An Introduction to Operating Systems

Amir H. Payberah
payberah@kth.se

2022

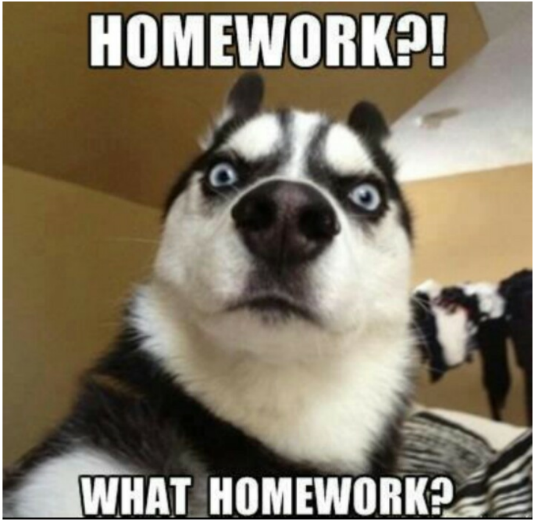
Course Information

Course Objective

- The purpose of this course is to teach the design of operating systems.
- The course has five modules:
  - Module 1: Process management
  - Module 2: Process synchronization
  - Module 3: Memory management
  - Module 4: Storage management
  - Module 5: File systems

Intended Learning Outcomes (ILOs)

- ILO1: Understand the main OS modules, i.e., managing process, memory, and storage.
- ILO2: Apply the grabbed knowledge to implement the given tasks in different OS modules.
- ILO3: Analyze the technical merits of a specific OS module.


HOMEWORK?!
WHAT HOMEWORK?

The Course Assessment

- Task1: the review questions.
- Task2: the lecture assignments.
- Task3: the lab assignments.
- Task4: the essay and the presentation.
- Task5: the final exam.

How Each ILO is Assessed?

	Task1	Task2	Task3	Task4	Task5
ILO1	x	x			x
ILO2		x	x		
ILO3				x	

Task1: The Review Questions

- One set of review questions per module.
- The review questions are graded P/F.
- They should be done individually.

Task2: The Lecture Assignments

- One lecture assignment per lecture.
- No deadline.

Task3: The Lab Assignments

- One lab assignment per module.
- The review questions are graded P/F.
- They should be done in group.

Task4: The Essay and The Presentation

- One module for each group: writing an essay and presenting it to their opponents (another group).
- Grading of this task has the following parts:
  - E: Essay (weight 50%)
  - P: Presentation (weight 20%)
  - Q: Reviewing another essay and asking questions (weight 20%)
  - A: Answering questions (weight 10%)
- Each part is graded A-F.
- The final grade is computed as $0.5 \times E + 0.2 \times P + 0.2 \times Q + 0.1 \times A$.

Task5: The Final Exam

- The final exam covers all the modules presented during the course
- It is graded A-F.

The Final Grade

- To pass the course: you must pass Task 1 and Task 3 and get at least E in Task 4 and Task 5.
- The final grade of the course is computed as $0.5 \times Task4 + 0.5 \times Task5$.

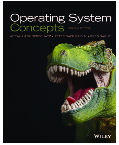
"Why is an A or B better than a C or D?
Aren't all letters equal in the eyes of God?"

How to Submit the Assignments?

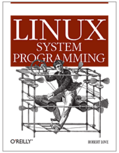
- Through Canvas.
- You will work individually on Task 1 and Task 5.
- You will work in groups of three or four on Task 3 and Task 4.

Course Textbooks

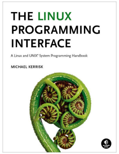
- Operating System Concepts, 10th Edition
  Avil Silberschatz et al., Wiley, 2018



- Linux System Programming, 2nd Edition
  Robert Love, O'Relly Media, 2013



- The Linux Programming Interface
  Michael Kerrisk, No Starch Press, 2010



The Course Web Page

`https://kth-os.github.io`

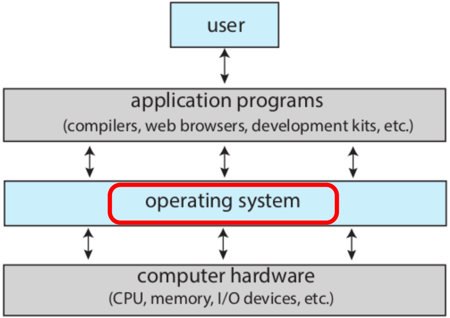
The Discussion Page

`https://tinyurl.com/35avmfea`

What is an Operating System?

