Operating System Overview

Chapter 2 of [OS4e], Chapters 3 and 1 of [OSC]:

- Operating System Objectives
- Operating System Functions
- System Calls
- Portability
- System Programs
- The Evolution of Operating Systems

Operating System Objectives (1)

OS can be viewed as a user/computer interface

Objective: convenience of use (running and developing application programs)
Operating System Objectives (2)

OS can be viewed as a resource manager

Objective: efficiency of processing, storage and communication

Operating System Objectives (3)

OS can be viewed as a program

Objective: reliability and ease of maintenance
Operating System Functions

What constitutes an operating system?
e.g. should it include Web browser, Java Virtual Machine and media player?

- Microsoft was taken to court in the United States for anti-competitive conduct in making Internet Explorer 4.0 (and a non-standard JVM) non-removable parts of Windows 98
  - Netscape Navigator and Java technology were set to become “middleware platforms” for application development, but …
  - they could be implemented on a variety of operating systems, so …
  - there would be less reason for users to stick with Microsoft Windows

- In 2004 the European Commission ruled that Microsoft broke EU competition law in tying Windows Media Player to its operating system

OS facilitates program execution

- loads instructions and data into main memory
- initialises I/O devices and files, etc.

It must be able to detect and respond to errors

- e.g. end program after illegal attempt to access memory
- e.g. retry operation when device not ready
- e.g. recover after power failure
Services are provided by OS in areas in addition to program execution, namely,
- file management
- device management
- information maintenance
- communications

_These are the subject of the follow-up unit “Operating Systems: I/O and Networks”_

Resource management activities performed by OS include
- scheduling user programs on processor(s)
- allocating space in main memory to programs and data

But because OS is itself a program
- it is only in control when its instructions are being executed, and it must frequently relinquish control to user programs
- some of its instructions and data occupy main memory
  - _kernel_ of OS may remain there permanently
Operating systems evolve over time
- as hardware changes
  - to take advantage of new features in processor
  - to manage larger memory address spaces
  - to support new devices
- as new services are required
  e.g., support for multimedia, networks, Java Virtual Machine
- as bugs are discovered and fixed
Modularity and good documentation are therefore essential

System Calls

*system calls*
- programming interface to OS services
  - caller executes in user mode
  - callee (e.g. UNIX kernel or Windows XP executive) executes in supervisor mode
- can be made directly in assembly language
  - by storing call number in register and invoking *trap* instruction (handled as an interrupt)
- can be made indirectly in C, C++, Perl
  - by calling library procedure that makes the actual system call
- UNIX man pages describe those procedures that directly invoke system calls (section 2) and those that don’t (section 3)
- Microsoft doesn’t publish system call interface to its Windows operating systems
  - instead it provides Windows Application Programming Interface (formerly known as Win32 API)
Portability

“a software developer who wishes to write an application in the C or C++ language to run on both the Windows/X86 and the MacOS/PPC runtime environments must write separate source code for each separate environment, must separately compile that source code, and then must produce, test, debug and distribute different machine code binaries for each runtime environment.”
J.A. Gosling, 1988

Suppose same OS is implemented on different hardware platforms

- Applications that only access hardware through system calls can run on any platform

Application could even be portable across different operating systems

- if it only uses system calls common to each OS
  e.g., POSIX.1 is a standard supported by proprietary UNIX operating systems and by Windows XP
- if it uses API provided by middleware, rather than OS
Primary goal of Java programming language was portability
- “write once, run anywhere”
- Java programs compile into bytecode

Java platform
- consists of Java Virtual Machine (JVM) that executes bytecode in a class file
- implemented on host OS, or as part of Web browser, or in hardware

"in the JAVA programming environment, only one version of a program’s source code needs to be developed and supported, only one compiler needs to be used, and only one binary version of the program needs to be generated, tested, debugged and distributed.” J.A. Gosling, 1998
Portability can be sacrificed by writing Java programs that invoke “native” methods.

