Midterm Exam Review

Weeks 1-7

Chapters 1-11

Chapter 1 Key Topics

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management
- Protection and Security
- Kernel Data Structures
- Computing Environments
- Open-Source Operating Systems

Chapter 2 Key Topics

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Operating System Debugging
- Operating System Generation
- System Boot

Key Topics Chapter 3

- Process Concept
- Process Scheduling
- Operations on Processes
- Inter-process Communication
- Examples of Inter -Process Communication (IPC) Systems
- Communication in Client-Server Systems

Key Topics Chapter 4

- Overview
- Multicore Programming
- Multithreading Models
- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples

Chapter 5: Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation

Chapter 6 - Synchronization

- Background
- The Critical-Section Problem
- Peterson’ s Solution
- Synchronization Hardware
- Mutex Locks
- Semaphores
- Classic Problems of Synchronization
- Monitors
- Synchronization Examples
- Alternative Approaches

Chapter 7 - Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Chapter 8: Memory Management Strategies

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table
- Example: The Intel 32 and 64-bit Architectures
- Example: ARM Architecture

Chapter 9: Virtual-Memory Management

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Memory-Mapped Files
- Allocating Kernel Memory
- Other Considerations
- Operating-System Examples

Chapter 10: File System

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection

Chapter 11: Implementing File Systems

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- NFS
- Example: WAFL File System

Section 1.1.1

True/***False***

All computer systems have some sort of user interaction.

Section 1.2.1

What role do device controllers and device drivers play in a computer system?

Answer: A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. Typically, operating systems have a device driver for each device controller. This device driver understands the device controller and presents a uniform interface for the device to the rest of the operating system.

Section 1.5.1

What are some other terms for kernel mode?

A. supervisor mode

B. system mode

C. privileged mode

D. All of the above

Section 1.5.1

The two separate modes of operating in a system are

A. supervisor mode and system mode

B. kernel mode and privileged mode

C. physical mode and logical mode

D. user mode and kernel mode

Section 1.11.8

Embedded computers typically run on a ____ operating system.

A. ***real-time***

B. Windows XP

C. Network

D. clustered

Section 2.4

_____ is not one of the major categories of system calls.

A. Process control

B. Communications

C. Protection

D. Security

Section 2.4.1

The Windows CreateProcess() system call creates a new process. What is the equivalent system call in UNIX:

1. NTCreateProcess()
2. process()
3. ***fork()***
4. getpid()

Section 2.4.3

True/False

Many operating system merge I/O devices and files into a combined file because of the similarity of system calls for each.

Section 2.7.3

A microkernel is a kernel ____.

A. containing many components that are optimized to reduce resident memory size

B. that is compressed before loading in order to reduce its resident memory size

C. that is compiled to produce the smallest size possible when stored to disk

D. that is stripped of all nonessential components

Section 2.7.4

_____ allow operating system services to be loaded dynamically.

A. Virtual machines

B. Modules

C. File systems

D. Graphical user interfaces

Section 3.1.1

The ____ of a process contains temporary data such as function parameters, return addresses, and local variables.

A. text section

B. data section

C. program counter

D. stack

Section 3.1.3

A process control block ____.

A. ***includes information on the process's state***

B. stores the address of the next instruction to be processed by a different process // program account

C. determines which process is to be executed next

D. is an example of a process queue

Section 3.1.2

Name and describe the different states that a process can exist in at any given time.

Answer: The possible states of a process are: new, running, waiting, ready, and terminated. The process is created while in the new state. In the running or waiting state, the process is executing or waiting for an event to occur, respectively. The ready state occurs when the process is ready and waiting to be assigned to a processor and should not be confused with the waiting state mentioned earlier. After the process is finished executing its code, it enters the termination state.

Section 3.2.2

Explain the difference between an I/O-bound process and a CPU-bound process.

Answer: The differences between the two types of processes stem from the number of I/O requests that the process generates. An I/O-bound process spends more of its time seeking I/O operations than doing computational work. The CPU-bound process infrequently requests I/O operations and spends more of its time performing computational work.

Section 3.3.1

Ordinarily the exec() system call follows the fork(). Explain what would happen if a programmer were to inadvertently place the call to exec() before the call to fork().

Answer: Because exec() overwrites the process, we would never reach the call to fork() and hence, no new processes would be created. Rather, the program specified in the parameter to exec() would be run instead.

Exercise 3.2

Describe the actions taken by a kernel to context-switch between processes

Answer:

In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must

also perform many architecture-specific operations, including flushing data and instruction caches.

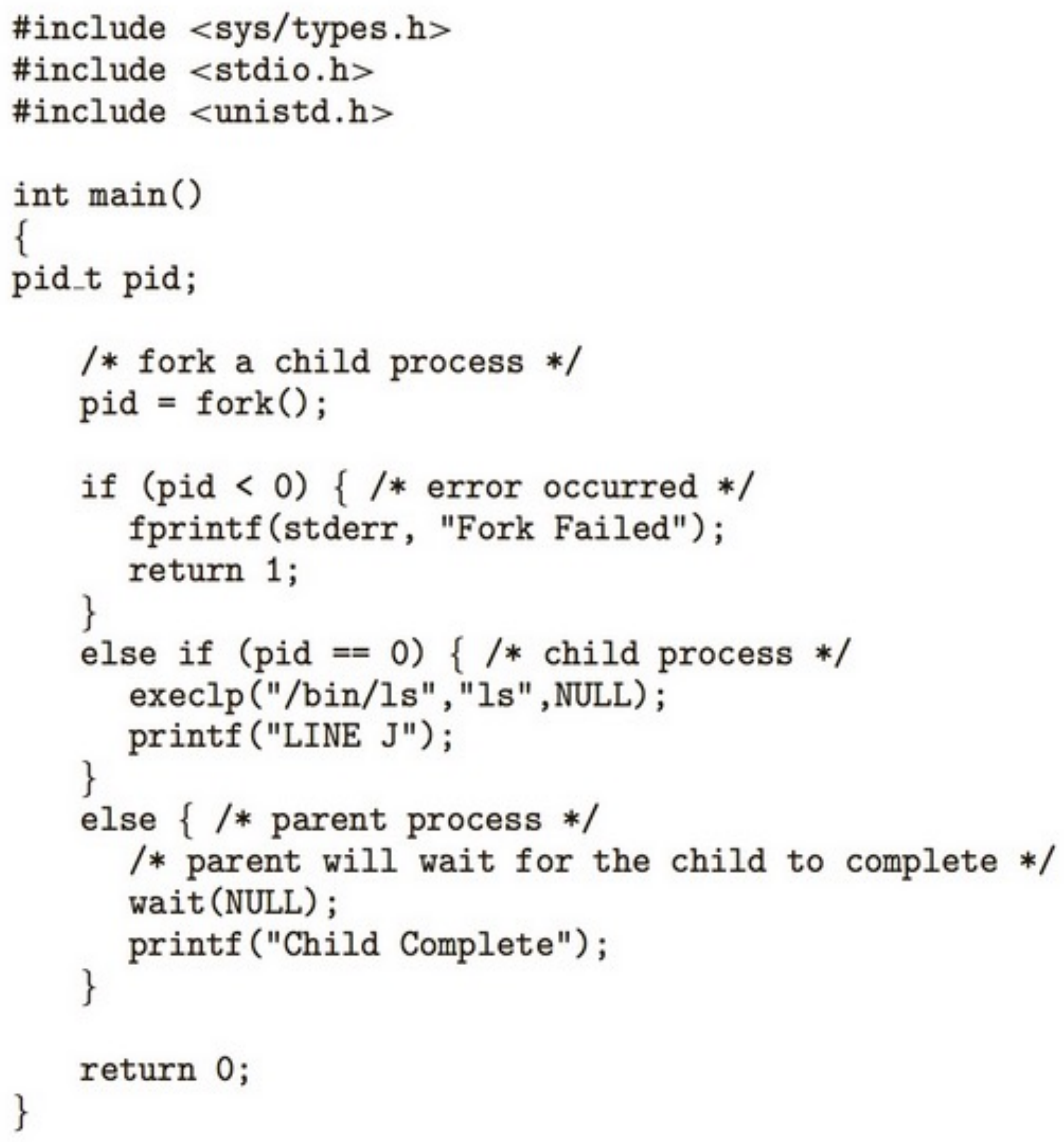
Exercise 3.6

Explain the circumstances when the line of code marked printf ("LINE J") in Figure is reached.

