CSE 451: Operating Systems
Spring 2020

Module -2
Learning to Walk

First of all

- Hi, I’m John
  - This is my dining room
- The goal of the course this quarter is the same as every quarter
  - Maximize useful learning/experiences per hour of your time spent
- We have unprecedented challenges
- We have unprecedented constraints
  - Constraints are the mother of invention
- Some things will be as always
  - lecture material (after this week)
  - projects (xk)
  - working in pairs
  - grading
  - exams (except we will have a midterm and will have a final)

Constraints

- First of all, absolutely no in person meetings
  - staff / students
  - staff / staff
  - students / students
- Second, we’re not even allowed on campus unless we can demonstrate some critical reason to be there
- So, all interactions are remote
  - classes, sections, office hours, exams, working on projects
- We’re not sure how to do all that

Classes This Week

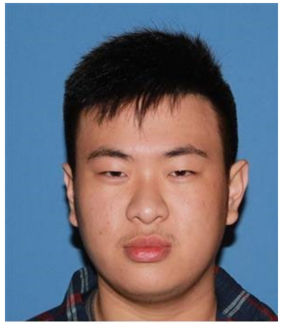
We’re required to spend the first week of classes figuring out how to make this work

- We cannot assign work to be graded
- We cannot present material that will be relevant to work that will be graded
- We must “meet”

- Some preliminary plans
  - Classes: Zoom
  - Sections: Zoom
  - Office Hours: Zoom (jz), ? (TAs)

Not A Course Intro (because that may not be allowed)

Rithik Duggempudi



Jonathan Jusuf

Nicolas Monsees

Brennan Stein

Haopeng Zhou

Not A Course Intro

- CSE 451 has two content streams
  - lectures -- we’ll be roughly following the text
  - sections / projects – not even roughly following the text
- We’ll be doing some of the traditional xk projects
  - We won’t start until next week
- We’ll be working in teams of two
- We’ll be having online exams

Departmental Computing Resources

- The same as always
  - except that you can’t go into the labs to use them
- New! Linux with remote desktop (rather than X forwarding): https://vdi.cs.washington.edu/
- There is an optional exercise for this week...
  - Not graded
  - Hopefully interesting

Using Zoom

- Is this format (screen sharing with talking head) effective?

- Can you raise your hand?
  - If you do, will I notice?

- Chat messages?
  - Private / public

- Can I manage breakout rooms?
  - Can you interact in them?

Performance

- The XX (the thing I'm not allowed to talk about until next week) has many jobs
  - For instance, x, y, z
- We require the XX to be correct
  - No crashing
- We require the XX to be secure
  - No undesired behaviors by anyone, including me
- We'd like the XX to be low overhead
- We'd like the XX to not interfere with good application performance

Static vs Dynamic Application Analysis

- “Static” means when the code isn’t running
  - The compiler has a static view of the application’s code
- “Dynamic” means when it is running
  - The XX, the runtime libraries, the app, and other apps (e.g., services) have a dynamic view of the application
  - (The CPU hardware has a dynamic view as well)

- Static can see all the code, and all possible paths in it
  - Can reason about code behavior and possibly apply powerful optimizations
- Dynamic sees which paths are actually being used
  - Can adapt to what the code is actually doing

Application (including XX) Performance

- What are the factors that influence the running time of an application?

  - 1) Algorithm/asymptotic running time
    - The OS can't rewrite your app to use a more efficient algorithm

  - 2) Code path length
    - Again, the OS can't really do much about this for the apps it runs
      - those apps can use an optimizing compiler
    - The implementer of the XX can try to optimize code its paths
      - Can involve a tradeoff in choosing abstractions for the OS to implement

Application (including XX) Performance (cont.)

- What are the factors that influence the running time of an application?
  - 3) Hardware:
    - CPU implementation
      - Instruction processing rate
        - For example, how many instructions can be executed at once?
    - Memory hierarchy
      - Sizes, organizations, and locations of caches
    - I/O
      - What can device do, what does OS need to do?
      - Number of simultaneous operations?
  - 4) Hardware/software interactions
    - Multi-core hw with threaded app
    - Program locality

Application (including XX) Performance (cont.)

