Metrics

(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 128MB/4KB ?
   E. How much is log2(8192)?
   F. 2^{30}bytes of storage equals?
   G. 1 Gbps network bandwidth equals how many bits per second (bps)?

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

(3) How is a Mebibyte different from Megabyte?

(4) 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

(5) 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

OS Overview, ISA

(1) What is an Operating System? List its primary responsibilities.

(2) What are the different ways in which OS code can be invoked? Explain.

(3) Explain the following interfaces in a computer system
   (f) Instruction Set Architecture (ISA)
   (g) User Instruction Set Architecture (User ISA),
   (h) System ISA,
   (i) Application Binary Interface (ABI).
   (j) Application Programmers' Interface (API)

(4) If the kernel has highest privileges, why would a kernel fault result in a crash?

(5) 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)?

(6) What is meant by virtualization? Give examples of many(virtual)-to-one(physical, one-to-many, and many-to-many resource virtualization.

(7) What was the first computer? First OS? First programmer? First language?

Concurrency

1. Define Concurrency. How does it differ from parallelism?

2. Explain the differences between apparent concurrency and true concurrency.

3. What is "atomicity" and why is it important? (in concurrency, not physics!)

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

5. What's wrong with associating locks with code rather than shared resources?

6. Describe the behavior of (a) UP and DOWN operations on a semaphore, (b) WAIT and SIGNAL operations on a condition variable. (c) (c) When would you use a semaphore? When would you use a condition variable?

7. 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?

8. When should you NOT use (a) blocking locks, (b) non-blocking locks, and (c) spin-locks?
9. What is the main difference between a binary semaphore and a counting semaphore?
10. What is priority inversion? How can prevent it?
11. 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.
12. 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 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.

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

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

15. 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)?

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. 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***.

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
```

==========Pseudo-code Skeleton ==========

```
#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 */
}
```

==========

17. 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.**

procedure Producer
begin
    /* complete this procedure */
end

procedure Consumer
begin
    /* complete this procedure */
end

```
monitor ProducerConsumer
        condition /* declare the condition variables you need */
        integer /* declare any other variables you need */

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

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

18. Consider the "events vs threads" argument in the context of monilithic 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?

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

20. 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 because of a hardware interrupt to the kernel, i.e. interrupt service routine, and 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.

21. Explain how you can implement the WAIT and SIGNAL operations on condition variable using the TSL instruction.

File Systems

1. What is a File system?

2. What's an i-node? Where is it stored?

3. What's the simplest data structure for an i-node? Then why is UNIX i-node so complicated?

4. 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.

5. If you collect a trace of I/O operations below the file system cache (at device driver or physical disk level), what type of I/O operations do you expect to see more of -- write I/O requests or read I/O requests? Explain why.

6. (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?

7. 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?

8. 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.

9. 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.

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

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

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

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

14. 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***.

15. 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?

16. Explain the steps involved in converting a path-name */usr/bin/ls* to its i-node number for the file *ls*.

17. 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?

18. 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.

19. In a UNIX-like i-node, suppose you need to store a file of size 32 Terabytes (32 * 240 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.

20. In a UNIX-based filesystems, approximately how big (in bytes) will be an inode for a 200 Terabyte (200 * 240 bytes) file? Assume 4096 bytes 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 (permissions etc).

21. 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).

22. 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.

23. 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.

24. 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?

25. 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?

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

27. Why did the authors of the "UNIX" paper consider the UNIX file-system to be their most important innovation?

28. 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.

How big (in bytes) will be **an inode** for a file that is **F bytes long**? Calculate your answer for each case when the file spans (a) direct, (b) single-, (b) double-, and (c) triple-indirect blocks.

I/O Models

1. In the I/O subsystem, explain the two stages by which data is delivered from an I/O device to a user-level process.
2. What are the five I/O models? How do they handle the two stages of data reception?
3. Why is process blocking generally not a concern for *write* I/O operation?
4. When can a process block on a write I/O operation?
5. Compare the blocking I/O model with the I/O Multiplexing model? How are they similar and how are they different?
6. Which I/O model is used for event-driven programming? Explain why.
7. Way is asynchronous I/O? In practice, why is it hard for any OS to support true asynchronous I/O?
8. In this course you learnt two I/O classifications. Under "I/O Devices" lecture, you learnt about programmed I/O, interrupt-driven I/O, and DMA-based I/O. In "I/O Models" lecture, you learnt about blocking, non-blocking, signal-driven, and asynchronous I/O models. Explain the analogies and differences between these two I/O classification systems.
9. Explain the two key steps in delivering data in packets from a network card to a user-level process reading from a network socket?
10. Explain the difference between (a) signal-driven I/O and asynchronous I/O, (b) blocking I/O and I/O multiplexing.

