# CSC 2/456: Operating Systems

Instructor: Sandhya Dwarkadas

TAs: Zhuojia Shen, Mohsen Mohammadi



## Prerequisites

- CSC 252 or equivalent
- C/C++ programming experience on Unix systems



## Meaning of Prerequisites

- You understand basic processor organization
- You are aware of virtual memory and its support
- You can read assembly language code
- You can write assembly language code
- You understand bit-wise operations (AND, OR, XOR)
- You can write C code that manipulates pointer-based data structures (e.g., linked lists, trees, etc.)



## General Course Information

- Course Web page:
  - www.cs.rochester.edu/u/sandhya/csc256
- Course-related announcement/correspondence:
  - Blackboard Discussion Board
- Texts
  - Tanenbaum et al, "Modern Operating Systems"
  - Silberschatz et al, "Operating System Concepts"

## Linux Kernel Books

- ***Understanding the Linux Kernel*** by Bovet and Cesati
- ***Professional Linux Kernel Architecture*** by Mauerer
- ***Linux Kernel Development*** by Love

5

## Unix API Books

- ***Advanced Programming in the UNIX Environment*** by Stevens and Rago
- Available online via the library

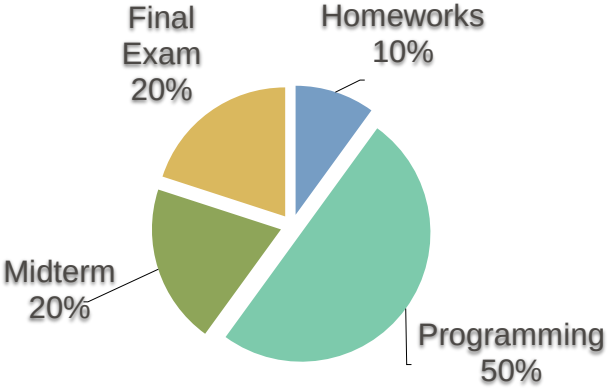
6

## Assignments and Exams

- Tentatively 5 programming assignments
  - Modify Linux kernel internals
- Midterm and final
- Homework/quizzes/other
  - Other: participation in class discussions, presentations
- "CSC456 Part" in assignments
- C programming
- Class presentation and end-of-term survey paper for CSC456 students

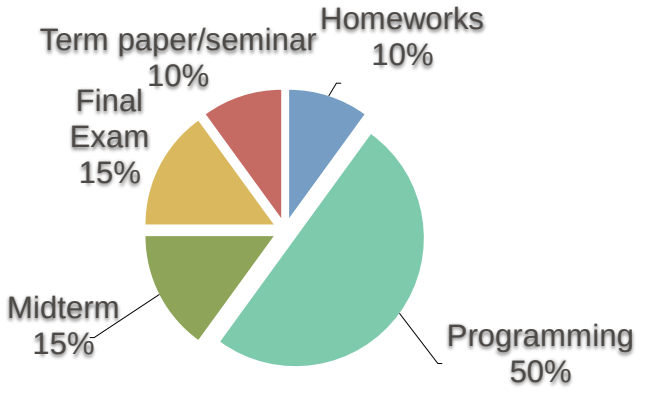
8/31/2018 CSC 2/456 7

## Grading for CSC 256




8

## Grading for CSC 456

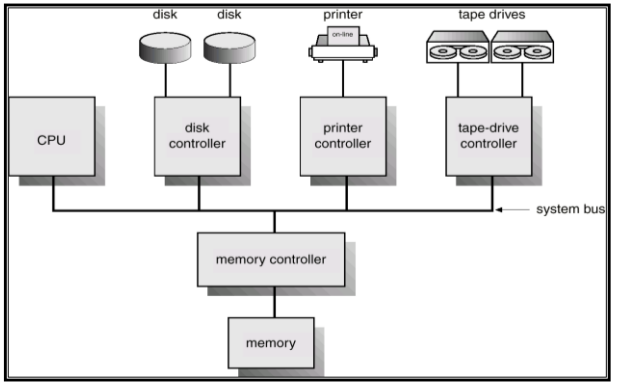



9

## If you don't have a Unix account, please get one ASAP!

10

## Computer-System Architecture




8/31/2018 CSC 2/456 11

## What is an Operating System?

- Software that abstracts the computer hardware
  - Hides the messy details of the underlying hardware
  - Presents users with a resource abstraction that is easy to use
  - Extends or virtualizes the underlying machine
- Manages the resources
  - Processors, memory, timers, disks, mice, network interfaces, printers, displays, …
  - Allows multiple users and programs to share the resources and coordinates the sharing

8/31/2018 CSC 2/456 12

## Why Study Operating Systems?

- Learn to design an OS or other computer systems
- Understand an OS
  - Understand the inner workings of an OS
  - Enable you to write efficient/correct application code



