Q.1 List five services provided by an operating system that are designed to make it more convenient for users to use the computer system. In what cases it would be impossible for user-level programs to provide these services? Explain.

Answer: **Program execution**. The operating system loads the contents (or sections) of a file into memory and begins its execution. A user-level program could not be trusted to properly allocate CPU time.

• **I/O operations**. Disks, tapes, serial lines, and other devices must be communicated with at a very low level. The user need only specify the device and the operation to perform on it, while the system converts that request into device- or controller-specific commands. User-level programs cannot be trusted to only access devices they should have access to and to only access them when they are otherwise unused.

• **File-system manipulation**. There are many details in file creation, deletion, allocation, and naming that users should not have to perform. Blocks of disk space are used by files and must be tracked. Deleting a file requires removing the name file information and freeing the allocated blocks. Protections must also be checked to assure proper file access. User programs could neither ensure adherence to protection methods nor be trusted to allocate only free blocks and deallocate blocks on file deletion.

• **Communications**. Message passing between systems requires messages be turned into packets of information, sent to the network controller, transmitted across a communications medium, and reassembled by the destination system. Packet ordering and data correction must take place. Again, user programs might not coordinate access to the network device, or they might receive packets destined for other processes.

• **Error detection**. Error detection occurs at both the hardware and software levels. At the hardware level, all data transfers must be inspected to ensure that data have not been corrupted in transit. All data on media must be checked to be sure they have not changed since they were written to the media. At the software level, media

must be checked for data consistency; for instance, do the number of allocated and unallocated blocks of storage match the total number on the device. There, errors are frequently process-independent (for instance, the corruption of data on a disk), so there must be a global program (the operating system) that handles all types of errors. Also, by having errors processed by the operating system, processes need not contain code to catch and correct all the errors possible on a system.

Q.2 What are the advantages and disadvantages of using the same systemcall interface for manipulating both files and devices?
Answer: Each device can be accessed as though it was a file in the file system. Since most of the kernel deals with devices through this file interface, it is relatively easy to add a new device driver by implementing the hardware-specific code to support this abstract file interface. Therefore, this benefits the development of both user program code, which can bewritten to access devices and files in the same manner, and device driver code, which can be written to support a well-defined API. The disadvantage with using the same interface is that it might be difficult to capture the functionality of certain devices within the context of the file access API, thereby either resulting in a loss of functionality or a loss of performance. Some of this could be overcome by the use of ioctl operation that provides a general purpose interface for processes to invoke operations on devices.

Q.3 Describe the actions taken by a kernel to context-switch between processes.
Answer: In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

Q.4 Provide two programming examples in which multithreading doesnot provide better performance than a single-threaded solution.
Answer: (1) Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

Q.5 Describe the actions taken by a thread library to context switch between user-level threads.
Answer: Context switching between user threads is quite similar to switching between kernel threads, although it is dependent on the threads library and how it maps user threads to kernel threads. In general, context switching between

user threads involves taking a user thread of its LWP and replacing it with another thread. This act typically involves saving and restoring the state of the registers.

Q.6 Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?
Answer: When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multi-threaded solution would perform better even on a single-processor system.

Q.7 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system?
Answer: A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

Q.8 Discuss how the following pairs of scheduling criteria conflict in certain settings.

a. CPU utilization and response time
b. Average turnaround time and maximum waiting time
c. I/O device utilization and CPU utilization

Answer:

- CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.
- Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy

could however starve long-running tasks and thereby increase their waiting time.

• I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

Q.9 What is the meaning of the termbusy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether?Explain your answer.

Answer: *Busy waiting* means that a process is waiting for a condition to be satisfied in a tight loop without relinquish the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

Q.10 Consider the deadlock situation that could occur in the dining-philosophers problem when the philosophers obtain the chopsticks one at a time. Discuss how the four necessary conditions for deadlock indeed hold in this setting. Discuss how deadlocks could be avoided by eliminating any one of the four conditions.

Answer: Deadlock is possible because the four necessary conditions hold in the following manner: 1) mutual exclusion is required for chopsticks, 2) the philosophers tend to hold onto the chopstick in hand while they wait for the other chopstick, 3) there is no preemption of chopsticks in the sense that a chopstick allocated to a philosopher cannot be forcibly taken away, and 4) there is a possibility of circularwait. Deadlocks could be avoided by overcoming the conditions in the following manner: 1) allow simultaneous sharing of chopsticks, 2) have the philosophers relinquish the first chopstick if they are unable to obtain the other chopstick, 3) allow for chopsticks to be forcibly taken away if a philosopher has had a chopstick for a long period of time, and 4) enforce a numbering of the chopsticks and always obtain the lower numbered chopstick before obtaining the higher numbered one.

Q.11 Under what circumstances would a user be better off using a timesharing system rather than a PC or single-user workstation?

Answer: When there are few other users, the task is large, and the hardware is fast, time-sharing makes sense. The full power of the system can be brought to bear on the user's problem. The problem can be solved faster than on a personal computer. Another case occurs when lots of other users need resources at the same time.
A personal computer is best when the job is small enough to be executed reasonably on it and when performance is sufficient to execute the program to the user's satisfaction.

Q.12 Answer: a. The four necessary conditions for a deadlock are (1) mutual exclusion; (2) hold-and-wait; (3) no preemption; and (4) circular wait.
The mutual exclusion condition holds as only one car can occupy a space in the roadway. Hold-and-wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly, there is indeed a circular wait as each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.
b. A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear they will not be able to immediately clear the intersection.

Q.13 Answer: Follow the note.