SELF-ASSESSMENT PROCEDURE XX

A self-assessment procedure on operating systems

by J. Rosenberg, A. L. Ananda, and B. Srinivasan

What is Self-Assessment Procedure XX?
This is the 20th self-assessment procedure. All the previous ones are listed on the facing page. The first 13 are available from ACM* in a single loose-leaf binder to which later procedures may be added.

This procedure is intended to allow computer professionals to test their knowledge of general concepts of operating systems and includes questions concerning terminology, process and memory management, capability, and file systems. In all except two cases, there is supposed to be only one correct answer for each of the multiple-choice questions.

The next few paragraphs repeat the introduction and instructions given with earlier procedures. Those who read them before may advance directly to the questions.

What is Self-Assessment?
Self-assessment is based on the idea that a procedure can be devised that will help a person appraise and develop his or her knowledge about a particular topic. It is intended to be an educational experience for a participant. The questions are only the *beginning* of the procedure. They are developed to help the participant think about the concepts and decide whether to pursue the matter further.

The primary motivation of self-assessment is *not* for an individual to satisfy *others* about his or her knowledge; rather it is for a participant to appraise and develop his or her own knowledge. This means that there are several ways to use a self-assessment procedure. Some people will start with the questions. Others will read the answers and refer to the references first. These approaches and others devised by the participants are all acceptable if at the end of the procedure the participant can say, "Yes, this has been a worthwhile experience" or "I have learned something."

How to Use the Self-Assessment Procedure
We suggest the following way of using the procedure, but as noted earlier, there are others. This is not a timed exercise; therefore, plan to work with the procedure when you have an hour to spare, or you will be shortchanging yourself on this educational experience. Go through the questions, and mark the responses you think are most appropriate. Compare your responses with those suggested by the Committee. In those cases where you differ with the Committee, look up the references if the subject seems pertinent to you. In those cases in which you agree with the Committee, but feel uncomfortable with the subject matter, and the subject is significant to you, look up the references.

Some ACM chapters may want to devote a session to discussing this self-assessment procedure or the concepts involved.

The Committee hopes some participants will send comments.

Authors' Addresses: J. Rosenberg, Dept. of Electrical Engineering and Computer Science, University of Newcastle, Newcastle, N.S.W. 2308, Australia; A. L. Ananda, Dept. of Information Systems and Computer Science, National University of Singapore, Singapore 0511; B. Srinivasan, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, Australia.

* Order number 203840. ACM Order Dept., P.O. Box 64145, Baltimore, MD 22164. Members—$15. Non-members—$25.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1990 ACM 0001-0782/90/0200-0190 $1.50

Previous Self-Assessment Procedures

Self-Assessment Procedure I
Three concept categories within the programming skills and techniques area
May 1976

Self-Assessment Procedure II
System organization and control with information representation, handling, and manipulation
May 1977

Self-Assessment Procedure III
Internal sorting
September 1977

Self-Assessment Procedure IV
Program development tools and methods, data integrity, and file organization and processing
February 1978

Self-Assessment Procedure V
Database systems
Peter Scheuermann and C. Robert Carlson
August 1978

Self-Assessment Procedure VI
Queueing network models of computer systems
J. W. Wong and G. Scott Graham
August 1979

Self-Assessment Procedure VII
Software science
M. H. Halstead and Victor Schneider
August 1980

Self-Assessment Procedure VIII
The programming language Ada
Peter Wegner
October 1981

Self-Assessment Procedure IX
Ethics in computing
Edited by Eric A. Weiss, from a book by Donn B. Parker
March 1982

Self-Assessment Procedure X
Software project management
Roger S. Gourd
December 1982

Self-Assessment Procedure XI
One part of early computing history
Eric A. Weiss
July 1983

Self-Assessment Procedure XII
Computer architecture
Robert I. Winner and Edward M. Carter
January 1984

Self-Assessment Procedure XIII
Binary search trees and B-Trees
Gopal K. Gupta
May 1984

Self-Assessment Procedure XIV
Legal issues of computing
Jane P. Devlin, William A. Lowell, and Anne E. Alger
May 1985

Self-Assessment Procedure XV
File processing
Martin K. Solomon and Riva Wenig Bickel
August 1986

Self-Assessment Procedure XVI
Computer organization and logic design
Glen G. Langdon, Jr.
November 1986

Self-Assessment Procedure XVII
ACM
Eric A. Weiss
October 1987

Self-Assessment Procedure XVIII
Data Communications
John C. Munson
March 1988

Self-Assessment Procedure XIX
Copyright Law
Riva W. Bickel
April 1989

Approved and submitted by the ACM Committee on Self-Assessment, a committee of the ACM Education Board

Chairman Neal S. Coulter
Department of Computer Science
Florida Atlantic University
Boca Raton, FL 33431

Members Richard E. Fairley
George Mason University

Mark J. Jensen
IBM-Austin, TX

Randy E. Michelsen
Los Alamos National Laboratory

Edward G. Pekarek, Jr.
Appalachian State University

Eric A. Weiss
Editor

Self-Assessment Procedure XX

This self-assessment procedure is not sanctioned as a test nor endorsed in any way by the Association for Computing Machinery. Any person using any of the questions in this procedure for the testing or certification of anyone other than himself or herself is violating the spirit of this self-assessment procedure and the copyright on this material.

