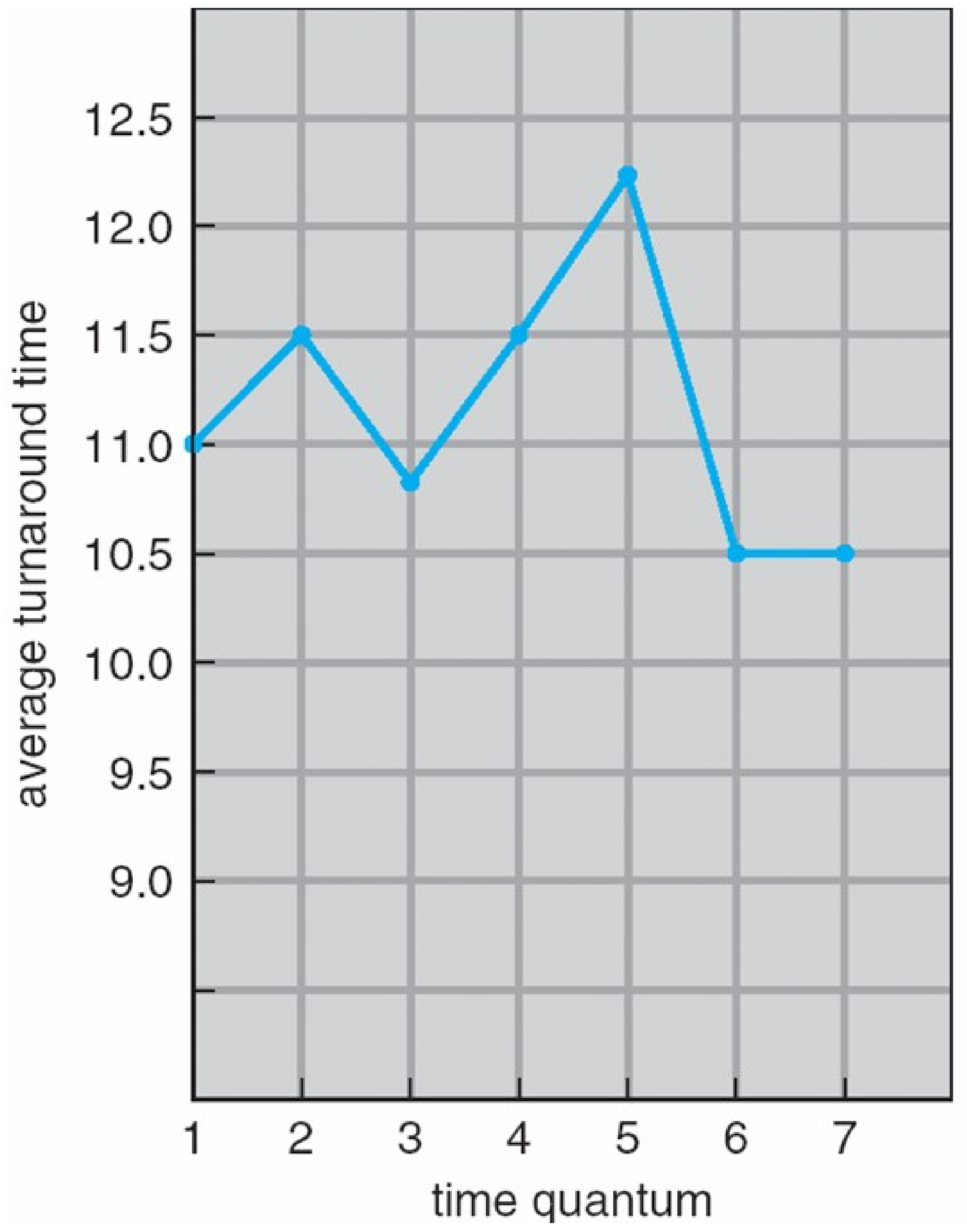
Example of RR with Time Quantum = 4

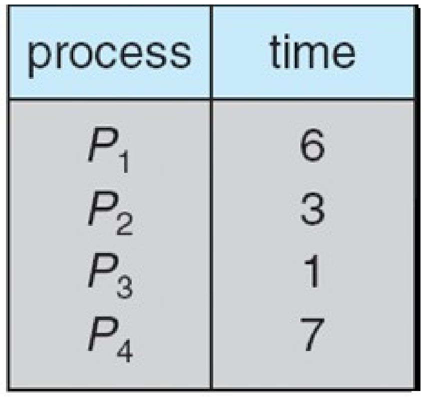
Process	Burst Time
P_1	24
P_2	3
P_3	3



- Waiting Time:
  - P1: (10-4) = 6
  - P2: (4-0) = 4
  - P3: (7-0) = 7
- Completion Time:
  - P1: 30
  - P2: 7
  - P3: 10
- Average Waiting Time: (6 + 4 + 7)/3= 5.67
- Average Completion Time: (30+7+10)/3=15.67

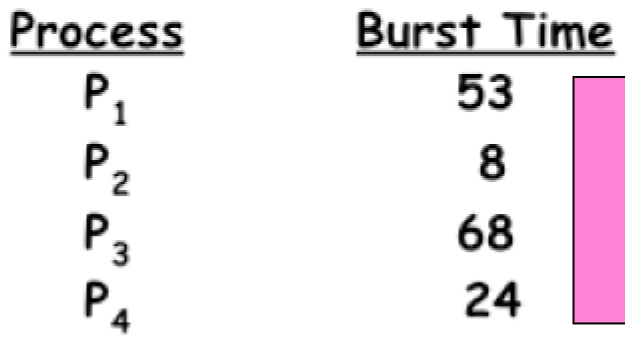
Turnaround Time Varies With The Time Quantum



