Report nhc29 pkk46

.1 What structures you used to implement the TCB, mutex, runqueues, any other queues or structures, etc.

Struct for the thread control block, here, it holds information about the thread, so I utilized thread_id, yield_id, status, and context
Struct containing Atomic_flag for mutex, which tells me if its locked or not
Struct for linked list queue, that holds the head, and tail of the queue. Each element in the queue is represented by a thread node that has a pointer to data for more information

2. What logic you used for implementing the thread and mutex API functions.

When a new thread is created, we allocate memory for its TCB, and initialize its fields. Then we add the TCB to the thread queue we made, and then we can set the initial status of the head of the queue. When we need to terminate a thread, we mark it as finished, and then remove the TCB from our queue.

Just utilized the built-in atomic functions inside GCC __atomic_test_and_set, __atomic_clear (I think it should be allowed), which handled creation, and setting. (Did not need to delete as I did not add any mutex memory to the heap)

3. What logic you used for implementing the scheduler(s). (How it keeps track of which threads are in what states, how it handles transitions between states, how it chooses which thread to pick next, etc.)

Here the scheduler will determine what thread runs next. We utilize a round-robin approach, which goes through the entire queue. Here we have our timer, which runs on a set quantum (defined at the top). Here, this is done by iterating the queue, checking the status of our current node's data, doing an operation, and requeuing the node for further changes. This means after a scheduled thread has run for a time period equal to the quantum, we reschedule it and add it back to the runqueue.

MLFQ scheduler logic:

The MLFQ (Multi-Level Feedback Queue) scheduling algorithm is implemented in the provided code as a preemptive, priority-based scheduler with dynamically changing priority levels. The algorithm maintains an array of queues (mlfq) representing different priority levels. The scheduler iterates through these queues, starting from the highest priority, and selects the first thread in the non-empty queue for execution. Upon selecting a thread, it sets its status to THREAD_STATUS_RUNNING and swaps the context to initiate its execution. After the thread completes its time quantum, the scheduler checks its status. If the thread is still running, indicating it used up its time slice, it is moved to a lower-priority queue. If the thread yielded or blocked, it remains in the same queue. This process continues until a thread is selected for

execution, promoting a dynamic adjustment of thread priority levels based on their behavior, ensuring fairness and responsiveness in handling varying workloads.

4. Collaboration and References: State clearly all people and external resources (including on the Internet) that you consulted. What was the nature of your collaboration or usage of these resources?

Stdatomic documentations for C from GCC for the mutex operations definitely helped.
Asking chatgpt for example code as reference and explanations to the example code to help me understand.
Stack overflow for previous online questions about context control, and why my threads were not exiting properly in terms of debugging purposes