Overview

(1) Solve the following:
   A. How much is $2^{13}$ in decimals?
   B. How much is roughly 1 billion in power of 2?
   C. How much is $2^{64}/2^{21}$ in power of 2?
   D. How much is $128$MB/4KB?
   E. How much is log2(8192)?

(2) When measuring I/O throughput, what is the difference between the units
   (1) MBps and Mbps
   (2) KBps and Kbps?

(3) How much are these units in decimals?
   (1) Pico
   (2) Nano
   (3) Micro
   (4) Milli
   (5) Kilo
   (6) Mega
   (7) Giga
   (8) Tera
   (9) Peta
   Hint:
   For metric system:
   See http://www.chemteam.info/Metric/Metric-Prefixes.html
   For size of information in computers see:
   https://web.stanford.edu/class/cs101/bits-gigabytes.html

(4) Replace “?” below with the correct answer
   A. 1 Nanosecond = ? seconds
   B. 500 Milliseconds = ? seconds
   C. 4 KB = ? bytes
   D. 4 Kilometers = ? meters
   E. 4Kbps = ? bits per second

(5) What is an Operating System? List its primary responsibilities.
(6) What are the three (or four) different ways in which OS code can be invoked? Explain.
(7) What is an Instruction Set Architecture (ISA)? What is the difference between User ISA and System ISA?
(8) What is a system call? How is a system call different from ordinary function calls?

(9) Explain the following interfaces in a computer system
(a) Instruction Set Architecture (ISA)
(b) User Instruction Set Architecture (User ISA),
(c) System ISA,
(d) Application Binary Interface (ABI).
(e) Application Programmers’ Interface (API)

(10) Why doesn’t a program (executable binary) that is compiled on the Linux machine execute on a Windows machine, even if the underlying CPU hardware is the same (say x86)?

(11) Let’s say that you are asked to modify the Linux OS so that programs and libraries compiled on Windows OS could run natively on Linux, meaning they should be executed as normal programs (without using any emulator or virtual machine). What would be your high-level approach?

(12) What hardware mechanism does x86 ISA provide to ensure that Operating System’s code and data are protected from user-level processes?

(13) What is the role of privilege levels (defined by the ISA) in a computer system? How many privilege levels are defined in the x86 ISA? In which privilege level does the OS execute?

(14) What is the difference between a hardware interrupt, a software interrupt (trap), and an exception? Give examples of each.
Processes

(1) (a) What is a process? (b) How is a process different from a program?

(2) In the memory layout of a typical process, why do stack and heap grow towards each other (as opposed to growing in the same direction)?

(3) In terms of call-return behavior, how are the fork() and exec() system calls different from other system calls?

(4) (a) Describe the process lifecycle illustrating the states and transitions. (b) Which transitions occur when a process (i) is pre-empted? (ii) voluntarily yields the CPU?

(5) What is a zombie process? Why does the Operating System maintain the state of zombie processes? List two ways in which a parent process can prevent a child process from becoming a zombie.

(6) Why are frequent context switches expensive in terms of system performance?

(7) What is cold-start penalty? What are some ways to reduce it?
# Threads

1. What are threads? How do they differ from processes? How are they similar?
2. What state do threads share? What state is different?
3. Why does context-switching between threads incur less overhead than between processes?
4. Briefly explain
   (a) User-level threads
   (b) Kernel-level threads
   (c) Local thread scheduling
   (d) Global thread scheduling
5. What are the benefits and disadvantages of using user-level and kernel-level threads?
6. What combinations or user/kernel threads and global/local scheduling are feasible and why?
7. What kind of applications benefit the most from kernel-level threads support? What kind of applications benefit most from user-level threads support? Explain why with examples?
8. Explain how a web server could use threads to improve concurrency when serving client requests.
9. What happens if a thread in a multi-threaded process crashes? How can you improve the robustness (fault-tolerance) of a multi-threaded application?
10. When would you prefer (a) event-based programming (b) threads-based programming? Why?

11. Event-driven programming
    (a) What is the “event-driven” programming model?
    (b) What does the structure of a typical event-driven program look like?
    (c) When would you prefer an event-driven programming model over a thread-based programming model?

12. What is the problem with long-running event handlers? How do threads solve this problem?
IPC
1. List any five inter-process communication mechanisms, with a one line description for each?

2. When using a pipe for inter-process communication, why should a process promptly close any unused write descriptors to the pipe? Also give an example of what happens if it doesn’t.

3. Let's say a chain of filters refers to a series of commands whose standard inputs and standard outputs are linked by pipes. For example,
   
   “ps -elf | grep bash | more”
   
   is a chain with three commands.

   In the general case,
   
   “command1 | command2 | command 3 | ... | command K”
   
   is a chain of filters with K commands.

   Suppose you were implementing a shell (e.g. csh, bash, tcsh, ksh, etc.), how would you go about supporting a chain of filters with arbitrary number of commands? Explain. Don’t write actual code.

