RTOS and Priority Preemption - Part 2

Group 7

Documentation for Priority Thread Assignment

Our task was to develop a real-time embedded application to control a stopwatch with start/stop and reset functionalities, along with LED indicators. The application uses POSIX threads with assigned priorities to handle button inputs, LED outputs, and the stopwatch counter display. Here's a documentation of the priority assignments to the threads, including the reasoning behind these choices:

Threads Overview:

1. **Button Listener Thread**: Listens for button press events (start/stop and reset).
2. **Stopwatch Counter Thread**: Manages the stopwatch's counting mechanism.
3. **LED Controller Thread**: Controls the LED indicators based on the stopwatch state.
4. **Display Update Thread**: Updates the display with the current stopwatch time.

Priority Assignments:

- **Button Listener Thread (Priority: 99)**: This thread is given the highest priority because user input should be responsive and have minimal latency. By prioritizing the button listener, we ensure that the system quickly acknowledges user inputs, even when busy with other tasks like updating the display or managing LEDs. This task is also being read every 10ms, which is the shortest period in comparison to other tasks.
- **Stopwatch Counter Thread (Priority: 70)**: This thread is crucial for the core functionality of the stopwatch; thus, it's given a high priority but lower than the button listener. The priority ensures the counter remains accurate and minimizes time drift, which is essential for a stopwatch. However, it doesn't need to be as high as the button listener since the exact moment of incrementing the counter (within a few milliseconds) is less critical than capturing the user's input.
- **LED Controller Thread (Priority: 50)**: **Reasoning**: The LED controller is important for providing visual feedback to the user but is considered less critical than capturing user input or maintaining accurate timing. Therefore, it receives a medium priority. This ensures that the LEDs reflect the current state of the application (e.g., running or stopped) in a timely manner without preempting more critical tasks.
- **Display Update Thread (Priority: 30)**: Updating the display, while important for user feedback, is the least time-sensitive task in this application. Given the lowest priority, this thread will not preempt any other thread unless they're idle or waiting, ensuring that the system prioritizes responsiveness and accuracy over display updates. The requirements also mention how it should be updated every 100ms, the longest of any other.

Summary:

The priority assignments are designed to optimize the application's responsiveness to user inputs and the accuracy of the stopwatch mechanism, while still providing timely feedback through the LEDs and display. This approach follows the Rate Monotonic Scheduling (RMS) principle we were tasked to implement, where tasks with shorter periods (or higher rates of execution) are given higher priorities. In this context, the button listener is seen as having the highest rate of execution necessity, followed by the stopwatch counter, LED controller, and finally the display updater.