No Electronics.

Write one of the words or terms from the following list into the blank appearing to the left of the appropriate definition. Note that there are more words and terms than definitions. (1 pt. each)

1. ______________________ A sequence of code that operates on shared state.
2. _______________________ The length of time that a task is scheduled before being preempted.
3. ______________________ When a scheduler takes the processor away from one task and gives it to another.
4. ______________________ When a thread uses a lock to prevent concurrent access to a shared data structure.
5. _______________________ A thread spins in a loop waiting for a concurrent event to occur, consuming CPU cycles while it is waiting.
6. _______________________ A synchronization variable that enables a thread to efficiently wait for a change to shared state protected by a lock.
7. ______________________ A scheduling policy where the OS assigns threads to processors without knowledge of the intent of the parallel application.
8. ______________________ A way to increase concurrency by partitioning an object’s state into different subsets each protected by a different lock.
9. ______________________ A scheduling policy for multiprocessors that performs all the runnable tasks for a particular process at the same time.
10. _____________________ An atomic read-modify-write instruction that reads a value from memory to a register and writes the value 1 to that memory location.
11. _____________________ An atomic read-modify-write instruction that first tests the value of a memory location, and if the value has not been changed, sets it to a new value.
12. _____________________ A scheduling policy where tasks are preferentially scheduled onto the same processor that they had previously been assigned, to improve cache reuse.
13. _____________________ Extra inter-processor communication required because a single cache entry contains portions of two different data structures with different sharing patterns.
14. _____________________ An instruction that prevent the compiler and hardware from reordering memory accesses across the barrier - no accesses before the barrier are moved after the barrier and no accesses after the barrier are moved before the barrier.
15. _____________________ A lock which allows multiple "reader" threads to access shared data concurrently provided they never modify the shared data, but still provides mutual exclusion whenever a "writer" thread is reading or modifying the shared data.
16. Consider a stack represented by the following declarations and initialized as shown.

```c
struct node{
    int data;
    struct node *next;
} d, e, f;
struct node *head;
head = &f;
f.next = NULL;
```

Now consider two threads each trying to push a node on to the stack.

```
T1S1: e.next = head;              T2S1: d.next = head;
T1S2: head = &e;                  T2S2: head = &d;
```

The sequence { T1S1; T1S2; T2S1; T2S2 } changes the state of the stack to: head->d->e->f->NULL.

Starting over from an initial stack of head->f->NULL, consider the sequence { T1S1; T2S1; T1S2; T2S2 }.

Show the state of the stack after this sequence. Is there a problem with this sequence? If so, explain. (4 pts.)

17. Consider implementing a spin lock. What is the difference between a normal test-and-set lock and the “test and test-and-set” lock? Why is there a performance benefit? (4 pts.)

18. What is the difference between the “test and test-and-set” lock and the Mellor-Crummey and Scott (MCS) lock? Why is there an additional performance benefit? (4 pts.)
19. This is the semaphore approach we reviewed in class for managing the allocation of \( k \) resources among \( n \) threads.

```c
semaphore s1(1);    // initialized to 1
semaphore sk(k);    // initialized to \( k \), where \( k > 0 \)
queue q;            // initialized with entries \( 0,1,\ldots,k-1 \)

thread( int i ){    
    int resource_id;
    P(sk);
    P(s1);
    resource_id = get(q);  // obtain one of \( k \) resource ids
    V(s1);
    work(resource_id);
    P(s1);
    put(q,resource_id);    // return resource id
    V(s1);
    V(sk);
}
```

Using lock(s) and condition variable(s) instead of semaphores, give the code with the equivalent functionality. Assume that you can use the same queue data type with get() and put() operations. (15 pts.)
20. Beside spurious wakeups, why must the CV::wait() operation be called from within a loop? (4 pts.)

21. Under what two conditions does the textbook say it is appropriate to replace a mutual exclusion lock with a readers/writers lock (RWLock)? (4 pts.)

22. What is the key idea of read-copy-update (RCU)? (4 pts.)

**Fully live / Starvation / Deadlock.** Circle **only one** of F, S, or D. (1 pt. each)

23. **F / S / D** No process under consideration can make forward progress.
24. **F / S / D** All processes under consideration can make forward progress.
25. **F / S / D** At least one process under consideration can make forward progress but at least one cannot.

**Deadlock Prevention / Avoidance / Detection and Recovery.** Circle **only one** of P, A, or D&R. (1 pt. each)

26. **P / A / D&R** Rollback and retry.
27. **P / A / D&R** Banker’s algorithm.
28. **P / A / D&R** Preempt resources.
29. **P / A / D&R** Hierarchical ordering of resource requests.
30. **P / A / D&R** Release currently-held resources before requesting more.

**True/False.** Circle **only one** of T or F. (2 pts. each)

Use this table for questions 31-33 for deadlock avoidance questions based on the Banker’s algorithm. Treat each question independently, starting each time from the values shown in the table.

<table>
<thead>
<tr>
<th>process</th>
<th>max need</th>
<th>allocated</th>
<th>remaining need</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Unused units = 6

31. **T / F** Can safely grant request of process A for 3 units.
32. **T / F** Can safely grant request of process B for 3 units.
33. **T / F** Can safely grant request of process C for 3 units.
34. Given the following list of tasks, arrival times, and service times, calculate the completion (i.e., departure) time and response (i.e., turnaround) time of each task for FIFO scheduling and again for SJF-preemptive scheduling. (16 pts.)

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>Service Time</th>
<th>FIFO Completion Time</th>
<th>FIFO Response Time</th>
<th>SJF-preemptive Completion Time</th>
<th>SJF-preemptive Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

35. Describe the workload in which FIFO results in pessimal (i.e., worst case) scheduling. (4 pts.)
36. If priority aging will eventually solve a priority inversion problem, why would you additionally implement priority donation? (4 pts.)

37. If additional resources are not available, what are two approaches identified in the textbook to manage overload? (4 pts.)

38. Define space-sharing scheduling and explain the performance benefit that can result. (4 pts.)