Contents

Part I. Questions

TERMINOLOGY

1. A *process* may be defined as:

   a. A set of instructions to be executed by a computer.
   b. A program in execution.
   c. A piece of hardware that executes a set of instructions.
   d. The main procedure of a program.

2. A *processor* in the context of computing is:

   a. A set of instructions to be executed on a computer.
   b. A program in execution.
   c. A piece of hardware that executes a set of instructions.
   d. The main procedure of a program.

3. A *multiprogramming* system may be defined as one in which:

   a. Programs are divided into pages.
   b. Input is accepted in batches of many jobs.
   c. Several programs can reside in memory at the same time.
   d. Many processes may share the same program residing in main memory.

4. The main distinction between a *multiprocessor* system and a *multiprogrammed* system is that in a multiprocessor system:

   a. The main storage is shared by several programs.
   b. The input is accepted in batches of many jobs.
   c. Processor time is shared among several processes.
   d. Many processors may be active simultaneously.

PROCESS MANAGEMENT

5. A user process can become blocked only if it is:

   a. In the ready state
   b. In the running state.
   c. In the blocked (or waiting) state.
   d. Waiting for a resource.

6. A counting semaphore was initialized to 9. Then 27 P (wait) operations and 23 V (signal) operations were completed on this semaphore. The resulting value of the semaphore is:

   a. 5
   b. 0
   c. 7
   d. 13

7. The main difference between binary semaphores and counting semaphores is that:

   a. Binary semaphores can only take the values 0 and 1, while counting semaphores can take any non-negative integer values.
   b. Binary semaphores can only be used to solve problems involving up to two processes sharing the same resource, while counting semaphores can be used to solve problems involving more than two processes sharing the same resource.
   c. Binary semaphores cannot solve all the problems that can be solved by counting semaphores.
   d. Counting semaphores must be controlled by a monitor, while binary semaphores are called directly by user processes.

8. A wait operation on a semaphore should not occur within a critical section controlled by that semaphore because:

a. A deadlock will occur.
b. A semaphore is not a shared variable.
c. Another process may then enter the critical section violating the mutual exclusion constraint.
d. A signal on a semaphore is always given from outside the critical section.

9. Which of the following statements is **false**?

a. Disjoint processes need not use critical sections.
b. Programs with critical sections can never be used simultaneously by more than one process.
c. A process wanting to enter a critical section currently in use must wait for the process utilizing the critical section to terminate.
d. Two different critical sections may be executed concurrently if they do not use the same shared variables.

10. The basic principle of a *monitor* is that:

a. Several resources can only be controlled by a monitor.
b. Several processes may concurrently execute a procedure of a given monitor.
c. Only one process may execute a procedure of a given monitor at any given time.
d. It schedules the execution of processes in a multiprocessor operating system.

11. Which of the following actions may result in a process becoming blocked?

a. A process executes a P (wait) operation on a semaphore.
b. A process executes a V (signal) operation on a semaphore.
c. A process exits from a critical section.
d. A process within a critical section changes the value of a shared variable.

12. Non-preemptive–process-scheduling policies:

a. Are indispensable for interactive systems.
b. Allocate the processor to a process for a fixed time period.
c. Always use a ready queue sorted in order of decreasing priority.
d. Make short jobs wait for long jobs.

13. The *pure-round-robin*–scheduling policy:

a. Responds poorly to short processes if the time slice is small.
b. Does not use any a priori information about the service times of processes.
c. Becomes equivalent to the Shortest-Job-First Policy when the time slice is made infinitely large.
d. Ensures that the ready queue is always the same size.

14. Which of the following statements is **true**?

a. A multiprogrammed system gives better average turnaround than a non-multiprogrammed system.
b. When a job-scheduling policy is changed, it is possible for the average turnaround time to decrease while the average priority-weighted turnaround time increases.
c. There is no job scheduler in a time-sharing system.
d. Indefinite postponement of a job is possible if the First-Come-First-Serve–job-scheduling policy is used.

