Documentation of Priority Assignments Using Rate Monotonic Scheduling (RMS)

In the `stopwatch.c` code, threads are assigned priorities based on the principles of Rate Monotonic Scheduling (RMS). RMS is a fixed-priority scheduling algorithm where shorter-period tasks are assigned higher priorities. This approach minimizes response time and ensures timely execution of periodic tasks.

Thread Descriptions and Their Periods

1. **Timer Update Thread (`update_timer`)**
   - **Function:** Increments the stopwatch counter by 10ms.
   - **Period:** 10ms (highest frequency task).
   - **Reasoning:**
     - This thread has the most frequent deadline (every 10ms) because it updates the stopwatch timer.
     - Missing a deadline for this thread would result in incorrect or delayed timer updates.
     - It must, therefore, have the highest priority.
2. **Button Input Handling Thread (`handle_button_input`)**
   - **Function:** Reads GPIO values to detect button presses and updates the running/paused/reset state of the stopwatch.
   - **Period:** 10ms (same as the timer thread).
   - **Reasoning:**
     - Although this thread also runs every 10ms, it processes user input, which is less critical than real-time timer updates.
     - Assigning it a slightly lower priority than `update_timer` ensures that timer updates are not delayed due to input processing.
3. **Display Thread (`display_timer`)**
   - **Function:** Prints the stopwatch time to the terminal every 100ms.
   - **Period:** 100ms (lowest frequency task).
   - **Reasoning:**
     - This thread has the lowest frequency of execution and provides user feedback.

- It is less critical than the timer update and button input threads.
- Assigning it the lowest priority ensures that it does not block higher-priority threads from completing on time.

Implementation in Code

The priorities are implemented using POSIX thread attributes (`pthread_attr_t`) and the `SCHED_FIFO` **policy**. Below is the relevant part of the `main` function:

```
pthread_attr_init(&attr);
pthread_attr_setschedpolicy(&attr, SCHED_FIFO);

int max_priority = sched_get_priority_max(SCHED_FIFO);
int min_priority = sched_get_priority_min(SCHED_FIFO);

// Timer thread (highest priority)
param.sched_priority = max_priority;
pthread_attr_setschedparam(&attr, &param);
pthread_create(&timer_thread, &attr, update_timer, NULL);

// Display thread (medium priority)
param.sched_priority = max_priority - 1;
pthread_attr_setschedparam(&attr, &param);
pthread_create(&display_thread, &attr, display_timer, NULL);

// Button thread (lowest priority)
param.sched_priority = min_priority;
pthread_attr_setschedparam(&attr, &param);
pthread_create(&button_thread, &attr, handle_button_input, NULL);
```

Conclusion

The priority assignments based on RMS in the `stopwatch.c` code ensure:

- Timely updates of the stopwatch counter.
- Responsive handling of user inputs.
- Reliable display of stopwatch time without disrupting critical tasks.

This priority scheme effectively balances real-time constraints and user feedback requirements.