Part 1: (36 points - 3 points for each problem)

1. Which statement is false?
   (A) An operating system is used to manage all devices.
   (B) In UNIX I/O devices are treated as special files.
   (C) The core part of the operating system is called kernel.
   (D) None of the above

2. Which POSIX system call replaces a process’ core image?
   (A) fork (B) execv (C) run (D) None of the above

3. Which statement about processes is false?
   (A) A process information is stored in the process data block.
   (B) A process can generate child processes.
   (C) Two processes can be connected by a pipe.
   (D) None of the above

4. Which is not the benefit of using threads?
   (A) Responsiveness (B) Economy (C) Fairness (D) None of the above

5. Which statement about kernel and user-level threads is true?
   (A) Both kernel and user-level threads share signals.
   (B) Both kernel and user-level threads have their own program counter, stack, and registers.
   (C) Kernel thread scheduling is faster than user-level thread scheduling.
   (D) None of the above

6. Which of the following instructions can be run in user mode?
   (A) Disable all interrupts. (B) Set the time-of-day clock.
   (C) Change the memory map. (D) None of the above

7. Which scheduler decides the degree of multiprogramming?
   (A) Admission scheduler (B) Memory scheduler (C) CPU scheduler (D) None of the above

8. Which is not a CPU scheduling criterion?
   (A) CPU utilization (B) Response time (C) Reliability (D) None of the above

9. Which is not a preemptive scheduling?
   (A) RR (B) SRTN (C) SJF (D) None of the above

10. Which scheduling algorithm is better for an interactive system?
    (A) RR (B) SRTN (C) SJF (D) None of the above

11. Which cannot be used to solve the critical-region problem?
    (A) Test And Set Lock (B) Monitor (C) Message passing (D) None of the above

12. The aging algorithm with \( a = 1/2 \) is being used to predict run times. The previous 3 runs, from oldest to most recent, are 20, 40, and 20 msec. What is the prediction of the next time?
    (A) 20 (B) 25 (C) 30 (D) None of the above

Part 2: (64 points)

1. (8 pts.) What are four conditions of a good solution to the critical-region problem?
   (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).
2. (9 pts.) Answer the following questions.

(a) (2 pts.) What is a shell?
   Ans: A shell is a process that reads commands from a terminal.

(b) (2 pts.) What is a daemon?
   Ans: A process running in the background to handle some activity is called daemon.

(c) (3 pts.) What are three process states?
   Ans: running, ready, blocked.

(d) (2 pts.) What is a race condition?
   Ans: 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.

3. (11 pts.) Consider the "DOWN" and "UP" operations defined on a shared integer S:

   DOWN(S): while (S <= 0) /* do nothing */ ;
   S = S - 1;

   UP(S): S = S + 1;

Assume that the operations are executed atomically (however, a process may be preempted at the beginning of the while loop if (S < 0)). A solution to the mutual exclusion problem, using these operations is shown below:

   shared int S = 1; /* S is initialized to 1 */

   Each process executes the following code:

   while (TRUE) {
      DOWN(S);
      - in critical section -
      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.
(d) If wait and signal are not executed atomically, what property of a good solution may be violated? Give an example.

Ans:

(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.
(d) The mutual exclusion is violated. When process A is in the while loop, the context switch happens. Process B could enter the critical region after executing the while loop and S = S - 1. Then another context switch happens. Process A could enter the critical region after executing S = S - 1.
4. (8 pts.) Write a semaphore solution to the Producer-Consumer Problem; recall that producers and consumers have access to a shared buffer that can hold up to \( N \) items. Remember to indicate the initial value of each semaphore.

```c
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer(void) {
    int item;
    while (TRUE) {
        item = produce_item();
        down(empty);
        down(mutex);
        insert_item(item);
        up(mutex);
        up(full);
    }
}

void consumer(void) {
    int item;
    while (TRUE) {
        item = remove_item();
        up(mutex);
        up(empty);
    }
    consume_item(item);
}
```

5. (8 pts.)

(a) How many processes will be created when the following program is executed?

(b) Assume that all fork system calls are successful. What will be printed?

(Hint: Be careful and draw a picture.)

```c
main() {
    int i, pid;
    for (i=1; i<=3; i++)
        pid = fork();
    if (pid > 0)
        printf("In parent %d \n",i);
    else
        printf("In child %d \n",i);
}
```

There are 8 processes (1 parent and 7 child processes) created when this program is executed. The following could be printed:

- In parent 4
- In child 4
- In parent 4
- In child 4
- In parent 4
- In child 4
- In parent 4
- In child 4
6. (12 pts.) 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>4</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3 = low</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1 = high</td>
</tr>
</tbody>
</table>

(a) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpre-emptive 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?

Ans:

(a) The four Gantt charts are

```
+---+-----+----+ FCFS
    | A   | B   | C   |
+---+-----+----+----+
0 4 6 9
```

```
+---+-----+----+ SJF
    | B   | C   | C   |
+---+-----+----+----+
0 2 5 9
```

```
+---+-----+----+ Priority
    | C   | A   | B   |
+---+-----+----+----+
0 3 7 9
```

```
+---+-----+----+ RR
    | A   | B   | C   | C |
+---+-----+----+----+
0 2 4 6 8 9
```

(b) Waiting time:

```
<table>
<thead>
<tr>
<th>FCFS</th>
<th>SJF</th>
<th>Priority</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
```

(c) Turnaround time:

```
<table>
<thead>
<tr>
<th>FCFS</th>
<th>SJF</th>
<th>Priority</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
```
7. (8 pts.) Use message passing to synchronize processes A, B, C, D, and E so that process A and B must finish before C starts, process B must finish before D starts, process C and D must finish before E starts. Use the notation \texttt{write(mbox, msg)} and \texttt{read(mbox, msg)} to write your solution.

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

```
A
 \ mbox1
  \ 
mbox1 C
 / \ 
B / \ E
/ 
mbox2 \ / mbox3
  D
```

The program can be as follows:

```
Process A:
----------
- do work of A
write(mbox1, msg); /* Let C start */

Process B:
----------
- do work of B
write(mbox1, msg); /* Let C start */
write(mbox2, msg); /* Let D start */

Process C:
----------
read(mbox1, msg); /* Block until A finished */
read(mbox1, msg); /* Block until B finished */
- do work of C
write(mbox3, msg); /* Let E start */

Process D:
----------
read(mbox2, msg); /* Block until B finished */
- do work of D
write(mbox3, msg); /* Let E start */

Process E:
----------
read(mbox3, msg); /* Block until C finished */
read(mbox3, msg); /* Block until D finished */
- do work of E
```