Answer:

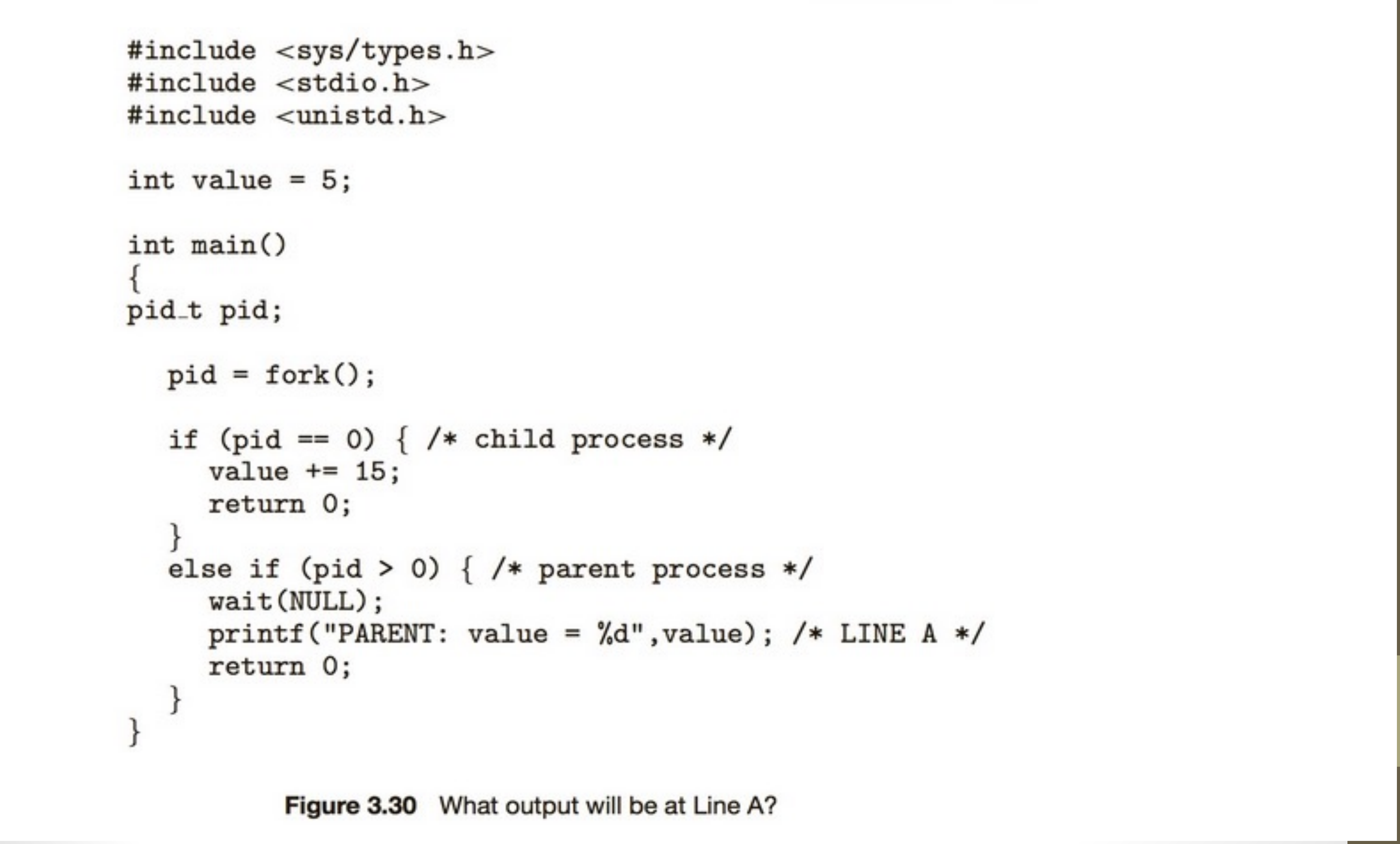
The call to exec() replaces the address space of the process with the program specified as the parameter to exec(). If the call to exec() succeeds, the new program is now running and control from the call to exec() never returns. In this scenario, the line printf("Line J");

would never be performed. However, if an error occurs in the call to exec(), the function returns control and therefor the line printf("Line J"); would be performed.



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
    }
    else if (pid == 0) { /* child process */
      execlp("/bin/ls","ls",NULL);
      printf("LINE J");
    }
    else { /* parent process */
      /* parent will wait for the child to complete */
      wait(NULL);
      printf("Child Complete");
    }

    return 0;
}
```

Optional Practice – Page 147



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main()
{
pid_t pid;

   pid = fork();

   if (pid == 0) { /* child process */
      value += 15;
      return 0;
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE A */
      return 0;
   }
}
```

Figure 3.30 What output will be at Line A?

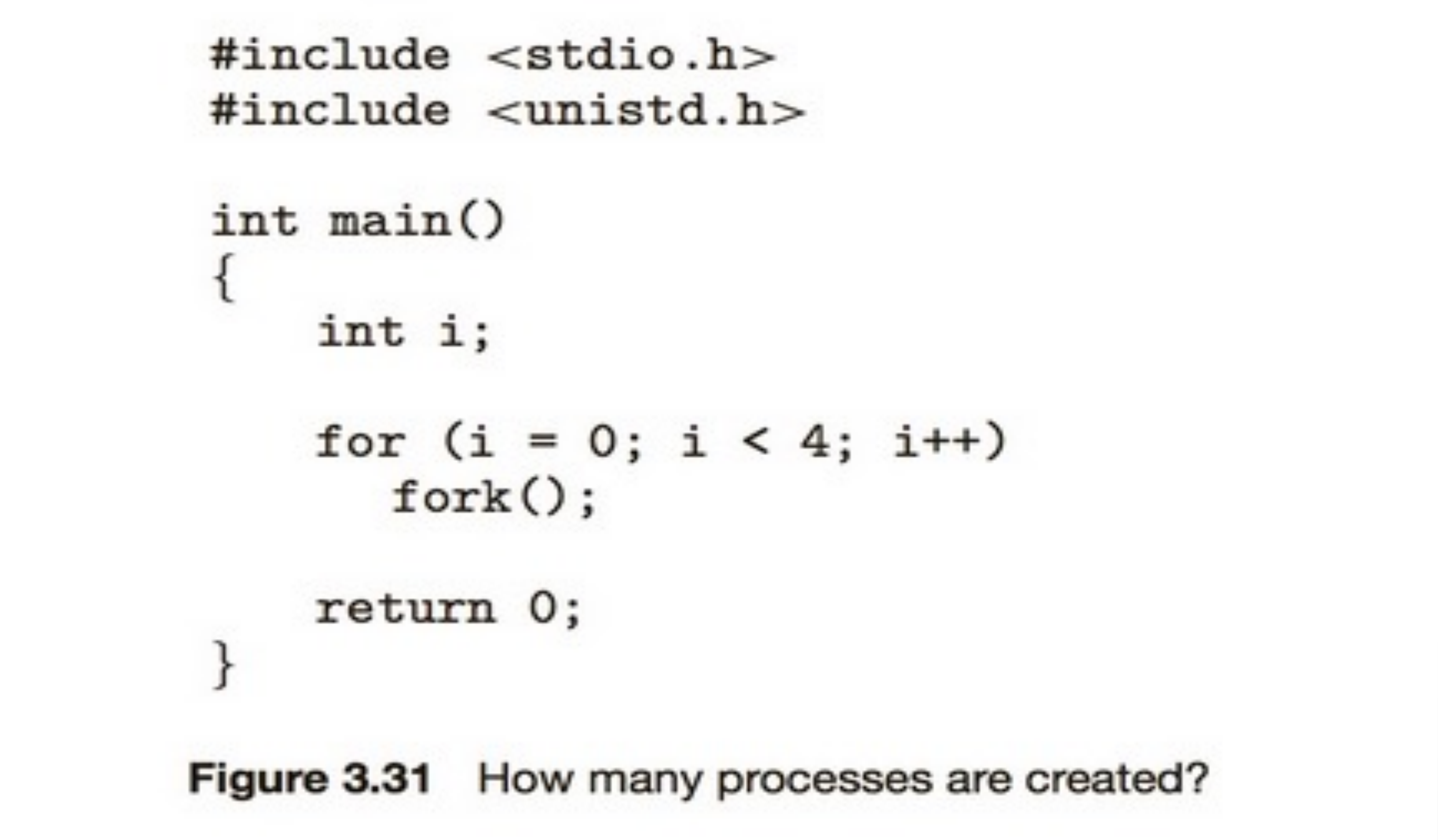
Optional Exercise

Using the program shown in Figure 3.30, explain what the output will be at Line A.

Answer:

The result is still 5 as the child updates its copy of value. When control returns to the parent, its value remains at 5.

Optional Practice – 3.1



```
#include <stdio.h>
#include <unistd.h>

int main()
{
    int i;

    for (i = 0; i < 4; i++)
       fork();

    return 0;
}
```

Figure 3.31 How many processes are created?

Optional Practice 3.1 answer

Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

Answer:

There are 8 processes created.

