Staff

• The course will be taught by
  • Ardalan Amiri Sani (ardalan@uci.edu)
Staff

Teaching Assistants:
  Tiago Muck (tmuck@ics.uci.edu)
  Paul Kirth (pkirth@uci.edu)
  Jia Chen (jiac5@ics.uci.edu)

Readers:
  Hamid Tavakoli (seyedhat@uci.edu)
  Aditya Rajendra Joshi (joshiar@uci.edu)
Course logistics and details

• Course web page -
  • http://www.ics.uci.edu/~ics143

• Discussions – Fridays 10:00-10:50 a.m, 11-11:50 a.m, 12-12:50 p.m, ICS 174
  • Pop quizzes
Course logistics and details

• Textbook:
  Operating System Concepts -- Ninth Edition
  A. Silberschatz, P.B. Galvin, and G. Gagne
  (Eighth, Seventh, Sixth and Fifth editions, and Java Versions are fine as well).

• Alternate Books
  • Operating Systems: Principles and Practice, by T. Anderson and M. Dahlin (second edition)
  • Modern Operating Systems, by Tanenbaum (Third edition)
Course logistics and details

• Homeworks and Assignments
  • 4 written homeworks in the quarter
  • 1 programming assignment (knowledge of C++ or Java required).
    • Handed out at midterm; submit/demo during Finals Week
    • Multistep assignment – don’t start in last week of classes!!!
  • Late homeworks will not be accepted.
  • All submissions will be made using the EEE Dropbox for the course

• Tests
  • Midterm – tentatively Thursday, Week 6 in class
  • Final Exam – per UCI course catalog
Grading Policy

• Homeworaks - 30%
  • 4 written homework each worth 5% of the final grade.
  • 1 programming assignment worth 10% of the final grade

• Midterm - 30% of the final grade

• Final exam - 40% of the final grade

• Final assignment of grades will be based on a curve.
Lecture Schedule

• Week 1
  • Introduction to Operating Systems, Computer System Structures, Operating System Structures

• Week 2
  • Processes and Threads

• Week 3
  • Processes and Threads, and CPU Scheduling

• Week 4
  • Scheduling

• Week 5
  • Process Synchronization
Course Schedule

• Week 6
  • Deadlocks, Midterm review and exam

• Week 7
  • Memory Management

• Week 8
  • Memory Management, Virtual Memory

• Week 9
  • File Systems Interface and Implementation

• Week 10
  • I/O Subsystems
Office hours

- Instructor
  - (Tentative) Tuesday 6:45pm-8:15pm (Location TBD)
- TA’s
  - Thursday 3pm-4:30pm (Location: ICS 424A)
Piazza

- [https://piazza.com/uci/spring2017/ics143a](https://piazza.com/uci/spring2017/ics143a)
- Post questions here.
Overview

• What is an operating system?
• Operating systems history
• Computer system and operating system structure
What is an Operating System?
What is an Operating System?

• OS is the software that acts as an intermediary between the user applications and computer hardware.
Computer System Components

• Hardware
  • Provides basic computing resources (CPU, memory, I/O devices).

• Operating System
  • Controls and coordinates the use of hardware among application programs.

• Application Programs
  • Solve computing problems of users (compilers, database systems, video games, business programs such as banking software).

• Users
  • People, machines, other computers
Abstract View of System

User 1

User 2

User 3

... User n

compiler

assembler

Text editor

Database system

Application Programs

Operating System

Computer Hardware
Operating system roles

• Referee
  • Resource allocation among users, applications
  • Isolation of different users, applications from each other
  • Communication between users, applications
Operating system roles

• Illusionist
  • Each application appears to have the entire machine to itself
  • Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport
Operating system roles

• Glue
  • Libraries, user interface widgets, …
  • Reduces cost of developing software
Example: file systems

• Referee
  • Prevent users from accessing each other’s files without permission

• Illusionist
  • Files can grow (nearly) arbitrarily large
  • Files persist even when the machine crashes in the middle of a save

• Glue
  • Named directories, printf, …
OS challenges
OS challenges

• Reliability
  • Does the system do what it was designed to do?
OS challenges

- Availability
  - What portion of the time is the system working?
  - Mean Time To Failure (MTTF), Mean Time to Repair
OS challenges

