Introduction

Tom Kelliher, CS 325

Jan. 28, 2009

1 Administrivia

Announcements

Assignment

Read Chapter 1.

Outline

1. Syllabus.
2. A "grand tour:" OS and system views, structure, and operation.

Coming Up

Continued "grand tour."

2 Syllabus

1. Objectives:
   (a) Study operating system *design*.

(b) Understand threads and concurrency: Banking example.

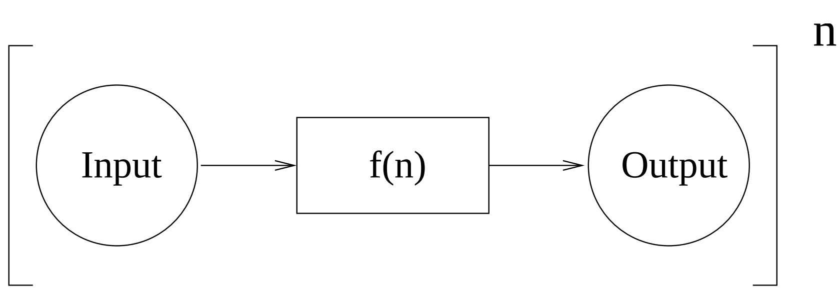
(c) Appreciate connections to other areas of computer science.

2. C refresher project.
3. Internet resources.
4. Linux internals project orientation.
5. Class preparation.
6. (Doubtful) Possibilities for *Other topics*: deadlock, distributed systems, security and protection.

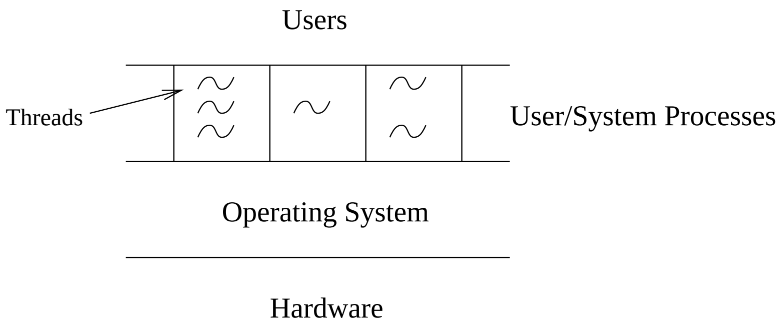
3 A Grand Tour

3.1 The Main Thing

An OS's responsibilities boil down to managing:



3.2 OS As Interface



(Process = running program. Separate address spaces. Threads share an address space.)

1. Top-down view: virtual machine abstraction — convenient "user" interface. Abstractions: files, applications. I/O devices integrated into filesystem.

2. Bottom-up view: management of real resources: CPU cycles, memory, disk space, device allocations.

3. Secondary concerns: efficiency, fairness.

Abstractions:

1. Multiprogramming, protection and security.

   Threads.

2. Virtual memory.

3. File systems.

4. Virtualization.

3.3 OS Components

1. Kernel. Static. Process/thread, memory, I/O management and access.

   Timers.

2. Daemons. Provide additional services.
3. System applications: compilers, linkers, loaders.
4. User applications: shells, windowing systems, browsers, editors, etc.

The "Hello world" program:

1. *Compiled* into assembly code.
2. *Assembled* in machine code.
3. *Written* to a file.
4. *Loaded* into memory.
5. *Linked* against system libraries.
6. *Executes*.
7. Makes *supervisor calls* to access I/O devices through OS.

3.4 Computing System Organization

1. CPU, memory, I/O devices block diagram.

   I/O device bandwidth/latency differences.

   Process execution within this context. Data/code locality: caches. VM. DMA.

2. Single CPU, multiple CPU chips, multiple cores, hyperthreading. (Phoenix: eight "CPUs.")

   Why can't we just turn up the clock?

   Efficiencies with multiple cores vs. multiple CPU chips.

   Programming consequences.

3. Process, do I/O, repeat model.

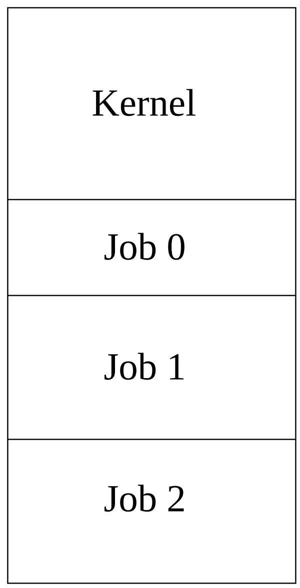
4. Memory hierarchy. Speed, density, cost, volatility.

5. I/O architecture. Abstract models, device drivers, devices. Plug'n'play/pray.

3.5 OS Structure

1. Multiprogramming.

   Mental memory model:



   Timesharing vs. batch.

   Short-term, long-term schedulers.

   Physical, virtual memory. Swapping.

2. Interrupt driven kernel operation.

   Dual (or more) CPU modes: user, kernel modes. Privileged operations and/or I/O spaces.

   Interrupts, traps. Hardware timers.