Assignment #2

[20 points]

To succeed in this assignment, prioritize conciseness—convey your ideas in as few words as possible. Embrace open-ended questions, where there's no single correct answer. Instead, make reasonable assumptions and respond as a wireless network designer, researcher, or mathematician would. For math questions, ensure your calculations are clear and logical, explaining your reasoning step-by-step. Best of luck!

1. **Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain. (2 pts.)**

2. **A multithreaded web server wishes to keep track of the number of requests it services (known as hits). Consider the two following strategies to prevent a race condition on the variable hits.**

   The first strategy is to use a basic mutex lock when updating hits:
   ```
   int hits;
   mutex lock hit lock;
   hit lock.acquire();
   hits++;
   hit lock.release();
   ```

   A second strategy is to use an atomic integer:
   ```
   atomic t hits;
   atomic inc(&hits);
   ```

   Explain which of these two strategies is more efficient. (2 pts.)

3. **For each scheduling algorithm (FCFS, SJF, Priority, and RR), calculate the following for each process:**
   - **Finish Time**
   - **Turnaround Time**
   - **Waiting Time**
   - **Draw a Gantt chart**

   Then, identify which process has the minimum waiting time. Time Quantum = 2. (8 pts.)

Processes

Process	**Arrival Time**	**Burst Time**	**Priority**
P1	0	5	3
P2	2	3	1
P3	4	1	4
P4	6	7	2

4. **Write a program that simulates a soda dispenser using a stack data structure. Create a class named SodaDispenser with methods to manage the dispenser. Implement the following methods: add_soda(self, soda) to add a soda (represented as a string) to the top of the dispenser, dispense_soda(self) to remove and return the soda from the top of the dispenser, handling the case where the dispenser is empty by raising an exception or returning a special value, is_empty(self) to return True if the dispenser is empty and False otherwise, and peek_soda(self) to return the soda at the top of the dispenser without removing it, also handling the empty dispenser scenario appropriately. Ensure your program includes test cases demonstrating the functionality of these methods. Submit your implementation in Blackboard as a file named SodaDispenser followed by your name and ID, using the appropriate file extension for the language you choose. Your submission will be evaluated on the correctness of the methods, handling of edge cases, code readability, and proper commenting. (8 pts.)**