• Security
  • Can the system be compromised by an attacker?
OS challenges

• Privacy
  • Data is accessible only to authorized users
OS challenges

• Performance
  • Latency/response time
    • How long does an operation take to complete?
  • Throughput
    • How many operations can be done per unit of time?
• Overhead
  • How much extra work is done by the OS?
• Fairness
  • How equal is the performance received by different users?
• Predictability
  • How consistent is the performance over time?
OS challenges

• Portability
  • For programs:
    • Application programming interface (API)
  • For the kernel
    • Hardware abstraction layer
OS needs to keep pace with hardware improvements

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniprocessor speed (MIPS)</td>
<td>1</td>
<td>200</td>
<td>2500</td>
<td>2.5K</td>
</tr>
<tr>
<td>CPUs per computer</td>
<td>1</td>
<td>1</td>
<td>10+</td>
<td>10+</td>
</tr>
<tr>
<td>Processor MIPS/$</td>
<td>$100K</td>
<td>$25</td>
<td>$0.20</td>
<td>500K</td>
</tr>
<tr>
<td>DRAM Capacity (MiB)/$</td>
<td>0.002</td>
<td>2</td>
<td>1K</td>
<td>500K</td>
</tr>
<tr>
<td>Disk Capacity (GiB)/$</td>
<td>0.003</td>
<td>7</td>
<td>25K</td>
<td>10M</td>
</tr>
<tr>
<td>Home Internet</td>
<td>300 bps</td>
<td>256 Kbps</td>
<td>20 Mbps</td>
<td>100K</td>
</tr>
<tr>
<td>Machine room network (shared)</td>
<td>10 Mbps</td>
<td>100 Mbps</td>
<td>10 Gbps</td>
<td>1000</td>
</tr>
<tr>
<td>Machine room network (switched)</td>
<td>1:1</td>
<td>1:several</td>
<td>100:1</td>
<td>100+</td>
</tr>
</tbody>
</table>
Why should I study Operating Systems?
Why should I study Operating Systems?

• Need to understand interaction between the hardware and software
• Need to understand basic principles in the design of computer systems
  • efficient resource management, security, flexibility
Why should I study Operating Systems?

• Because it enables you to do things that are difficult/impossible otherwise.
Example: Rio: I/O sharing implemented in the operating system kernel
Observation: I/O devices important for personal computers
A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor
A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera

- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor
A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera

- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

sensing
A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS

4G LTE
NFC
WiFi
Bluetooth
Infrared
64 GB internal storage (extended by microSD)

Adreno 330 GPU
Hexagon DSP
Multimedia processor
A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)

- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor
Multiple computers for unique I/O
Multiple computers for unique I/O
Multiple computers for unique I/O
I/O sharing
How to build this?
Application layer

Client

- IP Webcam
- Wi-Fi Speaker
- MightyText

Server

- Application
- Daemons, Libraries
- Device driver
- I/O device

Application
- Daemons, Libraries
- Device driver
- I/O device
Do not meet our criteria

- High engineering effort
- No support for legacy applications
- No support for all I/O device features
Rio: I/O servers for sharing I/O between mobile systems

Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong, "Rio: A System Solution for Sharing I/O between Mobile Systems," in Proc. ACM MobiSys, June 2014. (Best Paper Award)
Key idea: device files as the boundary

I/O devices abstracted as (device) files in Unix-like OSes
e.g., /dev/foo
Key idea: device files as the boundary
Key idea: device files as the boundary

Client

I/O device

File operations

User space

Kernel

Application

Server

Device file
/dev/foo

Device driver

I/O device

User space

Kernel
Key idea: device files as the boundary

Application

File operations

Virtual device file
/dev/foo

User space
Kernel

Device file
/dev/foo

Device driver

I/O device

User space
Kernel

Client

Server
Key idea: device files as the boundary

Client

Server

Application

File operations

User space

Kernel

Virtual device file
/dev/foo

Stub

Wireless Link

Device file
/dev/foo

Device driver

I/O device
Video demo of Rio

http://www.ruf.rice.edu/~mobile/rio.html
Operating systems are everywhere
Operating systems are everywhere
Operating systems are everywhere
Operating systems are everywhere
Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure
Operating systems history

• Early Operating Systems
• Simple Batch Systems
• Multiprogrammed Batch Systems
• Time-sharing Systems
• Personal and mobile Computer Systems
People-to-Computer Ratio Over Time

From David Culler (Berkeley)

- Number Crunching Data Storage
- Productivity Interactive
- Streaming Information To/From Physical World