Parent1

Child1

Child1 becomes **Parent2** for **Child2**

Child2 becomes **Parent3** for **Child3**

Child3 becomes **Parent4** for **Child4**

Exercise 4.11

As described in Section 4.7.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, many

operating systems—such as Windows XP and Solaris—treat processes and threads differently. Typically, such systems use a notation wherein the data structure for a process contains pointers to the separate threads

belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

Exercise 4.11 Answer

Answer:

On one hand, in systems where processes and threads are considered as similar entities, some of the operating system code could be simplified. A scheduler, for instance, can consider the different processes and threads on an equal footing without requiring special code to examine the threads associated with a process during every scheduling step. On the other hand, this uniformity could make it harder to impose process-wide resource constraints in a direct manner. Instead, some extra complexity is required to identify which threads correspond to which process and perform the relevant accounting tasks.

Example Question – Section 3.1.1

The ____ of a process contains temporary data such as function parameters, return addresses, and local variables.

A. text section

B. data section

C. program counter

D. stack

Example Question – Section 3.1.1

The ____ of a process contains temporary data such as function parameters, return addresses, and local variables.

A. text section

B. data section

C. program counter

D. stack

Example Question – Section 3.1.2

A process may transition to the Ready state by which of the following actions?

A. Completion of an I/O event

B. Awaiting its turn on the CPU

C. Newly-admitted process

D. All of the above

Example Question – Section 3.1.2

A process may transition to the Ready state by which of the following actions?

A. Completion of an I/O event

B. Awaiting its turn on the CPU

C. Newly-admitted process

D. All of the above

Example Question – Section 3.1.3

A process control block ____.

A. includes information on the process's state
B. stores the address of the next instruction to be processed by a different process
C. determines which process is to be executed next
D. is an example of a process queue

Example Question – Section 3.1.3

A process control block ____.

A. ***includes information on the process's state***
B. stores the address of the next instruction to be processed by a different process
C. determines which process is to be executed next
D. is an example of a process queue

Example Question – Section 3.2.1

The list of processes waiting for a particular I/O device is called a(n) ____.standby queue

A. device queue
B. ready queue
C. interrupt queue

Example Question – Section 3.2.1

The list of processes waiting for a particular I/O device is called a(n) ____.standby queue

A. device queue

B. ready queue

C. interrupt queue

Example Question – Section 3.2.3

- A _________________ saves the state of the currently running process and restores the state of the next process to run.
  A. save-and-restore
  B. state switch
  C. context switch
  D. none of the above

Example Question – Section 3.2.3

- A ___________________ saves the state of the currently running process and restores the state of the next process to run.

  A. save-and-restore

  B. state switch

  C. ***context switch***

  D. none of the above

Example Question – Section 4.1.1

Pthreads refers to ____.

A. the POSIX standard.

B. an implementation for thread behavior.

C. a specification for thread behavior.

D. an API for process creation and synchronization.

Example Question – Section 4.1.1

Pthreads refers to ____.

A. the POSIX standard.

B. an implementation for thread behavior.

C. ***a specification for thread behavior.***

D. an API for process creation and synchronization.

Example Question – Section 4.3.3

- The ____ multithreading model multiplexes many user-level threads to a smaller or equal number of kernel threads.

  A. many-to-one model

  B. one-to-one model

  C. many-to-many model

  D. many-to-some model

Example Question – Section 4.3.3

- The ____ multithreading model multiplexes many user-level threads to a smaller or equal number of kernel threads.
  A. many-to-one model
  B. one-to-one model
  C. ***many-to-many model***
  D. many-to-some model

Example Question – Section 4.6.2

Which of the following would be an acceptable signal handling scheme for a multithreaded program?

A. Deliver the signal to the thread to which the signal applies.
B. Deliver the signal to every thread in the process.
C. Deliver the signal to only certain threads in the process.
D. All of the above

Example Question – Section 4.6.2

Which of the following would be an acceptable signal handling scheme for a multithreaded program?

A. Deliver the signal to the thread to which the signal applies.

B. Deliver the signal to every thread in the process.

C. Deliver the signal to only certain threads in the process.

D. All of the above

Example Question – Section 4.4.1

Which of the following statements regarding threads is false?

A. Sharing is automatically provided in Java threads.

B. Both Pthreads and Win32 threads share global data.

C. The start() method actually creates a thread in the Java virtual machine.

D. The Java method join() provides similar functionality as the WaitForSingleObject in Win32.

Example Question – Section 4.4.1

Which of the following statements regarding threads is false?

A. ***Sharing is automatically provided in Java threads.***
B. Both Pthreads and Win32 threads share global data.
C. The start() method actually creates a thread in the Java virtual machine.
D. The Java method join() provides similar functionality as the WaitForSingleObject in Win32.

Example Question – Section 4.2

According to Amdahl's Law, what is the speedup gain for an application that is 60% parallel and we run it on a machine with 4 processing cores?

A. 1.82

B. .7

C. .55

D. 1.43

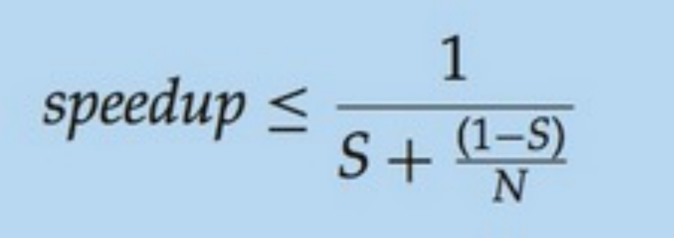
Example Question – Section 4.2

According to Amdahl's Law, what is the speedup gain for an application that is 60% parallel and we run it on a machine with 4 processing cores?

A. ***1.82***
B. .7
C. .55
D. 1.43

Amdahl's Law

If S is the portion of the application that must be performed serially on a system with N processing cores, the formula appears as follows:

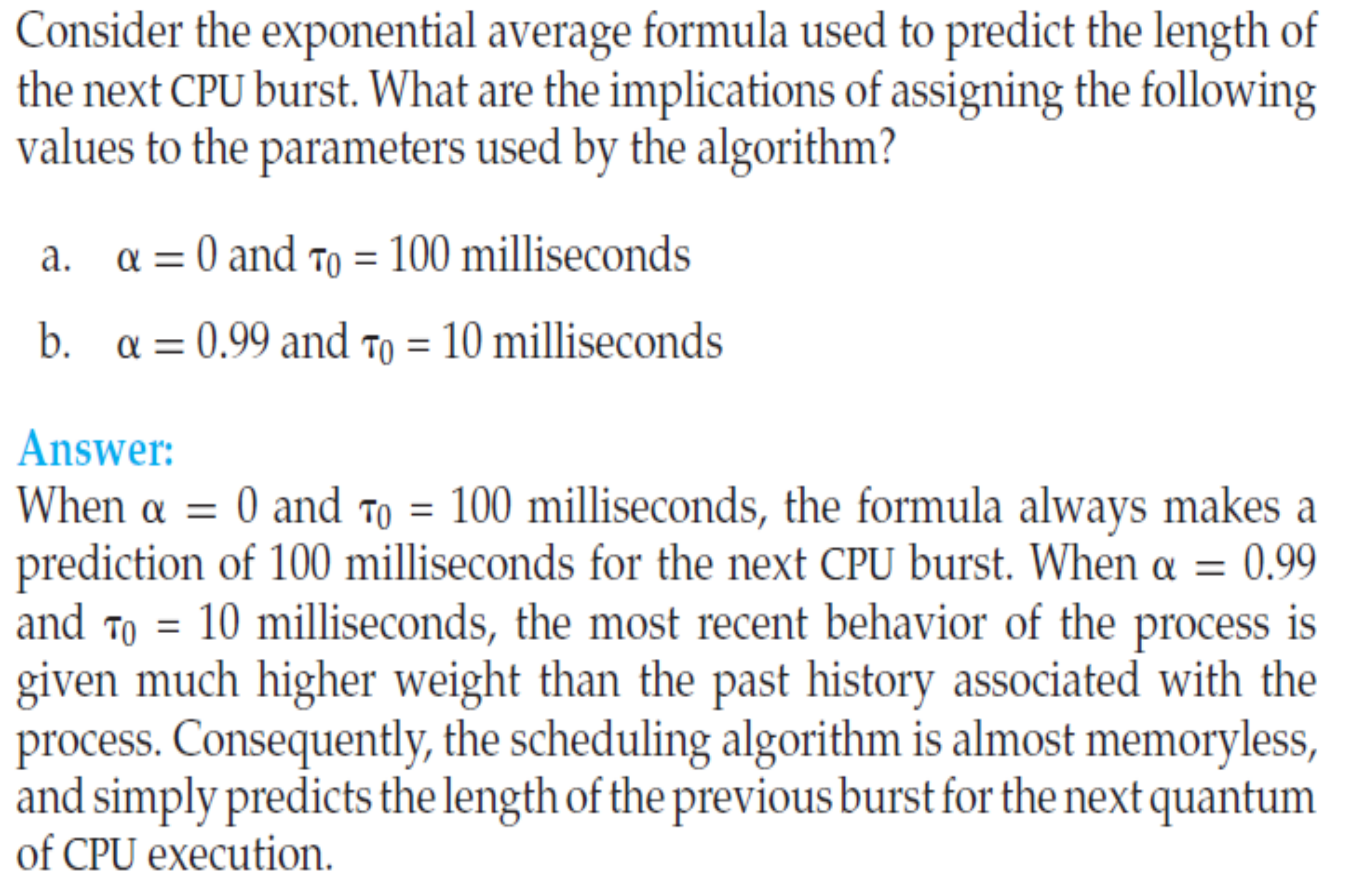


$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

60% parallel an h 4 processing cores – 40% is serial.

