SOLUTIONS MANUAL

OPERATING SYSTEMS
NINTH EDITION

CHAPTERS 10–20; APPENDIX A

WILLIAM STALLINGS

NOTICE

This manual contains solutions to the review questions and homework problems in *Operating Systems, Ninth Edition*. If you spot an error in a solution or in the wording of a problem, I would greatly appreciate it if you would forward the information via email to wllmst@me.net. An errata sheet for this manual, if needed, is available at http://www.box.net/shared/fa8a0oyxxl . File name is S-OS9e-mmyy.

W.S.

TABLE OF CONTENTS

CHAPTER 10 MULTIPROCESSOR, MULTICORE, AND REAL-TIME SCHEDULING

ANSWERS TO QUESTIONS

10.1 **Fine:** Parallelism inherent in a single instruction stream. **Medium:** Parallel processing or multitasking within a single application. **Coarse:** Multiprocessing of concurrent processes in a multiprogramming environment. **Very Coarse:** Distributed processing across network nodes to form a single computing environment. **Independent:** Multiple unrelated processes.

10.2 **Load sharing:** Processes are not assigned to a particular processor. A global queue of ready threads is maintained, and each processor, when idle, selects a thread from the queue. The term load sharing is used to distinguish this strategy from load-balancing schemes in which work is allocated on a more permanent basis. **Gang scheduling:** A set of related threads is scheduled to run on a set of processors at the same time, on a one-to-one basis. **Dedicated processor assignment:** Each program is allocated a number of processors equal to the number of threads in the program, for the duration of the program execution. When the program terminates, the processors return to the general pool for possible allocation to another program. **Dynamic scheduling:** The number of threads in a program can be altered during the course of execution.

10.3 **First Come First Served (FCFS):** When a job arrives, each of its threads is placed consecutively at the end of the shared queue. When a processor becomes idle, it picks the next ready thread, which it executes until completion or blocking. **Smallest Number of Threads First:** The shared ready queue is organized as a priority queue, with highest priority given to threads from jobs with the smallest number of unscheduled threads. Jobs of equal priority are ordered according to which job arrives first. As with FCFS, a scheduled thread is run to completion or blocking. **Preemptive Smallest Number of Threads First:** Highest priority is given to jobs with the smallest number of unscheduled threads. An arriving job with a smaller number of threads

than an executing job will preempt threads belonging to the scheduled job.

10.4 A **hard real-time task** is one that must meet its deadline; otherwise it will cause undesirable damage or a fatal error to the system. A **soft real-time task** has an associated deadline that is desirable but not mandatory; it still makes sense to schedule and complete the task even if it has passed its deadline.

10.5 An **aperiodic task** has a deadline by which it must finish or start, or it may have a constraint on both start and finish time. In the case of a **periodic task**, the requirement may be stated as "once per period *T*" or "exactly *T* units apart."

10.6 **Determinism:** An operating system is deterministic to the extent that it performs operations at fixed, predetermined times or within predetermined time intervals. **Responsiveness:** Responsiveness is concerned with how long, after acknowledgment, it takes an operating system to service the interrupt. **User control:** The user should be able to distinguish between hard and soft tasks and to specify relative priorities within each class. A real-time system may also allow the user to specify such characteristics as the use of paging or process swapping, what processes must always be resident in main memory, what disk transfer algorithms are to be used, what rights the processes in various priority bands have, and so on. **Reliability:** Reliability must be provided in such a way as to continue to meet real-time deadlines. **Fail-soft operation:** Fail-soft operation is a characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible.

10.7 **Static table-driven approaches:** These perform a static analysis of feasible schedules of dispatching. The result of the analysis is a schedule that determines, at run time, when a task must begin execution. **Static priority-driven preemptive approaches:** Again, a static analysis is performed, but no schedule is drawn up. Rather, the analysis is used to assign priorities to tasks, so that a traditional priority-driven preemptive scheduler can be used. **Dynamic planning-based approaches:** Feasibility is determined at run time (dynamically) rather than offline prior to the start of execution (statically). An arriving task is accepted for execution only if it is feasible to meet its time constraints. One of the results of the feasibility analysis is a schedule or plan that is used to decide when to dispatch this task. **Dynamic best effort approaches:** No feasibility analysis is performed. The system tries to meet all deadlines and aborts any started process whose deadline is missed.

10.8 **Ready time:** time at which task becomes ready for execution. In the case of a repetitive or periodic task, this is actually a sequence of times that is known in advance. In the case of an aperiodic task, this time may be known in advance, or the operating system may only be aware when the task is actually ready. **Starting deadline:** Time by which a task must begin. **Completion deadline:** Time by which task must be completed. The typical real-time application will either have starting deadlines or completion deadlines, but not both. **Processing time:** Time required to execute the task to completion. In some cases, this is supplied. In others, the operating system measures an exponential average. For still other scheduling systems, this information is not used. **Resource requirements:** Set of resources (other than the processor) required by the task while it is executing. **Priority:** Measures relative importance of the task. Hard real-time tasks may have an "absolute" priority, with the system failing if a deadline is missed. If the system is to continue to run no matter what, then both hard and soft real-time tasks may be assigned relative priorities as a guide to the scheduler. **Subtask structure:** A task may be decomposed into a mandatory subtask and an optional subtask. Only the mandatory subtask possesses a hard deadline.

ANSWERS TO PROBLEMS

10.1 For fixed priority, we do the case in which the priority is A, B, C. Each square represents five time units; the letter in the square refers to the currently running process. The first row is fixed priority; the second row is earliest deadline scheduling using completion deadlines.

A	A	B	B	A	A	C	C	A	A	B	B	A	A	C	C	A	A		
A	A	B	B	A	C	C	A	C	A	A	B	B	A	A	C	C	C	A	A

For fixed priority scheduling, process C always misses its deadline.

10.2 Each square represents 10 time units.

Earliest deadline

Earliest deadline with unforced idle times

FCFS

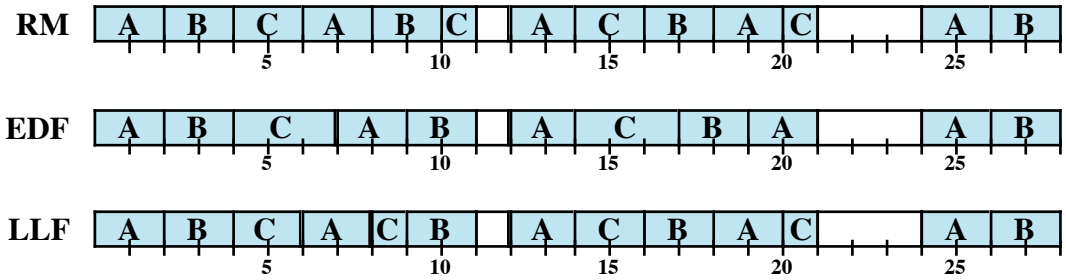
	A	A		C	C	E	E	D	D		
		B	B	C	C	E	E	D	D	A	A
	A	A		C	C	D	D				

10.3 **a.** The task may be delayed up to an interval of t and still meet its deadline.

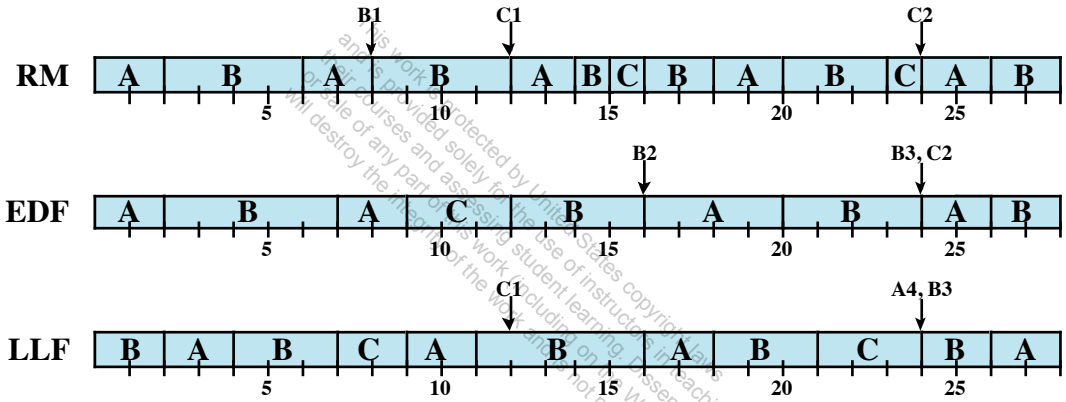
b. A laxity of 0 means that the task must be executed now or will fail to meet its deadline.

c. A task with negative laxity cannot meet its deadline.

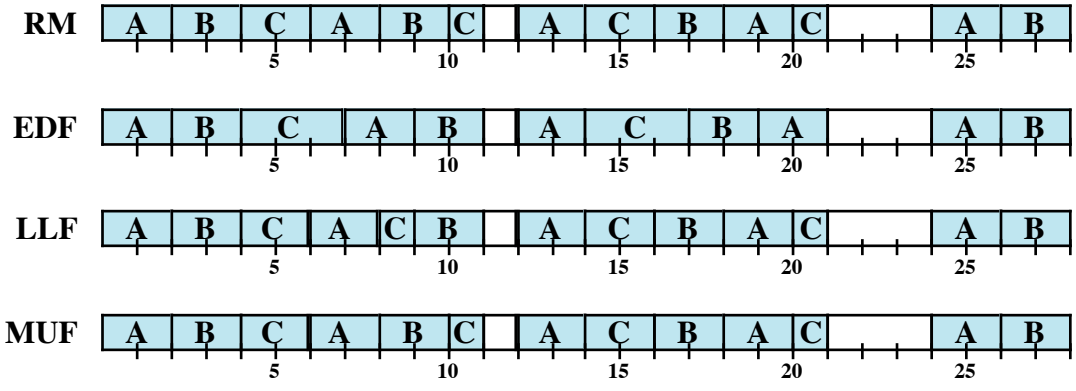
d. The task set has a total load of 83%. The resulting schedules differ in the number of (potentially costly) context switches: 13, 11, and 13, respectively. Any timeline repeats itself every 24 time units. Although the total load (83%) is higher than the RM limit for this case (78%), RM can schedule the set satisfactorily.



10.4 The total load now exceeds 100%. None of the methods can handle the load. The tasks that fail to complete vary for the various methods.

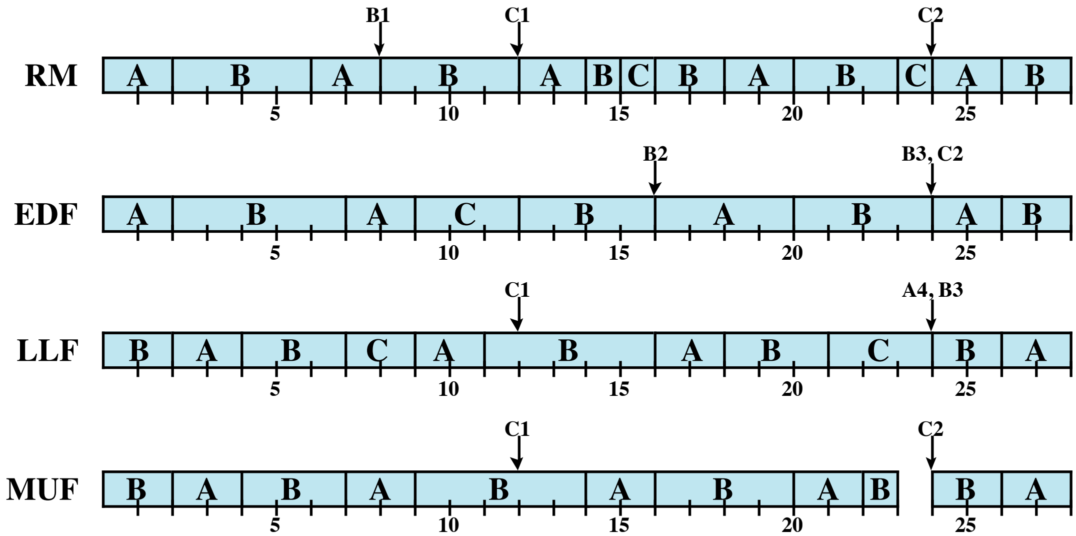


10.5 On this task profile, MUF behavior is identical to RM. Note that EDF has fewer context switches (11) than the other methods, all of which have 13.



7

10.6 The critical task set consists of A and B. Note that only MUF is the only algorithm that gets the critical set (A and B) scheduled in time.

