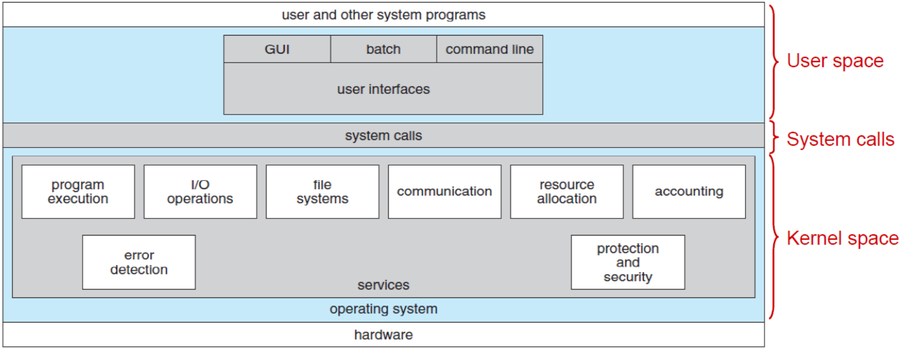
Introduction to Operating Systems (Part III)

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Operating System Structure

Operating System Structure

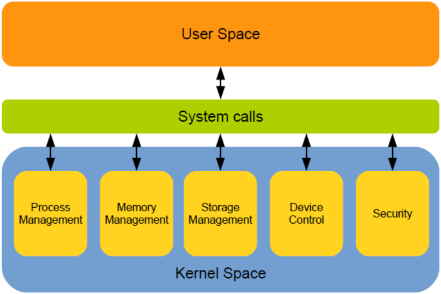


User Space

- System programs
- Application programs

Kernel Space

- Process management
- Memory management
- Storage management and File system
- Device control and I/O subsystem
- Protection and security



System Calls

System Calls

- Programming interface to the services provided by the OS.

System Calls

- Programming interface to the services provided by the OS.
- Typically written in a high-level language (C or C++).

System Calls

- Programming interface to the services provided by the OS.
- Typically written in a high-level language (C or C++).
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use.

Application Programming Interface (API)

- The API specifies a set of functions that are available to an application programmer.

Application Programming Interface (API)

- The API specifies a set of functions that are available to an application programmer.
    - It includes the parameters that are passed to each function and the return values the programmer can expect.

Application Programming Interface (API)

- The API specifies a set of functions that are available to an application programmer.
  - It includes the parameters that are passed to each function and the return values the programmer can expect.

- Three most common APIs:

Application Programming Interface (API)

- The API specifies a set of functions that are available to an application programmer.
  - It includes the parameters that are passed to each function and the return values the programmer can expect.

- Three most common APIs:
  - POSIX API for Unix, Linux, and Mac OS X

Application Programming Interface (API)

- The API specifies a set of functions that are available to an application programmer.
  - It includes the parameters that are passed to each function and the return values the programmer can expect.

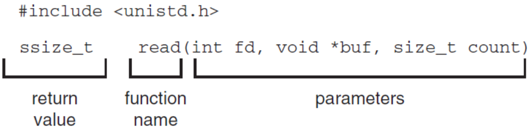
- Three most common APIs:
  - POSIX API for Unix, Linux, and Mac OS X
  - Win32 API for Windows

Application Programming Interface (API)

- The API specifies a set of functions that are available to an application programmer.
  - It includes the parameters that are passed to each function and the return values the programmer can expect.

- Three most common APIs:
  - POSIX API for Unix, Linux, and Mac OS X
  - Win32 API for Windows
  - Java API for the Java virtual machine (JVM)

Example of Standard API

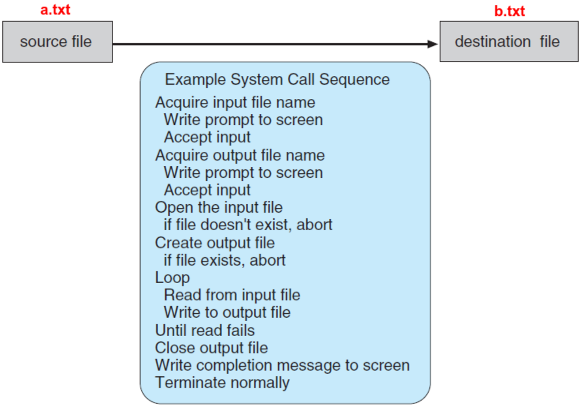



```
> man read
READ(2)            Linux Programmers Manual                    READ(2)

NAME
       read - read from a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t read(int fd, void *buf, size_t count);

DESCRIPTION
       read() attempts to read up to count bytes from file descriptor fd into the buffer starting
       at buf.

       If count is zero, read() returns zero and has no other results. If count is greater than
       SSIZE_MAX, the result is unspecified.

RETURN VALUE
       On success, the number of bytes read is returned (zero indicates end of file), and the
       file position is advanced by this number.

...
```

- Why would an application programmer prefer programming according to an API rather than invoking actual system calls?