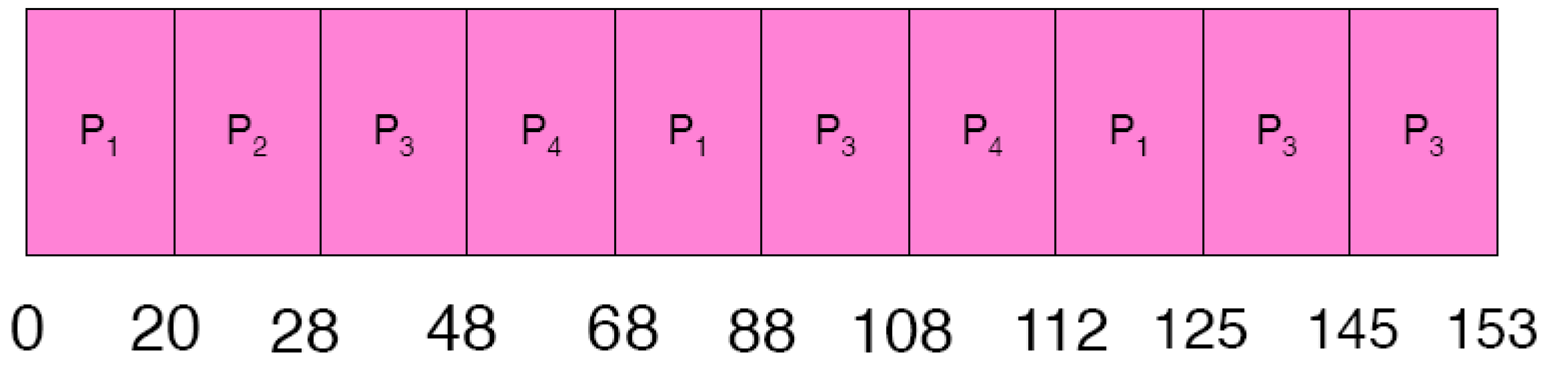


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Example of RR with Time Quantum = 20



Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

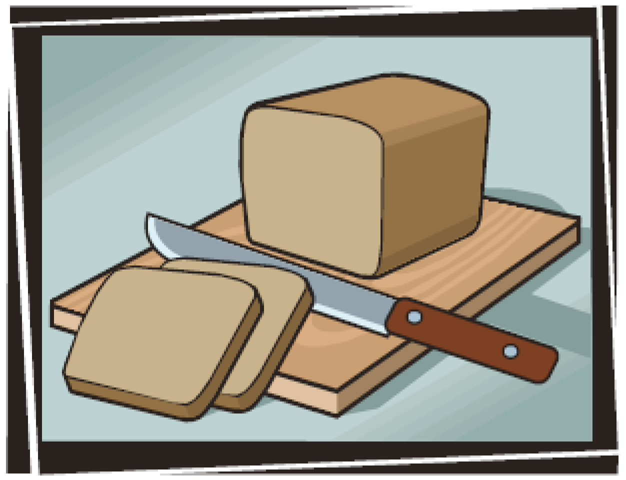


A process can finish before the time quantum expires, and release the CPU.

- Waiting Time:
  - P1: (68-20)+(112-88) = 72
  - P2: (20-0) = 20
  - P3: (28-0)+(88-48)+(125-108) = 85
  - P4: (48-0)+(108-68) = 88
- Completion Time:
  - P1: 125
  - P2: 28
  - P3: 153
  - P4: 112
- Average Waiting Time: (72+20+85+88)/4 = 66.25
- Average Completion Time: (125+28+153+112)/4 = 104.5

RR Summary

- Pros and Cons:
  - Better for short jobs (+)
  - Fair (+)
  - Context-switching time adds up for long jobs (-)
    - The previous examples assumed no additional time was needed for context switching – in reality, this would add to wait and completion time without actually progressing a process towards completion.
    - Remember: the OS consumes resources, too!
- If the chosen quantum is
  - too large, response time suffers
  - infinite, performance is the same as FIFO
  - too small, throughput suffers and percentage overhead grows
- Actual choices of timeslice:
  - UNIX: initially 1 second:
    - Worked when only 1-2 users
    - If there were 3 compilations going on, it took 3 seconds to echo each keystroke!
  - In practice, need to balance short-job performance and long-job throughput:
    - Typical timeslice 10ms-100ms
    - Typical context-switch overhead 0.1ms – 1ms (about 1%)