Log (people per computer) vs. Year

Time
Early Systems - Bare Machine (1950s)

**Hardware – expensive ; Human – cheap**

- **Structure**
  - Large machines run from console
  - Single user system
    - Programmer/User as operator
    - Paper tape or punched cards
- **Early software**
  - Assemblers, compilers, linkers, loaders, device drivers, libraries of common subroutines.
- **Secure execution**
- **Inefficient use of expensive resources**
  - Low CPU utilization, high setup time.
Batch Systems (1960’s)

• Reduce setup time by batching jobs with similar requirements.
• Hire an operator
  • User is NOT the operator
• Automatic job sequencing
  • Forms a rudimentary OS.
• Resident Monitor
  • Holds initial control, control transfers to job
    and then back to monitor.
• Problem
  • Need to distinguish job from job and data from program.
  • Special cards indicate what to do.
  • User program prevented from performing I/O
Batch Systems (1960’s)

- Solutions to speed up I/O:
  - Offline Processing
    - load jobs into memory from tapes, card reading and line printing are done offline.
    - User submits card deck
    - cards put on tape
    - tape processed by operator
    - output written to tape
    - tape printed on printer
    - Separate user from computer

- Problems
  - Long turnaround time - up to 2 DAYS!!!
  - Low CPU utilization
    - I/O and CPU could not overlap; slow mechanical devices.
Batch Systems (1960’s)

• Solutions to speed up I/O:
  • Spooling (Simultaneous Peripheral Operation On-Line)
    • Use disk (random access device) as large storage for reading as many input files as possible and storing output files until output devices are ready to accept them.
    • Allows overlap - I/O of one job with computation of another.
    • Introduces notion of a job pool that allows OS choose next job to run so as to increase CPU utilization.
Speeding up I/O: Direct Memory Access (DMA)

- Data moved directly between I/O devices and memory
- CPU can work on other tasks
Batch Systems - I/O completion

• How do we know that I/O is complete?
  • Polling:
    • Device sets a flag when it is busy.
    • Program tests the flag in a loop waiting for completion of I/O.
  • Interrupts:
    • On completion of I/O, device forces CPU to jump to a specific instruction address that contains the interrupt service routine.
    • After the interrupt has been processed, CPU returns to code it was executing prior to servicing the interrupt.
Multiprogramming

• Use interrupts to run multiple programs simultaneously
  • When a program performs I/O, instead of polling, execute another program till interrupt is received.
• Requires secure memory, I/O for each program.
• Requires intervention if program indefinite loops.
• Requires CPU scheduling to choose the next job to run.
Timesharing

Hardware – *getting cheaper*; Human – *getting expensive*

- Programs queued for execution in FIFO order.
- Like multiprogramming, but timer device interrupts after a quantum (timeslice).
  - Interrupted program is returned to end of FIFO
  - Next program is taken from head of FIFO
- Control card interpreter replaced by command language interpreter.
Timesharing (cont.)

• Interactive (action/response)
  • when OS finishes execution of one command, it seeks the next control statement from user.

• File systems
  • online filesystem is required for users to access data and code.

• Virtual memory
  • Job is swapped in and out of memory to disk.
Personal Computing Systems -
desktops

Hardware – cheap ; Human – expensive

• Single user systems, portable.
• I/O devices - keyboards, mice, display screens, small printers.
• Single user systems may not need advanced CPU utilization or protection features.
• Advantages:
  • user convenience, responsiveness, ubiquitous
Personal Computing Systems - Mobile and wearable Systems

- Single user, multiple computers
- Laptops
- Smartphones
- Tablets
- Smart glasses
- Smart watches

**Hardware – *very cheap* ; Human – *very expensive***
Overview

• What is an operating system?
• Operating systems history
• Computer system and operating system structure
Computer System & OS Structures

• Computer System Organization
• Process abstraction and hardware protection
• System call and OS services
• Storage architecture
• OS organization
• OS tasks
• Virtual Machines
Computer System Organization
CPU execution

Execution sequence:
- Fetch Instruction at PC
- Decode
- Execute (possibly using registers)
- Write results to registers/mem
- PC = Next Instruction(PC)
- Repeat

From Berkeley OS course
Computer System Organization
I/O devices

- I/O devices and the CPU execute concurrently.
- Each device controller is in charge of a particular device type
  - Each device controller has a local buffer. I/O is from the device to local buffer of controller
- CPU moves data from/to main memory to/from the local buffers
- Device controller interrupts CPU on completion of I/O
Interrupts

• Interrupt transfers control to the interrupt service routine
  • Interrupt Service Routine: Segments of code that determine action to be taken for interrupt.

