1. (True of False) Applications compiled on one operating system can be directly executable on other operating systems due to common structure.

False

2. (True of False) For a single-processor system, there will never be more than one process in the Running state.

True

3. (True of False) init is the very first process for a typical Linux system.

True

4. A message-passing model is ____

A) easier to implement than a shared memory model for intercomputer communication
B) faster than the shared memory model
C) a network protocol, and does not apply to operating systems
D) only useful for small simple operating systems

Answer : A

5. Which of the following is the correct program type for GRUB in Linux and Unix systems?

A) bootstrap program
B) compiler program
C) binder program
D) system utility

Answer: A

6. When a process fails, the operating system takes a _____ which can then be probed by a debugger for failure analysis?

A) capture of network statistic information
B) capture of CPU usage information
C) crash dump
D) core dump

Answer : D

7. The list of processes waiting to execute on a CPU is called a(n) ____.

A) standby queue

B) device queue

C) ready queue

D) interrupt queue

Answer : C

8. If process P_0 is switched to process P_1, state for P_0 will be saved into____, and state from ___ will be reloaded?

A) PCB_0, PCB_0

B) PCB_0, PCB_1

C) PCB_1, PCB_0

D) PCB_1, PCB_1

Answer : B

9. What is done by command "*ps auxl*" on Unix and Linux systems?

A) list complete information for all processes currently active in the system

B) list complete information for all processes currently running in background

C) list complete information for all files currently open in the system

D) list complete information for all folders currently open in the system

Answer: A

10. Of the following five forms of storage, rank them from fastest to slowest in terms of access time: (1) main memory, (2) magnetic disk, (3) registers, (4) solid state disk, (5) cache.

3 - 5 - 1 - 4 - 2

11. Child processes inherit UNIX ordinary pipes from their parent process because: A) The pipe is part of the code and children inherit code from their parents. B) A pipe is treated as a file descriptor and child processes inherit open file descriptors from their parents. C) The STARTUPINFO structure establishes this sharing. D) All IPC facilities are shared between the parent and child processes.

Answer: B

12. Explain why an operating system can be viewed as a resource allocator.

An Operating System (OS) can be viewed as a resource allocator because the system's CPU, memory space, file-storage space, and I/O devices are among the resources that the operating system must manage. The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.

To be Specific, the operating system is responsible for the following activities in connection with process management:

- creating and deleting processes
- scheduling processes and threads on CPUs
- suspending and resuming processes
- providing mechanisms for process synchronization and communication

For memory space, the OS is responsible for:

- keeping track of which part of memory is being used and which process is using them
- allocating and deallocating memory when needed
- deciding which part of data/process to move in/out of the memory

For file-storage space, the OS is responsible for :

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto mass storage
- Backing up files on stable (nonvolatile) storage media

For I/O device, the OS is responsible as :

- A memory-management component that includes buffering, caching, and spooling
- A general device-driver interface
- Drivers for specific hardware devices

Note: Answer referred from text book - Operating system concepts, 10th edition, chapter I.

13. What is a bootstrap program, and where is it stored?

Bootstrap program is a program that loaded at power-up or reboot, it initializes the system, loads operating system kernel and starts execution.

Bootstrap program is typically stored at ROM or EPROM, generally known as firmware

14. There are two different ways that commands can be processed by a command interpreter. One way is to allow the command interpreter to contain the code needed to execute the command. The other way is to implement the commands through system programs. Compare and contrast the two approaches.

In the first way, the CLI jump to the corresponding code upon user request, execute the code and then return control back to the user, the logic is simple and, because all the needed code is embedded in the CLI, the execution speed is fast. However, if the user want to have a new command, one has to rewrite the CLI for it to contain the necessary code, which can be messy and with more commands it can be complicated and large in memory.

In the second way, the CLI fetch and load the corresponding code in memory to execute the program. Compared to directly loading the code from itself, this approach can be slower in speed and may also involve more complicated message passing models. The advantage is the new commands can be executed without updating the CLI.

15. Explain why a modular kernel may be the best of the current operating system design techniques.

A modular kernel design combines the best of both monolithic kernel (which linux is using) and microkernel:

First of all, as the majority of the machines now have a monolithic kernel design, which is by its nature - storing under the same memory address - unsafe, optimizing it with modular kernel design can effectively separate different parts of the kernel and create a safe environment with allowance for the kernel to recover failure from its subsystems.

Additionally, the advantage of modular kernel design can be dynamic loading and unloading, thus maximizing the resource allocation efficiency ; and with modular kernel design, the kernel doesn't need to re-compile every time with new updates, the drivers can now be developed and distributed separately from the kernel.

16. Name and describe the different states that a process can exist in at any given time.

Processes can have such states:

new: the process is being created
running: the instructions are being executed.
waiting: the process is waiting for some event to occur
ready: the process is waiting to be assigned to a processor
terminated: the process has finished execution

17. Explain the concept of a context switch.

The context switch is the operation performed by OS to switch CPU from one process to another, which involves saving the context of the current process and then loading the saved context for the new process, after the new process finishes execution, the context switch loads the previously saved context again

18. Ordinarily the exec() system call follows the fork(). Explain what would happen if a programmer were to inadvertently place the call to exec() before the call to fork().

The exec(), if called before fork, will overwrite the process and the exec'd data and code will be thrown away, a brand new process will be created to replace the old one at the same memory address... however, the fork() system call will not be executed since it had been thrown away with the old process.

19. Describe how UNIX and Linux manage orphan processes.

orphan process is created when its parent process terminates before it does. The UNIX and LINUX system assign the init process, which is the ancestor of all processes, to be the parent of the orphan process

After the init process adopted the orphan process, it continues to function normally, and the init process periodically checks it and performs a wiat() like system call on them once it has finished executing, to retrieve their allocated resources.

20. Explain the term marshalling.

Marshalling is the process of converting the memory representation of an object to a data format that is suitable for storage or transmission, especially between different runtimes.