15. Which of the following statements if **false**?

a. I/O-bound processes should be given priority in scheduling over CPU-bound processes to ensure good turnaround time.
b. Users can exploit a multilevel feedback-scheduling policy by breaking a long job into several small jobs.
c. The processor scheduler normally classifies a process as being a CPU-bound process if it uses most of the previous time slice allocated to it.
d. The *round-robin*–scheduling policy allocates a time slice to a process depending on the number of time slices it has already used.

16. Which of the following is **not** a necessary condition for a deadlock?

a. Mutually exclusive use of a resource by processes.
b. Partial allocation of resources to a process.
c. Preemptive scheduling of resources.
d. Circular waiting by processes.

17. One solution to the Dining Philosophers problem which avoids deadlock is:

a. Non-preemptive scheduling.
b. Ensuring that all philosophers pick up their left fork before they pick up their right fork.
c. Ensuring that all philosophers pick up their right fork before they pick up their left fork.
d. Ensuring that odd philosophers pick up their left fork before they pick up their right fork and even philosophers pick up their right fork before they pick up their left fork.

18. Which of the following statements is **true** for the Banker's algorithm?

a. It cannot be used for a system with many resources, each of which is unique with no multiple copies.

b. It is used to detect deadlock.
c. It is not applicable when a resource is shared simultaneously by many users.
d. An unsafe situation will always lead to a deadlock.

19. Consider a system in which the total available memory is 48K and in which memory once allocated to a process cannot be preempted from that process. Three processes A, B, and C have declared in advance that the maximum amount of memory that they will require is 25K, 15K, and 41K words respectively. When the three processes are all in execution and using 3K, 9K, and 24K words of memory respectively, which one of the following requests for additional allocation can be granted with a guarantee that deadlock will not occur as a result of the allocation.

a. A requests 9K words.
b. C requests 7K words.
c. B requests 6K words.
d. A requests 6K words.

MEMORY MANAGEMENT

20. Which of the following statements is **true**?

a. When the best-fit method of allocating segments is used, it is preferable to order the list of free blocks according to increasing memory addresses.
b. The best-fit method chooses the largest free block in which the given segment can fit.
c. In general, the first-fit allocation algorithm will be faster than the best-fit algorithm.
d. The tagged method of deallocating segments is fast when the list of free blocks is ordered according to increasing memory addresses.

21. In a variable partition-memory management scheme, internal fragmentation occurs when:

a. Sufficient memory is available to run a program, but it is scattered between existing partitions.
b. Insufficient memory is available to run a program.
c. The partition allocated to a program is larger than the memory required by the program.
d. A program is larger than the size of memory on the computer.

22. The FIFO page-replacement policy:

a. Is based on program locality.
b. Sometimes can cause more page faults when memory size is increased.
c. Is not easy to implement, and hence, most systems use an approximation of FIFO.

23. Which of the following statements is **false**?

a. With the Least Recently Used (LRU) page-replacement policy, when the page size is halved, the number of page faults can be more than double the original number of page faults.
b. The working set size is a monotonically nondecreasing function of the working set parameter.
c. When the working set policy is used, main memory may contain some pages which do not belong to the working set of any program.

24. It is advantageous for the page size to be large because:

a. Less unreferenced data will be loaded into memory.
b. Virtual addresses will be smaller.
c. Page tables will be smaller.
d. Large programs can be run.

25. It is advantageous for the page size to be small because:

a. Less unreferenced data will be loaded into memory.
b. Virtual addresses will be smaller.
c. Page tables will be smaller.
d. Large programs can be run.

26. For a certain page trace starting with no page in the memory, a demand-paged memory system operated under the LRU replacement policy results in 9 and 11 page faults when the primary memory is of 6 and 4 pages, respectively. When the same page trace is operated under the optimal policy, the number of page faults may be:

a. 9 and 7.
b. 7 and 9.
c. 10 and 12.
d. 6 and 7.

27. In a paged segmented scheme of memory management, the segment table points to a page table because:

a. The segment table may occasionally be too large to fit in one page.

b. Each segment may be spread over a number of pages.
c. Page size is usually larger than the segment size.
d. The page table may be too large to fit into a single segment.

28. Sharing in a paged memory system is done by:

a. Giving a copy of the shared pages to each process.
b. Dividing the program into procedures and data and allowing only the procedures to be shared.
c. Several page table entries pointing to the same frame in the main memory.

29. One of the ways of sharing segments in a segmented system is by:

a. Maintaining a common segment table containing the information about shared segments.
b. Dividing the program into procedures and data and allowing only the procedures to be shared.
c. Dividing the shared segment into a set of pages and allowing only certain pages to be shared.
d. None of the above, as segments are larger than pages, and hence, cannot be shared.

