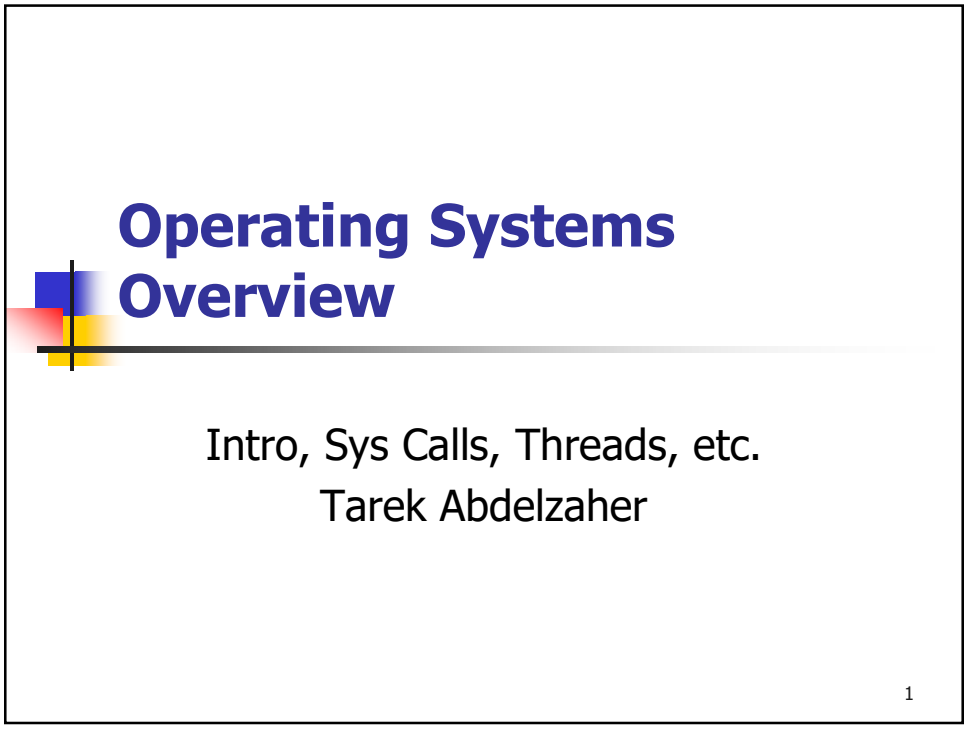
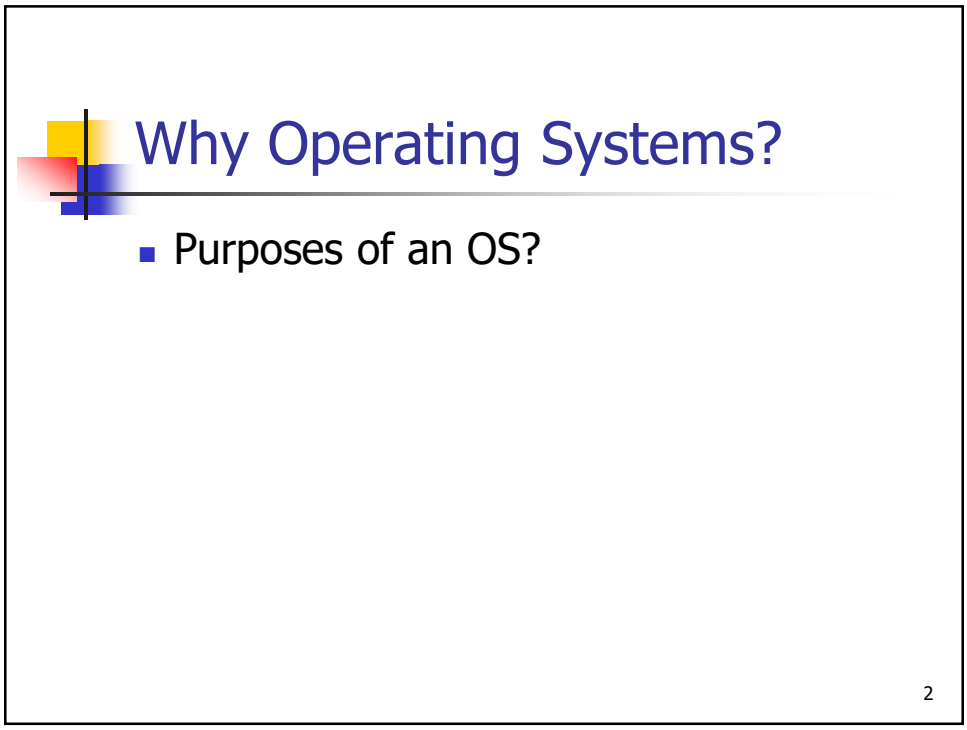
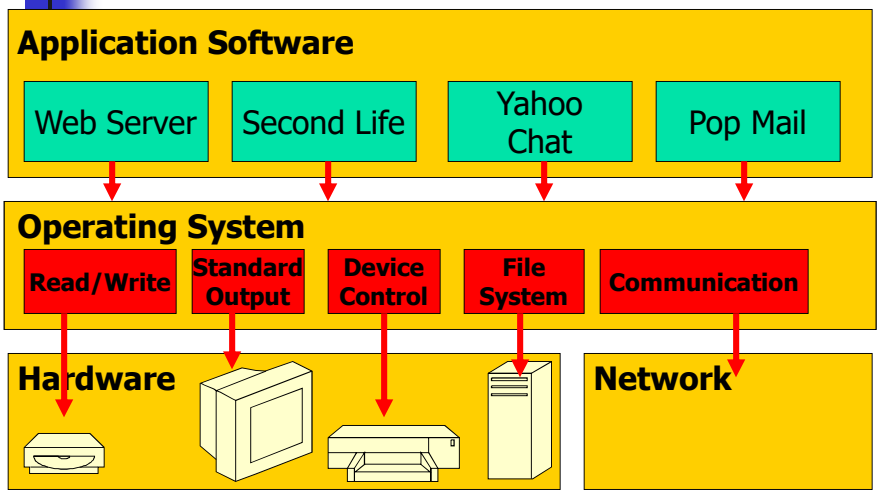

