1. (5 points) Which of the following creates a binary from multiple object files?
   - compiler
   - assembler
   - linker
   - interpreter

2. (5 points) Giving each process an illusion of (almost) infinite amount memory corresponds to which role of the operating systems?
   - Referee
   - Illusionist
   - Glue
   - Resource manager

3. (3 points) When a process divides 1 by 0, what will happen next?
   - CPU detects this exception
   - OS kernel detects this exception

4. (5 points) Choose a circumstance that will NOT trigger a user to kernel transition?
   - Exception
   - Upcall
   - Interrupt
   - System call

5. (5 points) Why DMA is more efficient than programmed I/O?
   - DMA reduces memory latencies.
   - DMA reduces device-to-CPU communication latency.
   - DMA does not require device to communicate with CPU through ports.

6. (5 points) One approach to provide virtual memory is to use base and bound: each process is only allowed to access a region of the physical memory, and the region is specified as a base and a bound. For example, if base = 5 and bound = 10, the process is only allow to access memory between 5 and 15. What’s NOT an disadvantage of this approach?
   - Expanding the heap is difficult
   - Expanding the stack is difficult
   - Memory sharing is difficult
   - Memory isolation is difficult

7. (3 points) Virtual memory requires hardware support.
   - True.
   - False.

8. (5 points) The screen is 80 × 25 in size. Each pixel requires 2 bytes. The first byte is the ASCII code for the character and the second byte tells what color to draw the character. You know that the video buffer starts at VIDEO_MEMORY (type is void*) and the memory is organized line-by-line. Which of the following is the correct implementation of putchar?
   - void putchar(char val, int row, int col) {
     char* output_pos = VIDEO_MEMORY + row * 80 + col;
     *output_pos = val;
     *(output_pos + 1) = 0x7;
   }
   - void putchar(char val, int row, int col) {
     char* output_pos = VIDEO_MEMORY + row * 80 + col * 25;
     *output_pos = val;
     *(output_pos + 1) = 0x7;
   }
9. (3 points) An open file descriptor in a Unix process is an integer.
   ○ True.  ○ False.

10. (5 points) How many times does the following program print “Hello!”?

```c
int a = 0;
int rc = fork();
a++;
if (rc == 0) {
    rc = fork();
a++;
} else {
a++;
}
printf("Hello!\n");
printf("a is %d\n", a);
```

   ○ 2  ○ 3  ○ 4  ○ 6  ○ None of the above

11. (5 points) Considering the same program, what will be the largest value of a, displayed by the program?
   ○ 2  ○ 3  ○ 5  ○ None of the above

12. (5 points) A buddy memory allocator is an approximation of
   ○ first fit  ○ last fit  ○ best fit  ○ worst fit

13. (3 points) The heap manager (the memory lab) is a
   ○ User-level library  ○ OS kernel service

14. (5 points) In the memory lab, you have implemented a heap manager. The idea is to divide the heap region
    into many aligned blocks. Each block is prepended with a header (metadata_t). Now the user wants to allocate
    numbytes of data, and your algorithm decides to split a new block from the block represented by metadata_t*
    header.

   The definition of metadata_t is given below.

```c
typedef struct metadata {
    size_t size;
    struct metadata* next;
    struct metadata* prev;
} metadata_t;
```

   Which of the following is the correct implementation to get the address of the new header.
15. (5 points) Threads of the same process do NOT share:
   - Code
   - Global variables
   - Heap
   - Stack

16. (5 points) Consider the following solution proposed by your classmate to the Too Much Milk Problem. You can assume there is no instruction reordering. `note` is a global variable shared between A and B, and it is initialized to 0. Safety means it should not be the case that both A and B purchase milk. Liveness means as long as one of threads exists, there is milk eventually. This solution is:

   ```c
   Thread A
   if (note == 0) {
     if (milk == 0) {
       buy_milk();
     }
     note = 1;
   }

   Thread B
   if (note == 1) {
     if (milk == 0) {
       buy_milk();
     }
     note = 0;
   }
   ```

   - correct
   - does not ensure safety
   - does not ensure liveness
   - does not ensure either safety or liveness

17. (5 points) Considering the same question but for three roommates. You can assume there is no instruction reordering. Initially, `noteA = noteB = noteC = 0`. This solution is:

   ```c
   Thread A
   noteA = 1
   while (noteB == 1 or noteC == 1) {
     if (milk == 0) {
       buy_milk();
     }
     noteA = 0
   }

   Thread B
   noteB = 1
   if (noteA == 0 and noteC == 0) {
     if (milk == 0) {
       buy_milk();
     }
     noteB = 0
   }

   Thread C
   noteC = 1
   if (noteA == 0 and noteB == 0) {
     if (milk == 0) {
       buy_milk();
     }
     noteC = 0
   }
   ```

   - correct
   - does not ensure safety
   - does not ensure either safety or liveness

18. (3 points) User-level applications need to turn off/on hardware interrupts to create critical sections.
   - True.
   - False.

19. (6 points) Using an R/W lock implementation (discussed in class), your classmate wrote the following code to protect the access to some shared data structure:

   ```c
   rwlock.acquire_read(); // A
   // read the shared data structure
   rwlock.release_read(); // B
   rwlock.acquire_write(); // C
   // modify the shared data structure
   rwlock.release_write(); // D
   ```
You classmate know that inside the implementation of rwlock, there is a mutex. If you forget what’s discussed in the class, just think of a simplest implementation of r/w lock using mutex and condition variable. How many times does each line of the above code acquire the mutex? (≥1 means at least once, but maybe more due to different thread schedules. 1 means exactly once in all thread schedules.)

- A=1, B=1, C=1, D=1
- A≥1, B=1, C≥1, D=1
- A=1, B=1, C≥1, D≥1
- A≥1, B≥1, C≥1, D≥1

20. (3 points) After return from a call to `cv.wait(cv, lock)` the calling thread can rely that the lock is locked.
- True
- False

21. (3 points) BONUS QUESTION: Consider the following two threads, to be run concurrently in a shared memory (all variables are shared between the two threads). No instruction reordering. Initially, x = 0.

```
Thread A
for (i=0;i<5;i++)
    x += 1;

Thread B
for (j=0;j<5;j++)
    x += 1;
```

Assuming a single-processor system, that load and store are atomic, that x is initialized to 0, and that x must be loaded into a register before being incremented (and stored back to memory afterwards), what is the MINIMUM POSSIBLE value for x after both threads have completed?

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10