30. With reference to Multics, which of the following statements is **false**?

a. Every process must have a separate linkage segment for every shared segment.
b. The linkage segment need not be used to resolve internal references of a segment.
c. When control is transferred from one segment to another, the linkage pointer must be changed to point to the new linkage segment.
d. The linkage segment is constructed at the time of linking.

31. An advantage of dynamic linking is that:

a. The segments that are not used in a run need not be linked into the process address space.
b. It reduces execution time overhead.
c. Debugging is simplified because programs are modular.
d. The linker need not construct the known segment table.

CAPABILITY

32. In operating systems, a *capability* is:

a. A facility which provides global access to all data in the system.
b. A unique and nonforgeable name identifying an object in the system together with access information.
c. A user-maintained list of access privileges of the objects in the system.
d. A table of the available operating system resources.

33. In capability-based systems, which of the following statements is **true**?

a. The unique name of a capability is reused after every time slice.
b. The unique name of a capability is never reused.
c. The unique name of a capability can be reused if there are no references to it.

FILE SYSTEMS

34. Disk scheduling involves:

a. Allocating disk space to users in a fair manner.
b. Validating the file control information stored in the file.
c. Examining pending disk requests to determine the most efficient way to service the requests.
d. Reorganizing disk requests to maximize seek time.

35. Which of the following is **not** normally contained in the directory entry of a file?

a. Creation date.
b. Access control list.
c. A count of the number of free blocks in the disk.
d. Filename and extension.

36. Which of the following is an example of a spooled device?

a. A line printer used to print the output of a number of jobs.
b. The terminal used to enter the input data for a Fortran program being executed.
c. The secondary memory device in a virtual memory system.
d. The swapping area on a disk used by the swapper.

Part II. Suggested Responses

1. b The term *process* was borrowed from chemical engineering by the designers of the **Multics** system in the 1960s and is used interchangeably with the term *task*. The generally accepted definition of a process is a program in execution.
2. c A *processor* is any piece of hardware that executes a set of instructions.
3. c Multiprogramming is, by definition, having several programs in memory at the same time. In time-sharing systems, for example, a process currently requiring an input/output operation will yield the CPU to another process which is ready to run. Such switching is also called CPU scheduling. A time-sharing system generally has multiprogramming too.
4. d A *multiprocessor* system has more than one processor. Each processor can have its own memory or share a common memory. The former belongs to the class of loosely coupled systems and the latter to the tightly coupled systems. Each processor executes a separate process, and many processors may be active simultaneously.
5. a, b A process can become blocked if it is either ready or in the running state. A process can become blocked while running if it requests a resource in use by another process. It is also possible for a process to become blocked while ready. For example, in a preemptive resource-allocation scheme, a low-priority process could be ready to run. While that process is waiting in the ready queue, if a higher-priority process requests a resource that is held by a ready process, the operating system can preempt the resource and allocate it to the higher-priority process, thereby blocking the process which was in the ready state.
6. a Each P (wait) operation would decrement the semaphore value by one while each V (signal) operation would increment the semaphore value by one. Thus, for an initial value of 9, 27 waits and 23 signals would result in a semaphore value of $9 - 27 + 23 = 5$.
7. a The power of counting semaphores and binary semaphores is the same. The counting semaphore is particularly useful when a resource is to be allocated from a pool of identical resources. The counting semaphore for such a resource is initialized to the number of identical resources in the pool. Each P operation decrements the value of the semaphore indicating that another resource has been allocated, and each V operation increments the semaphore value by 1 indicating that a process has returned the resource to the pool. If a process attempts a P operation on the semaphore whose value is zero, then the process has to wait until a resource is returned to the pool.
8. a When a semaphore is used to implement mutual exclusion it is initialized to 1. In order to enter the critical region a process executes a wait on the semaphore which decrements it to zero. The next process attempting to enter will then wait. On exitting the critical region, the process executes a signal which activates the next process (if there is one waiting) or increments the semaphore. If a process executes a wait on the semaphore while in the critical region it will be suspended, and no other process will be able to enter. Thus the processes waiting for the semaphore will be deadlocked.
9. b Disjoint processes that do not share any data need not require critical sections. A critical section contains shared data, and hence, any process wanting to enter the critical section which is in use by another process must wait for the process utilizing the critical section to terminate. Hence, programs with critical sections can be shared by many processes as long as only one process is accessing the data in the critical section at any time.
10. c The *monitor* is a synchronization mechanism based on data abstraction. The shared data is encapsulated in routines, and automatic mutual exclusion is provided on these routines; only one process may execute a procedure of a monitor at any time. Monitors also have a queuing mechanism for waiting on conditions.
11. a V operations can never cause a process to become blocked. Processes can be blocked by executing a P operation.
12. d In general, a non-preemptive process-scheduling policy can make short jobs wait longer in the ready queue, thereby giving poor response to those processes. This is particularly true when the system is heavily loaded with long jobs.
13. b With the *Round-Robin*-scheduling policy, processes are dispatched in FIFO order but are given a limited amount of CPU time, called the *time slice* or *quantum*. If a process does not complete before the time slice expires, it is preempted, and the CPU is given to the next waiting process in the ready queue. The preempted process is placed at the back of the ready queue. Hence no a priori knowledge is used to determine the next process to schedule.
14. b Multiprogramming systems do not always perform better than single program systems because of the overhead of switching between processes. It is possible for the average turnaround to improve because of the change in the job scheduling policy, but at the same time it may result in a longer turnaround for high-priority jobs. This could happen when the priority of the job is given less importance in the new scheduling algorithm. Time-sharing systems sometimes have a job scheduler to limit the number of logged-on users to a level that

