Introduction

The goal of this lab is the implementation of the `timer` function. Instead of *busy-waiting*, a thread is blocking and put to sleep for a certain amount of time (*ticks*).

Data Structures

The only data structure added to the unmodified codebase is a counter variable `sleep_counter` in the `thread` struct in `thread.h`. This simply contains the remaining number of *ticks* to sleep for at any given time.

Algorithms

In `timer_sleep`, the number of ticks to sleep is saved in the current `thread` struct and the thread is suspended using `thread_block`. Invalid `ticks` values are rejected, terminating the function. This is shown in fig. 1.

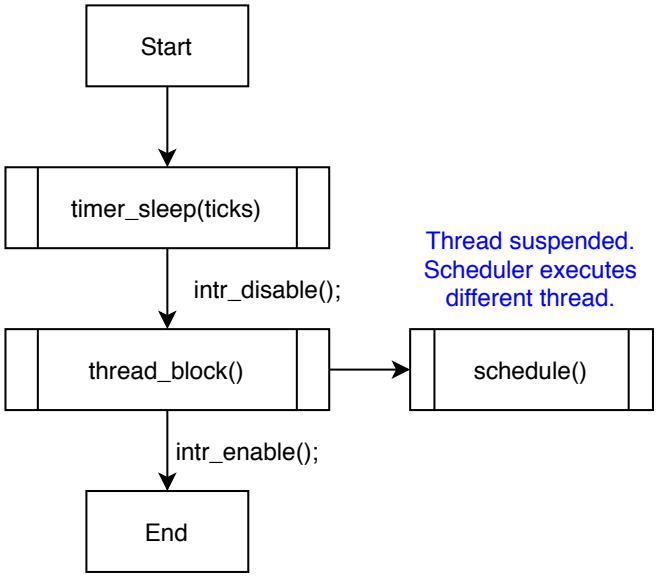

Figure 1: `timer_sleep` called by programs in userspace. The thread will suspend execution and the scheduler will activate a different thread.

The `timer_interrupt` interrupt handler function is invoked on every tick. This calls `thread_forall()` to iterate over all threads in the system and, in each thread context run a predefined function, which we have created and called `wakeup`. `wakeup` checks if the current thread is blocked, and if so either decrements the thread's `sleep_counter` or, if it has reached zero, unblocks the thread using `thread_unblock()`. This is shown in more detail in fig. 2

Synchronisation

Since two or more threads are not accessing or operating on the same data when being put to sleep, no additional synchronisation methods were required to ensure the correctness of the program.

There is no shared data as such, but interrupts are disabled when threads must not be preempted, such as when calling `thread_block()`.

The `thread_block()` recommends the use of synchronization primitives when invoking, though it is unclear how this would be of improvement.

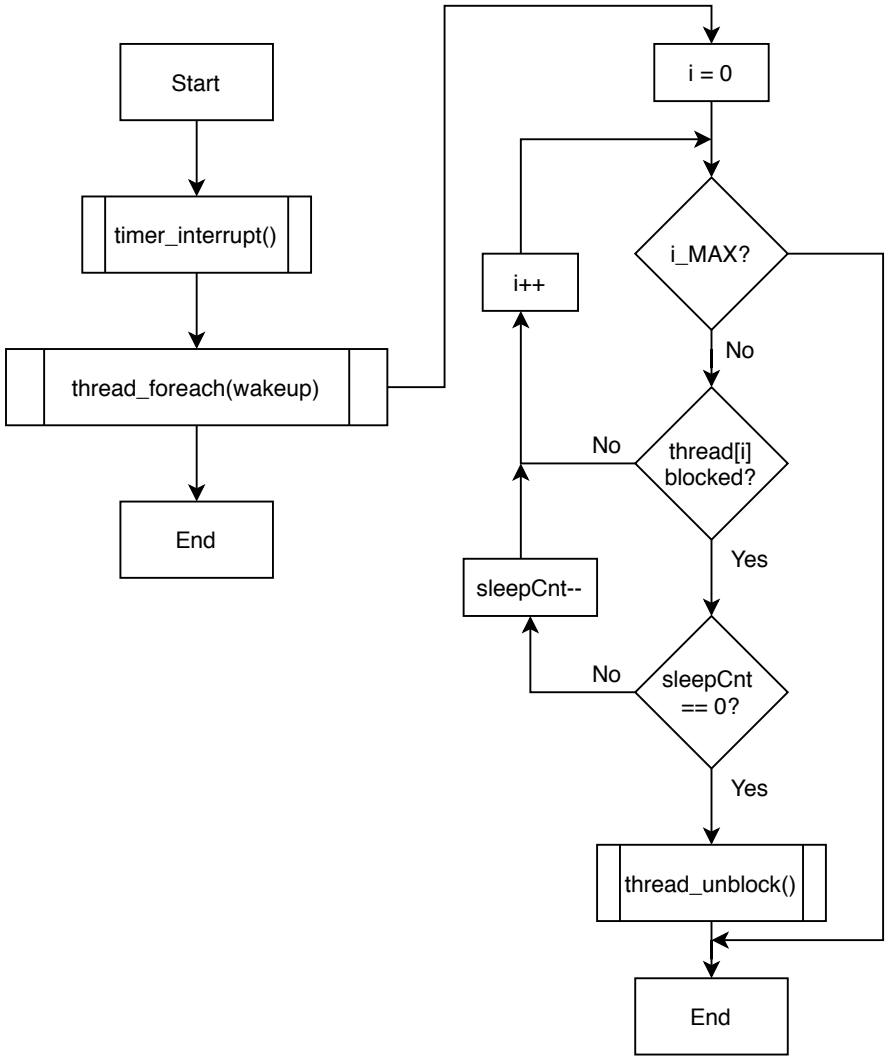

Figure 2: The timer interrupt handler iterates over all threads, checks if they are blocked and unblocks them if enough time has elapsed.

Rationale

The algorithm performs a comparison and decrements a sleep counter on each tick. A different approach might have been to compare elapsed time with the starting time (when `timer_sleep` was called), though the simplicity of the first approach was preferred since it executes in an interrupt context. The latter approach calls additional functions which is undesirable.

Addendum

No effort was made to make the system pass the `batch-scheduler` test. Of note is that, after modification of the timer code, that particular test times out. This may be indicative of undesired behaviour.