

# CS 370: OPERATING SYSTEMS

## [INTRODUCTION]

Shrideep Pallickara  
Computer Science  
Colorado State University

COMPUTER SCIENCE DEPARTMENT



1

## Frequently asked questions from the previous class survey

- Kernel and the OS: *Nota bene* the terms can be used interchangeably
  - Useful analogy: engine (kernel) and car (OS)
- How does the OS talk to the hardware? Does an OS need to support all hardware?
- How does the OS balance so many things simultaneously?
- What's a floppy disk?
- Who comes up with abstractions?
- Can we use GenAI pseudo code? NO!!!
- How does a OS handle apps built for other OS?
- How do applications inherit from the OS?



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.2

2

## Topics covered in this lecture

- A (very) brief history of Operating Systems
- CPU, Caches and main memory
- Secondary storage
- Relative speeds of the memory hierarchy
- The Kernel Abstraction



COLORADO STATE UNIVERSITY


Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.3

3

↕ Status	↕ Name 🔍	↕ Total Submissions	↕ Last Submission Status	↕ Last Submission Date
<input checked="" type="checkbox"/>	C01 - Hello World	98	✔	1/22/2025, 11:35:36 PM
<input checked="" type="checkbox"/>	C02 - Sum	17	✔	1/22/2025, 4:59:45 PM
<input checked="" type="checkbox"/>	C03 - Even Odd	20	✔	1/22/2025, 8:33:26 PM
<input checked="" type="checkbox"/>	C04 - Largest	14	✔	1/22/2025, 8:49:55 PM
<input checked="" type="checkbox"/>	C05 - Multiplication	11	✔	1/22/2025, 8:03:14 PM
<input checked="" type="checkbox"/>	C06 - Factorial	14	✔	1/22/2025, 8:33:28 PM
<input checked="" type="checkbox"/>	C07 - Reverse	11	✔	1/22/2025, 7:30:28 PM
<input checked="" type="checkbox"/>	C08 - Vowels	3	✔	1/21/2025, 10:42:54 PM
<input checked="" type="checkbox"/>	C09 - Primes	3	✔	1/21/2025, 11:00:43 PM
<input checked="" type="checkbox"/>	C10 - Sort	5	✔	1/21/2025, 11:46:08 PM
<input checked="" type="checkbox"/>	C11 - Pointers	6	✔	1/22/2025, 11:15:16 AM
<input checked="" type="checkbox"/>	C12 - Matrix	5	✔	1/22/2025, 4:48:44 PM
<input checked="" type="checkbox"/>	C++01 - Calculator	6	✔	1/22/2025, 5:50:36 PM
<input checked="" type="checkbox"/>	C++02 - Stats	7	✔	1/22/2025, 7:19:05 PM
<input checked="" type="checkbox"/>	C++03 - Bank	2	✔	1/17/2025, 10:43:49 AM
<input checked="" type="checkbox"/>	HW1: Memory Allocations/deallocations and Avoiding Resource Leaks	17	✔	1/22/2025, 7:49:41 PM

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

4

## A (VERY) BRIEF HISTORY OF OPERATING SYSTEMS

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

5

The first true digital computer was designed by Charles Babbage (1792-1871)

- Spent most of his life and fortune trying to build the analytical engine
- Never got it working properly
  - ▣ Purely mechanical
  - ▣ Technology of the day could not produce wheels, cogs, gears to the required precision
- Did not have an operating system



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.6

6

## Babbage realized he would need software for his analytical engine

- Hired Ada Lovelace as the worlds first programmer
  - ▣ Daughter of British poet Lord Byron
- The programming language Ada<sup>®</sup> is named after her



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.7

7

## The First Generation (1945-55) Vacuum Tubes

- First fully functioning digital computer built at Iowa State University
  - ▣ Prof. John Atanasoff and grad student Clifford Berry
- All programming in absolute machine language
  - ▣ Also, by wiring up electrical circuits
    - Connect 1000s of cables to plug boards to control machine's basic functions
  - ▣ Operating Systems were unheard of
- Straightforward numerical calculations
  - ▣ Produce tables of sines, cosines, logarithms



