Answers to Exercises

1. 1. What to you feel are the limitations of a computer that has no operating system? How would a user load and execute a program?

Ans.

The operating system does so much for the user. Without it, either the user or the application would be responsible for process management, memory management, I/O and protection. Each program the user wanted to run would need its own driver for the various hardware devices (video card, sound card, hard drive, etc.). If users needed to compile a program before executing it, they would need to load the compiler first, and then compile. To execute a program, the user would need to include the appropriate job control language statements. If virtual memory were to be used, the user would need to divide the program up into overlays manually. The user interface to the machine would be nonexistent.

---

2. Microkernels attempt to provide as small a kernel as possible, putting much of the operating system support into additional modules. What do you feel are the minimum services that the kernel must provide?

Ans.

The microkernel supports only those services that are necessary to allow other system services (executed in user space) to run. A typical microkernel should support creating address spaces, low-level management of processes and threads (create, terminate, suspend, resume, priorities, fork, synchronization, etc.), interprocess communication (message passing), and interrupt handling (possibly virtual memory management and I/O, depending on system).

---

3. If you were writing code for a real-time operating system, what restrictions might you want to impose on the system?

Ans.

The system performance should be, to the greatest extent, deterministic. Anything that might affect the predictable execution time must be carefully designed. The system must be designed with a short interrupt latency and fast context switching. A RTOS that provides virtual memory support must provide a way for a process (task) to lock its code and data into real memory, so that it can guarantee predictable response time. All task scheduling must be done within strict time bounds. RTOSs are often present in embedded systems, and, depending on the system, this might require a small footprint for the OS.

---

4. What is the difference between multiprogramming and multiprocessing? Multiprogramming and multithreading?

Ans.

Multiprogramming means having multiple programs (processes) running on one processor, whereas multiprocessing means having multiple processors. Multiprogramming typically refers to multiple processes from multiple users, whereas multithreading generally refers to multiple threads from one user process.

♦5. Under what circumstances is it desirable to collect groups of processes and programs into subsystems running on a large computer? What advantages would there be to creating logical partitions on this system?

Ans.

If processes share a specific set of resources, it might be reasonable to group them as a subsystem. If a given set of processes is used for testing the system, it would be wise to group them as a subsystem as if they crash or do something "weird", only the subsystem in which they are running is affected. If you are giving access to a specific group of people

for a limited time or to limited resources, you may want these user processes to be grouped as a subsystem as well.

---

6. What advantages would there be to using both subsystems and logical partitions on the same machine?

Ans.

Subsystems establish logically distinct environments that can be individually configured and managed, whereas LPARs create distinct machines within a given physical system. Subsystems cannot provide low-level segmentation of the machine and its resources. By adding LPARs to a given subsystem, more protection can be provided for the resources within that subsystem.

♦7. When is it appropriate to use nonrelocatable binary program code? Why is relocatable code preferred?

Ans.

Nonrelocatable code is often used when code needs to be more compact. Therefore, it is common in embedded systems which have space constraints (such as your microwave or you car computer). Nonrelocatable code is also faster, so it is used in systems that are sensitive to small timing delays, such as real-time systems. Relocatable code requires hardware support, so nonrelocatable code would be used in those situations that might not have that support (such as in Nintendo).

---

8. Suppose there were no such thing as relocatable program code. How would the process of memory paging be made more complex?

Ans.

If code were not relocatable, any pages that were swapped out would need to be loaded into the exact same location each time. Not only would this be inefficient use of memory, but it might also create thrashing situations.

♦9. Discuss the advantages and disadvantages of dynamic linking.

Ans.

Dynamic linking saves disk space, results in fewer system errors, and allows for code sharing. However, dynamic linking can cause load-time delays, and if dynamic link library routines are changed, others using modified libraries could end up with difficult bugs to track down.

---

10. What problems does an assembler have to overcome in order to produce complete binary code in one pass over the source file? How would code written for a one-pass assembler be different from code written for a two-pass assembler?

Ans.

The key problem that has to be addressed is forward referencing (an address is referenced before it is defined). One solution would be to require all data values to be declared before being used. This won't help with forward references to labels for such things as jumps. Another would be to use some sort of list (or other mechanism) to handle forward references for jumps and subroutine calls.

