Chapter 1

1.1 How are network computers different from traditional personal computers? Describe some usage scenarios in which it is advantageous to use network computers

Answer(book): A network computer relies on a centralized computer for most of its services. It can therefore have a minimal operating system to manage its resources. A personal computer on the other hand has to be capable of providing all of the required functionality in a stand-alone manner without relying on a centralized manner. Scenarios where administrative costs are high and where sharing leads to more efficient use of resources are precisely those settings where network computers are preferred.

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

Answer(book): 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.

1.8 Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs.

Answer(book): The processor could keep track of what locations are associated with each process and limit access to locations that are outside of a program's extent. Information regarding the extent of a program's memory could be maintained by using base and limits registers and by performing a check for every memory access.

1.14 Discuss, with examples, how the problem of maintaining coherence of cached data manifests itself in the following processing environments:
a. Single-processor systems
b. Multiprocessor systems
c. Distributed systems

Answer(book): In single-processor systems, the memory needs to be updated when a processor issues updates to cached values. These updates can be performed immediately or in a lazy manner. In a multiprocessor system, different processors might be caching the same memory location in its local caches. When updates are made, the other cached locations need to be invalidated or updated. In distributed systems, consistency of cached memory values is not an issue. However, consistency problems might arise when a client caches file data.

1.22 Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?

Answer(book): Symmetric multiprocessing treats all processors as equals, and I/O can be processed on any CPU. Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes tasks among the slaves, and I/O is usually done by the master only. Multiprocessors can save money by not duplicating power supplies, housings, and peripherals.

They can execute programs more quickly and can have increased reliability. They are also more complex in both hardware and software than uniprocessor systems.

1.24 What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

Answer (book): An interrupt is a hardware-generated change of flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction. A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.

1.25 Consider an SMP system similar to what is shown in Figure 1.6. Illustrate with an example how data residing in memory could in fact have two different values in each of the local caches.

Answer(book): Say processor 1 reads data A with value 5 from main memory into its local cache. Similarly, processor 2 reads data Ainto its local cache as well. Processor 1 then updates A to 10. However, since A resides in processor 1's local cache, the update only occurs there and not in the local cache for processor 2.

Chapter 2

2.1 What are the five major activities of an operating system with regard to file management?

Answer(book):

* `Creation and deletion of files.`
* `Creation and deletion of directories.`
* `Supporting primitives for manipulating files and directories.`
* `Mapping the files onto secondary storage.`
* `Backing up files on nonvolatile storage media.`

2.6 Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

Answer: Seu trabalho 01.

2.10 What is the main advantage of the layered approach to system design? What are the disadvantages of using the layered approach?

Answer: As in all cases of modular design, designing an operating system in a modular way has several advantages. The system is easier to debug and modify because changes affect only limited sections of the system rather than touching all sections of the operating system. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer. Some of the disadvantages include:difficulty with the layered approach involves careful definition of the layer; it tends to be inefficient in some circumstances because operations may involve several layers.

2.17 In what ways is the modular kernel approach similar to the layered approach? In what ways does it differ from the layered approach?

Answer(net): The modular kernel approach requires subsystems to interact with each other through carefully constructed interfaces that are typically narrow (in terms of the functionality that is exposed to external modules). The layered kernel approach is similar in that respect. However, the layered kernel imposes a strict ordering of subsystems such that subsystems at the lower layers are not allowed to invoke operations corresponding to the upper-layer subsystems. There are no such restrictions in the modular-kernel approach, herein modules are free to invoke each other without any constraints.

2.19 What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices?

Answer(book): 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 be written 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 resulting in either a loss of functionality or a loss of performance. Some of this could be overcome by the use of the ioctl operation that provides a general-purpose interface for processes to invoke operations on devices.

2.21 Why do some systems store the operating system in firmware, while others store it on disk?

Answer(book): For certain devices, such as handheld PDAs and cellular telephones, a disk with a file system may be not be available for the device. In this situation, the operating system must be stored in firmware.