1.82 = 1/0.4 + (1-0.4)/4

Exercise 5.5



Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

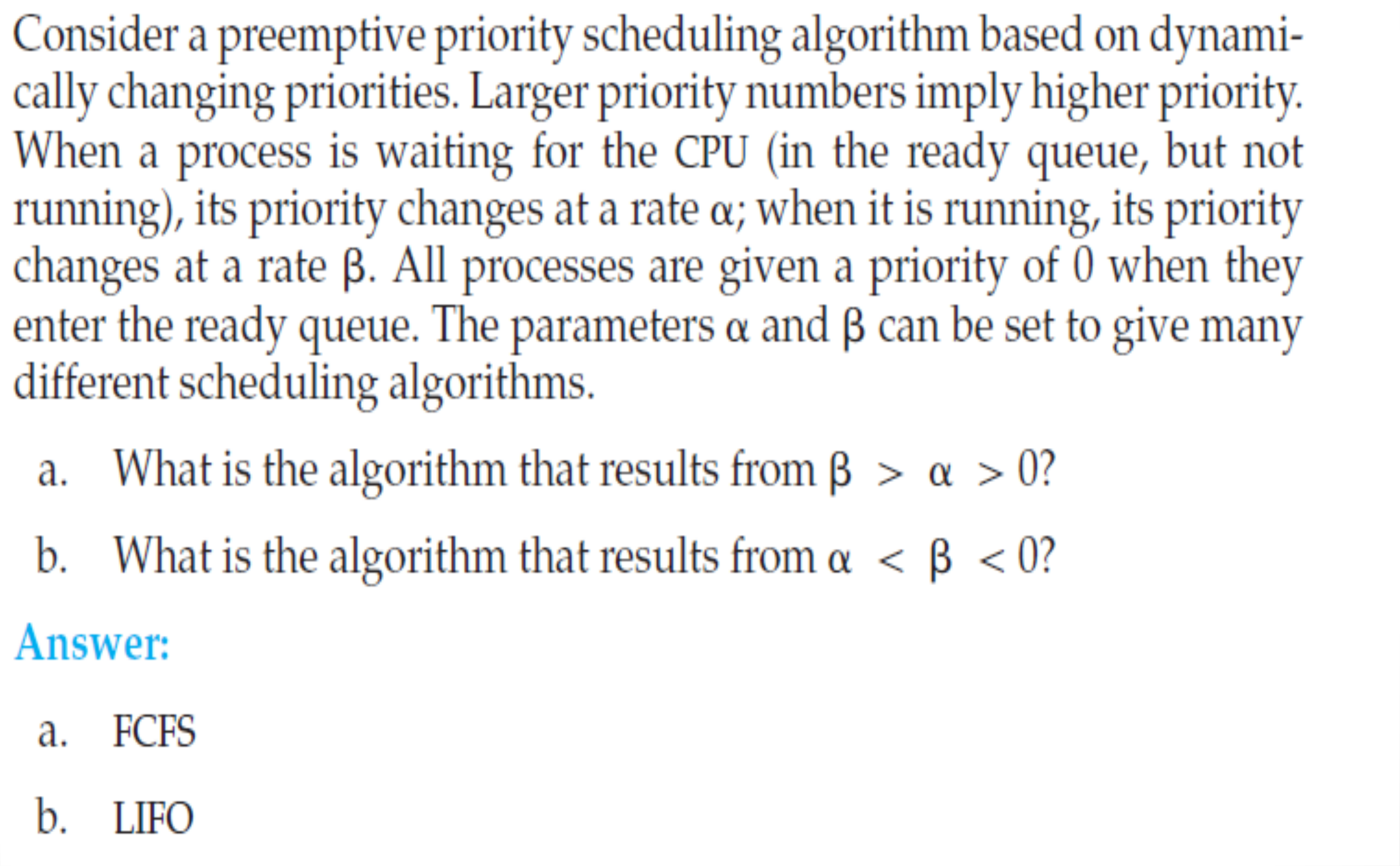
a. $\alpha = 0$ and $\tau_0 = 100$ milliseconds

b. $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds

Answer:

When $\alpha = 0$ and $\tau_0 = 100$ milliseconds, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds, the most recent behavior of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memoryless, and simply predicts the length of the previous burst for the next quantum of CPU execution.

Exercise 5.14



Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α; when it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

a. What is the algorithm that results from $\beta > \alpha > 0$?

b. What is the algorithm that results from $\alpha < \beta < 0$?

Answer:

a. FCFS

b. LIFO

Optional Exercises – 5.8

Which of the following scheduling algorithms could result in starvation?

a) First-come, first-served

b) Shortest job first

c) Round robin

d) Priority

Optional Exercises – 5.8

Which of the following scheduling algorithms could result in starvation?

a) First-come, first-served
b) Shortest job first
c) Round robin
d) Priority

Answer: Shortest job first and priority-based scheduling algorithms could result in starvation.

Optional Exercise - 5.16

Using the Windows scheduling algorithm, determine the numeric priority of each of the following threads.

a. A thread in the REALTIME PRIORITY CLASS with a relative priority of NORMAL
b. A thread in the ABOVE NORMAL PRIORITY CLASS with a relative priority of HIGHEST
c. A thread in the BELOW NORMAL PRIORITY CLASS with a relative priority of ABOVE NORMAL

Optional Exercise - 5.16

Using the Windows scheduling algorithm, determine the numeric priority of each of the following threads.

a. A thread in the REALTIME PRIORITY CLASS with a relative priority of NORMAL
b. A thread in the ABOVE NORMAL PRIORITY CLASS with a relative priority of HIGHEST
c. A thread in the BELOW NORMAL PRIORITY CLASS with a relative priority of ABOVE NORMAL

Answer:

a. 26

b. 8

c. 14

Optional exercise - 5.17

Assuming that no threads belong to the REALTIME PRIORITY CLASS and that none may be assigned a TIME CRITICAL priority, what combination of priority class and priority corresponds to the highest possible relative priority in Windows scheduling?

Optional exercise - 5.17

Assuming that no threads belong to the REALTIME PRIORITY CLASS and that none may be assigned a TIME CRITICAL priority, what combination of priority class and priority corresponds to the highest possible relative priority in Windows scheduling?

Answer:

HIGH priority class and HIGHEST priority within that class. (numeric priority of 15)

Optional exercise - 5.22

Consider two processes, P1 and P2, where p1 = 50, t1 = 25, p2 = 75, and t2 = 30.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 5.16–Figure 5.19.

b. Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

Optional exercise - 5.22

Consider two processes, P1 and P2, where p1 = 50, t1 = 25, p2 = 75, and t2 = 30.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 5.16–Figure 5.19.

b. Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

Answer:

Consider when P_1 is assigned a higher priority than P_2 with the rate monotonic scheduler. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. When P_1 is assigned a lower priority than P_2, then P_1 does not meet its deadline since it will not be scheduled in time.

Optional exercise 5.23

Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.