API and System Calls

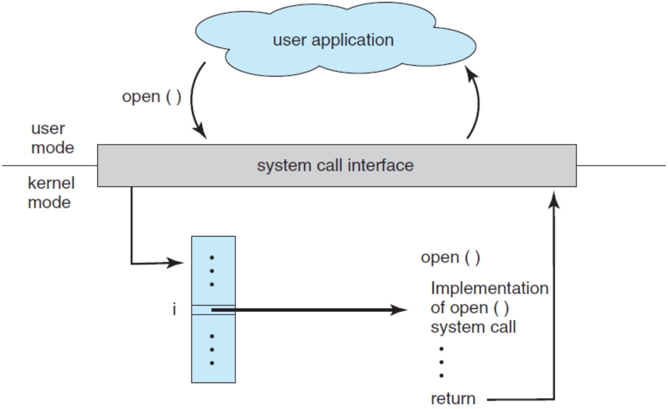
```
> cp a.txt b.txt
```



API and System Calls

```
> strace cp a.txt b.txt
execve("/bin/cp", ["cp", "a.txt", "b.txt"], [/* 49 vars */]) = 0
brk(0)                                    = 0x8a2d000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb76ff000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=108563, ...}) = 0
mmap2(NULL, 108563, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76e4000
close(3)                                  = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0@A\0\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=120748, ...}) = 0
mmap2(NULL, 125852, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb76c5000
mmap2(0xb76e2000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c) = 0xb76e2000
close(3)                                  = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320\30\0\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=30684, ...}) = 0
mmap2(NULL, 33360, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb76bc000
mmap2(0xb76c3000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6) = 0xb76c3000
close(3)                                  = 0
...
```

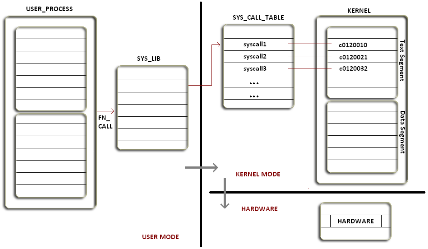
System Call Interface



- The system call interface intercepts function calls in the API and invokes the necessary system calls within the OS.

System Calls Implementation (1/2)

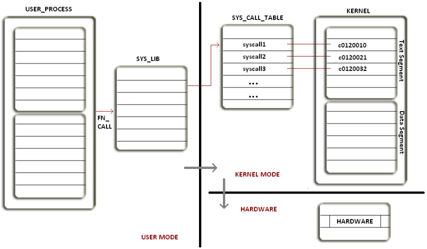
- Typically, a number associated with each system call.



[http://www.rootkitanalytics.com/kernelland]

System Calls Implementation (1/2)

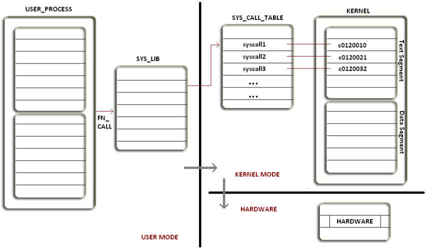
- Typically, a number associated with each system call.
- System-call interface maintains a table indexed according to these numbers.



[http://www.rootkitanalytics.com/kernelland]

System Calls Implementation (1/2)

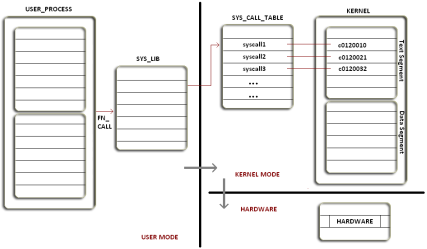
- Typically, a number associated with each system call.
- System-call interface maintains a table indexed according to these numbers.
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values.



[http://www.rootkitanalytics.com/kernelland]

System Calls Implementation (2/2)

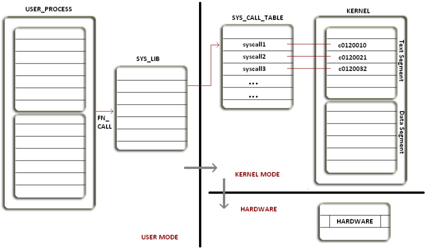
- The caller does not need to know about the system call implementation.