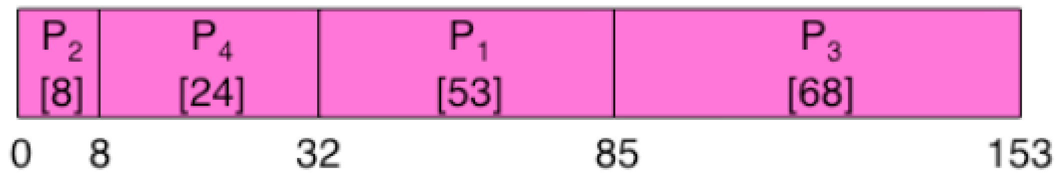


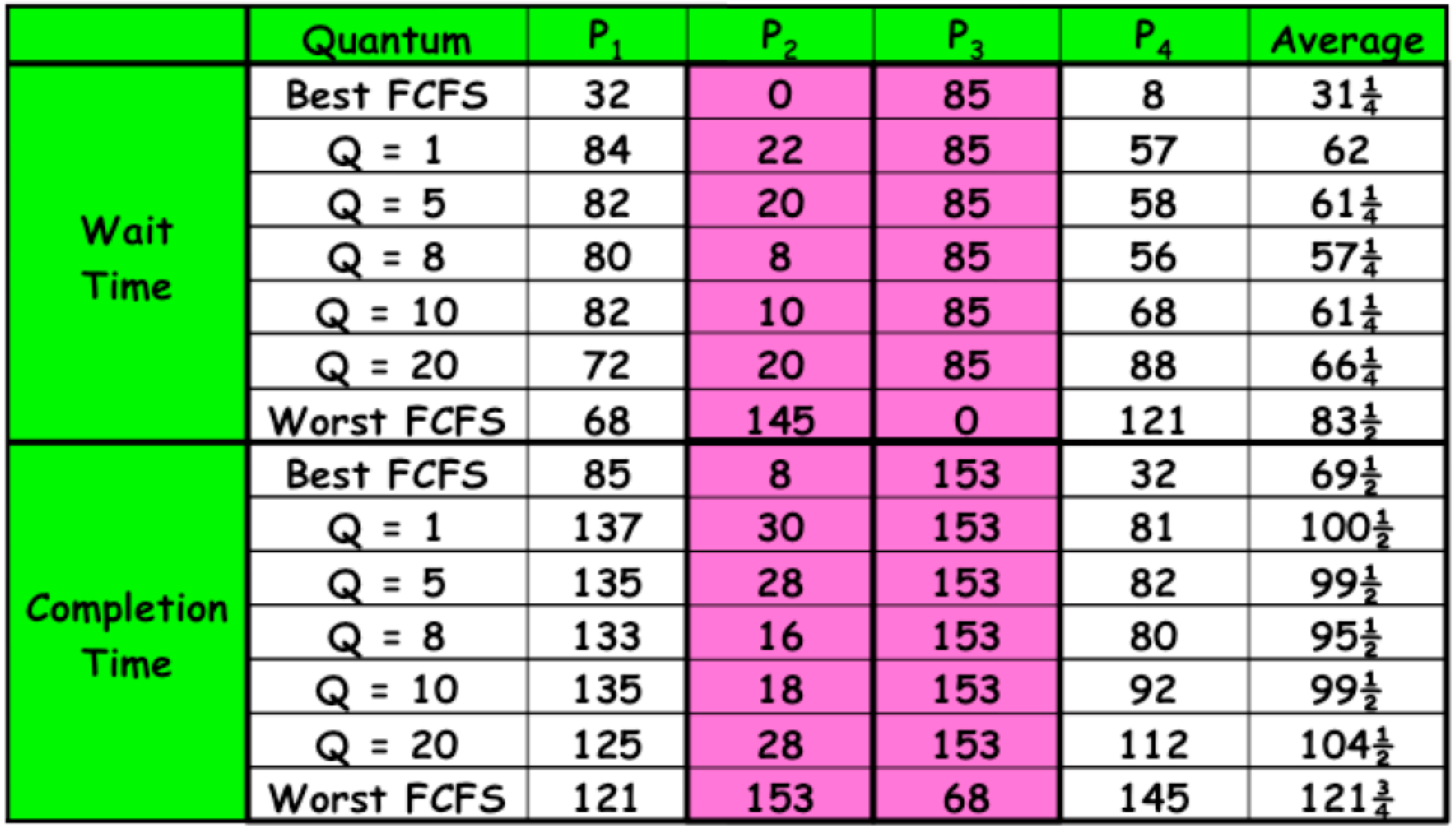
Comparing FCFS and RR

- Assuming zero-cost context switching time, is RR always better than FCFS?
- Assume 10 jobs, all start at the same time, and each require 100 seconds of CPU time
- RR scheduler quantum of 1 second
- Completion Times (CT)
  - Both FCFS and RR finish at the same time
  - But average response time is much worse under RR!
    - Bad when all jobs are same length
- Also: cache state must be shared between all jobs with RR but can be devoted to each job with FIFO
  - Total time for RR longer even for zero-cost context switch!

Job #	FCFS CT	RR CT
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

Comparing FCFS and RR





	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	$31\frac{1}{4}$
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	$61\frac{1}{4}$
	Q = 8	80	8	85	56	$57\frac{1}{4}$
	Q = 10	82	10	85	68	$61\frac{1}{4}$
	Q = 20	72	20	85	88	$66\frac{1}{4}$
	Worst FCFS	68	145	0	121	$83\frac{1}{2}$
Completion Time	Best FCFS	85	8	153	32	$69\frac{1}{2}$
	Q = 1	137	30	153	81	$100\frac{1}{2}$
	Q = 5	135	28	153	82	$99\frac{1}{2}$
	Q = 8	133	16	153	80	$95\frac{1}{2}$
	Q = 10	135	18	153	92	$99\frac{1}{2}$
	Q = 20	125	28	153	112	$104\frac{1}{2}$
	Worst FCFS	121	153	68	145	$121\frac{3}{4}$

Scheduling

- The performance we get is somewhat dependent on what “kind” of jobs we are running (short jobs, long jobs, etc.)
- If we could “see the future,” we could mirror best FCFS
- Shortest Job First (SJF) a.k.a. Shortest Time to Completion First (STCF):
  - Run whatever job has the least amount of computation to do
- Shortest Remaining Time First (SRTF) a.k.a. Shortest Remaining Time to Completion First (SRTCF):
  - Preemptive version of SJF: if a job arrives and has a shorter time to completion than the remaining time on the current job, immediately preempt CPU
- These can be applied either to a whole program or the current CPU burst of each program
  - Idea: get short jobs out of the system
  - Big effect on short jobs, only small effect on long ones
  - Result: better average response time

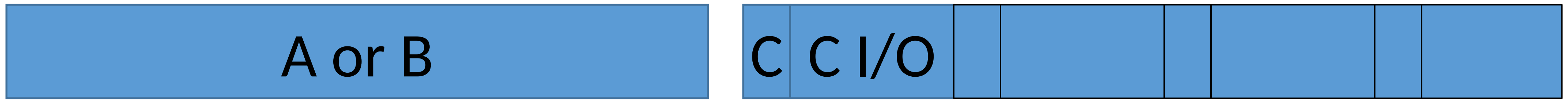
Scheduling

- But, this is hard to estimate
- We could get feedback from the program or the user, but they have incentive to lie!
- SJF/SRTF are the best you can do at minimizing average response time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - Since SRTF is always at least as good as SJF, focus on SRTF
- Comparison of SRTF with FCFS and RR
  - What if all jobs are the same length?
  - What if all jobs have varying length?