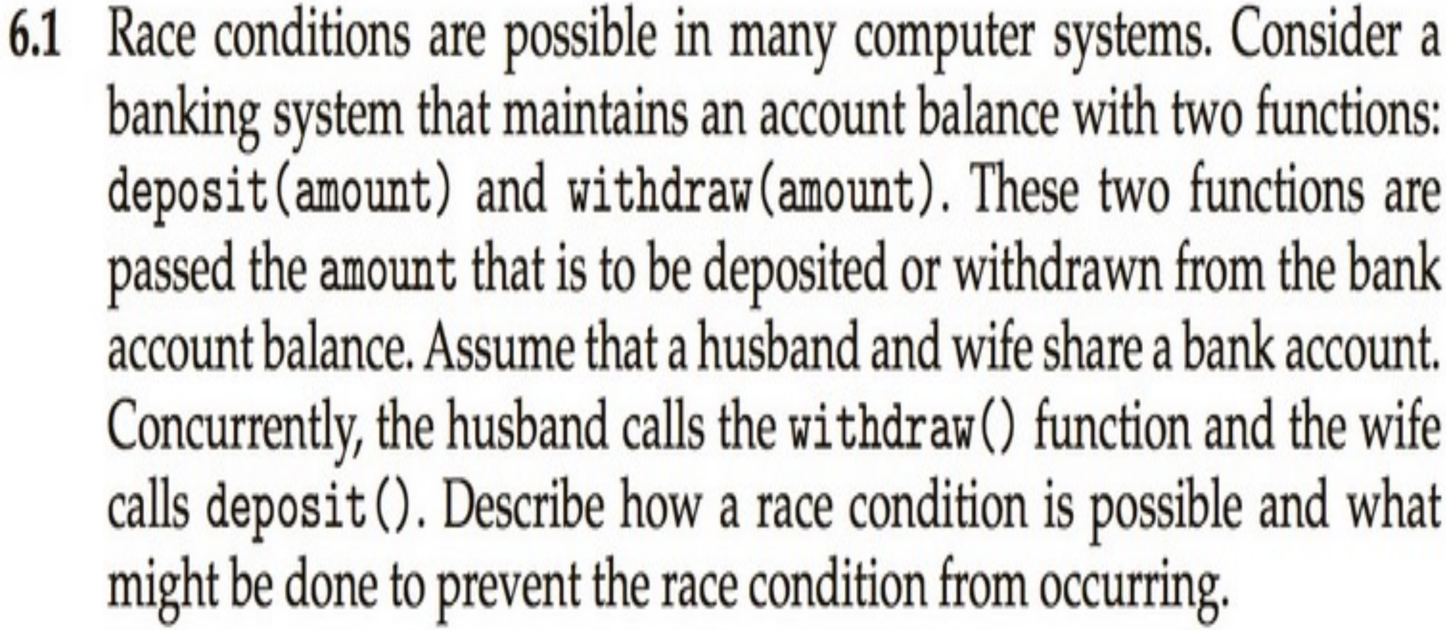
Optional exercise 5.23

Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.

Answer:

Following tasks: save the currently executing instruction, determine the type of interrupt, save the current process state, and then invoke the appropriate interrupt service routine. Dispatch latency is the cost associated with stopping one process and starting another. Both interrupt and dispatch latency needs to be minimized in order to ensure that real-time tasks receive immediate attention. Furthermore, sometimes interrupts are disabled when kernel data structures are being modified, so the interrupt does not get serviced immediately. For hard real-time systems, the time-period for which interrupts are disabled must be bounded in order to guarantee the desired quality of service.

Exercise 6.1



6.1 Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions: `deposit(amount)` and `withdraw(amount)`. These two functions are passed the `amount` that is to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account. Concurrently, the husband calls the `withdraw()` function and the wife calls `deposit()`. Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

6.1 Answer

Answer:

Assume the balance in the account is 250.00 and the husband calls withdraw(50) and the wife calls deposit(100). Obviously the correct value should be 300.00 Since these two transactions will be serialized, the local value of balance for the husband becomes 200.00, but before he can commit the transaction, the deposit(100) operation takes place

and updates the shared value of balance to 300.00 We then switch back to the husband and the value of the shared balance is set to 200.00 - obviously an incorrect value.

Exercise 7.8

Consider a system consisting of m resources of the same type being shared by n processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between one resource and m resources.

b. The sum of all maximum needs is less than m + n.

Answer 7.8

Answer:

Using the terminology of Section 7.6.2, we have:

a. _n

$i = 1$ $Maxi < m + n$

b. $Maxi \geq 1$ for all i

Proof: $Needi = Maxi - Alloca\ tioni$

If there exists a deadlock state then:

c. _n

$i = 1$ $Alloca\ tioni = m$

Use a. to get:_ $Needi$ + _ $Alloca\ tioni$ = _ $Maxi < m + n$

Use c. to get:_ $Needi + m < m + n$

Rewrite to get:_n

$i = 1$ $Needi < n$

This implies that there exists a process *Pi* such that $Needi = 0$. Since $Maxi \geq 1$ it follows that *Pi* has at least one resource that it can release. Hence the system cannot be in a deadlock state.

Week 4 Programming Project

Purpose

To assess your ability to:

Explain the critical section problem.

Present both software and hardware solutions of the critical section problem.

Examine several classical process-synchronization problems.

Describe several tools that are used to solve process-synchronization problems.

Define *deadlocks.*

Present a number of different methods for preventing or avoiding deadlocks in a computer system.

Action Items

Complete the following end-of-chapter exercise from your textbook:

- **Programming Project: The Sleeping Teaching Assistant**

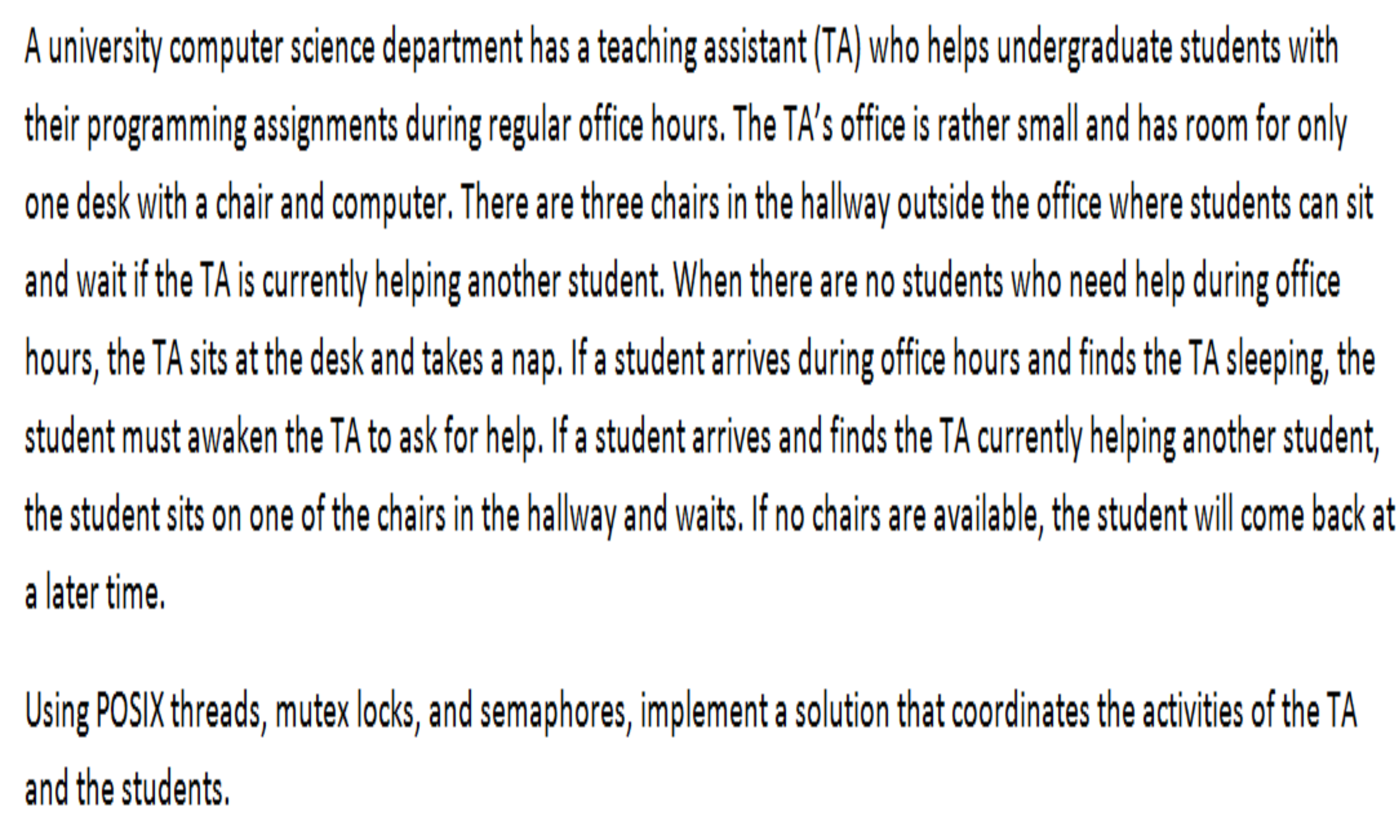
Submission

Complete and submit this assignment by Saturday 2-22-14 11:59pm

Grading Criteria

0 - 20 points

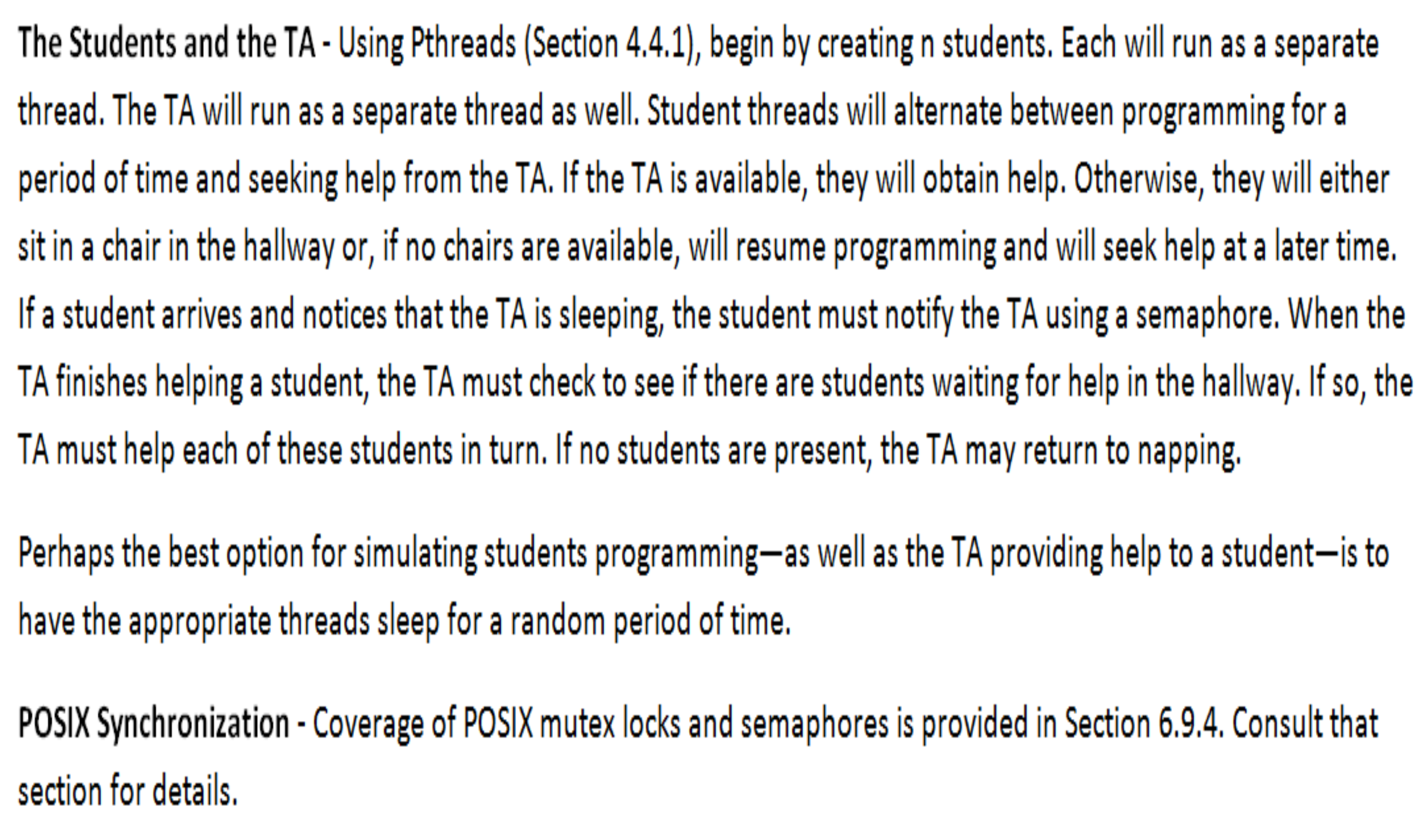
Project 1—the Sleeping Teaching Assistant



A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back at a later time.

Using POSIX threads, mutex locks, and semaphores, implement a solution that coordinates the activities of the TA and the students.

Project 1—the Sleeping Teaching Assistant



The Students and the TA - Using Pthreads (Section 4.4.1), begin by creating n students. Each will run as a separate thread. The TA will run as a separate thread as well. Student threads will alternate between programming for a period of time and seeking help from the TA. If the TA is available, they will obtain help. Otherwise, they will either sit in a chair in the hallway or, if no chairs are available, will resume programming and will seek help at a later time. If a student arrives and notices that the TA is sleeping, the student must notify the TA using a semaphore. When the TA finishes helping a student, the TA must check to see if there are students waiting for help in the hallway. If so, the TA must help each of these students in turn. If no students are present, the TA may return to napping.

Perhaps the best option for simulating students programming—as well as the TA providing help to a student—is to have the appropriate threads sleep for a random period of time.

POSIX Synchronization - Coverage of POSIX mutex locks and semaphores is provided in Section 6.9.4. Consult that section for details.

Optional Exercise 1

Assume a multithreaded application uses only reader—writer locks for synchronization. Applying the four necessary conditions for deadlock, is deadlock still possible if multiple reader—writer locks are used?

Optional Exercise 1

Assume a multithreaded application uses only reader—writer locks for synchronization. Applying the four necessary conditions for deadlock, is deadlock still possible if multiple reader—writer locks are used?

Answer: YES.

(1) Mutual exclusion is maintained, as they cannot be shared if there is a writer.

(2) (2) Hold-and-wait is possible, as a thread can hold one reader—writer lock while waiting to acquire another.

(3) (3) You cannot take a lock away, so no preemption is upheld.

(4) (4) A circular wait among all threads is possible.

Optional Exercise 2

Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.

Optional Question 2

Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.

Answer:

Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the

process disabling interrupts cannot guarantee mutually exclusive access to program state.

Optional exercise 3 – section 6.2

A race condition ____.

A. results when several threads try to access the same data concurrently

B. results when several threads try to access and modify the same data concurrently

C. will result only if the outcome of execution does not depend on the order in which instructions are executed

D. None of the above

Optional exercise 3 – section 6.2

A race condition ____.

A. results when several threads try to access the same data concurrently

B. results when several threads try to access and modify the same data concurrently

C. will result only if the outcome of execution does not depend on the order in which instructions are executed

D. None of the above

Optional exercise 4 – Section 6.4

An instruction that executes atomically ____.

A. must consist of only one machine instruction
B. executes as a single, uninterruptible unit
C. cannot be used to solve the critical section problem
D. All of the above

Optional exercise 4 – Section 6.4

An instruction that executes atomically ____.

A. must consist of only one machine instruction

B. executes as a single, uninterruptible unit

C. cannot be used to solve the critical section problem

D. All of the above

Optional exercise 5 – section 6.7.2

The first readers-writers problem ____.

A. requires that, once a writer is ready, that writer performs its write as soon as possible.

B. is not used to test synchronization primitives.

C. requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared database.

D. requires that no reader will be kept waiting unless a reader has already obtained permission to use the shared database.

Optional exercise 5 – section 6.7.2

The first readers-writers problem ____.

A. requires that, once a writer is ready, that writer performs its write as soon as possible.

B. is not used to test synchronization primitives.

C. requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared database.

D. requires that no reader will be kept waiting unless a reader has already obtained permission to use the shared database.

Optional exercise 6 – section 6.6

What is the correct order of operations for protecting a critical section using a binary semaphore?

A. release() followed by acquire()

B. acquire() followed by release()

C. wait() followed by signal()

D. signal() followed by wait()

Optional exercise 6 – section 6.6

What is the correct order of operations for protecting a critical section using a binary semaphore?

A. release() followed by acquire()

B. acquire() followed by release()

C. ***wait() followed by signal()***

D. signal() followed by wait()

Optional exercise 7 – section 7.1

A deadlocked state occurs whenever ____.

A. a process is waiting for I/O to a device that does not exist

B. the system has no available free resources

C. every process in a set is waiting for an event that can only be caused by another process in the set

D. a process is unable to release its request for a resource after use

Optional exercise 7 – section 7.1

A deadlocked state occurs whenever ____.

A. a process is waiting for I/O to a device that does not exist

B. the system has no available free resources

C. every process in a set is waiting for an event that can only be caused by another process in the set

D. a process is unable to release its request for a resource after use

Optional exercise 8 – section 7.5.1

Which of the following statements is true?

A. A safe state is a deadlocked state.

B. A safe state may lead to a deadlocked state.

C. An unsafe state is necessarily, and by definition, always a deadlocked state.

D. An unsafe state may lead to a deadlocked state.

Optional exercise 8 – section 7.5.1

Which of the following statements is true?

A. A safe state is a deadlocked state.

B. A safe state may lead to a deadlocked state.

C. An unsafe state is necessarily, and by definition, always a deadlocked state.

D. An unsafe state may lead to a deadlocked state.

Optional exercise 9 – section 7.1

Explain what has to happen for a set of processes to achieve a deadlocked state.

Optional exercise 9 – section 7.1

Explain what has to happen for a set of processes to achieve a deadlocked state.

Answer: For a set of processes to exist in a deadlocked state, every process in the set must be waiting for an event that can be caused only be another process in the set. Thus, the processes cannot ever exit this state without manual intervention.

Optional exercise 10 - section 7.2

True/False

The circular-wait condition for a deadlock implies the hold-and-wait condition.

Optional exercise 10 - section 7.2

True

The circular-wait condition for a deadlock implies the hold-and-wait condition.

