SOFTENG370 Notes 2017

Theodore Oswandi

July 27, 2017

1 Lecture 1

1.1 Generics

Operating System The software that makes the computer usable. Using modern computers without an OS is "impossible"

Examples: Windows, OSX, Linux, Unix, iOS, Android, etc...

1.2 Approaches to Understanding

Minimalist

- mostly going to be using this one
- OS contains minimum amount of software to function
- archlike

Maximalist

- All software comes with standard OS release.
- Contains many utilities and programs.
- ubuntuish

1.3 Usable vs Efficient

- make sure you make OS suited for needs
- either specialised or more general purpose
- Think of who you expect to use the system
- If creating a realtime system with potentially thousands of operations in a short amount of time, have to consider efficiency
- Same with battery life if you expect the system to be used in a mobile setting.

1.4 OS themes

Manager Model

- OS is colleciton of managers, ensuring proper use of devices.
- Managers are independent.
- look out for everything associated with computer
- tie in with hardware. Current state of HW lets OS do more/less things

Onion Model

- Onions have layers (Abstractions)
- resources contained in lower layers.
- Lower layers can't access higher level layers but other way around possible
- Very difficult to get these layers 'right'
- can use in terms of security. Very good idea

Resource Allocator Model

- similar to manager model
- emphasis on fairness and providing services

Dustbin Model

- contains middleware that not considered part of OS
- Sees OS as bits no-one wants to do

Getting Work Done Model

- Idea of it is we use computers to do something else.
- Goal for OS is to help be able to get it all done.

1.5 OS design

1.5.1 Themes

All in one

- All OS components freely interact with each other
- MS-DOR and Early Linux

Separate Layers (Onion Model)

- Simplify verificiation and debugging
- Correct design difficult to get

Modules

- All in one with modules for some features
- Linux and Windows.

Microkernels

- Client/Server model
- make OS as small as possible
- **Exokernel** puts kernel outside. OS's job only need to authenticate people to use hardware.

VMs

- Java is an example of this

1.5.2 MS-DOS

- Written to provide the most functionality in the least amount of space
- not divided into modules
- Something exokernels trying to do. Make application program access hardware directly.

1.5.3 Early Unix

- UNIX OS in 2 parts. **Kernel** and **System Programs**
- Provides:
  - File System
  - CPU sheduling
  - Memory management
  - Other OS functions

- Ken Thompson and Dennis Ritchie
- Make OS as simple as possible.
- Simple 2 letter commands.
- Ideas of pipelining and process communication

1.5.4 THE Multiprogramming System

- THE was the first to use the layered system
- Contains 6 layers:

  5 User programs
  4 Input/Output buffering
  3 Operator-Console device driver
  2 Memory Management
  1 CPU scheduling
  0 Hardware

1.5.5 WinNT and Client/Server

- WinNT still being still run
  - Win10 now has Windows Subsystem for Linux
- NT provide env subsystem to run code written for differnt OS
- NT and successors are hybrid systems. Parts are layered but some merged to improve performance.

Lecture 2: History of OS Start at mainframes. Early PDAs were similar to mainframes. Had no memory protection. Then go to Minicomputers And then desktop And how handheld computers

Go through cycle of: No software Compulers Multiuser Networked Distributed Systems Multiprocessor Fault tolerant.

Total Control Computers expensive in 50s Data Programs saved on magnetic strip, or punch cards. Programmers knew how the computer worked. They were very knowledgable about computers. Computers did 10,000s instructions per second, but were idle a lot

Old school computing IO polling, since no other programs running in bg, can just poll. CPU doing nothing.

Progressing Goal is to reduce the time CPU was doing nothing. Operators now just "use" the computer. No need for programmer. If something crashes, then just start the next program. Batch similar jobs together, maximise usage of computer.

Offlining With Big Expensive Computer BEC, but they are just waiting for IO a lot of time. Therefore want to make IO as fast as possible. Use smaller computers to convert slower paper to faster magnetic tape. Then that magnetic tape is used as IO for the BEC This is the same for output. Have another smaller cheaper computer offload the output magnetic tape from BEC to a printer.

Resident Monitor Keep some code in memory. It did the work that some operators were doing. Things like clearing memory, and reading start of new program that needs to be loaded. Can also do some of the IO routines.

Control Programs Standardise the language to communicate with the Resident Monitor.

After all of this COPY RELEVANT FROM SLIDE

Changes in hardware Disk drives Faster IO Processors that you can interrupt. No more reliance on polling.

Change from Offlining to Onlining SPOOLING. Meaning that when interrupt, contents of cards read to disk. Therefore current program interrupted.

Multiprogramming Putting multiple programs on at once. Need more memory to do this. Now also need for scheduler to manage multiple users' program needs. Need to figure out how to manage stuff. Priority of jobs, how much time to allocate for these jobs, etc... No memory protection, so programs could overwrite other program's chunk of memory. Java is an example of somehting that doesn't give you direct access to memory in JVM. Memory Protection better done by hardware than having software impose limits.

2 Modes for memory protection User/Restricted Mode Kernel Mode (SU)

Systems works differently in the 2 modes.

Mode bit can be used to signify which mode its in. If do something in User mode, then go to Kernel mode and show exception before going back to User mode.