• Determining the type of interrupt
  • Polling: same interrupt handler called for all interrupts, which then polls all devices to figure out the reason for the interrupt
  • Interrupt Vector Table: different interrupt handlers will be executed for different interrupts

<table>
<thead>
<tr>
<th>Interrupt Number</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0003h</td>
</tr>
<tr>
<td>1</td>
<td>000Bh</td>
</tr>
<tr>
<td>2</td>
<td>0013h</td>
</tr>
<tr>
<td>3</td>
<td>001Bh</td>
</tr>
<tr>
<td>4</td>
<td>0023h</td>
</tr>
<tr>
<td>5</td>
<td>002Bh</td>
</tr>
<tr>
<td>6</td>
<td>0033h</td>
</tr>
<tr>
<td>7</td>
<td>003Bh</td>
</tr>
<tr>
<td>8</td>
<td>0043h</td>
</tr>
<tr>
<td>9</td>
<td>004Bh</td>
</tr>
<tr>
<td>10</td>
<td>0053h</td>
</tr>
<tr>
<td>11</td>
<td>005Bh</td>
</tr>
<tr>
<td>12</td>
<td>0063h</td>
</tr>
<tr>
<td>13</td>
<td>006Bh</td>
</tr>
<tr>
<td>14</td>
<td>0073h</td>
</tr>
<tr>
<td>15</td>
<td>007Bh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt Number</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0083h</td>
</tr>
<tr>
<td>17</td>
<td>008Bh</td>
</tr>
<tr>
<td>18</td>
<td>0093h</td>
</tr>
<tr>
<td>19</td>
<td>009Bh</td>
</tr>
<tr>
<td>20</td>
<td>00A3h</td>
</tr>
<tr>
<td>21</td>
<td>00ABh</td>
</tr>
<tr>
<td>22</td>
<td>00B3h</td>
</tr>
<tr>
<td>23</td>
<td>00B8h</td>
</tr>
<tr>
<td>24</td>
<td>00C3h</td>
</tr>
<tr>
<td>25</td>
<td>00CBh</td>
</tr>
<tr>
<td>26</td>
<td>00D3h</td>
</tr>
<tr>
<td>27</td>
<td>00DBh</td>
</tr>
<tr>
<td>28</td>
<td>00E3h</td>
</tr>
<tr>
<td>29</td>
<td>00EBh</td>
</tr>
<tr>
<td>30</td>
<td>00F3h</td>
</tr>
<tr>
<td>31</td>
<td>00FBh</td>
</tr>
</tbody>
</table>
Interrupt handling

• OS preserves the state of the CPU
  • stores registers and the program counter (address of interrupted instruction).
• Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
Different types of I/O processing for programs

• Synchronous I/O
  • After I/O is requested, *wait* until I/O is done. Program will be idle.

• Asynchronous I/O
  • After I/O is requested, control returns to user program without waiting for I/O completion.
Direct Memory Access (DMA)

- Used for high speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
Process Abstraction
Process Abstraction

• Process: an \textit{instance} of a program, running with limited rights
Process Abstraction

• Process: an \textit{instance} of a program, running with limited rights
  – Thread: a sequence of instructions within a process
    • Potentially many threads per process (for now 1:1)
  – Address space: set of rights of a process
    • Memory that the process can access
    • Other permissions the process has (e.g., which system calls it can make, what files it can access)
How to limit process rights?
Hardware Protection

• Dual Mode Operation

• Memory Protection

• CPU Protection

• I/O Protection
Should a process be able to execute any instructions?
Should a process be able to execute any instructions?

- No
  - Can alter system configuration
  - Can access unauthorized memory
  - Can access unauthorized I/O
  - etc.
- How to prevent?
Dual-mode operation

• Provide hardware support to differentiate between at least two modes of operation:
  1. User mode -- execution done on behalf of a user.
  2. Kernel mode (monitor/supervisor/system mode) -- execution done on behalf of operating system.

• “Privileged” instructions are only executable in the kernel mode

• Executing privileged instructions in the user mode “traps” into the kernel mode
  • Trap is a software generated interrupt caused either by an error or a user request
Dual-mode operation (cont.)