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.8

8

## The Second Generation (1955-1965): Transistors and Batch Systems

- **Separation** between designers, builders, operators, programmers, and maintenance
- Machines were called **mainframes**
- Write a program on paper, then punch it on cards
  - Give card deck to operator and go drink coffee
  - Operator gives output to programmer



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.9

9

## The Third Generation (1965-1980) ICs and Multiprogramming

- Managing different product lines was expensive for manufacturers
  - Customers would start with a small machine, and then outgrow it
- IBM introduced the Systems/360
  - Series of **software-compatible** machines
  - All machines had the same instruction set
    - Programs written for one machine could run on all machines



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.10

10

## The Fourth Generation (1980-Present) Personal Computers

- Large Scale Integration circuits (LSI)
  - ▣ Thousands of transistors on a square centimeter of silicon
- 1974: Intel came out with the 8080
  - ▣ General purpose 8-bit CPU
- Early 1980s IBM designed the IBM PC
  - ▣ Looked for an OS to run on the PC
  - ▣ Microsoft purchased Disk Operating System and went back to IBM with MS-DOS



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.11

11

## Over the past 50 years ...

- The most striking aspect has been Moore's Law and comparable advances in related technologies, such as memory and disk storage
- The cost of processing and memory has decreased by almost  $10^6$  over this period; the cost of disk capacity has decreased by  $10^7$ 
  - ▣ Disk latency has improved, but at a much slower rate than disk capacity
- These relative changes have **radically altered** both the use of computers and the **tradeoffs** faced by operating system designers



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.12

12

## Operating systems tend to be huge, complex and long-lived

- Source code of an OS like Linux or Windows?
  - ▣ Order of 5 million lines of code (for kernel)
    - 50 lines page, 1000 pages/volume = 100 volumes
- Application programs such as GUI, libraries and application software?
  - ▣ 10-20 times that

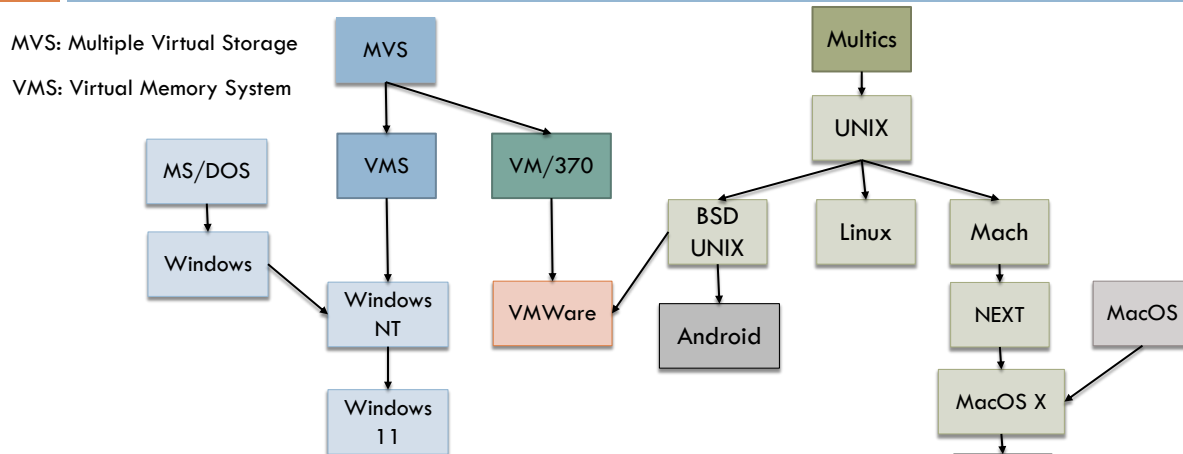


## Why do operating systems live for a long time?

- Hard to write and folks are loath to throw it out
- Typically **evolve** over long periods of time
  - ▣ Windows 95/98/Me is one OS
  - ▣ Windows NT/2000/XP/Vista/7/8/10/11 is another
  - ▣ System V, Solaris, BSD derived from original UNIX
  - ▣ Linux is a fresh code base
    - Closely modeled on UNIX and highly compatible with it
  - ▣ Apple OS X based on XNU (X is not Unix) which is based on the Mach microkernel and BSD's POSIX API



