Csci 4061 - Last Meeting

- *Administrative*

  * Any questions on practice final

  * Final Exam. - stuents w/ conflict
  - Pl. oyour namesde info. on the signup!

  * Recitation Schedule:
  - HW 5 grades, grade record verification
  - Another Practice Exam.

- *Discussion:*

  * Final Exam. details

  * Summarize the course

Final Exam.

- *Basic Information*
  - Place: Classroom
  - Time: 1830-2030, Mon 12/20/99 (eve. Sec.)
  - Time: 1030-1230, Tues. 12/21/99 (day sec.)
- *Nature*
  - Open book, man-pages, classnotes
  - Closed neighbors, computers etc.
  - Syllabus: Chapters 1, 2, 3, 5, 9, 10, 12

Final Exam.

- *Nature*
  - * Problem solving - calculate file sizes, etc.
  - * Analysis - output for a given program
  - * True/False
  - * Match items from two tables
  - * Few definitions, comparisons, discussions
- *Practice Exam. this Wednesday in recitation*
- *A Note on True/False Questions*
  - * Justifications are more important !
  - * Absolute truth is not being looked for.
  - * Ex. UICI uses private channels ...
  - - Each connection request results in a new channel
  - - But fork() by client after u_connect ...

Final Exam. Details

- *Details - Important System Calls*

  * Ch. 1.: perror, strerror

  * Ch. 2.: getpid, getppid, getenv, setsid
  - fork, exit, wait, waitpid, execl

  * Ch. 3.: getcwd, chdir, opendir, readdir, closedir
  - status, open, read, write, close, dup2, pipe

  * Ch. 5.: kill, raise, alarm,
  - sigprocmask, sigaction, pause, sigsuspend
  - (sigempty, sigfillset, sigaddset, sigdelset, sigismember)

Final Exam. Details

- *Details* - *calls from ANSI C Standard Libraries*
  - C Memory Management: malloc, free
- *Details* - *Important Shell symbols and Commands*
  - Ch. 1.: man, make, cc,
  - Ch. 2.: ps, env, &, bg, fg,
  - Ch. 3.: cd, ls, find, ln, |, <, >, >>,
  - Ch. 5.: kill, intr (ˆC)

Final Exam. Details

- *Details* - *Important System Calls*

  * Ch. 9.: pthread_create/exit/kill/join

  * Ch. 10.: pthread_mutex_init/destroy/lock/unlock/trylock
  - sem_init/destroy/wait/post/trywait (Ch. 8.3)
  - pthread_kill, pthread_sigmask, sigwait

  * Ch. 12.: u_open/close/listen/connect/read/write
  - socket, bind, listen, accept, connect, read, write

Final Exam. Details

- *Details - Important Concepts*
- *Chapter 9*
  - * Client request processing architecture for Servers
  - - serial vs. multi-threaded vs. multi-processes
  - * thread properties
  - * thread implementations: user-level, kernel-level
- *Big picture issues*
  - * threads vs. procedures
  - * threads vs. processes

Final Exam. Details

- *Details - Important Concepts*
- *Chapter 10*

  * Race conditions in MT programs

  * Synchronization methods
  - locks, semaphores, condition variables

  * Threads and signals
  - synchronous, asynchronous, directed signals
  - signal handling in Multi-Threaded programs

- *Big picture issues*

  * pthread_join vs. other synchronization methods

  * disjoint address space as a synchronization method

  * thread-safety vs. signal-safety vs. reentrant

Final Exam. Details

- *Details - Important Concepts*
- *Chapter 12*
  - * Client, server, naming, communication
  - * Naming: host, port
  - * Communication: connection-less vs. connection based
  - * UICI vs. Sockets
  - * Communication architectures for Servers
- *Big picture issues*
  - * Formats: Little-Endian vs. Big-Endian
  - * Naming: port vs. process-id/thread-id
  - * UICI channels vs. pipes (chapter 3)

Course Summary

- *Goals: Understand concurrency*
  - * Why concurrency?
  - * Sources of Concurrency
  - - I/O, signals, processes, threads, client-server
  - * Effects of concurrency
  - - race conditions
- *Focus*
  - * Server - software concurrently shared by many
  - * User level - commands, shell
  - * Power Users - system calls, C programs
- *Out of Scope*
  - * Operating System Theory - e.g. CPU scheduling
  - * Vendor specific features - e.g. Win32

Unix Standards

- *Why Standards?*

  * Multiple flavours of Unix: HPUX, Solaris, Linux, ...

  - Two distinct lineage - BSD and System V

  * Non-Unix OS: NT, Windows 3.1/95/98/..., MacOS, ...

  * System calls are often OS specific!

  * Overhead of porting across OS.