- Mode bit added to computer hardware to indicate the current mode: kernel(0) or user(1).
- When an interrupt or trap occurs, hardware switches to kernel mode.
How to isolate memory access?
Process address space

- Address space ⇒ the set of accessible addresses + state associated with them:
  - For a 32-bit processor there are $2^{32} = 4$ billion addresses
Virtual Address

Virtual Addresses (Process Layout)

- Code
- Data
- Heap
- Stack

Physical Memory

- Code
- Data
- Heap
- Stack
Providing the Illusion of Separate Address Spaces

Translation Map 1

Proc 1
Virtual Address Space 1

Translation Map 2

Proc 2
Virtual Address Space 2

Load new Translation Map on Switch
Address translation and memory protection

- Processor
- Virtual address
- Translation
- Physical address
- Physical Memory
- Physical address
- Data
- Valid
- Invalid
- Raise Exception
Memory Protection

• Must provide memory protection at least for the interrupt vector and the interrupt service routines.
• When a process is running, only memory in that process address space must be accessible.
• When executing in kernel mode, the kernel has unrestricted access to all memory.
Memory Protection: base and bounds

- To provide memory protection, add two registers that determine the range of legal addresses a program may address.
  - Base Register - holds smallest legal physical memory address.
  - Limit register - contains the size of the range.
- Memory outside the defined range is protected.
The load instructions for the base and limit registers are privileged instructions.
CPU Protection

• How to prevent a process from executing indefinitely?
CPU Protection

- Timer - interrupts computer after specified period to ensure that OS maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches a value of 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
- Timer is also used to compute the current time.
- Load timer is a privileged instruction.
I/O Protection

• All I/O instructions are privileged instructions.

• Must ensure that a user program could never gain control of the computer in kernel mode, e.g., a user program must not be able to store a new address in the interrupt vector.
Question

• Given the I/O instructions are privileged, how do users perform I/O?
Question

• Given the I/O instructions are privileged, how do users perform I/O?

• Via system calls - the method used by a process to request action by the operating system.
System Calls

- User code can issue a syscall, which causes a trap
- Kernel handles the syscall
System Calls

- Interface between running program and the OS.
  - Assembly language instructions (macros and subroutines)
  - Some higher level languages allow system calls to be made directly (e.g. C)
- Passing parameters between a running program and OS via registers, memory tables or stack.
- Linux has about 300 system calls
  - read(), write(), open(), close(), fork(), exec(), ioctl(),.....
System services or system programs

• Convenient environment for program development and execution. User view of OS is defined by system services, not system calls.
  • Command Interpreter (sh, csh, ksh) - parses/executes other system programs
  • File manipulation - copy (cp), print (lpr), compare(cmp, diff)
  • File modification - editing (ed, vi, emacs)
  • Application programs - send mail (mail), read news (rn)
  • Programming language support (cc)
  • Status information, communication
  • etc….
Command Interpreter System

• Commands that are given to the operating system via command statements that execute

• Obtains the next command and executes it.

• Programs that read and interpret control statements also called -
  • Control card interpreter, command-line interpreter, shell (in UNIX)
Storage Structure

• Main memory - only large storage media that the CPU can access directly.

• Secondary storage - extension of main memory that has large nonvolatile storage capacity.
  • Magnetic disks - rigid metal or glass platters covered with magnetic recording material.
    • Disk surface is logically divided into tracks, subdivided into sectors.
    • Disk controller determines logical interaction between device and computer.
Storage Hierarchy

• Storage systems are organized in a hierarchy based on
  • Speed
  • Cost
  • Volatility

• Caching - process of copying information into faster storage system; main memory can be viewed as fast cache for secondary storage.
Storage Device Hierarchy

- registers
- cache
- main memory
- solid-state disk
- hard disk
- optical disk
- magnetic tapes
Operating Systems: How are they organized?

• Simple
  • Only one or two levels of code

• Layered
  • Lower levels independent of upper levels

• Modular
  • Core kernel with Dynamically loadable modules

• Microkernel
  • OS built from many user-level processes
OS Structure - Simple Approach

• MS-DOS - provides a lot of functionality in little space.
  • Not divided into modules, Interfaces and levels of functionality are not well separated
Original UNIX System Structure

• Limited structuring, has 2 separable parts
  • Systems programs
  • Kernel
    • everything below system call interface and above physical hardware.
    • Filesystem, CPU scheduling, memory management
Layered OS Structure

• OS divided into number of layers - bottom layer is hardware, highest layer is the user interface.

