

CS110 Course Syllabus

Overview of Linux Filesystems

- Linux and C libraries for file manipulation: **stat**, **struct stat**, **open**, **close**, **read**, **write**, **readdir**, **struct dirent**, file descriptors, regular files, directories, soft and hard links, programmatic manipulation of them, implementation of **ls**, **cp**, **cat**, etc.
- naming, abstraction and layering concepts in systems as a means for managing complexity, blocks, **inodes**, **inode** pointer structure, **inode** as abstraction over blocks, direct blocks, indirect blocks, doubly indirect blocks, design and implementation of a file system.
- additional systems examples that rely on naming, abstraction, modularity, and layering, including DNS, TCP/IP, network packets, databases, HTTP, REST, descriptors and **pids**.
- building modular systems with simultaneous goals of simplicity of implementation, fault tolerance, and flexibility of interactions.

Multiprocessing and Exceptional Control Flow

- introduction to multiprocessing, **fork**, **waitpid**, **execvp**, process ids, inter-process communication, context switches, user versus supervisor mode.
- protected address spaces, virtual memory, main memory as cache, virtual to physical address mapping.
- concurrency versus parallelism, multiple cores versus multiple processors, concurrency issues with multiprocessing.
- interrupts, faults, systems calls, signals, design and implementation of a simple shell.
- virtualization as a general systems principle, with a discussion of processes, RAID, load balancers, AFS servers and clients.

Threading and Concurrency

- sequential programming, VLIW concept, desire to emulate the real world with parallel threads, free-of-charge exploitation of multiple cores (two per **myth** machine, eight per **rice** machine, 24 per **barley** machine), pros and cons of **threading** versus **forking**.
- C++ **threads**, **thread** construction using function pointers, blocks, functors, **join**, **detach**, race conditions, **mutex**, IA32 implementation of **lock** and **unlock**, spin-lock, busy waiting, preemptive versus cooperative multithreading, **yield**, **sleep_for**.
- condition variables, rendezvous and thread communication, **unique_lock**, **wait**, **notify_one**, **notify_all**, deadlock.
- semaphore concept and **semaphore** implementation, generalized counter, pros and cons of **semaphore** versus exposed condition variables, thread pools, cost of threads versus processes.

- active threads, blocked threads, ready thread queue, high-level implementation details of the thread manager, **mutex**, and **condition_variable_any**.
- pure C alternatives via **pthread**s, pros of **pthread**s over C++ thread package.

Networking and Distributed Computing

- client-server model, peer to peer model, protocols, request and response as a way to organize modules and their interactions to support a clear set of responsibilities.
- stateless versus keep-alive connections, latency and throughput issues, **gethostbyname**, **gethostbyaddr**, IPv4 versus IPv6, **struct sockaddr** hierarchy of **structs**, network-byte order.
- ports, socket file descriptors, **socket**, **connect**, **bind**, **accept**, **read**, **write**, simple echo server, time server, concurrency issues, spawning threads to isolate and manage single conversations.
- C++ layer over raw I/O file descriptors, introduction to **sockbuf** and **sockstream** C++ classes.
- HTTP 1.0 and 1.1, header fields, **GET**, **HEAD**, **POST**, complete versus chunked payloads, response codes, web caching and consistency protocols.
- IMAP, custom protocols, Dropbox and iCloud reliance on variations of HTTP.
- MapReduce programming model, implementation strategies using multiple threads and/or processes, comparison to previous systems that do the same thing, but not as well.
- non-blocking I/O, where normally slow system calls like **open**, **accept**, **read**, and **write** return immediately instead of blocking, **select**, **epoll_*** set of functions, **libev** and **libuv** open source libraries.

Phil Levis contributed to this handout.