System Software for Program Execution: Introduction to Operating Systems

COMP 229 (Section PP) Week 5

Prof. Richard Zanibbi
Concordia University
January 30, 2006

Application Software and System Software

Application Software

Designed to solve a specific problem

e.g. inventory control, email, spreadsheets, games

System Software

A set of tools designed to simplify programming for a specific machine (or *hardware system*)

Concerned with using hardware efficiently

Used to construct and run Application Software ("applied" to create implementations of solutions (or "applications"))

e.g. **text editors**, assemblers, compilers, linkers, loaders, files, operating system (file system, networking, etc.)

A Note on Application Software

(Usually) Justifies Cost of a Computer

Why build it if it isn't useful?

Problems/needs tend to drive technological advances:

- Health (vaccination, pasteurization, insulin)
- Decryption (ENIGMA)
- War (unfortunately)
- Nuclear power
- Manufacturing costs
- Analyzing data using statistics
- Need for entertainment (e.g. building a "better Doom")

"System software and hardware exist to support the creation and effective use of application software"

System Software: Program Run-Time Environment

Operating System

Abstracts machine-level details (e.g. processes, storage, network, input/output devices, window/display management)

Program Loader

Loads programs into memory and starts their execution

Program Libraries

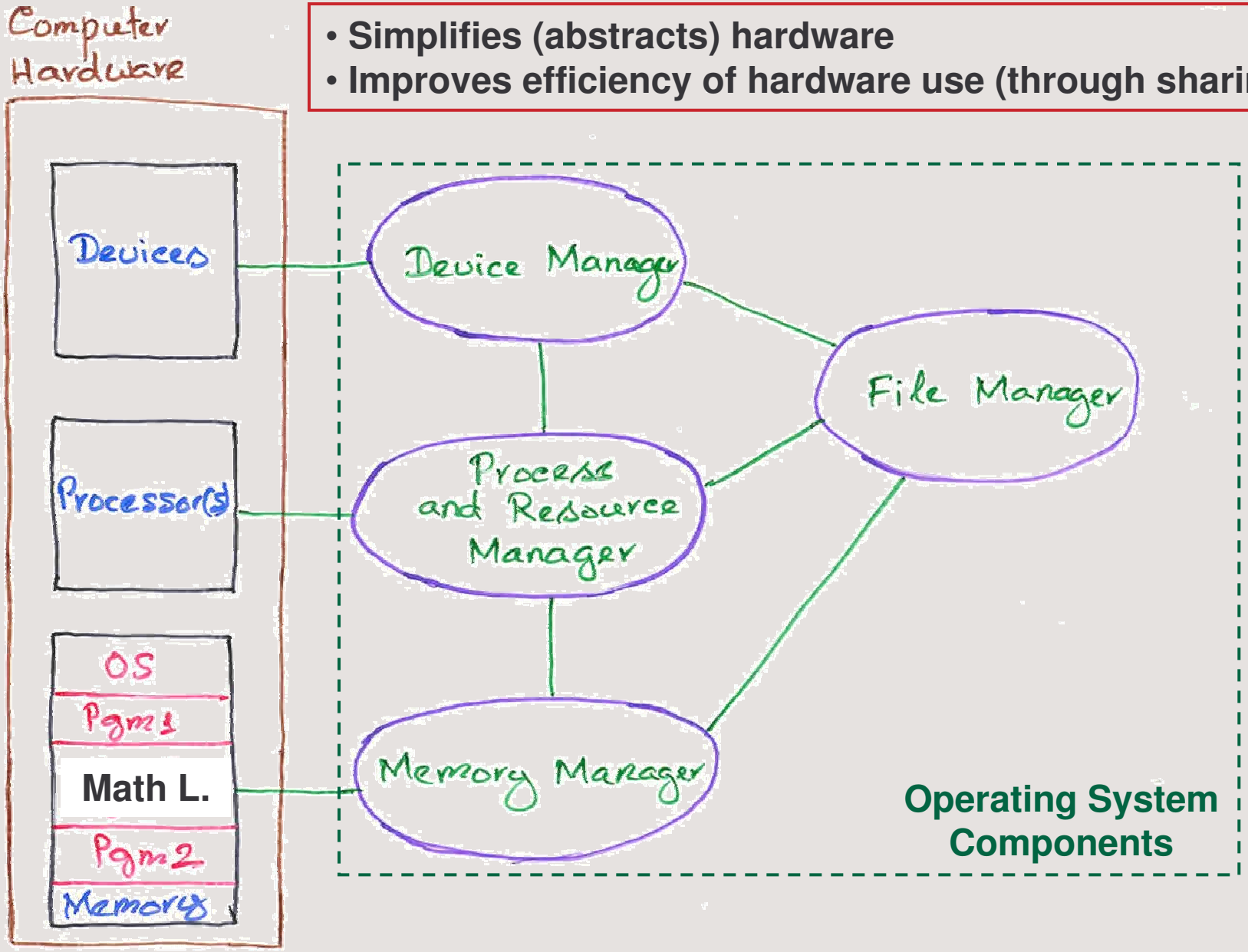
Sets of functions for use by other programs (e.g. math library)