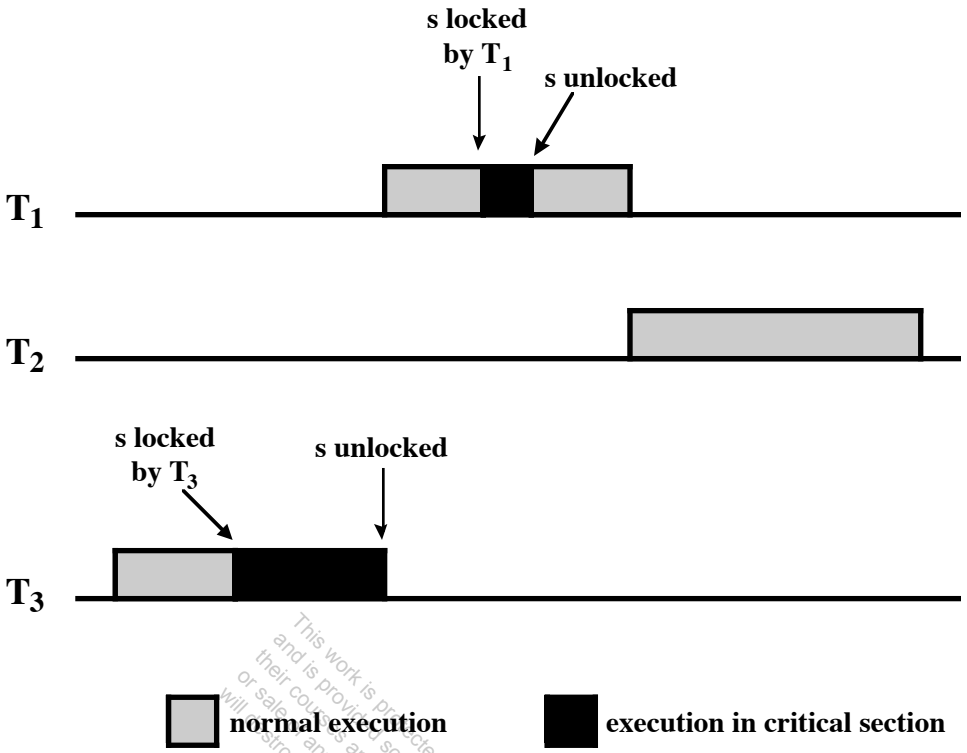


10.7 a. The total utilization of P_1 and P_2 is 0.41, which is less than 0.828, the bound given for two tasks by Equation 10.2. Therefore, these two tasks are schedulable.

b. The utilization of all the tasks is 0.86, which exceeds the bound of 0.779.

c. Observe that P_1 and P_2 must execute at least once before P_3 can begin executing. Therefore, the completion time of the first instance of P_3 can be no less than 20 + 30 + 68 = 118. However, P1 is initiated one additional time in the interval (0, 118). Therefore, P_3 does not complete its first execution until 118 + 20 = 138. This is within the deadline for P_3. By continuing this reasoning, we can see that all deadlines of all three tasks can be met.

10.8



Once T_3 enters its critical section, it is assigned a priority higher than T1. When T3 leaves its critical section, it is preempted by T_1.

CHAPTER 11 I/O MANAGEMENT AND DISK SCHEDULING

ANSWERS TO QUESTIONS

11.1 **Programmed I/O:** The processor issues an I/O command, on behalf of a process, to an I/O module; that process then busy-waits for the operation to be completed before proceeding. **Interrupt-driven I/O:** The processor issues an I/O command on behalf of a process, continues to execute subsequent instructions, and is interrupted by the I/O module when the latter has completed its work. The subsequent instructions may be in the same process, if it is not necessary for that process to wait for the completion of the I/O. Otherwise, the process is suspended pending the interrupt and other work is performed. **Direct memory access (DMA):** A DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

11.2 **Logical I/O:** The logical I/O module deals with the device as a logical resource and is not concerned with the details of actually controlling the device. The logical I/O module is concerned with managing general I/O functions on behalf of user processes, allowing them to deal with the device in terms of a device identifier and simple commands such as open, close, read, write. **Device I/O:** The requested operations and data (buffered characters, records, etc.) are converted into appropriate sequences of I/O instructions, channel commands, and controller orders. Buffering techniques may be used to improve utilization.

11.3 **Block-oriented** devices stores information in blocks that are usually of fixed size, and transfers are made one block at a time. Generally, it is possible to reference data by its block number. Disks and tapes are examples of block-oriented devices. **Stream-oriented** devices transfer data in and out as a stream of bytes, with no block structure. Terminals, printers, communications ports, mouse and other pointing devices, and most other devices that are not secondary storage are stream oriented.

11.4 Double buffering allows two operations to proceed in parallel rather than in sequence. Specifically, a process can transfer data to (or from) one buffer while the operating system empties (or fills) the other.

11.5 Seek time, rotational delay, access time.

11.6 **FIFO:** Items are processed from the queue in sequential first-come-first-served order. **SSTF:** Select the disk I/O request that requires the least movement of the disk arm from its current position. **SCAN:** The disk arm moves in one direction only, satisfying all outstanding requests en route, until it reaches the last track in that direction or until there are no more requests in that direction. The service direction is then reversed and the scan proceeds in the opposite direction, again picking up all requests in order. **C-SCAN:** Similar to SCAN, but restricts scanning to one direction only. Thus, when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again.

11.7 **0:** Non-redundant **1:** Mirrored; every disk has a mirror disk containing the same data. **2:** Redundant via Hamming code; an error-correcting code is calculated across corresponding bits on each data disk, and the bits of the code are stored in the corresponding bit positions on multiple parity disks. **3:** Bit-interleaved parity; similar to level 2 but instead of an error-correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks. **4:** Block-interleaved parity; a bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk. **5:** Block-interleaved distributed parity; similar to level 4 but distributes the parity strips across all disks. **6:** Block-interleaved dual distributed parity; two different parity calculations are carried out and stored in separate blocks on different disks.

11.8 512 bytes.

ANSWERS TO PROBLEMS

11.1 If the calculation time exactly equals the I/O time (which is the most favorable situation), both the processor and the peripheral device running simultaneously will take half as long as if they ran separately. Formally, let C be the calculation time for the entire program and let T be the total I/O time required. Then the best possible running time with buffering is max(C, T), while the running time without buffering is C + T; and of course $((C + T)/2) \leq \max(C, T) \leq (C + T)$.

11.2 The best ratio is (n + 1):n.

11.3 a. Disk head is initially moving in the direction of decreasing track number:

FIFO		SSTF		SCAN		C-SCAN	
Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed
27	73	110	10	64	36	64	36
129	102	120	10	41	23	41	23
110	19	129	9	27	14	27	14
186	76	147	18	10	17	10	17
147	39	186	39	110	100	186	176
41	106	64	122	120	10	147	39
10	31	41	23	129	9	129	18
64	54	27	14	147	18	120	9
120	56	10	17	186	39	110	10
Average	61.8	Average	29.1	Average	29.6	Average	38

b. If the disk head is initially moving in the direction of increasing track number, only the SCAN and C-SCAN results change:

SCAN		C-SCAN	
Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed
110	10	110	10
120	10	120	10
129	9	129	9
147	18	147	18
186	39	186	39
64	122	10	176
41	23	27	17
27	14	41	14
10	17	64	23
Average	29.1	Average	35.1

11.4 It will be useful to keep the following representation of the N tracks of a disk in mind:

0	1	• • •	j – 1	• • •	N – j	• • •	N – 2	N – 1

a. Let us use the notation Ps [j/t] = Pr [seek of length j when head is currently positioned over track t]. Recognize that each of the N tracks is equally likely to be requested. Therefore the unconditional probability of selecting any particular track is 1/N. We can then state:

$$\text{Ps}[j/t] = \frac{1}{N} \quad \text{if} \quad t \le j - 1 \ \text{ OR } \ t \ge N - j$$

$$\text{Ps}[j/t] = \frac{2}{N} \quad \text{if} \quad j - 1 < t < N - j$$

In the former case, the current track is so close to one end of the disk (track 0 or track N – 1) that only one track is exactly j tracks away. In the second case, there are two tracks that are exactly j tracks away from track t, and therefore the probability of a seek of length j is the probability that either of these two tracks is selected, which is just 2/N.

b. Let Ps [K] = Pr [seek of length K, independent of current track position]. Then:

$$\text{Ps}[K] = \sum_{t=0}^{N-1} \text{Ps}[K/t] \times \Pr[\text{current track is track t}]$$
$$= \frac{1}{N} \sum_{t=0}^{N-1} \text{Ps}[K/t]$$

From part (a), we know that Ps[K/t] takes on the value 1/N for 2K of the tracks, and the value 2/N for (N – 2K) of the tracks. So

$$\text{Ps}[K] = \frac{1}{N}\left(\frac{2K}{N} + \frac{2(N-K)}{N}\right) = \frac{2}{N^2}(N-K)$$

c.

$$\text{E}[K] = \sum_{K=0}^{N-1} K \times \text{Ps}[K] = \sum_{K=0}^{N-1} \frac{2K(N-K)}{N^2}$$

$$= \frac{2}{N} \sum_{K=0}^{N-1} K - \frac{2}{N^2} \sum_{K=0}^{N-1} K^2$$

$$= (N+1) - \frac{2(N+1)(2N+1)}{6N}$$

$$= \frac{N^2 - 1}{3N}$$

d. This follows directly from the last equation.

11.5 Define

A_i = Time to retrieve a word from memory level i

H_i = Probability that a word is in memory level i and no higher-level memory

B_i = Time to transfer a block of data from memory level (i + 1) to memory level i

Let cache be memory level 1; main memory, memory level 2, and so on, for a total of N levels of memory. Then, we can write

$$T_s = \sum_{i=1}^{N} A_i H_i$$

Now, recognize that if a word is in M_1 (cache), it is read immediately. If it is in M_2 but not in M_1, then a block of data is transferred from M_2 to M_1 and then read. Thus $A_2 = B_1 + A_1$

Further $A_3 = B_2 + A_2 = B_1 + B_2 + A_1$

Generalizing, $$A_i = A_1 + \sum_{j=1}^{i-1} B_j$$

So $$T_s = T_1 \sum_{i=1}^{N} H_i + \sum_{L=2}^{N} \sum_{J=1}^{L-1} B_j H_i$$

But $$\sum_{i=1}^{N} H_i = 1$$

Finally $$T_s = T_1 + \sum_{L=2}^{N} \sum_{J=1}^{L-1} B_j H_i$$

11.6 a. The middle section in Figure 11.11b is empty. Thus, this reduces to the strategy of Figure 11.11a.

b. The old section consists of one block, and we have the LRU replacement policy.

11.7 Each sector can hold 4 logical records. The required number of sectors is 300,000/4 = 75,000 sectors. This requires 75,000/96 = 782 tracks, which in turn requires 782/110 = 8 surfaces.

11.8 There are 512 bytes/sector. Since each byte generates an interrupt, there are 512 interrupts. Total interrupt processing time = $2.5 \times 512 = 1280$ μs. The time to read one sector is:

((60 sec/min) / (360 rev/min)) / (96 sectors/track)
= 0.001736 sec = 1736 μs

Percentage of time processor spends handling I/O:

$(100) \times (1280/1736) = 74\%$

11.9 With DMA, there is only one interrupt of 2.5 μs. Therefore, the percentage of time the processor spends handling I/O is

$$(100) \times (2.5/1736) = 0.14\%$$

11.10 Only one device at a time can be serviced on a selector channel. Thus,

Maximum rate = $800 + 800 + 2 \times 6.6 + 2 \times 1.2 + 10 \times 1 = 1625.6$ KBytes/sec

11.11 It depends on the nature of the I/O request pattern. On one extreme, if only a single process is doing I/O and is only doing one large I/O at a time, then disk striping improves performance. If there are many processes making many small I/O requests, then a nonstriped array of disks should give comparable performance to RAID 0.

11.12

RAID 0: 800 GB	RAID 4: 600 GB
RAID 1: 400 GB	RAID 5: 600 GB
RAID 3: 600 GB	RAID 6: 400 GB

CHAPTER 12 FILE MANAGEMENT

ANSWERS TO QUESTIONS

12.1 A **field** is the basic element of data containing a single value. A **record** is a collection of related fields that can be treated as a unit by some application program.

12.2 A **file** is a collection of similar records, and is treated as a single entity by users and applications and may be referenced by name. A **database** is a collection of related data. The essential aspects of a database are that the relationships that exist among elements of data are explicit and that the database is designed for use by a number of different applications.

12.3 A file management system is that set of system software that provides services to users and applications in the use of files.