[http://www.rootkitanalytics.com/kernelland]

System Calls Implementation (2/2)

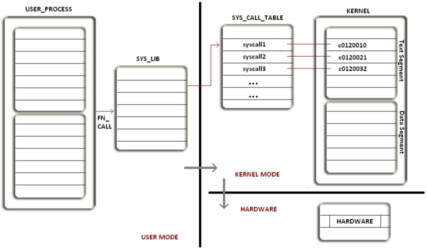
- The caller does not need to know about the system call implementation.
- Just needs to obey API and understand what OS will do as a result call.



[http://www.rootkitanalytics.com/kernelland]

System Calls Implementation (2/2)

- The caller does not need to know about the system call implementation.
- Just needs to obey API and understand what OS will do as a result call.
- Most details of OS interface hidden from programmer by API.



[http://www.rootkitanalytics.com/kernelland]

System Call Parameter Passing (1/2)

- Three general methods used to pass parameters to the OS.

System Call Parameter Passing (1/2)

- Three general methods used to pass parameters to the OS.

1. Pass the parameters in registers.
   - In some cases, may be more parameters than registers.

System Call Parameter Passing (1/2)

- Three general methods used to pass parameters to the OS.

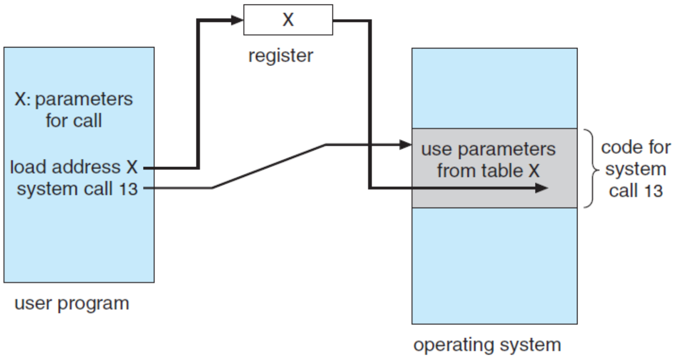
1. Pass the parameters in registers.
   - In some cases, may be more parameters than registers.
2. Parameters stored in a block or table, in memory, and address of block passed as a parameter in a register.

System Call Parameter Passing (1/2)

- Three general methods used to pass parameters to the OS.

1. Pass the parameters in registers.
   - In some cases, may be more parameters than registers.
2. Parameters stored in a block or table, in memory, and address of block passed as a parameter in a register.
3. Parameters pushed onto the stack by the program and popped off the stack by the OS.

System Call Parameter Passing (2/2)



[Passing of parameters as a table]

Types of System Calls

- System calls can be grouped roughly into six major categories:

1. Process control
2. File manipulation
3. Device manipulation
4. Information maintenance
5. Communications
6. Protection

Process Control System Calls

- create process, terminate process

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

Process Control System Calls

- create process, terminate process
- end, abort

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

Process Control System Calls

- create process, terminate process
- end, abort
- load, execute

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

Process Control System Calls

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

Process Control System Calls

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

Process Control System Calls

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

Process Control System Calls

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()

File Management System Calls

- create file, delete file

	Windows	Unix
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()

File Management System Calls

- create file, delete file
- open, close

	Windows	Unix
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()

File Management System Calls

- create file, delete file
- open, close
- read, write, reposition

	Windows	Unix
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()

File Management System Calls

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

	Windows	Unix
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()

Device Management System Calls

- request device, release device

	Windows	Unix
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()

Device Management System Calls

- request device, release device
- read, write, reposition

	Windows	Unix

Device Manipulation	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		write()

Device Management System Calls

- request device, release device
- read, write, reposition
- get device attributes, set device attributes

	Windows	Unix
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()

Information Management System Calls

- get/set time or date

	Windows	Unix
Information Maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`

Information Management System Calls

- get/set time or date
- get/set system data

	Windows	Unix
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()

Information Management System Calls

- get/set time or date
- get/set system data
- get/set process, file, or device attributes

	Windows	Unix
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()

Communications System Calls

- create, delete communication connection

	Windows	Unix
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()

Communications System Calls

- create, delete communication connection
- send, receive messages

	Windows	Unix
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()

Communications System Calls

- create, delete communication connection
- send, receive messages
- transfer status information

	Windows	Unix
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()

Communications System Calls

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

	Windows	Unix
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()

Operating System Architecture

Operating System Architecture (1/2)

- General-purpose OS is very large program.