---

11. Why should assembly language be avoided for general application development? Under what circumstances is assembly language preferred or required?

Ans.

Assembly programming is more difficult to program and more time consuming to debug than higher-level languages. Assembly language programs are also harder to maintain, as there are more statements for any given amount of functionality that in the equivalent high-level language program. If execution time or space are critical issues, portions of code can be rewritten in assembly language (although most compilers today do a very good job of optimizing the object code). There are typically small chunks of code that are run many times, and these should be optimized so they don't become bottlenecks. Generally between 10-20% of the code requires 80-90% of the execution time; it is these code segments that should be investigated. Regarding space: if a programmer is experienced in assembly language, he or she can generally cut out everything that doesn't need to be in the code, resulting in less code. Lastly, it might be that the HLL one is using cannot access the machine operations or device in the manner necessary. In these situations, assembly language is very useful.

---

12. Under what circumstances would you argue in favor of using assembly language code for developing an application program?

Ans.

For the most efficient programs (in terms of both time and space), some assembly code must be written by hand (this might mean the entire program, or certain portions of code). So applications where performance and footprint really matters are good candidates for assembly language. These include embedded applications, some graphics-oriented applications (such as games), and real-time applications. In addition, code written for a specific machine could most likely be optimized by hand (there are machine-dependent optimizations that compilers won't utilize).

---

13. What are the advantages of using a compiled language over an interpreted one? Under what circumstances would you choose to use an interpreted language?

Ans.

Interpreters do not optimize as nicely as compilers do. In addition, compiled code is generally faster than interpreted code. Interpreted languages are also less in control of the hardware as a general rule. Compilers are better in production environments (for many reasons, not the least of which is that with interpreted languages, errors may go unnoticed, since an error is not detected unless that particular statement is actually executed), situations in which the programs are repetitive (will be run many times), and situations in which execution speed and size matter. So, if you want to save memory space, increase speed, and have enhanced control over the hardware, use a compiled language. Interpreters are useful in a development and testing phase (since large programs can take a long time to compile, simple changes in compiled languages can result in long waits, whereas in with interpreted languages programs can easily be debugged and modified), for programs that are run once and then discarded, or where the execution speed is secondary to another factor better handled by an interpreted language (for example, communication costs versus execution speed). Interpreted code is often more portable (as long as the correct machine-specific interpreter is installed, the same source code can be run on different platforms). In addition, everyone can read and modify the source code for interpreted programs, which may be a plus or a minus. Interpreted languages are typically used for scripting.

---

14. Discuss the following questions relative to compilers:
    a) Which phase of a compiler would give you a syntax error?
    b) Which phase complains about undefined variables?
    c) If you try to add an integer to a character string, which compiler phase would emit the error message?

Ans.

a) Lexical analyzer

b) Syntax analysis

c) Semantic analyzer

---

15. Why is the execution environment of a Java class called a virtual machine? How does this virtual machine compare to a real machine running code written in C?

Ans.

The JVM is a virtual computer, implemented in software on top of a "real" hardware platform. Code written in C on a real machine is first compiled to machine language, and then this machine language is run on the real machine. Java, on the other hand, is first compiled to bytecode; this bytecode is then run on the virtual machine. Java bytecode can be thought of as the machine language for the JVM, but it is not real machine language.

---

16. Why do you suppose that the method area of a JVM is global to all of the threads running in the virtual machine environment?

Ans.

The method area is where class variable data structures and program statements required by the class reside. It may also contain the entire runtime representation of a class. If it weren't global, all of this would need to be repeated for each thread. One of the major advantages of Java is its support for multithreading, which centers around coordinating access to data shared among multiple threads.

---

17. We stated that only one method at a time can be active within each thread running in the JVM. Why do you think that this is the case?

Ans.

When a thread invokes a method, it creates a memory frame for the method, where part of this frame is used for the method's local variables, and another part for its private stack. After the thread has defined the method stack, it pushes the method's parameters and points its program counter to the first executable statement of the method. If more than one method were active, there would be multiple copies of these entities, which would imply synchronization and consistency issues.