Scheduling

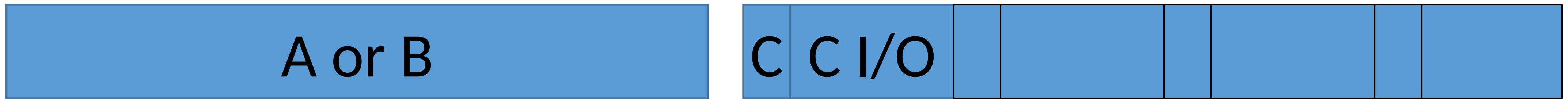
- But, this is hard to estimate
- We could get feedback from the program or the user, but they have incentive to lie!
- SJF/SRTF are the best you can do at minimizing average response time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - Since SRTF is always at least as good as SJF, focus on SRTF
- Comparison of SRTF with FCFS and RR
  - What if all jobs are the same length?
    - SRTF becomes the same as FCFS (i.e. FCFS is the best we can do)
  - What if all jobs have varying length?
    - SRTF (and RR): short jobs are not stuck behind long ones

Example: SRTF

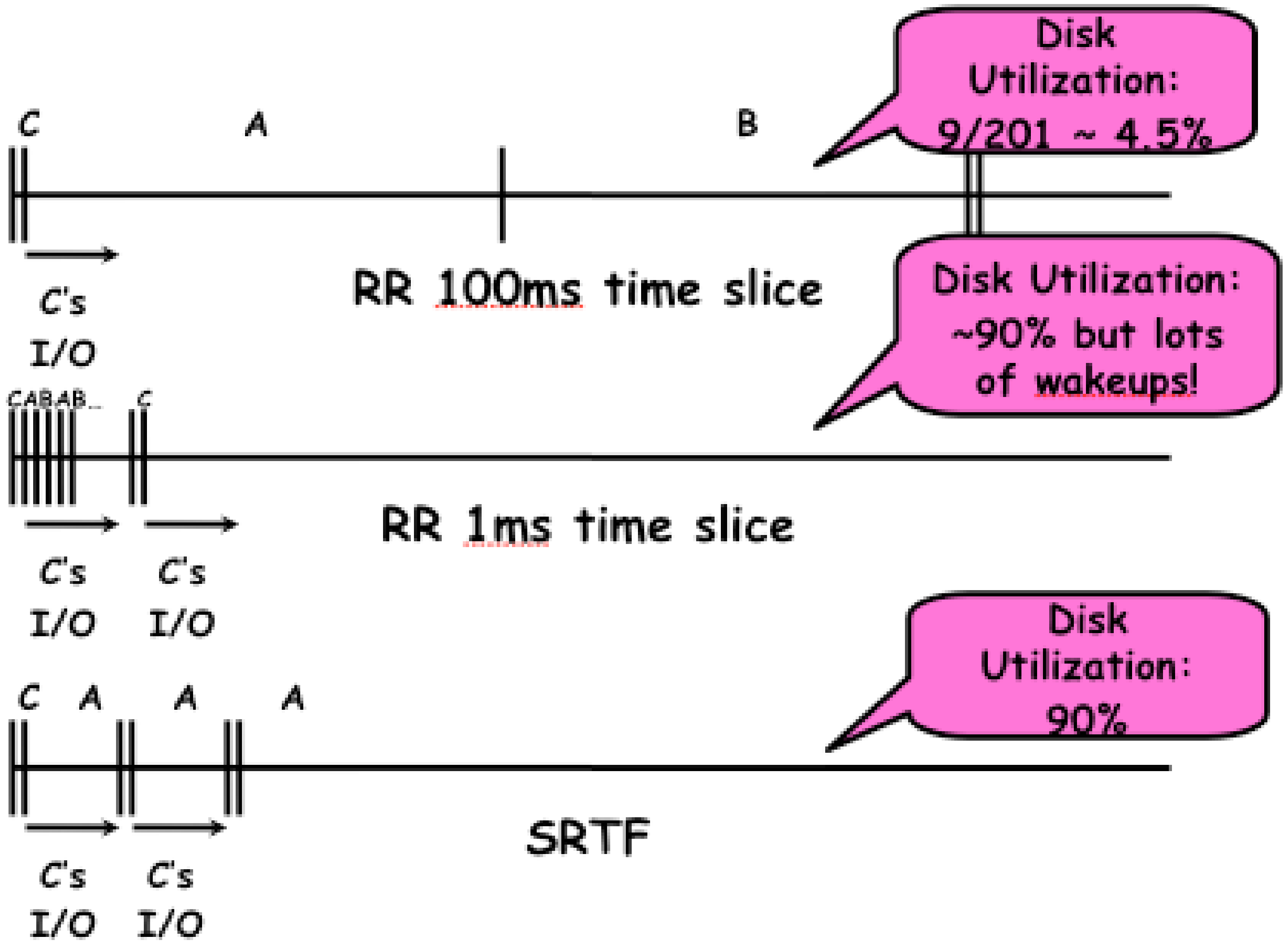


- A,B: both CPU bound, run for a week
- C: I/O bound, loop 1ms CPU, 9ms disk I/O
- If only one at a time, C uses 90% of the disk, A or B could use 100% of the CPU
- With FIFO: Once A and B get in, the CPU is held for two weeks
- What about RR or SRTF?
  - Easier to see with a timeline

Example: SRTF



- A,B: both CPU bound, run for a week
- C: I/O bound, loop 1ms CPU, 9ms disk I/O



Last Word on SRTF

- Starvation
  - SRTF can lead to starvation if many small jobs!
  - Large jobs never get to run
- Somehow need to predict future
  - How can we do this?
  - Some systems ask the user
    - When you submit a job, you have to say how long it will take
    - To stop cheating, system kills job if it takes too long
  - But even non-malicious users have trouble predicting runtime of their jobs
- Bottom line, can’t really tell how long job will take
  - However, can use SRTF as a yardstick for measuring other policies, since it is optimal
- SRTF Pros and Cons
  - Optimal (average response time) (+)
  - Hard to predict future (-)
  - Unfair, even though we minimized average response time! (-)

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive (if a higher priority process enters, it receives the CPU immediately)
  - Nonpreemptive (higher priority processes must wait until the current process finishes; then, the highest priority ready process is selected)
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Scheduling Details

- Result approximates SRTF
  - CPU bound jobs drop rapidly to lower queues
  - Short-running I/O bound jobs stay near the top
- Scheduling must be done between the queues
  - Fixed priority scheduling: serve all from the highest priority, then the next priority, etc.
  - Time slice: each queue gets a certain amount of CPU time (e.g., 70% to the highest, 20% next, 10% lowest)
- Countermeasure: user action that can foil intent of the OS designer
  - For multilevel feedback, put in a bunch of meaningless I/O to keep job’s priority high
  - But if everyone does this, it won’t work!
  - Consider an Othello program, playing against a competitor. Key was to compute at a higher priority than the competitors.
    - Put in printf’s, run much faster!