## Genealogy of modern operating systems



COLORADO STATE UNIVERSITY

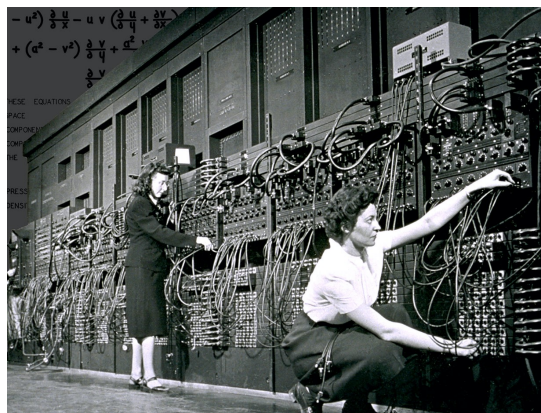
Professor: SHRIDEEP PALICKARA  
 COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.15

15

## Modern computer systems design

- Has its origins dating back to 1947 as part of work on ENIAC
- Breaks up a computing machine into three main subsystems:
  - ▣ The central processing unit (**CPU**) for performing arithmetic operations
  - ▣ The **memory** for storage of instructions and data
  - ▣ The input-output (**I/O**) unit for communicating with the external world
- This way of organizing a machine became known as the “**von Neumann architecture**”



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
 COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.16

16



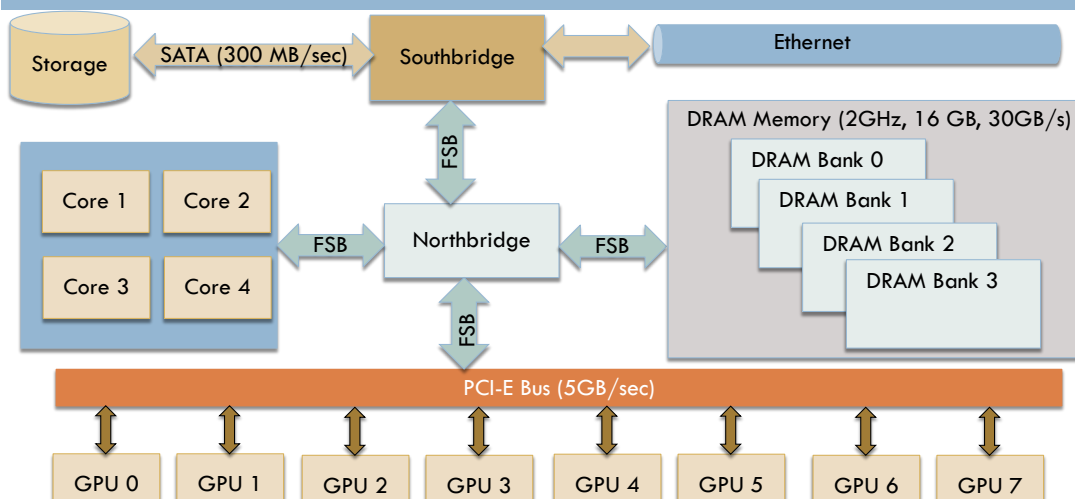
## COMPONENTS OF A COMPUTER

COMPUTER SCIENCE DEPARTMENT



17

### Typical PC architecture



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.18

18

## Processors

- **Brain** of the computer
- Each CPU has a specific set of instructions that it can execute
  - Pentium cannot execute SPARC and vice versa

\* The instruction set architecture (ISA) is how software interacts with the hardware.