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.


user
application programs
(compilers, web browsers, development kits, etc.)
operating system
computer hardware
(CPU, memory, I/O devices, etc.)

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.

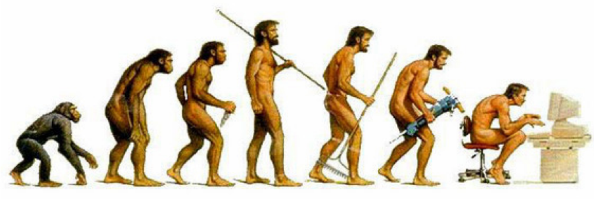
What Operating Systems Do?

- OS is a resource allocator
  - Manages all resources.
  - Decides between conflicting requests for efficient and fair resource use.
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer.

Operating Systems Definition

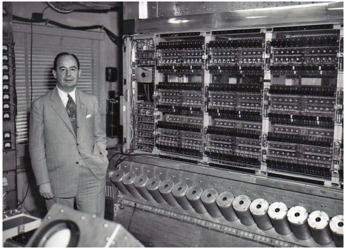
- The operating system is the one program running at all times on the computer, usually called the kernel.
- Everything else is either a system program or an application program.

A Brief History of Operating Systems



First Generation: 1945-1955 (1/2)

- No operating system
- Human was the operator and programmer.
- Computers were programmed by physically re-wiring them.
- Programs written in machine or assembly language.



[http://ysfine.com/wigner/neumann.html]

First Generation: 1945-1955 (2/2)

- Problems:
  - Serial processing: users had access to the computer one by one in series.
  - Users had to write again and again the same routines.

Second Generation: 1955-1965 (1/5)

- Mainframes

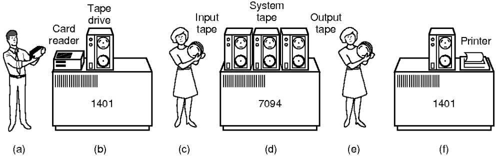
IBM 7094 at Columbia University
[http://www.columbia.edu/cu/computinghistory/1965.html]

Second Generation: 1955-1965 (2/5)

- Separation between operators and programmers.
  - The programmer: prepares her/his job off-line.
  - The operator: runs the job and delivers a printed output.
- Job
  - A program or set of programs.
  - A programmer would punch it on cards.
  - Programs are in FORTRAN or in assembly language.

Second Generation: 1955-1965 (3/5)

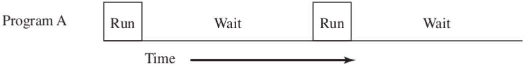
- Batch the jobs together.
- The operator pre-reads jobs onto a magnetic tape.
- The operator loads a special program (monitor) that reads the jobs from the tapes and run them sequentially.
- The monitor program writes the output of each job on a second magnetic tape.
- The operator brings the full output tape for offline printing.



[A.S. Tanenbaum et al., Operating Systems Design and Implementation, 2006]

Second Generation: 1955-1965 (4/5)

- Problems:
  - A lot of CPU time is still wasted waiting for I/O instructions to complete.
  - I/O devices much slower than processor (especially tapes!)



[W. Stallings, Operating Systems: Internals and Design Principles, 2011]

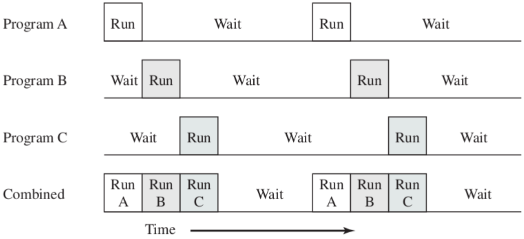
Second Generation: 1955-1965 (5/5)

- More important problems:
  - Operating mainframes was viewed as a low-level and low-value work.
  - Racist and sexist job: operators were often women.