Operating Systems
Overview
Intro, Sys Calls, Threads, etc.
Tarek Abdelzaher
1

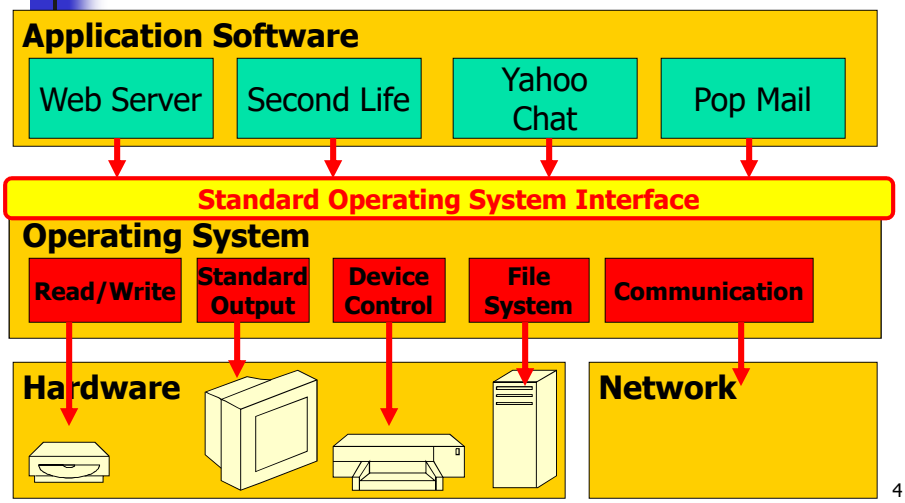

Why Operating Systems?
Purposes of an OS?
2

An Operating System Performs Common Application Functions

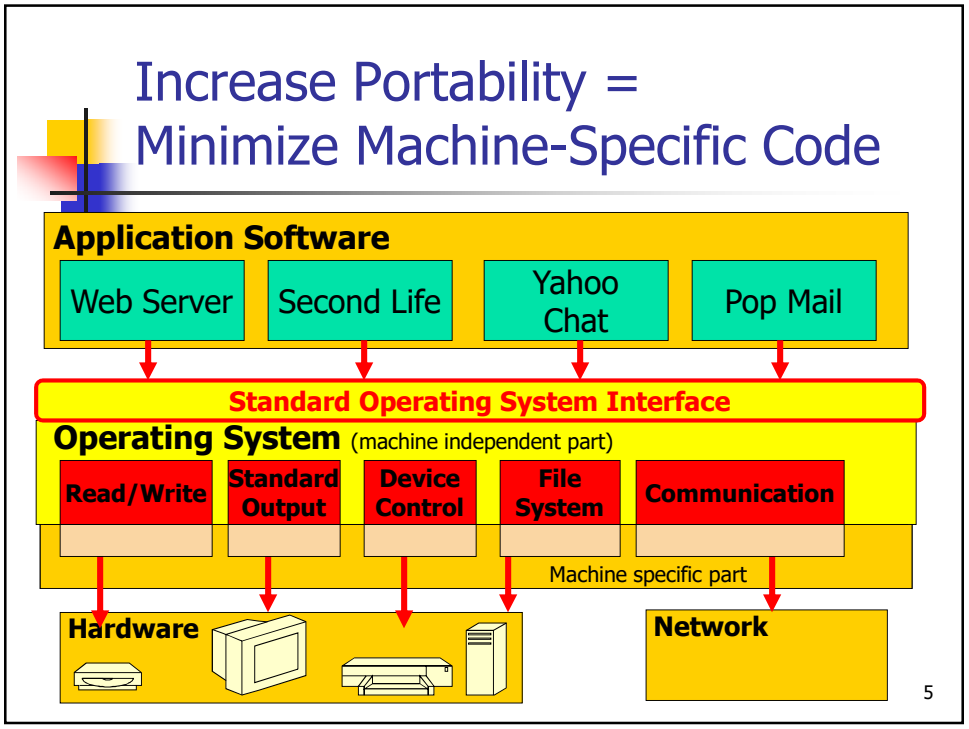


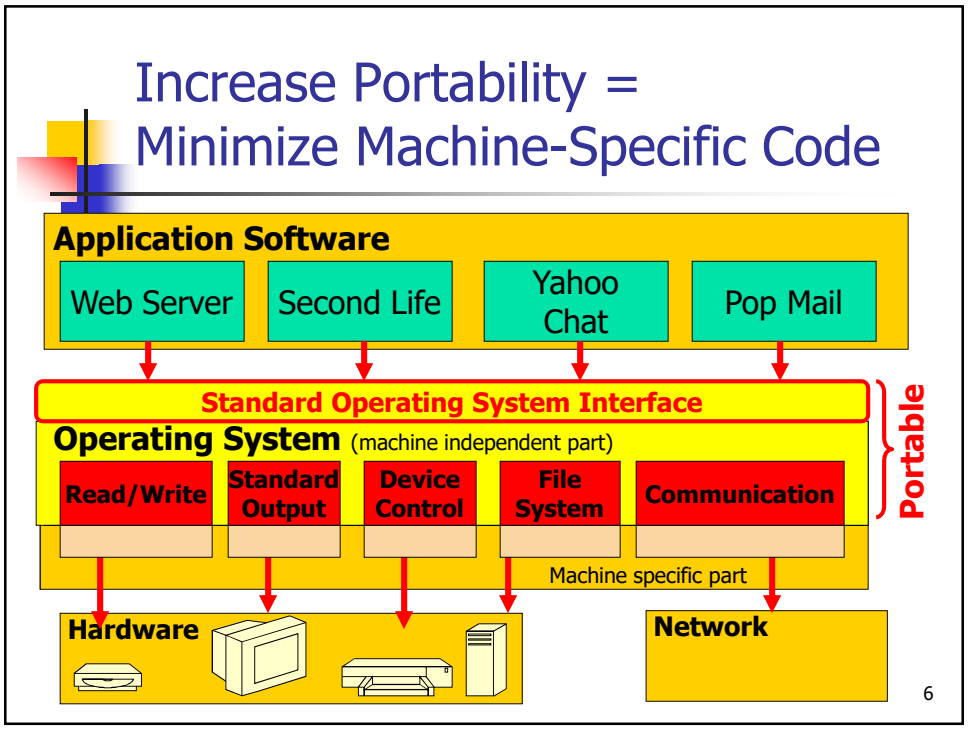
3

OS Exports a User Interface