Java Native Interface (JNI)
- allows Java code to operate with applications and libraries written in other languages, such as C, C++ and assembly
- used when application cannot be written entirely in Java
  - e.g. to access platform-dependent features
  - e.g. to reuse non-Java legacy code
  - e.g. to implement time-critical code in lower-level language
- http://java.sun.com/docs/books/tutorial/native1.1/

System Programs

*systems programs (utilities)*
- provided with OS to solve common problems or perform common operations
  - user interfaces to system calls
  - compilers, assemblers, interpreters, linkers, loaders, debuggers, libraries for programming languages
  - editors, browsers, telnet clients and servers, etc.
- execute in user mode
command interpreter (shell)
- system program that provides user interface
e.g., command lines or clickable menus
- runs when job initiated or user logs on

UNIX shells
- treat command as the name of a file
  - search for file with that name and execute it
- advantages are shells are small and extensible
- disadvantages are
  - file with same name may be inadvertently executed
  - passing parameters from shell to executable file may be inefficient

The Evolution of Operating Systems
Adapted from “Operating Systems Lecture Notes”, Copyright 1997 Martin C. Rinard, MIT
- late 1940’s: Expensive Hardware, Cheap People
  Goal: maximise hardware utilisation
- now: Cheap Hardware, Expensive People
  Goal: make it easy for people to use computer
In the early days, computers were

- huge machines, expensive to buy, run and maintain
- operated in single user, interactive mode
  - programmers interacted with the machine at very low level (flicked console switches, dumped cards into card reader, etc.)
  - interface was basically raw hardware

Problem: Programs that manipulate external I/O devices are very difficult to write

Solution: Build a subroutine library (device drivers)
  - loaded into the top of memory and stays there
  - first example of something that would grow into an operating system
Because the machine is so expensive, it is important to keep it busy

Problem: Computer is idle while programmer sets things up, resulting in poor utilisation of huge investment

Solution: Hire an operator to do setup

Problem: Set up is still too slow

Solution: Build a batch monitor
  – store jobs on a disk (*spooling*)
  – computer reads them in one at a time and executes them

Batch processing represented significant change in computer usage:
  – debugging now done off-line from print outs and memory dumps
  – no more instant feedback

Problem: At any given time, job requires either the CPU or an I/O device
  – rest of machine idle and therefore unutilised

Solution: Allow the job to overlap computation and I/O
  – buffering and interrupt handling added to subroutine library
Problem: one job can't keep both CPU and I/O devices busy
  - compute-bound jobs mainly use CPU
  - I/O-bound jobs mainly use I/O devices
Solution: *multiprogramming* - several jobs share system
  - dynamically switch from one job to another when the running job does I/O

Big issue: *protection* - don't want one job to affect the results of another
  - memory protection and relocation added to hardware
  - OS must manage new hardware functionality
  - OS starts to become a significant software system
  - OS also starts to take up significant resources on its own
Phase shift: Computers become much cheaper, People costs become significant

Issue: It becomes important to make computers easier to use and to improve the productivity of the people

One big productivity sink: having to wait for batch output (but is this really true?)

Problem: It is important to run interactively, but computers are still so expensive that you can’t buy one for every person

Solution: time sharing - multiprogramming used to handle multiple interactive jobs

Problem: Old batch schedulers were designed to run a job for as long as it was utilising the CPU effectively

– in practice, until it tried to do some I/O

But now, people need reasonable response time from the computer

Solution: preemptive scheduling - timer interrupts job
Problem: People need to have their data and programs around while they use the computer

Solution: Add file systems for quick access to data
- computer becomes a repository for data
- people don't have to use card decks or tapes to store their data

Problem: The boss logs in and gets terrible response time because the machine is overloaded

Solution: *prioritised scheduling*
- the boss gets more of the machine than the peons

But, CPU scheduling is just an example of resource allocation problems

Time-shared systems are full of limited resources
- e.g., CPU time, disk space, main-memory space

It became the responsibility of the OS to mediate allocation of resources
- e.g., introduction of quotas
Overall, time sharing was a success
  - however, it was a limited success
  - in practical terms, every timeshared computer became overloaded
  - response time dropped to annoying or unacceptable levels
  - people compensated by working at night
  - birth of the computer nerd (pasty-looking, unhealthy insomniac, addicted to caffeine)

Computers become even cheaper
  - practical to give one computer to each user
  - initial cost is very important in market
  - minimal hardware (no networking or hard disk, very slow microprocessors and almost no memory)

Shipped with minimal OS (MS-DOS)
  - protection, security less of an issue
  - OS resource consumption a big issue (computer only has 640K of memory)
  - OS back to a shared subroutine library
Hardware becomes cheaper and users more sophisticated

- people need to share data and information with other people
- computers become more information transfer, manipulation and storage devices rather than machines that perform arithmetic operations
- networking becomes very important, and as sharing becomes an important part of the experience so does security

Operating systems become more sophisticated

- start putting back features present in the old time sharing systems (OS/2, Windows NT, Linux)

Operating systems nowadays also control

- **handheld computers** (PDAs and mobile phones)
  - limited memory
  - slow processors
  - small display screens
- **embedded computers**
  - simply everywhere! (e.g. cars, ovens, tickets)
  - real-time control
  - limited features
  - little or no user interface