[https://www.nytimes.com/2019/02/13/magazine/women-coding-computer-programming.html]

Third Generation: 1965-1980 (1/3)

- Multiprogrammed batch systems.
- Jobs are kept in main memory at the same time and the CPU is multiplexed among them or multiprogrammed.



[W. Stallings, Operating Systems: Internals and Design Principles, 2011]

Third Generation: 1965-1980 (2/3)

- Tasks kept running until they performed an operation that required waiting for an external event such as I/O.
- But, in a multiple-user system, users want to see their program running as if it was the only program in the computer.
- Solution? time-sharing or preemptive multitasking systems.

Third Generation: 1965-1980 (3/3)

- Time-sharing
  - Time sharing is a logical extension of multiprogramming for handling multiple interactive jobs among multiple users.
  - Hardware timer interrupt: switching jobs.
- Birth of UNIX!

UNIX®

Fourth Generation: 1980-Present (1/3)

- Personal Computers (PCs)
- Transition from human operators to software (Operating Systems)

[https://metagamer.nl/tips/is-ips-monitor-goed-voor-gaming]

Fourth Generation: 1980-Present (2/3)

- From multiple users back to a single user.
- Multitasking a central feature of modern PC operating systems.
- PC systems emphasize user convenience.

Fourth Generation: 1980-Present (3/3)

- GNU (GNU’s Not Unix!): 1983
- Mac OS: 1984
- Microsoft Windows: 1985
- Linux: 1991

From Mainframe to PC

- Solves many techincal problems, but ...
- Hollywood reinforced stereotypes of PCs as a boys' toy (War Games).
- The result: parents were twice as likely to buy computers for their boys than their girls.
- University CS departments were often elitist, sexist, racist, ableist, and dominated by men.

[https://tv.apple.com/se/movie/wargames/umc.cmc.4n8grrnb4vq7tgygwcd1cxzcq]

From Hobby to Marketplace

- Variety of OS, borrowing liberally from each others' innovations.
- This liberal copying/sharing was also accompanied by fierce, anti-competitive practices.
- These business trends mainly followed free-market policies (neoliberalism).

[https://criticallyconsciouscomputing.org/operating]

Free Software Foundation (1/3)

- In 1971 Richard Matthew Stallman (RMS) joined MIT.
- At that time, all the programmers used to share their code freely.
- In 1980, software companies refused to share the code (copyright).
- In 1985, in response, Stallman, founded the Free Software Foundation (FSF) and published the GNU manifesto.

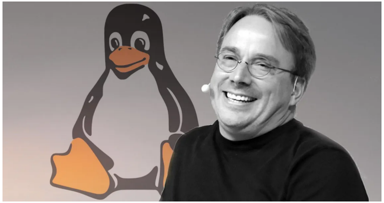
Free Software Foundation (2/3)

- In 1989, Stallman released the first program independent GNU General Public Licence (GPL) or copyleft.
- Now the only thing that GNU lacked was a completely free OS kernel: GNU Hurd kernel
- In 1985, Andy Tanenbaum wrote a Unix like OS from scratch, called Minix.

[https://commons.wikimedia.org/wiki/File:Andrew_S._Tanenbaum.jpg]

Free Software Foundation (3/3)

- In 1990, Linus Torvalds wanted to improve Minix.
- But he was prohibited by Tanenbaum to do so.
- So, Linus implemented his own kernel and released it under GPL: Linux kernel
- Linux, is then, used as the kernel of the GNU in many distributions.

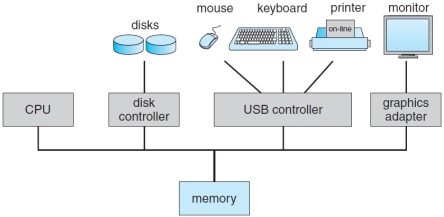


[https://gridinsoft.com/blogs/linus-torvalds-approved-exclusion-of-the-terms-slave-blacklist-and-others-from-the-linux-kernel-code/]

Computer System Operation

Computer-System Operation

- One or more CPUs, and device controllers connect through common bus providing access to shared memory.
- The CPU and the device controllers can execute in parallel, competing for memory cycles.
- Device controllers inform CPU that it is finished with the operation by causing an interrupt.

