EECS3221 notes

Jerry Wu

Week I

Chapter 1:: Intro

Abstract

In general, the computer system stack is divided into 4 components from bottom up:

- Hardware (CPU, memory, storage, IO, etc.)
- OS - Controls and coordinates use of hardware among various applications and users
- Applications (word processors, compilers, web browsers, database system, games, etc.)
- Users (the people using the system, other computers, etc.)

What is an OS, what do they do?

There is no universally accepted definition of what an OS is, however, operating systems are generally a **medium** between the user and a set of hardware such that the user can accomplish an arbitrary objective. This would depend on the user in question since multiple people would have different goals in mind that they want to accomplish.

Operating systems allocate resources and acts as a control program in order to make optimal use of hardware while managing execution of programs and applications. For example, the operating system on a desktop PC would optimize performance, while most mobile operating systems are resource poor, but optimize usability and battery life.

We can divide an operating system into its own parts as well::

- Kernel - the part of the OS that is running at all times
- System program - something that is included with the OS, but not part of the kernel

- Application program - everything else that isn't associated with the core OS.

Modern operating systems for general purpose/mobile computing also come with **middleware**, a set of software frameworks (APIs) that provide additional services to application developers such as databases, multimedia, graphics, etc. A good example is the .net framework in Windows, or gcc being included with most linux distributions.

Computer system organization

- One or more CPUs, device controllers connect through a common **bus** providing access to shared memory
- Concurrent executions of CPUs and devices competing for memory cycles
- IO devices can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating sstem **device driver** to manage it
- CPU moves data from/to main memory to/from local buffers
- IO is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

Interrupts

- Interrupt transfers control the **interrupt service routine** generally through the **interrupt vector**, which contains the addresses of all service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or an **exception** is a software generated interrupt caused either by an error or a user request
- All operating systems are **interrupt driven**

Handling interrupts

- OS preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred
- Separate segments of code to determine what action should be taken for each type of interrupt

IO structure

- After IO starts, controller returns to user program only upon IO completion
-

Storage structure

Storage systems can be organized in a hierarchy of factors::

- Speed
- Cost
- Volatility

Caching

The act of copying information into a faster storage system such that the main memory can be viewed as a cache for secondary storage.

Device driver

For each device controller to manage IO

- Provides uniform interface between controller and kernel

As of today, **SSDs are strictly slower than RAM chips**.

Magnetic drives like floppy disks, hard drives, tape, etc. are **sequential**, in that if one wishes to access something in the middle of the storage system, they would have to iterate through everything before that index.

Proessors

- Most systems use a single general purpose processor along with a few special purpose processors
- **Multiprocessor** systems are growing in use and importance, advantages include, but are not limited to::
  - Increased throughput
  - Economy of scale
  - Increased reliability (graceful degradation or fault tolerance)
    - **Graceful degradation** - don't show a cryptic message when something bad happens. Doesn't enable hackers to extract sensitive information
    - **Fault tolerance** - quickly recover from crashes and exceptions
- There are two types of multiprocessors
  - **Asymmetric multiprocessing** - each processor is assigned a specific task
  - **Symmetric multiprocessing** - each processor performs all tasks (the most common one)

Clustered systems

- Similar to multiprocessor systems, but multiple systems working together independently from one another
  - usually sharing storage via a **storage area network** (SAN)
  - Proides a **high availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot standby mode
    - **Symmetric clustering** has multiple nodes running applications monitoring each other
  - Some clusters are for **high performance computing** (HPC)
    - Applicaations must be written to use parallelization
  - Some have **distributed lock manager** (DLM) to avoid conflicting operations

OS operations

- Bootstrap program - simple code to initialize the system and load the kernel
- Kernel loads
- Starts system daemons (services provided outside of the kernel)
- Kernel interrupt driven (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (exception or trap):
    - Software error (e.g. zero division error)
    - Request for operating system service - **system call**
    - Other process problems including infinit eloops, processes modifying each other or the OS

Multiprogramming and multitasking

Before OS was a thing, computers could only run one program at a time. Multiple programs would be executed sequentially.

Multiprogramming (batch system)

- Needed for efficiency
- Single user cannot keep CPU and IO devices busy at all times
- Multiprogramming organizes jobs (code and data) such that CPU can always have one to execute
- A subset of total jobs in the system is kept in memory
- One job selected and run via **job scheduling**
- When it has to wait (for IO for example), OS will switch to another job

Tunesharing (multitasking)

A logical extension in which the CPU switches jobs so frequently that the user can interact with each job while it is running, creating **interactive computing**.

- Response time should be $< 1s$
- Each user has at least one program executing in memory (process)
- If several jobs are ready to run at the same time, use CPU scheduling
- If processes don't fit in memory, swapping moves them in and out to run
- Virtual memory allows execution of processes not completely in memory

Nowadays, we see less and less memory errors since computers have so much more memory real estate to use for running programs.