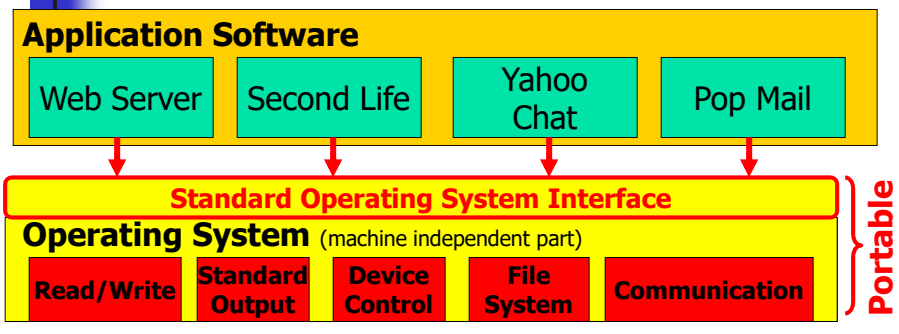


4


Increase Portability =
Minimize Machine-Specific Code
Application Software
Web Server
Second Life
Yahoo Chat
Pop Mail
Standard Operating System Interface
Operating System (machine independent part)
Read/Write
Standard Output
Device Control
File System
Communication
Machine specific part
Hardware
Network
5

