**Consider a system in which threads are implemented entirely in user space, with the runtime system getting a clock interrupt once a second. Suppose that a clock interrupt occurs while some thread is executing in the runtime system. What problem might occur? Can you suggest a way to solve it?**

**Step 1:**

Multiple threads at user space are mapped with a single thread in the kernel level. In other words, all user-level threads execute on the same kernel thread. Since there is only one kernel thread, only one user-level thread runs at a time. Each user thread makes a system call to create a new kernel thread.

It's possible that the runtime system is busy altering the scheduling queues just as a thread is about to be blocked or unblocked. It would be a very bad idea for the clock interrupt handler to start analysing those queues to see if it was time to swap threads now, because they could be in an inconsistent state.

**Step 2: Solution for the above problem**

One option is to set a flag when the system's runtime is entered, allowing the clock handler to observe it and set its own flag before returning to its previous place. After the runtime system has completed its tasks, it will examine the clock flag to check if a clock interrupt has occurred, and then execute the clock handler.