먼저 test의 주 목적인 **timer_sleep()** 함수를 살펴 봄
어떤 user code 내에서 timer_sleep(int64, ticks) 는
tick수 만큼 이 함수를 호출한 thread를 block시킴

```
/* Sleeps for approximately TICKS timer ticks.  Interrupts must
   be turned on. */
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  /* KNU-COMP312 HINT (Alarm Clock):
     This busy-wait loop wastes CPU time and must be removed.
     Instead, you should implement a mechanism to put the thread to sleep.

     [Steps]
     1. Maintain a sleep list to track sleeping threads and their wake-up times.
     2. Insert the current thread into the sleep list with its wake-up time.
        (You may want to keep the list sorted by wake-up time.)
     3. Block the thread using thread_block().
     4. In your timer interrupt handler, wake up threads whose time has come.

     This approach allows the CPU to do useful work instead of busy-waiting. */

  // while (timer_elapsed (start) < ticks)
  //   thread_yield ();

  /* 이 코드까지 진행 하기전 tick으로 정해진 시간을 다 지났으면 sleep 하지 않음 */
  if(timer_elapsed (start) < ticks)
    thread_sleep(start+ticks);
}
```

busy wating 코드를 지워 놓음

변수 :
start = timer_ticks (); 현재의 OS 의 tick(booting 후 tick수)
timer_elapsed (start) - thread가 start후 지난 tick
ticks - block()할 시간을 지정한 tick수

thread_sleep 함수 = thread.c

```
void
thread_sleep (int sleep_time)
{
```

```
  struct thread *cur = thread_current();
  enum intr_level old_level = intr_disable();

  ASSERT(!intr_context());

  cur->wakeup_time = wakeup_time;
  list_insert_ordered(&sleep_list, &cur->elem, compare_wakeup_time, NULL);
  cur->status = THREAD_BLOCKED;
  schedule();

  intr_set_level(old_level);
}
```

알 수 있는 것
thread 구조체에 **wakeup_time** 이라는 변수가 필요
**sleep_list**라는 잠자는 thread를 저장하기 위한 list 가 필요
thread 구조체에 (sleep_list안의 node 정보)를 담기 위한 elem 라는 filed 필요
sleep_list는 순서대로 저장 - compare_wakeup_time 에 의해 정렬됨
compare_wakeup_time이라는 callback 함수 필요

thread 구조체 ( thread.h )

```
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purposes). */
    uint8_t *stack;                     /* Saved stack pointer. */
    int priority;                       /* Priority. */
    struct list_elem allelem;           /* List element for all threads list. */

    /* KNU-COMP312 HINT (Alarm Clock):
       To implement sleeping behavior in timer_sleep() without busy waiting,
       consider adding a field here to store the thread's wake-up tick. */

    /* KNU-COMP312 HINT (Advanced Scheduler - 4.4BSD MLFQ):
       You may want to add fields to store each thread's nice value and recent_cpu,
       as required by the 4.4BSD scheduler.

       Note that recent_cpu is a real number, but Pintos does not support floating-point arithmetic.
       You must use fixed-point arithmetic, as provided in fixed-point.h. */

    /* Shared between thread.c and synch.c. */
    struct list_elem elem;              /* List element. ready_list용임 나중에 나옴*/

#ifdef USERPROG
```

```
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /* Page directory. */
#endif

    /* Owned by thread.c. */
    unsigned magic;                     /* Detects stack overflow. */
  };
```

구조를 보면 두 개의 list를 위한 node 필드가 존재
1) allelem
2) elem  - 이것을 ready_list 용임 나중에 설명 있음

sleep list용 새로운 node 저장용 변수를 만들어야 한다

((작업 시작))
thread.h
thread 구조체에 wakeup_time 필드 추가

thread.c
sleep_list 추가 및 thread_init에 초기화 과정 추가

```
  /* 추가 => 나중에는 형태가 바뀜 */
  /* 나중에 timer.c에서도 쓰기 위해 */
  /* 여기서는 struct list sleep_list,  thread.h 에 extern struct list sleep_list */
  /* timer.c 에서는 thread.h를 include 해서 사용.  지금은 그냥 이렇게 */
  static struct list sleep_list;

void
thread_init (void)     // 이 함수는 booting 되면서 시작되는 것인지? 그래서 모든 list를 초기화?
{
  ASSERT (intr_get_level () == INTR_OFF);

  lock_init (&tid_lock);
  list_init (&ready_list);
  list_init (&all_list);
  list_init (&sleep_list)

  /* Set up a thread structure for the running thread. */
  initial_thread = running_thread ();
  init_thread (initial_thread, "main", PRI_DEFAULT);
  initial_thread->status = THREAD_RUNNING;
  initial_thread->tid = allocate_tid ();
}
```

thread.h  의 thread structure에 node 추가

```
  struct list_elem sleep_elem;  /*  thread가 sleep_list안에 추가될 때 해당 node */
```

