1. **POSIX** is …
   a) An operating system that implements a standard set of system calls and system behaviors.
   b) A specification for how an operating system must implement a standard set of system calls and system behaviors.
   c) A specification for a programming language and library that provide a standard set of system calls and system behaviors.
   d) A specification for a standard set of system calls and system behaviors that can be implemented in different ways by operating systems.

2. Which of the following is a correct use of formatted I/O?
   a) `printf("%s\d\n", 'h', 5);`
   b) `printf("%lf\n", x); /* assume float x; */`
   c) `scanf("%d\n", m); /* assume int m; */`
   d) `scanf("%d\n", &m); /* assume int m; */`

3. A difference between system calls and function calls is that:
   a) Only function calls preserve the stack frame; system calls do not.
   b) Function calls can be called from a signal handler but system calls cannot.
   c) System calls require a change of privilege level but function calls do not.
   d) System calls can pass arguments on the stack; function calls cannot.

4. In POSIX-compatible systems, the file descriptor object is:
   a) Created and returned by the open system call.
   b) Created by the fopen C library call (without a system call) and managed by the program.
   c) Created by the open system call, stored within the OS, and a pointer to it is returned.
   d) Exists permanently in the operating system and a pointer to it is returned by the open system call.
   e) Exists permanently on disk and read in by the operating system when requested, and a pointer to it is returned by the open system call.

5. The fork system call:
   a) Runs a new program in a child process.
   b) Provides an exact copy of the current process’s address space to an existing child process.
   c) Creates a new child process and gives it an exact copy of the current process’s address space.
   d) Creates a new child thread and gives it an exact copy of the current process’s address space.

6. A process is said to **terminate abnormally** if it exits because:
   a) It calls exit with a non-zero argument.
   b) It encounters a fatal error, e.g., an illegal memory access exception (SEGV).
   c) It receives a SIGKILL signal.
   d) Any of the above happens.
   e) Either b) or c) happens, but not a).

7. For a process that **terminates normally**, which of the following is incorrect?
   a) Its address space is reclaimed by the operating system.
   b) All output file descriptor buffers are flushed and all file descriptors are closed.
   c) Output file descriptor buffers are not flushed but all file descriptors are closed.
   d) The parent process is notified via a signal.
8. Which of the following kinds of data are automatically visible (i.e., a pointer does not have to be passed explicitly) to all POSIX threads in a multithreaded program:
   a) Global and stack variables and heap objects
   b) Global variables and heap objects only
   c) Global and stack variables only
   d) Global variables only
   e) Heap objects only

9. Which of the following causes a POSIX process (not just the calling thread) to exit:
   a) A call to `pthread_exit` by a child thread but not the main thread.
   b) A call to `pthread_exit` by the main thread.
   c) A call to `pthread_join` by the main thread that returns after a child thread (not necessarily the last one) has exited.
   d) A call to `pthread_join` by the main thread that returns after the last child thread has exited.
   e) None of the above.

10. Which of the following X values is NOT possible after both threads complete execution? (X is a global variable and initially X = 0.)

<table>
<thead>
<tr>
<th>Thread 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 0; i &lt; 2; i++) {</td>
</tr>
<tr>
<td>int c = X;</td>
</tr>
<tr>
<td>c++;</td>
</tr>
<tr>
<td>X = c;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 0; i &lt; 2; i++) {</td>
</tr>
<tr>
<td>int c = X;</td>
</tr>
<tr>
<td>c++;</td>
</tr>
<tr>
<td>X = c;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

   a) 4
   b) 3
   c) 2
   d) 1

11. Which scheduling algorithm gives the smallest average wait time?
   a) First Come First Served.
   b) Round Robin.
   c) Shortest Job First.
   d) Priority

12. Consider the following program snippets:

   Snippet i:
   ```c
   char *foo(char *ptr) {
     return ptr;
   }
   ```

   Snippet ii:
   ```c
   char *foo(char *ptr) {
     char a[100];
     ptr = a;
     return ptr;
   }
   ```