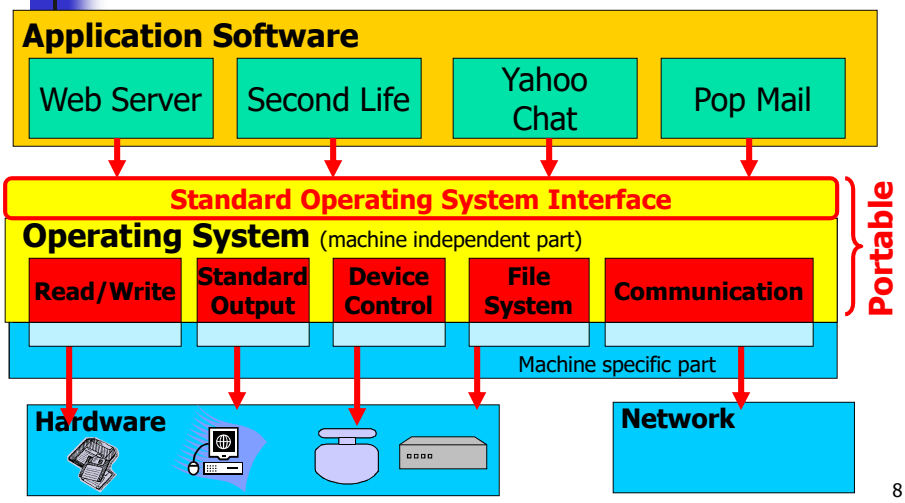

Increase Portability =
Minimize Machine-Specific Code
Application Software
Web Server
Second Life
Yahoo Chat
Pop Mail
Standard Operating System Interface
Operating System (machine independent part)
Read/Write
Standard Output
Device Control
File System
Communication
Portable
Machine specific part
Hardware
Network
6

OS Runs on Multiple Platforms while Presenting same Interface



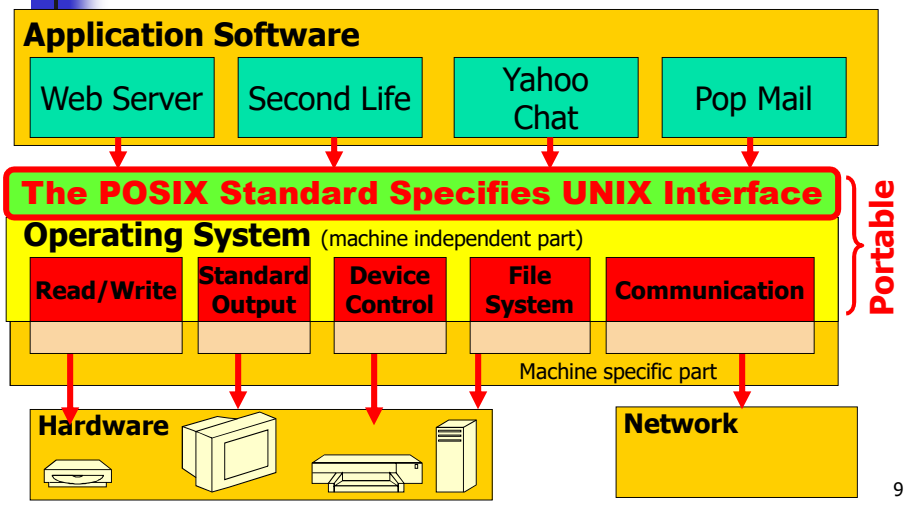
7

OS Runs on Multiple Platforms while Presenting same Interface



8

Linux
(The POSIX Standard)



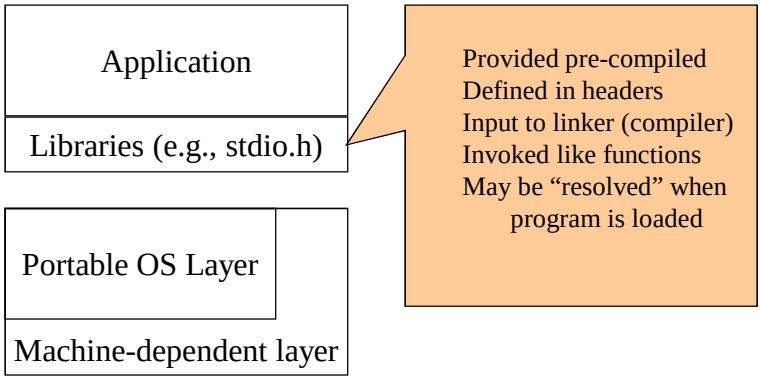
9

OS Goals

- Provide an interface for applications to do common functions.

