Okay, human.

Huh?

Before you hit compile; listen up.

You know when you’re falling asleep, and you imagine yourself walking or something.

And suddenly you misstep, stumble, and jolt awake?

Yeah!

Well, that’s what a segfault feels like.

Double-check your damn pointers, okay?

http://xkcd.com
CSC209: Software tools ...

- Unix
  - files and directories
  - permissions
  - utilities/commands

- Shell
  - programming
  - quoting
  - wild cards
  - files
... and programming in C ...

- C
  - basic syntax
  - types
  - arrays
  - pointers
  - functions
  - strings
  - structs
  - header files
...and UNIX system programming

- System calls
- Files
- Processes (fork, exec)
- Inter-process Communication
  - signals
  - pipes
  - sockets
  - select
- Concurrency and threads
What can you do?

- Write a shell script to automate tasks
  - Run some tests multiple times (e.g., automarkers 😊)
- Write a program to run and monitor other programs
  - Kill them if they take too long
- Write a program that splits tasks into multiple processes to take advantage of multiple cores
- Use a Makefile to build a large system
What else?

- Write a shell!
- Write a web server!

- But more importantly, you can begin to understand what happens when
  - A program “hangs”
  - A program “crashes”
  - Two programs share the same file
  - A process has terminated but is still in the process table (zombie process)
Course Topics

- Topics covered
  - Part I: Unix, Shell & Shell Programming
  - Part II: Programming in C
  - Part III: UNIX System Programming
Part I

Unix, Shell & Shell Programming
Unix Philosophy

- Write programs that do one thing well
- Write programs that work together
- Write programs to handle text streams because that is the universal interface
File interface

- “Everything is a file”
- We treat all sorts of devices as if they were files, and use the file interface (open, read, write, close) all over the place
  - files
  - directories
  - pipes
  - sockets
  - kernel info via /proc (interface to kernel data structures)
Shell Concepts

- **Stdin, stdout, stderr**

- **I/O redirection (>, <, ...)**
  - `ls -l > contents.txt`

- **Process Control (ps, kill,...)**
  - `ps aux | grep steven` (check all processes for user “steven”)
  - `kill -INT 1049` (send SIGINT signal to process 1049)

- **Job control (fg, bg, %, &, ...)**
  - Send job to background (&), etc.

- **Pipes (|)**
  - `ls -l | wc -l`
Bourne shell programming

- **quoting**
  - single quotes (' ... ') inhibit wildcard replacement, variable substitution and command substitution
  - double quotes (" ... ") inhibit wildcard replacement only
  - back quotes (` ... `) cause command substitution

- **variables – environment and local**
  - str1="string"
  - str2="string"
  - if test $str1 = $str2; then ... fi
Bourne shell programming

- `test -f filename` - test if a file exists
  - Or use `[ ]` notation
  - `test -d dirname` - test if a directory exists

- Command line arguments
  - `$0 = name of script, $1 .. $n = arguments`

- `set` assigns positional parameters to a list of words

- `read` - reads from stdin

- `expr` - math functions

- Programming language structural constructs
  - If statements, for loops, while loops!
Compiler vs. Interpreter

- Compiler translates whole program to object code
  - produces the most highly optimized code
- Interpreter translates one line of code at a time
  - can quickly make changes and try things out
- C – compiled
- Java – compiled to byte code, then interpreted
- Shell – interpreted
Software Tools

- Tools save you time and make you a better programmer:
  - editor, language choice, debugger, build system, version control system, issue tracking, profiling and monitoring are all examples of software tools

- High-level scripting languages make it possible to glue tools together to do all kinds of time-saving tasks
Part II

Programming in C
Programming in C

- Memory model
  - pointers are addresses with a type
- Remember that variables are not automatically initialized (except for static ones – file or function scope)

- Arrays
  - contiguous region of memory with fixed size
  - provide random access
  - array name is a pointer to the first element

- Pointers
  - stores an address
  - dereference with *
  - get the address of a variable with &
Strings

- Arrays of characters
- Remember the null termination character ("\0")
- Most string functions depend on it
- Whenever possible use the string functions rather than re-implementing them
- E.g., use `strncpy` rather than copying each character
- Be careful to ensure that you don't walk off the end of a character array
Dynamic memory allocation

- `malloc`, `calloc`, `realloc`
- Memory allocated using `malloc` should be freed when it is no longer needed
- Keep a pointer to the beginning of the region so that it is possible to free
- **Memory leak** occurs when you no longer have a pointer to a region of dynamically allocated memory
When to use malloc?

