**CS 241 Final Exam Study Guide (Updated 11:23 am May 10, 2014)**The final examination will be held in from 8:00am to 11:00am on Friday, May 16. Based on your last name, the room your will take the final exam in is:
 **A - Le**: 1404 SC
 **Li - Z**: 151 Loomis

You may not consult any materials during the exam: no textbooks, no crib sheets, no calculator, etc.

The exam will contain a set of multiple choice questions and long answer questions. About half the total points on the exam will be for each type of problem. Each long answer may consist of multiple parts and will carry approximately equal weight overall, but may break down unevenly among the parts. These questions may ask you to reason about or write some code or to determine the behavior of an algorithm. On the long answer questions on the exam, you must show all work and reasoning, writing both work and solution legibly, and should box all answers if the page also contains intermediate work. If the course staff cannot read a solution, no credit will be given.

The final exam is cumulative. Very roughly, 33% of the exam will be drawn from material also covered on the midterm exam.

1. **C Programming**
	1. What is POSIX?
	2. What is a library function? What is a system call? What is the difference? Give an example of a pure library function and a pure system call.
	3. How does pointer arithmetic work?
	4. What is the \* operator? What does it do? What is the & operator? What does it do?
	5. What is a function pointer? How do you define a function pointer? What functions have you learned about in CS 241 that take a function pointer as a parameter?
	6. What is a “C string”? How is a “C string” represented in memory?
	7. What is NULL?
	8. What is the difference between strlen() and sizeof()?
	9. What’s the difference between a stack and a heap variable? What about global and static variables?
	10. How do malloc() and free() work?
	11. What is the difference between a string and a string literal?
	12. How do strcpy(), strcat(), strncpy(), and strncat() work?
	13. How do printf() and scanf() work? What are the common formatting arguments?
	14. How do you read a series of lines from a file or stdin using getline()?
2. **Memory**
3. What is the difference between physical and virtual memory?
4. What are common memory allocation algorithms and what are the advantages of each?
5. How are virtual addresses translated to physical addresses in multi-level page tables?
6. How do page size and the number of levels of page tables affect the number of entries in a page table?
7. What is the difference between internal and external fragmentation?
8. Describe how the buddy system works and the run time for its operations.
9. What is thrashing? When does it occur?
10. What causes a SEGFAULT and what happens when one occurs?
11. When is a process swapped out to disk?
12. Name three benefits of virtual memory (as opposed to allowing programs to directly access physical memory).
13. Name one advantage of segmentation over paging, and one advantage of paging over segmentation.
14. How is a page table similar to an inode? What is the difference between these structures?
15. Assuming a 32-bit address space and 4 KB pages, what is the virtual page number and offset for virtual address 0xd34f6a5?
16. Give an example of a page fault that is an error, and an example of a page fault that is not an error.
17. How many page faults occur in the above example?
18. Suppose we have a 64-bit address space and 16 KB pages. How big is the page table of a single process, if the system uses single-level page tables? What is the problem here? How would multi-level page tables help solve this problem?
19. How does the virtual memory subsystem know the exact location where a particular page is stored on disk, if it is swapped out of memory?
20. Compare and contrast (give one benefit and one disadvantage) for: implicit, explicit, segregated, and buddy free lists.
21. **Processes and Threads**
	1. What resources are shared between threads of the same process? What resources are not shared?
	2. Invent some code using pthread\_create() statements. What could be its output?
	3. Invent some code where two threads use a shared variable X. What are the possible values for X after both threads complete execution? (X is a global integer variable and initially X = 0.)
	4. What happens when a thread calls exit()?
	5. What happens to a process’s resources when it terminates normally?
	6. Describe what happens when a process calls fork(). Be able to trace through the code.
	7. Under what conditions would a process exit normally?
	8. Explain the actions needed to perform a process context switch.
	9. Explain the actions needed to perform a thread process switch.
	10. What are the advantages and disadvantages of kernel-level threads over user-level threads?
	11. Compare the use of fork() to the use of pthread\_create().
	12. In a multiprocessor system, what system characteristics will cause other threads of the same process to block?
	13. How can a process become orphaned and what does the OS do with it? What's a zombie?
	14. Write a piece of code using fork() to create a process tree of depth n, where each process (a node in the tree) except for the "leaf" processes has exactly m child processes.
	15. Describe how to use the POSIX call wait().
	16. Explain what happens when a process calls exec().
	17. Explain how thread-safe functions are used in C.
	18. If a process spawns a number of threads, in what order will these threads run?
	19. Explain how to use pthread\_join().
	20. Explain how a shell process can execute a different program without using system().
	21. Explain how one process can wait on the return value of another process.
	22. Describe the transitions between running, ready and blocked in the 5 state model.
	23. Understand how pthread\_exit() differs from exit().
22. **Scheduling**
	1. Why do processes and threads need to be scheduled?
	2. Describe the following scheduling algorithms and their advantages and disadvantages:
		1. Round-Robin (RR)
		2. First Come First Serve (FCFS)
		3. Shortest Job First (SJF)
		4. Preemptive Priority
	3. What is starvation? Which scheduling policies have the possibility of resulting in starvation?
	4. Define waiting time and response time in the context of scheduling algorithms.
	5. Which scheduling algorithm results the smallest average wait time?
	6. What scheduling algorithm has the longest average response time?
	7. Why might Shortest Job First be hard to implement in real systems?
	8. What does it mean to preempt a process?
	9. In the context of scheduling, what is a time quantum?