12.4 Rapid access, ease of update, economy of storage, simple maintenance, reliability.

12.5 **Pile:** Data are collected in the order in which they arrive. Each record consists of one burst of data. **Sequential file:** A fixed format is used for records. All records are of the same length, consisting of the same number of fixed-length fields in a particular order. Because the length and position of each field is known, only the values of fields need to be stored; the field name and length for each field are attributes of the file structure. **Indexed sequential file:** The indexed sequential file maintains the key characteristic of the sequential file: records are organized in sequence based on a key field. Two features are added; an index to the file to support random access, and an overflow file. The index provides a lookup capability to reach quickly the vicinity of a desired record. The overflow file is similar to the log file used with a sequential file, but is integrated so that records in the overflow file are located by following a pointer from their predecessor record. **Indexed file:** Records are accessed only through their indexes. The result is that there is now no restriction on the placement of records as long as a pointer in at least one index refers to that record. Furthermore, variable-length records can be employed. **Direct, or hashed, file:** The direct file makes use of hashing on the key value.

12.6 In a sequential file, a search may involve sequentially testing every record until the one with the matching key is found. The indexed sequential file provides a structure that allows a less exhaustive search to be performed.

12.7 Search, create file, delete file, list directory, update directory.

12.8 The pathname is an explicit enumeration of the path through the tree-structured directory to a particular point in the directory. The working directory is a directory within that tree structure that is the current directory that a user is working on.

12.9 None, knowledge of, read, write, execute, change protection, delete.

12.10 **Fixed blocking:** Fixed-length records are used, and an integral number of records are stored in a block. There may be unused space at the end of each block. This is referred to as internal fragmentation. **Variable-length spanned blocking:** Variable-length records are used and are packed into blocks with no unused space. Thus, some records must span two blocks, with the continuation indicated by a pointer to the successor block. **Variable-length unspanned blocking:** Variable-length records are used, but spanning is not employed. There is wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.

12.11 **Contiguous allocation:** a single contiguous set of blocks is allocated to a file at the time of file creation. **Chained allocation:** allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. **Indexed allocation:** the file allocation table contains a separate one-level index for each file; the index has one entry for each portion allocated to the file.

ANSWERS TO PROBLEMS

12.1 **Fixed blocking:** F = largest integer $\leq \frac{B}{R}$

When records of variable length are packed into blocks, data for marking the record boundaries within the block has to be added to separate the records. When spanned records bridge block boundaries, some reference to the successor block is also needed. One possibility is a length indicator preceding each record. Another possibility is a special separator marker between records. In any case, we can assume that each record requires a marker, and we assume that the size of a marker is about equal to the size of a block pointer. For

spanned blocking, a block pointer of size P to its successor block may be included in each block, so that the pieces of a spanned record can easily be retrieved. Then we have

Variable-length spanned blocking: $F = \frac{B - P}{R + P}$

With unspanned variable-length blocking, an average of R/2 will be wasted because of the fitting problem, but no successor pointer is required:

Variable-length unspanned blocking: $F = \frac{B - \frac{R}{2}}{R}$

12.2 **a.** $\log_2 \frac{N}{F}$

b. Less than half the allocated file space is unused at any time.

12.3 **a.** Indexed

b. Indexed sequential

c. Hashed or indexed

12.4 The result is identical to Figure 12.5 d, except for the following: At the lowest level, the rightmost block contains 90, 96, 97. The block to the left of the rightmost block contains 73, 85.

12.5 **a.** This technique allows us to insert a key into a B-tree in a single pass down the tree from the root to a leaf. This saves time in the current operation.

b. The principal disadvantage is that the tree is subdivided and, potentially, new layers are added to the tree unnecessarily. This may slow down future searches.

12.6 $h \leq \log_d \frac{n+1}{2}$

Proof: The root of a B-tree T contains at least two pointers, and all other nodes contain at least *d* pointers. Thus T, whose height is *h*, has at least 2 nodes at depth 1, at least 2*d* nodes at depth 2, at least $2d^2$ nodes at depth 3, and so on, until at depth h it has at least $2d^{h-1}$ nodes. Thus, the number of keys n satisfies:

$$n \geq 1 + (d-1)\sum_{i=1}^{h} 2d^{i-1}$$

$$n \geq 1 + 2(d-1)\left(\frac{d^h - 1}{d-1}\right)$$

$$n \geq 2d^h - 1$$

12.7

File size	41,600 bytes	640,000 bytes	4,064,000 bytes
No. of blocks	3 (rounded up to whole number)	40	249
Total capacity	$3 \times 16K = 48K = 49,152$ bytes	$40 \times 16K = 640K = 655,360$ bytes	$249 \times 16K = 3984K = 4,079,616$ bytes
Wasted space	7,552 bytes	15,360 bytes	15,616 bytes
% of wasted space	15.36%	2.34%	0.38%

12.8 Directories enable files to be organized and clearly separated on the basis of ownership, application, or some other criterion. This improves security, integrity (organizing backups), and avoids the problem of name clashes.

12.9 Clearly, security is more easily enforced if directories are easily recognized as special files by the operating system. Treating a directory as an ordinary file with certain assigned access restrictions provides a more uniform set of objects to be managed by the operating system and may make it easier to create and manage user-owned directories.

12.10 This is a rare feature. If the operating system structures the file system so that subdirectories are allowed underneath a master directory, there is little or no additional logic required to allow arbitrary depth of subdirectories. Limiting the depth of the subdirectory tree places an unnecessary limitation on the user's organization of file space.

12.11 **a.** Yes. the method employed is very similar to that used by many LISP systems for garbage collection. First we would establish a

data structure representing every block on a disk supporting a file system. A bit map would be appropriate here. Then, we would start at the root of the file system (the "/" directory), and mark every block used by every file we could find through a recursive descent through the file system. When finished, we would create a free list from the blocks remaining as unused. This is essentially what the UNIX utility `fsck` does.

b. Keep a "backup" of the free-space list pointer at one or more places on the disk. Whenever this beginning of the list changes, the "backup" pointers are also updated. This will ensure you can always find a valid pointer value even if there is a memory or disk block failure.

12.12

Level	Number of Blocks	Number of Bytes
Direct	10	10K
Single indirect	256	256K
Double indirect	$256 \times 256 = 65K$	65M
Triple indirect	$256 \times 65K = 64M$	16G

The maximum file size is over 16G bytes.

12.13 **a.** Find the number of disk block pointers that fit in one block by dividing the block size by the pointer size:

$$8K/4 = 2K \text{ pointers per block}$$

The maximum file size supported by the inode is thus:

12	+ 2K	+ $(2K \times 2K)$	+ $(2K \times 2K \times 2K)$
Direct	Indirect – 1	Indirect – 2	Indirect – 3
12	+ 2K	+ 4M	+ 8G blocks

Which, when multiplied by the block size (8K), is

$$96KB + 16MB + 32GB + 64TB$$

Which is HUGE.

b. There are 24 bits for identifying blocks within a partition, so that leads to:

$$2^{24} \times 8K = 16M \times 8K = 128 \text{ GB}$$

c. Using the information from (a), we see that the direct blocks only cover the first 96KB, while the first indirect block covers the next 16MB. the requested file position is 13M and change, which clearly falls within the range of the first indirect block. There will thus be

two disk accesses. One for the first indirect block, and one for the block containing the required data.

CHAPTER 13 EMBEDDED OPERATING SYSTEMS

ANSWERS TO QUESTIONS

13.1 A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In many cases, embedded systems are part of a larger system or product, as in the case of an antilock braking system in a car.

13.2 • Small to large systems, implying very different cost constraints, thus different needs for optimization and reuse
• Relaxed to very strict requirements and combinations of different quality requirements, for example, with respect to safety, reliability, real-time, flexibility, and legislation
• Short to long life times
• Different environmental conditions in terms of, for example, radiation, vibrations, and humidity
• Different application characteristics resulting in static versus dynamic loads, slow to fast speed, compute versus interface intensive tasks, and/or combinations thereof
• Different models of computation ranging from discrete-event systems to those involving continuous time dynamics (usually referred to as hybrid systems)

13.3 An embedded OS is an OS designed for the embedded system environment.

13.4 • **Real time operation:** In many embedded systems, the correctness of a computation depends, in part, on the time at which it is delivered. Often, real-time constraints are dictated by external I/O and control stability requirements.
• **Reactive operation:** Embedded software may execute in response to external events. If these events do not occur periodically or at predictable intervals, the embedded software may need to take into account worst-case conditions and set priorities for execution of routines.
• **Configurability:** Because of the large variety of embedded systems, there is a large variation in the requirements, both qualitative and

quantitative, for embedded OS functionality. Thus, and embedded OS intended for use on a variety of embedded systems lend itself to flexible configuration so that only the functionality needed for a specific application and hardware suite is provided.

• **I/O device flexibility:** There is virtually no device that needs to be supported by all versions of the OS, and the range of I/O devices is large.

• **Streamlined protection mechanisms:** Embedded systems are typical designed for a limited, well-defined functionality. Untested programs are rarely added to the software. After the software has been configured and tested, it can be assumed to be reliable. Thus, apart from security measures, embedded systems have limited protection mechanisms. For example, I/O instructions need not be privileged instructions that trap to the OS; tasks can directly perform their own I/O. Similarly, memory protection mechanisms can be minimized.

• **Direct use of interrupts:** General-purpose operating systems typically do not permit any user process to use interrupts directly.

13.5 An advantage of an embedded OS based on an existing commercial OS compared to a purpose-built embedded OS is that the embedded OS derived from a commercial general-purpose OS is based on a set of familiar interfaces, which facilitates portability. The disadvantage of using a general-purpose OS is that it is not optimized for real-time and embedded applications. Thus, considerable modification may be required to achieve adequate performance.

13.6 A wireless sensor network.

13.7 •**Allow high concurrency:** In a typical wireless sensor network application, the devices are concurrency intensive. Several different flows of data must be kept moving simultaneously. While sensor data is input in a steady stream, processed results must be transmitted in a steady stream. In addition, external controls from remote sensors or base stations must be managed.

• **Operate with limited resources:** The target platform for TinyOS will have limited memory and computational resources and run of batteries or solar power. A single platform may offer only kilobytes of program memory and hundreds of bytes of RAM. The software must make efficient use of the available processor and memory resources while enabling low power communication.

• **Adapt to hardware evolution:** Mote hardware is in constant evolution; applications and most system services must be portable across hardware generations. Thus, should be possible to upgrade the hardware with little or no software change, if the functionality is the same.

• **Support a wide range of applications:** Applications exhibit a wide range of requirements in terms of lifetime, communication, sensing, and so on. A modular, general-purpose embedded OS is desired so that a standardized approach leads to economies of scale in developing applications and support software.
• **Support a diverse set of platforms:** As with the preceding point, a general-purpose embedded OS is desirable.
• **Be robust:** Once deployed, a sensor network must run unattended for months or years. Ideally, there should be redundancy both within a single system and across the network of sensors. However, both types of redundancy require additional resources. One software characteristic that can improve robustness is to use highly modular, standardized software components.

13.8 An embedded software system built using TinyOS consists of a set of small modules, called components, each of which performs a simple task or set of tasks, and which interface with each other and with hardware in limited and well-defined ways.

13.9 The scheduler plus a number of components.

13.10 The default scheduler in TinyOS is a simple FIFO (first-in-first-out) queue.

ANSWERS TO PROBLEMS

13.1 The basic reason is that the system is almost completely idle. You only need intelligent scheduling when a resource is under contention.

13.2 **a.** Clients are not able to monopolize the resource queue by making multiple requests.

b. The basic reason is that a component may still hold a lock after it calls release. It may still hold the lock because transferring control to the next holder requires reconfiguring the hardware in a split-phase operation. The lock can be granted to the next holder only after this reconfiguration occurs. This means that technically, a snippet of code such as

```
release()
request()
```

can be seen as the lock as re-request. There are other solutions to the problem (e.g., give the lock a special "owner" who holds it while reconfiguring), but this raises issue when accounting for CPU cycles or energy.

13.3 This ensures proper exclusion with code running on both other processors and this processor.

13.4 Wherever this is called from, this operation effectively pauses the processor until it succeeds. This operations should therefore be used sparingly, and in situations where deadlocks cannot occur.

13.5 There should be no processors attempting to claim the lock at the time this function is called, otherwise the behavior is undefined.

13.6 The mutex should be unlocked and there should be no threads waiting to lock it when this call in made.

13.7 Timeslicing within a given priority level is irrelevant since there can be only one thread at each priority level.

13.8 If a thread has locked a mutex and then attempts to lock the mutex again, typically as a result of some recursive call in a complicated call graph, then either an assertion failure will be reported or the thread will deadlock. This behavior is deliberate. When a thread has just locked a mutex associated with some data structure, it can assume that that data structure is in a consistent state. Before unlocking the mutex again it must ensure that the data structure is again in a consistent state. Recursive mutexes allow a thread to make arbitrary changes to a data structure, then in a recursive call lock the mutex again while the data structure is still inconsistent. The net result is that code can no longer make any assumptions about data structure consistency, which defeats the purpose of using mutexes.

