**1. Thread Behaviour (CPU-bound vs. IO-bound):**

The threads in the given C program simulate workloads by executing the same function (threadFunction) but with different sleep times.

* **CPU-bound threads** are tasks that perform intense computation and require a lot of CPU resources. In this case, the threads are **not CPU-bound** because they primarily sleep, which means they don't engage the CPU much. Instead, they spend time in a "waiting" state.
* **IO-bound threads**, on the other hand, spend most of their time waiting for an external event (such as I/O operations) to complete. In this program, the threads simulate **IO-bound behaviour** because they are all sleeping for different durations (1, 3, and 5 seconds). During the sleep time, the threads are not actively using the CPU but are instead idle, which resembles the behaviour of IO-bound tasks.

**2. CPU Scheduling with Multi-level Feedback Queue (MLFQ) Scheduling:**

In the case of **Multi-level Feedback Queue (MLFQ)** scheduling, the operating system maintains several queues, each with different priority levels. The threads are initially placed in the highest priority queue (which gives them a chance to run sooner), and if they don't finish quickly, they are moved to lower priority queues. The priority of the threads is dynamically adjusted based on their behaviour.

How the **MLFQ** scheduling works in this scenario with respect to the different sleep times of the threads:

* **Thread with the shortest sleep time (1 second)**: It will finish its execution quickly and will likely stay in the highest priority queue. Since it has the least sleep time, the scheduler will give it CPU time and allow it to finish before moving to lower-priority queues.
* **Thread with the middle sleep time (3 seconds)**: This thread may be moved to a lower priority queue once it exceeds the expected time quantum or finishes its initial task. The scheduler might decide that this thread requires less immediate attention and therefore lower priority.
* **Thread with the longest sleep time (5 seconds)**: This thread will likely move to the lowest priority queue because it has the longest sleep time. The system will prioritize threads that finish their tasks faster and assign more CPU time to those tasks.

In **MLFQ scheduling**, threads with shorter execution times (like the one with the 1-second sleep time) generally remain in the high-priority queues, while those with longer execution times (like the 5-second sleep time) may be demoted to lower-priority queues. This approach ensures that CPU resources are allocated efficiently and that shorter tasks are completed faster, promoting fairness in the scheduling process.

**3. Round-Robin Scheduling and Quantum Time Effect:**

The **Round-Robin (RR)** scheduling algorithm allocates a fixed time slice or **quantum** to each thread. The threads are scheduled in a cyclic order, and each thread gets the CPU for a fixed amount of time before being pre-empted and moved to the back of the queue. The key performance factor in Round-Robin scheduling is the **quantum time** (the time slice allocated to each thread).

The effect of different quantum times:

**Short Quantum Time**: If the quantum time is very short, the threads will be frequently switched in and out of the CPU. In this case, the threads that sleep for shorter times (like the 1-second thread) will likely finish quickly, but threads that sleep for longer times (like the 5-second thread) will spend more time waiting for their turn.

* **Performance impact**: Frequent context switching can lead to increased overhead due to the time spent switching between threads. This overhead can slow down overall performance, especially if the threads are lightweight or don't need much CPU time.

**Long Quantum Time**: If the quantum time is long, threads will be allowed to run for a longer time before being pre-empted. This might be more efficient for CPU-bound threads, as they have enough time to execute without frequent interruptions.

* **Performance impact**: Threads with shorter sleep times (1-second thread) might remain idle while waiting for the long quantum time to expire. Threads that sleep for 3 or 5 seconds might have a chance to complete their execution more efficiently without being pre-empted

A balance needs to be found to minimize overhead while ensuring that each thread is given a fair amount of CPU time. In the case of this program, the quantum time should ideally be set based on the sleep times to prevent excessive context switching and to optimize for responsiveness and efficiency.