Lab 2 - Group 58

William Leven (levenw@student.chalmers.se)
Vidar Magnusson (mvidar@student.chalmers.se)

1 Data structures

For our implementation we added an `alarm_tick` integer to the `thread` struct that tracks after which tick that thread should wakeup. Furthermore, we created one data structure and used multiple ones. The data structure we created is a list entry struct that is used with the built in `lib/kernel/list` to track which threads are currently being blocked by the timer, this struct can be seen in figure 1.

```
/* A list entry to track that a thread is blocked. */
struct blocked_thread {
    struct list_elem elem;
    struct thread *thread;
};
```

Fig. 1. blocked_thread, the data structure we created

2 Algorithms

When `timer_sleep()` is called we save the wakeup tick in the thread struct and create an entry in out list of sleeping threads containing a pointer to the thread. When this is done we block the thread.

When the timer generates an interrupt we go through the list of sleeping threads and wake up (unblock) any sleeping threads whose wakeup tick is set to a value lower than (or equals to) the current tick.

When the thread is woken up it takes care of cleanup (removing its alarm from the global list of alarms and freeing up memory) before returning from the `timer_sleep()` function.

3 Synchronization

For synchronization we use the lock data structure found in `threads/synch`. We make sure to acquire this lock before adding or removing elements from the list of blocked threads. We do not lock before iterating the list as list edits wont break iteration.

4 Rationale

Why we chose the data structures we used was mostly because they were readily available in the project and as such did not require any implementation of our own as well as avoiding the many issues that comes with using third party libraries.

We let threads do their own cleanup to reduce time spent in interrupts. This might introduce some extra overhead as we have to use synchronization primitives. However, the overhead will only be paid by threads that want to sleep, instead of it being paid by all threads on the system as would be the case if we did the clean up in the interrupt.

The reason for why we do not lock when traversing or doing other forms of reading from the list is because removals wont destroy the iteration. The one thing that could destroy the iteration is if the element is destroyed while we are reading it but there is no risk of this happening as no threads run while we are in an interrupt.

We modified the thread structure by adding the wakeup time member variable to it because we were told by the supervisor that we would be unconditionally failed on the lab if we didn't do this. The supervisors argument was that not storing that variable in the thread would be a ”sub-optimal solution”. We strongly disagree with this argument as the modification of internal structures breaks the modularity of the timer device. We think that the better solution is to store the wakeup time in the list element (in the list of blocked threads) so that they can be accessed while we loop over the list of blocked threads.