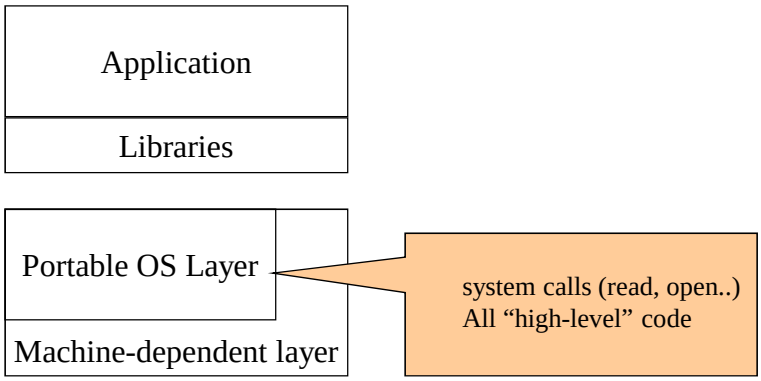
10

Software Layers: Applications call Libraries



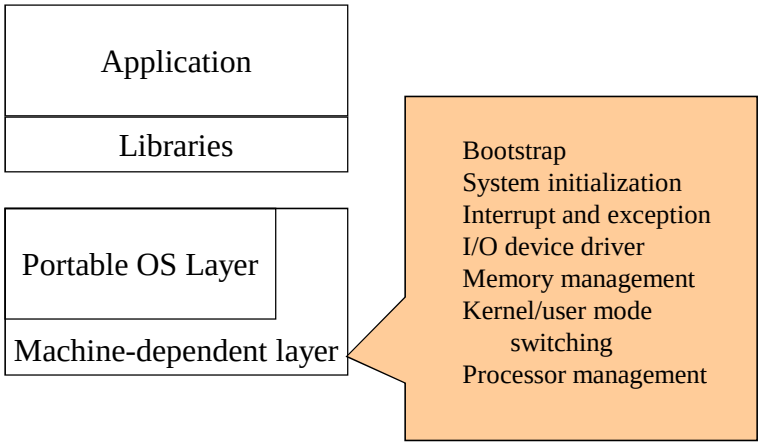
11

Software Layers: Libraries make OS System Calls



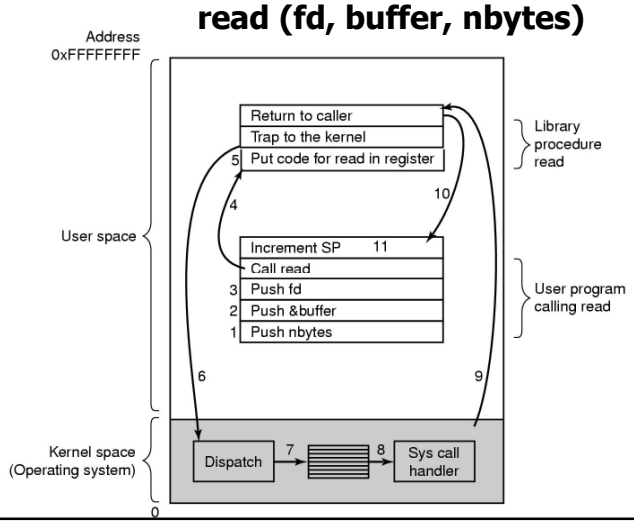
12

Software Layers: System Calls access Drivers, etc.