Static: is included with calling object program when linking

Dynamic or Shared: a separate copy of the library in memory is called by programs at run-time

Example: Program Runtime Environment

- **Simplifies (abstracts) hardware**
- **Improves efficiency of hardware use (through sharing)**



This Week: Introduction to Operating Systems

System Software for Program Execution (pp. 1-18)

Systems and application software (again)
Resource Abstraction
Resource Sharing

Operating System Strategies (pp. 18-39)

A brief history of computing and operating systems
(batch, timesharing, personal, SCC's, embedded, networks)
The modern OS as integration/evolution of previous systems

Goals:

1. Learn the function and operation of operating systems
2. Learn how to exploit OS design during programming (for program execution)

System Software for Program Execution

Part II, pp. 1-18

The Operating System

Purpose

Provides **interface** (e.g. API) and **abstractions** (e.g. files) for hardware, coordinates **sharing** of hardware (e.g. multiprogramming)

Notes

- Is the software implemented “closest to hardware”
- General-purpose OS: domain-independent
- Embedded OS: often domain and/or device specific

Application Programming Interface (API)

Set of functions exported by the OS for invoking services by application programs (e.g. I/O, graphics)

OS Examples: Figures from Text

Figure 1.2 (page 3): Different Computer Perspectives

End user (e.g. word processing)

Application Programmer (e.g. C implementation)

OS Programmer (e.g. Linux kernel)

Figure 1.3 (page 5): Using System Software

(each system software component has its own API)

Virtual Terminals (e.g. X, Mac & Win Desktops, GNOME, KDE**)**

DBMS (customizable for different data types, searches/updating mechanisms)

Command line interpreter (shells: e.g. tcsh, bash, dos shell)

OS Main Functions: 1. Resource Abstraction

System Resources

System hardware ("physical") and software "artifacts" ("abstract")

Each resource has its own interface

Resource Abstractions

- Abstract (simplify) the interface to a device
  - e.g. writing to a "file" versus disk operation sequence
  - e.g. using printf() from stdio.h (C) versus setting screen bitmap
- Often convenient to use one abstraction for multiple devices (e.g. "file" for hard disks, floppy disks, compact discs, USB sticks...)
- "Generality has its price in specificity" (p. 7)
- Good abstractions: easy for programmers to understand, provide programmer with necessary control

Example: Resource Abstraction for Writing Data to Disk

Figure 1.4 (Disk Abstractions)

Example of increasingly abstract “write to disk” operations

See accompanying text on bottom page 9

Note: error (?) for the write() function (notice the arguments track and sector, which aren’t used in the body of the function)

OS Main Functions: 2. Resource Sharing

Multiplexing (in communication)

Sending multiple messages along a single communication channel simultaneously

(can be thought of roughly as “sharing” the channel)

Time-multiplexed resource sharing

Entire resource is shared by “taking turns”

e.g. processor being shared by executing programs

Space-multiplexed resource sharing

Resource is divided into elements that are used separately

e.g. memory being shared by executing programs

Example

Parking spaces in a parking lot (time and space-multiplexed sharing)

Abstract Machines and Transparent Resource Sharing

Abstract Machines

- OS-provided representation of system resources on which programs are “run” by users and programmers
- Allows program to be implemented as though running on a separate machine
- At run-time, operating system **transparently (invisibly) allocates resources** while simulating the execution of multiple abstract machines

Process:

A program running on an abstract machine

Example

Figure 1.5, page 12 (abstract machines)

Concurrent and Parallel Execution

Concurrent Execution

The appearance (through scheduling) or real situation that two or more programs are executing simultaneously

Parallel Execution

Two or more programs actually executing simultaneously (e.g. on multiple processors)

Multiprogramming (Concurrency)

