CMSC 421 Section 0101
Fall 1999

Professor Ethan Miller
`elm@csee.umbc.edu`
`http://www.csee.umbc.edu/~elm/`

Course Information

- Course staff
  - » Professor: Dr. Ethan Miller (office in 225H ECS)
  - » Lecturer (Tuesday classes): Naomi Avigdor
  - » TA: Zhou Zhang
  - » Email: `{elm,navigd1,zzhang}@csee.umbc.edu`
- Office hours:
  - » Professor Miller: Thu 1-2, Fri 11-noon
  - » Zhou Zhang: Mon 4-5, Tue 4-5
- Web page:
  - » `http://www.csee.umbc.edu/courses/undergraduate/CMSC421/Fall99/0101/`
  - » Assignments, slides, and notes all available on Web page
  - » Check the Web page regularly!

Course Requirements

- Two exams
  - » Midterm (late October)
  - » Final exam
- Projects
  - » 3-4 projects during the semester
  - » ~ 3 weeks per project
  - » Will require lots of C programming
- Homework
  - » 6 homeworks during the semester
  - » 1 week per homework
  - » Graded, but individual homeworks not required to pass class
  - » Hand in online using `submit`

 1-3

Grading

- Final grades based on:
  - » Projects: 35% (distributed evenly across all projects)
  - » Homeworks: 17% (distributed evenly across all homeworks)
  - » Midterm: 20%
  - » Final: 25%
  - » Class participation: 3%
- Grade ranges:
  - » A: 89% - 100%
  - » B: 79% - 88%
  - » C: 69% - 78%
  - » D: 60% - 68%
- To pass the class, you ***must*** take both exams and hand in a reasonable attempt at all projects

 1-4

Project Information

- Write the core of an operating system
  - » Runs on simulated hardware (DLX emulator)
    - – Emulator runs on Linux
    - – Cross-compiler runs on Linux
  - » Implement
    - – Synchronization
    - – User-level processes
    - – Virtual memory
    - – File system
- Learn about operating system structures
- Work with a partner on a big project
  - » Grades for both people are the same...

Project Logistics

- For each project, hand in
  - » Detailed design description (due 1 week into project)
  - » Code files used to implement the project
- Use UMBC `submit` program
- Work may be done on campus or elsewhere
  - » Code must work on campus!
  - » Try out code before handing it in
- Projects done individually or in pairs

Class Outline

- Introduction and historical perspective
- Process Management, IPC & Threads
- Synchronization: semaphores and monitors, deadlocks
- Process Scheduling
- Address spaces, multiprogramming, and I/O
- Memory management, address translation, and virtual memory
- File systems & Secondary Storage
- Security and Cryptography
- Distributed systems

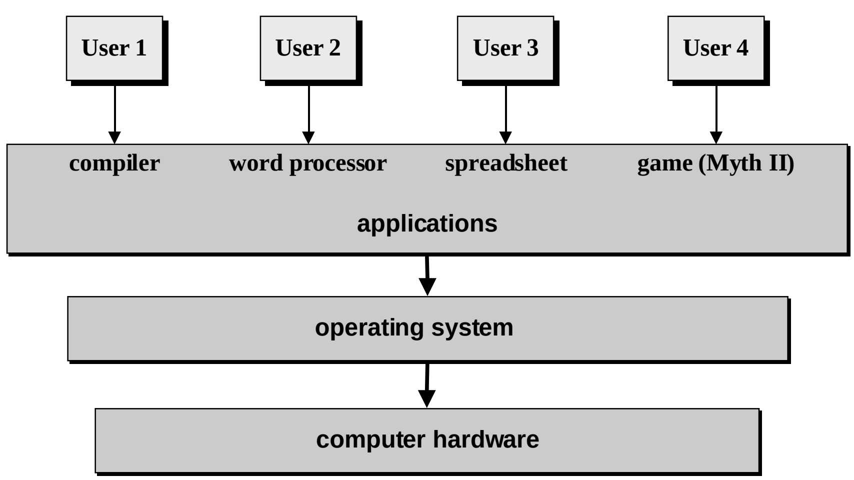
What's an Operating System?