the system can cope with. Under the first-come-first-serve algorithm, it is not possible for a job to be indefinitely postponed because jobs are executed in the order of their arrival.

15. d I/O-bound processes use little CPU time in general, and hence, in order to have better *turnaround time* (defined as the number of processes completing execution per unit of time), I/O processes should be given more priority over CPU-bound processes. In the case of a multilevel feedback system, multiple wait queues are maintained. The queue on which a process waits for CPU allocation is determined by its previous queue and the amount of CPU time used in the previous allocated CPU time slice. If a process uses its entire time slice (i.e., a CPU-bound process) it is moved down in the hierarchy and will thus have a lower priority in the next CPU-time-slice allocation. Hence, response to a long job can be improved by dividing it into several smaller jobs. For discussion about the Round-Robin–scheduling policy, refer to question 13.

16. c In a deadlock, processes never finish executing, and system resources are tied up. This prevents other jobs from ever starting. The following are the necessary conditions that must be satisfied for a deadlock to exist.

- Processes claim exclusive control of the resources they require (*mutual exclusion* condition)
- Processes hold resources already allocated to them while waiting for additional resources (*hold and wait* condition)
- Resources cannot be removed from the processes holding them until the resources are used to completion (*no preemption* condition)
- A circular chain of processes exists in which each process holds one or more resources that are requested by the next process in the chain (*circular-wait* condition).

17. d The scheduling algorithm given (d) will guarantee that there will be no deadlock. Both (b) and (c) allow the possibility of all philosophers picking up one fork resulting in deadlock. There are other solutions to the Dining Philosophers problem.

18. c The Banker's algorithm (initially described by Dijkstra in 1965) is an algorithm which will detect whether a given allocation of resources could result in a deadlock. It does not detect the existence of a deadlock. The algorithm can handle single or multiple equivalent or independent resources. The algorithm is pessimistic in that it flags situations as being potential deadlocks when, in fact, a deadlock may not occur. Simultaneously shared resources can never be in a deadlock situation, and thus, the algorithm is inapplicable to these cases.

19. c After initial allocation to the processes A, B, and C, the remaining available memory is 12K. If process A makes a further request of 9K and if the request is satisfied, it could still require 13K at some point before its completion. This requirement cannot be satisfied at this point of time because, if A gets 9K more, no process can get its maximum memory request, so deadlock is possible. A similar argument holds good for the choices (b) and (d). However, if 6K is allocated to process B (which can be satisfied from the available memory), its maximum requirement is satisfied, and hence it can run to completion. After a finite amount of time the memory held by process B will be released. Then process C's requirements can be met, and in a finite amount of time it will complete its execution which in turn allows process A to complete. Hence, the allocation of 6K memory for process B under the current allocation will result in a *safe* situation.

20. c In the first-fit allocation algorithm, the free blocks are searched sequentially until a free block whose size is greater than or equal to the requested memory size is found. The best-fit algorithm tries to find and allocate the free block (if it exists) that is the closest (but at least as large) to the required size. Empirical study shows that a first-fit allocation is generally faster than a best-fit allocation although there are circumstances where either one can be better than the other. For a detailed comparison of the first-fit and best-fit strategies refer to Shore, J. E., "On the External Storage Fragmentation Produced by First-Fit and Best-Fit Allocation Strategies," (*Communications* August 1975, pp. 433–440).

The tagged method is used (by using tag fields to indicate whether the block is free or used by a process) for coalescing adjacent free blocks when a block of memory is released.

21. c Internal fragmentation occurs when a partition allocated to a program is larger than that required. The difference between these two sizes is the amount of memory that is wasted. Choice (a) refers to external fragmentation.