Scheduling Details

- It is apparent that scheduling is facilitated by having a “good mix” of I/O bound and CPU bound programs, so that there are long and short CPU bursts to prioritize around.
- There is typically a long-term and a short-term scheduler in the OS.
- We have been discussing the design of the short-term scheduler.
- The long-term scheduler decides what processes should be put into the ready queue in the first place for the short-term scheduler, so that the short-term scheduler can make fast decisions on a good mix of a subset of ready processes.
- The rest are held in memory or disk
  - Why else is this helpful?

Scheduling Details

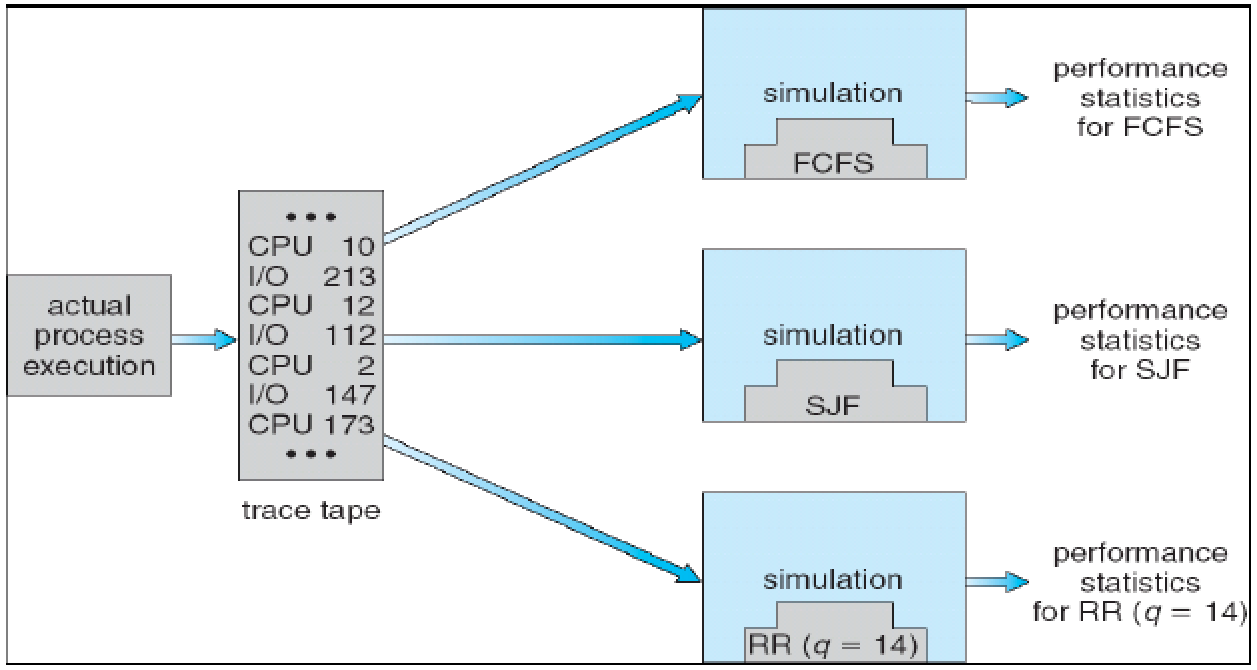
- It is apparent that scheduling is facilitated by having a “good mix” of I/O bound and CPU bound programs, so that there are long and short CPU bursts to prioritize around.
- There is typically a long-term and a short-term scheduler in the OS.
- We have been discussing the design of the short-term scheduler.
- The long-term scheduler decides what processes should be put into the ready queue in the first place for the short-term scheduler, so that the short-term scheduler can make fast decisions on a good mix of a subset of ready processes.
- The rest are held in memory or disk
    - This also provides more free memory for the subset of ready processes given to the short-term scheduler.

Fairness

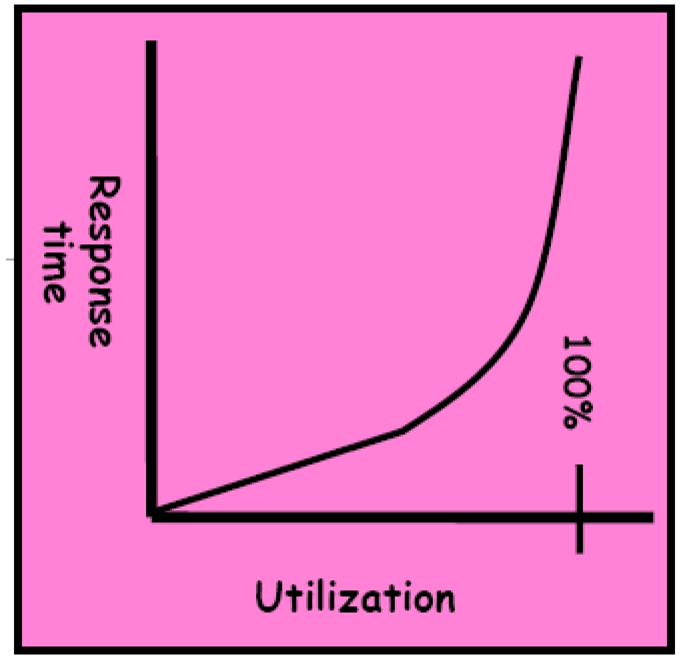
- What about fairness?
  - Strict fixed-policy scheduling between queues is unfair (run highest, then next, etc.)
    - Long running jobs may never get the CPU
    - In Multics, admins shut down the machine and found a 10-year-old job
  - Must give long-running jobs a fraction of the CPU even when there are shorter jobs to run
    - Tradeoff: fairness gained by hurting average response time!
- How to implement fairness?
  - Could give each queue some fraction of the CPU
    - i.e., for one long-running job and 100 short-running ones?
    - Like express lanes in a supermarket – sometimes express lanes get so long, one gets better service by going into one of the regular lines
  - Could increase priority of jobs that don’t get service (as seen in the multilevel feedback example)
    - This was done in UNIX
    - Ad hoc – with what rate should priorities be increased?
    - As system gets overloaded, no job gets CPU time, so everyone increases in priority
      - Interactive processes suffer

Scheduling Algorithm Evaluation

- Deterministic Modeling
  - Takes a predetermined workload and compute the performance of each algorithm for that workload
- Queuing Models
  - Mathematical Approach for handling stochastic workloads
- Implementation / Simulation
  - Build system which allows actual algorithms to be run against actual data. Most flexible / general.