- What are the factors that influence the running time of an application?
  - 5) XX/application interactions:
    - Program packaging
      - Processes vs. threads
        - Inter-process communication vs. thread synchronization
      - Single machine vs. distributed
    - Use of XX functions
      - Memory intensive?
      - I/O intensive?
      - Thread intensive?

How expensive/important are the following?

- This is a pretty arbitrarily chosen set...
  - Loop control overhead
  - Procedure call overhead
    - Overhead as a function of number of arguments passed
  - Memory locality
    - Good temporal
    - Good spatial
    - Predictable stride
    - Random
  - Multi-threaded execution memory effects
    - As a function of number of cores

How expensive/important are the following?

- This is a pretty arbitrarily chosen set... (cont.)
  - System calls
    - Overhead to enter/exit the XX
    - open/close a file without app layer buffering
    - open/close a file with app layer buffering
    - create a new process (fork only) and wait for it to terminate
    - create/join a new thread

Any guesses?

Function	Time (nsec.)
Loop iteration	?
Null procedure call	?
8 argument procedure call	?
good locality / bad locality ratio	? / ?
null syscall	?
file open	?
process create	?
thread create	?

Measurement

- I’ve written some code that tries to measure some of these things
- Measurement is really hard!
  - My code could have bugs
  - It could be measuring something different than I thought (say because the hardware acts differently than I thought)
  - It could be measuring something different than I thought (because the compiler produced much different code than I thought)
  - Very counter-intuitive results could be right, they could be wrong
  - I may be measuring the wrong things
    - What are the interesting things to measure?

Optional Exercise for This Week

- Fetch my code and do one or more of the following
  - Figure out how to build an application from it
    - $ gcc *.c *will get a build error. Figure out why and fix it.*
  - Run the tests and examine the results
    - Are they more or less in line with what you expected?
    - Try running on different (kinds of) computers. How much does the hardware platform affect the relative results (what's fast and what's slow and by how much)?
    - Try building with optimization on *($ gcc –O2 *.c...)* and run them again
      - What changes? Why?
  - Think of something interesting to measure and add code to measure it

The Measurement Code

- It’s in gitlab:

```
$ git clone git@gitlab.cs.washington.edu:zahorjan/cse451-20wi-distributables.git
```

- Let’s have a brief look at the code
- Let’s run it on my office desktop

Another thing to try...

- Linux includes a utility, strace, that traces all system calls made by a process
  - Based on the ptrace system call facility
- You can use it to get an idea of:
  - how frequently system calls are being made (by an individual process)
  - which system calls are common (at least for that process)

strace Example

- *$ time strace /usr/bin/google-chrome*
  - Manually killed when chrome appeared on the screen
  - Elapsed time: 3.14 seconds
  - 11,155 system calls / second

14646 recvmsg	544 fstat	41 dup	11 wait4	4 arch	2 shmdt	1 setsockopt
3118 poll	345 fstatfs	33 unlink	10 getgid	4 getresuid	2 set	1 nanosleep
2405 futex	309 writev	32 uname	9 socketpair	4 brk	2 sched	1 clock
2268 read	298 readlink	32 getdents64	8 getpriority	4 symlink	2 inotify	1 exit
1445 openat	244 munmap	28 lseek	7 pipe	4 getresgid	2 mkdir	1 bind
1421 stat	197 fcntl	27 ftruncate	7 socket	3 lstat	2 gettid	1 rmdir
1382 sendto	175 mprotect	27 fallocate	7 prlimit64	3 shmctl	2 creat	1 prctl
1374 madvise	161 fadvise64	26 clone	7 ioctl	3 shmat	2 getsockname	1 getppid
1151 close	111 getrandom	14 recvfrom	6 eventfd2	3 shmget	2 shutdown	1 getpgrp
1010 write	107 rt	12 getuid	6 dup2	3 getpeername	2 setpriority	
862 mmap	101 sendmsg	12 geteuid	5 connect	3 sysinfo	1 rename	
796 access	90 getpid	11 getegid	4 statfs	2 execve	1 listen	