Course Review

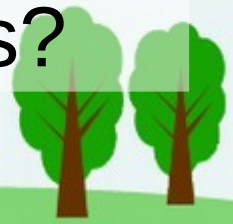
- operating systems overview
- interfaces: APIs, shells, window systems
- processes and threads, scheduling, interrupts, IPC and synchronization, context switches
- exec, fork, memory management, signals
- I/O and device drivers
- file systems
- security
- operating system structure: distributed, cluster, and grid computing
- projects
- discussion

Operating Systems Overview

- why do we need operating systems?
  - to control and give us structured ("machine independent") access to the hardware
  - to allow "virtual processors" (processes) to cooperate but not harm each other
  - to run virtual machines independently of each other
- "operating system" always includes the kernel, and may or may not include:
  - the programs (applications) available under an operating system
  - the device drivers available under an operating system
- there is a large variety of operating systems, some mainstream, most experimental or otherwise limited

OS Interfaces

- APIs: programmer access to OS services
- device driver API: programmer access required to support specific hardware
- shells: user access to OS services and programs
- window systems: visual shells (including browsers)
- how much should the OS be modified to support specific shells and window systems?



Processes and Threads

- processes have separate resources, esp. address spaces
- threads within a process have shared global resources, esp. address spaces, file descriptors, perhaps signal masks
- linux **task** can be a classical process or a classical thread, different resources may or may not be shared

Scheduling

- non-real-time tasks:
  - round-robin
  - priority queues
  - many more that vary in the details
- real-time tasks:
  - earliest-deadline first
    - works for periodic and non-periodic tasks
  - rate monotonic

Synchronization

- interrupts (asynchronous creation of new threads)
- inter-process communication (IPC): shared memory, message passing (including pipes), issues of race conditions, deadlock prevention and detection, synchronization
- locks (mutex variables, atomic test-and-set), semaphores
- context switches:
  - what is the context? virtual memory mappings, process entry, CPU registers
  - where is the context saved? TLB is already backed by page table, process table entry is switched by changing a pointer, CPU registers may be saved on the stack or in the process table entry, stack pointer must be changed
  - how is the context restored? in Minix as a result of returning from an interrupt, other times through careful assembly language programming

Implementation of Processes

- Exec: bring image from file system into memory, address space may have to be changed to fit the new image, context is set up for the initial "restore"
- Fork: memory management sets up new mapping, file system may need to copy (or share) file descriptors
- in Minix, both of these require the kernel to perform basic operations, e.g. starting execution of the new process and filling in the new memory map
- memory management: paging, segmentation, protection, page fault, page cache, file system buffer cache, memory management algorithms including first fit and next fit
- signals as interrupt analogs, and operating system implementation of signals

I/O

- I/O devices: the hardware matters, the controller provides the interface
- device drivers: provide the OS with access to the device, and service interrupts from the device with service
- DMA
- clocks
- disks, including floppy disk, hard disk, CD, and RAM disk
- terminals and bitmapped displays
- networking devices

File Systems

- structure and organization:
  - (mostly) hierarchical
  - links,
  - symbolic links,
  - volumes (disks),
  - remote volumes (network file systems)
- implementation:
  - directories stored as files,
  - inodes (sometimes),
  - attributes (dates, permissions, ownership),
  - block indices and index blocks,
  - buffer cache and caching strategies
  - Virtual File System layer

Security

- protection among processes
- controlled switching between processes
- controlled sharing among processes
- human factors in security
- network attacks

Operating System Structure

- microkernels
- distributed, cluster, and grid computing
- embedded systems

Projects

- project 1: shells, APIs
- project 2: add a system call
- project 3: interrupt handling and context switches
- project 4: new disk device
- project 5: design your project
- project 6: implement your project
- all projects: hands on familiarity with operating system concepts, "getting your hands dirty"

Discussion

- market concerns: who wins? who makes the most money?
- scope: what should be part of the operating system? Is efficiency a concern? the only concern?
- programming languages: is it good to use something other than C? If so, what?
- how important is simplicity as opposed to completeness? for what applications?
- what do you know about the OS you will be using in the year 2020? 2030? 2040?
- any other questions