Purpose

Scheduling execution of processes by the OS so that they share system resources (esp. processor(s))

Processes and Resources

OS time-multiplex shares processors

OS space-multiplex shares memory

OS determines scheduling (e.g. may switch when an active process starts waiting for input)

Timeslice

The amount of execution time (often a fraction of a second) given to a process during its “turn” on a processor

See Figure 1.6 (page 13): Mutliprogramming

Multiprogramming and Performance

Performance Effects

- We cannot do better than running a process without interruption
- However, multiprogramming can improve overall system performance

Why Multiprogramming Can Improve Performance:

- I/O much slower than processor operations
- Processor is not needed during I/O (e.g. while a user enters information, or debugs etc.)
- For interactive processes, I/O time often dwarfs processing time
- In a conventional computer, there are multiple devices but one processor

Expected (but not guaranteed) Performance:

$$\max(t1,...,tn) \leq T \leq t1 + ... + tn$$

Example of Best Case Performance (*max(t1,...tn)*)

Fig. 1.8 (page 15)

(try visualizing how processor is used when running P1...PN)

Multiprogramming and Performance (continued)

Expected Perf. May Not Happen When:

- Poor balance between processing and I/O
- Enough time for OS to compute schedule
- There must be few/no conflicting device requests

Analogy: Painting Cars

Problem: what is the best sequence of cars to apply coats of paint to, if different paints dry at different rates?

(here the person painting is time-shared in a way similar to the processor in a computer executing programs concurrently; in either case, we want to get programs or cars through as quickly as possible, with the least "idle time").

Figure 1.7 (page 14)

Note: (error?) **this is an example of parallelization** (space-multiplexed sharing of available resources: we "split" the vacuum-wash-dry units), **not concurrency** (time-multiplexed sharing of a single resource; e.g. cars don't take turns getting partially washed)

Explicit Resource Sharing

Purpose

Allows programmers to specify how resources are to be shared at execution time (e.g. shared memory)

Resource Isolation

OS insures that (normally) only one abstract machine can access a resource at any time (e.g. part of memory; processor)

Security Issues

If we allow processes to share resources when they ask, we need to guard against malicious access

“Trusted Software”

System Call Interface

Another name for the OS Application Programming Interface (API)

Summary of Application and System Software, OS

Figure 1.9 (page 18)

Operating System Strategies

Part II, pp. 18-39

Operating System Strategies

OS Strategy

General characteristics of the programmer's abstract machine (e.g. fixed number, whether there's interaction)

Early Days (approx. pre-1960)

One program at a time, basically no OS

Strategies

1. **Batch systems:** executed non-interactively, multiprogramming
2. **Timesharing systems:** multiple users interacting with the system; extended multiprogramming, security
3. **Personal Computers, and Workstations:** single-user machines; multiprogrammed, minimize wait time rather than max. hardware use
4. **Embedded systems:** guaranteed response times ("real time")
5. **Small, Communicating Computers (SCC's):** new resource management, power management, device management
6. **Network Technology**: handle resources, information on networks

1. Batch Systems

Job Control Specification (e.g. IBM JCL)

Sequence of OS commands defining a batch "job" (e.g. payroll)

1960's: batch jobs entered as deck of keypunched cards

Now: "batch jobs" defined using files (e.g. tcsh, .bat), but modern OS's do not use pure batch processing

Scheduling

1. Incoming jobs are organized into batches by an input spooler (early days: cards converted to magnetic tape using special computer)
2. OS reads job descriptions, schedules the job
   - **Medium-term scheduler:** allocates memory for jobs
   - **Short-term scheduler:** allocates processor to a job in memory when processor becomes idle
   - **Swapping systems:** remove jobs (processes) from memory and store them on disk in order to provide memory for other processes
3. When all batch jobs have been completed, results stored in the output spool (originally tape) and printed for users

Batch Systems and the User

Users of Batch Systems

Users were usually not permitted to enter their own jobs into the spooler, or to remove output from the system line printer.

Tracing your code in the old days...

It was not unusual to only have two chances a day to submit a job.

If both runs crashed, that day was a loss.

