CSC 369H1S
Operating Systems
Spring 2007
Professor Angela Demke Brown
U of T

Administrivia

- Instructor Contact:
  - Email: demke369@cs.toronto.edu
  - Office: BA 4266
  - Office Hours: Tuesdays, 4-5p.m. and Thursdays 4-5p.m.
- Newsgroup:
  - news://newssrv.cdf.utoronto.ca/ut.cdf.csc369h
- Webpage:
- Course Info Sheet (due dates, policies, etc.):
  - .../369S.07/Handouts/info.pdf

Course Overview

- Objective: To understand the role of the OS, the major subsystems that make up a modern OS, and the design principles and implementations behind them; To understand concurrency fully
- How are we going to get there?
  - Lectures, textbook readings, handouts
  - Tutorial sections (discussion, review, questions, etc.)
  - Simulated OS development project (OS/161)

Assignments (40%)

- Code Reading component:
  - Questions about the code we give you
  - Reinforce lecture concepts, familiarize you with the code before you start writing
- Programming component
  - Language for this course is C
  - Correctness and performance considerations
- Design Document component
  - Think about what you need to build before you start building it.
  - Tell us in plain English what you were thinking.
  - Tell us how to find your code.
## Workload

- This course is very work-intensive
- Lectures cover a lot of new concepts
  - Many of these are very abstract
  - Only way to really understand how it works is to try doing it yourself
- Assignments build on a realistic OS
  - There is a lot of existing code given to you
  - You should not expect to understand all of it
  - You should be comfortable with prereq material
    - Unix tools (cvs, debugger, scripts)
    - Computer organization, memory model, etc.
    - C programming (especially pointers!)

## Academic Dishonesty

- Plagiarism and cheating
  - Very serious academic offences
  - Can discuss OS161, tools, concepts with your classmates
  - Can discuss solutions to the assignments with your partner(s) only!
    - There is a clear distinction between collaboration and cheating.
  - All potential cases will be investigated fully

## Wrong & Right Examples

- Use the newsgroup to ask questions, but don’t discuss your solution or approach.
  - **Wrong:** “I want to track how long a process has been running (so I can decrement its priority periodically) by calling `gettime()` in `hardclock()`. I record the time a process starts running in its process control structure, and then in `hardclock()` I can get the interval between when it started and the current time, and then set a new priority. But when I put a call to `gettime()` in `hardclock()`, the kernel hangs without printing anything to the console. Has anyone else seen this problem? Can someone explain what is going on?”
  - **Right:** “When I put a call to `gettime()` in `hardclock()`, the kernel hangs without printing anything to the console. Has anyone else seen this problem? Can someone explain what is going on?”

## Exams

- Midterm (20%)
  - See info sheet for date
  - Covers material up to end of Week 6
- Final (40%)
  - Will be scheduled by Arts & Sciences
  - Will be cumulative
  - I will be explicit about what you are responsible for on the final
  - Final exams from 369 will be made available for study
Course Content

- Tentative list of topics, in order:
  - Introduction (Today)
  - Processes and Threads
  - Concurrency (Synchronization & Deadlock)
  - CPU Scheduling
  - Memory Management
  - File and I/O Systems
  - Introduction to Distributed Systems
  - Security

Introduction

- Read Chapter 1
  - Some (much?) of this should be review

What is an operating system?

- The layer of software between user applications and the hardware

Other views of the OS

- An OS is a resource allocator
  - allows the proper use of resources (hardware, software, data) in the operation of the computer system
  - provides an environment within which other programs can do useful work
- An OS is a control program
  - controls the execution of user programs to prevent errors and improper use of the computer
  - especially concerned with the operation and control of I/O devices
Goals & Roles of an OS

• Primary goal: convenience for the user
  • An OS is supposed to make it easier to compute with it than without it
• Secondary goal: efficient operation of the computer system
  • The two goals are sometimes contradictory
  • Which goal takes precedence depends on the purpose of the computer system
• OS and computer architecture have mutually influenced each other

Short Historical Review

• Simple batch systems
  • IBSYS - monitor concept (1950s)
• Multiprogrammed batch systems
  • IBM's OS/360 (early 60s)
• Time-sharing systems
  • MIT's CTSS (1962)
• Personal computer systems
• Parallel systems
• Distributed systems
• Real-time systems

Review: Computer System Structures

Modern Computer: A CPU and a # of device controllers, connected through a common bus, providing access to shared memory

Storage Structure

• Main memory (DRAM) stores programs and data during program execution
  • DRAM cannot store these permanently because it is too small & it is a volatile storage device
  • Auxiliary memory: hold large quantities of data (including programs) permanently
• Main memory is only storage (other than registers) CPU can access directly
  • Forms a large array of bytes (1 byte = 8 bits) of memory, each with its own address
  • We say that main memory is byte-addressable
Aside: C Programming & Memory

"C combines all the power of assembly language with all the ease of use of assembly language." -Unknown

- A variable in a C program is a symbolic name for a data item, stored in memory
  - The type of the variable indicates how much storage (how many bytes) it needs
  - Type also determines alignment requirements
  - The address of the variable is an index into the big array of memory words where the data item is stored
  - The value of the variable is the actual contents of memory at that location
- A pointer type variable is just a data item whose contents are a memory location (usually the address of another var)

C Example

```c
int main(){
    char a = 'h';
    int b = 0xdeadbeef;
    char *c = &a;
    int *d = &b;
    printf("b=%d (0x%x)\n", b, b);
}
```