## System Calls and Interfaces/Abstractions

- Examples: Win32, POSIX, or Java APIs
- Process management
  - fork, waitpid, execve, exit, kill
- Exceptions, interrupts (events)
  - signals
- File management
  - open, close, read, write, lseek
- Directory and file system management
  - mkdir, rmdir, link, unlink, mount, umount
- Inter-process communication
  - sockets, pipe, ipc (msg, shm, sem)



## Reasons for Abstractions

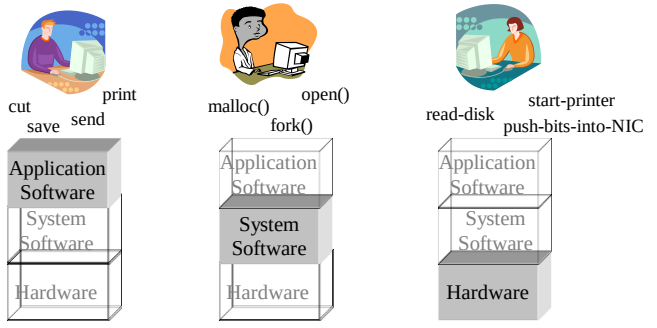
- Reduce functional complexity
- Provide single abstraction over multiple devices
- Resource sharing



## Objectives when Sharing Resources

- Efficiency
- Fairness
- Security/protection

## Perspectives of the Computer




(a) End User View

(b) Application Programmer View

(c) System Programmer View

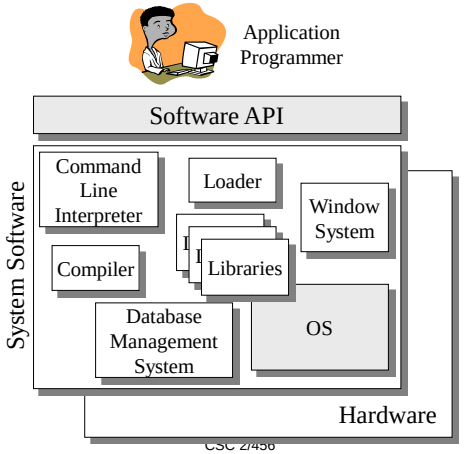


## System Software

- A general piece of software with common functionalities that support many applications
- Examples
  - C compiler and library functions
  - Shell – command line interpreter
  - A window system
  - A database management system
  - The operating system
    - A thin layer of software that operates directly on the raw hardware

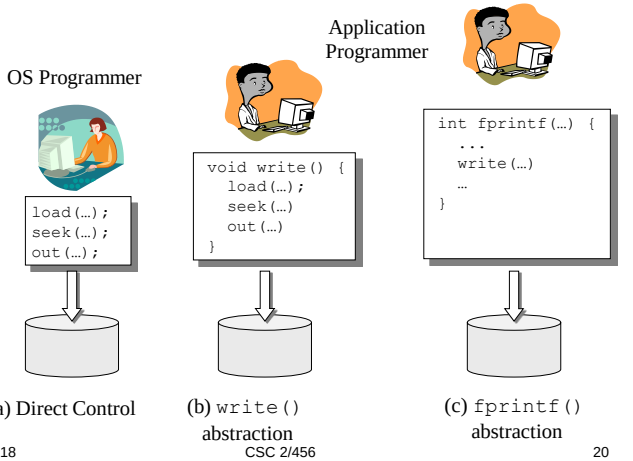


## The Structure of Computer Systems






## Disk Abstractions




(a) Direct Control

(b) `write()` abstraction

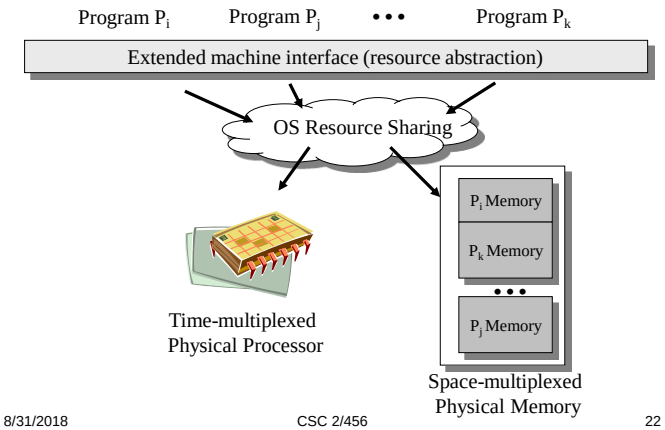
(c) `fprintf()` abstraction

## Under the Abstraction

- functional complexity
- a single abstraction over multiple devices
- replication → reliability
- resouce sharing