Exercise 8.11

Program binaries in many systems are typically structured as follows. Code is stored starting with a small, fixed virtual address, such as 0. The code segment is followed by the data segment that is used for storing the program variables. When the program starts executing, the stack is allocated at the other end of the virtual address space and is allowed to grow toward lower virtual addresses. What is the significance of this structure for the following schemes?

a. Contiguous memory allocation

b. Pure segmentation

c. Pure paging

Exercise 8.11 solution

1) Contiguous-memory allocation requires the operating system to allocate the entire extent of the virtual address space to the program when it starts executing. This could be much larger than the actual memory requirements of the process.

2) Pure segmentation gives the operating system flexibility to assign a small extent to each segment at program startup time and extend the segment if required.

3) Pure paging does not require the operating system to allocate the maximum extent of the virtual address space to a process at startup time, but it still requires the operating system to allocate a large page table spanning all of the program's virtual address space. When a program needs to extend the stack or the heap, it needs to allocate a new page but the corresponding page table entry is preallocated.

Exercise 8.16

Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512-MB of physical memory. How many entries are there in each of the following?

a. A conventional single-level page table
b. An inverted page table

Exercise 8.16 solution (a)

a. 2^{20} entries - Size of logical address space = 2^{m} = of pages × page size

2^{32} = of pages × 2^{12} of pages =2^{32} / 2^{12}= 2^{20} pages

Exercise 8.16 solution (B)

Single Level

512MB /4K = 128K entries.

512 MB physical memory

– 2^{29} bytes/2^{12} byte pages = 2^{17} physical pages

Multi Level

Assume page table entry is 4 bytes

– 4 KB page / 4 bytes per page table entry =

1024 entries

Exercise 8.25

Assume that a system has a 32-bit virtual address with a 4-KB page size. Write a program that is passed a virtual address (in decimal) and have it output the page number and offset for the given address.

As an example, your program would run as follows

Input:

```
        19986
```

Output:

```
        The address 19986 contains: page number = 4 offset =
3602
```

Exercise 8.25 solution

Program that masks page number and offset from an unsigned 32-bit address. The size of a page is 4 KB (12 bits)

A memory reference appears as:

```
 |------------|-----|
 31          12 11  0
```

Exercise 8.25 solution

```
#include <stdio.h>
#include <unistd.h>
#define PAGE_NUMBER_MASK 0xFFFFF000
#define OFFSET_MASK 0x00000FFF
int main(int argc, char *argv[]) {
        if (argc != 2) {
                fprintf(stderr,"Usage: ./a.out <virtual address>\n");
                return -1;}
        int page_num, offset;
        unsigned int reference;
        reference = (unsigned int)atoi(argv[1]);
        printf("The address %d contains:\n",reference);
        // first mask the page number
        page_num = (reference & PAGE_NUMBER_MASK) >> 12;
        offset = reference & OFFSET_MASK;
        printf("page number = %d\n",page_num);
        printf("offset = %d\n",offset);
        return 0;}
```

Optional Exercise 1 – Section 8.1.3

The mapping of a logical address to a physical address is done in hardware by the _________.

A. ***memory-management-unit (MMU)***

B. memory address register

C. relocation register

D. dynamic loading register

Optional Exercise 2 - Section 8.1.5

In a dynamically linked library, ____.

A. loading is postponed until execution time

B. system language libraries are treated like any other object module

C. more disk space is used than in a statically linked library

D. a stub is included in the image for each library-routine reference

Optional Exercise 3 - Section 8.2

The ______ binding scheme facilitates swapping.

A. interrupt time

B. load time

C. assembly time

D. execution time

Optional Exercise 4- Section 8.3.2

_____ is the dynamic storage-allocation algorithm which results in the smallest leftover hole in memory.

A. First fit

B. Best fit

C. Worst fit

D. None of the above

Optional Exercise 5 - Section 8.3.2

_____ is the dynamic storage-allocation algorithm which results in the largest leftover hole in memory.

A. First fit

B. Best fit

C. ***Worst fit***

D. None of the above

Optional Exercise 6 - Section 8.5

Consider a logical address with 18 bits used to represent an entry in a conventional page table. How many entries are in the conventional page table?

A. ***262144***
B. 1024
C. 1048576
D. 18

Optional Exercise 7 - Section 8.5

Given the logical address 0xAEF9 (in hexadecimal) with a page size of 256 bytes, what is the page number?

A. ***0xAE***

B. 0xF9

C. 0xA

D. 0x00F9

Optional Exercise 8 - Section 8.8

True/False

The ARM architecture uses both single-level and two-level paging.

Optional Exercise 9 - Section 8.6.3

True/**False**

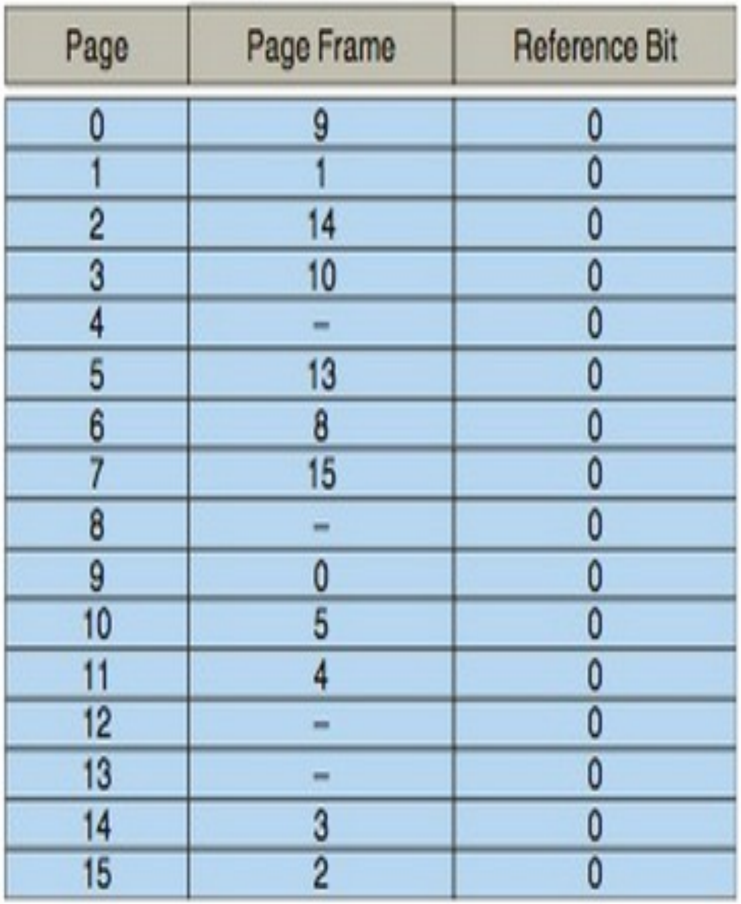
Inverted page tables require each process to have its own page table.

Optional Exercise 10 - Section 8.5

True/**False**

A 32-bit logical address with 8 KB page size will have 1,000,000 entries in a conventional page table.

Exercise 9.9



Page	Page Frame	Reference Bit
0	9	0
1	1	0
2	14	0
3	10	0
4	–	0
5	13	0
6	8	0
7	15	0
8	–	0
9	0	0
10	5	0
11	4	0
12	–	0
13	–	0
14	3	0
15	2	0

Figure 9.31 Page table for Exercise 9.9.

- The page table shown in Figure 9.31 is for a system with 16-bit virtual and physical addresses and with 4,096-byte pages.
- The reference bit is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit.
- A dash for a page frame indicates the page is not in memory.
- The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.

Exercise 9.9

- Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses. You may provide answers in either hexadecimal or decimal. Also set the reference bit for the appropriate entry in the page table.
  - 0xE12C • 0x3A9D • 0xA9D9 • 0x7001 • 0xACA1
- Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that result in a page fault.
- From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?
-

Exercise 9.9 solution

Answer:

- 0xE12C→0x312C
- 0x3A9D→0xAA9D
- 0xA9D9→0x59D9
- 0x7001→0xF001
- 0xACA1→0x5CA1

- The only choices are pages 4, 8, 12, and 13. Thus, example addresses

include anything that begins with the hexadecimal sequence

0x4..., 0x8..., 0xC..., and 0xD....

- Any page table entries that have a reference bit of zero. This includes

the following frames {9, 1, 14, 13, 8, 0, 4}

Exercise 9.12

- Discuss situations in which the most frequently used (MFU) page- replacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm.
- Also discuss under what circumstances the opposite holds.

Exercise 9.12 solution

Answer:

Consider the sequence in a system that holds four pages in memory: 1 2 3 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence “1 2 3 4 4 4 5 1,” the LRU algorithm makes the right decision.

Programming Project: Exercise 9.26

- Write a program that implements the FIFO, LRU, and optimal page- replacement algorithms presented in this chapter.
- First, generate a random page-reference string where page numbers range from 0 to 9.
- Apply the random page-reference string to each algorithm and record the number of page faults incurred by each algorithm.
- Implement the replacement algorithms so that the number of page frames can vary from 1 to 7. Assume that demand paging is used.

