Topics for Final Exam

CSE-381: Systems 2

|  |  |  |
| --- | --- | --- |
| ☞ | Location: | EGB 270 (usual lecture room) |
| Section C & D: | Mon Dec 9 from 12:15 pm – 2:15 pm |

|  |
| --- |
| The topics enumerated below are a succinct list of major concepts that you are expected to know for the final examination. It may not include several details covered in classes, laboratory experiments, and homework. Consequently, it 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.  The exam will be closed notes and closed books. A two-sheet handout with common C++ methods and Linux commands will be supplied along with the exam (see common methods sheet posted off the course website). No other reference materials or discussions will be permitted. Use of electronic equipment is strictly prohibited.  ***Sample Exams:***  Some of the earlier exams in the operating systems course (that used a different syllabus, textbook, C/Assembly, and sequence of topics) are supplied merely to some example questions for this exam. Some of the questions in the sample exam do not pertain to the content in the exam. Refer to the topics listed in this document and to appropriately review questions in the sample exam.  ***Type of questions***:  Same general format as previous exams: Multiple-choice, fill in the blanks, short answer, “what does the following code do”, and “write the C++ code to do the following" types of questions. Specifically, you will be expected to read, comprehend, analyze, troubleshoot, and develop C++ programs involving concepts covered in the course. Program will involve object oriented concepts and C++ 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 1: Introduction
  + Chapter 2: Operating System Structures
  + Chapter 3: Processes
  + Chapter 4: Threads
  + Chapter 6: Synchronization
  + Chapter 10: File system interface
  + Chapter 11: File system implementation (fi
  + Chapter 15: Security
  1. **Concepts pertaining to systems**
     1. The 3 key aspects that define a system (you must memorize the definition)
     2. Open vs. closed systems
     3. Common examples of systems
     4. Concept of systems-of-systems
     5. Key components of a computing system (aka computer)
     6. Terminology & expansion for acronyms: System, sub-system, CPU, ALU, Register, RAM, ROM, DBMS, File System, NIC, TCP, IP
  2. **Functions of an operating system with examples**

1. Types of OS: single user vs. multi-user, single tasking vs. multi-tasking
2. Batch processing vs. multiprocessing.
   1. Context switching -- preemptive vs. non-preemptive
3. Memory management, basics of virtual memory, pages and paging, mapping virtual memory to real memory.
4. Storage management. Basics of creating directories, files, and navigating directories using Linux shell commands.
5. Device management and concept of interrupt driven I/O.
6. Security and Privacy:
   1. Security rings
   2. User-id, effective-user-id, group-id
   3. File permissions -- using chmod to assign users, group members, and other users different read, write, and execute permissions to manage privacy.
   4. **System calls**
7. Basics of system calls
8. Three common approaches for implementing system calls
9. Tracing system calls using strace.
   1. First (exec) and last (exit\_group) system calls observed via strace.
   2. **Design of operating systems**
10. Factors influencing design of OS
11. Design strategies for OS
    1. Monolithic,
    2. Microkernels,
    3. Layered,
    4. Hypervisors, Type-1 and Type-2 hypervisors
    5. Client-Server
12. Advantages & disadvantages of various OS design approaches
13. Identification of ideal OS design approach from scenario descriptions.
    1. **Booting & Boot loaders**
14. Concept of booting
15. Steps in booting
16. Boot loader concepts and design strategies (single vs. 2-stage booting)
    1. **Basics of processes**
17. Concept of processes. Process vs. program
18. Basic memory layout of processes (Text, Data, Heap, & Stack)
19. Observing processes using ps and top
20. Processes hierarchies and tracing hierarchies in Linux
21. Terminating processes using kill
22. Process lifecycle: New→Ready→Running→Waiting→Terminating
    1. Scenarios when different transitions occur
    2. **Process creation (expect 1-or-2 programming questions)**
       1. Creating processes using fork system call
          1. Creating different hierarchies of processes
          2. Understanding cloning of I/O streams
       2. Replacing existing program with execvp system call
          1. Passing command-line arguments
          2. Basic idea of PATH to locate programs
       3. **Using waitpid to wait for process to complete**
          1. **Using exit code (return value for main of child process)**
    3. **Inter-Process Communication (IPC) (no programming questions but Linux commands)**
       1. Motivation for IPC
       2. Shared memory vs. message passing
       3. Using pipes for IPC
       4. Named pipes/fifo
          1. Creating FIFO (mkfifo) & setting permissions
          2. Redirecting input/output at the shell
       5. Using anonymous pipes at the shell
          1. Creating ad hoc software pipelines via bash
       6. Creating and using anonymous pipes from a program
          1. Concept for file descriptor (fd)
          2. Using pipe and dup2 system calls
          3. Simple pipes for IPC between parent & child
          4. Using anonymous pipes with exec
    4. **Basics of Managing threads in Linux**
23. Basics of process & thread management and process hierarchies.
24. Viewing threads in Linux using ps -fealL
25. Starting and killing threads via Linux terminal.
26. Setting thread priorities and nice values.
27. Process IDs, Parent process IDs, and Monitoring processes using ps command.
    1. **Basics of Linux operations**
28. Creating and navigating directories via Linux terminal.
29. Creating, copying, and deleting files using commands in a Linux terminal.
30. Basics of compiling and running C/C++ program in Linux in a Linux terminal.
31. Foreground vs. background processes in Linux.
    1. **Threads and multithreading (topic of emphasis with most programming questions)**
32. Concurrency and multi-tasking.
33. Concept of multithreading.
34. Concept of a thread. Single vs. multi-threaded processes.
35. Resources shared between threads versus resources available only to a thread.
36. Processes vs. Threads --advantages vs. disadvantages.
37. Thread lifecycle (same as process life cycle):

