1.1 What are the three main purposes of an operating system?

2.1 What is the purpose of system calls?

2.2 What are the five major activities of an operating system with regard to process management?

2.3 What are the three major activities of an operating system with regard to memory management?

2.4 What are the three major activities of an operating system with regard to secondary-storage management?

2.5 What is the purpose of the command interpreter? Why is it usually separate from the kernel?

3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?
#include <stdio.h>
#include <unistd.h>
int main()
{
    /* fork a child process */
    fork();
    /* fork another child process */
    fork();
    /* and fork another */
    fork();
    return 0;
}

3.5 When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process?
   a. Stack
   b. Heap
   c. Shared memory segments

3.8 Describe the differences among short-term, medium-term, and long-term scheduling.

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

3.12 Including the initial parent process, how many processes are created by the program?
#include <stdio.h>
#include <unistd.h>
int main()
{
    int i;
    for (i = 0; i < 4; i++)
        fork();
    return 0;
}
3.13 Explain the circumstances under which the line of code marked `printf("LINE J")` will be reached.

```c
#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 */
        wait(NULL);
        printf("Child Complete");
    }
    return 0;
}
```

3.14 Using the program in Figure 3.34, identify the values of `pid` at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

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

int main()
{
    pid_t pid, pid1;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    } else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d",pid); /* A */
        printf("child: pid1 = %d",pid1); /* B */
    } else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d",pid); /* C */
        printf("parent: pid1 = %d",pid1); /* D */
        wait(NULL);
    }
    return 0;
}
```
3.17 Using the program shown, explain what the output will be at lines X and Y.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
    int i;
    pid_t pid;
    pid = fork();
    if (pid == 0) /
        for (i = 0; i < SIZE; i++) /
            printf("CHILD: %d ",nums[i]); /* LINE X */
    )
    else if (pid > 0) /
        wait(NULL);
    for (i = 0; i < SIZE; i++) /
        printf("PARENT: %d ",nums[i]); /* LINE Y */
    )
    return 0;
}
```

3.18 What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.
   a. Synchronous and asynchronous communication
   b. Automatic and explicit buffering
   c. Send by copy and send by reference
   d. Fixed-sized and variable-sized messages

4.6 Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

4.7 Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

4.8 Which of the following components of program state are shared across threads in a multithreaded process?
   a. Register values
   b. Heap memory
   c. Global variables
   d. Stack memory

4.9 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain.

4.11 Is it possible to have concurrency but not parallelism? Explain.

4.16 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, other operating systems, such as Windows, treat processes and threads differently. Typically, such systems use a notation in which 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.
5.1 In Section 5.4, we mentioned that disabling interrupts frequently can affect the system’s clock. Explain why this can occur and how such effects can be minimized.

5.3 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

5.5 Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

5.6 Illustrate how a binary semaphore can be used to implement mutual exclusion among $n$ processes.

5.7 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.

5.14 Describe how the compare and swap() instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement.

5.16 The implementation of mutex locks provided in Section 5.5 suffers from busy waiting. Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available.

5.21 Servers can be designed to limit the number of open connections. For example, a server may wish to have only $N$ socket connections at any point in time. As soon as $N$ connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

5.29 How does the signal() operation associated with monitors differ from the corresponding operation defined for semaphores?

6.1 A CPU-scheduling algorithm determines an order for the execution of its scheduled processes. Given $n$ processes to be scheduled on one processor, how many different schedules are possible? Give a formula in terms of $n$.

6.2 Explain the difference between preemptive and nonpreemptive scheduling.

6.3 Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>8</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0.4</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1.0</td>
<td>1</td>
</tr>
</tbody>
</table>

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process $P_1$ at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes $P_1$ and $P_2$ are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling.

6.4 What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?
6.5 Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithm for each queue, the criteria used to move processes between queues, and so on. These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?

a. Priority and SJF
b. Multilevel feedback queues and FCFS
c. Priority and FCFS
d. RR and SJ

6.6 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

6.9 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (recent CPU usage / 2) + base

where

base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage is 40 for process P1, 18 for process P2, and 10 for process P3. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

6.10 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

6.16 Consider the following set of processes, with the length of the CPU burst given in milliseconds:

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?
c. What is the waiting time of each process for each of these scheduling algorithms?
d. Which of the algorithms results in the minimum average waiting time (over all processes)?

6.17 The following processes are being scheduled using a preemptive, round robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as P_idle). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

<table>
<thead>
<tr>
<th>Thread</th>
<th>Priority</th>
<th>Burst</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>40</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>30</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>P3</td>
<td>30</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>P4</td>
<td>35</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>P6</td>
<td>10</td>
<td>10</td>
<td>105</td>
</tr>
</tbody>
</table>

a. Show the scheduling order of the processes using a Gantt chart.
b. What is the turnaround time for each process?
c. What is the waiting time for each process?
d. What is the CPU utilization rate?

6.19 Which of the following scheduling algorithms could result in starvation?
   a. First-come, first-served
   b. Shortest job first
   c. Round robin
   d. Priority

6.20 Consider a variant of the RR scheduling algorithm in which the entries in the ready queue are pointers to the PCBs.
   a. What would be the effect of putting two pointers to the same process in the ready queue?
   b. What would be two major advantages and two disadvantages of this scheme?
   c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

6.21 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks. Describe the CPU utilization for a round-robin scheduler when:
   a. The time quantum is 1 millisecond
   b. The time quantum is 10 milliseconds

6.24 Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes:
   a. FCFS
   b. RR
   c. Multilevel feedback queues