CSE-381 Exam #2 Topics Systems 2

**Topics for Exam #2** CSE-381: Systems 2

As per the syllabus, Exam #2 for CSE-381: Systems 2 is scheduled

**Thu, Nov 14 (Time: 7 PM to 9 PM) In Hug 141 (Hughes hall)**

The topics enumerated below are a succinct list of major concepts that you are expected to know for Exam 2. It may not include all details covered in lectures and labs. Consequently, this is not an exhaustive list and you must use in-class exercises, lab exercises, and homework assignments in addition to these topics to fully prepare for the exam. Particularly this is the time to review the notes you have been diligently recording during lectures.

The exam will be closed notes and closed books. A one-sheet handout on common methods and Linux commands will be supplied along with the exam -- see CommonMethodsAndCommands.pdf. No other reference materials or discussions will be permitted. Use of electronic equipment (other than those required for life support and have suitable doctor certification along with them) other than calculators (calculator must not have a qwerty keyboard or a graphical stylus input pad) is strictly prohibited.

***Type of questions***: Same general format as most exams: Multiple-choice, fill in the blanks, short answer, “what does the following C++/assembly code do”, and “write the C++/assembly code to do the following" types of questions. Specifically, you will be expected to read, comprehend, analyze, troubleshoot, and develop programs involving concepts covered in the course. Programs will involve suitable object oriented concepts and standard library concepts covered in the course.

**Text coverage:**

E-book titled “Operating System Concepts" -- Link in Syllabus page on Canvas (all students have free access to the electronic book):

▪ Chapter 4: Threads

▪ Chapter 6: Synchronization

▪ Chapter 10: File system interface

▪ Chapter 11: File system implementation

Page 1 of 6

CSE-381 Exam #2 Topics Systems 2

• **Basics of Managing threads in Linux** i. Basics of process & thread management and process hierarchies. ii. Viewing threads in Linux using ps -fealL a. Interpreting output from ps -fealL iii. Starting and killing threads via Linux terminal. iv. Setting thread priorities and nice values.

v. Process IDs, Parent process IDs, and Monitoring processes using ps command.

• **Basics of Linux operations** i. Creating and navigating directories via Linux terminal. ii. Creating, copying, and deleting files using commands in a Linux terminal. iii. Basics of compiling and running C/C++ program in Linux in a Linux terminal. iv. Foreground vs. background processes in Linux.

• **Threads and multithreading (Chapter 4)** i. Concurrency and multi-tasking. ii. Concept of multithreading. iii. Concept of a thread. Single vs. multi-threaded processes. iv. Resources shared between threads versus resources available only to a thread.

v. Processes vs. Threads --advantages vs. disadvantages. vi. Thread lifecycle (same as process life cycle):

New→Ready→Running→Waiting→Terminated vii. Creating threads in C++ viii. Foreground vs. background (detached) threads

ix. Developing synchronization-free multithreaded programs in C++

a. One thread per task (using std::async) b. Multithreaded web-server c. Multithreaded client d. Data parallel program -- several items per thread.

• **Synchronization (Chapter 6)** i. Race conditions -- identifying and demonstrating race conditions ii. Identification of code snippets with race conditions / incorrect multi-threading. iii. Symptoms of race conditions. iv. Need for synchronization.

v. Concept of a critical section. **vi. 4 Rules to create critical sections.** vii. Concept of a Semaphore and mutex viii. Using std::mutex to create a critical section

ix. Need and use of a std::lock\_guard with std::mutex

x. Identification of critical sections in a code fragment. xi. Concept of deadlocks

a. Locking multiple mutexes using std::lock to avoid deadlocks xii. Priority inversion xiii. Producer-consumer multithreading model with fixed size shared queue.

Page 2 of 6

CSE-381 Exam #2 Topics Systems 2

a. Busy-wait/spin-lock approach -- advantages vs. disadvantages xiv. Using Monitors (or condition variables) to avoid busy waiting

a. Using std::condition\_variable b. Understanding wait-notify c. Advantages vs. disadvantages over busy-wait xv. Using std::async for multithreading xvi. Using std::atomic for MT-Safe operations on primitive data types

• **Networking -- Concepts related to WWW and HTTP** i. Terminology and acronyms ii. Concept of a protocol iii. Basics of HTTP protocol and line endings "\r\n" iv. Basic structure of GET requests

a. URL encoding & decoding v. HTTP headers

a. HTTP response headers b. Basic content types in HTTP response c. Concept of MIME type in HTTP response vi. Basics of system integration via fork & exec vii. Identifying parameters from an HTML form

• **File systems (Chapter 10, 11)** i. Need for a file system ii. Relative and absolute paths iii. Motivation and use of key data structures in a file system

a. Root directory and use of inodes b. Inspecting inode information and interpreting output of fstat

command c. Tracing File chains -- example: File Allocation Table (FAT). See lab

exercise for example of tracing file blocks iv. Relative and absolute path

• Conversion between relative ⇔ absolute path v. Links a. Links (or hard links) -- see ln command

b. Symbolic or soft links -- see ln -s command c. Influence on file sizes and use of disk storage vi. Security and Privacy:

a. File permissions -- using chmod to assign users, group members, and

other users different read, write, and execute permissions to manage privacy.