- When passing a pointer to a new region of memory back from a function
- When you don't know until runtime how much space you need
Header files contain function prototypes and type definitions

Header files are useful when your program is divided into multiple files

Use Makefile to compile programs. Saves typing and takes advantage of separate compilation
Bit sets

- Binary representation
- Logical operations – Not, And, Or, Xor, Shift operations (!, &, |, ^, >>, << ..)
  - Practice!
- Bit masks are present everywhere in a system
  - Some file modes (O_RDONLY, O_CREAT, etc.)
  - Signal masks (sigprocmask)
  - File descriptor sets (select)
  - Etc.
Part III

UNIX System Programming
System Calls

- Perform subroutine calls into the Unix kernel
  - Interface to the kernel

- main categories
  - file management
  - process management
  - error handling
  - communication

- Error handling
  - system calls usually return -1  *(Always check!)*
  - errno
Processes

- **process state:** *running, ready, blocked*
- **fork()** – creates a duplicate process
- **exec()** – replaces the program being run by a different one
- **file descriptors maintained across fork and exec**
- **process ids** – getpid(), getppid()
Process Termination

- **Orphan process:**
  - a process whose parent is the init process because its original parent died

- **Zombie process:**
  - a process that has completed execution but still has an entry in the process table
  - it is “waiting” for its parent to accept its termination status

- Use macros to check the status:
  - WIFEXITED, WIFSIGNALED, WEXITSTATUS
Signals

- Signals are software **interrupts**, a way to handle asynchronous event (i.e., unexpected)
- Examples: control-C, termination of child, floating point error, broken pipe
- Normal processes can send signals
  - `kill(pid, SIG)` — sent SIG to pid
    - Recall how you’d do this at the commandline?
      - `kill -INT 1047`
  - `sigaction()` — install a new signal handler for a signal
  - `sigprocmask()` — block signals
Inter-process Communication (IPC)

- Data exchange between process:
  - message passing: files, pipes, sockets

- Limitations of **files** for IPC data exchange
  - slow
  - possibly altered by other processes

- Limitations of **pipes**:
  - two processes must be running on the same machine
  - two processes must be related

- **Sockets** overcome these limitations
Streams? File Descriptors?

- Unix has two main mechanisms for managing file access
  - **streams**: high-level, more abstract (and portable)
    - you deal with a pointer to a FILE structure, which keeps track of info you don’t need to know
    - `fopen()`, `fprintf()`, `fread()`, `fgets()`
  - **file descriptors**: each file identified by a small integer (on Unix), low-level, used for files, sockets and pipes
    - `open()`, `close()`, `read()`, `write()`, etc.
Sockets

- Sockets allow communication between machines
- TCP/IP protocol – internet address, ports
- Protocol families: PF_INET, PF_LOCAL
- Server side initialization takes 4 steps
  - `socket()` – initialize protocol
  - `bind()` – initialize addresses
  - `listen()` – initialize kernel structures for pending connections
  - `accept()` – block until a connection is received.
Sockets

- Client initializes socket using `socket()`, and then calls `connect()`
- Need to be wary of host byte orders
- Communication is done by reading and writing on file descriptors (sockets)
- **Ports** are divided into three categories: well-known, registered, and dynamic (or private)
- **Socket types:**
  - `SOCK_STREAM` = TCP
  - `SOCK_DGRAM` = UDP
Multiplexing I/O

- `select()` allows a process to block on a set of file descriptors until one or more of them are ready.
- Read calls on a “ready” file descriptor will only block while the data is transferred from kernel to user space.
- Makes it easier for one process to handle multiple sources of input.
- `select()` takes “file descriptor sets” as arguments.
- The macros FD_SET, FD_ISSET etc. are used to manipulate the bit set data structure.


Threads

- Processes have two limitations:
  - it is expensive to create a new one and switch between processes
  - processes cannot share memory (easily)
- Threads allow multiple instruction streams (threads of execution) in a single address space and solve both these problems
- Thread libraries also contain higher-level synchronization mechanisms (mutex-es) and conditional variables
Concurrent Programming

- Race condition: final outcome depends on the order in which things run
- Must protect shared resources
- Use mutexes/locks to protect critical sections
- Producer/Consumer Problem:
  - consumer should block when buffer is empty
  - producer should block when buffer is full
  - only one should be updating the buffer at a time
- A pipe is an example of producer/consumer
Final Exam

- Course Material & How to study
  1. Lecture slides – plus any extras posted!
  2. Assignments – make sure you understand concepts and code!
  3. Lab material – review the lab exercises, try things, PRACTICE!
  4. Textbooks (ref: King, Kerrisk, Haviland)

- Covers everything in the course, except concurrency!

- Closed book exam – No Aids Allowed
  - The exam will contain an aid sheet with prototypes and shell info (as needed)
Exam info and tips

- 45% of the grade, MUST get >40% on the final

- Combination of multiple choice, coding questions, conceptual questions, one-liner solutions, etc.

- Use time wisely! Don’t linger on a question too long! Get easy questions done fast, and then revisit harder problems.

- Exam API sheet will be provided (you may detach during exam)
Remainder

- Pre-exam Office Hours:
  - Thursday April 9 @ 12:30-3pm, in BA 5205

- Final Exam
  - WHEN: Monday April 13, @19:00-22:00
  - WHERE: SF3202/EX300/310/320 (check schedule for your assigned room)

- Course Evaluations