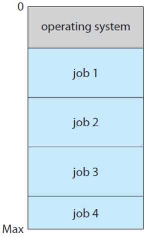


Interrupt

- Hardware may trigger an interrupt at any time by sending a signal to the CPU.
- Software may trigger an interrupt by executing a special operation called a system call.
- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to an address where the service routine for the interrupt is located.
- The CPU resumes the interrupted computation, when the interrupt service routine completes.

Multiprogramming

- Multiprogramming (batch system): needed for efficiency.
- Organizes jobs (code and data), so CPU always has one to execute.
- A subset of total jobs in system is kept in memory.
- One job selected and run via job scheduling.
- When it has to wait (for I/O for example), OS switches to another job.



Time-sharing

- Time-sharing (multitasking): CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing.
  - Providing each user with a small portion of a time-shared computer.
  - Each user has at least one separate program in memory, called a process.
  - Each process typically executes for only a short time.
  - If several jobs ready to run at the same time → CPU scheduling
  - If processes don't fit in memory, swapping moves them in and out to run.

Operating System Structure

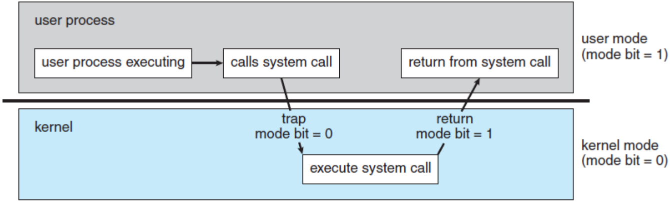
Dual-Mode Operation (1/2)

- The OS and the users share the hardware and software resources of the computer system.

- We need to make sure that an error in a user program could cause problems only for the one program running.
  - E.g., stucking in a finite loop

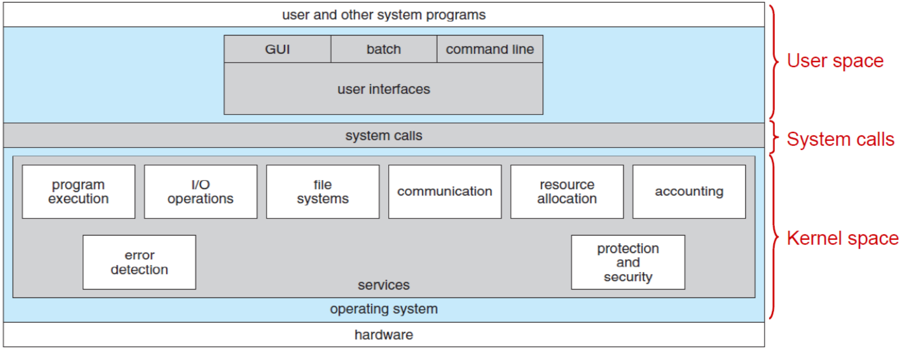
Dual-Mode Operation (2/2)

- Dual-mode operation allows OS to protect itself and other system components.
  - User mode and kernel mode.
  - System call changes mode to kernel, return from call resets it to user.



[Transition from user to kernel mode]

Operating System Structure



User Space

Programs

- Kernel: the program running at all times on a computer.
- Everything else is either:
  - a system program
  - an application program