22. b It would seem reasonable to expect that if more pages are allocated to a process, then it should experience fewer page faults. But under the FIFO page-replacement strategy, certain page-reference patterns actually cause more page faults when the number of pages allocated to the process is increased. This phenomenon is known as *Belady's anomaly*. For more information refer to *Belady, L. A., Nelson, R. A., and Shedler,* G. S., "An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Environment" (*Communications* December 1969, pp. 349–353).

23. a For a given pattern of references and a given page size, the number of page faults is governed by the number of different pages that are accessed. If the page size is halved, then no more than the original number of full-sized pages can be accessed, and thus, no more than twice the original number of page faults can occur.

24. c Increasing the size of the page will decrease the number of pages of the address space, and hence, the number of entries in the page table will be smaller.

25. a By reducing the page size, the amount of unreferenced data held in memory will be reduced. However, the size of the page table will increase.

26. b The LRU scheme results in 9 and 11 page faults for memory sizes of 6 and 4 pages, respectively. Hence, the optimal (OPT) scheme must have a and b number of page faults for a memory size of 6 and 4 pages, respectively, where $a \leq 9$ and $b \leq 11$ and $0 \leq a \leq b$. This is because OPT must be at least as good as LRU and must not have more faults if there is more memory. Initially, the memory is empty, and hence, in both cases, there must be an initial set of accesses requiring more than the total number of pages of memory in order to cause page faults greater than the number of memory pages. Thus, even with OPT, there must be at least a number of page faults greater than the number of pages of memory for each memory size, that is $a > 6$ and $b > 4$. Hence, $6 < a \leq 9$ and $4 < b \leq 11$. Therefore, it should be noted that neither OPT nor LRU scheme suffers from *Belady's anomaly*.

27. b Segments are variable-sized memory blocks. In a paged segmented system, each segment is divided into one or more fixed-sized pages. Hence, an entry for a segment points to a page table which contains the addresses of the pages corresponding to that segment.

28. a, c Sharing pages can be achieved either by providing a copy of the page or by having the same entry in the page tables that point to the shared page. The latter does not have the duplicate copy of the page, and hence, the memory-space requirement and the overhead involved in maintaining the consistency of shared pages is reduced. There need be no restriction that only procedures and not data can be shared in a paged system.

29. a The sharing of segments in a segmented system can be done by having a common segment table. All virtual addresses above a certain segment number can be translated using the information in the common segment table rather than the entries in the local segment table. This method will avoid duplication of the entries in the shared segments in each local segment table.

30. a A linkage segment is only required when there are external references from a shared segment. Hence, if some objects are shared but have no external references, then there will be no linkage segment.

31. a Dynamic linking allows procedures to be linked and loaded into memory when they are first called. Hence, procedure segments that are not required for a particular execution need not be in memory, thereby reducing the amount of memory required.

32. b A capability is a unique nonforgeable name directly identifying an object in a system, together with access information. Users cannot manufacture or modify capabilities. Possession of a capability gives the right to access the object, governed by the access information.

33. c The unique name in capabilities can be reused, but only if it can be guaranteed that no references remain to the old use of the name.

34. c In multiprogrammed systems, many processes may be generating requests for reading and writing disk records. Because these processes often make requests faster than they can be retrieved by moving head disks, queues build up for such devices. Some systems use the *first-come-first-serve* (FCFS) policy to service the disk queues. But FCFS exhibits a seek pattern in which successive requests can cause time consuming seek delays. To minimize the time spent in seeking records, it is reasonable to order the request queue. Such a method of ordering the disk requests is called *disk scheduling*. Disk scheduling generally tries to minimize the head-movement time.

35. c The directory entry generally contains information pertaining to a file that includes the file name and possibly some or all of the following: file extension, date of creation, date of last modification, size (may be in kilobytes), access rights, etc. The number of free blocks on the disk is information related to the whole disk and is not kept as part of a file.

36. a In spooling (Simultaneous Peripheral Operation On Line), a buffer is interposed between a running program and a slow-speed device involved with the program for input/output. For example, instead of writing lines directly to the line printer, the lines are written (temporarily) onto a disk. Thus, the program is allowed to run to completion without waiting for the (slow) printer to finish. When the printer becomes free, the lines can be printed from the disk file.

Part III. Reference Table

The following table gives the page numbers of the suggested references where a discussion on the question can be found. Suggested references are fully cited in Part IV, Reference Titles.