RAID

1. Distinction between logical and physical I/O address spaces.
2. What was the original & current motivation for RAID?
3. Why is a multiple-disk system less reliable than a single disk?
4. How does Mean Time to Failure (MTTF) change as number of components in a system increases?
5. What are the different levels of RAID and how do each of them work?
6. What are the relative benefits/drawbacks of each RAID level?
7. How is data distributed in each RAID level?
8. How is parity calculated and stored in each RAID level?
9. What is the extent of read and write parallelism in each level?
10. How is the parity calculation bottleneck in RAID 4 solved?
11. In RAID-5, explain how can you perform a single logical write operation in no more than one physical read and two physical writes?
12. Consider RAID levels 0, 1, 3, 4, and 5: Which RAID level provides the best (a) reliability (b) I/O Parallelism. Explain why.
13. In order to save power, disks are usually spun down (placed in sleep or low-power mode). This works well if there is only one disk in the system, if all data resides on the single disk, and if performance is not a major concern. Consider a RAID-5 system consisting of N+1 disks. Explain how you can redesign RAID-5 so that all the following requirements are satisfied: (1) fault-tolerance of original RAID-5 is maintained under all conditions, (2) energy consumption is minimized by spinning down one or more disks whenever possible, and (3) performance (read/write throughput) of the system is maximized to the extent possible. Again, while there is no single correct answer, you must explain all salient aspects of your design, justify any assumptions you make, and examine any design tradeoffs (e.g. energy savings to performance).

14. In RAID 5, describe how you can complete a write I/O operation using just 2 disk reads and 2 disk writes.
15. (a) Explain (with formula), how does parity computation differ between RAID 3 and RAID 4? (b) How does parity placement on the disk (not parity computation) differ between RAID 4 and RAID 5? Explain with example.

16. How should parity be computed in RAID 5 to increase parallelism of write operations? Explain with parity computation formula.

17. What is the write parallelism problem in RAID and how is it solved?

18. Describe the design of a parity-based RAID system that can survive two-disk failures (as

opposed to single-disk failure discussed in class). In your design, be sure to explain the following: (a) How your system would compute the parity required for recovery from a two-disk failure? (b) How your system would recover from two-disk and single-disk failures, (c) How much additional space would parity information occupy, compared to data, and (d) How many parallel read and write I/O operations can your system support?

19. For a RAID system with N data disks and 1 parity disk, compare the level of parallelism provided by RAID 3, RAID 4, and RAID 5 for multiple simultaneous (i) read I/O operations, (ii) write I/O operations, and (iii) combination of read and write I/O operations? Explain your answers.

20. Consider RAID levels 1, 3, 4, and 5 (forget about RAID 0 and 2). Which RAID level provides the best (a) reliability (b) I/O Parallelism. Explain why.

Memory management

(1) Show the typical memory hierarchy of a computer system. Explain the tradeoffs between latency (access times), capacity, and persistence, across different levels of memory hierarchy.

(2) What is a page table?

(3) How many page tables are maintained by the operating system?

(4) If there were no TLB, how would memory accesses be affected?

(5) What are the following? What do they do? Where are they located?

A. Memory Management Unit (MMU)
B. Translation Lookaside Buffer (TLB)
C. Page tables
D. Swap device

(6) Mark the statements that are true

A. A page table is an array of physical memory pages
B. Page table entries dictates whether a process can read, write, or execute the contents of a physical page.
C. A page table maps physical page numbers to virtual page numbers
D. Page table entries can be used to track which memory pages are infrequently accessed.

(7) How is a virtual address converted to a physical address in a virtual memory system? Explain the roles of MMU, TLB, and Page Tables.

(8) If you increase or decrease the page size in a system, how (and why) will it affect ***(a)*** the size of the page tables, and ***(b)*** the TLB miss ratio?

(9) What's TLB Coverage? Why is TLB coverage important?

(10) How can one increase the TLB coverage?

(11) In memory management, what is meant by relocation and protection?

(12) Consider a machine having a 64-bit virtual address and 16KB page size.

a) What is the size (in bytes) of the virtual address space of a process?
b) How many bits in the virtual address represent the byte offset into a page?
c) How many bits in the virtual address are needed to determine the page number?
d) How many page-table entries does a process' page-table contain?