Operating System Architecture (1/2)

- General-purpose OS is very large program.
- A common approach is to partition the task into small components, rather than have one monolithic system.

Operating System Architecture (1/2)

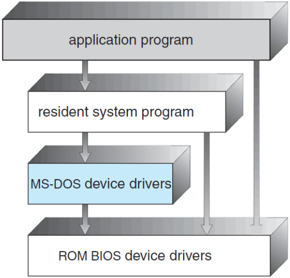
- General-purpose OS is very large program.
- A common approach is to partition the task into small components, rather than have one monolithic system.
- Each component should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.

Operating System Architecture (2/2)

- Various ways to structure ones:
  - Simple structure, e.g., MS-DOS
  - More complex structure, e.g., Unix
  - Layered, an abstraction
  - Microkernel, e.g., Mach

Simple Structure

- Provide the most functionality in the least space.
- Not divided into modules.
- Its interfaces and levels of functionality are not well separated.



More Complex Structure (1/2)

- the original Unix had limited structuring.

More Complex Structure (1/2)

- the original Unix had limited structuring.
- The Unix consists of two separable parts:

More Complex Structure (1/2)

- the original Unix had limited structuring.
- The Unix consists of two separable parts:

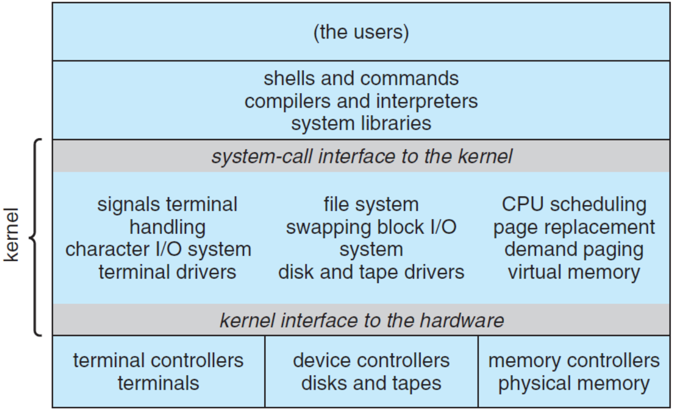
1. Systems programs
2. The kernel

More Complex Structure (1/2)

- the original Unix had limited structuring.
- The Unix consists of two separable parts:

1. Systems programs
2. The kernel
    - Everything below the system call interface and above the hardware.
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

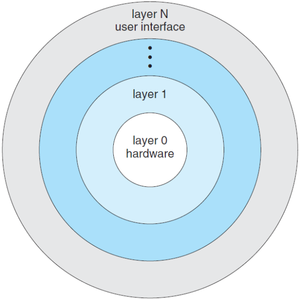
More Complex Structure (2/2)



[Traditional Unix system structure]

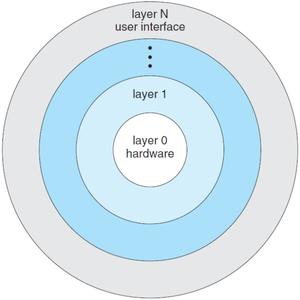
Layered Structure

- The operating system is divided into a number of layers.



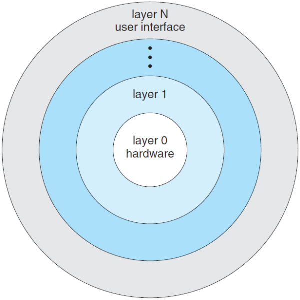
Layered Structure

- The operating system is divided into a number of layers.
- Each layer is built on top of lower layers.



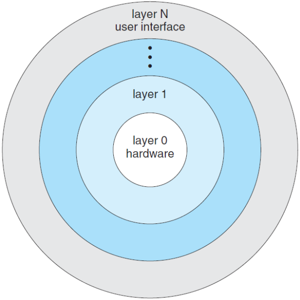
Layered Structure

- The operating system is divided into a number of layers.
- Each layer is built on top of lower layers.
- The bottom layer is the hardware.
- The highest layer is the user interface.