	Calingaert	Deitel	Janson	Peterson, Silberschatz	Tanenbaum	Theaker, Brookes
1	53	55	24	105–106	46–47	—
2	53–55	63–64	4–6	—	—	—
3	21–22	7	64–68	17–18	8–11	10–11
4	203–208	278	23–26	33–34	—	—
5	54–57	56–57	36–38	107–108	48–49	165–167
6	95–100	89–93	39–40	340–344	60–62	163–169
7	89–95	89–93	32–42	340–344	61–63	163–169
8	95–100	89–90	42–44	375–378	63–67	165–169
9	84–92	77–78	42–44	327–347	53	169–172
10	104–105	103–104	44–45	384–393	63–67	184–185
11	95–100	93	72–75	340–344	60–62	165
12	57–60	252–253	72–75	120–122	80–81	58–59
13	59–60	252–256	77–78	122–125	81–82	63–64
14	65–67	252–261	—	115–136	80–87	—
15	57–65	252–261	72–75	115–125	80–87	42–55
16	108	131	51–52	275–276	124	136–137
17	—	119–121	—	347–349	75–78	—
18	110–111	135–139	54–56	287–289	131–135	140–142
19	110–111	135–139	54–56	287–289	131–135	139–140
20	46–47	169–174	111–112	76	201–203	124–126
21	22	—	—	151	205–206	77–79
22	39–40	218–220	126–127	218–220	215–216	117
23	35–41	222–226	129–130	217–237	213–219	115–117
24	35–36	229–233	117–134	241–242	222–223	92–95
25	35–39	229–233	117–134	241–242	222–223	92–95
26	39–40	217–221	118–122	220–224	213–217	115–116
27	42–45	203–208	133–136	191–194	209–212	97–98
28	—	194–195	—	178–179	207–226	92–97
29	—	202–208	—	188–190	—	85–86
30	—	—	179–184	187–195	210–212	—
31	34	—	188–189	187–195	—	—
32	—	454–456	216–217	412–413	293–295	153–154
33	—	454–456	216–222	412–413	293–295	153–157
34	128–130	301–320	97–98	259–270	144–146	—
35	146–156	334–336	148–156	82–89	254–256	128–129
36	143–144	35	10–12	15–17	121–122	20–23

Part IV. Reference Titles

Suggested References

1. Calingaert, P. *Operating System Elements—A User Perspective.* Prentice Hall, Englewood Cliffs, N.J., 1982.
2. Deitel, H.M. *An Introduction to Operating Systems.* Addison-Wesley, Reading, Mass., 1984 (revised first edition).
3. Janson, P.A. *Operating Systems Structures and Mechanisms.* Academic Press, New York, 1985.
4. Peterson, J. L., and Silberschatz, A. *Operating System Concepts.* 2d ed. Addison-Wesley, Reading, Mass., 1985.
5. Tanenbaum, A.S. *Operating Systems—Design and Implementation.* Prentice Hall, Englewood Cliffs, N.J., 1987.
6. Theaker, C.J., and Brookes, G.R. *A Practical Course on Operating Systems.* Macmillan Press Ltd., United Kingdom, 1983.

Additional References

7. Bic, L., and Shaw, A.C. *Logical Design of Operating Systems.* 2d ed. Prentice Hall, Englewood Cliffs, N.J., 1988.
8. Christian, K. *The Unix Operating Systems.* John Wiley & Sons, New York, 1983.
9. Comer, D. *Operating System Design—The XINU Approach.* Prentice Hall, Englewood Cliffs, N.J., 1984.

10. Davis, W.S. *Operating Systems—A Systematic View.* 3d ed. Addison-Wesley, Reading, Mass., 1987.
11. Kaisler, S.H. *The Design of Operating Systems for Small Computer Systems.* John Wiley & Sons, New York, 1983.
12. Maekawa, M., Oldehoeft, A.E., and Oldehoeft, R.R. *Operating Systems—Advanced Concepts.* Benjamin/Cummings Publishing Co. Inc., 1987.
13. Milenkovic, M. *Operating Systems—Concepts and Design.* McGraw-Hill, New York, 1987.
14. Turner, R.W. *Operating Systems—Design and Implementation.* MacMillan Publishing Company, New York, 1986.

Part V. Acknowledgments

The authors would like to acknowledge the referees for their constructive criticisms and the valuable suggestions which made this assessment procedure more consistent and readable.

Epilogue

Now that you have reviewed this self-assessment procedure and have compared your responses to those suggested, you should ask yourself whether this has been a successful educational experience. The Committee suggests that you conclude that it has only if you have

—discovered some concepts that you did not previously know about or understand, or
—increased your understanding of those concepts that were relevant to your work or valuable to you.

ACM Self-Assessment Procedures
Guide for Prospective Authors

Self-assessment procedures are intended to be fairly short mechanisms to help members of ACM appraise and develop their knowledge of subjects important to them in their roles as computer professionals. The purpose of the procedures is tutorial. The subjects of the procedures should be about computing, of widespread interest or importance to ACM members, and comprehensible to the average ACM member after a reasonable amount of effort. However, the subjects need not be of universal interest within the ACM community. The procedure need not present a balanced view of all known ways of solving or viewing a particular problem as long as the procedure is accurate.