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.19

19

## Rationale for registers inside the CPU

- Accessing memory to get instruction or data
  - **Much longer** than executing the instruction
- Registers hold:
  - Key variables
  - Temporary results



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.20

20

## What the instruction set looks like

- Load a word from memory into register
  - ▣ And, from register into memory
- Combine two (or more) operands from register, memory, or both into a result
  - ▣ E.g., add two words and store result in a register or in memory



## Besides the registers to hold variable and temporary results there are special registers

- **Program Counter**
  - ▣ Contains the memory address of the next instruction to be fetched
- **Stack pointer**
  - ▣ Points to the top of the current stack in memory
- Program Status Word
  - ▣ Stores condition code bits and other control code bits
  - ▣ Plays an important role in system calls and I/O



## Memory hierarchy: Cache memory

- Mostly *controlled by hardware*
- Main memory divided up into **cache lines**
  - Usually 64 bytes
  - Addresses 0-63 in cache line 1, 64-127 in cache line 2, and so on
- Most heavily used cache lines are stored in a high-speed cache



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.23

23

## When a program needs to read a memory word

- Cache hardware checks if the needed line is in the cache
- If it is, that's a **cache hit**
  - Request satisfied from cache in about *2 clock cycles*
  - No memory access needed
- If needed line is not present in cache
  - **Cache miss**, and must access memory
  - **Substantial** time penalty



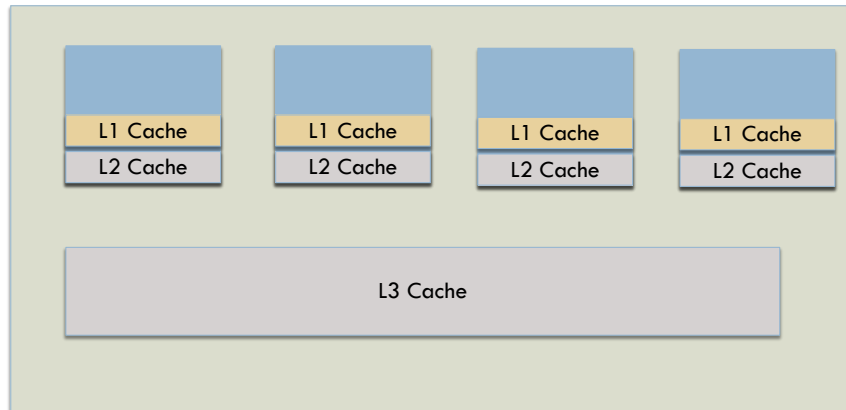
COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.24

24

## Logical view of the Processor, Cores, and Caches



L1: Typically in the order of 16 KB

L2, L3: Typically in the order of MBs

25

## Caching is a powerful concept used elsewhere too. Let's see when ...

- ① Large resource *can be divided* into pieces
  - ② Some pieces *used more heavily* than others
- OS caching examples:
    - ▣ Pieces of heavily used files in main memory
      - Reduce disk accesses
    - ▣ Conversion of file names to disk addresses
    - ▣ Addresses of Web pages (URLs) as hosts



26

## Main Memory

- Usually called **RAM** (Random Access Memory)
- Cache misses go to the main memory
- **Volatile**
  - Contents lost when power is turned off
- Memory size is in the order of several GB in most modern desktops



## Memory

- Ideally the memory should be
  - Extremely **fast**: Faster than executing an instruction
    - CPU should not be held up by the memory
  - Abundantly **large**
  - Dirt **cheap**
- No current technology satisfies all these goals



## Computers run most of their programs from (rewriteable) main memory

- Typically implemented in a technology called DRAM (dynamic random access memory)
- Ideal Scenario: Programs and data reside permanently in main memory. BUT ...
  - Space is *limited*
  - Main memory is *volatile* storage



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.29

29

## Secondary storage is needed to hold large quantities of data permanently

- Programs use the disk as the source and destination of processing
- Seek time 7 ms
- SPIN: 7200 – 15000 RPM
- Transfer rate
  - Disk-to-buffer: 70 MB/sec (SATA)
  - Buffer-to-Computer: 300 MB/sec
- Mean time between failures
  - 600,000 hours
- 1 TB capacity for less than \$75