Programming Project: Exercise 9.26

- First, generate a random page reference string where page numbers range from 0 to 9.
- Apply the random page-reference string to each algorithm, and record the number of page faults incurred by each algorithm.
- Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used.
- Your algorithms will be based on the abstract class depicted in Figure 9.2.

Replacement algorithm

```
public abstract class ReplacementAlgorithm
{
// the number of page faults protected int pageFaultCount;
// the number of physical page frame protected int pageFrameCount;
// pageFrameCount - the number of physical page frames public
ReplacementAlgorithm(int pageFrameCount) {
if (pageFrameCount < 0)
throw new IllegalArgumentException();
this.pageFrameCount = pageFrameCount; pageFaultCount = 0; }
// return - the number of page faults that occurred. public int
getPageFaultCount() {
return pageFaultCount;
}
// int pageNumber - the page number to be inserted
public abstract void insert(int pageNumber); }
```

Programming Project: Exercise 9.26

Design and implement two classes—LRU and FIFO—that extend **ReplacementAlgorithm**.

Each of these classes will implement the insert() method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm.

PageGenerator—a class that generates page-reference strings with page numbers ranging from 0 to 9. The size of the reference string is passed to the PageGenerator constructor.

PageGeneratorobjectisconstructed,thegetReferenceString() method returns the reference string as an array of integers.

Test—used to test your FIFO and LRU implementations of the ReplacementAlgorithm abstract class. Test is invoked as

java Test <reference string #> <# of page frames>

Resources

http://codex.cs.yale.edu/avi/os-book/OSE2/java-dir/9.pdf

http://cprogrammingguide.wordpress.com/2012/07/10/simulation-of-page-replacement-algorithms-fifo-lru-optimal/

http://codearea.in/page-replacement-algorithms-fifo-lru-optimal/

Optional Exercise 1- Section 9.1

- Which of the following is a benefit of allowing a program that is only partially in memory to execute?
  A. Programs can be written to use more memory than is available in physical memory.
  B. CPU utilization and throughput is increased.
  C. Less I/O is needed to load or swap each user program into memory.
  D. All of the above

Optional Exercise 2 - Section 9.4.2

Suppose we have the following page accesses: 1 2 3 4 2 3 4 1 2 1 1 3 1 4 and that there are three frames within our system. Using the FIFO replacement algorithm, what is the number of page faults for the given reference string?

A. 14

B. 8

C. 13

D. 10

Optional Exercise 3 – Section 9.4.2

Suppose we have the following page accesses: 1 2 3 4 2 3 4 1 2 1 1 3 1 4 and that there are three frames within our system. Using the FIFO replacement algorithm, what will be the final configuration of the three frames following the execution of the given reference string?

A. 4, 1, 3

B. 3, 1, 4

C. 4, 2, 3

D. 3, 4, 2

Optional Exercise 4 – Section 9.4.4

Given the reference string of page accesses: 1 2 3 4 2 3 4 1 2 1 1 3 1 4 and a system with three page frames, what is the final configuration of the three frames after the LRU algorithm is applied?

A. 1, 3, 4

B. 3, 1, 4

C. 4, 1, 2

D. 1, 2, 3

Optional Exercise 5 – section 9.4

- _____ is the algorithm implemented on most systems.
  A. FIFO
  B. Least frequently used
  C. Most frequently used
  D. LRU

Optional Exercise 6 – section 9.6

- _____ occurs when a process spends more time paging than executing.

  A. ***Thrashing***

  B. Memory-mapping

  C. Demand paging

  D. Swapping

Optional Exercise 7 – section 9.7

______ allows a portion of a virtual address space to be logically associated with a file.

A. ***Memory-mapping***

B. Shared memory

C. Slab allocation

D. Locality of reference

Optional Exercise 8 – Section 9.1

True/**False**

In general, virtual memory decreases the degree of multiprogramming in a system.

Optional Exercise 9 – section 9.2.2

How is the effective access time computed for a demand-paged memory system?

Answer: In order to compute the effective access time, it is necessary to know the average memory access time of the system, the probability of a page fault, and the time necessary to service a page fault. The effective access time can then be computed using the formula:

effective access time = (1 – probability of page fault) * memory access time + probability of page fault * page fault time.

Optional Exercise 10 – section 9.8

Which of the following statements is false with regard to allocating kernel memory?

A. Slab allocation does not suffer from fragmentation.

B. Adjacent segments can be combined into one larger segment with the buddy system.

C. Because the kernel requests memory of varying sizes, some of which may be quite small, the system does not have to be concerned about wasting memory.

D. The slab allocator allows memory requests to be satisfied very quickly.

10.6

If the operating system knew that a certain application was going to access file data in a sequential manner, how could it exploit this information to improve performance?

Answer:

When a block is accessed, the file system could prefetch the subsequent blocks in anticipation of future requests to these blocks. This prefetching

Optimization would reduce the waiting time experienced by the process for future requests.

10.7

Give an example of an application that could benefit from operating-system support for random access to indexed files.

Answer:

An application that maintains a database of entries could benefit from such support. For instance, if a program is maintaining a student database, then accesses to the database cannot be modeled by any predetermined access pattern. The access to records are random and locating the records would be more efficient if the operating system were to provide some form of tree-based index.

11.7

Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:

a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)

b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

11.7 - Solution

Let *Z* be the starting file address (block number).

- **Contiguous**. Divide the logical address by 512 with *X* and *Y* the resulting quotient and remainder respectively.

  a. Add *X* to *Z* to obtain the physical block number. *Y* is the displacement into that block.

  b. 1

- **Linked**. Divide the logical physical address by 511 with *X* and *Y* the resulting quotient and remainder respectively.

  a. Chase down the linked list (getting *X* + 1 blocks). *Y* + 1 is the displacement into the last physical block.

  b. 4

- **Indexed**. Divide the logical address by 512 with *X* and *Y* the resulting quotient and remainder respectively.

  a. Get the index block into memory. Physical block address is contained in the index block at location *X*. *Y* is the displacement into the desired physical block.

  b. 2

Optional exercise 1 – Section 10.1

Suppose that the operating system uses two internal tables to keep track of open files. Process A has two files open and process B has three files open. Two files are shared between the two processes. How many entries are in the per-process table of process A, the per-process table of process B, and the system-wide tables, respectively?

A. 5, 5, 5

B. 2, 3, 3

C. 2, 3, 5

D. 2, 3, 1

Optional exercise 2 – Section 10.1

An exclusive lock ____.

A. ***behaves like a writer lock***

B. ensures that a file can have only a single concurrent shared lock

C. behaves like a reader lock

D. will prevent all other processes from accessing the locked file

Optional exercise 3 – Section 10.2.1

The simplest file access method is ____

A. ***sequential access.***

B. logical access.

C. relative access.

D. direct access.

Optional exercise 4 – Section 10.1.2

If you were creating an operating system to handle files, what would be the six basic file operations that you should implement?

Answer: The six basic file operations include: creating a file, writing a file, reading a file, repositioning within a file, deleting a file, and truncating a file. These operations comprise the minimal set of required file operations.

Optional exercise 5 – Section 10.3.5

Distinguish between an absolute path name and a relative path name.

Answer: An absolute path name begins at the root and follows a path of directories down to the specified file, giving the directory names on the path. An example of an absolute path name is

/home/osc/chap10/file.txt. A relative path name defines a path from the current directory. If the current directory is /home/osc/, then the relative path name of chap10/file.txt refers to the same file as in the example of the absolute path name.

Optional exercise 6 – Section 11.1

Transfers between memory and disk are performed a ____.

A. byte at a time

B. file at a time

C. ***block at a time***

D. sector at a time

Optional exercise 7 – section 11.1

Order the following file system layers in order of lowest level to highest level.

[1] I/O control
[2] logical file system
[3] basic file system
[4] file-organization module
[5] devices

A. 1, 3, 5, 4, 2
B. 5, 1, 3, 2, 4
C. 1, 5, 3, 4, 2
D. 5, 1, 3, 4, 2

Optional exercise 9 – Section 11.1

______ includes all of the file system structure, minus the actual contents of files.

A. ***Metadata***
B. Logical file system
C. Basic file system
D. File-organization module

Optional exercise 8 – Section 11.3.2

How is a hash table superior to a simple linear list structure? What issue must be handled by a hash table implementation?

Answer: A hash table implementation uses a linear list to store directory entries. However, a hash data structure is also used in order to speed up the search process. The hash data structure allows the file name to be used to help compute the file's location within the linear list. Collisions, which occur when multiple files map to the same location, must be handled by this implementation.

Optional exercise 10 – Section 11.4.2

How many disk accesses are necessary for direct access to byte 20680 using linked allocation and assuming each disk block is 4 KB in size?

A. 1

B. 6

C. 7

D. 5