Layered Structure

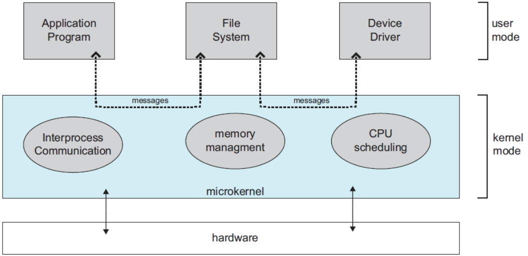
- The operating system is divided into a number of layers.
- Each layer is built on top of lower layers.
- The bottom layer is the hardware.
- The highest layer is the user interface.



- Layers are selected such that each uses functions and services of only lower-level layers.

Microkernel Structure (1/2)

- Moves as much from the kernel into user space.
- Communication takes place between user modules using message passing.



Microkernel Structure (2/2)

- Advantages:
  - Easier to extend a microkernel.
  - Easier to port the OS to new architectures.
  - More reliable (less code is running in kernel mode).
  - More secure.

Microkernel Structure (2/2)

- Advantages:
  - Easier to extend a microkernel.
  - Easier to port the OS to new architectures.
  - More reliable (less code is running in kernel mode).
  - More secure.

- Disadvantages:
  - Performance overhead of user space to kernel space communication.

Modules

- Many modern operating systems implement loadable kernel modules.
  - Uses object-oriented approach.
  - Each core component is separate.
  - Each talks to the others over known interfaces.
  - Each is loadable as needed within the kernel.

Modules

- Many modern operating systems implement loadable kernel modules.
  - Uses object-oriented approach.
  - Each core component is separate.
  - Each talks to the others over known interfaces.
  - Each is loadable as needed within the kernel.

- Overall, similar to layers but with more flexible.
  - Linux, Solaris, ...

Hybrid Systems

- Most modern operating systems are actually not one pure model.

- Hybrid combines multiple approaches to address performance, security, usability needs.

Hybrid Systems

- Most modern operating systems are actually not one pure model.
- Hybrid combines multiple approaches to address performance, security, usability needs.
- Linux and Solaris are monolithic, plus modular for dynamic loading of functionality.

Hybrid Systems

- Most modern operating systems are actually not one pure model.
- Hybrid combines multiple approaches to address performance, security, usability needs.
- Linux and Solaris are monolithic, plus modular for dynamic loading of functionality.
- Windows mostly monolithic, plus microkernel for different subsystem personalities.

Hybrid Systems

- Most modern operating systems are actually not one pure model.
- Hybrid combines multiple approaches to address performance, security, usability needs.
- Linux and Solaris are monolithic, plus modular for dynamic loading of functionality.
- Windows mostly monolithic, plus microkernel for different subsystem personalities.
- Apple Mac OS X kernel consists of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules

Operating System Design and Implementation

Operating System Design Goals

- The first problem in designing a system is to define goals and specifications.

Operating System Design

- The design of the system will be affected by the choice of hardware and the type of system:
  - batch
  - time sharing
  - single user/multiuser
  - distributed
  - real time
  - ...

Users Goals vs. System Goals

- User goals: OS should be convenient to use, easy to learn, reliable, safe, and fast.
- System goals: OS should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

How vs. What

- Separating policy from mechanism:
    - Policy: what to do?
    - Mechanism: how to do?

How vs. What

- Separating policy from mechanism:
  - Policy: what to do?
  - Mechanism: how to do?
- Mechanisms determine how to do something, policies decide what will be done.

How vs. What

- Separating policy from mechanism:
  - Policy: what to do?
  - Mechanism: how to do?
- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism allows maximum flexibility if policy decisions are to be changed later.

Implementation

- Early OSes in assembly language.
- Then system programming languages like Algol, PL/1.
- Now C and C++.
- Actually usually a mix of languages:
  - Lowest levels in assembly.
  - Main body in C.
  - Systems programs in C, C++, scripting languages, e.g., Python.

Implementation

- Early OSes in assembly language.
- Then system programming languages like Algol, PL/1.
- Now C and C++.
- Actually usually a mix of languages:
  - Lowest levels in assembly.
  - Main body in C.
  - Systems programs in C, C++, scripting languages, e.g., Python.
- More high-level language easier to port to other hardware, but slower.

Summary

Summary

- Operating-system structure: user-space, system calls, kernel-space

Summary

- Operating-system structure: user-space, system calls, kernel-space
- System calls:
  - File manipulation
  - Device manipulation
  - Information maintenance
  - Communications
  - Protection

Summary

- Operating-system structure: user-space, system calls, kernel-space
- System calls:
  - File manipulation
  - Device manipulation
  - Information maintenance
  - Communications
  - Protection
- Operating-system architecture: simple, layered, micro-kernel, hybrid

Questions?