System Programs

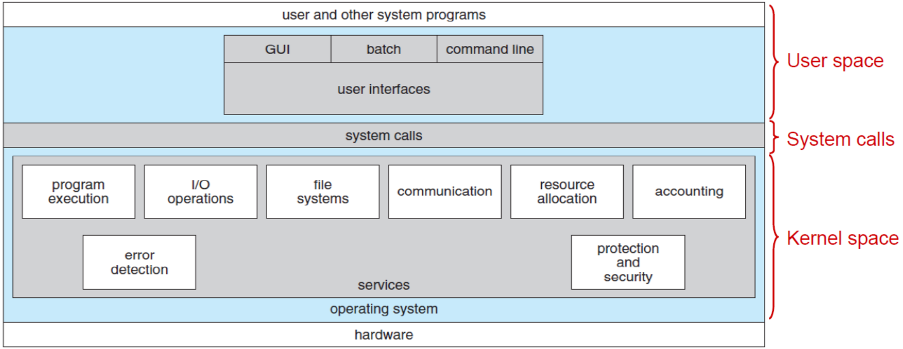
- An environment for program development and execution.
- System programs include:
  - File manipulation, e.g., copy, delete, rename, and edit files
  - Status information, e.g., date, time, and available memory
  - Programming language support, e.g., assemblers, and debuggers
  - Program loading and execution, e.g., loaders
  - Communications, e.g., services to make connections among processes, users, and hardware
  - Background services, e.g., services and daemons

Application Programs

- Don't pertain to system.
- Run by users.
- Not typically considered part of OS.
- Launched by command line, mouse click, finger poke.
- Web browsers, word processors, database systems, compilers, games, ...

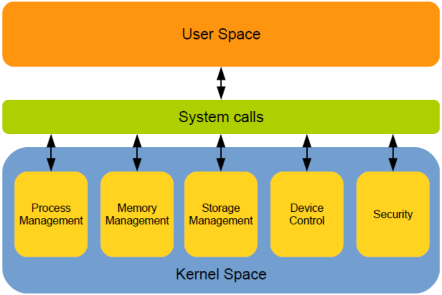
Kernel Space

Operating System Structure



Splitting the Kernel

- The kernel's role can be split into the following parts
  - Process management
  - Memory management
  - Storage management and File system
  - Device control and I/O subsystem
  - Protection and security



Process Management (1/2)

- A process is a program in execution.
  - Program is a passive entity, process is an active entity.
- A process needs resources to accomplish its task.
  - CPU, memory, I/O, files, initialization data, ...
- Process termination requires reclaim of any reusable resources.

Process Management (2/2)

- Process management activities:
  - Scheduling processes and threads on the CPUs.
  - Creating and deleting both user and system processes.
  - Suspending and resuming processes.
  - Providing mechanisms for process synchronization.
  - Providing mechanisms for process communication.

Memory Management (1/2)

- To execute a program all (or part) of the instructions must be in memory.
- All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and when.
  - Optimizing CPU utilization and computer response to users.

Memory Management (2/2)

- Memory management activities:
  - Keeping track of which parts of memory are currently being used and by whom.
  - Deciding which processes (or parts of) and data to move into and out of memory.
  - Allocating and deallocating memory space as needed.

Storage Management (1/3)

- Usually disks used to store data that does not fit in main memory or data that must be kept for a long period of time.

- Disk management activities:
  - Free-space management
  - Storage allocation
  - Disk scheduling

Storage Management (2/3)

- OS provides uniform and logical view of information storage.

- OS abstracts physical properties to logical storage unit, called file.
  - A file is a collection of related information (programs or data).
  - Files usually organized into directories.

- OS maps files onto physical media and accesses these files via the storage devices, e.g., disk drive, tape drive.

Storage Management (3/3)

- File management activities:
  - Creating and deleting files and directories.
  - Primitives to manipulate files and directories.
  - Mapping files onto secondary storage.
  - Backup files onto stable (non-volatile) storage media.

I/O Subsystem

- One purpose of OS is to hide details of hardware devices from the user.
- The I/O subsystem consists of several components:
  - General device-driver interface.
  - Drivers for specific hardware devices.
  - Memory management of I/O.

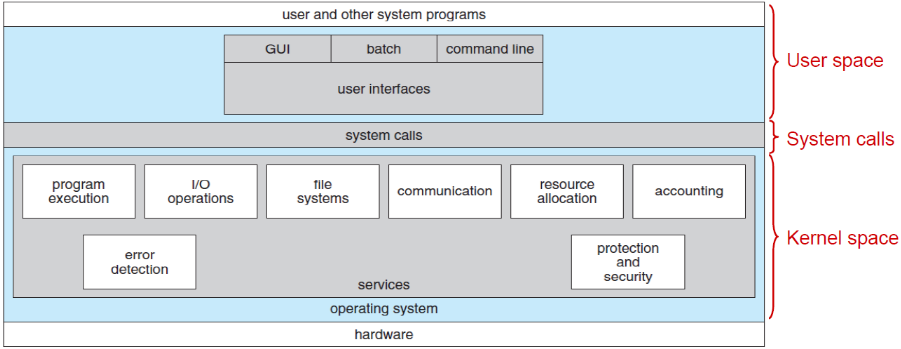
Protection and Security

- **Protection**: any mechanism for controlling access of processes or users to resources defined by the OS.