The procedure should be aimed at the general ACM membership, not at specialists. The set of items in the procedure seldom would make a good graduate student examination, although some of the items conceivably might be used in such a context.

It is important to keep in mind that the self-assessment procedure is not intended as a test or certification of knowledge for anyone other than the person reading the procedure.

The items in the procedure should be of widely varying difficulty; a few should be easy enough for virtually any ACM member to answer or make a reasonable guess at. The author should supply about 30 items, some or all of which may be based on short examples placed in the procedure. Most of the items in published procedures have been in multiple-choice form, but this is not necessary as long as reasonably short responses can be provided. Some items have had more than one correct response, which is fine as long as the item is appropriately worded. It is suggested that the items not be arranged in order of increasing difficulty and that some easy items appear very near or at the beginning, and occasionally throughout.

Responses should be provided for almost all of the items. Occasionally, an open question might be included (a procedure consisting entirely of open questions would be unusual).

Every item and its response should be associated with a reference. These references should be as precise as possible (including page and, if appropriate, line or paragraph number). References should be only to a few publicly available documents. One should be able to obtain the references without having access to a huge library. If the author can find no references for a response, this probably indicates that the subject or item is too new to appear in a self-assessment procedure.

It is desirable to provide an additional short bibliography for readers who become interested enough to read further. If a good bibliography has already been published, a reference to it should be included as well.

Authors of published procedures have found it useful to test the procedures by asking colleagues and students to work them through. The Committee strongly recommends that this be done prior to submission of a draft.

Please supply the ACM Self-Assessment Committee with your proposed procedure including the following sections: items, responses, references for each item, and bibliography. The Committee will review your procedure and will get technical reviews by experts as needed. If the Committee accepts your procedure, it may ask you to attend a committee meeting to go over any proposed changes. After the authors of accepted procedures sign copyright agreements, the Committee will have the procedure published with an appropriate introduction in *Communications*. The authors of the procedure will be listed as such, as with other *Communications* articles. The membership of the Committee will be listed as part of the procedure.

CONTACT: Neal S. Coulter
Department of Computer Science
Florida Atlantic University
Boca Raton, FL 33431

ACM SPECIAL INTEREST GROUPS

ARE YOUR TECHNICAL INTERESTS HERE?

The ACM Special Interest Groups further the advancement of computer science and practice in many specialized areas. Members of each SIG receive as one of their benefits a periodical exclusively devoted to the special interest. The following are the publications that are available—through membership or special subscription.

SIGACT NEWS (Automata and Computability Theory)

SIGAda Letters (Ada)

SIGAPL Quote Quad (APL)

SIGARCH Computer Architecture News (Architecture of Computer Systems)

SIGART Newsletter (Artificial Intelligence)

SIGBDP DATABASE (Business Data Processing)

SIGBIO Newsletter (Biomedical Computing)

SIGCAPH Newsletter (Computers and the Physically Handicapped) Print Edition

SIGCAPH Newsletter, Cassette Edition

SIGCAPH Newsletter, Print and Cassette Editions

SIGCAS Newsletter (Computers and Society)

SIGCHI Bulletin (Computer and Human Interaction)

SIGCOMM Computer Communication Review (Data Communication)

SIGCPR Newsletter (Computer Personnel Research)

SIGCSE Bulletin (Computer Science Education)

SIGCUE Bulletin (Computer Uses in Education)

SIGDA Newsletter (Design Automation)

SIGDOC Asterisk (Systems Documentation)

SIGFORTH Newsletter (FORTH)

SIGGRAPH Computer Graphics (Computer Graphics)

SIGIR Forum (Information Retrieval)

SIGMETRICS Performance Evaluation Review (Measurement and Evaluation)

SIGMICRO Newsletter (Microprogramming)

SIGMOD Record (Management of Data)

SIGNUM Newsletter (Numerical Mathematics)

SIGOIS Newsletter (Office Information Systems)

SIGOPS Operating Systems Review (Operating Systems)

SIGPLAN Notices (Programming Languages)

SIGPLAN FORTRAN FORUM (FORTRAN)

SIGSAC Newsletter (Security, Audit, and Control)

SIGSAM Bulletin (Symbolic and Algebraic Manipulation)

SIGSIM Simuletter (Simulation and Modeling)

SIGSMALL/PC Newsletter (Small and Personal Computing Systems and Applications)

SIGSOFT Software Engineering Notes (Software Engineering)

SIGUCCS Newsletter (University and College Computing Services)

See the ACM membership application in this issue for additional information.