PRINCIPLES OF COMPUTER SYSTEMS DESIGN

CSE130

Winter 2020

History

Baskin Engineering
UC SANTA CRUZ

Notices

- **Administration 1 & 2** due 23:59 **Wednesday January 15**
- **Lab 1** due 23:59 **Sunday January 19 CHANGED!**
- **NO LECTURE Friday January 10**

Today's lecture

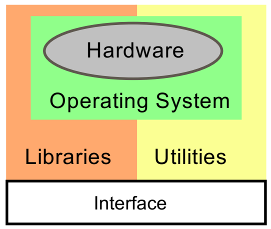
- Brief History of Computer Systems
- Structure of the course
- Lab 1 "Secret Sauce"

What is a Modern Operating System?

- The GUI is not the OS, it's just the visible part
- We are interested in the basic engineering principles underlying the most used computer systems
- **The OS is the resource manager for a computer**
  - Controls and arbitrates access to hardware:
    - **CPU**
    - **Memory**
    - **I/O Devices**
  - It provides high level abstractions to application programmers for hardware/device control
  - It provides services for multiprogramming (usually)

Basic Computer System Architecture

- Hardware:
  - CPU
  - Memory
  - I/O Devices
- Operating System:
  - Coordinates use of resources amongst applications and users
- Libraries and Utilities:
  - Provide standard functionality



Operating System goals

- **Efficient use of Resources**
  - Maximize utilization of resources
  - Avoid bottlenecks
- **User Convenience and Productivity**
  - User time is expensive
- **Availability and Reliability**
  - Many systems are critical
  - Data is valuable

Efficiency
Convenience
Availability
Reliability

Design Tradeoffs

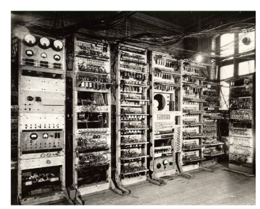
- Cannot simultaneously optimize **efficiency** *and* **user convenience**
- Choice depends on context...

Efficient — Convenient

Super Computer	Personal Computer
expensive hardware	cheap hardware
optimize *utilization*	optimize *response*

Evolution of Computer System

- 70+ years of development
  - Simple, single-purpose systems to all-singing, all-dancing hand-held devices, via general-purpose super computers



Manchester Mark I (1946 - 1949)
OS: Humans
Speed: 1,500 instructions / second

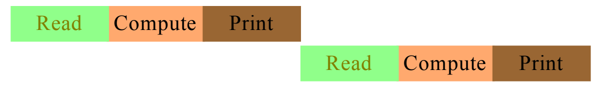
Blue Waters (2007 - Date)
OS: Cray Linux
Speed: 13,300,000,000,000,000 flops

Batching Systems

- With early machines the operators had to manually set up the runtime context for each program - a labor intensive task
  - To reduce the load on the (human) operator, jobs were sorted into batches with similar runtime requirements
  - Setting up the environment for a new batch required considerable effort and time so the human operators made little devices or wrote little utilities to make life easier for themselves
- **Operating Systems started as batch job control systems**

Batching Systems

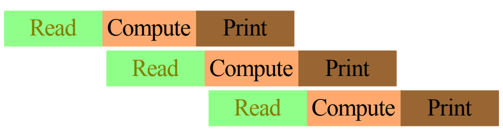
- These original Operating Systems were simple control programs that automatically worked through individual jobs in a batch (sequence of jobs)



- IO (read/print) slow, CPU fast(er) => under utilization

Spooled Batching

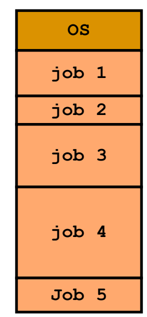
- Next increment was spooling batch systems
  - This is the same idea as CPU instruction pipelining



- Requires hardware interrupts (more on these later)
- IO still slow, CPU still fast(er) => a bit better

Multi-Programmed Batching

- Multi-programming allows multiple jobs (programs) to be resident (in memory) at the same time, all sharing a single CPU
- When one program is blocked on IO, we simply pick another and execute it
- As long as at least one job remains ready to execute, the CPU is 'fully' utilized

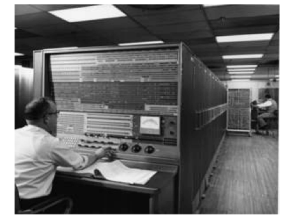


Multi-Programmed Batching

- The start of sophisticated Operating Systems
  - Job scheduling (which job in batch is CPU resident)
  - Memory management (and protection)
  - Requires CPU scheduling (fairness between jobs)
- Still batch though, with human controllers (operators) and no direct user interaction
- Very efficient, high hardware utilization

The Stretch

- In November 1956, IBM set out to build a computer 100 x faster than its current 704 model - a lofty goal of 4 million instructions a second (MIPS)
- The 7030 was IBM's first transistorized computer and pioneered many advanced computing concepts:
  - **Interrupts**
  - Memory Interleaving
  - **Memory Protection**
  - Memory Error Detection and Correction
  - **Multiprogramming**
  - Pipelining
  - Immediate operands
  - Instruction pre-fetch
  - Speculative execution, result forwarding
- Only achieved ~30x the 704 (1.2 MIPS)
- Fastest computer in the world for 4 years
- First machine shipped to Los Alamos in 1961



