Chapter 4

1. In general, the PCB contains all the information that the operating system chooses to maintain about a process to describe its attributes and keep track of its status. For example: process name, process priority, process state, process context, memory allocation, resource allocation, accounting information and information on interrelationships between this process and other processes.

2. In general all information that the process maintained in the CPU registers, status words, address space, stacks and current location in the program must be saved when a process is interrupted. When the process resumes everything in its processing environment must be exactly as it was when the process stopped executing.

3. SJN would process them in 2, 3, 5, 6, 10 order.

4. It will be interrupted two times and will finish its execution in the third queue.

   Queue 1  5ms
   Queue 2  10ms
   Queue 3  20ms
   Queue 4  40ms
   etc.

7 a. FCFS

   J1  J2  J3  J4  J5
   0--12--15--16--21

b. SJN

   J1  J4  J2  J3  J5
   0--10--11--13--16--21

c. SRT

   J1  J2  J4  J3  J5
   0--1--3--4--7--12--21

d. Round Robin

   J1  J2  J3  J4  J5  J1  J3  J5  J1  J5  J1  J1
   0--2--4--6--7--9--11--12--14--16--17--19--21

8 a. FCFS

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>CPU Finish Cycle Time</th>
<th>Wait Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>
### 8 b. SJN

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>CPU Cycle Time</th>
<th>Finish Time</th>
<th>Wait Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>16</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>21</td>
<td>12</td>
<td>17</td>
</tr>
</tbody>
</table>

### c. SRT

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>CPU Cycle Time</th>
<th>Finish Time</th>
<th>Wait Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>21</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>12</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

### d. Round Robin

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>CPU Cycle Time</th>
<th>Finish Time</th>
<th>Wait Time</th>
<th>Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>21</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>17</td>
<td>8</td>
<td>13</td>
</tr>
</tbody>
</table>

### 9 a. FCFS

- Total Wait-time = 43
- Average Wait-time = 43/5 = 8.6
- Total Turn-time = 64
- Average Turn-time = 64/5 = 12.8

### b. SJN

- Total Wait-time = 40
- Average Wait-time = 40/5 = 8
- Total Turn-time = 61
- Average Turn-time = 61/5 = 12.2

### c. SRT

- Total Wait-time = 16
- Average Wait-time = 16/5 = 3.2
- Total Turn-time = 37
- Average Turn-time = 37/5 = 7.4

### d. Round Robin

- Total Wait-time = 30
- Average Wait-time = 30/5 = 6
- Total Turn-time = 51
- Average Turn-time = 51/5 = 10.2

In this example SRT gives the best results because it has the lowest average waiting and turnaround times.
Chapter 5

1. In deadlock at least two processes are waiting for resources that are held by each other and are therefore unavailable. Neither process can continue without the requested resource and the resources already allocated to them cannot be released therefore impacting on the execution of other processes. A deadlock, as the name implies, locks up a subset of processes on a system and it may, in extreme cases, lock up the entire system.

Starvation occurs when the execution of a process is indefinitely postponed or delayed. In this case the system is not affected, but one or more processes are.

Race occurs when the scheduling of two processes is so critical that variations in the order of their scheduling result in different computations. In this case neither the system nor the processes are affected, but the result (or outcome) is affected by the order in which the processes are executed.

2. Any response that makes sense is acceptable. Deadlock: consider a savings and loan institution where more savers want to withdraw money from their accounts than the institution has available. The institution cannot force the borrowers to return the money they borrowed. This is a deadlock where the borrowers hold the resource (money) that the institution needs to satisfy the request of the savers.

Starvation: consider a very timid customer waiting to be served at a crowded counter where no service policy has been set up. The more aggressive customers capture the attention of the server and postpone service to the timid customer.

Race: consider the case where you write a check knowing that you will be overdrawn if your paycheck does not get deposited (and cleared) before the check you just wrote is cashed.

6 a. No. Not yet because one of the R2 can be allocated to either P1 or P2.
   b. No, not yet.

   c. (Diagram)

   d. (Diagram)

   e. (1) If R2 is given to P2 then the system is in an unstable state. There is no guarantee that P1 will release one of R1 needed by P2 to finish up.