	10. How does the length of the time quantum affect Round-Robin scheduling?
	11. A process was switched from running to ready state. Describe the characteristics of the scheduling algorithm being used.
	12. Which properties of scheduling algorithms affect the performance of interactive systems?
23. **Synchronization**
	1. What is the readers-writers problem?
	2. What is the producers-consumers problem?
	3. What is the dining philosopher problem?
	4. Recognize a correct solution to the readers-writers problem, the producers-consumers problem, the sleeping barber problem, and the dining philosopher. Be able to identify and explain an error in a specific implementation of any of the classic synchronization problems.
	5. What happens when readers are prioritized over writers in the classic readers-writers problem? How about if writers are prioritized over readers?
	6. What is required so that deadlock and starvation do not occur in the dining philosopher’s problem? Give examples of solutions.
	7. What is the difference between starvation, deadlock, race conditions and critical sections? Describe each.
	8. What would happen if a system’s hardware synchronization primitive were replaced with a software function?
	9. Which type of variables must be protected against concurrent readers and writers in any combination?
	10. What is progress?
	11. What is mutual exclusion?
	12. Given two threads running example code that contains a critical section, be able to identify if progress and mutual exclusion are ensured.
24. **Synchronization primitives**
	1. Understand the common semaphore and mutex functions (sem\_wait(), sem\_post(), etc).
	2. What are proper and improper code replacements for a test\_and\_set() operation?
	3. How does the internal counter of a POSIX semaphore work? What does it mean if the value of the semaphore is 1?
	4. What happens to a thread when it waits on a semaphore whose current value is 1?
	5. How can the reader-writer problem be solved using only POSIX mutexes?
	6. Using only one mutex, is it possible to create a semaphore? If so, how? If not, why?
	7. Understand how to solve the producer-consumer problem using mutexes and semaphores.
	8. What is a buffer overflow? What is a buffer underflow? Understand how failures in synchronization could cause buffer over and underflows.
	9. What are condition variables? Understand how they can be used in code.
	10. Understand how to fix deadlocks and starvation in code involving mutexes, semaphores, and conditional variables.
25. **Deadlock**
26. Define deadlock.
27. Define circular wait, mutual exclusion, hold and wait, and no preemption. How are these related to deadlock?
28. How would the implementation of a web server using threads differ from one using processes?
29. What can happen if synchronization in a multiple-threaded program is not programmed carefully?
30. Why might an operating system use a resource allocation graph?
31. What are the components of a resource allocation graph?
32. What techniques can be used to prevent deadlock?
33. What does **waitpid()** do?
34. What is the difference between Deadlock Prevention, Deadlock Detection, and Deadlock Avoidance? What deadlock handling mechanism would you use?
35. What problem does the Banker’s Algorithm solve? Given a set of processes how would you use the Banker’s Algorithm?
36. What is a safe state and how can you determine if a system is in a safe state?
37. **IPC**
38. What is the difference between a FIFO and a pipe?
39. How would you redirect standard out to a file?
40. How do you create and use a pipe?
41. What do **dup()** or **dup2()** do, and why would you use them?
42. What is the difference between a pipe, a FIFO, and an ordinary file on disk?
43. What happens when two processes read and write to a memory mapped file?
44. Explain how two processes can share memory using **mmap**.
45. Describe the purpose of a POSIX signal.
46. Explain how a process can set custom signal handlers.
47. What might happen to a signal if a process hasn't set a custom handler?
48. Explain how a process can set which signals are caught or ignored using a signal set.
49. How can one process send a signal to another?
50. Some signals cannot be caught or ignored. Which signals are they and why shouldn’t they be allowed to be caught?
51. What does “**kill -<parameter> pid**” do?
52. How does the function **alarm()** work?
53. **Networking and I/O multiplexing**
54. Explain the difference between a pipe and a socket.
55. When do you use the **close()** or **shutdown()** system call with sockets?
56. Discuss how a multithreaded web server running on a single processor system could be optimized using the process scheduling methods discussed in class. Which do you recommend?
57. How does **epoll()** work? What problem does it solve?
58. Describe the POSIX **socket()**, **bind()**, **listen()**, and **accept()** functions.
59. How does HTTP work?
60. What is the difference between persistent and non-persistent HTTP?
61. Describe the services provided by TCP.
62. How does TCP connection establishment work?
63. Describe the differences between TCP and UDP. Which is better? Which would you use?
64. How do sockets support the client-server model?
65. How does the Domain Name System (DNS) work?
66. How does DNS use caching?
67. How is DNS related to IP?
68. What is NAT? How does NAT work?
69. Understand how to read an HTTP header using C code.
70. What is MapReduce? How does it work?
71. **File systems and I/O**
72. What information is stored in an i-node? What information isn’t?
73. What data structure best describes an i-node?
74. If an i-node based file system has a certain number of direct and single-indirect, and double-indirect blocks, what is the maximum size of a file in the filesystem?
75. How many i-node operations are required to fetch a file at /path/to/file?
76. Given the description of an i-node file system, how many i-node accesses are required to read the entire contents of a file of a given size? How many blocks does this file consume on disk?
77. When the size of a block changes in an i-node based file system, how does this change the maximum size of a file?
78. How does the page-out process (controlled by the kernel) relate to file systems?
79. Where does **fstat()** look to find the information that it returns?
80. **Security**
	1. What is a hash function?
	2. What is a salt?
	3. Why is a salt necessary?
	4. When would a hash function be used?
	5. Give an example of a situation where a hash function could not be used.
	6. What is a denial of service attack?
	7. What is a DNS reflection and amplification attack? Why does it use DNS?