Wed Sep 6, 2006
Lecture 1: Introduction to OS

September 13, 2006

1 Mechanics of Course: V22.0202

Grade Grade is curved: Written Homework (20%), Programming Homework/Project (35%), Midterm (15%), Final (30%).

General Remarks about READING. Try to read ahead of our lectures. Even reading for 10 minutes (just the introduction or section 1 of a chapter) is useful to get an orientation before class lectures. It does not matter if you don't understand all the concepts you read – you actually do gain some rough impressions and recognize the buzzwords of the topic. This alone will improve your understanding of lectures.

Assigned READING for first 2 weeks. There is a lot of information being summarized in the PART ONE (first 3 chapters) of the book. Read them on your own time:
Chap.1 of Text (Intro – read lightly – read up to 1.4 for now)
Chap.2 of Text (Computer System Structures – read up to 2.4)
Chap.3 of Text (OS Structures – read up to 3.4)
We will begin with Chap.4 (process management). Read up to Section 4.5.

Goals of this course We want to study the organizational principles of modern operating systems. Part of of this knowledge will be acquired through programming assignments and projects.

Besides Windows, we want students to also become familiar with various aspects of Unix (or its derivatives). There is a very easy way to get a free Unix-like environment via Cygwin. Go to my website http://cs.nyu.edu/ yap/prog/cygwin/ for more information and download. Unix development is based on the concept of modular units (programs) called "tools". For instance, we will learn to use the Make program for organizing programming projects.

On Programming: We will use Java and C. It is essential to do C because this is the language used for most systems projects. As students may not be familiar with C, so we will try to provide the necessary information.

2 Overview of OS

A. Abstract Levels of a Computer (3 levels)

- Hardware/architecture – CPU, MM, IO/Physical Devices
- OS
- Application Programs – word processors, compilers, browsers

B. What is an OS? Book says an OS is like a "goverment", that does no useful function of its own, but to enable others to do useful function.

Three main characterizations of the functions of an OS:

- OS as a manager of processes.

  Perhaps the most natural (from a user-viewpoint) characterization of an OS.

- OS as an extended or virtual machine.

  The OS extends the hardware's raw capability in simple, natural and standard way. E.g., read/write to disk is available in hardware, but the OS adds various conveniences in the system calls.

  The set of these convenient functions (i.e., OS's system calls) constitute the API of the OS.

  E.g., all Unix-derived OS's implement a set of about 100 system calls. This set is defined by an international IEEE standard called POSIX (= Portable Operating System Interface for uniX).

- OS as a resource manager:

  E.g., managing multiple users, CPU time, memory space and hardware devices. The goal of the manager is efficiency, fairness, protection, as well as convenience.

C. OS Architectural Elements

- CPU

  Instruction set, registers

  Program Counter (PC) (or IP, Instruction Pointer)

  Stack Pointer (SP)

  Stack contains a frame for each procedure that is entered

  Program Status Word (PSW) – for system calls, errors, etc

- RAM (or Main Memory, MM)

  Accessed directly by CPU thru LOAD/STORE

  MMU (Memory Management Unit) between CPU and RAM

  MMU to map (i.e., generate physical addr from virtual/logical addr) and do protection checking

- Disk (or Secondary Memory)

  Magnetic technology

  Connected via I/O Bus (ATA,SCSI, etc)

  Disc Controllers

- IO Devices

- Device controllers

- Busses

Interrupts/Events/System Calls

- CPU can execute in one of two Modes: user/kernel

  Kernel mode has more previleges than User mode. Normally, the OS routines are executed in kernel mode an user routines are executed in user mode.

- Interrupt (aka trap, exception, system call) – generated by user or system or hardware

- E.g., to make a system call for system service

  E.g., I/O controller signals completion of its task

  E.g., Divide by zero error

- Interrupt Processing:

  1. CPU disables further interrupts.

  2. Stops current process and stores the context of the process.

  3. Determines the type of interrupt that has occurred: "polling" or "vectored interrupt system".

  3. Transfers control to a fixed location, which has the starting address of service routine. This address is called the INTERRUPT Vector.

  4. Executes the service routine in KERNEL MODE.

  5. Returns to the interrupted process (in USER MODE).

- FORMS of interrupts:

  We can think of "INTERRUPTS" as a generic name for any action that causes the CPU to instantly respond. But often, it is associated with such actions which are initiated by hardware devices. E.g., I/O Interrupt.

  TRAP: a machine level instruction which is executed during an interrupt.

  SYSTEM CALL: user-initiated interrupts provided by the OS to provide kernel mode service. The set of such calls essentially defines the OS.

  EXCEPTION: error-generated interrupts (e.g., divide-by-zero exception)

- EXAMPLE: "read" system call in UNIX.

  In C program, we can make the system call

  count = READ(fd, buffer, nbytes);

  (1) Calling program program pushes parameters onto stack and then calls the library procedure.

  (2) Library procedure, probably written in assembly, puts the system call number in a standard place (some register), and executes a TRAP instruction.

  (3) This switches from User to Kernel mode, and begins execution at a fixed address in the kernel.

  (4) The kernel code examines the System Call Number and dispatches to the system call handler.