Conclusion



- Scheduling: selecting a waiting process from the ready queue and allocating the CPU to it
- When do the details of the scheduling policy and fairness really matter?
  - When there aren't enough resources to go around
- When should you simply buy a faster computer?
  - Or network link, expanded highway, etc.
  - One approach: buy it when it will pay for itself in improved response time
    - Assuming you're paying for worse response in reduced productivity, customer angst, etc.
    - Might think that you should buy a faster X when X is utilized 100%, but usually, response time goes to infinite as utilization goes to 100%
  - Most scheduling algorithms work fine in the "linear" portion of the load curve, and fail otherwise
  - Argues for buying a faster X when utilization is at the "knee" of the curve

- FCFS scheduling, FIFO Run Until Done:
  - Simple, but short jobs get stuck behind long ones
- RR scheduling:
  - Give each thread a small amount of CPU time when it executes, and cycle between all ready threads
  - Better for short jobs, but poor when jobs are the same length
- SJF/SRTF:
  - Run whatever job has the least amount of computation to do / least amount of remaining computation to do
  - Optimal (average response time), but unfair; hard to predict the future
- Priority Scheduling:
  - Preemptive or Non-preemptive
  - Priority Inversion

Unit 3
Threads

Threads

- Overview
- Multicore Programming
- Multithreading Models
- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples

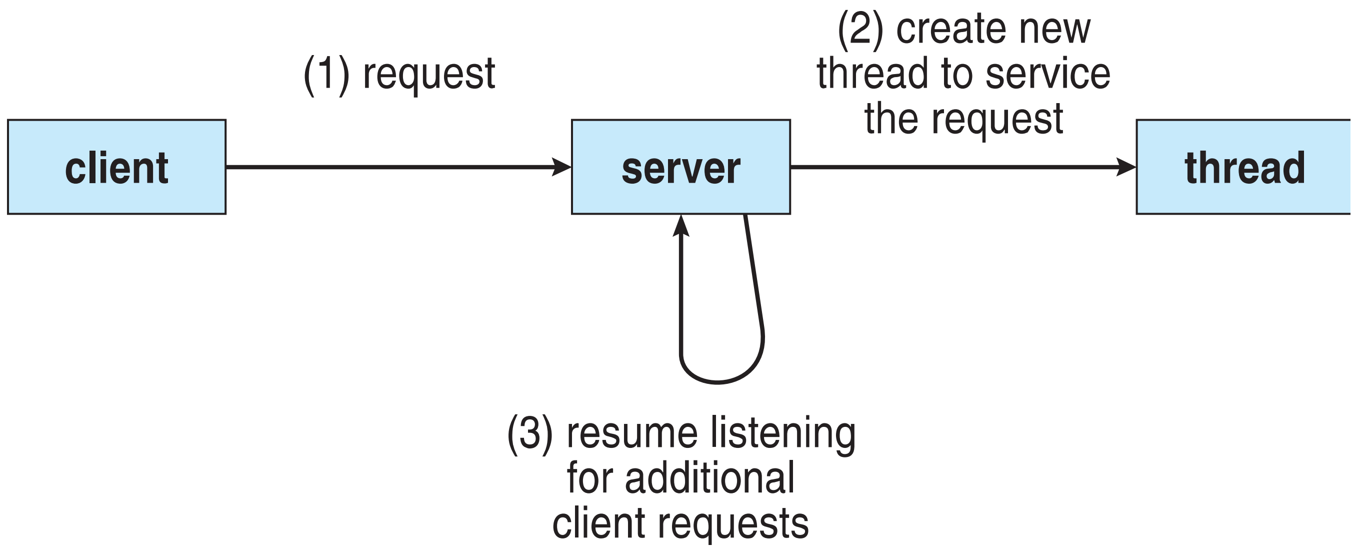
Objectives

- To introduce the notion of a thread—a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To discuss the APIs for the Pthreads, Windows, and Java thread libraries
- To explore several strategies that provide implicit threading
- To examine issues related to multithreaded programming
- To cover operating system support for threads in Windows and Linux

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Multithreaded Server Architecture


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectures

Multicore Programming

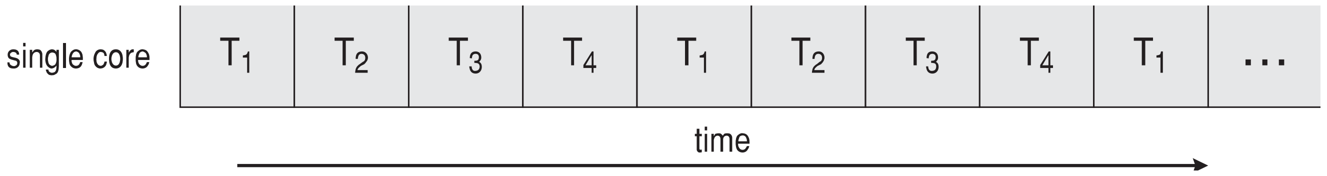
- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multicore Programming (Cont.)

