# CSC209 Review

# CSC209: Software tools …

- Unix
  - files and directories
  - permissions
  - utilities/commands
- Shell
  - programming
  - quoting
  - wild cards
  - files

# ... and C programming ...

- C
  - basic syntax
  - functions
  - arrays
  - structs
  - strings
  - pointers (!!!)
  - function pointers
  - header files

# … and systems programming

- System calls
- Files
- Processes (fork, exec)
- Inter-process Communication
  - signals
  - pipes
  - sockets
  - select
- Concurrency and Threads

# Shell Concepts

- stdin, stdout, stderr
- I/O redirection
  - csh – `prog >& outfile` – stdout and stderr to outfile
  - sh – `prog > outfile 2>&1` – same
- Job control
- Pipes

# 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
- 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

# 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, regression testing, issue tracking, profiling and monitoring.
- High-level scripting languages make it possible to glue programs together to do all kinds of time-saving tasks.

# Programs as Data

- Executables are just files that can be copied, moved, searched and even edited
- Compilers are just programs that operate on source code and produce executables
- Programming tools treat program source code as data
- High-level programming languages give us easier ways to operate on programs:
  - automated testing, build systems, version control

# Programming in C

- Memory model
  - pointers are addresses with a type
- Remember that local variables are not automatically initialized.
- Arrays
  - contiguous region of memory with fixed size
- Pointers
  - dereference with *
  - get the address of a variable with &

# Strings

- 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 of the end of a character array.

# Dynamic memory allocation

- memory allocated using `malloc` should be `free`d when it is no longer needed (unless you are about to exit)
- 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.
- This is a poor use of malloc:

```
main() {
    char *str1 = malloc(MAXLEN);
    ...
    free(str1)
    return 0;
}
```

# Header files

- Header files contain function prototypes and type definitions.
- Never `#include` a file containing functions and variable declarations file. You will run into trouble.
- Header files are useful when your program is divided into multiple files.
- Use Makefiles to compile programs. Saves typing and takes advantage of separate compilation.

# System Calls

- Perform a subroutine call 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 is “waiting” for its parent to accept its termination status.

```
wait(int *status);
r = waitpid(pid_t pid, int *status, int options);
```

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

# 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's) and conditional variables.

# Concurrency

- Race condition: final outcome depends on the order in which things run.
- 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

# 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.
  - Binary versus text I/O

# Signals

- Signals are software interrupts, a way to handle asynchronous event.
- Examples: control-C, termination of child, floating point error, broken pipe.
- Normal processes can send signals.
- `kill(pid, SIG)` – sent SIG to pid
- `sigaction()` – install a new signal handler for a signal
- `sigprocmask()` – block signals

# 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.
- 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.

# 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`

# Final Exam

- How to study
  - Look at previous exams for structure.
  - Play with example code provided.
- Closed book exam except…
  - Bring one hand-written 8.5”x11” sheet of paper
    - double-sided (no magnifying glasses allowed)
  - The exam also contains an aid sheet with prototypes and shell info.
    - published on the course web site (don’t bring it to exam!)

# Remainder

- Check web page for office hours
- Review session
  - When?
    - Thursday December 15 or Monday December 19?
- Please submit any remark requests promptly.
- All remark requests must be submitted before the exam.

- *Please verify that posted marks are correct before the exam!*

# 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.

*Good luck on the final,*

*and have a good holiday!*