13

Steps in Making a System Call



14



Some System Calls For Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

15

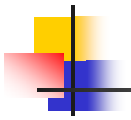


Some System Calls For File Management

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

16

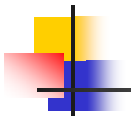


Some System Calls For Directory Management

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

17

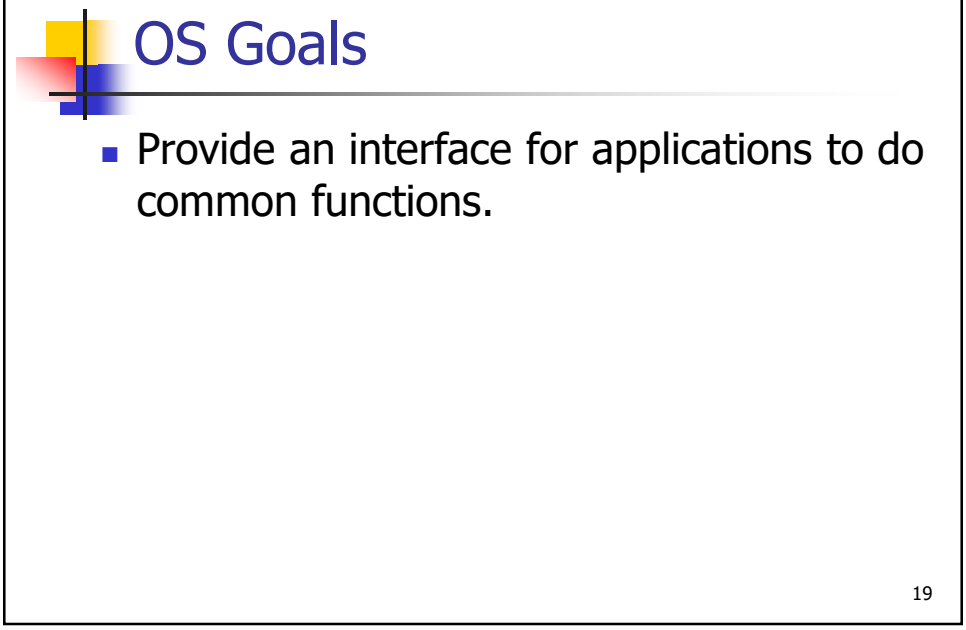


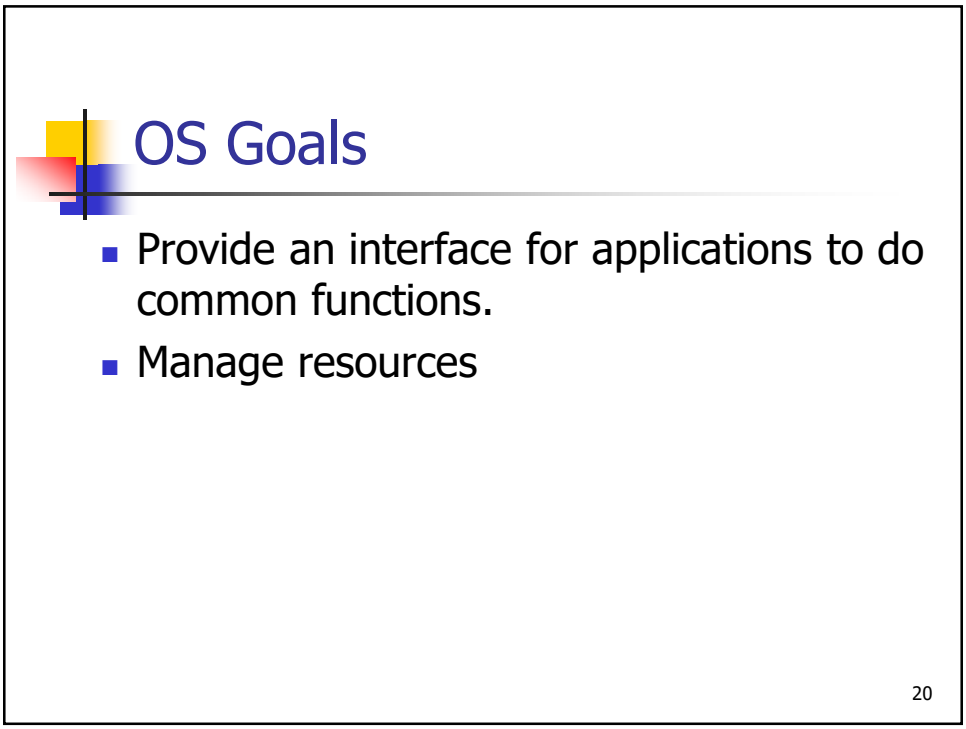
Some System Calls For Miscellaneous Tasks

Miscellaneous

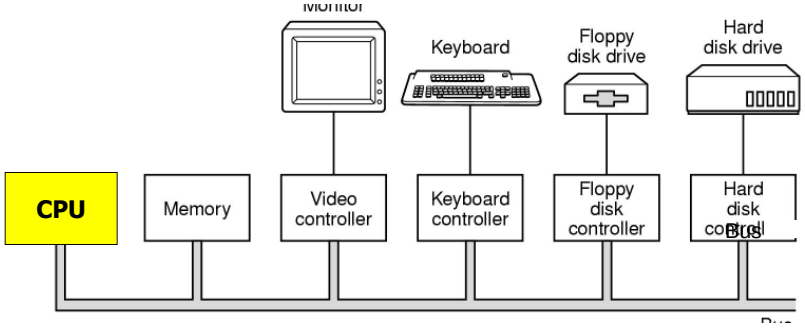
Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

18


OS Goals
Provide an interface for applications to do common functions.
19


OS Goals
Provide an interface for applications to do common functions.
Manage resources
20

Managed Resources - CPU



Process/thread scheduling, context switching, etc.

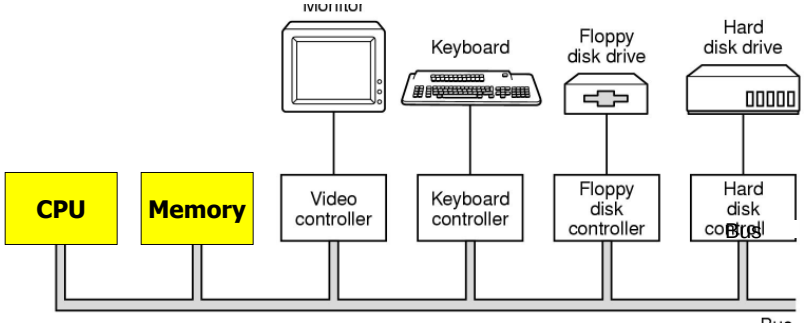
21

What Scheduling Policies are Available in Linux?

- SCHED_FIFO, SCHED_RR, SCHED_NORMAL, SCHED_FAIR (2.6+), …
- What is nice()?
- What are priorities?
- What are real-time priorities?

22

Managed Resource - Memory



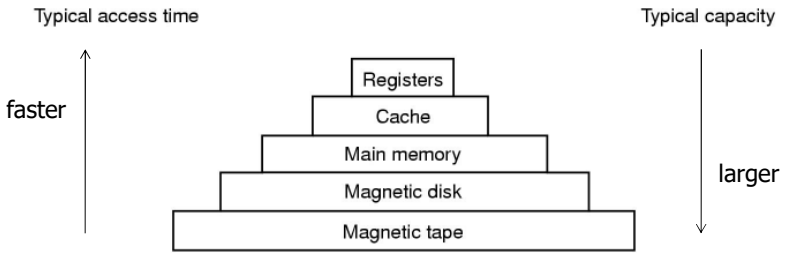
Paging, swapping, virtual memory management, etc.