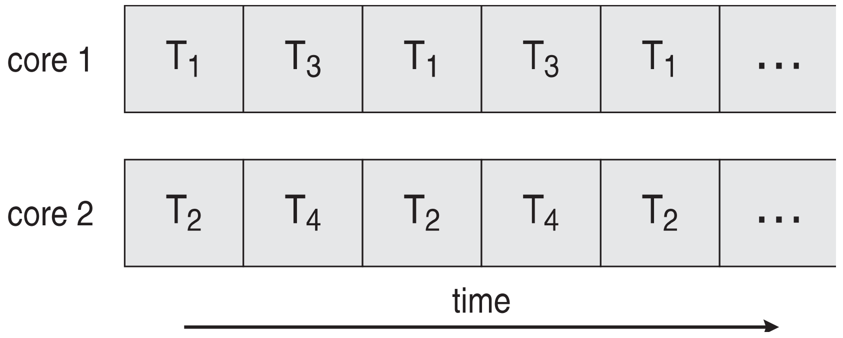
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

Concurrency vs. Parallelism

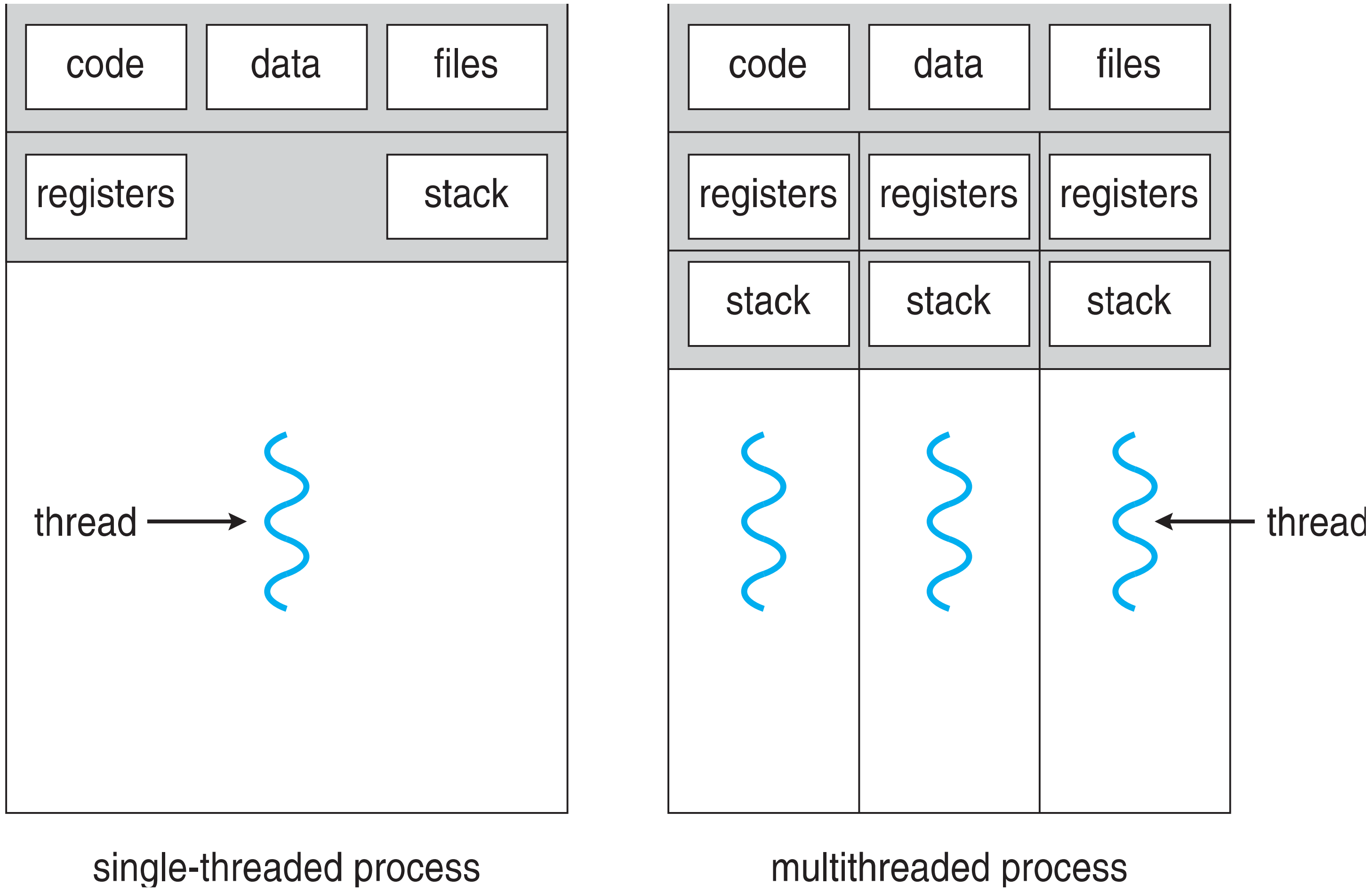
- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Single and Multithreaded Processes



Amdahl’s Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- *S* is serial portion
- *N* processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As *N* approaches infinity, speedup approaches 1 / *S*

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

- But does the law take into account contemporary multicore systems?

User Threads and Kernel Threads

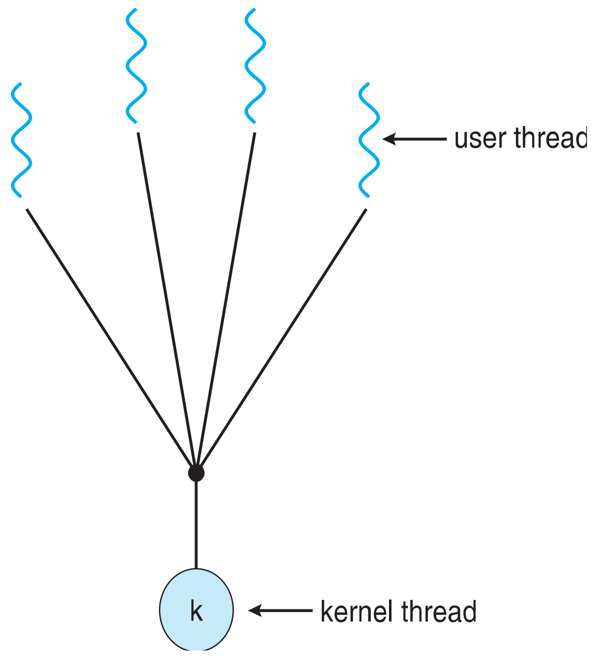
- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