Time Sharing Systems

- Natural evolution from multi-programmed systems
- Add user interaction
  - Switch between user programs frequently, so users can interact with each program
  - **N** people, so each person effectively has **1/N** of the computer's time
    - **Or do they?**
  - Many users sharing same machine - IO blocks on user response and program instances are switched by the OS
- Resulted in a significant change in OS complexity
  - There were many, many failures in the development of time sharing systems ¯_(ツ)_/¯
- **Time Sharing ⇒ Multiprogramming**
- **Multiprogramming $\not\Rightarrow$ Time Sharing**

The CDC 6600

- Considered the first supercomputer, delivered 1964
- Fastest computer in the world until 1969
  - 10 MIPs @ 10 MHz
- 10 superscalar functional units in the CPU
- Introduced a new OS
  - Simultaneous Processing Operating System
- SPOS was a failure ☹
  - Reverted to a simple Job Control

Cray Supercomputers

- CDC 7600 Replaced the CDC 6600 in 1969
- Fastest Super Computer until 1975
  - 10 to 36 MIPS @ 36 MHz
- First functional timesharing OS
  - Livermore Time Sharing System (LTSS)

- Seymour Cray left CDC in 1972 and formed Cray Research
- Cray-1 launched in 1976
  - 160 MIPs @ 80 MHz
- Cray Time Sharing System (CTSS) derived from LTSS on the CDC 7600

Personal Computers

Apple 1 (1976)

Commodore Pet (1977)

IBM 5150 (1981)

Sharp MZ80K (1978)

Personal Computers

- Paradigm shift
  - The user is now most important
  - Hardware is low cost
- Goal is convenience and responsiveness, not hardware utilization
- Did not initially perform multitasking or handle multiple simultaneously connected users
- As power increased, these (and many other) features migrated into the mainstream
- Modern PCs now provide multiple processing units in a shared memory architecture

Cluster Computing

- Many cheap commodity computers
- Clustered: Linked with a high speed LAN
- MIMD Parallelism
  - Multi Instruction, Multi Data
- Beowulf cluster – Linux
- Can be for availability, performance, or both (load balanced)

Distributed Computer Systems

- A loosely coupled collection of processors:
  - Do not share clock or memory
  - Do share a communications network
  - Heterogeneous
  - Permit sharing of resources (CPU, files, …)
  - Issues include:
    - load balancing
    - process migration
    - distributed transactions
    - security
    - communications
    - reliability

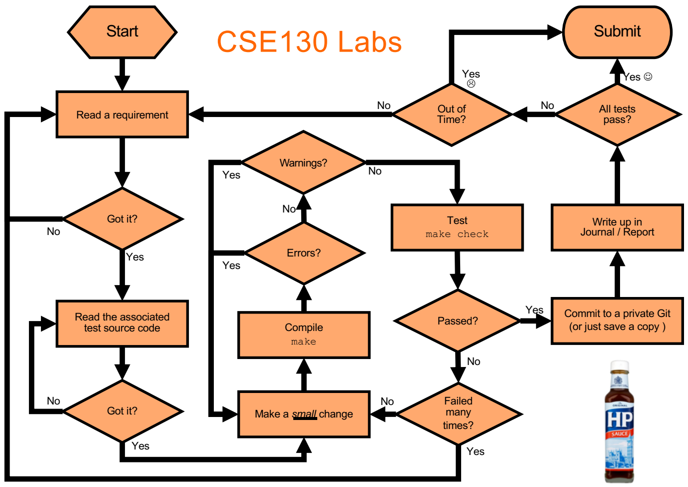
Structure of the course

- Five areas:
  - Processes, Threads, and Concurrency
  - CPU Scheduling
  - Memory Management
  - File Systems and Mass Storage
  - Miscellaneous
    - Virtualisation, Security, History, etc.
- Final examination has five questions, one for each area

What is Secret Sauce?

- General advice
- Extra detail on labs
- Video explanations
- Step-by-step guides
- Pseudo code
- After-the-fact feedback

CSE130 Labs



What do you need to do for Lab 1?

- In the `timer_sleep()` function:
  - Instead of busy-waiting, put the current thread to sleep and immediately return
- At regular points in the future:
  - Wake up sleeping threads at or past their wakeup time

The `timer_wait()` function

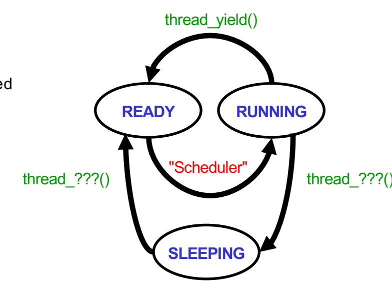
```
/*
 * Sleeps for approximately TICKS timer ticks.
 * Interrupts must be turned on.
 */
void
timer_sleep(int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT(intr_get_level () == INTR_ON);

  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```


Pintos Thread States & Transitions

READY
Waiting to run on the CPU

RUNNING
Binary instructions are being executed

SLEEPING
Waiting until it's time to wakeup



Questions to ask yourself (not exhaustive)

- About Pintos
  - Where are timer functions defined / implemented?
  - What is the C type of a thread?
  - How do I get a reference to the current thread?
  - How do I put a thread to sleep?
  - How do I wake a thread up again?
  - When do I check if anything needs waking up?
    - i.e. what function gets called when the timer interrupt fires? (massive hint)
  - How do I turn interrupts on and off?
- About your solution
  - How might I store references to threads I want to wake?
  - How do I store the wakeup times?
  - Would my solution handle 1,000's of threads?

Next Lecture

- **NO LECTURE Friday January 10**
- Monday: Processes & Threads I