• Each layer uses functions and services of only lower-level layers.

• THE Operating System and Linux Kernel has successive layers of abstraction.

<table>
<thead>
<tr>
<th>User Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Primitives</td>
</tr>
<tr>
<td>Device Drivers and Schedulers</td>
</tr>
<tr>
<td>Virtual Memory</td>
</tr>
<tr>
<td>I/O</td>
</tr>
<tr>
<td>CPU Scheduling</td>
</tr>
<tr>
<td>Hardware</td>
</tr>
</tbody>
</table>
Layered Operating System
Modules-based Structure

- Most modern operating systems implement modules
  - 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 flexibility
  - Linux makes extensive use of modules
Monolithic vs. Microkernel OS

• Monolithic OSes have large kernels with a lot of components
  • Linux, Windows, Mac
• Microkernels moves as much from the kernel into “user” space
  • Small core OS running at kernel level
  • OS Services built from many independent user-level processes

• Communication between modules with message passing
• Benefits:
  • Easier to extend a microkernel
  • Easier to port OS to new architectures
  • More reliable (less code is running in kernel mode)
  • Fault Isolation (parts of kernel protected from other parts)
  • More secure

• Detriments:
  • Performance overhead severe for naïve implementation
A microkernel OS

Monolithic Kernel based Operating System

Monolithic Kernel based Operating System

Application

System Call

VFS

IPC, File System

Scheduler, Virtual Memory

Device Drivers, Dispatcher, ...

Hardware

Application IPC

UNIX Server

Device Driver

File Server

Basic IPC, Virtual Memory, Scheduling

Hardware

Slide adapted from http://web.cecs.pdx.edu/~walpole/class/cs533/fall2015/home.html
OS Task: Process Management

- Process - fundamental concept in OS
  - Process is an instance of a program in execution.
  - Process needs resources - CPU time, memory, files/data and I/O devices.

- OS is responsible for the following process management activities.
  - Process creation and deletion
  - Process suspension and resumption
  - Process synchronization and interprocess communication
  - Process interactions - deadlock detection, avoidance and correction
OS Task: Memory Management

• Main Memory is an array of addressable words or bytes that is quickly accessible.
• Main Memory is volatile.
• OS is responsible for:
  • Allocate and deallocate memory to processes.
  • Managing multiple processes within memory - keep track of which parts of memory are used by which processes. Manage the sharing of memory between processes.
  • Determining which processes to load when memory becomes available.
OS Task: Secondary Storage and I/O Management

• Since primary storage is expensive and volatile, secondary storage is required for backup.

• Disk is the primary form of secondary storage.
  • OS performs storage allocation, free-space management and disk scheduling.

• I/O system in the OS consists of
  • Buffer caching and management
  • Device driver interface that abstracts device details
  • Drivers for specific hardware devices
OS Task: File System Management

- File is a collection of related information defined by creator - represents programs and data.

- OS is responsible for
  - File creation and deletion
  - Directory creation and deletion
  - Supporting primitives for file/directory manipulation.
  - Mapping files to disks (secondary storage).
  - Backup files on archival media (tapes).
Protection mechanisms control access of programs and processes to user and system resources.

- Protect user from himself, user from other users, system from users.

Protection mechanisms must:

- Distinguish between authorized and unauthorized use.
- Specify access controls to be imposed on use.
- Provide mechanisms for enforcement of access control.
- Security mechanisms provide trust in system and privacy
  - authentication, certification, encryption etc.
OS Task: Networking

• Connecting processors in a distributed system
• Distributed System is a collection of processors that do not share memory or a clock.
• Processors are connected via a communication network.
• Advantages:
  • Allows users and system to exchange information
  • provide computational speedup
  • increased reliability and availability of information
Virtual Machines

**Physical Machine**

- Hardware
- OS
- Application
Virtual Machines

Virtual Machine 1
- Application
- OS

Virtual Machine 2
- Application
- OS

Virtual Machine 3
- Application
- OS

Virtual Machine Monitor (VMM) (aka Hypervisor)

Hardware
Virtual Machines

• Use cases
  • Resource configuration
  • Running multiple OSes, either the same or different OSes
  • Run existing OS binaries on different architecture
Summary of this week’s lecture

• What is an operating system?
• Operating systems history
• Computer system and operating system structure