---

18. The Java bytecode for access to the local variable array for a class is at most two bytes long. One byte is used for the opcode, the other indicates the offset into the array. How many variables can be held in the local variable array? What do you think happens when this number is exceeded?

Ans.

This is an unsigned integer which would limit the local variables to 256. In the rare case that this number of locals is exceeded, the bits would just overflow and the value MOD 256 would be used as the pointer. (The compiler could use a different instruction -- with the same effect-- which had a 16-bit unsigned operand.)

♦19. Java is called an interpreted language, yet Java is a compiled language that produces a binary output stream. Explain how this language can be both compiled and interpreted.

Ans.

The Java source code is first compiled into a standard, platform independent bytecode file. This file is then interpreted at runtime by a machine-specific interpreter (the Java Virtual Machine).

---

20. We stated that the performance of a Java program running in the JVM cannot possibly match that of a regular compiled language. Explain why this is so.

Ans.

Interpreted code tends to run slower than compiled code because each statement must be analyzed before it can be "executed" (thus resulting in "interpretive overhead"). Interpreted programs are run inside the interpreting program, so the process of interpretation is in essence a program running a program. In fully interpreted languages, access to variables is also slower because the process of mapping identifiers to memory locations must be done over and over at run time (not once by building a symbol table as in done with a compiler). Although Java, which is compiled and interpreted, tends to run faster than other interpreted code, is it typically not as fast as compiled native machine code.

♦21. Answer the following with respect to database processing:

- ♦ a) What is a race condition? Give an example.
- ♦ b) How can race conditions be prevented?
- ♦ c) What are the risks in race condition prevention?

Ans.

a) A race condition occurs when different computational results (e.g. output, values of data variables) occur depending on the particular timing and resulting order of execution of statements across separate threads, processes, or transactions. Suppose we have the following two transactions accessing an account with an initial balance of 500:

TransactionA	TransactionB
Get Balance of Account	Get Balance of Account
Add 100 to Balance	Subtract 100 from Balance
Store new Balance	Store new Balance

The value of the new balance depends on the order in which the transactions statements are run (interleaved).

b) Race conditions can be prevented by running transactions in isolation and providing atomicity. Atomic transactions in databases are assured via locking.

c) Using locks can result in deadlocks. Suppose Transaction T1 gets an exclusive lock on data item X (which means no other transaction can share the lock), and then Transaction T2 gets an exclusive lock on data item Y. Now, suppose T1 needs to hold on to X but now needs Y, and T2 must hold on to Y but now needs X. We have a deadlock as they are each waiting on the other and will not release the locks they have.

---

22. In what ways are n-tiered transaction processing architectures superior to single-tiered architectures? Which usually costs more?

Ans.

In n-tiered architectures, each tier is represented by a different hardware platform. This allows the best hardware suited to a particular task to be used for that task. Multitiered architectures often cost more, not necessary because of hardware, but because they can be difficult to build (communication issues, etc.)

---

23. To improve performance, your company has decided to replicate its product database across several servers so that not all transactions go through a single system. What sorts of issues will need to be considered?

Ans.

Replication introduces problems with consistency among the replications. What happens if a replicated data element X is locked for writing at one server and another transaction wants to read X at a different server? To maintain the ACID properties, locking must ensure this doesn't happen. There would also need to be a method in place to decide which transactions went where.

---

24. We said that the risk of deadlock is always present anytime a system resource is locked. Describe a way in which deadlock can occur.

Ans.

Suppose T1 has X locked, and T2 has Y locked. Now suppose T1 needs Y and T2 needs X. Neither will release the resource it has locked and is waiting on the other.

---

* 25. Research various command-line interfaces (such as Unix, MS-DOS and VMS) and different windows interfaces (such as any Microsoft windows product, MAC-OS and KDE).

a) Consider some of the major commands, such as getting a directory listing, deleting a file, or changing directories. Explain how each of these commands is implemented on the various operating systems you studied.

b) List and explain some of the commands that are easier using a command-line interface versus a GUI. List and explain some of the commands that are easier using a GUI versus using a command-line interface.

c) Which type of interface do you prefer? Why?

Ans.

No answer given.