New→Ready→Running→Waiting→Terminated

1. Creating threads in C++
   1. 2-loops patterns: Programming pattern of using 2-separate loops, 1 loop to start threads and another separate loop to join threads.
2. Foreground vs. background (detached) threads
3. Developing synchronization-free multithreaded programs in C++
   1. One thread per task (using std::async) and 2-loops pattern
   2. Multithreaded web-server
   3. Multithreaded client
   4. Data parallel program -- several items per thread.
   5. **Synchronization (topic of emphasis with most programming questions)**
4. Race conditions -- identifying and demonstrating race conditions
5. Identification of code snippets with race conditions / incorrect multi-threading.
6. Symptoms of race conditions.
7. Need for synchronization.
8. Concept of a critical section.
9. **4 Rules to create critical sections.**
10. Concept of a Semaphore and mutex
11. Using std::mutex to create a critical section
12. Need and use of a std::lock\_guard with std::mutex
13. Identification of critical sections in a code fragment.
14. Dining philosophers conceptual model for multiple, shared resources
15. Concept of deadlocks
    1. Locking multiple mutexes using std::lock to avoid deadlocks
16. Priority inversion
17. Producer-consumer multithreading model with fixed size shared queue.
    1. Busy-wait/spin-lock approach -- advantages vs. disadvantages
18. Using Monitors (or condition variables) to avoid busy waiting
    1. Using std::condition\_variable
    2. Understanding wait-notify
    3. Advantages vs. disadvantages over busy-wait
19. Using std::atomic for MT-Safe operations on primitive data types
    1. **File system concepts**
20. Need for a file system
21. Motivation and use of key data structures in a file system
    1. Root directory (inodes)
    2. File chains -- example: File Allocation Table (FAT)
22. Relative and absolute path
23. Links
    1. Links (or hard links) -- see ln command
    2. Symbolic or soft links -- see ln -s command
    3. **Virtualization**
24. Virtual Machines and 3 use cases
    1. Server virtualization

Many physical servers are under utilized in data centers, we can run many VMs on 1 server to reduce hardware costs, increase energy efficiency, reduce maintenance, and improve fault tolerance

* 1. Desktop virtualization

Users may need to use different OSes, and we might need to run our software on different platforms or legacy systems

* 1. Security

For the purposes of testing out unknown software that may have malware or viruses, or to reverse engineer malicious software, we can use an isolated VM as our sandbox with roll-back features and encrypted storage and advanced logging.

1. Hypervisors
   1. Type 1 Hypervisor

No native OS, can be viewed as the OS. Runs on bare metal.

* 1. Type 2 Hypervisor
     1. What is qemu?

Qemu Is a type-ii hypervisor that emulates different kinds of CPU and can run different operating systems simultaneously

1. Concept of cloud computing
   1. Infrastructure as a Service (IaaS)

Service is just the VM, possibly without an OS, the user can install any OS and software as needed

* 1. Platform as a Service (PaaS)

Service provider installs and runs specific OS and infrastructure, user can install whatever software they want on the OS and run it

* 1. Software as a Service (SaaS)

Service provider installs and runs the software(DOCS, Microsoft 360)

* 1. **Basics of Security**
* Concepts & Terminology – Virus, Worms

Viruses are simply code that automatically copies itself and attaches to other programs or files. In order to spread, it requires user intervention. Worms do the exact same thing over a network.