4. What’s the difference between byte-stream vs. message oriented communication?
System Calls

1. What is a system call? How do system calls differ from ordinary function calls?

2. What steps take place when a system call is invoked by a process?

3. What is a system call table? Why is it needed? OR What role does it play in OS security?

4. Explain the CPU-privilege transitions during a system call.

5. (a) Why do some operating systems, such as Linux, map themselves (i.e. the kernel code and data) into the address space of each process? (b) What is the alternative?
Concurrency

1. Define Concurrency. How does it differ from parallelism?
2. Explain the differences between apparent concurrency and true concurrency.

3. Briefly explain with examples
   A. Critical Section
   B. Race condition
   C. Deadlock

4. What’s wrong with associating locks with code rather than shared resources?
5. Describe the behavior of (a) UP and DOWN operations on a semaphore, (b) WAIT and SIGNAL operations on a condition variable.

6. Under what situation would you use (a) Blocking locks, (b) Non-blocking locks, and (c) Spin locks. Which of these locks can be used in interrupt handlers and how?

7. When should you NOT use (a) blocking locks, (b) non-blocking locks, and (c) spin-locks?
8. What is the main difference between a binary semaphore and a counting semaphore?
9. What is priority inversion? How can prevent it?
10. Explain how a deadlock can occur in the operating system between code executing in the user-context and code executing in interrupt handlers. Also explain how you would prevent such a deadlock.
11. Multiple processes are concurrently acquiring and releasing a subset of locks from a set of N locks L1, L2, L3, …., LN. A process may try to acquire any subset of the N locks. What is the convention that all processes must follow in order to guarantee that there would be no deadlocks? Explain with an example where two processes need to acquire different but intersecting subsets of the N locks above.

12. How does the Test-and-Set Lock (TSL) instruction work?

13. Explain how you can implement the UP and DOWN operations on a mutex (binary semaphore) using the TSL instruction.

14. How does the compare-and-set instruction work? (b) How can you implement a DOWN operation on a mutex (binary semaphore) using a compare-and-set instruction (such as CMPXCHG in x86)?

15. Consider the classical producer-consumer problem. Producers produce items and insert them in a common buffer. Consumers remove items from the common buffer and consume them. In the following skeleton of pseudo-code, demonstrate the use of SEMAPHORES and MUTEXES to complete the pseudo-code for producer and consumer functions. Your code should have no race conditions and no busy loops.
16. Consider the classical producer-consumer problem. Producers produce items and insert them in a common buffer. Consumers remove items from the common buffer and consume them. Complete the following skeleton pseudo-code to explain how you can solve the producer-consumer problem using a monitor and condition variables.

```pseudo
# define N 100 /* Number of slots in the buffer */
typedef int semaphore; /* semaphores are a special kind of counter */
semaphore mutex = (initialize this); /* figure out the role of mutex */
semaphore empty = (initialize this); /* figure out the role of empty sem */
semaphore full = (initialize this); /* figure out the role of full sem */

void producer(void)
{
   /* complete this function */
}

void consumer(void)
{
   /* complete this function too */
}
```

---

You can assume that the following functions are available to you. You shouldn’t need anything more than these functions in your pseudo-code.

- `produce_item()` produces and returns an item
- `insert_item(item)` inserts the item in the common buffer
- `remove_item()` removes and returns an item at the head of the buffer
- `consume_item(item)` consumes the item supplied

`up(&semaphore)` and `down(&semaphore)` have their usual meanings

---

You can assume that the following functions are available to you. You shouldn’t need anything more than these functions in your pseudo-code.
procedure insert(item)
begin
    /* complete this procedure */
end

procedure item *remove()
begin
    /* complete this procedure */
end
end

monitor

17. Consider the “events vs threads” argument in the context of monolithic operating system kernels (like Linux or Windows). (a) Which model do these operating systems primarily use -- events or threads? Why? (b) Let’s say you that have to design an operating system that uses the opposite model to what you just answered in (a). What would be the major design changes you would make to the kernel in terms of CPU scheduling, memory management, and I/O processing subsystems?

18. What are the tradeoffs in using semaphores versus monitors with condition variables?

19. You are given a function f() in the Linux kernel that constitutes a critical section, i.e. no two parts of the kernel should execute f() concurrently. Assume that when the function f() is invoked anywhere in kernel, you call it using the following convention.

    Do some form of locking;
    Invoke function f()
    Do some form of unlocking.

Explain what type of locking/unlocking mechanism would you choose under each of the following situations and justify your answer:

a. Function f() executes for a very short time. It can be called concurrently from two or more threads within the kernel (meaning either processes or conventional threads currently in the kernel context, such as within a system call), but NEVER from the within an interrupt context. (Interrupt context refers to the code that is executed immediately as a result of a hardware interrupt to the kernel, i.e. interrupt service routine, and also to the code that executes immediately following an ISR, but just before resuming the interrupted thread.)

b. Function f() can execute for a very long time. Otherwise, just as in the previous case, it can be called concurrently from two or more threads within kernel, but never from the within an interrupt context.

c. Function f() executes for a very short time. It can be called concurrently from two or
more threads within kernel, and **also** from the within an interrupt context.

