CS 447 - Operating Systems

Syllabus (facultyweb.cs.wwu.edu/~phil/classes/f26/447)

Assignments

- Toy Operating System, Toy File System

Environment

UNIX (Linux, OS X, NetBSD, FreeBSD ....)

Machine RISCV -- very new architecture (And the MangoPi ... possibly)

Toy OS ... object oriented OS written in C++

Reading the book ... you should do it!

C++ used in Toy OS

C++ is a very large language ... won't be using much of it.

C is supposed to be a subset language. (But not quite!)

Standard ideas used in 347 ... includes, functions, pointers, if, for, while, do, function parameters, ...

Except ... libraries: ToyOS has its own, very few of the "normal" ones:

- Limited version of printf (kprintf inside the kernel)
- some string functions: mem* str* (not the full set, see include/string.h)

C++ features used

Objects: (Should be similar to java ... assuming you have used java objects)

- definition in .h files (mostly) (Well, I use .hxx files.)
- use of private and public members (no protected), friend classes/functions
- use of base classes and inheritance (include/list.hxx,lib/list.cxx, test/list_test.cxx)
  - Test program uses "new" keyword for allocation of objects.
  - Kernel does not use "new", no dynamic allocation of objects.
- use of "inline" functions, full declaration in .h file/Class definition
- object initialization via constructor

Namespace (for lib::...) again, see list.hxx

Call by reference parameters:

- C: void swap (int *x; int *y) { int z; z = *x; *x = *y; *y = z; }
- C++: void swap (int &x, int &y) { int z; z = x; x = y; y = z; }
- 347 C: char *arg_parse(char *line, int *argcptr);
- C++: char *arg_parse(char *line, int &argc);

None of the following things you may have heard about in C++

- templates, standard template library
- IOStream I/O
- Multiple inheritance, other advanced features of C++

You will NOT be an expert on C++ at the end of this class.

Introduction (Chapt 1)

What is an operating system?

- Hardware manager
- Allows "user" (application) programs to utilize the hardware
- Two Views:
  - User view: "Abstract machine"
  - System View: glue between Abstract machine and real machine

Computer systems architecture ...

- CPUs, single, multiple (SMP, parallel, distributed, clusters)
- Memory Hierarchy: registers, cache, main memory, SSD, HardDisk, ...
- I/O Devices: Disks, Tape, USB, video, ...
- Other: interrupts (read book), interrupt driven I/O, timers, ...

Primary Hardware Mechanism

- Dual Mode: supervisor (system, privileged) vs user
- CPU hardware operation state
- User mode provides restriction of use of hardware
- Methods to switch between the two
- Multiprogramming -- Multiple programs in memory at the same time
- Multitasking -- runs multiple programs by switching between them or running at the same time
  - Example: RISCV -- 3 modes: Machine, supervisor, user

Other issues and abstractions

Processes

- Abstraction for code execution
- Uses memory and Dual Mode mechanisms
- Process Management

Other issues

- Memory Management
- File-System and Mass-Storage Management
- Caching
- I/O System Management
- Protection and Security
- Networking, Virtualization and Distributed Systems
- Special Purpose systems: real time, multimedia, handheld
- OS Data structures: lists, stacks, queues, hash tables, bit maps, ...

Other Issues (page 2)

Computing Environments

- Traditional Computing
- Mobile Computing
- Client-Server Computing
- Peer-to-peer Computing
- Cloud Computing
  - Public, Private, Hybrid
- Real-Time Embedded

Open Source Operating Systems

- Linux -- Senior CS Major started it ... (GNU utilities)
- BSD -- Berkeley Computer Science Research Group
- OpenIndiana -- Fork of Open Solaris when Oracle went to Solaris Express
- Plan 9 -- last release in 2015
- ReactOS -- Windows replacement
- Android -- cell/tablet... (Linux at the core)
- Several others ...

Operating System Structures (Ch 2)

Services

- Program Execution
- I/O operations
- File systems -- data storage
- Communications -- (process to process, network ...)
- Error Detection
- Resource Allocation (memory, disk, cpu time, ...)
- Accounting/Logging
- Protection and Security
- User Interface?
  - Some by the OS: e.g. Windows
  - Some not by the OS: UNIX
  - GUI vs Command Line vs Touch Screen

Complete OS distributions have more than OS code

- Kernel: the actual OS itself
- Utilities: User land code to make things work
  - UNIX kernel alone is rather useless!
  - Linux distribution
    - Linux Kernel (essentially same for all distros)
    - GNU utilities
    - Other programs
    - Each distro has a different set/ordering
  - BSD distributions ....
    - Kernel -- unique to the xxxBSD
    - Core Utilities
    - 3rd party Utilities (e.g. NetBSD pkgsrc, FreeBSD Ports)

System Calls

The way a User Process requests services from the kernel

- Syscall API (Application Programming Interface)
- Transition from "User mode" to "kernel mode"
- Controlled entry into the kernel

Types of system calls, see section 2.3.3 for examples

and Windows Vs Unix system calls

- Process Control
- File Management
- Device Management
- Information maintenance (time of day, getpid(),...)
- Communications
- Protection

Library Routines

- Often supplied as part of the Distro/OS
- Integrated into API
- Part of some language .. e.g. C library

Operating System Design and Implementation

Design Goals

- kind of OS -- batch, real time, time sharing, mobile, embedded, parallel ...
- mechanisms and policies --
  - separation
  - policy regardless of mechanism
  - mechanism how to implement policy
- implementation
  - Choice of programming language ...