- **Security**: defense of the system against internal and external attacks.
  - E.g., denial-of-service, worms, viruses, identity theft, theft of service, ...

System Calls

Operating System Structure



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.
  - It includes the parameters that are passed to each function and the return values the programmer can expect.

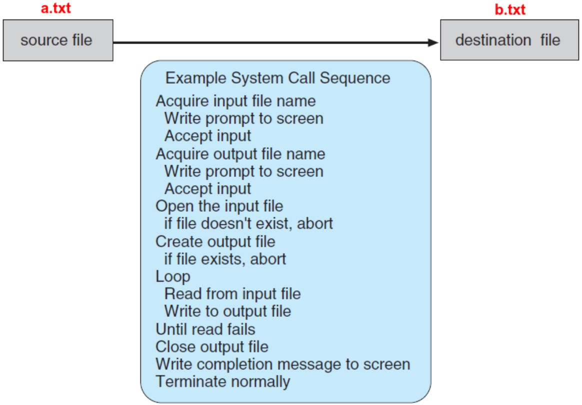
- Three most common APIs:
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Windows API for Windows
  - Java API for the Java virtual machine (JVM)

API and System Calls (1/4)

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

API and System Calls (2/4)

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



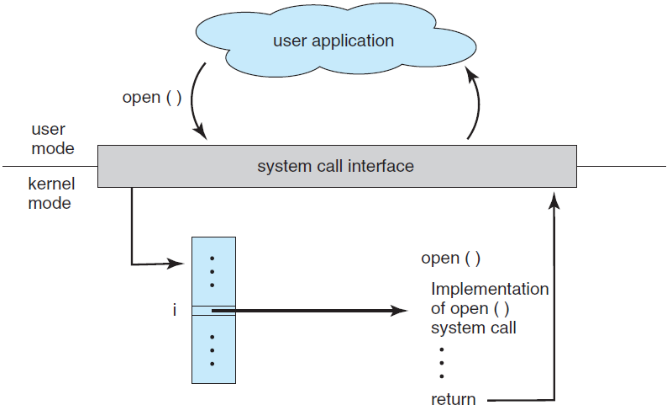
Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

API and System Calls (3/4)

```
> 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
...
```

API and System Calls (4/4)

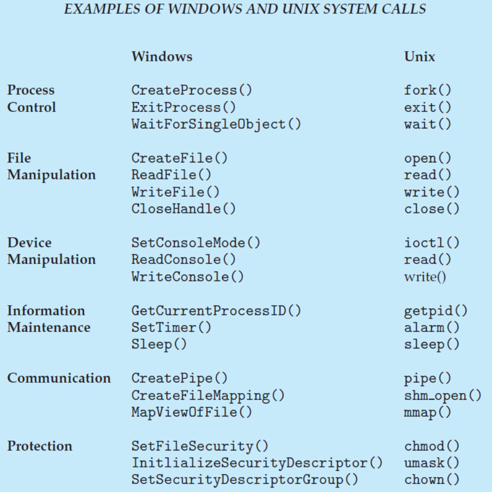


Types of System Calls (1/2)

- 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

Types of System Calls (2/2)



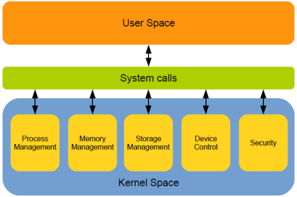
EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

Summary

Summary

- Computer-system organization: CPU, I/O devices, interrupt
- Operating-system structure: user-space, system calls, kernel-space
- Splitting the kernel:



Questions?