Justify your answers, keeping in mind that the system can have either just a single-processor or multiple processors. Try to give the best possible locking mechanism, not just something that works. If possible, you can support your answer with real examples from within Linux source code where each of the above types of locking/unlocking approaches are used.
**File Systems**

1. What is a File system

2. In a file-system, (a) What is meta-data? (b) Where is meta-data stored? (c) Why is it important for a file system to maintain the meta-data information? (d) List some of the typical information that is part of the meta-data.

3. (a) Suppose you collect a trace of I/O operations above the file system layer (in applications or in system calls). Do you expect to see more write I/O operations or read I/O operations? (b) Now suppose you collect a similar trace of I/O operations below the block device layer (in the disk or device driver). Do you expect to see more write I/O operations or read I/O operations? Explain why?

4. If you increase or decrease the disk block size in a file system, how (and why) will it affect (a) the size of the inode, and (b) the maximum size of a file accessible only through direct block addresses?

5. How does the inode structure in UNIX-based file-systems (such as Unix V7) support fast access to small files and at the same time support large file sizes.

6. What does the file system cache do and how does it work? Explain with focus on the data structures used by the file system cache.

7. Explain the role of file system cache during (a) read I/O operations and (b) write I/O operations.

8. Describe two different data structures using which file system can track free space on the storage device. Explain relative advantages/disadvantages of each.

9. How does a log-structured file system work? Why is its performance (typically) better than conventional file systems?

10. In a file-system, explain how two different directories can contain a common (shared) file. In other words, how do hard links work?

11. How does the inode structure in UNIX-based file-systems (such as Unix V7) support fast access to small files and at the same time support large file sizes.

12. Explain the structure of a UNIX i-node. Why is it better than having just a single array that maps logical block addresses in a file to physical block addresses on disk?

13. Explain the steps involved in converting a path-name /usr/bin/ls to its i-node number for the
14. What’s wrong with storing file metadata as content within each directory “file”? In other words, why do we need a separate i-node to store metadata for each file?

15. Assume that the
• Size of each disk block is B.
• Address of each disk block is A bytes long.
• The top level of a UNIX i-node contains D direct block addresses, one single-indirect block address, one double-indirect block address, and one triple-indirect block address.
   (a) What is the size of the largest “small” file that can be addressed through direct block addresses?
   (b) What is the size of the largest file that can be supported by a UNIX inode?
Explain your answers.

16. In a UNIX-like i-node, suppose you need to store a file of size 32 Terabytes (32 * 2^40 bytes). Approximately how large is the i-node (in bytes)? Assume 8096 bytes (8KB) block size, 8 bytes for each block pointer (entry in the inode), and that i-node can have more than three levels of indirection. For simplicity, you can ignore any space occupied by file attributes (owner, permissions etc) and also focus on the dominant contributors to the i-node size.

17. In a UNIX-based filesystems, approximately how big (in bytes) will be an inode for a 400 Terabyte (400 * 2^40 bytes) file? Assume 4096 bytes (4KB) block size and 8 bytes for each entry in the inode that references one data block. For simplicity, you can ignore intermediate levels of indirections in the inode data structure and any space occupied by other file attributes (owner, permissions etc).

18. Assume that the size of each disk block is 4KB. Address of each block is 4 bytes long. What is the size of the largest file that can be supported by a UNIX inode? What is the size of the largest “small” file that can be addressed through direct block addresses? Explain how you derived your answer.

19. Assume all disk blocks are of size 8KB. Top level of a UNIX inode is also stored in a disk block of size 8KB. All file attributes, except data block locations, take up 256 bytes of the top-level of inode. Each direct block address takes up 8 bytes of space and gives the address of a disk block of size 8KB. Last three entries of the first level of the inode point to single, double, and triple indirect blocks respectively. Calculate (a) the largest size of a file that can be accessed through the direct block entries of the inode. (b) The largest size of a file that can be accessed using the entire inode.

20. In the “UNIX/Ritchie” paper, consider three major system components: files, I/O devices, and memory. UNIX treats I/O devices as special files in its file system. What other mappings are possible among the above three components? (In other words, which component can be
treated as another component)? What would be the use for each possible new mapping?

21. Suppose your filesystem needs to store lots of uncompressed files that are very large (multiple terabytes) in size. (a) Describe any alternative design to the traditional UNIX inode structure to reduce the size of inodes wherever possible (NOT reduce the file content, but reduce inode size)? (Hint: maybe you can exploit the nature of data stored in the file, but there may be other ways too). (b) What could be the advantage of your approach compared to just compressing the contents of each file?

22. Why doesn’t the UNIX file-system allow hard links (a) to directories, and (b) across mounted file systems?

23. Why did the authors of the “UNIX” paper consider the UNIX file-system to be their most important innovation?