   (2) If R2 is given to P1 then it will finish execution and release its resources so that P2 will also finish. The system will remain stable.
15. CPU. To prevent deadlock we can remove the “no preemption” condition. This is practical, relatively easy to implement and is present in systems that support preemptive CPU policies, such as Round Robin.

Memory. To prevent deadlock we can remove the “no preemption” condition. This is true of all systems using the virtual memory approach. If a process needs to expand its working set, then another process’ working set may be swapped out to secondary storage freeing up some page frames.

Storage devices. To prevent deadlock we can remove the “mutual exclusion” condition from disk drives, but not from tape drives. It may be that for these devices prevention of deadlock is not possible, but avoidance is possible by using the Banker’s Algorithm.

Files. To prevent deadlock we can remove the “hold and wait” (Resource Holding) condition by requiring preallocation. A process must request its file resources initially and cannot proceed until they have been allocated, although this would entail a substantial cost in inefficiency of resource utilization. It may be better to implement a “lock” which can keep all other processes out until the first process is finished. In this case the other processes are not deadlocked, just temporarily blocked.
Chapter 6

1. One of the goals of multiprocessing systems is to increase throughput, another is to increase the operating system's efficiency by specialization, a third is to have system reliability, a fourth is to provide services to different users (interactive and batch).

4. When the programmer specifically indicates, by enclosing sections of code with COBEGIN/COEND constructs, then we have explicit parallelism. It is a time consuming operation that places certain responsibilities with the programmer who may erroneously indicate that certain operations can be performed in parallel when in reality they can not. On the other hand, the programmer may miss situations in which parallelism could be exploited.

5. Implicit parallelism is intrinsic to the algorithm but is not explicitly stated by the programmer. Compilers, operating systems and computer hardware will need detection mechanisms to take advantage of implicit parallelism. In terms of speed of execution and correctness of code, as well as program maintenance and modification, this is a better alternative than explicit parallelism.

7. This problem is based on the readers and writers algorithm or, better yet, the competing writers. The simulation has to consider queue management and competing car processes which can be implemented using P and V operations. The processes enforce mutual exclusion of the critical region, the intersection, by using a binary semaphore “light” which is initialized to 1.

The following modules define the processes for queue management and the P and V operations.

```plaintext
binary semaphore light: go = 1, no go = 0

P(light)
    IF light > 0
        THEN set light to 0
        ELSE put-in-queue(Q, pointer, car-count)
    END-IF

V(light)
    IF car-count > 0
        THEN remove-from-queue(Q, pointer, car-count)
        ELSE set light to 1
    END-IF

procedure carA(light, Q, car-count, pointer)
    begin
        P(light)
        cross intersection
        V(light)
    end

procedure carB(light, Q, car-count, pointer)
    begin
        P(light)
        cross intersection
        V(light)
    end
```

The main program would then be similar to the following:

```
set light to 1
COBEGIN
    repeat carA(light, Q1, car-count1, j)
    repeat carB(light, Q2, car-count2, k)
COEND
```
10a.

    DO I = 1, 12
        READ *, X
        IF (X .EQ. 0) THEN
            Y(I) = 0
        ELSE
            Y(I) = 10
        END IF
    END DO

10b.

    I = 1
    DO J = 1, 3
        READ *, X(I), X(I+1), X(I+2), X(I+3)
        CODEGN
            IF (X(I) .EQ. 0) THEN
                Y(I) = 0
            ELSE
                Y(I) = 10
            END IF
            IF (X(I+1) .EQ. 0) THEN
                Y(I+1) = 0
            ELSE
                Y(I+1) = 10
            END IF
            IF (X(I+2) .EQ. 0) THEN
                Y(I+2) = 0
            ELSE
                Y(I+2) = 10
            END IF
            IF (X(I+3) .EQ. 0) THEN
                Y(I+3) = 0
            ELSE
                Y(I+3) = 10
            END IF
        ENDCOND
    I = I + 4
    END DO

10c. In this example it will take 12 time units to complete part (a) but only 3 time units to complete part (b) because the number of iterations through the loop has been reduced from 12 to 3. It may be possible to get the real time, or the number of machine cycles, that it takes for a specific computer to perform an IF-THEN-ELSE instruction and then use this information to compute and compare part (a) and (b)'s execution speeds.