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

---------- DATA STRUCTURES ----------

- Added a new attribute **int64_t sleep_ticks** to struct threads in file $HOME/pintos/src/thread/**thread.h**
  Justification : To maintain separate wakeup time values for every thread in kernel
- Added a function `static bool less_value(struct list_elem *a, struct list_elem *b` to file $HOME/pintos/src/devices/**timer.c**
  Justification : Comparator for wakeup times of thread structures to maintain the wakeup order
- Declared a global variable of type struct list thread_list in file $HOME/pintos/src/devices/**timer.c**
  Justification : The list thread_list maintain the list of blocked threads

---------- ALGORITHMS ----------

***A2**: Briefly describe what happens in a call to timer_sleep(),*
timer_init() : Called list_init() to initialize thread_list.
Implementation : **timer_sleep() -**

- Set sleep time
- ASSERT that external interrupts are enabled
- Disable the interrupts
- Push in list in sorted order
- Context switch
- Enable the interrupts

including the effects of the timer interrupt handler.
interrupt _handler :

- Context switches threads from blocked to ready queue if the sleep time is over

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

- Tads are maintained in a priority queue with priority assigned with respect to the sleep_times.

---------- SYNCHRONIZATION ----------

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

1. ASSERT (intr_get_level == INTR_ON)

The above chunk of code prevents any race conditions that might happen when multiple threads are trying to access the critical section in the timer_sleep() function.
ASSERT() call waits till some other thread enables the external interrupts.

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

2. intr_disable();
3. /*
4. Critical Section
5. */
6. intr_enable();

We disable any external interrupts before entering the critical section which blocks any timer interrupt calls. After processing the critical section we enable the external interrupts.

---------- RATIONALE ----------

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

1. It ensures synchronization avoiding race conditions
2. Compared to the other design considered (Design 1) it takes minimum amount of context switches.

Previous Designs Considered :
Design 1 : Sleep for a constant period of time (T_SLEEP) :
Implementation :

```
1. while( timer_elaspsed(start)< ticks ){
2.         thread_sleep(T_SLEEP);        // sleep for a T_SLEEP mseconds
3. }
```

Drawbacks :

1. For threads taking large CPU times, it takes [CPU_Time / T_SLEEP] context switches to put the process to sleep.