- A program that runs on the "raw" hardware
  - » Acts as an intermediary between computer and users
  - » Standardizes the interface to the user across different types of hardware
- Operating system goals:
  - » Execute user programs
  - » Make the computer system easier to use
  - » Manage hardware resources
- Potentially conflicting goals:
  - » Use hardware efficiently
  - » Give maximum performance to users

Pieces of a Computer System

- Hardware: provides basic resources
  - » CPU
  - » Memory
  - » I/O devices (networks, disks, display, etc.)
- Operating system: controls and coordinates hardware usage
- Applications: allow users to solve specific problems
  - » Games
  - » “Office” apps (spreadsheets, databases,word processing,…)
  - » Development applications (compilers, etc.)
- (Users)
  - » People or machines that use the computer system

System Components



Operating System Terms

- Kernel
  - » The basic "program" that's always running
  - » Runs other (application) programs
- Resource
  - » Commodity to be allocated among applications & the operating system
  - » Operating system manages this allocation
- Multiprogramming
  - » The ability to run more than one job at a time
- Multitasking (time sharing)
  - » The ability to run multiple jobs and switch quickly between them
  - » Gives the illusion of having an entire computer to yourself

 1-11

Early Systems: Bare Machines

- Large machines run from consoles
  - » Single user system (no multiprogramming)
  - » Programmer was operator & user
  - » Programmed by punched tape or punch cards
- Early software
  - » Development tools (assemblers, later compilers)
  - » System tools (linkers & loaders)
  - » Software libraries
  - » Device drivers
- Secure
- Used hardware inefficiently
  - » Too much setup time per task
  - » CPU wasted while task waited for I/O

 1-12

Next Step : Simple Batch Systems

- Full-time operator
  - » Users didn't run the computer directly
  - » Operator batched similar jobs together
- Job sequencing
  - » Card reader could load in next job while current job running
  - » Control automatically transferred from one job to another
    - – First rudimentary operating system
- Full-time resident "operating system" code (monitor)
  - » Initial control when machine turned on
  - » Transfer control to job when loaded
  - » Return control to monitor when job finished

 1-13

Issues with Simple Batch Systems

- How does the monitor know job details?
  - » Fortran vs. assembly language?
  - » Which resident job to execute next?
- How does the monitor distinguish information
  - » End of one job from the start of another job?
  - » Job program from job data?
- Solution: control cards
  - » Special cards that describe the other cards
    - – $DATA, $JOB, $END, $FTN
  - » Special cards that provide instructions for the monitor
    - – $RUN
  - » Distinguished from "normal" cards with special characters in particular columns

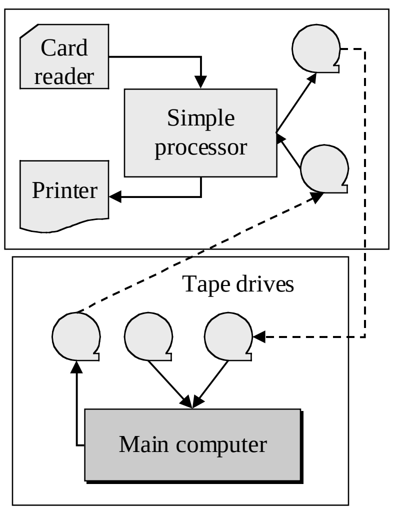
 1-14

Resident Monitor

- Program that runs other programs
  - » Control card interpreter : reads control cards and carries out their requests
  - » Loader : loads system programs and regular applications into memory
  - » Device drivers: know how to interface with particular devices on the system
- Problem: slow performance
  - » I/O and CPU can't overlap
  - » Card reader very slow
- Solution: offline operation
  - » Do all I/O to or from magnetic tapes (reasonably fast)
  - » Card reading and printing done from tapes offline

Tapes & Off-Line Operation

- Use simpler hardware to
  - » Read cards onto tape
  - » Read output from tape to printer
- Keep main computer free for actual data processing
- No changes to applications to allow off-line processing
- Real gains
  - » Utilize main computer more efficiently
  - » Multiple card readers & printers for a single computer