- *Which Standards?*

  * ANSI C

  * POSIX - IEEE Portable Operatig System Interface

  - Table 1.3 provide POSIX standards

  * if not covered by POSIX

  - Spec 1170

  - System V Release 4

- *Q? Did we study any non-POSIX systems calls/concepts?*

What is Concurrency?

- *Concurrency:*
  - * Sharing of resource in the same time-frame
  - * Ex. two program executing concurrently
  - * Q? Which resources are they sharing?
- *What is hard about Concurrency?*
  - * Race conditions
  - * Non-deterministic behaviour
  - * Bugs do not show up on a regular basis
- *Trends leading to Concurrency*
  - * Servers - Web, DBMS, Mail, ...
  - * Graphical User Interfaces
  - - Animation of multiple objects
  - * Multiprocessors
  - * Distributed Systems, e.g. internet

What is Hard about Concurrency?

- *Shared functions/libraries*
  - * should be safe for reentry
- *Non-Reentrant functions*
  - * Self modifying code
  - * functions using static/global variables
  - * Problems with multiple simultaneous invocations
- *Reentrant functions*
  - * Allow multiple simultaneous invocations
  - * Needed for signal handler, server with many clients, ...
  - * Two aspects -

  - Thread safe: can be called concurrently by 2 threads

  - Async. Signal safe: can be called inside a signal handler

  - without restriction
- *Q? Compare signal-safe (SS) and thread-safe (TS).*
  - * Provide a function which TS but not SS.

Units of Concurrency

- *Process: (Ch. 2)*
  - * instance of a program in execution
  - * multiple processes on one machine
- *Procedures (Ch. 3, 5)*
  - * system call, e.g. asynchronous I/O vs. computing
  - * signal handlers
- *Threads within a process: (Ch. 9, 10)*
  - * finer granualarity
  - * Share code, heap, globals
- *Communication (Ch. 12)*
  - * processes across network (Client-server)

Ch. 2: Processes

- *Motivation*

  * Structure real-time program with multiple tasks

- *Process: a program in execution*

  * Attributes: pid, ppid,

  * Operations: fork, exit, join, wait, ...

- *Implementation Details*

  * States: new, running, blocked, ready, done

  * Layout: Code, global data, heap, stack, env.

- *Cooperating Processes*

  * Parent - child relationship

  * exit() - wait() coordination

- *Background Processes, Daemon processes*

Ch. 3. Input and Output

- *Motivation*
  - * Coordinate resources with varying speed
  - * Why should an application developer learn this?
  - * You may develop performance critical applications
  - - Ex. real-time - Pacemaker
  - - Ex. Web servers, transaction processors - ebay, amazon, ...
- *Ex. asynchronous I/O*
  - * A process itseld can do other things
  - * while waiting for an I/O, i.e. synchronous read()
  - * instead of getting swapped out by OS
- *Ex. monitoring multiple input source on network*
  - * Standard blocking I/O is not suitable!
- *Concurrency*
  - * Subprogram handling file/network I/O
  - * Subprograms computing during wait for I/O

Ch. 5. Signals

- *Motivation*
  - * Q? How do you stop a program in an infinite loop?
  - * Other usage: timers, job control, aynch. I/O, ...
- *Signal = software notification of an event*
  - * Ex. hardware events, e.g. ctrl-c, I/O complete
  - * Q? Provide examples of synchronous signals.
- *Life cycle of a Signal*
  - * Event of interest occurs
  - * Signal is generated
  - * OS sets a flag for the relevant process
  - * Signal is caught by the process
  - * Process invokes a handler subroutine
  - * Analogy - "You have mail" flag
- *Concurrency: main program, signal handler subroutine*
  - * Implication: restriction on signal handler
  - * Sharing a global variable => special protection

Ch. 9. Threads and Resource Sharing

- *Motivation* - *What is the unit of concurrency?*
  - * Traditional unit = process
  - * Emerging finer unit = thread
- *Processes* - *Generated via fork() call*
  - * Coordinate termination via wait()
  - * Communicate via pipes (common ancestors),
  - - or signals, messages, shared memory, etc.
  - * Pros: stronger security boundaries
  - * Cons: high overhead
- *Threads* - *provide concurrency within a process*
  - * threads of execution = program counter value streams
  - * Finer level of concurrency
  - * Low overhead in creating and context switching
  - * standards are emerging now!
- *Concurrency*
  - * Multiple processes or Multiple threads within a process

Ch. 12. Network as the Computer

- *Motivation - internet!, intranet, networks, ...*
  - Multiple services: ftp, email, ...
  - Million of clients accessing Web services
- *Client-Server = A model of distributed computing*
  - Client = caller of a service
  - Server = provider of a service
  - Analogy with procedure call, caller, callee
- *Details*
  - Clients and Servers may be on different machines
  - Communication via messages or remote procedure calls
  - Signals, Pipes, shared memory are not common
- *Concurrency*
  - Server and client are concurrent
  - Multiple Servers and multiple clients