SOLUTIONS TO CHAPTER 8 PROBLEMS

1. Both USENET and SETI@home could be described as wide area distributed systems. However, USENET is actually more primitive than the scheme of Fig. 8-1(c), since it does not require any network infrastructure other than point-to-point connections between pairs of machines. Also, since it does no processing work beyond that necessary to ensure proper dissemination of news articles, it could be debated whether it is really a distributed system of the sort we are concerned with in this chapter. SETI@home is a more typical example of a wide area distributed system; data are distributed to remote nodes which then return results of calculations to the coordinating node.

2. Depending on how CPUs are connected to memory, one of them gets through first, for example, seizes the bus first. It completes its memory operation, then another one happens, etc. It is not predictable which one goes first, but if the system has been designed for sequential consistency, it should not matter.

3. A 200-MIPS machine will issue 200 million memory references/sec, consuming 200 million bus cycles or half of the bus' capacity. It takes only two CPUs to consume the entire bus. Caching drops the number of memory requests/sec to 20 million, allowing 20 CPUs to share the bus. To get 32 CPUs on the bus, each one could request no more than 12.5 million requests/sec. If only 12.5 million of the 200 million of the memory references go out on the bus, the cache miss rate must be 12.5/200, or 6.25%. This means the hit rate is 93.75%.

4. CPUs 000, 010, 100, and 110 are cut off from memories 010 and 011.

5. Each CPU manages its own signals completely. If a signal is generated from the keyboard and the keyboard is not assigned to any particular CPU (the usual case), somehow the signal has to be given to the correct CPU to handle.

6. To issue a system call, a process generates a trap. The trap interrupts its own CPU. Somehow, the information that a slave CPU has had a trap has to be conveyed to the master CPU. This does not happen in the first model. If there are interprocessor trap instructions, this can be used to signal the master. If no such instructions exist, the slave can collect the parameters of the system call and put them in a data structure in memory that the master polls continuously when it is idle.

7. Here is a possible solution:

```
enter_region:
        TST LOCK              | Test the value of lock
        JNE ENTER_REGION      | If it is nonzero, go try again
        TSL REGISTER,LOCK     | Copy lock to register and set lock to 1
        CMP REGISTER,#0       | Was lock zero?
```

```
JNE ENTER_REGION      | If it was nonzero, lock was set, so loop
RET                   | Return to caller; critical region entered
```

8. As noted in the text, we have little experience (and tools) for writing highly parallel desktop applications. Although desktop applications are sometimes multithreaded, the threads are often used to simplify I/O programming and thus they are not compute-intensive threads. The one desktop application area that has some chance at large-scale parallelization is video gaming, where many aspects of the game require significant (parallel) computation. A more promising approach is to parallelize operating system and library services. We have already seen examples of this in current hardware and OS designs. For example, network cards now have on-board parallel processors (network processors) that are used to speed up packet processing and offer higher-level network services at line speeds (e.g., encryption, intrusion detection, etc). As another example, consider the powerful processors found on video cards used to offload video rendering from the main CPU and offer higher-level graphics APIs to applications (e.g., Open GL). One can imagine replacing these special-purpose cards with single-chip multicore processors. Moreover, as the number of cores increases, the same basic approach can be used to parallelize other operating system and common library services.

9. Probably locks on data structures are sufficient. It is hard to imagine anything a piece of code could do that is critical and does not involve some kernel data structure. All resource acquisition and release uses data structures, for example. While it cannot be proven, probably locks on data structures are enough.

10. It takes 16 bus cycles to move the block and it goes both ways for each TSL. Thus every 50 bus cycles, 32 of them are wasted on moving the cache block. Consequently, 64% of the bus bandwidth is wasted on cache transfers.

11. Yes, it would, but the interpoll time might end up being very long, degrading performance. But it would be correct, even without a maximum.

12. It is just as good as TSL. It is used by preloading a 1 into the register to be used. Then that register and the memory word are atomically swapped. After the instruction, the memory word is locked (i.e., has a value of 1). Its previous value is now contained in the register. If it was previously locked, the word has not been changed and the caller must loop. If it was previously unlocked, it is now locked.

13. The loop consists of a TSL instruction (5 nsec), a bus cycle (10 nsec), and a JMP back to the TSL instruction (5 nsec). Thus in 20 nsec, 1 bus cycle is requested occupying 10 nsec. The loop consumes 50% of the bus.

14. Affinity scheduling has to do with putting the right thread on the right CPU. Doing so might well reduce TLB misses since these are kept inside each CPU. On the other hand, it has no effect on page faults, since if a page is in memory for one CPU, it is in memory for all CPUs.

15. (a) 2 (b) 4 (c) 8 (d) 5 (e) 3 (f) 4.

16. On a grid, the worst case is nodes at opposite corners trying to communicate. However, with a torus, opposite corners are only two hops apart. The worst case is one corner trying to talk to a node in the middle. For odd k, it takes $(k - 1)/2$ hops to go from a corner to the middle horizontally and another $(k - 1)/2$ hops to go to the middle vertically, for a total of $k - 1$. For even k, the middle is a square of four dots in the middle, so the worst case is from a corner to the most distant dot in that four-dot square. It takes $k/2$ hops to get there horizontally and also $k/2$ vertically, so the diameter is k.

