Assignment - 5
Pintos Alarm Clock Design Document

Group - G7
Gurjot Singh Suri, 17CS10058
Kumar Abhishek, 17CS10022

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission, notes for the
>> TAs, or extra credit, please give them here.

>> Please cite any offline or online sources you consulted while
>> preparing your submission, other than the Pintos documentation, course
>> text, lecture notes, and course staff.

- Busy Waiting
  http://en.wikipedia.org/wiki/Busy_waiting

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.

Added a new member in struct thread to store ticks till which the thread should sleep.

```
struct thread {
        int64_t suspend_ticks;
};
```

A list ordered by suspend_ticks in ascending order used to store threads that are put to sleep.

```
static struct list sleep_list;
```

---- ALGORITHMS ----

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

In a call to timer_sleep(),

1. The current timer ticks are obtained.
2. Interrupts are disabled.
3. Current thread's suspend_ticks is set to the given sleep_ticks plus the current ticks.
4. Current thread is inserted into sleep_list in ascending order of suspend_ticks.
5. Block current thread and schedule next thread.
6. Reset interrupts level to its old one.

In timer interrupt handler,

1. Interrupts are disabled.
2. Iterates the sleep_list from front and checks if current thread is to be woken, it is popped from the list and is unblocked and then moves to the next thread. If current thread is to be woken up later or the list is empty, breaks.
3. Reset interrupts level to its old one.

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

An ordered list which is sorted and inserted by suspend_ticks is used, so that we can iterate the list from the beginning and stop whenever the suspend_ticks is larger that the current ticks, which guarantees the later threads in the sleep list don't need to be checked. By this means, we can minimize the time spent.

---- SYNCHRONIZATION ----

>> A4: How are race conditions avoided when multiple threads call
>> timer_sleep() simultaneously?
Interrupts are turned off when the blocking of a thread occurs, not allowing another thread to attempt to be put to sleep at the same time.

>> A5: How are race conditions avoided when a timer interrupt occurs
>> during a call to timer_sleep()?
As the interrupts are disabled in tiimer_sleep(), timer interrupt cannot occur so there is no chance of race conditions.

---- RATIONALE ----

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

It was important to keep a sorted list of all the threads that were currently sleeping because it takes a shorter amount of time to traverse a sorted list than alllist or an unsorted list of sleeping threads.
Also thread_block was used because it performed the functionality of taking a thread off of the ready list and yielding to the next scheduled thread.