---- GROUP 17 ----

Sahil Jindal, 18CS10048
Ashutosh Varshney, 18CS30009

---- PRELIMINARIES ----
Have passed alarm-single, alarm-multiple, alarm-simultaneous, alarm-zero, alarm-negative.
Need to make few changes after installation apart from that given in the assignment:

1. In file “$HOME/pintos/src/threads/Make.vars”, change the last line to: SIMULATOR = --qemu
2. In `devices/shutdown.c` patch `shutdown_power_off` as follows:

```
void shutdown_power_off (void)
{ // ...
printf ("Powering off...\n");
serial_flush ();
outw (0xB004, 0x2000); // <-- Add this line
// ...
}
```

ALARM CLOCK
===========

---- DATA STRUCTURES ----

A1: Copy here the declaration of each new or changed `struct' or `struct' member, global or static variable, `typedef', or enumeration. Identify the purpose of each in 25 words or less.

1. Added an attribute `int64_t wake_up_time` to the `struct thread` in `thread.h` for maintaining wake up time for each thread in the kernel, that is, the time at which they should be released from the block state.
2. Added a global list `struct list waiting_list` in `timer.c` which stores the list of blocked threads. This list is sorted on the basis of wake_up_time, that is, the thread with the earliest `wake_up_time` is in the front.
3. Added a comparator function `bool compare_waking_time(struct list_elem *a, struct list_elem *b)` in `timer.c` which compares wake-up times of two threads in order to add a new thread in `waiting_list`.

---- ALGORITHMS ----

A2: Briefly describe what happens in a call to timer_sleep(), including the effects of the timer interrupt handler.

In the function `timer_init()`:

1. The `waiting_list` is initialized in the `timer_init()` function.

Then in the function `timer_sleep()`:

2. Make sure that external interrupts are enabled.
3. Assign the wake_up_time to the thread which is the sum of the start time and the number of ticks for which the thread has to be blocked.
4. Disable the interrupts.
5. Insert the thread in the `waiting_list` by comparing the wake up time. (Threads with sooner `wake_up_time` comes in front)
6. Switch context to block the thread.
7. Enable the interrupts.

Then finally in the function `timer_interrupt()`:

8. Iterate on the priority list `waiting_list` from the front.
9. Check if the current thread has surpassed its `wake_up_time`. If yes, then pop it out of the `waiting_list` and repeat the process otherwise stop.

A3: What steps are taken to minimize the amount of time spent in the timer interrupt handler?

By maintaining a global `waiting_list`, time spent in the `timer_interrupt()` is minimized since the list maintains all the threads sorted on the basis of `wake_up_time`. So the interrupt handler only has to look in the front of the list and pop out the elements till it sees a thread that doesn't need to be woken up. Thus the interrupt handler unblocks all the threads that have passed the `wake_up_time` in one call minimizing the time spent. There is at most only one thread that need not be woken up by the interrupt.

---- SYNCHRONIZATION ----

A4: How are race conditions avoided when multiple threads call timer_sleep() simultaneously?

For each call to `timer_sleep()`, it is checked whether the interrupts are enabled or not before entering the critical section. This avoids race conditions as a thread entered in the critical section of the code first needs to enable the interrupt again before exiting the function. Therefore, the call will wait till the interrupts are not enabled again.

A5: How are race conditions avoided when a timer interrupt occurs during a call to timer_sleep()?

The interrupts are disabled before entering the critical section to ensure that no interrupt stops its execution and thus avoids the race condition. After the completion, the interrupts are enabled again.

---- RATIONALE ----

A6: Why did you choose this design? In what ways is it superior to another design you considered?

We chose this design because it ensures synchronization and avoids racing conditions. It is also efficient in minimizing the amount of time spent in the timer interrupt as the timer_interrupt doesn’t need to check all the threads. This design was pretty straightforward to us thus no other design was considered.