## Resource Sharing



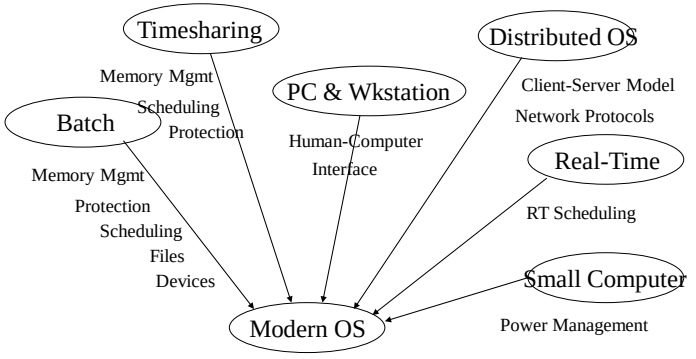



## Objectives of Resource Sharing

- Efficiency
- Fairness
- Security/protection



## Evolution of Modern OS



## History

- Machine language
- Batch systems (mainframes)
- Multiprogramming and time sharing
- Graphical user interfaces, virtual memory, protection, network/distributed operating systems



## Examples of Modern OSes

- UNIX variants (e.g., Mac OS X, FreeBSD, AIX, Solaris, Linux) -- have evolved since 1969
- Windows 78/10//NT/2K -- has evolved since 1989
- Smartphone Oses: Android, iOS, … (most based on Unix family)
- Other OSes –
  - microkernel
  - extensible OS
  - virtual machines
  - sensor OS
  - Software isolated processes
  - special-purpose OS – for highly concurrent Internet servers
  - still evolving …



## Operating Systems Concepts

- Processes
- Memory management
- File systems
- Device management
- Security/protection



## Processes

- Definition: A process is an instance of a ***running*** program.
- One of the most profound ideas in computer science
- Not the same as “program” or “processor”
- Program = Set of Instructions
  - May not be running
- Processor = Hardware that executes programs

## Processes

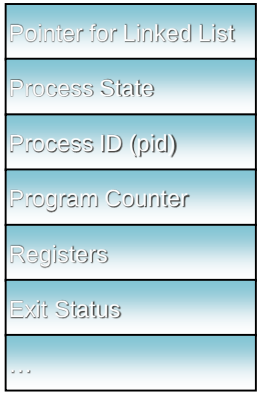
- Process provides each program with two key abstractions:
  - Logical control flow: each program seems to have exclusive use of the CPU
  - Private address space: each program seems to have exclusive use of main memory
- How are these illusions maintained?
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system

29

## Process Control Block (PCB)

OS data structure (in kernel memory) for maintaining information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- Information about open files




30

## Procfs: The /proc filesystem [Killian'84]

- Processes as files: a pseudo-file system
  - a file system interface to kernel in-memory data structures
- Linux implementation originated with Bell Labs' Plan 9
- Hierarchical file system
  - Each live process has its own directory (numbered with pid)
  - Non-process-related system information in named files: e.g., cpuinfo, meminfo

8/31/2018 CSC 2/456 31

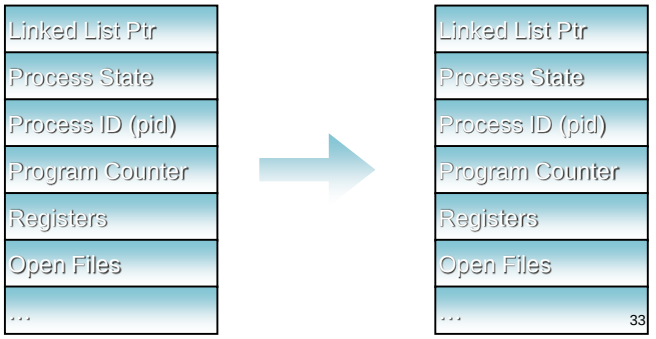
## Process Creation

- When a process (parent) creates a new process (child)
  - Execution sequence?
  - Address space sharing?
  - Open files inheritance?
  - … …
- UNIX examples
  - fork() system call creates new process with a duplicated copy of everything
  - exec() system call replaces process' memory space with a new program
    - Typically used after a call to fork
  - Child and parent compete for CPU like two normal processes

32

## The fork() system call

Duplicates a process




33

## Assignment #1

- Exclusively outside of the OS
- Part I: observing the OS through the /proc virtual file system
- Part II: building a shell (command-line interpreter)
  - Support foreground/background executions
  - Support pipes

8/31/2018 CSC 2/456 49

## Disclaimer

- Parts of the lecture slides contain original work from Gary Nutt, Andrew S. Tanenbaum, Dave O'Hallaron, Randal Bryant, and Kai Shen. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).

8/31/2018 CSC 2/456 51