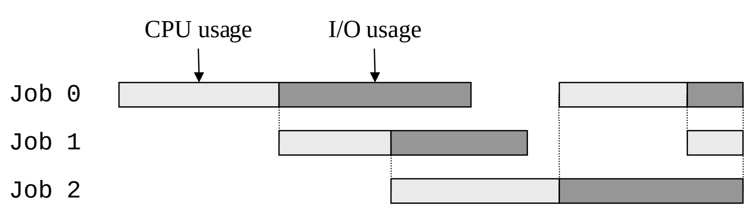


Spooling

- Simultaneous peripheral operation on-line
  - » Overlap computation of one job with the I/O for another job
  - » Write jobs onto disk while working on another
  - » Output result of previous job onto disk while working on another
- Keep a *job pool*
  - » Set of jobs on disk ready to run
  - » Allow CPU to select next job to run by scheduling algorithm
- As long as there are enough jobs, CPU will be utilized well

Multiprogrammed Batch Systems

- Keep several jobs in memory at once
  - » Multiplex CPU among them
  - » Allow one job to run while another is waiting for I/O
- Benefit: CPU never idle if there are enough jobs
  - » Better CPU utilization
  - » Better job turnaround



Features Needed for Multiprogramming

- I/O routines supplied by the operating system
  - » Manage the I/O resources between jobs
  - » Provide a standard interface to devices
- Memory management
  - » Allocate memory between jobs
  - » Prevent jobs from interfering with one another
- CPU scheduling
  - » Decide which job gets the CPU next
- I/O device reservation
  - » Allocate some devices (printer, etc.) to a particular job

Modern Time Sharing Systems

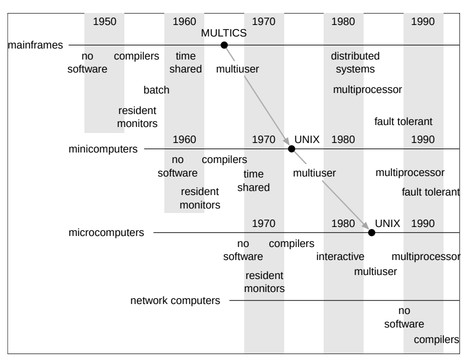
- CPU multiplexed among several jobs kept in memory
  - » Switching occurs rapidly (a few milliseconds per job)
  - » Jobs moved in and out of memory to keep active jobs available
- Operating system takes commands from users
  - » System executes user's job
  - » System requests new command from console
- File system gives users a place to store long-term data
- Result: system that gives users the illusion of having the entire machine
  - » Cost-effective: users don't need whole machine most of the time
  - » Allows resource sharing (only one printer needed…)

Personal Computers

- Computers cheap enough to put one (or several) on each person's desk (Macs, PCs)
  - » No need to time share the CPU?
- Design criteria
  - » Cost is very important - must be inexpensive
  - » Ease of use is crucial
  - » Efficiency not as important
- Techniques from time sharing systems may not be fully implemented in personal computers
  - » Memory protection
  - » Full job scheduling
- Advanced techniques becoming common in personal computer operating systems

Evolution of OS Features

- Computers have become cheaper over time
- Software does both more and less
  - » Early systems did everything
  - » Later systems more specialized
- Operating systems designed to meet specific needs of the computer



Operating Systems for Multicomputers

- Many computer systems have multiple CPUs
  - » Several CPUs in a single box (parallel computing)
  - » Several CPUs connected by networks (distributed computing)
- Operating systems have new duties
  - » Manage resources across several CPUs
  - » Move jobs from one CPU to another?
- Goal: make multiple CPUs as easy to use as a single CPU
  - » Create the illusion of a highly reliable, very fast single CPU
  - » Allow users to use any CPU without noticing any difference
  - » Balance the work across CPUs and other resources

Real-Time Operating Systems

- Some computers must respond in a particular time interval
  - » Dedicated application (robot, anti-lock brakes, airplane cockpit systems, medical appliances)
  - » Well-defined time constraints
- Two kinds of real-time systems
  - » "Hard" real-time systems
    - – System must respond in a fixed time
    - – Failure to do so means the system fails
    - – Use only ROM & semiconductor memory
  - » "Soft" real-time systems
    - – Some processes have higher priorities and should be done as quickly as possible
    - – Used for less time-critical applications (virtual reality, multimedia) where minor delays are OK

Modern Operating Systems

- Time sharing systems
  - » True time sharing (users protected from one another)
  - » Allow hundreds (or more) users per system
  - » Very complex: up to millions of lines of code
  - » UNIX (and derivatives)
  - » IBM MVS
  - » Windows NT
- Personal computers
  - » Memory protection recent (or not present)
  - » Multitasking
  - » Macintosh OS
  - » Windows 95/98
  - » Linux?