Examples

See Figures 1.10 (page 21) and 1.11 (page 23)

Batch Systems in Pictures

IBM 029 Key Punch (1964)

http://www.columbia.edu/acis/history/029.html

A Punch Card with a Job Control Specification (in JCL)

http://www.columbia.edu/acis/history/cards.html

IBM 360 (1964): running OS/360...VM/CMS

http://www.computermuseum.li/Testpage/IBM-360-1964.htm

2. Timesharing Systems

Interactive and Equal Treatment for Multiple Users

Divide processor time equally between users at terminals

Aim for “responsiveness” rather than jobs/hour (as in batch systems)

Multitasking Systems

Timesharing systems allowing multiple processes **(tasks)** per user

- Processes could be added/removed by users at any time
- More than one program might be running at a given time

Security

With multiple users interacting with the system, keeping user data and processes separate and safe became important (e.g. memory, user file system)

And the Users Rejoiced...

- **Users could now log in using a keyboard and display device (terminal), and interact with processes, while multitasking**

- **Encouraged more experimentation; didn't have to wait for printouts (or trace *all* your code *all* the time)**

Early Timesharing Systems

- **Compatible Time Sharing System (MIT, mid-1960's): memory-management, scheduling**
- **Multics (replaced CTSS) virtual memory, protection, security**
- **Cal**
- **UNIX (AT&T Bell Labs, 1970) intended to be simpler than Multics, for research machines. Used a small "kernel" that supported OS services executing as application programs**
  - UNIX was designed to be able to be ported to any small computer (later, FreeBSD and Linux arrive)

Timesharing Systems in Pictures

Figure 1.12 (page 25)

PDP-8 (1965): various OS, including timesharing ones

http://research.microsoft.com/~gbell/Digital/timeline/1965-2.htm

PDP-11 (1970): various OS, esp. UNIX (machine, console)

http://research.microsoft.com/~gbell/Digital/timeline/1970-2.htm

http://www.psych.usyd.edu.au/pdp-11/11_45.html

Richie and Thompson Porting UNIX to the PDP-11:

http://cm.bell-labs.com/cm/cs/who/dmr/picture.html

VAX 11/780 (1977): VMS, Ultrix, BSD, ...

http://ed-thelen.org/comp-hist/vs-dec-vax-11-780.jpg

Machine “Types” by Size

Mainframes

Monolithic machines that required air-conditioned, environment-controlled rooms with special power resources

Minicomputers

Much smaller than mainframes; could be installed in any room

Could act as a personal machine or timesharing system

(e.g. Digital PDP-8, PDP-11, VAX)

Microcomputers

Smaller than minicomputers (e.g. Apple II, IBM-PC)

Key attribute: single integrated-circuit processor implementation

3. Personal Computers and Workstations

Personal Computer (arrived late 1970's)

- Designed for the home market
- For about ten years, these systems had no multiprogramming
- Early systems: "OS" was simply routines for controlling devices (e.g. tape drive, display) stored in ROM

Early Personal Computer OS's

- CP/M (eventually displaced by...)
- MS-DOS / PC-DOS (IBM version)
- *Key innovation of these 3 OS's*: adding a file system

System BIOS

BIOS

IBM “Basic Input/Output System” (part of the IBM-PC)

Abstracted system devices (and still does!)

Stored in Read-only-memory (ROM), loaded on system start

Similar Systems

Many other early “personal computer OS’s” were also device abstractions stored in ROM

- (e.g. early Commodore, Sinclair, and Apple machines had BASIC routines that would check and set memory, write to a tape drive)

Personal Computers in Pictures

Apple II (1977): BASIC "operating system"
http://www.edwardsamuels.com/illustratedstory/chapter%204/AppleII.jpg

Macintosh (1984, Mac "Plus" shown): MacOS (first widely available windowing OS)
http://koti.mbnet.fi/~oju/retro/MacSystem.jpg

Commodore 64 (1982): Commodore BASIC
http://people.clarkson.edu/~johndan/datacloud/images/commodore_64.jpg

Commodore PET (first v. 1977): BASIC OS by Gates/Allen burned in ROM
http://webpages.charter.net/thecomputercollection/micros/pet.jpg

Sinclair ZX81 (1981): Yet Another BASIC-based OS (also in ROM)
http://www.myoldcomputers.com/museum/comp/museumpics/zx81.jpg

IBM PC (1981): MS-DOS/PC-DOS (with BIOS in ROM)
http://www.vintage-computer.com/images/ibmpc.jpg

OSBORNE-1 (1981): CP/M ("luggable" computer)
http://oldcomputers.net/osborne.html

Workstations

Personal Computing (1980's – early 1990's)

"Light computation": spreadsheets, word processing, simple games, (very) simple programs