   Snippet iii:
   ```c
   char *foo(char *ptr) {
     char a[100];
     strcpy(a, ptr);
     return a;
   }
   ```

   Snippet iv:
   ```c
   char *foo(char *ptr) {
     char a[100];
     strcpy(a, ptr, 10);
     a[10] = '0';
     return a;
   }
   ```

   Snippet v:
   ```c
   char *foo(char *ptr) {
     char *a = (char *)malloc(strlen(ptr) + 1);
     strcpy(a, ptr);
     return a;
   }
   ```
Which of the above snippets are correct (in terms of memory usage):

a) (i) only
b) (ii) only
c) (iii) only
d) (i) and (iv)
e) (i) and (v)

13. What is the output of the following code?
char str[] = "I start from NULL and end at 0, but do I stop at \0 or NULL?";
printf("%s", str);
a) I start from
b) I start from NULL and end at
c) I start from NULL and end at 0, but do I stop at
d) I start from NULL and end at 0, but do I stop at \0 or
e) I start from NULL and end at 0, but do I stop at \0 or NULL?

14. What would be the output of the following code?
void *print_thread(void *ptr) {
    int i;
    for(i = 0; i < 10000 ; i++) {
        printf("%d\n", i);
    }
}
int main() {
    pthread_t p;
    pthread_create(&p, NULL, print_thread, NULL);
    pthread_exit(NULL);
    return 0;
}

a) No output
b) It is not possible to write 'pthread_exit' within main()
c) 0 through 9999

d) Different output in different runs
e) None of the above

15. Which of the following is NOT true for a system call?
   a) The OS handles the system calls.
   b) System calls are more heavyweight than normal function calls.
   c) In a system call, caller and callee are at the same address space.
   d) The state of the hardware (CPU) has to be saved before a system call is handled.
   e) None of the above.

16. Threads belonging to the same process share
a) stack
b) data section
c) register set
d) thread ID

17. Consider the following scheduling policies: (i) First In, First Out (FIFO), (ii) Round Robin with a small quantum, (iii) Round Robin with a large quantum, (iv) Preemptive Shortest Job First (SJF), (v) Preemptive Priority Scheduling (PPRI). Which of these policies can lead to starvation of processes?
a) (i), (iii), and (iv) only
b) (iv) and (v) only
c) (i) and (iii) only
d) (iii), (iv), and (v) only
e) (i), (ii), (iii), (iv), and (v)
18. Which of the following systems may never exhibit a “hold-and-wait” situation (“hold and wait” is one of the preconditions of deadlock)? You may assume that the only blocking in these systems occurs on mutexes.
   a) Systems that ensure that all resources needed for an application are locked in a single atomic operation that either succeeds and locks all requested resources or fails and locks none.
   b) Systems with only one mutex.
   c) Systems where all mutexes are numbered. A user cannot lock a mutex with a lower number, X, after they have locked a mutex with a larger number, Y > X.
   d) Systems of type (a) and (b) only
   e) Systems of type (a), (b) and (c)

19. A database system allows process A to kill process B if the latter holds a resource that the former needs. The resource is then released and given to A. Process B is rolled back to a previous state in which it was before it acquired the resource, and is allowed to retry later. What is this mechanism an example of?
   a) An example of allowing preemption
   b) An example of a hold-and-wait condition
   c) An example of allowing circular wait
   d) An example of deadlock
   e) An example of disallowing mutually exclusive access to resources

20. Which of the following lock request sequences results in a deadlock? Assume that no locks are released in each of the sequences below. Assume no resources were locked at the beginning of each sequence. Assume that a request for a lock succeeds in acquiring the lock if it has not already been acquired by a previous request in the sequence. A request for locked resource blocks. Each sequence is a different case, independent from the other sequences. Processes are denoted by letters A, B, C, … and locks are denoted by letters R1, R2, R3, …
   a) A requests R1. B requests R1. C requests R1
   b) A requests R1. B requests R2. C requests R3. D requests R1
   d) A requests R1. A requests R2. B requests R2. C requests R1
   e) None of the above deadlocks

Good luck!