13.9 a. This listing is an example using a condition variable. Thread A is acquiring data that are processed by Thread B. First, B executes. On line 24, B acquires the mutex associated with the condition variable. There is no data in the buffer and `buffer_empty` is initialized so `true`, so B calls `cyg_cond_wait` on line 26. This call suspends B waiting for the condition variable to be set and it unlocks the mutex `mut_cond_var`. Once A has acquired data, it sets `buffer_empty` to false (line 11). A then locks the mutex (line 13), signals the condition variable (line 15), and then unlocks the mutex (line 17). B can now run. Before returning from `cyg_cond_wait`, the mutex `mut_cond_var` is locked and owned by B. B can now get the data buffer (line 28) and set `buffer_empty` to `true` (line 31). Finally, the mutex is released by B (line 33) and the data in the buffer is processed (line 35)

b. If this code were not atomic, it would be possible for B to miss the signal call from A even though the buffer contained data. This is because `cyg_cond_wait` first checks to see if the condition variable

is set and, in this case, it is not. Next, the mutex is released in the `cyg_cond_wait` call. Now, A executes, putting data into the buffer and then signals the condition variable (line 15). Then, B returns to waiting. However, the condition variable has been set.

c. This ensures that the condition that B is waiting on is still true after returning from the condition wait call. This is needed in case there are other threads waiting on the same condition.

13.10 Even if the two threads were running at the same priority, the one attempting to claim the spinlock would spin until it was timesliced and a lot of processor time would be wasted. If an interrupt handler tried to claim a spinlock owned by a thread, the interrupt handler would loop forever.

CHAPTER 14 VIRTUAL MACHINES

ANSWERS TO QUESTIONS

14.1 A type 1 hypervisor is loaded as a software layer directly onto a physical server, much like an OS is loaded. The type 1 hypervisor can directly control the physical resources of the host. Once it is installed and configured, the server is then capable of supporting virtual machines as guests. A type 2 hypervisor exploits the resources and functions of a host OS and runs as a software module on top of the OS. It relies on the OS to handle all of the hardware interactions on the hypervisor's behalf.

14.2 A virtualization container runs on top of the host OS kernel and provides an isolated execution environment for applications. Unlike hypervisor-based VMs, containers do not aim to emulate physical servers. Instead, all containerized applications on a host share a common OS kernel. This eliminates the resources needed to run a separate OS for each application and can greatly reduce overhead.

14.3 The hypervisor can use some portion of that allocated memory for another VM by reclaiming older pages that are not being used. The reclamation process is done through **ballooning**. The hypervisor activates a balloon driver that (virtually) inflates and presses the guest OS to flush pages to disk. Once the pages are cleared, the balloon driver deflates and the hypervisor can use the physical memory for other VMs. This process happens during times of memory contention.

14.4 The goal of a Java Virtual Machine is to provide a runtime space for a set of Java code to run on any OS staged on any hardware platform, without needing to make code changes to accommodate the different OSs or hardware. The JVM is described as being an abstract computing machine, consisting of an **instruction set**, a pc (program counter) **register**, a **stack** to hold variables and results, a **heap** for runtime data and garbage collection, and a **method** area for code and constants. The JVM can support multiple threads and each thread has its own register and stack areas, though the heap and method areas are shared among all of the threads.

ANSWERS TO PROBLEMS

14.1 The answer is no. The aggregate of the virtual machines that are hosted on a single host cannot exceed the physical abilities of the hardware they reside upon. Virtualization facilitates more efficient use of the server resources – memory, CPU, network and storage IO – but the physical limits of the equipment are the ceiling of what can be done.

14.2 Type-2 hypervisors offer the benefits of virtualization into work areas that are typically not supporting full time applications, or want to run virtualized workloads along side other applications that are not virtualized on the same system. The widest use cases are in development, testing, support, and education. Each of these provides a standard user desktop (Windows, Linux) for normal work, but the hypervisor allows the user to then create and use other workspaces for transitory functions that require specific or specialized environments. For example, a software developer can stand up multiple environments to create and test, without having to worry about affecting their main working environment. A suspended or shutdown virtual machine can be passed to someone else as a set of files for validation or troubleshooting. A support organization could provide a library of 'standard release' virtual machines that could be used by technical engineers to resolve support issues instead of requiring a hardware infrastructure for each possibility. Type-2 hypervisors allow students to work in separate environments for different classes, starting up virtual machines to have network or database applications available for them, or for specific math, science, or other applications, with having them installed on their system where they might negatively interact with each other or consume resources when they aren't in use.

14.3 Consolidating multiple workloads on a single server drove the need for servers with more physical CPUs, more bandwidth, and more RAM. If an average physical server's CPU was only 5% busy, at 80% busy, you could stack sixteen virtual machines in the identical hardware. In that case, though, instead of 4GB of RAM, the server would have 32GB or 64GB to insure adequate resources. Additional network connections would need to be provided in the form of more network interface cards (NICs) to handle the additional network and storage IO. Initially, memory or RAM was often the gating factor of how many virtual machines could be hosted on a physical server, but with today's systems offering a terabyte or more, there are instances with CPU is now becoming the limit.

14.4 Converged networks bundle large amounts of traffic and different types of traffic on the same network connections. This potentially can mix user application data with network management traffic and storage IO together on the same wires. One possible issue with this is

that as packets on a wire, there is no way to prioritize types of traffic so that a low priority function might consume large amounts of bandwidth to the detriment of other more important applications. Physical networks manage this in a variety of ways using quality-of-service techniques to provide dedicated bandwidth, guarantee levels of service to specific traffic types or data streams, and reduce data retransmission due to packet loss. Initially, this same degree of control and sophistication was not available inside of the virtual networks hypervisors could provide. Today, however, virtual networks are every bit as powerful as their physical counterparts offering the ability to granularly control network resources to insure that same quality-of-service capability.

14.5 Storage and storage subsystems are a very mature technology and there are few items that are a part of the virtual infrastructure that can optimize storage. Most of the existing storage optimization best practices are applicable to a virtualized environment so techniques such as de-duplication or thin-provisioning translate from physical to a virtual environment with great efficacy. Continued hardware improvements like solid state drives (SSDs) are also extremely effective allowing virtual administrators to tactically use SSDs for paging space, or VM staging for rapid deployments of identical desktop clones. Storage tiering is another technology beginning to show wide adoption, allowing different classes of service to be delivered based on policy and need. Some innovation has occurred in the offloading area where older hypervisors did the work of creating files systems and initialization on storage arrays. This would force the host server to devote CPU cycles to storage maintenance. Newer hypervisors take advantage of storage array intelligence, offloading the tasks to the array through APIs, allowing the array to perform the work it was designed to do, many times faster than through the hypervisor and without needing to consume additional host CPU cycles.

CHAPTER 15 OPERATING SYSTEM SECURITY

ANSWERS TO QUESTIONS

15.1 None, knowledge of, read, write, execute, change protection, delete.

15.2 **Masquerader:** An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account. **Misfeasor:** A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges. **Clandestine user:** An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

15.3 **Something the individual knows:** Examples includes a password, a personal identification number (PIN), or answers to a prearranged set of questions.
Something the individual possesses: Examples include electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a token.
Something the individual is (static biometrics): Examples include recognition by fingerprint, retina, and face.
Something the individual does (dynamic biometrics): Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

15.4 **Discretionary access control (DAC)** controls access based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do. This policy is termed *discretionary* because an entity might have access rights that permit the entity, by its own volition, to enable another entity to access some resource. **Role-based access control (RBAC)** controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles. RBAC may have a discretionary or mandatory mechanism.

15.5 The programming languages vulnerable to buffer overflows are those without a very strong notion of the type of variables, and what

constitutes permissible operations on them. They include assembly language, and C and similar languages. Strongly typed languages such as Java, ADA, Python, and many others are not vulnerable to these attacks.

15.6 Two broad categories of defenses against buffer overflows are: compile-time defenses which aim to harden programs to resist attacks in new programs; and run-time defenses which aim to detect and abort attacks in existing programs.

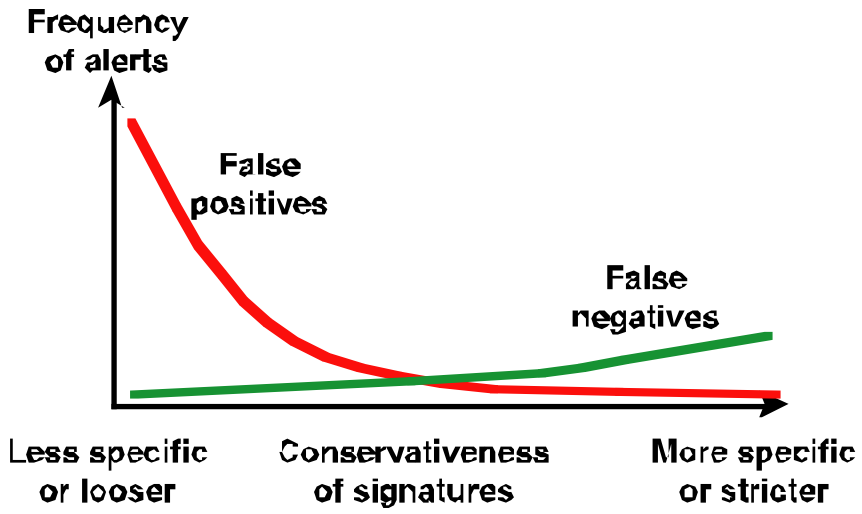
15.7 Compile-time defenses include: writing programs using a modern high-level programming language that is not vulnerable to buffer overflow attacks; using safe coding techniques to validate buffer use; using language safety extensions and/or safe library implementations; or using stack protection mechanisms.

15.8 Run-time defenses that provide some protection for existing vulnerable programs include: using "Executable Address Space Protection" that blocks execution of code on the stack, heap, or in global data; using "Address Space Randomization" to manipulate the location of key data structures such as the stack and heap in the processes address space; or by placing **guard pages** between critical regions of memory in a processes address space.

ANSWERS TO PROBLEMS

15.1 If a process running as root is compromised, then any child-processes the attacker spawns (such as a remote shell) will also run as root. If such a process can be used to read data, it will be able to read any local file. If such a process spawns a process that listens on UDP or TCP ports, that process can be bound to privileged ports (TCP 22, TCP 80, and all other TCP/UDP ports lower than 1024). If such a process is merely sloppily coded, the impact of bug behavior may be much greater than if the process didn't run as root (overwriting important files, interfering with other processes, etc.).

15.2 This is a typical example:

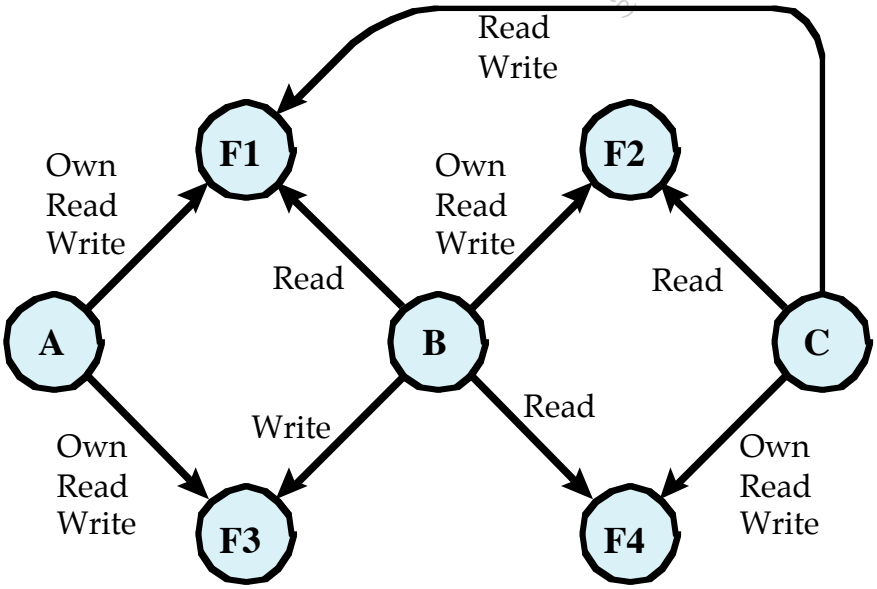


15.3 Corrected version of the program shown in Figure 15.2a (see bold text):

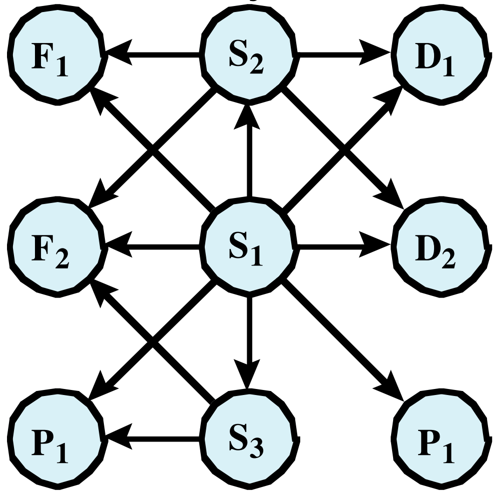
```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    fgets(str2, sizeof(str2), stdin);
    if (strncmp(str1, str2, sizeof(str2)) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```


15.4 a.



b. For simplicity and clarity, the labels are omitted. Also, there should be arrowed lines from each subject node to itself.



c. A given access matrix generates only one directed graph, and a given directed graph yields only one access matrix, so the correspondence is one-to-one.

15.5 The 'find' command allows for searches on the permissions, using the 'perm' option, '-perm +4000' will find all files with the SUID bit set. The full command, with output to file, is:

```
find / -perm +4000 > somelogfile
```

To find both SUID and SGID files (and to show the symbolic form of perms) you can use:

```
find / -perm +u+s -o -perm +g+s > somelogfile
```

Once a list of files has been generated, the system operator can check the modification time and other statistics (like size, owner) to ensure the file is the genuine article. However, this is still a manual process, prone to human error. System Security Tools (like "tripwire") could be applied to the files in the list to actively monitor for modifications.

15.6

```
drwxr-x---    2 ahmed staff    0 Jul 21 07:58 stuff
-rw-rw----    1 ahmed staff    0 Jul 21 08:00 ourstuff
```

NOTE: it's acceptable (albeit redundant) for the sticky bit to be set on "stuff," so long as the write bit is not. If you don't want group-users to *either* create or delete files in a directory, it's sufficient to simply turn off the write bit.

15.7 Suppose that the directory **d** and the file **f** have the same owner and group and that **f** contains the text *something*. Disregarding the superuser, no one besides the owner of **f** can change its contents can change its contents, because only the owner has write permission. However, anyone in the owner's group has write permission for **d**, so

that any such person can remove **f** from **d** and install a different version, which for most purposes is the equivalent of being able to modify **f**. This example is from Grampp, F., and Morris, R. "UNIX Operating System Security." *AT&T Bell Laboratories Technical Journal*, October 1984.

15.8 A default UNIX file access of full access for the owner combined with no access for group and other means that newly created files and directories will only be accessible by their owner. Any access for other groups or users must be explicitly granted. This is the most common default, widely used by government and business where the assumption is that a person's work is assumed private and confidential.

A default of full access for the owner combined with read/execute access for group and none for other means newly created files and directories are accessible by all members of the owner's group. This is suitable when there is a team of people working together on a server, and in general most work is shared with the group. However there are also other groups on the server for which this does not apply. An organization with cooperating teams may choose this.

A default of full access for the owner combined with read/execute access for both group and other means newly created files and directories are accessible by all users on the server. This is appropriate for organization's where users trust each other in general, and assume that their work is a shared resource. This used to be the default for University staff, and in some research labs. It is also often the default for small businesses where people need to rely on and trust each other.

15.9 In order to provide the Web server access to a user's 'public_html' directory, then search (execute) access must be provided to the user's home directory (and hence to all directories in the path to it), read/execute access to the actual Web directory, and read access to any Web pages in it, for others (since access cannot easily be granted just to the user that runs the web server). However this access also means that any user on the system (not just the web server) has this same access. Since the contents of the user's web directory are being published on the web, local public access is not unreasonable (since they can always access the files via the web server anyway). However in order to maintain these required permissions, if the system default is one of the more restrictive (and more common) options, then the user must set suitable permissions every time a new directory or file is created in the user's web area. Failure to do this means such directories and files are not accessible by the server, and hence cannot be access over the web. This is a common error. As well the fact that

at least search access is granted to the user's home directory means that some information can be gained on its contents by other users, even if it is not readable, by attempting to access specific names. It also means that if the user accidentally grants too much access to a file, it may then be accessible to other users on the system. If the user's files are sufficiently sensitive, then the risk of accidental leakage due to inappropriate permissions being set may be too serious to allow such a user to have their own web pages.

15.10 a. $\sum_{i=1}^{N}(U_i \times P_i)$

b. $\sum_{i=1}^{N}(U_i + P_i)$

15.11 Logs provide audit trails of system and application events, and are useful for identifying problems, analyzing security breaches, analyzing system/application failures. (Bonus points if student mentions that in regulated or otherwise controlled environments, logs are usually mandated by industry or governmental auditors.) Logs may even, if monitored closely, provide an early warning of failures or attacks in progress. But logs only capture the level of detail

15.12 'Normal' behavior would generally involve users creating, using or deleting files belonging to either the individual user or to a group to which they belong. Normal behavior would not involve attempting to gain superuser or root privileges or in any other way altering the operating system or attempting to perform what could be considered administrator functions. In particular the rules should watch for:

- bad or repeated login attempts
- copying large numbers of files to either external media or remote locations
- attempts to access system files or log files;
- accessing directories, files or programs that are not usually accessed;
- changing security settings.
- attempts to become a superuser using the su or sudo commands.

A couple of SWATCH examples are:

watchfor /failed/ # echo bold # mail addressess=root,subject=Failed Authentication

watchfor /su:/ # echo bold # mail addresses=root,subject=Someone sued to root access

15.13 The key advantage of file integrity checking tools is that they can precisely locate potentially damaging changes in a filesystem arising

from deviations in expected system behavior, without the need to explicitly codify the kinds of events that might produce such changes.

In the context of intrusion detection, this means that an attempted attack can be flagged by its effect on the integrity of particular system files and not by any a priori knowledge of the behavioral characteristics of the attack. Such information is required to create an meaningful attack signature in signature-based intrusion detection systems, and since new attack patterns cannot be detected until they have been codified, the corpus of known signatures needs to be continually updated. Further, integrity based tools can also detect aberrant system behavior that has nothing to do with malicious activity, such as misbehaving programs and inadvertent user actions that damage files and directories.

The problem with the file-integrity tools (and indeed all anomaly-based systems) is that it is often very difficult to characterize normal system conditions with sufficient accuracy to allow intrusive or aberrant activity to be clearly distinguished. For usability, the rate of false alarms needs to be low, or system administrators will be continually responding to normal or unusual activity that is otherwise perfectly legitimate. The other problem is that integrity checking only provides evidence of malicious or aberrant activity after the fact. It does nothing to prevent it, and you still need effective recovery strategies as an adjunct to the intrusion detection process.

In general, it is desirable that executables (especially those in /bin and /usr/sbin) and critical system information resources (e.g. password files, configuration files for core services, and so on) be stringently monitored. These should not change except as a result of authorized system administrative action. Similarly, logfiles should be monitored to ensure that they are not modified or truncated. Directories, such as /root and /home should be carefully monitored for unauthorized changes, such as additions or modification to permissions. However, it is simply not sensible or feasible to monitor all files in this way, particularly those that change frequently as shared documents and user home accounts.

15.14 Unix and Linux systems use a small number of access rights for subjects being owner/group/other across nearly all resources/objects – this means it is a simple model, but because the same rights are used everywhere, their meaning can differ for different objects, and sometimes it is hard or impossible to specify some desired complex requirements.

Windows systems use a much larger set of access rights, which differ for different types of objects – a more complex model, which can be harder to master, but which may allow better specification of some desired complex requirements.

An increase in security features is desirable provided the system administrator is competent and completely understands the purpose and use of each of the security features, how they interact with each other, and which feature is best used in specific circumstances.

If the administrator was not as familiar with all the complex security features then there would be a benefit in having less security features available. This is because having less to understand means less chance of making mistakes.

CHAPTER 16 CLOUD AND IOT OPERATING SYSTEMS

ANSWERS TO QUESTIONS

16.1 **Cloud computing** is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

16.2 **Software as a service (SaaS)**: Provides service to customers in the form of software, specifically application software, running on and accessible in the cloud.
Platform as a service (PaaS): Provides service to customers in the form of a platform on which the customer's applications can run.
Infrastructure as a service (IaaS): Provides the customer access to the underlying cloud infrastructure.

16.3 The NIST cloud computing reference architecture focuses on the requirements of "what" cloud services provide, not a "how to" design solution and implementation. The reference architecture is intended to facilitate the understanding of the operational intricacies in cloud computing. It does not represent the system architecture of a specific cloud computing system; instead it is a tool for describing, discussing, and developing a system-specific architecture using a common framework of reference.

16.4 •Virtual computing: controls virtual machines within the IaaS cloud computing environment.
•Virtual storage: provides data storage services for the cloud infrastructure.
•Virtual network: provides networking services for the cloud infrastructure.
•Database and object storage: provides block, file, and object access to stored data.
•Management and orchestration: controls the IaaS environment.

16.5 In essence, the cloud OS is the software that implements IaaS.

16.6 OpenStack is an open source software project of the OpenStack Foundation that aims to produce an open source cloud operating system.

16.7 The Internet of things (IoT) is a term that refers to the expanding interconnection of smart devices, ranging from appliances to tiny sensors.

16.8 •Sensor: A sensor measures some parameter of a physical, chemical, or biological entity and delivers an electronic signal proportional to the observed characteristic, either in the form of an analog voltage level or a digital signal. In both cases, the sensor output is typically input to a microcontroller or other management element.
•Actuator: An actuator receives an electronic signal from a controller and responds by interacting with its environment to produce an effect on some parameter of a physical, chemical, or biological entity.
•Microcontroller: The "smart" in a smart device is provided by a deeply embedded microcontroller.
•Transceiver: A transceiver contains the electronics needed to transmit and receive data. Most IoT devices contain a wireless transceiver, capable of communication using Wi-Fi, ZigBee, or some other wireless scheme.
•Radio-Frequency Identification (RFID): (RFID) technology, which uses radio waves to identify items, is increasingly becoming an enabling technology for IoT. The main elements of an RFID system are tags and readers. RFID tags are small programmable devices used for object, animal, and human tracking. They come in a variety of shapes, sizes, functionalities, and costs. RFID readers acquire and sometimes rewrite information stored on RFID tags that come within operating range (a few inches up to several feet). Readers are usually connected to a computer system that records and formats the acquired information for further uses.

16.9 Small memory footprint, support for heterogeneous hardware, network connectivity, energy efficiency, real-time capabilities, security.

16.10 RIOT is an open-source OS designed specifically for constrained IoT devices.

CHAPTER 17 NETWORK PROTOCOLS

ANSWERS TO QUESTIONS

17.1 The network access layer is concerned with the exchange of data between a computer and the network to which it is attached.

17.2 The transport layer is concerned with data reliability and correct sequencing.

17.3 A protocol is the set of rules or conventions governing the way in which two entities cooperate to exchange data.

17.4 The software structure that implements the communications function. Typically, the protocol architecture consists of a layered set of protocols, with one or more protocols at each layer.

17.5 Transmission Control Protocol/Internet Protocol (TCP/IP) are two protocols originally designed to provide low level support for internetworking. The term is also used generically to refer to a more comprehensive collection of protocols developed by the U.S. Department of Defense and the Internet community.

17.6 A sockets interface is an API that enables programs to be writing that make use of the TCP/IP protocol suite to establish communication between a client and server.

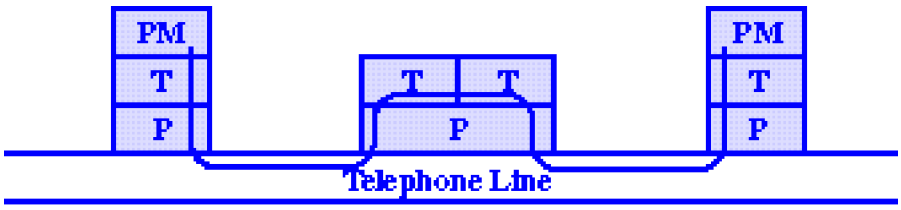
ANSWERS TO PROBLEMS

17.1 a.

The PMs speak as if they are speaking directly to each other. For example, when the French PM speaks, he addresses his remarks directly to the Chinese PM. However, the message is actually passed through two translators via the phone system. The French PM's

translator translates his remarks into English and telephones these to the Chinese PM's translator, who translates these remarks into Chinese.

b.



An intermediate node serves to translate the message before passing it on.

17.2 Perhaps the major disadvantage is the processing and data overhead. There is processing overhead because as many as seven modules (OSI model) are invoked to move data from the application through the communications software. There is data overhead because of the appending of multiple headers to the data. Another possible disadvantage is that there must be at least one protocol standard per layer. With so many layers, it takes a long time to develop and promulgate the standards.

17.3 Data plus transport header plus internet header equals 1820 bits. This data is delivered in a sequence of packets, each of which contains 24 bits of network header and up to 776 bits of higher-layer headers and/or data. Three network packets are needed. Total bits delivered = $1820 + 3 \times 24 = 1892$ bits.

17.4 UDP has a fixed-sized header. The header in TCP is of variable length.

17.5 Suppose that A sends a data packet k to B and the ACK from B is delayed but not lost. A resends packet k, which B acknowledges. Eventually A receives 2 ACKs to packet k, each of which triggers transmission of packet (k + 1). B will ACK both copies of packet (k + 1), causing A to send two copies of packet (k + 2). From now on, 2 copies of every data packet and ACK will be sent.

17.6 TFTP can transfer a maximum of 512 bytes per round trip (data sent, ACK received). The maximum throughput is therefore 512 bytes divided by the round-trip time.

17.7 The "netascii" transfer mode implies the file data are transmitted as lines of ASCII text terminated by the character sequence {CR, LF}, and that both systems must convert between this format and the one they use to store the text files locally. This means that when the "netascii" transfer mode is employed, the file sizes of the local and the

remote file may differ, without any implication of errors in the data transfer. For example, UNIX systems terminate lines by means of a single LF character, while other systems, such as Microsoft Windows, terminate lines by means of the character sequence {CR, LF}. This means that a given text file will usually occupy more space in a Windows host than in a UNIX system.

17.8 If the same TIDs are used in twice in immediate succession, there's a chance that packets of the first instance of the connection that were delayed in the network arrive during the life of the second instance of the connection, and, as they would have the correct TIDs, they could be (mistakenly) considered as valid.

17.9 TFTP needs to keep a copy of only the last packet it has sent, since the acknowledgement mechanism it implements guarantees that all the previous packets have been received, and thus will not need to be retransmitted.

17.10 This could trigger an "error storm". Suppose host A receives an error packet from host B, and responds it by sending an error packet back to host B. This packet could trigger another error packet from host B, which would (again) trigger an error packet at host A. Thus, error messages would bounce from one host to the other, indefinitely, congesting the network and consuming the resources of the participating systems.

17.11 The disadvantage is that using a fixed value for the retransmission timer means the timer will not reflect the characteristics of the network on which the data transfer is taking place. For example, if both hosts are on the same local area network, a 5-second timeout is more than enough. On the other hand, if the transfer is taking place over a (long delay) satellite link, then a 5-second timeout might be too short, and could trigger unnecessary retransmissions. On the other hand, using a fixed value for the retransmission timer keeps the TFTP implementation simple, which is the objective the designers of TFTP had in mind.

17.12 TFTP does not implement any error detection mechanism for the transmitted data. Thus, reliability depends on the service provided by the underlying transport protocol (UDP). While the UDP includes a checksum for detecting errors, its use is optional. Therefore, if UDP checksums are not enabled, data could be corrupted without being detected by the destination host.

CHAPTER 18 DISTRIBUTED PROCESSING, CLIENT/SERVER, AND CLUSTERS

ANSWERS TO QUESTIONS

18.1 A networked environment that includes client machines that make requests of server machines.

18.2 **1.** There is a heavy reliance on bringing user-friendly applications to the user on his or her own system. This gives the user a great deal of control over the timing and style of computer usage and gives department-level managers the ability to be responsive to their local needs. **2.** Although applications are dispersed, there is an emphasis on centralizing corporate databases and many network management and utility functions. This enables corporate management to maintain overall control of the total capital investment in computing and information systems, and to provide interoperability so that systems are tied together. At the same time it relieves individual departments and divisions of much of the overhead of maintaining sophisticated computer-based facilities, but enables them to choose just about any type of machine and interface they need to access data and information. **3.** There is a commitment, both by user organizations and vendors, to open and modular systems. This means that the user has greater choice in selecting products and in mixing equipment from a number of vendors. **4.** Networking is fundamental to the operation. Thus, network management and network security have a high priority in organizing and operating information systems.

18.3 It is the communications software that enables client and server to interoperate.

18.4 **Server-based processing:** The rationale behind such configurations is that the user workstation is best suited to providing a user-friendly interface and that databases and applications can easily be maintained on central systems. Although the user gains the advantage of a better interface, this type of configuration does not generally lend itself to any significant gains in productivity or to any fundamental changes in the actual business functions that the system supports. **Client-based processing:** This architecture is enables the user to employ applications tailored to local needs. **Cooperative processing:** This

type of configuration may offer greater user productivity gains and greater network efficiency than other client/server approaches.

18.5 **Fat client:** Client-based processing, with most of the software at the client. The main benefit of the fat client model is that it takes advantage of desktop power, offloading application processing from servers and making them more efficient and less likely to be bottlenecks. **Thin client:** Server-based processing, with most of the software at the server. This approach more nearly mimics the traditional host-centered approach and is often the migration path for evolving corporate wide applications from the mainframe to a distributed environment.

18.6 **Fat client:** The main benefit is that it takes advantage of desktop power, offloading application processing from servers and making them more efficient and less likely to be bottlenecks. The addition of more functions rapidly overloads the capacity of desktop machines, forcing companies to upgrade. If the model extends beyond the department to incorporate many users, the company must install high-capacity LANs to support the large volumes of transmission between the thin servers and the fat clients. Finally, it is difficult to maintain, upgrade, or replace applications distributed across tens or hundreds of desktops. **Thin client:** This approach more nearly mimics the traditional host-centered approach and is often the migration path for evolving corporate wide applications from the mainframe to a distributed environment. It does not provide the flexibility of the fat client approach.

18.7 The middle tier machines are essentially gateways between the thin user clients and a variety of backend database servers. The middle tier machines can convert protocols and map from one type of database query to another. In addition, the middle tier machine can merge/integrate results from different data sources. Finally, the middle tier machine can serve as a gateway between the desktop applications and the backend legacy applications by mediating between the two worlds.

18.8 Middleware is a set of standard programming interfaces and protocols that sit between the application above and communications software and operating system below. It provides a uniform means and style of access to system resources across all platforms

18.9 TCP/IP does not provide the APIs and the intermediate-level protocols to support a variety of applications across different hardware and OS platforms.

18.10 **Nonblocking primitives** provide for efficient, flexible use of the message-passing facility by processes. The disadvantage of this approach is that it is difficult to test and debug programs that use these primitives. Irreproducible, timing-dependent sequences can create subtle and difficult problems. **Blocking primitives** have the opposite advantages and disadvantages.

18.11 **Nonpersistent binding**: Because a connection requires the maintenance of state information on both ends, it consumes resources. The nonpersistent style is used to conserve those resources. On the other hand, the overhead involved in establishing connections makes nonpersistent binding inappropriate for remote procedures that are called frequently by the same caller. P**ersistent binding**: For applications that make many repeated calls to remote procedures, persistent binding maintains the logical connection and allows a sequence of calls and returns to use the same connection.

18.12 The **synchronous RPC** is easy to understand and program because its behavior is predictable. However, it fails to exploit fully the parallelism inherent in distributed applications. This limits the kind of interaction the distributed application can have, resulting in lower performance. To provide greater flexibility, **asynchronous RPC** facilities achieve a greater degree of parallelism while retaining the familiarity and simplicity of the RPC. Asynchronous RPCs do not block the caller; the replies can be received as and when they are needed, thus allowing client execution to proceed locally in parallel with the server invocation.

18.13 **Passive Standby:** A secondary server takes over in case of primary server failure. **Separate Servers:** Separate servers have their own disks. Data is continuously copied from primary to secondary server. **Servers Connected to Disks:** Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server. **Servers Share Disks:** Multiple servers simultaneously share access to disks.

ANSWERS TO PROBLEMS

18.1 **a.** MIPS rate = $[n\alpha + (1 - \alpha)]x = (n\alpha - \alpha + 1)x$
b. $\alpha = 0.6$

18.2 **a.** One computer executes for a time T. Eight computers execute for a time T/4, which would take a time 2T on a single computer. Thus the total required time on a single computer is 3T. Effective speedup = 3. $\alpha = 0.75$.

b. New speedup = 3.43
c. α must be improved to 0.8.

18.3 **a.** Sequential execution time = 1,051,628 cycles
b. Speedup = 16.28
c. Each computer is assigned 32 iterations balanced between the beginning and end of the I-loop.
d. The ideal speedup of 32 is achieved.

CHAPTER 19 DISTRIBUTED PROCESS MANAGEMENT

ANSWERS TO QUESTIONS

19.1 **Load sharing:** By moving processes from heavily loaded to lightly loaded systems, the load can be balanced to improve overall performance. **Communications performance:** Processes that interact intensively can be moved to the same node to reduce communications cost for the duration of their interaction. Also, when a process is performing data analysis on some file or set of files larger than the process's size, it may be advantageous to move the process to the data rather than vice versa. **Availability:** Long-running processes may need to move to survive in the face of faults for which advance notice can be achieved or in advance of scheduled downtime. If the operating system provides such notification, a process that wants to continue can either migrate to another system or ensure that it can be restarted on the current system at some later time. **Utilizing special capabilities:** A process can move to take advantage of unique hardware or software capabilities on a particular node.

19.2 The following alternative strategies may be used. **Eager (all):** Transfer the entire address space at the time of migration. **Precopy:** The process continues to execute on the source node while the address space is copied to the target node. Pages modified on the source during the precopy operation have to be copied a second time. **Eager (dirty):** Transfer only those pages of the address space that are in main memory and have been modified. Any additional blocks of the virtual address space will be transferred on demand only. **Copy-on-reference:** This is a variation of eager (dirty) in which pages are only brought over when referenced. **Flushing:** The pages of the process are cleared from the main memory of the source by flushing dirty pages to disk. Then pages are accessed as needed from disk instead of from memory on the source node.

19.3 Nonpreemptive process migration can be useful in load balancing. It has the advantage that it avoids the overhead of full-blown process migration. The disadvantage is that such a scheme does not react well to sudden changes in load distribution.

19.4 Because of the delay in communication among systems, it is impossible to maintain a system wide clock that is instantly available to all systems. Furthermore, it is also technically impractical to maintain one central clock and to keep all local clocks synchronized precisely to that central clock; over a period of time, there will be some drift among the various local clocks that will cause a loss of synchronization.

19.5 In a fully **centralized algorithm**, one node is designated as the control node and controls access to all shared objects. When any process requires access to a critical resource, it issues a Request to its local resource-controlling process. This process, in turn, sends a Request message to the control node, which returns a Reply (permission) message when the shared object becomes available. When a process has finished with a resource, a Release message is sent to the control node. In a distributed algorithm, the mutual exclusion algorithm involves the concurrent cooperation of distributed peer entities.

19.6 Deadlock in resource allocation, deadlock in message communication.

ANSWERS TO PROBLEMS

19.1 **a.** Eager (dirty)
b. Copy on reference

19.2 Process P_1 begins with a clock value of 0. To transmit message *a*, it increments its clock by 1 and transmits (*a*, 1, 1), where the first numerical value is the timestamp and the second is the identity of the site. Similarly, P_4 increments its clock by 1 and transmits issues (*q*, 1, 4). Both messages are received by the other three sites. Both *a* and *q* have the same timestamp, but P_1's numerical identifier is less than P_4's numerical identifier ($1 < 4$). Therefore, the ordering is {*a*, *q*} at all four sites.

19.3 P_i can save itself the transmission of a Reply message to P_j if P_i has sent a Request message but has not yet received the corresponding Release message.

19.4 **a.** If a site *i*, which has asked to enter its critical section, has received a response from all the others, then (1) its request is the oldest (in the sense defined by the timestamp ordering) of all the requests that may be waiting; and (2) all critical sections requested earlier

have been completed. If a site *j* has itself sent an earlier request, or if it was in its critical section, it would not have sent a response to *i*.

b. As incoming requests are totally ordered, they are served in that order; every request will at some stage become the oldest, and will then be served.

19.5 The algorithm makes no allowance for resetting the time stamping clocks with respect to each other. For a given process, P_i for example, *clock* is only used to update, on the one hand, *request* [*i*] variables in the other processes by way of request messages, and, on the other hand, *token* [*i*] variables, when messages of the token type are transferred. So the clocks are not used to impose a total ordering on requests. They are used simply as counters that record the number of times the various processes have asked to use the critical section, and so to find whether or not the number of times that P*i* has been given this access, recorded as the value of *token* [*i*], is less than the number of requests it has made, known to P_j by the value of $request_j$ [*i*]. The function *max* used in the processing associated with the reception of requests results in only the last request from P_j being considered if several had been delivered out of sequence.

19.6 **a.** Mutual exclusion is guaranteed if at any one time the number of variables *token_present* that have the value true cannot exceed 1. Since this is the initial condition, it suffices to show that the condition is conserved throughout the procedure. Consider first the prelude. The variable for P_i, which we write $token\_present_i$, changes its value from false to true when P_i receives the token. If we now consider the postlude for process P_j that has issued the token, we see that P_j has been able to do so only if $token\_present_j$ had the value true and P_j had changed this to false before sending the token.

b. Suppose that all the processes wish to enter the critical section but none of them has the token, so they are all halted, awaiting its arrival. The token is therefore in transit. It will after some finite time arrive at one of the processes, and so unblock it.

c. Fairness follows from the fact that all messages are delivered within a finite time of issue. The postlude requires that P_i transfers the token to the first process P_j, found in scanning the set in the order j = i+1, i+2, . . ., n, 1, . . ., i-1, whose request has reached Pi; if the transmission delays for all messages are finite (i.e., no message is lost), all the processes will learn of the wish for some P_j to enter the critical section and will agree to this when its turn comes.

19.7 The receipt of a request from P_j has the effect of updating the local variable, `request (j)`, which records the time of P_j's last request. The max operator assures that the correct order is maintained.

CHAPTER 20 OVERVIEW OF PROBABILITY AND STOCHASTIC PROCESSES

ANSWERS TO PROBLEMS

20.1 You should always switch to the other box. The box you initially chose had a 1/3 probability of containing the banknote. The other two, *taken together*, have a 2/3 probability. But once I open the one that is empty, the remaining box that you did not choose *now* has a 2/3 probability all by itself. Therefore by switching you raise your chances from 1/3 to 2/3.

Don't feel bad if you got this wrong. Because there are only two boxes to choose from at the end, there is a strong intuition to feel that the odds are 50-50. Marilyn Vos Savant published the correct solution in *Parade Magazine*, and subsequently (December 2, 1990) published letters from several distinguished mathematicians blasting her (they were wrong, not her). A cognitive scientist at MIT presented this problem to a number of Nobel physicists and they systematically gave the wrong answer (Piattelli-Palmarini, M. "Probability: Neither Rational nor Capricious." *Bostonia*, March 1991).

20.2 The same *Bostonia* article referenced above reports on an experiment with this problem. Most subjects gave the answer 87%. The vast majority, including many physicians, gave a number above 50%. We need Bayes' Theorem to get the correct answer:

$$\Pr[\text{Disease}/\text{positive}] = \frac{\Pr[\text{positive}/\text{Disease}]\Pr[\text{Disease}]}{\Pr[\text{positive}/\text{Disease}]\Pr[\text{Disease}] + \Pr[\text{positive}/\text{Well}]\Pr[\text{Well}]}$$

$$= \frac{(0.87)(0.01)}{(0.87)(0.01) + (0.13)(0.99)} = 0.063$$

Many physicians who guessed wrong lamented, "If you are right, there is no point in making clinical tests!"

20.3 Let WB equal the event {witness reports Blue cab}. Then:

$$\Pr[\text{Blue}/\text{WB}] = \frac{\Pr[\text{WB}/\text{Blue}]\Pr[\text{Blue}]}{\Pr[\text{WB}/\text{Blue}]\Pr[\text{Blue}] + \Pr[\text{WB}/\text{Green}]\Pr[\text{Green}]}$$

$$= \frac{(0.8)(0.15)}{(0.8)(0.15) + (0.2)(0.85)} = 0.41$$

This example, or something similar, is referred to as "the juror's fallacy."

20.4 a. If $K > 365$, then it is impossible for all values to be different. So, we assume $K \leq 365$. Now, consider the number of different ways, N, that we can have K values with no duplicates. We may choose any of the 365 values for the first item, any of the remaining 364 numbers for the second item, and so on. Hence, the number of different ways is:

$$N = 365 \times 364 \times \ldots \times (365 - K + 1) = \frac{365!}{(365 - K)!}$$

If we remove the restriction that there are no duplicates, then each item can be any of 365 values, and the total number of possibilities is 365^K. So the probability of no duplicates is simply the fraction of sets of values that have no duplicates out of all possible sets of values:

$$Q(K) = \frac{365!/(365 - K)!}{365^K} = \frac{365!}{(365 - K)! \times 365^K}$$

b. $P(K) = 1 - Q(K) = 1 - \dfrac{365!}{(365 - K)! \times 365^K}$

Many people would guess that to have a probability greater than 0.5 that there is at least one duplicate, the number of people in the group would have to be about 100. In fact, the number is 23, with P(23) = 0.5073. For K = 100, the probability of at least one duplicate is 0.9999997.

20.5 a. The sample space consists of the 36 equally probable ordered pairs of integers from 1 to 6. The distribution of X is given in the following table.

x_i	1	2	3	4	5	6
p_i	1/36	3/36	5/36	7/36	9/36	11/36

For example, there are 3 pairs whose maximum is 2: (1, 2), (2, 2), (2, 1).

b. $= \mathrm{E}[X] = 1\left(\frac{1}{36}\right) + 2\left(\frac{3}{36}\right) + 3\left(\frac{5}{36}\right) + 4\left(\frac{7}{36}\right) + 5\left(\frac{9}{36}\right) + 6\left(\frac{11}{36}\right) = \frac{161}{36} \approx 4.5$

$$\mathrm{E}\left[X^2\right] = 1\left(\frac{1}{36}\right) + 4\left(\frac{3}{36}\right) + 9\left(\frac{5}{36}\right) + 16\left(\frac{7}{36}\right) + 25\left(\frac{9}{36}\right) + 36\left(\frac{11}{36}\right) = \frac{791}{36} \approx 22.0$$

Var[X] = E[X^2] – μ^2 = 1.75 $\qquad \sigma_X = \sqrt{1.75} = 1.3$

20.6 a.

x_i	–1	2	3	–4	5	-6
p_i	1/6	1/6	1/6	1/6	1/6	1/6

b. E[X] = –1/6. On average the player loses, so the game is unfair

20.7 a.

x_i	–E	E	2E	3E
p_i	125/216	75/216	15/216	1/216

b. E[X] = –17E/216 = –0.8E. Here's another game to avoid.

20.8 a. E[X^2] = Var[X] + μ^2 = 2504
b. Var[2X + 3] = 4Var[X] + Var[3] = 16; standard deviation = 4
c. Var[–X] = (–1)^{2}Var[X] = 4; standard deviation = 2

20.9 The probability density function f(r) must be a constant in the interval (900, 1100) and its integral must equal 1; therefore f(r) = 1/200 in the interval (900, 1100). Then

$$\Pr[950 \le r \le 1050] = \frac{1}{200}\int_{950}^{1050} dr = 0.5$$

20.10 Let Z = X + Y

Var[Z] = E[(Z – μ_Z)^2] = E[(X + Y – μ_X – μ_Y)^2]
= E[(X – μ_X)^2] + E[(Y – μ_Y)^2] + 2E[(X – μ_X)(Y – μ_Y)]
= Var[X] + Var[Y] + 2r(X, Y)$\sigma_X\sigma_Y$

The greater the correlation coefficient, the greater the variance of the sum. A similar derivation shows that the greater the correlation coefficient, the smaller the variance of the difference.

20.11 E[X] = Pr[X = 1]; E[Y] = Pr[Y = 1]; E[XY] = Pr[X = 1, Y = 1]

If X and Y are uncorrelated, then r(X, Y) = Cov[X, Y] = 0. By the definition of covariance, we therefore have E[XY] = E[X]E[Y]. Therefore

Pr[X = 1, Y = 1] = Pr[X = 1]Pr[Y = 1]

By similar reasoning, you can show that the other three joint probabilities are also products of the corresponding marginal (single variable) probabilities. This proves that P(x, y) = P(x)P(y) for all (x, y), which is the definition of independence.

20.12 a. No. X and Y are dependent because the value of X determines the value of Y.

b. E[XY] = (–1)(1)(0.25) + (0)(0)(0.5) + (1)(1)(0.25) = 0
E[X] = (–1)(0.25) + (0)(0.5) + (1)(0.25) = 0
E[Y] = (1)(0.25) + (0)(0.5) + (1)(0.25) = 0.5
Cov(X, Y) = E[XY] – E[X]E[Y] = 0

c. X and Y are uncorrelated because Cov(X, Y) = 0

20.13 $\mu(t) = E[g(t)] = g(t)$; $Var[g(t)] = 0$; $R(t_1, t_2) = g(t_1)g(t_2)$

20.14 E[Z] = μ(5) = 3; E[W] = μ(8) = 3
$E[Z^2] = R(5, 5) = 13$; $E[W^2] = R(8, 8) = 13$
E[ZW] = R(5, 8) = 9 + 4 exp(–0.6) = 11.195
$Var[Z] = E[Z^2] - (E[Z])^2 = 4$; Var[W] = 4
Cov(Z, W) = E[ZW] – E[Z]E[W] = 2.195

20.15 We have $E[Z_i] = 0 \qquad Var[Z_i] = E[Z_i^2] = 1 \qquad E[Z_i Z_j] = 0 \quad \text{for } i \neq j$

The autocovariance: $C(t_m, t_{m+n}) = R(t_m, t_{m+n}) - E[Z_m]E[Z_{m+n}] = E[Z_m Z_{m+n}]$

$$E[Z_m Z_{m+n}] = E\left[\left(\sum_{i=0}^{K} \alpha_i Z_{m-i}\right)\left(\sum_{j=0}^{K} \alpha_j Z_{m+n-j}\right)\right]$$

Because $E[Z_i Z_j] = 0$, only the terms with (m – i) = (m + n – j) are nonzero. Using $E[Z_i^2] = 1$, we have

$$C[t_m t_{m+n}] = \sum_{i=0}^{K} \alpha_i \alpha_{n+i} \qquad \text{with the convention that } \alpha_j = 0 \text{ for } j > K$$

The covariance does not depend on m and so the sequence is stationary.

20.16 $E[A] = E[B] = 0; \quad E[A^2] = E[B^2] = 1; \quad E[AB] = 0; \quad E[X_n] = 0$

$$C(t_m, t_{m+n}) = R(t_m, t_{m+n}) - E[X_m]E[X_{m+n}] = E[X_m X_{m+n}]$$
$$= E[(A\cos(m\lambda) + B\sin(m\lambda))(A\cos((m+n)\lambda) + B\sin((m+n)\lambda))]$$
$$= \cos(m\lambda)\cos((m+n)\lambda) + \sin(m\lambda)\sin((m+n)\lambda) = \cos(n\lambda)$$

The final step uses the identities

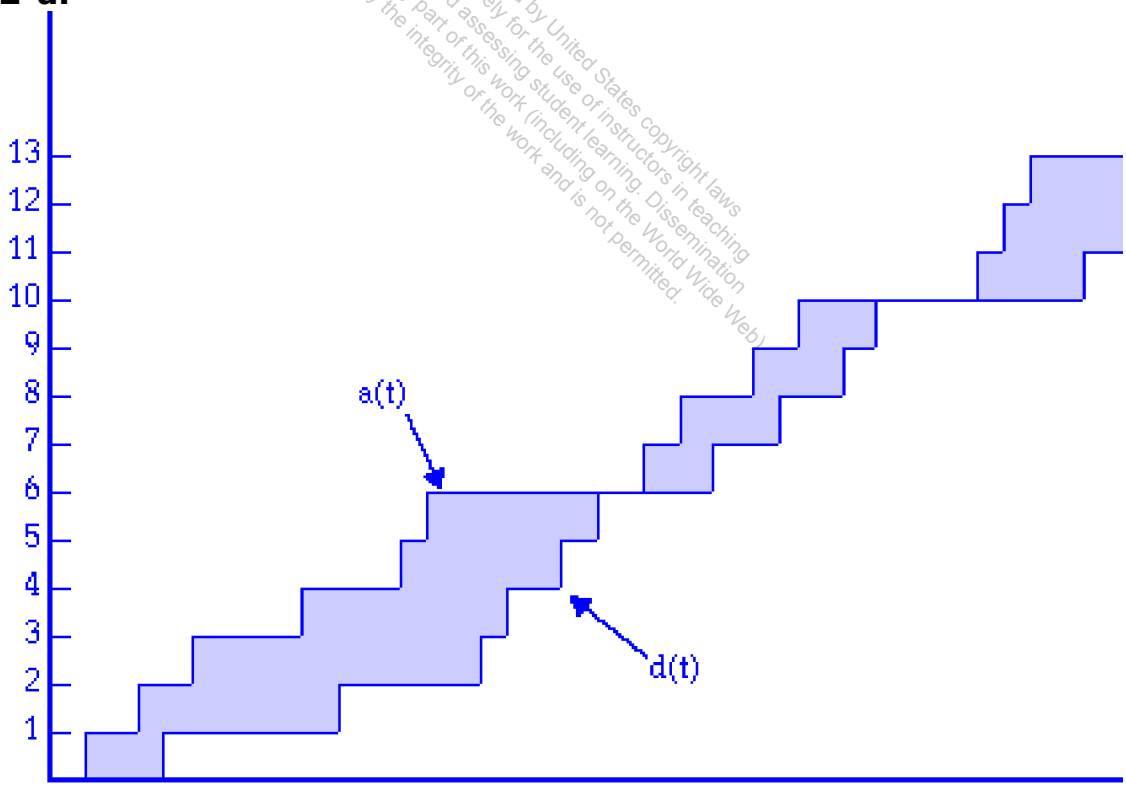
$$\sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta;$$
$$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$

CHAPTER 21 QUEUEING ANALYSIS

ANSWERS TO PROBLEMS

21.1 Consider the experience of a single item arriving. When the item arrives, it will find on average r items already in the queue or being served. When the item completes service and leaves the system, it will leave behind on average the same number of items in the queue or being served, namely r. This is in accord with saying that the average number in the system is r. Further, the average time that the item was in the system is T_r. Since items arrive at a rate of λ, we can reason that in the time T_r, a total of λT_r items must have arrived. Thus $r = \lambda T_r$.

21.2 a.



The height of the shaded portion equals n(t).

$$\int_0^{\tau} [a(t) - d(t)]dt = \sum_{k=1}^{a(\tau)} t_k$$

b. Select a time period that begins and ends with the system empty. Without loss of generality, set the beginning of the period as t = 0 and the end as t = τ. Then a(0) = d(0) = 0 and a(τ) = d(τ). The shaded area can be computed in two different ways, which must yield the same result:

$$\int_0^{\tau} \big[a(t) - d(t)\big]dt = \sum_{k=1}^{a(\tau)} t_k$$

If we view the shaded area as a sequence (from left to right) of vertical rectangles, we get the integral on the left. We can also view the shaded area as a sequence (from bottom to top) of horizontal rectangles with unit height and width t_k, where t_k is the time that the kth item remains in the system; then we get the summation on the right. Therefore:

$$\int_0^{\tau} \big[n(t)\big]dt = a(\tau)T_r$$

$$\tau r = \lambda \tau T_r$$

$$r = \lambda T_r$$

21.3 $T_q = q/\lambda = 8/18 = 0.44$ hours

21.4 No. $12.356 \neq 25.6 \times 20.34$

21.5 If there are N servers, each server expects $\lambda T/N$ customers during time T. The expected time for that server to service all these customers is $\lambda T T_S/N$. Dividing by the length of the period T, we are left with a utilization of $\lambda T_S/N$.

21.6 $\rho = \lambda T_S = 2 \times 0.25 = 0.5$
Average number in system = $r = \rho/(1 - \rho) = 1$
Average number in service = $r - w = \rho = 0.5$

21.7 We need to use the solution of the quadratic equation, which says that for

$ax^2 + bx + c = 0$, we have $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$w = \rho^2/(1 - \rho) = 4$

$\rho^2 + 4\rho - 4 = 0$

$\rho = -2 + \sqrt{8} = 0.828$

21.8 $\rho = \lambda T_s = 0.2 \times 2 = 0.4$

$T_w = \rho T_s/(1 - \rho) = 0.8/0.6 = 4/3$ minutes = 80 seconds

$T_r = T_w + T_s = 200$ seconds

$$m_{T_w}(90) = \frac{T_w}{} \ln\left(\frac{100}{100-90}\right) = \frac{1.333}{0.4}\ln(4) = 4.62 \text{ minutes}$$

21.9 T_s = (1000 octets $\times$ 8 bits/octet)/9600 bps = 0.833 secs. $\rho = 0.7$

Constant-length message: $T_w = \rho T_s/(2 - 2\rho) = 0.972$ secs.

Exponentially-distributed: $T_w = \rho T_s/(1 - \rho) = 1.944$ secs.

21.10 $T_s = (1)(0.7) + (3)(0.2) + 10(0.1) = 2.3$ ms

Define S as the random variable that represents service time. Then

$$\sigma_{T_s}^2 = \text{Var}[S] = E\left[\left(S - T_s\right)^2\right] = (1-2.3)^2(0.7) + (3-2.3)^2(0.2) + (10-2.3)^2(0.1) = 7.21 \text{ ms}$$

Using the equations in Figure 21.6a:

$A = 0.5\ (1 + (7.21/5.29)) = 1.18$

a. $\rho = \lambda T_s = 0.33 \times 2.3 = 0.767$

$T_r = T_s + (\rho T_s A)/(1 - \rho) = 2.3 + (0.767 \times 2.3 \times 1.18)/(0.233) =$ 11.23 ms

$r = \rho + (\rho^2 A)/(1 - \rho) = 3.73$ messages

b. $\rho = \lambda T_s = 0.25 \times 2.3 = 0.575$

$T_r = 2.3 + (0.575 \times 2.3 \times 1.18)/(0.425) = 5.97$ ms

$r = 0.575 + (0.575 \times 0.575 \times 1.18)/(0.425) = 1.49$ messages

c. $\rho = \lambda T_s = 0.2 \times 2.3 = 0.46$

$T_r = 2.3 + (0.46 \times 2.3 \times 1.18)/(0.54) = 4.61$ ms

$r = 0.46 + (0.46 \times 0.46 \times 1.18)/(0.54) = 0.92$ messages

21.11 $\lambda = 0.05$ msg/sec; $T_s = (14{,}400 \times 8)/9600 = 12$ sec; $\rho = 0.6$

a. $T_w = \rho T_s/(1 - \rho) = 18$ sec

b. $w = \rho^2/(1 - p) = 0.9$

21.12 **a.** mean batch service time = $T_{sb} = MT_s$; batch variance = $\sigma_{T_{sb}}^2 = M\sigma_{T_s}^2$

To get the waiting time, T_{wb}, use the equation for T_w from Table 21.6a:

$$T_w = \frac{\rho T_s A}{1-\rho} = \frac{\lambda\left(T_s^2 + \sigma_{T_s}^2\right)}{2(1-\rho)}; \quad \text{then,}$$

$$T_{wb} = \frac{\frac{\lambda}{M}\left(M^2 T_s^2 + M\sigma_{T_s}^2\right)}{2(1-\rho)} = \frac{\lambda\left(MT_s^2 + \sigma_{T_s}^2\right)}{2(1-\rho)}, \quad \text{where } \rho = \frac{\lambda}{M}(MT_s) = \lambda T_s$$

b. $T_1 = \frac{1}{M} \times 0 + \frac{1}{M} \times T_S + \frac{1}{M} \times 2T_S + \ldots + \frac{1}{M} \times (M-1)T_S = \frac{M-1}{2} T_S$

$$T_w = T_{wb} + T_1 = \frac{\lambda\left(MT_s^2 + \sigma_{T_s}^2\right)}{2(1-\rho)} + \frac{M-1}{2} T_s$$

c. The equation for T_w in (b) reduces to the equation for T_w in (a) for M = 1. For batch sizes of M > 1, the waiting time is greater than the waiting time for Poisson input. This is due largely to the second term, caused by waiting within the batch.

21.13 **a.** $\rho = \lambda T_s = 0.2 \times 4 = 0.8$; $r = 2.4$; $\sigma_r = 2.4$

b. $T_r = 12$; $T_q = 9.24$

21.14 $T_s = T_{s1} = T_{s2} = 1$ ms

a. If we assume that the two types are independent, then the combined arrival rate is Poisson.
$\lambda = 800$; $\rho = 0.8$; $T_r = 5$ ms

b. $\lambda_1 = 200$; $\lambda_2 = 600$; $\rho_1 = 0.2$ $\rho_2 = 0.6$; $T_{r1} = 2$ ms; $T_{r2} = 6$ ms

c. $\lambda_1 = 400$; $\lambda_2 = 400$; $\rho_1 = 0.4$ $\rho_2 = 0.4$; $T_{r1} = 2.33$ ms; $T_{r2} = 7.65$ ms

d. $\lambda_1 = 600$; $\lambda_2 = 200$; $\rho_1 = 0.6$ $\rho_2 = 0.2$; $T_{r1} = 3$ ms; $T_{r2} = 11$ ms

21.15 $T_S = (100 \text{ octets} \times 8)/(9600 \text{ bps}) = 0.0833$ sec.

a. If the load is evenly distributed among the links, then the load for each link is 48/5 = 9.6 packets per second.
$\rho = 9.6 \times 0.0833 = 0.8$; $T_r = 0.0833/(1 - 0.8) = 0.42$ sec

b. $\rho = \lambda T_s/N = (48 \times 0.833)/5 = 0.8$

$$T_r = T_s + \frac{C \times T_s}{N \times (1-\rho)} = 0.13 \text{ sec}$$

21.16 **a.** The first character in a M-character packet must wait an average of M – 1 character interarrival times; the second character must wait M – 2 interarrival times, and so on to the last character,

which does not wait at all. The expected interarrival time is $1/\lambda$. The average waiting time is therefore $W_i = (M - 1)/2\lambda$.

b. The arrival rate is $K\lambda/M$, and the service time is $(M + H)/C$, giving a utilization of $\rho = K\lambda(M + H)/MC$. Using the M/D/1 formula,

$$W_o = \frac{\rho T_s}{2-2\rho} = \frac{K\lambda(M+H)^2/(C^2M)}{2-(2K\lambda(M+H)/CM)}$$

c.

$$T = W_i + W_o + T_s$$

$$= \frac{M-1}{2\lambda} + \frac{K\lambda(M+H)^2}{2MC^2 - 2K\lambda C(M+H)} + \frac{M+H}{C}$$

The plot is U-shaped.

21.17 a. System throughput: $\Lambda = \lambda + P\Lambda$; therefore $\Lambda = \lambda/(1 - P)$
Server utilization: $\rho = \Lambda T_s = \lambda T_s/(1 - P)$

$$T_q = \frac{T_s}{1-\rho} = \frac{(1-P)T_s}{1-P-\lambda T_s}$$

b. A single customer may cycle around multiple times before leaving the system. The mean number of passes J is given by:

$$J = 1(1-P) + 2(1-P)P + 3(1-P)P^2 + \ldots = (1-P)\sum_{i=1}^{\infty} iP^{i-1} = \frac{1}{1-P}$$

Because the number of passes is independent of the queuing time, the total time in the system is given by the product of the two means, JT_r.

APPENDIX A TOPICS IN CONCURRENCY

ANSWERS TO PROBLEMS

A.1 The answer is no for both questions.

A.2 **a.** Change *receipt* to an array of semaphores all initialized to 0 and use *enqueue2*, *queue2*, and *dequeue2* to pass the customer numbers.

b. Change *leave_b_chair* to an array of semaphores all initialized to 0 and use enqueue1(custnr), queue1, and dequeue1(b_cust) to release the right barber.

The figure below shows the program with both of the above modifications.


```
program   barbershop2;
var max_capacity: semaphore (:= 20);
    sofa: semaphore (:= 4);
    barber_chair, coord: semaphore (:= 3);
    mutex1, mutex2, mutex3: semaphore (:=1);
    cust_ready, payment: semaphore (:= 0);
    finished, leave_b_chair, receipt: array[1..50] of semaphore (:=0);
    count: integer;

procedure customer;
var custnr: integer;
begin
wait(max_capacity);
enter shop;
wait(mutex1);
count := count + 1;
custnr := count;
signal(mutex1);
wait(sofa);
sit on sofa;
wait(barber_chair);
get up from sofa;
signal(sofa);
sit in barber chair;
wait(mutex2);
enqueue1(custnr);
signal(cust_ready);
signal(mutex2);
wait(finished[custnr]);

procedure barber;
var b_cust: integer;
begin
repeat
wait(cust_ready);
wait(mutex2);
dequeue1(b_cust);
signal(mutex2);
wait(coord);
cut hair;
signal(coord);
signal(finished[b_cust]);
wait(leave_b_chair[custnr]);
signal(barber_chair);
forever
end;

procedure cashier;
var b_cust: integer;
begin
repeat
wait(payment);
wait(mutex3);
dequeue2(c_cust);
signal(mutex3);
wait(coord);
accept pay;
signal(coord);
signal(receipt[b_cust]);
forever
end;
```

```
signal(leave_b_chair[custnr]);
pay;
wait(mutex3);
enqueue2(custnr);
signal(payment);
signal(mutex3);
wait(receipt[custnr]);
exit shop;
signal(max_capacity)
end;

begin (*main program*)
    count := 0;
    parbegin
      customer; . . . 50 times; . . . customer;
      barber; barber; barber;
      cashier
    parend
end.
```