Workstations

- First workstation produced by Sun in 1982 (running SunOS,a UNIX variant)
- Up to the mid-90's, workstations were similar in size but much more powerful than personal computers (faster processor, more memory, more storage, larger displays, etc.)
- Generally used for what was considered "heavy computation" by individuals: simulations, visualizing data, complex programming, etc.
- Usually had some UNIX variant as the OS, complete with timesharing and multiprogramming, network protocols

Today: Workstations and Personal Computers are not distinguished

Workstations in Pictures

Sun Workstation (1982): SunOS (BSD-based)

http://www.digibarn.com/history/03-01-09-SmithsonianVisit/micros/Image55b.jpg

DEC Mice

http://hp.vector.co.jp/authors/VA000175/image/Deci.jpg

DEC Alpha 21064 Workstation (1992): VMS, Digital Unix

http://sites.inka.de/pcde/pic/dec3000_300lx_2.jpg

Symbolics LISP machines (workstations; first appeared 1980/1981): Symbolics OS

http://www.frobenius.com/sym.gif

4. Embedded Systems

Definition

Computer dedicated to supporting a more complex system

No human user: user is a set of sensors and actuators

Motivation

Cheaper to use a combination of hardware and software than hardware alone (e.g. disk controllers)

Properties

Targeted: some embedded systems run one application

Level of abstraction tends to be closer to the machine than "general purpose" operating systems

Embedded Systems Examples, Pictures

Systems in Nuclear Power Plants

Systems in Cars

Kettle that can be text-messaged to boil water (2005)

http://www.techdigest.tv/images/orange_pg_tips.jpg

VxWorks Embedded OS

Figure 1.14 (page 33)

Scheduling in Embedded Systems

Goals

- Fast response
- Minimizing memory, processor cycles used by OS (overhead)
- Minimize power consumption for running programs
- These days: maintain consistency when devices are spontaneously powered down (e.g. disks, memory, storage), and keep running

Real-time Systems

Required to complete tasks by fixed deadlines (e.g. maintaining nuclear reactor temperature)

Issues: how to guarantee response times, how to achieve minimum response times

Soft Deadlines

System should try to meet deadline, but complete task if the deadline is missed (e.g. media streaming)

5. Small, Communicating Computers

Examples

MP3 players
Personal Digital Assistants (PDA's)
Internet-Capable Mobile Phones
Digital Recorders (TIVO, etc.)

Main Issue:

Scarce resources (bandwidth, memory, power)

Pictures:

Apple Newton (released 1993): Newton OS
http://members.aol.com/stevenw9/mp2000.jpg
(Palm) Pilot (1996): PalmOS
http://upload.wikimedia.org/wikipedia/ca/8/83/180px-PalmPilot5000.jpg
BlackBerry (1999): BlackBerry OS
http://uis.georgetown.edu/images/hardware/blackberry/7230lg.jpg
Windows CE (Figure 1.15, page 36)

SCC OS's vs. OS's for Larger Machines

- **Hardware (and usage) is different, so hardware abstractions are also different**
- **Different members of the same "family" may have different abstract machine policies**
- **Traditional OS's not concerned with real-time processing (vs. media streaming on an MP3 player, for e.g.)**
- **Thread rather than process-based (threads use fewer resources than processes)**
- **Dynamic rather than fixed scheduling policies and resource management (e.g. network)**

6. Networks

Personal Computers, Workstations

Created demand for sharing devices (e.g. printers) and information between machines

Issues

Resource isolation, sharing, abstraction for local/remote resources

Some Popular Network Protocols

IP: Internet Protocol

TCP: Transmission Control Protocol

HTTP: Hyper Text Transfer Protocol

Network Technology

Pre-1980

- Point-to-point connections, < 10 kilobits/second
- Computer network: all machines connected, or routing network used (machines required to pass data for other machines on)

Local Area Networks (LANs)

- 1980: 10-16 megabits/second (Ethernet, Token Ring)
- Present: 10-1000 megabits/second

Wireless LANS (~2000)

- Bluetooth/WiFi: trasmit in small area (<100 feet)
- Transmission rate: 11 megabits/second
- Network is frequently changing (dynamic) and distributed (vs. fixed LAN)

Summary: Evolution of the Modern OS, Different Windows OS's

Figure 1.16 (page 39)

Modern OS as descendent of different strategies we have looked at

Figure 1.13 (page 30)

Summary of different Windows Operating system API's

Next Week:

User-Level View of Operating Systems

read pp. 42-73