Chapter 3

3.2 Consider the RPC mechanism. Describe the undesirable consequences that could arise from not enforcing either the "at most once" or "exactly once" semantic. Describe possible uses for a mechanism that has neither of these guarantees.

Answer (net): If an RPC mechanism cannot support either the “at most once” or “at least once” semantics, then the RPC server cannot guarantee that a remote procedure will not be invoked multiple occurrences. Consider if a remote procedure were withdrawing money from a bank account on a system that did not support these semantics. It is possible that a single invocation of the remote procedure might lead to multiple withdrawals on the server. For a system to support either of these semantics generally requires the server maintain some form of client state such as the timestamp described in the text. If a system were unable to support either of these semantics, then such a system could only safely provide remote procedures that do not alter data or provide time-sensitive results. Using our bank account as an example, we certainly require “at most once” or “at least once” semantics for performing a withdrawal (or deposit!). However, an inquiry into an account balance or other account information such as name, address, etc. does not require these semantics.

3.3 With respect to the RPC mechanism, consider the "exactly once" semantic. Does the algorithm for implementing this semantic execute correctly even if the ACK message back to the client is lost due to a network problem? Describe the sequence of messages and discuss whether "exactly once" is still preserved.

Answer (book): The “exactly once” semantics ensure that a remore procedure will be executed exactly once and only once. The general algorithm for ensuring this combines an acknowledgment (ACK) scheme combined with timestamps (or some other incremental counter that allows the server to distinguish between duplicate messages). The general strategy is for the client to send the RPC to the server along with a timestamp. The client will also start a timeout clock. The client will then wait for one of two occurrences: (1) it will receive an ACK from the server indicating that the remote procedure was performed, or (2) it will time out. If the client times out, it assumes the server was unable to perform the remote procedure so the client invokes the RPC a second time, sending a later timestamp. The client may not receive the ACK for one of two reasons: (1) the original RPC was never received by the server, or (2) the RPC was correctly received—and performed—by the server but the ACK was lost. In situation (1), the use of ACKs allows the server ultimately to receive and perform the RPC. In situation (2), the server will receive a duplicate RPC and it will use the timestamp to identify it as a duplicate so as not to perform the RPC a second time. It is important to note that the server must send a second ACK back to the client to inform the client the RPC has been performed.

3.6 The Sun UltraSPARC processor has multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

Answer (book): The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and be moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

3.7 Construct a process tree similar to Figure 3.9. To obtain process information for the UNIX or Linux system, use the command ps -ael. Use the command man ps to get more information about the ps command. On Windows systems, you will have to use the task manager.

3.9 Describe the differences among short-term, medium-term, and long-term scheduling.

Answer(book): Short-term (CPU scheduler)—selects from jobs in memory those jobs that are ready to execute and allocates the CPU to them. Medium-term—used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.· Long-term (job scheduler)—determines which jobs are brought into memory for processing. The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

3.10 Including the initial parent process, how many processes are created by the program below?

```
#include  <stdio.h>
#include <unistd.h>
int main(){
/* fork a child process*/
fork();
/* fork another child process*/
fork();
/*and fork another*/
```

```
fork();
return 0;
}
```

Answer (book): There are 8 processes created.

Chapter 4

4.1 Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

Answer:

a. Qualquer programa com fortes dependências entre tarefas ou sequencial

b. Aplicações intensivas em dados utilizando o mesmo dispositivo de I/O .

4.2 Write a multithreaded Java, Pthreads, or Win32 program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.

4.3 Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values

b. Heap memory

c. Global variables

d. Stack memory

Answer (book): The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

4.6 What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

Answer (book):

a. User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.

b. On systems using either M:1 or M:N mapping, user threads are scheduled by the thread library and the kernel schedules kernel threads.

c. Kernel threads need not be associated with a process whereas every user thread belongs to a process. Kernel threads are generally more expensive to maintain than user threads as they must be represented with a kernel data structure.

4.10 What resources are used when a thread is created? How do they differ from those used when a process is created?

Answer (book): Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

4.15 Describe the actions taken by a thread library to context-switch between user-level threads.

Answer (book): Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.