  (5) After the system call handler returns, control MAY return to the user-space at the instruction following TRAP instruction.

  WHY "MAY" return? Because the system may block the caller, so some other process may be run. In this case, the current process is also blocked.

  (6) This library procedure returns to the user program.

  (7) User program then cleans up the usual way after a procedure call.

Memory Hierarchy WLOG, all memory is divided into fixed units called words (32 bits or 64 bits).

Many considerations in memory management are based on the existence of the different types of memory that form a linear **Memory Hierarchy**:

- register (< 20)
- cache ($< 1MB$)
- (electronic disk)
- magnetic disc (RAM) (< 1 TeraByte)

- (optical disk)
- tape.

The hierarchy is characterized by the capacity-speed tradeoff: as a particular memory type increases in speed, its capacity is decreased.

Multiprogramming: why we need it We take multiprogramming for granted today. But it is instructive to see why it is useful.

What is the alternative to multiprogramming? A sequential job queue (perhaps modified by priority). This is old-fashion (up to late 70's) way of running a computer system. Users submit jobs (a stack of punched cards!) in an input-tray, and return an hour later to pick up the output of running their job. If the job did not compile or fail for some reason, you fixed the punched cards and resubmit.

Consider how multiprogramming wins over this scenario. There are two basic SCENARIOS:

(1) Processes can be classified as I/O-bound or CPU-bound.

I/O-bound, e.g., reading and writing files, or interactive programs. CPU-bound, e.g., pure number-crunching application (prime factorization)

While running I/O bound processes, the CPU would be under-utilized while waiting for I/O to complete. Thus, we want to have other processes when this happens. Note that in the pre-1980 era, there were no interactive programs, but there are still I/O bound jobs in printing jobs.

Thus, by introducing multiprogramming, we greatly improve the utilization of CPU cycles, for a slight overhead of slowing down I/O-bound processes.

(2) Processes can be classified into big jobs and small ones. We do not want short jobs to be held up a long time because a long job was just a little ahead of it. So we want to give fixed quantums to each job in the current queue. This way, the user's wait time is roughly proportional to the size of their jobs, all other things equal. This seems fair.

Thus, by introducing multiprogramming, we greatly improve the fairness to small jobs, for a slight overhead of slowing down the big jobs. The swapping of jobs incurs a slight waste of CPU cycles.

Note that the motivation for multiprogramming in these 2 cases are different: the former has to do with optimizing resource usage, the latter has to do with fairness.

3 OS FUNCTIONS – Chap 3

A. Processes Management

- Perhaps the most important concrete concept!
- Process = program in execution (PC, Image, etc)
- OS has to create/delete/suspend/resume processes
- OS provide for synchronization + communication of processes

B. Main Memory Management

- An array of addressable words, faster than secondary memory
- Processes are swapped in/out of MM
- DMA driven I/O requires no CPU intervention

C. File Management

- Different physical media (disk, tape, CD, etc)
- files and directories
- create/delete/modify
- organization of file (mapping)

D. Other issues

- I/O Management
- Networking
- Protection
- Security
- *etc*, ...

4 VARIETIES OF OS – Chap 1

- Multiprogrammed systems. Rationale: jobs waiting for I/O. Scheduling needed.
- Timesharing systems. Extension of multiprogramming. More complex – need virtual memory for instance.
- Desktop systems. Single user, many jobs.
- Multiprocessor systems. VERY LARGE VARIETY (how tightly coupled, communication, topology, etc). Symmetric multiprocessing (SMP) where each processor runs a copy of OS. Asymetric multiprocessing is more master-slave relations.
- Distributed sytems. Loosely coupled, geographically distributed. Computer networks (LAN, WAD, MAN). Client-Server systems. Peer-to-peer systems. CLUSTERS.
- Realtime systems. Physical process controllers (e.g., Nuclear reactor). Embedded system. Multimedia applications. Soft-realtime.
- Handheld systems. PDAs.

5 Metric Unit Prefixes

Positive Powers of 1000: Kilo (3), Mega (6), Giga (9), Tera (12), Peta (15), Exa (18).

Negative Powers of 1000: milli (-3), micro (-6), nano (-9), pico (-12), femto (-15).

REMEMBER: $2^{10} = 1000$ and $2^{20} = (2^{10})^2 = 1,000,000$.

"One order of magnitude" means "a multiple of 10".

E.g., CPU speed is 3 orders of magnitude faster than I/O speed means CPU speed is 1000 times faster than I/O.

E.g., Moore's Law says: "chip density doubles every 18 months" This usually taken to imply that CPU speeds doubles every 18 months, but this correlation is not direct. Assuming that computing speeds does double every 18 months, it means that we can get 3 orders of magnitude in speedup every 15 years.

6 SHORT HISTORY – Chap 1

First digital computer: Charles Babbage (1850's)

1950-65: mainframe with no OS. batch processing. introduction of punched card programs.

1960s: IBM System/360, designed for scientific (number crunching) as well as commercial use (word processing). Multiprogramming introduced.

1970s: Multics (time share). Unix from Bell Labs (Ken Thomson)

1980: Personal Computers, DOS ("Disk Operating System") from Microsoft. Windows 95.

1990: Windows 2000, Linux