23

What is the Purpose of a Memory Hierarchy?

24

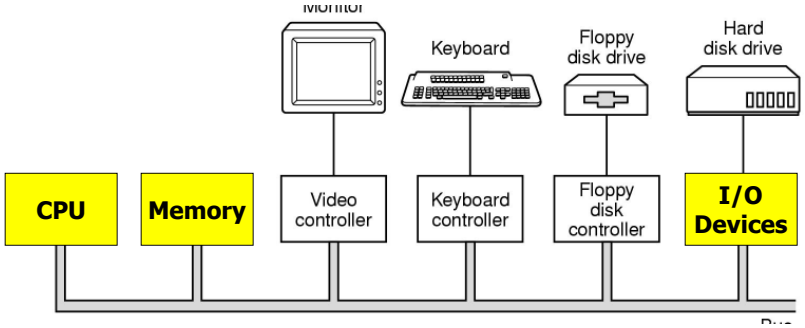
Memory Hierarchy



- Make it look like it's as large as the disk and as fast as the cache

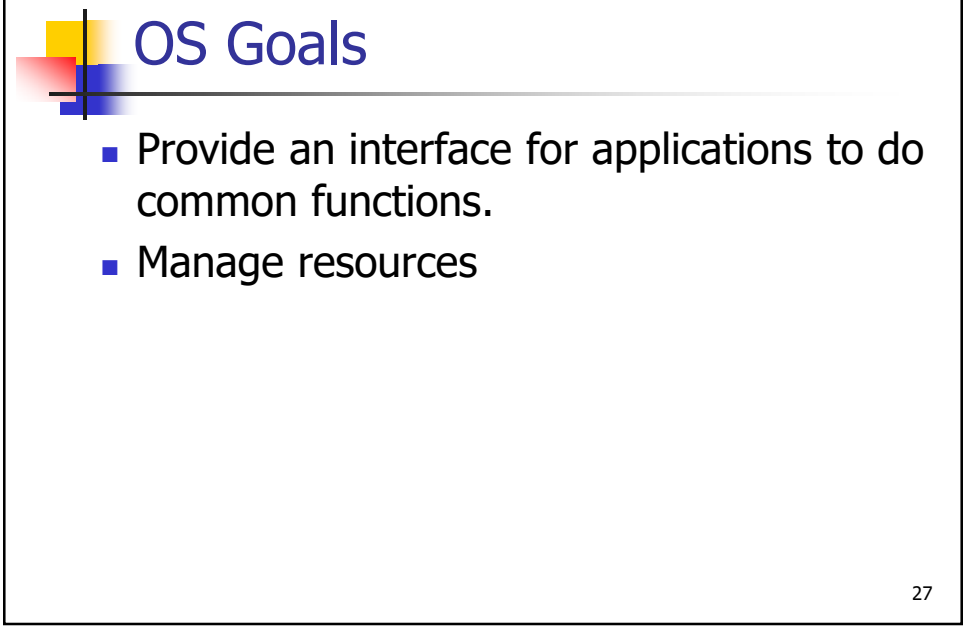
25

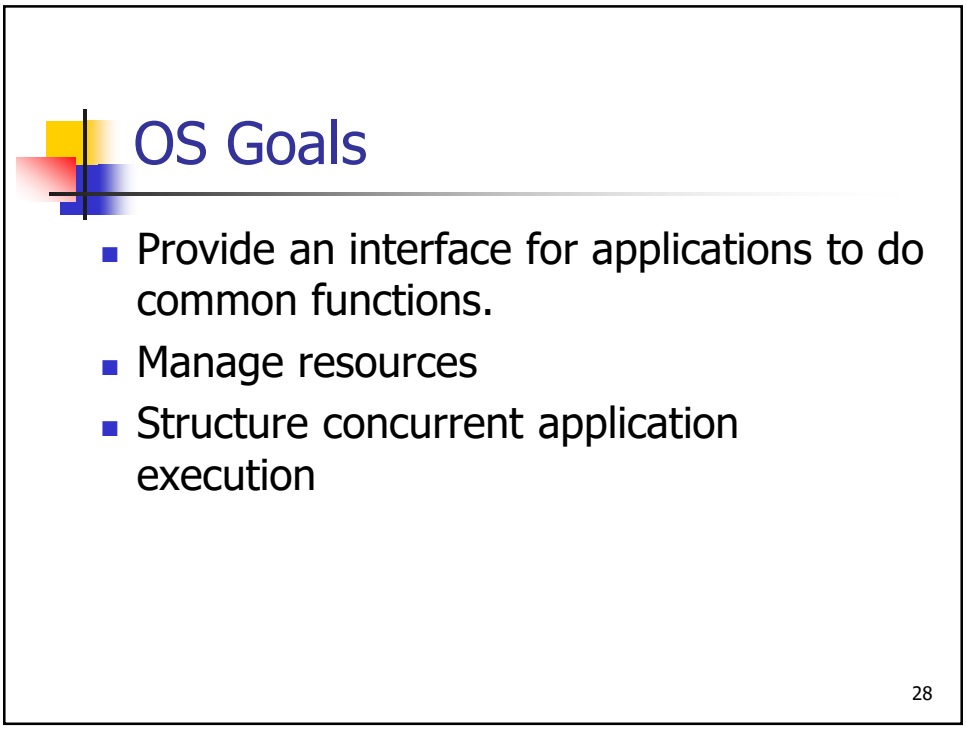
Managed Resource – I/O

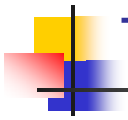


Drivers, interrupts, timers, files, etc.

26


OS Goals
Provide an interface for applications to do common functions.
Manage resources
27


OS Goals
Provide an interface for applications to do common functions.
Manage resources
Structure concurrent application execution
28



Threads and Processes

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

29



When Should my Application Use Multiple Threads?

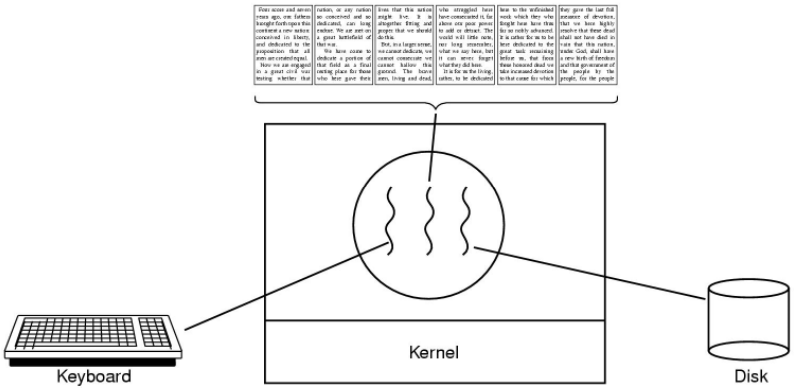
30

Things Suitable for Threading

- Block for potentially long waits
- Use many CPU cycles
- Respond to asynchronous events
- Execute functions of different importance
- Execute parallel code

31

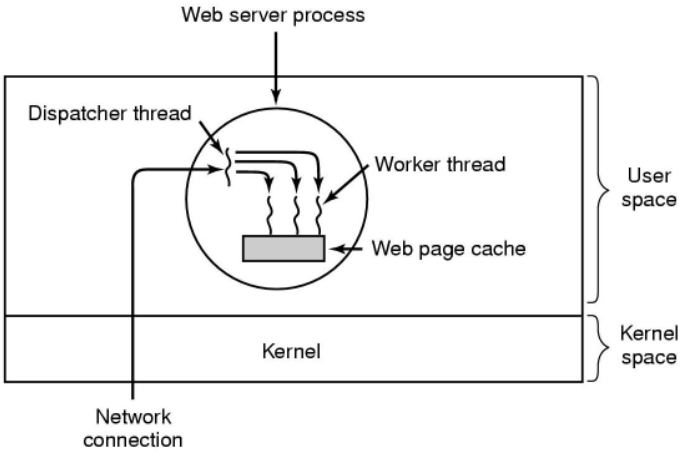
Thread Usage: Word Processor



- What if it is single-threaded?

32

Thread Usage: Web Server



33

Common Multi-thread Software Architectures

- ***Manager/worker***
  - a single thread, the *manager* assigns work to other threads, the *workers*. Typically, the manager handles all input and parcels out work to the other tasks
- ***Pipeline***
  - a task is broken into a series of sub-operations, each of which is handled by a different thread. An automobile assembly line best describes this model
- ***Peer***
  - similar to the manager/worker model, but after the main thread creates other threads, it participates in the work.

34



User-level Threads

- **Advantages**
  - Fast Context Switching:
    - User level threads are implemented using user level thread libraries, rather than system calls, hence no call to OS and no interrupts to kernel
    - When a thread is finished running for the moment, it can call **thread_yield.** This instruction (a) saves the thread information in the thread table, and (b) calls the thread scheduler to pick another thread to run.
    - The procedure that saves the local thread state and the scheduler are local procedures, hence no trap to kernel, no context switch, no memory switch, and this makes the thread scheduling very fast.
  - Customized Scheduling

35



Kernel-level Threads

- Kernel can schedule threads in addition to processes.
- Multiple threads of a process can run simultaneously on multiple CPUs.
- Synchronization more efficient than for processes (but less than for user-level threads).
- Kernel-level threads can make blocking I/O calls without blocking other threads of same process

36