Part 1: (60 points - 4 points for each problem)

1. An operating system is usually not viewed as a:
   (D) None of the above
   (A) Extended machine (B) Resource manager (C) Virtual machine

2. In an operating system a utility which lets the users issue and execute commands is called:
   (B) None of the above
   (A) Terminal Handler (B) Command Interpreter (C) Kernel

3. Which contains only essential functions of the operating system?
   (B) None of the above
   (A) Minikernel (B) Microkernel (C) Macrokernel

4. Which statement about user-level threads and kernel threads is correct?
   (C) User-level thread scheduling is faster than kernel thread scheduling.
   (A) User-level threads can write into each other’s memory space whereas kernel threads are independent of each other.
   (B) User level threads require memory management where kernel threads do not.
   (D) None of the above

5. The purpose of multiprogramming is to:
   (A) Utilize CPU better
   (B) Make the computer hardware more user friendly
   (C) Make it easy for the users to run programs
   (D) To get the most out of slow input-output devices

6. CPU burst distribution is generally characterized as
   (D) Exponential or hyper-exponential
   (A) Constant (B) Linear (C) Polynomial

7. Which is not a CPU scheduling criterion?
   (B) Burst time (C) Throughput
   (A) CPU utilization (D) Response time

8. The scheduler that brings processes into memory and swaps them out on disk as needed is referred to as:
   (B) Admission scheduler (C) Memory scheduler
   (A) CPU scheduler (D) None of the above

9. Which is a preemptive scheduling?
   (A) RR (B) FCFS (C) SJF
   (D) None of the above

10. If the time quantum is very large, a RR (Round-Robin) scheduling is the same as:
    (D) multilevel queue
        (A) FCFS (B) SJF (C) SRTN

11. Which is not a solution to the critical-region problem?
    (D) Shared memory
        (A) Semaphore (B) Monitor (C) Peterson’s solution

12. Which one is high-level language structure to solve the critical-region problem?
    (C) Shared memory
        (A) Test And Set Lock (B) Monitor (C) Shared memory

13. Which is not the necessary condition of a deadlock?
    (D) Circular wait
        (A) Mutual exclusion (B) Hold and wait (C) No starvation

14. Which is not the method for handing deadlocks?
    (D) Deadlock Detention
        (A) Deadlock Avoidance (B) Deadlock Prevention (C) Deadlock Ignorance

15. Which scheduling algorithm can avoid deadlock?
    (C) Banker’s algorithm
        (A) Ostrich algorithm (B) Rollback algorithm (C) Banker’s algorithm

    (D) None of the above
Part 2: (90 points)

1. Describe the process status. (6 points)

   running: Instructions are being executed.
   ready: The process is waiting to be assigned to a process.
   blocked: The process is waiting for some event to occur.

2. What is a race condition? How can we prevent a race condition? (6 points)
   
   (a) The race condition is the situation where several processes access and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.
   (b) To prevent race conditions, concurrent processes must be synchronized.

3. What are four conditions of a good solution to the critical-region problem? (8 points)
   
   (a) Mutual exclusion is guaranteed.
   (b) Progress is maintained. No process running outside its critical region may block other processes.
   (c) Bounded waiting is assured. No process should have to wait forever to enter its critical region.
   (d) No assumptions are made about the speeds of processes or the number of processors (CPUs).

4. How many processes will be created when the following program is executed?
   Assume that all fork system calls are successful. What will be printed? (10 points)
   (Hint: Be careful and draw a picture.)

   ```c
   main()
   {
       int i=1;
       int ret_val;

       while(i <= 3)
       {
           if ((ret_val = fork()) == 0) /* Child’s code */
               printf("In child %d. \n", i);
           i = i + 1;
       } else /* Parent’s code */
           printf("In parent %d. \n", i);
       exit(0);
   }
   }
   ```

   There are 4 processes (1 parent and 3 child processes) created when this program is executed. The following could be printed:

   In parent 1.
   In child 1.
   In parent 2.
   In child 2.
   In parent 3.
   In child 3.
5. Consider the following semaphore definition: A semaphore S is an integer variable that can only be accessed via two indivisible (atomic) operations: (12 points)

\[
\text{DOWN}(S): \text{while (} S \leq 0 \text{) /* do nothing */ ;} \\
\quad S = S - 1;
\]

\[
\text{UP}(S): \quad S = S + 1;
\]