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.30

30

## Improvements in hard disk capacity

- 1980 - 5 MB
- 1991 - 100 MB
- 1995 - 2 GB
- 1997 - 10 GB



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.31

31

## Improvements in hard disk capacity

- 2002 - 128 GB addressing space barrier [28 bits]
  - Old IDE/ATA interface: 28-bit addressing
  - $2^{28} \times 512 = 2^{28} \times 2^9 = 2^{37} = 128 \text{ GB} = 137,438,953,472 \text{ bytes}$
- 2003 – Serial ATA introduced
  - Bus interface providing connections to mass storage devices
- 2005 – 500 GB hard drives
- 2008 – 1 TB hard drives



COLORADO STATE UNIVERSITY

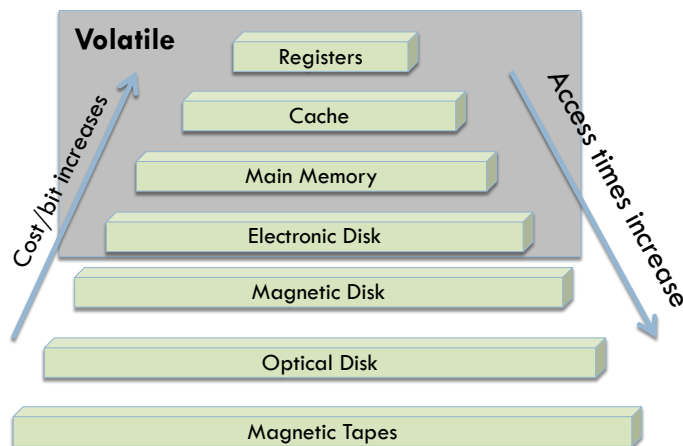
Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.32

32



## Storage system hierarchy based on speed, cost, size and volatility



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
 COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.33

33

## Characteristics of peripheral devices & their speed relative to the CPU

Item	time	Scaled time in human terms (2 billion times slower)
Processor cycle	0.5 ns (2 GHz)	1 second
Cache access	1 ns (1 GHz)	2 seconds
Memory access	70 ns	140 seconds
Context switch	5,000 ns (5 $\mu$ s)	167 minutes
Disk access	7,000,000 ns (7 ms)	162 days
Quantum	100,000,000 ns (100 ms)	6.3 years



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
 COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.34

34

## Mechanical nature of disks limits their performance

- Disk access times *have not* decreased exponentially
  - Processor speeds are growing *exponentially*
- Disparity between processor and disk access times continues to grow
  - 1:14,000,000



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.35

35

## RELATIVE SPEEDS OF THE MEMORY HIERARCHY

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

36

Since caches have limited size, cache management is critical

Level	1	2	3	4
Name	registers	cache	Main memory	Disk Storage
Typical Size	< 1 KB	< 16 MB	< 64 GB	> 1 TB
Implementation Technology	Custom memory, CMOS	On/off chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access times	0.25 ns	0.5-25 ns	80-250 ns	> 5 ms
Bandwidth (MB/sec)	20,000 – 100,000	5000-10,000	1000-5000	80-300
<b>Managed by</b>	compiler	hardware	OS	OS
Backed by	cache	Main memory	Disk	CD/Tape



COLORADO

KERNELS L2.37

37

## ONTOGENY RECAPITULATES PHYLOGENY

COMPUTER SCIENCE DEPARTMENT

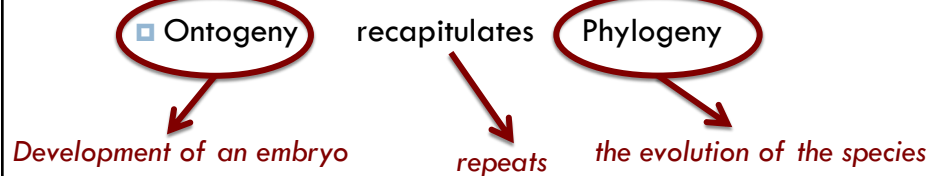


COLORADO STATE UNIVERSITY

38

## After Charles Darwin's book ON THE ORIGIN OF SPECIES was published

- German zoologist Ernst Haeckl stated



- i.e. human egg goes through stages of being a fish, ... , before becoming human baby
- Modern biologists think this is a gross simplification!