17. The network can be sliced in two by a plane through the middle, giving two systems, each with a geometry of $8 \times 8 \times 4$. There are 128 links running between the two halves, for bisection bandwidth of 128 Gbps.

18. If we just consider the network time, we get 1 nsec per bit or 512-nsec delay per packet. To copy 64 bytes 4 bytes at a time, 320 nsec are needed on each side, or 640 nsec total. Adding the 512-nsec wire time, we get 1132 nsec total. If two additional copies are needed, we get 1792 nsec.

19. If we consider only the wire time, a 1-Gbps network delivers 125 MB/sec. Moving 64 bytes in 1152 nsec is 55.6 MB/sec. Moving 64 bytes in 1792 nsec is 35.7 MB/sec.

20. The time to move k bytes by programmed I/O is $20k$ nsec. The time for DMA is $2000 + 5k$ nsec. Equating these and solving for k we get the breakeven point at 133 bytes.

21. Clearly, the wrong thing happens if a system call is executed remotely. Trying to read a file on the remote machine will not work if the file is not there. Also, setting an alarm on the remote machine will not send a signal back to the calling machine. One way to handle remote system calls is to trap them and send them back to the originating site for execution.

22. First, on a broadcast network, a broadcast request could be made. Second, a centralized database of who has which page could be maintained. Third, each page could have a home base, indicated by the upper k bits of its virtual address; the home base could keep track of the location of each of its pages.

23. In this split, node 1 has *A*, *E*, and *G*, node 2 has *B* and *F*, and node 3 has *C*, *D*, *H*, and *I*. The cut between nodes 1 and 2 now contains *AB* and *EB* for a weight of 5. The cut between nodes 2 and 3 now contains *CD*, *CI*, *FI*, and *FH* for a weight of 14. The cut between nodes 1 and 3 now contains *EH* and *GH* for a weight of 8. The sum is 27.

24. The table of open files is kept in the kernel, so if a process has open files, when it is unfrozen and tries to use one of its files, the new kernel does not know about them. A second problem is the signal mask, which is also stored on the original kernel. A third problem is that if an alarm is pending, it will go off on the wrong machine. In general, the kernel is full of bits and pieces of information about the process, and they have to be successfully migrated as well.

25. Ethernet nodes must be able to detect collisions between packets, so the propagation delay between the two most widely separated nodes must be less than the duration of the shortest packet to be sent. Otherwise the sender may fully transmit a packet and not detect a collision even though the packet suffers a collision close to the other end of the cable.

26. The middleware runs on different operating systems, so the code is clearly different because the embedded system calls are different. What they have in common is producing a common interface to the application layer above them. If the application layer makes calls only to the middleware layer and no system calls, then all the versions of it can have the same source code. If they also make true system calls, these will differ.

27. The most appropriate services are

(a) Unreliable connection.
(b) Reliable byte stream.

28. It is maintained hierarchically. There is a worldwide server for *.edu* that knows about all the universities and a *.com* server that knows about all the names ending in *.com*. Thus to look up *cs.uni.edu*, a machine would first look up *uni* at the *.edu* server, then go there to ask about *cs*, and so on.

29. A computer may have many processes waiting for incoming connections. These could be the Web server, mail server, news server, and others. Some way is needed to make it possible to direct an incoming connection to some particular process. That is done by having each process listen to a specific port. It has been agreed upon that Web servers will listen to port 80, so incoming connections directed to the Web server are sent to port 80. The number itself was an arbitrary choice, but some number had to be chosen.

30. Physical I/O devices still present problems because they do not migrate with the virtual machine, yet their registers may hold state that is critical to the proper functioning of the system. Think of read or write operations to devices

(e.g., the disk) that have been issued but have not yet completed. Network I/O is particularly difficult because other machines will continue to send packets to the hypervisor, unaware that the virtual machine has moved. Even if packets can be redirected to the new hypervisor, the virtual machine will be unresponsive during the migration period, which can be long because the entire virtual machine, including the guest operating system and all processes executing on it, must be moved to the new machine. As a result packets can experience large delays or even packet loss if the device/hypervisor buffers overflow.

31. One way would be for the Web server to package the entire page, including all the images, in a big zip file and send the whole thing the first time so that only one connection is needed. A second way would be to use a connectionless protocol like UDP. This would eliminate the connection overhead, but would require servers and browsers to do their own error control.

32. Having the value of a read depend on whether a process happens to be on the same machine as the last writer is not at all transparent. This argues for making changes only visible to the process making the changes. On the other hand, having a single cache manager per machine is easier and cheaper to implement. Such a manager becomes a great deal more complicated if it has to maintain multiple copies of each modified file, with the value returned depending on who is doing the reading.

33. Shared memory works with whole pages. This can lead to false sharing, in which access to unrelated variables that happen to lie on the same page causes thrashing. Putting each variable on a separate page is wasteful. Object-based access eliminates these problems and allows a finer grain of sharing.

34. Hashing on any of the fields of the tuple when it is inserted into the tuple space does not help because the *in* may have mostly formal parameters. One optimization that always works is noting that all the fields of both *out* and *in* are typed. Thus the type signature of all tuples in the tuple space is known, and the tuple type needed on an *in* is also known. This suggests creating a tuple subspace for each type signature. For example, all the (int, int, int) tuples go into one space, and all the (string, int, float) tuples go into a different space. When an *in* is executed, only the matching subspace has to be searched.