* Common software vulnerabilities
  + Buffer overflow

If programs exceed the bounds of arrays or buffers, we can flag things as true, or insert data places we shouldn’t be able to do that.

* + Denial of Service attacks

Occurs when we generate an abundance of connections performing an action/hanging, and no threads remain to take new requests, preventing other folks from.

* + Distributed Denial of Service attacks

Same as above, except with multiple users involved, attacking a single endpoint.

* Phishing.

Convincing someone to give away information utilizing convincing emails/text messages.

* 1. **Networking -- Concepts related to WWW and HTTP**

1. Terminology and acronyms
2. Concept of a protocol
3. Basics of HTTP protocol and line endings "\r\n"
4. Basic structure of GET requests
   1. URL encoding & decoding
5. HTTP headers
   1. HTTP response headers
   2. Basic content types in HTTP response
   3. Concept of MIME type in HTTP response
6. Basics of system integration via fork & exec
7. Identifying parameters from an HTML form

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

* 1. **Basic program constructs**

1. Stages in compiling a C++ program.
2. Variables & expressions
3. Constant variables vs. literal constants
4. Signed vs. unsigned data types
5. if and if-else statements
6. switch statement
7. Looping constructs (for, while, do-while, range-for)
8. 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

1. Functions/methods
   * + 1. Pass by value versus pass by reference
          1. Preferred approach for primitive data types vs. objects
       2. Memory/copy impact of pass-by-value
       3. Using const keyword for parameters.
2. Default values for parameters
   1. **Basics of objects**
3. Differences between primitive and object data types in C++
4. Calling methods on objects (e.g.: string::length)
5. 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)

* 1. **Arrays**

1. Basics of old-style arrays.
2. 1-D arrays
3. 2-D arrays
4. 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.
   1. **Basics of Pointers**
5. Concept of memory and address
6. Basics of pointers to hold addresses
7. Basic pointer operators
   * + - Address of operator (&)
       - Indirections/dereferencing a pointer (\*)
       - Using object dereference operator (->)
8. Pointer arithmetic
9. Pointers ↔ array operation similarities and code conversion
10. Understanding command-line arguments

* Arrays of pointers

1. Using shared\_ptr in lieu of pointers
   1. **Vectors**
2. Use of vectors instead of arrays for processing data
3. Differences between vectors and arrays
4. Defining and using vectors of different data types
5. Using vectors in method definitions and method calls
6. Create type aliases via the using clause in C++
   * + 1. Creating aliases given English description
       2. Tracing aliases back to their original types.
7. Operations on a vector: adding elements, accessing elements, removing elements, etc.
8. Reading/printing/writing vectors to I/O streams
9. Vectors of user-defined classes
   1. **Hash maps (unordered\_map)**
10. Concept of unordered\_map
11. Using unordered\_map as associative arrays
12. Defining and using unordered\_maps of different data types
13. Looking-up values in unordered\_maps
14. Iterating over all the entries in a map and processing them
15. Reading/printing/writing vectors to I/O streams
16. Maps of user-defined classes
    1. **Basic text file I/O operations**
17. Reading and writing data to console using std::cin and std::cout.
18. Using stream-insertion (<<) and stream-extraction (>>) operators to read and write data.
    * + 1. Understanding these operators and how they handle whitespaces.
19. Using std::getline method to read a full line of text
20. Using std::ifstream and std::ofstream to read/write text files.
21. Using std::istringstream and std::ostringstream to perform I/O with strings.
    1. **Other exercises**
22. Converting English statements to corresponding C++ statements
23. Describing C++ statements in English
24. Code walkthroughs to determine operation and output from a C++ program
25. Developing a C++ program given a functional description
26. Identifying performance or memory issues in C++ programs
27. Rewriting C++ program to address memory or performance issue
    1. **Linux commands and shell**
28. Basic operations at the shell prompt
    * Navigating directory structures
    * Listing files
    * Copying files -- **including using scp**
    * Troubleshooting common problems given error message(s)
29. Compiling and running programs
30. 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
31. Using /usr/bin/time to measure runtime characteristics of programs
    * Elapsed time, %CPU
32. 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.