## Something vaguely similar has happened in the computer industry

- Each new species (*type of computer*) goes through the development its ancestors did
  - Both in hardware and software
  - Mainframe, mini computers, PC, handheld, etc



## Much of what happens in computing and other fields is technology driven

- Ancient Romans lacked cars not because they liked walking
  - ▣ It is because they didn't know to build cars
- PCs exist not because people have a centuries-old pent-up desire to own one
  - ▣ It is now possible to manufacture them cheaply



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.41

41

## Technology affects our view of systems

- A **change** in technology renders some idea *obsolete*
  - ▣ Another change could *revive* it
- Especially true when change has to do with **relative performance**
  - ▣ Of different parts of the system



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.42

42

## Let's look at this **relative** performance

- When CPUs become faster than memories?
  - ▣ Caches become important to speed-up slow memory
- If new memory technology makes memories much faster than CPUs?
  - ▣ Caches will vanish!
- In biology extinction is forever
  - ▣ In computer science, it is sometimes only for a few years



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.43

43

## Historical developments

### Large Memories

- IBM 7090/7094 1959-1964
  - ▣ 128 KB of memory
  - ▣ Programmed in assembly language (even the OS)
  - ▣ With time FORTRAN/COBOL and assembly was dead
- PDP-1 had only 4096 18-bit words of memory
  - ▣ Assembly is back!
  - ▣ Over time memory increases, assembly is out
- Microcomputers in 1980s
  - ▣ 4 KB memory and assembly is back again



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.44

44

## Other places where such a cycle has gone on?

- Protection hardware
- Disks
- Virtual memory

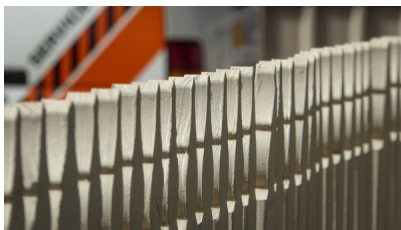


COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.45

45



Good fences make good neighbors  
*17th century proverb*

## THE KERNEL ABSTRACTION

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

46

## A central role of the OS is **protection** — the isolation of potentially misbehaving applications and users

- Implementing protection is the job of the OS **kernel**
- The kernel has full access to all of the machine hardware
  - ▣ The lowest level of software running on the system
  - ▣ Necessarily trusted to do anything with the hardware
- Everything other than the kernel — that is, the untrusted software running on the system — is run in a restricted environment
  - Less than complete access to the full power of the hardware



## What hardware is needed to let the kernel provide isolation?

- At a minimum, the hardware must support **three** things:
- **Privileged Instructions:** All potentially unsafe instructions are prohibited when executing in user mode
- **Memory Protection:** All memory accesses outside of a process's valid memory region are prohibited when executing in user mode
- **Timer Interrupts:** Regardless of what the process does, the kernel must have a way to periodically regain control from the current process





## Conceptually, the kernel/user mode is a one-bit register

- When set to 1, the processor is in kernel mode and can do anything
- When set to 0, the processor is in user mode and is restricted



## Privileged Instructions

- Instructions available in kernel mode, but not in user mode, are called **privileged instructions**
- To do its work, the kernel must be able to execute these instructions
  - Change privilege levels, adjust memory access, and disable and enable interrupts, set/reset timers
- If these instructions were available to applications?
  - A rogue application would in effect have the power of the kernel



## Making memory sharing safe

- The kernel must be able to configure the hardware so that each application process can read and write **only its own memory**
  - ▣ Not the memory of the operating system or any other application
- While it might seem that read-only access to memory is harmless, the OS needs to provide both security and privacy
  - ▣ For example, user passwords may be stored in kernel memory while they are being verified



## Hardware timers

- Timers can be set to **interrupt** the processor after a specified delay
  - ▣ Either in time or after some number of instructions have been executed
- Each timer interrupts one processor ... separate timer for each CPU
  - ▣ The kernel might set each timer to expire every few milliseconds
- Resetting the timer is a privileged operation
  - ▣ User-level process cannot inadvertently or maliciously disable the timer
- How does the kernel know if an application is in an infinite loop?
  - ▣ It doesn't!