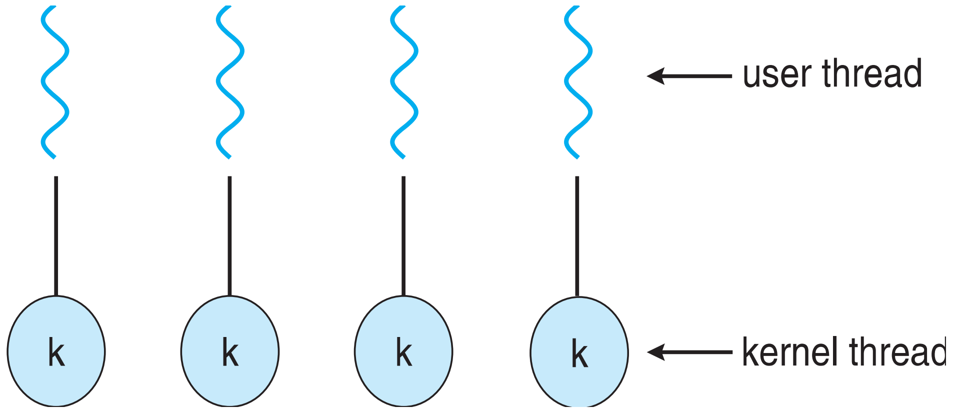
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



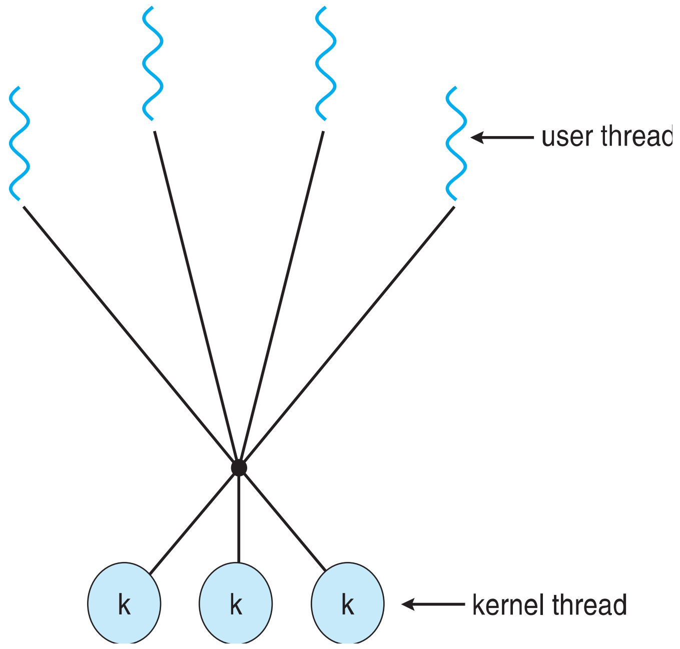
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



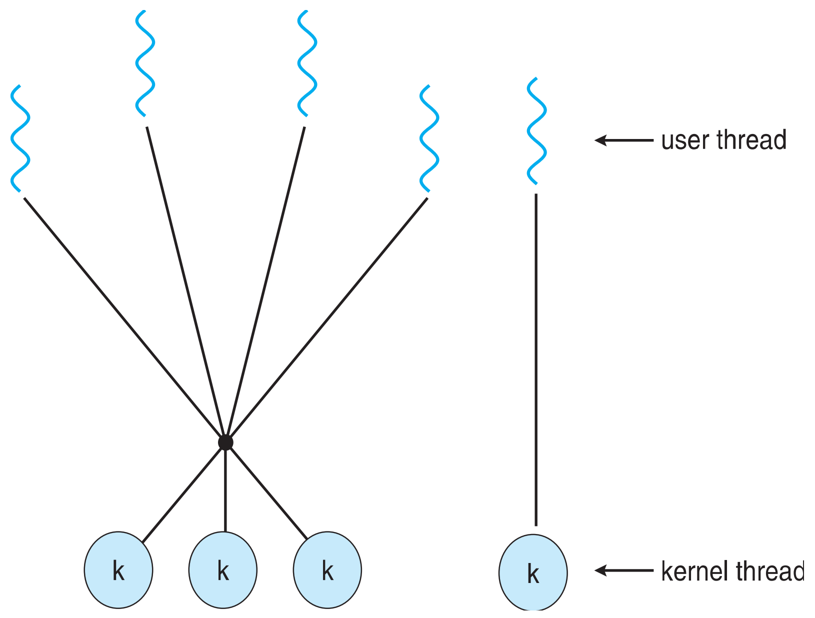
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

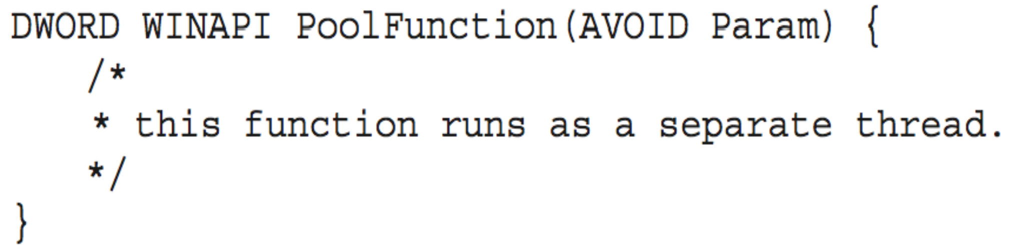
- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Implicit Threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Three methods explored
  - Thread Pools
  - OpenMP
  - Grand Central Dispatch
- Other methods include Microsoft Threading Building Blocks (TBB), **`java.util.concurrent`** package

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
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - i.e.Tasks could be scheduled to run periodically
- Windows API supports thread pools:



```
DWORD WINAPI PoolFunction(AVOID Param) {
    /*
     * this function runs as a separate thread.
     */
}
```

Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations