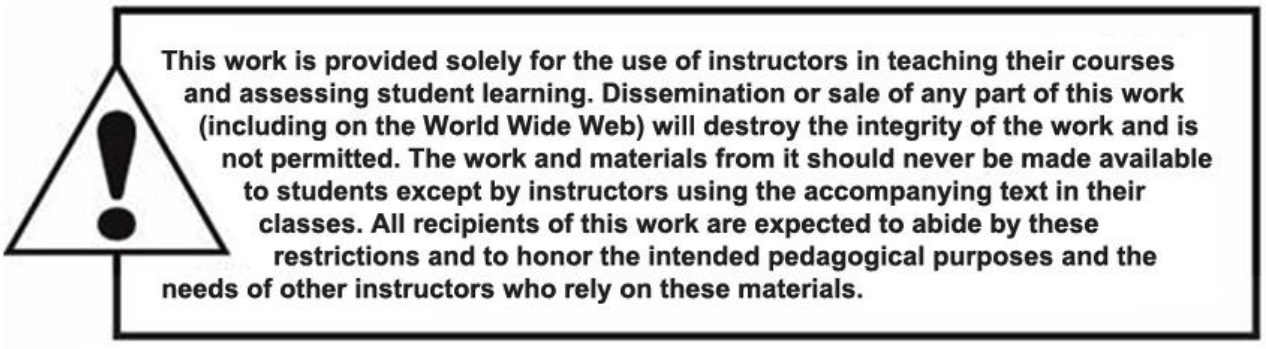
SOLUTIONS MANUAL

Operating Systems
Eighth Edition
Global Edition

Chapters 1–9

William Stallings

© Pearson Education Limited 2015



Copyright © Pearson Education Limited 2015.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

ISBN-10: 1-292-06135-9
ISBN-13: 978-1-292-06135-1

NOTICE

This manual contains solutions to the review questions and homework problems in *Operating Systems, Eighth Edition, Global 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.

W.S.

TABLE OF CONTENTS

Chapter 1 Computer System Overview 5
Chapter 2 Operating System Overview 11
Chapter 3 Process Description and Control 15
Chapter 4 Threads 23
Chapter 5 Mutual Exclusion and Synchronization 29
Chapter 6 Deadlock and Starvation 49
Chapter 7 Memory Management 65
Chapter 8 Virtual Memory 73
Chapter 9 Uniprocessor Scheduling 84

CHAPTER 1 COMPUTER SYSTEM OVERVIEW

ANSWERS TO QUESTIONS

1.1 A processor, which controls the operation of the computer and performs its data processing functions ; a **main memory**, which stores both data and instructions; **I/O modules**, which move data between the computer and its external environment; and the system bus, which provides for communication among processors, main memory, and I/O modules.

1.2 **User-visible registers:** Enable the machine- or assembly-language programmer to minimize main memory references by optimizing register use. For high-level languages, an optimizing compiler will attempt to make intelligent choices of which variables to assign to registers and which to main memory locations. Some high-level languages, such as C, allow the programmer to suggest to the compiler which variables should be held in registers. **Control and status registers:** Used by the processor to control the operation of the processor and by privileged, operating system routines to control the execution of programs.

1.3 These actions fall into four categories: **Processor-memory:** Data may be transferred from processor to memory or from memory to processor. **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module. **Data processing:** The processor may perform some arithmetic or logic operation on data. **Control:** An instruction may specify that the sequence of execution be altered.

1.4 Multiple interrupts may be serviced by assigning different priorities to interrupts arising from different sources. This enables a higher-priority interrupt to be serviced first when multiple requests arrive simultaneously; it also allows a higher-priority interrupt to preempt a lower-priority interrupt. For example, suppose a system has assigned a higher priority to a communication line and a lower priority to a magnetic disk. When two simultaneous requests arrive, the computer services the communication line. Similarly, if some disk operations are ongoing when a request for the communication line arrives, the state of the disk is put in a stack and the communication line operations are catered to.

1.5 In interrupt-driven I/O, when data is available in the peripheral, an interrupt facility and special commands inform the interface to issue an interrupt request signal. In the meantime, the CPU can continue its other activities. When the CPU detects an external signal-interrupt, it momentarily stops the task it is processing, services the I/O transfer process, and then resumes the original task.

1.6 The characteristics observed while going up the memory hierarchy are **a.** increasing cost per bit, **b.** decreasing capacity, **c.** decreasing access time, and **d.** increasing frequency of access to the memory by the processor.

1.7 The main trade-offs for determining the cache size are the speed and the cost of the cache.

1.8 A multicore computer is a special case of a multiprocessor, in which all of the processors are on a single chip.

1.9 The cache write policies are as follows:

a. Write through: Whenever a block in the cache is altered, it is immediately written to the main memory.

b. Write back: The contents of a cache block are written to the main memory only when that block is replaced from the cache.

1.10 **Spatial locality** is generally exploited by using larger cache blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic. **Temporal locality** is exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy.

ANSWERS TO PROBLEMS

1.1 Memory (contents in hex): 300: 3005; 301: 5940; 302: 7006
Step 1: 3005 → IR; **Step 2:** 3 → AC
Step 3: 5940 → IR; **Step 4:** 3 + 2 = 5 → AC
Step 5: 7006 → IR; **Step 6:** AC → Device 6

1.2 **1. a.** The PC contains 300, the address of the first instruction. This value is loaded in to the MAR.
b. The value in location 300 (which is the instruction with the value 1940 in hexadecimal) is loaded into the MBR, and the PC is incremented. These two steps can be done in parallel.
c. The value in the MBR is loaded into the IR.

2. a. The address portion of the IR (940) is loaded into the MAR.
b. The value in location 940 is loaded into the MBR.
c. The value in the MBR is loaded into the AC.

3. a. The value in the PC (301) is loaded in to the MAR.
b. The value in location 301 (which is the instruction with the value 5941) is loaded into the MBR, and the PC is incremented.
c. The value in the MBR is loaded into the IR.

4. a. The address portion of the IR (941) is loaded into the MAR.
b. The value in location 941 is loaded into the MBR.
c. The old value of the AC and the value of location MBR are added and the result is stored in the AC.

5. a. The value in the PC (302) is loaded in to the MAR.
b. The value in location 302 (which is the instruction with the value 2941) is loaded into the MBR, and the PC is incremented.
c. The value in the MBR is loaded into the IR.

6. a. The address portion of the IR (941) is loaded into the MAR.
b. The value in the AC is loaded into the MBR.
c. The value in the MBR is stored in location 941.

1.3 a. 2^{24} = 16 MBytes

b. (1) If the local address bus is 32 bits, the whole address can be transferred at once and decoded in memory. However, since the data bus is only 16 bits, it will require 2 cycles to fetch a 32-bit instruction or operand.

(2) The 16 bits of the address placed on the address bus can't access the whole memory. Thus a more complex memory interface control is needed to latch the first part of the address and then the second part (since the microprocessor will end in two steps). For a 32-bit address, one may assume the first half will decode to access a "row" in memory, while the second half is sent later to access a "column" in memory. In addition to the two-step address operation, the microprocessor will need 2 cycles to fetch the 32 bit instruction/operand.

c. The program counter must be at least 24 bits. Typically, a 32-bit microprocessor will have a 32-bit external address bus and a 32-bit program counter, unless on-chip segment registers are used that may work with a smaller program counter. If the instruction register is to contain the whole instruction, it will have to be 32-bits long; if it will contain only the op code (called the op code register) then it will have to be 8 bits long.

1.4 In cases **(a)** and **(b)**, the microprocessor will be able to access 2^{16} = 64K bytes; the only difference is that with an 8-bit memory each access will transfer a byte, while with a 16-bit memory an access may

transfer a byte or a 16-byte word. For case **(c)**, separate input and output instructions are needed, whose execution will generate separate "I/O signals" (different from the "memory signals" generated with the execution of memory-type instructions); at a minimum, one additional output pin will be required to carry this new signal. For case **(d)**, it can support 2^8 = 256 input and 2^8 = 256 output byte ports and the same number of input and output 16-bit ports; in either case, the distinction between an input and an output port is defined by the different signal that the executed input or output instruction generated.

1.5 Clock cycle = $\frac{1}{8\ \text{MHZ}}$ = 125 ns

Bus cycle = 4 × 125 ns = 500 ns

2 bytes transferred every 500 ns; thus transfer rate = 4 MBytes/sec

Doubling the frequency may mean adopting a new chip manufacturing technology (assuming each instructions will have the same number of clock cycles); doubling the external data bus means wider (maybe newer) on-chip data bus drivers/latches and modifications to the bus control logic. In the first case, the speed of the memory chips will also need to double (roughly) not to slow down the microprocessor; in the second case, the "word length" of the memory will have to double to be able to send/receive 32-bit quantities.

1.6 a. Input from the Teletype is stored in INPR. The INPR will only accept data from the Teletype when FGI=0. When data arrives, it is stored in INPR, and FGI is set to 1. The CPU periodically checks FGI. If FGI =1, the CPU transfers the contents of INPR to the AC and sets FGI to 0.

When the CPU has data to send to the Teletype, it checks FGO. If FGO = 0, the CPU must wait. If FGO = 1, the CPU transfers the contents of the AC to OUTR and sets FGO to 0. The Teletype sets FGI to 1 after the word is printed.

b. The process described in **(a)** is very wasteful. The CPU, which is much faster than the Teletype, must repeatedly check FGI and FGO. If interrupts are used, the Teletype can issue an interrupt to the CPU whenever it is ready to accept or send data. The IEN register can be set by the CPU (under programmer control)

1.7 If a processor is held up in attempting to read or write memory, usually no damage occurs except a slight loss of time. However, a DMA transfer may be to or from a device that is receiving or sending data in a stream (e.g., disk or tape), and cannot be stopped. Thus, if the DMA module is held up (denied continuing access to main memory), data will be lost.

1.8 Let us ignore data read/write operations and assume the processor only fetches instructions. Then the processor needs access to main memory

once every microsecond. The DMA module is transferring characters at a rate of 1200 characters per second, or one every 833 μs. The DMA therefore "steals" every 833rd cycle. This slows down the processor approximately $\frac{1}{833} \times 100\% = 0.12\%$

1.9 a. The processor can only devote 5% of its time to I/O. Thus the maximum I/O instruction execution rate is $10^6 \times 0.05 = 50{,}000$ instructions per second. The I/O transfer rate is therefore 25,000 words/second.

b. The number of machine cycles available for DMA control is

$10^6(0.05 \times 5 + 0.95 \times 2) = 2.15 \times 10^6$

If we assume that the DMA module can use all of these cycles, and ignore any setup or status-checking time, then this value is the maximum I/O transfer rate.

1.10 a. A reference to the first instruction is immediately followed by a reference to the second.

b. The ten accesses to `a[i]` within the inner for loop which occur within a short interval of time.

1.11 Let the three memory hierarchies be M_1, M_2, and M_3.

Let us define the following parameters:

T_s = average system access time

T_1, T_2, and T_3 = access time of M_1, M_2, and M_3 respectively.

h_1, h_2 = hit ratios of memories M_1 and M_2.

C_s = average cost per bit of combined memory.

C_1, C_2, and C_3 = cost per bit of M_1, M_2, and M_3 respectively.

Extension of Equation (1.1) to 3-level memory hierarchy:

We can say that a word is found in M_1 with a probability h_1.
So the word is not found in M_1 with a probability $(1 - h_1)$.
Or in other words, memory M_2 is accessed with a probability $(1 - h_1)$.
As the hit ratio of M_2 is h_2, the word is found in M_2 with a probability $(1 - h_1)h_2$.
So, memory M_3 is accessed with a probability $(1 - h_1)(1 - h_2)$.
If we multiply the probabilities of access with the access times and sum up, we will get the average system access time.
Hence, $T_S = h_1 . T_1 + (1 - h_1)h_2 . T_2 + (1 - h_1)(1 - h_2).T_3$

Extension of Equation (1.2) to 3-level memory hierarchy:

Average cost = Total cost/Total size of memory

$$= \frac{C_1 S_1 + C_2 S_2 + C_3 S_3}{S_1 + S_2 + S_3}$$

1.12 **a.** Cost of system without cache memory, $C_1 = 256 \times 2^{20} \times 0.0001$
$= 26843.5456$ cents
$= 268.435$ dollars

b. Cost of cache memory, $C_2 = 32 \times 2^{10} \times 0.1 = 3276.8$ cents
$= 32.768$ dollars
Cost of system having cache memory, $C_1 + C_2 = 301.203$ dollars

c. Effective access time without using cache, $T_1 = 100$ ns
Effective access time using cache, $T_2 = 0.85 \times 10 + (1 - 0.85) \times 100$
$= 23.5$ ns
Percentage decrease in time = (decrease in time/original time) $\times 100$
$= 76.5$ %

1.13 Cache access time, $t_c = 60$ ns
Main memory access time, $t_m = 300$ ns
Hit ratio, $h = 0.9$
Percentage of read operations, $p_r = 80\% = 0.8$
Percentage of write operations, $p_w = 100 - 80 = 20\% = 0.2$
Average access time of read operations $= ht_c + (1 - h)t_m$
$= 0.9 \times 60 + (1 - 0.9) \times 300$
$= 84$ ns
Average access time of write operations $= 300$ ns
Effective time, t_{eff} for read operations $= 0.8 \times 84$
$= 67.2$ ns
Effective time, t_{eff} for write operations $= 0.2 \times 300$
$= 60$ ns
Total time = t_{eff} for read + t_{eff} for write $= 67.2 + 60$
$= 127.2$ ns

1.14 Yes, if the stack is only used to hold the return address. If the stack is also used to pass parameters, then the scheme will work only if it is the control unit that removes parameters, rather than machine instructions. In the latter case, the processor would need both a parameter and the PC on top of the stack at the same time.

CHAPTER 2 OPERATING SYSTEM OVERVIEW

ANSWERS TO QUESTIONS

2.1 **Convenience:** An operating system makes a computer more convenient to use. **Efficiency:** An operating system allows the computer system resources to be used in an efficient manner. **Ability to evolve:** An operating system should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

2.2 The two modes of operation in an operating system are user mode and kernel mode. In user mode, the computer system executes on behalf of the user application so that certain areas of memory are protected from user's use and certain instructions are not executed. In kernel mode, the computer system renders certain operating system services whereby privileged instructions may be executed and protected areas of memory may be accessed.

2.3 The advantages of multiprogramming over single processing are:

- **a.** CPU utilization is high, as CPU is seldom idle.
- **b.** CPU throughput is more.
- **c.** Multiple simultaneous users are supported, who may work interactively on separate terminals.
- **d.** Memory utilization is found to increase.
- **e.** Disk usage and peripheral usage also increases.

2.4 A process is a program in execution. A process is controlled and scheduled by the operating system.

2.5 The **execution context,** or **process state,** is the internal data by which the operating system is able to supervise and control the process. This internal information is separated from the process, because the operating system has information not permitted to the process. The context includes all of the information that the operating system needs to manage the process and that the processor needs to execute the process properly. The context includes the contents of the various processor registers, such as the program counter and data registers. It also includes information of use to the operating system, such as the priority of the process and whether

the process is waiting for the completion of a particular I/O event.

2.6 **Process isolation:** The operating system must prevent independent processes from interfering with each other's memory, both data and instructions. **Automatic allocation and management:** Programs should be dynamically allocated across the memory hierarchy as required. Allocation should be transparent to the programmer. Thus, the programmer is relieved of concerns relating to memory limitations, and the operating system can achieve efficiency by assigning memory to jobs only as needed. **Support of modular programming:** Programmers should be able to define program modules, and to create, destroy, and alter the size of modules dynamically. **Protection and access control:** Sharing of memory, at any level of the memory hierarchy, creates the potential for one program to address the memory space of another. This is desirable when sharing is needed by particular applications. At other times, it threatens the integrity of programs and even of the operating system itself. The operating system must allow portions of memory to be accessible in various ways by various users. **Long-term storage:** Many application programs require means for storing information for extended periods of time, after the computer has been powered down.

2.7 Time slicing is a technique adopted in time-sharing systems to distribute CPU time to multiple users. In this technique, a system clock generates interrupts at a particular rate. At each clock interrupt, the OS regains control and can assign the processor to another user. Thus, at regular time intervals, the current user is preempted and another user is loaded in. To preserve the old user program status for later resumption, the old user programs and data are written out to disk before the new user programs and data are read in. Subsequently, the old user program code and data are restored in main memory when that program is given a turn in a subsequent time-slice.

2.8 Round robin is a scheduling algorithm in which processes are activated in a fixed cyclic order; that is, all processes are in a circular queue. A process that cannot proceed because it is waiting for some event (e.g. termination of a child process or an input/output operation) returns control to the scheduler.

2.9 A **monolithic kernel** is a large kernel containing virtually the complete operating system, including scheduling, file system, device drivers, and memory management. All the functional components of the kernel have access to all of its internal data structures and routines. Typically, a monolithic kernel is implemented as a single process, with all elements sharing the same address space. A **microkernel** is a small privileged operating system core that provides process scheduling, memory