## Mode transitions

- The kernel places a user process in a carefully constructed **sandbox**
  - The next question is how to safely transition from executing a user process to executing the kernel, and vice versa
- These transitions are **not rare events**
  - E.g.: A web server might switch between user mode and kernel mode thousands of times per second
- Transitions must be both fast and safe



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.53

53

## There are three reasons for the kernel to take control from a user process

- Reasons: interrupts, processor exceptions, and system calls
- Asynchronous events
  - **Interrupts** are triggered by an *external event* and can cause a transfer to kernel mode after any user-mode instruction
- Synchronous events
  - **Processor exceptions** and **system calls** are triggered by *process execution*
  - The term **trap** refers to any synchronous transfer of control from user mode to the kernel



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA  
COMPUTER SCIENCE DEPARTMENT

PROCESSORS, MEMORY HIERARCHY & KERNELS L2.54

54

## Interrupts are also used to inform the kernel of the completion of I/O requests

- Mouse device hardware triggers an interrupt every time the user moves or clicks on the mouse
  - ▣ The kernel, in turn, notifies the appropriate user process — the one the user was “mousing” across
- Virtually **every I/O device generates an interrupt** whenever some input arrives for the processor and whenever a request completes
  - ▣ E.g.: the Ethernet, WiFi, hard disk, thumb drive, keyboard, mouse, etc.



## As the processor executes instructions, it checks for whether an interrupt has arrived

- If so, it completes or stalls any instructions that are in progress
  - ▣ Instead of fetching the next instruction, the processor hardware saves the current execution state
  - ▣ Starts executing at a specially designated interrupt handler in the kernel



## Processor exceptions

- A processor exception is a **hardware event** caused by user program behavior that causes a transfer of control to the kernel
- A processor exception occurs whenever a process
  - Attempts to perform a privileged instruction
  - Accesses memory outside of its own memory region
  - Causes an arithmetic overflow. E.g., divide-by-zero
  - Accesses a word of memory with a non-aligned address
  - Attempts to write to read-only memory



## User processes can also transition into the kernel voluntarily

- To request that the kernel perform an operation on the user's behalf
- A **system call** is any procedure provided by the kernel that can be called from the user level
  - Examples include system calls to establish a connection to a web server, to send or receive packets over the network, to create or delete files, to read or write data into files, and to create a new user process



## To protect the kernel from misbehaving user programs

- It is key that the hardware transfers control on a system call to a pre-defined address
  - User processes cannot be allowed to jump to arbitrary places in the kernel
- The kernel handles the details of:
  - Checking and copying arguments
  - Performing the operation, and
  - Copying return values back into the process's memory



## System calls provide the illusion that the kernel is simply a set of library routines available to users

- Implementing system calls requires the operating system to define a **calling convention**
- Once the arguments are in the correct format, the user-level program can issue a system call by executing the trap instruction to transfer control to the kernel
- The kernel implement its system calls in a way that **protects itself** from all errors and attacks that might be launched
  - Extreme version of defensive programming: *always* assume that system call parameters are intentionally designed to be as malicious as possible!



## The contents of this slide-set are based on the following references

- *Thomas Anderson and Michael Dahlin. Operating Systems: Principles and Practice, 2<sup>nd</sup> Edition. Recursive Books. ISBN: 0985673524/978-0985673529. [Chapter 2]*
- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4<sup>th</sup> Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 1]*
- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9<sup>th</sup> edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330. [Chapter 1, 2]*
- *Kay Robbins & Steve Robbins. Unix Systems Programming, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2. [Chapter 1]*
- *Peter J. Denning and Matti Tedre. Computational Thinking. MIT Press. ISBN-13: 978-0262536561. [Chapter 3]*