Memory

- char data type is 1 byte in size
- int data type is 1 word in size (32 bits for most current architectures)
  - occupies 4 bytes, should be word-aligned
- pointer types are all 1 word in size
- endian issues for multi-byte types

Storage Hierarchy

- processor registers, main memory, and auxiliary memory form a rudimentary memory hierarchy
- the hierarchy can be classified according to memory speed, cost, and volatility
- caches can be installed to hide performance differences when there is a large access-time gap between two levels

Caching

- when the processor accesses info at some level of the storage hierarchy, that info may be copied to a cache memory closer to the processor, on a temporary basis
  - a cache is smaller and costlier
- because caches have limited sizes, cache management is an important design problem
  - Coherency
  - Consistency
Concurrency

- Every modern computer is a multiprocessor
  - CPU and device controllers can execute concurrently, competing for memory cycles
  - A memory controller synchronizes access to shared memory
  - Interrupts allow device controllers to signal the CPU that some event has occurred (e.g., disk I/O complete, network packet arrived, etc.)
  - Generated by a hardware device
  - Interrupts are also used to signal errors (e.g., division by zero) or requests for OS service from a user program (a system call)
  - These types of interrupts are called traps or exceptions

An Operating System is an event-driven program

Protection Domains

- Dual-mode operation: user mode and system mode (a.k.a. supervisor mode, monitor mode, or privileged mode)
  - Intel actually has 4 “rings” for protection
- Add a mode bit to the hardware and designate some instructions as privileged instructions
- Protect the operating system from access by user programs, and protect user programs from each other
- What instructions/operations would you expect to be privileged?

Hardware Support for OSs

- Protection domains → mode bit
- Memory Management unit
- Interrupts
- Timers
- Other hardware

Bootstrapping

- Hardware stores small program in non-volatile memory
  - BIOS – Basic Input Output System
    - Knows how to access simple hardware devices
    - Disk, keyboard, display
- When power is first supplied, this program starts executing
- What does it do?
Operating System Startup

- Machine starts in system mode, so kernel code can execute immediately
- OS initialization:
  - Initialize internal data structures
  - Machine dependent operations are typically done first
  - Create first process
  - Switch mode to user and start running first process
  - Wait for something to happen
  - OS is entirely driven by external events

Requesting OS Services

- Operating System and user programs are isolated from each other
- But OS provides service to user programs...
- So, how do they communicate?

Memory Layout (Linux, x86)

- Code (Text Segment)
- Static Data (Data Segment)
- Heap (created at runtime by malloc)
- Unused

Boundary Crossings

- Getting to kernel mode
  - Boot time (not really a crossing, starts in kernel)
  - Explicit system call - request for service by application
  - Hardware interrupt
  - Software trap or exception
  - Hardware has table of "Interrupt service routines"
- Kernel to user
  - OS sets up registers, MMU, mode for application
  - Jumps to next application instruction
System Call Interface

- User program calls C library function with arguments
- C library function arranges to pass arguments to OS, including a system call identifier
- Executes special instruction to trap to system mode
  - Interrupt/trap vector transfers control to a system call handling routine
- Syscall handler figures out which system call is needed and calls a routine for that operation
- How does this differ from a normal C language function call? Why is it done this way?

System Call Operation

- Kernel must verify arguments that it is passed
  - Why?
- A fixed number of arguments can be passed in registers
  - Often pass the address of a user buffer containing data (e.g., for write())
  - Kernel must copy data from user space into its own buffers
- Result of system call is returned in register

OS/161 System Calls

- User level C library end of system call interface is prototyped (mostly) in src/include/unistd.h
- Numeric codes for system calls are listed in src/kern/include/kern/callno.h
  - OS defines available system calls; this header is copied to user include space during system installation
- Actual C library functions are generated from callno.h using "callno-parse.sh" and "syscall-mips.S" in src/lib/libc/
- System call handler is in src/kern/arch/mips/mips/sySCALL.C mips_syscall()

Operating System Structures

- (process, main memory, file, I/O system, auxiliary storage, network) management
- Taxonomy:
  - Modules - specific OS services and abstractions
  - Interfaces - specific operations provided by modules
  - Structures - the way modules are connected
- distinction: mechanism (how to do something) and policy (what will be done)
  - E.g. "How do I guarantee the kernel gets control back from a user program?" vs. "What amount of time should I let user processes run?"
Ways to build an OS: Monolithic

- All code associated with the operating system executes as part of the "kernel" and is privileged
  - E.g., Microsoft Windows, BSD Unix, Linux, OS161

Ways to build an OS: Open Systems

- Applications, libraries, kernel all in the same address space
- Crazy?
  - MS-DOS, MAC OS 9 and earlier, Windows ME, 98, etc., PalmOS

Ways to build an OS: Microkernels

- Minimize privileged code; most OS function is implemented as user-level servers that communicate with the kernel
  - Mach, Apple OS X (sort of), L4, QNX

Process Concept

- process = job
- "definition": a process is a program in execution (an active entity)
  - How does a C program become a process?
Process states & state changes

- The OS manages processes by keeping track of their state
- Different events cause changes to a process state, which the OS must record/implement

Next time...

- Read about processes & threads (Chapter 2.1-2.2)
- Get the OS/161 distribution and get your account set up (see web page)
  - Assignment 0 (Warm-Up) will be out by noon tomorrow