**C/C++ programming concepts:**

• **Basic program constructs**

i. Stages in compiling a C++ program. ii. Variables & expressions

Page 3 of 6

CSE-381 Exam #2 Topics Systems 2

iii. Constant variables vs. literal constants iv. Signed vs. unsigned data types

v. if and if-else statements vi. switch statement vii. Looping constructs (for, while, do-while, range-for) viii. Basic mathematical problem solving concepts

▪ Deciding number is even/odd, positive/negative, factor/divisor/dividend/quotient

▪ Using division and modulo operations for basic number manipulation, e.g.: reverse a number with loops & math (without using string)

▪ Detecting if a number is prime.

▪ Identifying largest/smallest number in a set of inputs

▪ Finding average (i.e., mean) of a given set of numbers ix. Functions/methods

• Pass by value versus pass by reference

a. Preferred approach for primitive data types vs. objects

• Memory/copy impact of pass-by-value

• Using const keyword for parameters. x. Default values for parameters

• **Basics of objects**

i. Differences between primitive and object data types in C++ ii. Calling methods on objects (e.g.: string::length) iii. Using std::string

• Constructors for string.

• String comparisons

• Methods for operating and accessing strings

• Conversion to-and-from numeric data types to std::string.

• Formatting strings into HTML, given HTML tags to use (you don't need to know HTML)

• **Arrays**

i. Basics of old-style arrays. ii. 1-D arrays iii. 2-D arrays iv. Command-line arguments

• Designing programs that use command-line arguments

• Figuring out what and how many command-line arguments a program ought to take.

• When to prefer command-line arguments instead of reading data from files/console.

• **Basics of Pointers**

Page 4 of 6

CSE-381 Exam #2 Topics Systems 2

i. Concept of memory and address ii. Basics of pointers to hold addresses iii. Basic pointer operators

• Address of operator (&)

• Indirections/dereferencing a pointer (\*)

• Using object dereference operator (->) iv. Pointer arithmetic

v. Pointers ↔ array operation similarities and code conversion vi. Understanding command-line arguments

• Arrays of pointers vii. Using shared\_ptr in lieu of pointers

• **Vectors**

i. Use of vectors instead of arrays for processing data ii. Differences between vectors and arrays iii. Defining and using vectors of different data types iv. Using vectors in method definitions and method calls

v. Create type aliases via the using clause in C++

• Creating aliases given English description

• Tracing aliases back to their original types. vi. Operations on a vector: adding elements, accessing elements, removing

elements, etc. vii. Reading/printing/writing vectors to I/O streams viii. Vectors of user-defined classes

• **Hash maps (unordered\_map)**

i. Concept of unordered\_map ii. Using unordered\_map as associative arrays iii. Defining and using unordered\_maps of different data types iv. Looking-up values in unordered\_maps

v. Iterating over all the entries in a map and processing them vi. Reading/printing/writing vectors to I/O streams vii. Maps of user-defined classes

• **Basic text file I/O operations**

i. Reading and writing data to console using std::cin and std::cout. ii. Using stream-insertion (<<) and stream-extraction (>>) operators to read

and write data.

• Understanding these operators and how they handle whitespaces. iii. Using std::getline method to read a full line of text iv. Using std::ifstream and std::ofstream to read/write text files.

v. Using std::istringstream and std::ostringstream to perform

I/O with strings.

Page 5 of 6

CSE-381 Exam #2 Topics Systems 2

• **Other exercises**

i. Converting English statements to corresponding C++ statements ii. Describing C++ statements in English iii. Code walkthroughs to determine operation and output from a C++ program iv. Developing a C++ program given a functional description

v. Identifying performance or memory issues in C++ programs vi. Rewriting C++ program to address memory or performance issue

• **Linux commands and shell**

i. Basic operations at the shell prompt

• Navigating directory structures

• Listing files

• Copying files -- **including using scp**

• Troubleshooting common problems given error message(s) ii. Compiling and running programs iii. Using pipes to create ad hoc software pipelines

• Redirection to create (>) or append (>>) to existing files

• Redirection to supply input from a file (<)

• Using pipe (|) to create software pipelines

iv. Using /usr/bin/time to measure runtime characteristics of programs

• **Elapsed time, %CPU** v. Interpreting output of /usr/bin/time for single vs. multithreaded

programs

**Preparation Suggestions:**

1. As a general note you should expect to repeat questions from lab exercises and

homework. 2. You should know all the material in lecture slides. 3. Do read the E-book materials used in homework while paying attention to

implementation/application details. 4. Redo lab exercises. Develop short programs to test/verify your understanding of

concepts. Review developing classes and overloading operators. Review how to call overloaded operators. Review vectors, how to use vectors. Review unordered\_map and how to use it. 5. Review homework solutions on Canvas. 6. Review the functionality of pertinent methods and commands in the supplied

method/command sheet. 7. Review the handouts material and videos on Canvas.

Page 6 of 6