A solution to the mutual exclusion problem, using these operations is shown below:

\[
\text{semaphore } S = 1; /* S is initialized to 1 */
\]

Each process executes the following code:

\[
\text{while (TRUE) } \{
\text{DOWN}(S); \\
\quad \text{- in critical section -}
\text{UP}(S);
\}
\]

(a) Does the above solution involve any busy-waiting? Explain.
(b) What is the "priority inversion" problem?
(c) If priority scheduling is used, can the above solution lead to the priority inversion problem? If so, give an example; otherwise, explain why not.

(a) This semaphore definition involves the busy waiting because a waiting process is looping for the shared variable S.
(b) A priority inversion problem is a situation in which a low-priority process is blocking a high-priority process.
(c) Yes, the above solution can lead to the priority inversion problem if a high-priority process is waiting for a low-priority process to leave the critical section and signal the shared variable S.

6. Add the semaphores necessary to synchronize processes A, B, C, D, and E so that process A and B must finish executing before C starts, process C must finish before D or E starts. Show your solution. Remember to indicate the initial value of each semaphore. (12 points)

The relations among processes can be represented in the following diagrams.

\[
\begin{array}{c}
A \quad \backslash \quad / \quad D \\
S1 \quad \backslash \quad / \\
\quad C \quad S3 \\
S2 \quad / \quad \backslash \\
B \quad / \quad \backslash \quad E
\end{array}
\]

The program can be as follows:

\[
\text{semaphore } S1, S2, S3 = 0, 0, 0;
\]

Process A:

\[
\text{-------------}
\quad \text{- do work of A}
\text{UP}(S1); \quad /* \text{Let C start} */
\]

Process B:

\[
\text{-------------}
\quad \text{- do work of B}
\]
Process C:
----------
DOWN(S1); /* Block until A finished */
DOWN(S2); /* Block until B finished */
- do work of C
UP(S3); /* Let D or E start */
UP(S3); /* Let D or E start */

Process D:
----------
DOWN(S3); /* Block until C finished */
- do work of D

Process E:
----------
DOWN(S3); /* Block until C finished */
- do work of E

7. Here is a solution to the readers writers problem using semaphores:

```c
semaphore db, mutex = 1,1;
shared int reader_count = 0;

Reader: Writer:

DOWN(mutex);
reader_count++; - write to database -
if (reader_count==1) DOWN(db);
UP(db);
UP(mutex);
- read from database -
DOWN(mutex);
reader_count--;
if (reader_count==0) UP(db);
UP(mutex);
```

Rewrite the above solution using message passing instead of semaphores; in particular use mailboxes (UNIX pipes) and the notation `write(mbox, msg)` and `read(mbox, msg)`. (12 points)

```c
msg = 0;
Reader: Writer:

read(mutex,msg);
reader_count++;
if (reader_count==1) read(db, msg);
write(mutex);
- read from database -
read(mutex,msg);
reader_count--;
if (reader_count==0) write(db,msg);
write(mutex,msg);
```
8. Suppose that the following processes arrive for execution at time 0 in the order A, B, C:

<table>
<thead>
<tr>
<th>Process</th>
<th>Run Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>1 = high</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3 = low</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

(a) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 2) scheduling.

(b) What is the waiting time of each process for each of the scheduling algorithms?

(c) What is the turnaround time of each process for each of the scheduling algorithms?

(12 points)

(a) The four Gantt charts are

(a) The four Gantt charts are

(b) Waiting time:

(c) Turnaround time:

9. P is a set of processes. R is a set of resources. E is a set of request or assignment edges. The sets P, R, and E are as follows:

\[ P = \{P_1, P_2, P_3\}, \quad R = \{R_1, R_2, R_3\}, \]

\[ E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, P_3 \rightarrow R_2, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\} \]

R_1 has one instance. R_2 has two instances. R_3 has one instance.

(a) Draw the resource-allocation graph.

(b) Is there any deadlock in this situation? Briefly Explain.

(12 points)

(a) See the graph.

(b) Consider the resource-allocation graph. Two cycles exist in the system.

P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1

P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2

P_1, P_2, and P_3 are deadlocked.