Operating System General Structures

- Simple Structure (aka the big mess) / Monolithic
- Layered approach (software belongs to a specific layer)
- Microkernel and "servers", message passing
- Modules (aka object oriented)
- Hybrid systems (Mac OS X, iOS, Android)

Operating System Debugging

- fixing errors, performance tuning, removing bottlenecks, ...

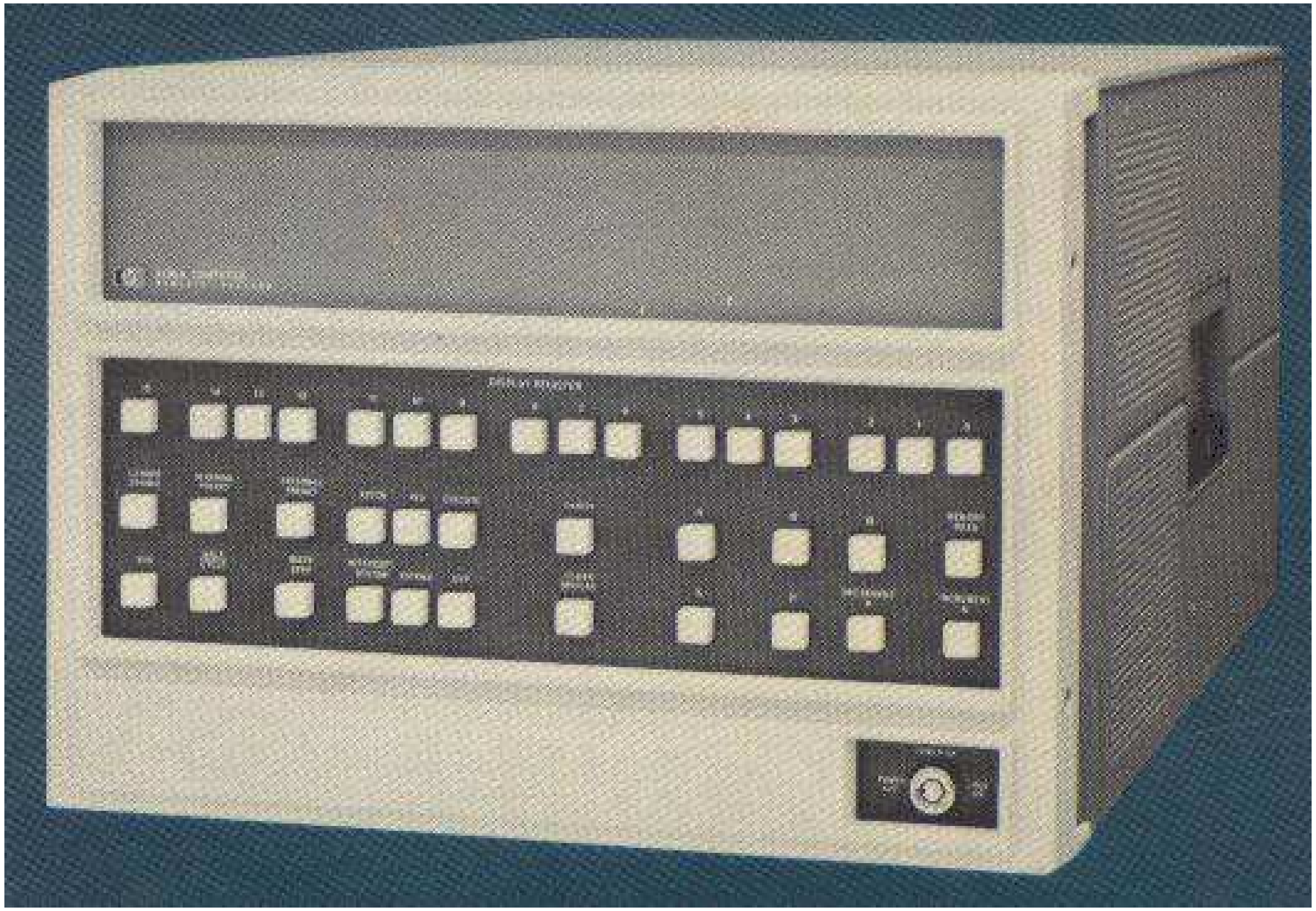
Virtual Machines

- Experiment with OSes (eg Toy OS)
- VM can provided idealized machines
- Abstract VMs (java VM, ...)
- Helps with OS debugging
  - Slow down the machine (possibly)
  - Simulation can make things repeatable

Real OS debugging?

- Windows -- second machine as debugger
- Kernel debuggers
- kernel dumps (blue screen or panic)

System boot ... How?



System boot ... How? (page 2)

- Turn on, enter program via buttons, push "run button"
- Basic Binary Loader, read "paper tape" or "mag tape"

Todays machines -- Boot process

- Power up starts running at location 0.
- Hardware typically maps a ROM image at location 0
  - Or forces a jump to a ROM image
- ROM has "machine monitor" program
- May look for OS/Boot code on a Disk, net, ...

- Qemu / RISCV
  - qemu-system-riscv64 -kernel kernel-file -machine virt ....
  - Starts execution at 0x1000
  - ... and a "ROM" does an immediate jump to 0x80000000
- MangoPi -- single board RISCV computer
  - ROM does a jump to 0x40000000
  - xfel -- not run on MangoPi
    - program to talk to MangoPi ROM