thread.c 의 thread_sleep 수정

```
thread_sleep (int sleep_time)
{
  struct thread *cur = thread_current();
  enum intr_level old_level = intr_disable();

  ASSERT(!intr_context());

  cur->wakeup_time = sleep_time; /* 매개 변수로 넘어온 것으로 바꿔 줌 */
  /* 비교할 해당 node 사용으로 변경 */
  list_insert_ordered(&sleep_list, &cur->sleep_elem, compare_wakeup_time, NULL);

  //cur->status = THREAD_BLOCKED;
  //schedule();
  /* 위 두줄을 쓸 수도 있고, thread_block() 함수를 써도 되는데 실제로는 thread_block()이 더 안전
     하다는 chatgpt 말씀임,  일단 그대로 둠  오류가 있으면 thread_block()으로 대체, 난 대체하였음 */
  thread_block();

  intr_set_level(old_level);
}
```

thread.c 및 thread.h
compare_wakeup_time callback 함수 작성 및 thread.h 에 선언하자

실제로 sleep_list의 element 두 개 주면 두 개의 해당 thread를 찾아 wakeup_time을 비교해 주는 함수 작성

```
bool compare_wakeup_time(const struct list_elem *a,
                         const struct list_elem *b,
                         void *aux UNUSED) {
  struct thread *ta = list_entry(a, struct thread, sleep_elem);
  struct thread *tb = list_entry(b, struct thread, sleep_elem);
  return ta->wakeup_time < tb->wakeup_time;
}
```

수정된 thread 구조체

```
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purposes). */
    uint8_t *stack;                     /* Saved stack pointer. */
    int priority;                       /* Priority. */
```

```
    struct list_elem allelem;           /* List element for all threads list. */

    /* KNU-COMP312 HINT (Alarm Clock):
       To implement sleeping behavior in timer_sleep() without busy waiting,
       consider adding a field here to store the thread's wake-up tick. */

    int64_t wakeup_time; // 깨어나야 할 시간 tick값

    /* KNU-COMP312 HINT (Advanced Scheduler - 4.4BSD MLFQ):
       You may want to add fields to store each thread's nice value and recent_cpu,
       as required by the 4.4BSD scheduler.

       Note that recent_cpu is a real number, but Pintos does not support floating-point arithmetic.
       You must use fixed-point arithmetic, as provided in fixed-point.h. */

    /* Shared between thread.c and synch.c. */
    struct list_elem elem;              /* List element. */
    struct list_elem sleep_elem;  /* List element for sleep_list */

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /* Page directory. */
#endif

    /* Owned by thread.c. */
    unsigned magic;                     /* Detects stack overflow. */
  };
```

## 참고 원형들

```
#1 void list_insert_ordered(struct list *list, struct list_elem *elem,
                            list_less_func *less, void *aux)
```

-- list에 순서대로 입력,
-- thread의 list_elem을 list에 삽입(이중 list구조)
-- list_less_func을 이용해 정렬

```
#2 typedef bool list_less_func (const struct list_elem *a,
                                const struct list_elem *b,
                                void *aux);

#3 #define list_entry(LIST_ELEM, STRUCT, MEMBER) \
  ((STRUCT *) ((uint8_t *) (LIST_ELEM) - offsetof(STRUCT, MEMBER)))
```

기능 LIST_ELEM이라는 포인터가 STRUCT 구조체의 MEMBER라는 필드 안에 있다고 할 때,
그 MEMBER가 속한 원래 구조체 STRUCT *를 역으로 찾아주는 거

지금까지 code는

1. 유저 프로그램에서 timer_sleep(int64_t ticks) 로 얼마를 재울지 결정하기 위해 실행하면
2. threa.c 의 thread_sleep을 이용하여 thread_block()을 시행하고
    sleep_list에 thread를 정렬해서 넣고 thread->wakeup_time을 설정

이제 code는 어떻게 언제 깨울지를 coding 해야 함

timer_interrupt()를 이용, timer_interrupt()가 실행 될 때 마다 sleep_list에서 깰 시간이 된 thread들을 꺼집어 내고, thread_unblock()실행 == thread를 활성화 하고 ready_list에 넣음

device/timer.c  새로 고친 code

```
/* Timer interrupt handler. */
static void
timer_interrupt (struct intr_frame *args UNUSED)
{
  ticks++;

  /* KNU-COMP312 HINT (Alarm Clock):
     Check if any threads in the sleep list need to be woken up.

     [Steps]
     1. Iterate through the sleep list and find threads whose wake-up time has arrived (i.e., wake-up time <= current tick).
     2. Remove those threads from the sleep list.
     3. Unblock them using thread_unblock() so they can resume execution.

     This ensures sleeping threads are resumed at the correct time. */
     enum intr_level old_level = intr_disable();

     while (!list_empty(&sleep_list)) {
       struct thread *t = list_entry(list_front(&sleep_list), struct thread, sleep_elem);
       if (t->wakeup_time <= ticks) {
         list_pop_front(&sleep_list);
         thread_unblock(t);
       } else {
         break;
       }
     }

     intr_set_level(old_level);

     thread_tick ();
}
```

정렬되어 있으니 앞에부터 깨우고, 안 깨울 놈이 나오면 break함

sleep_list 변수 관련

sleep_list는 이전에 thread.c에 정의 해 놓음 static로
여기서도 사용이 필요하므로
thread.c에서 static을 제거 struct list sleep_list
timer.c에서 사용하기 위해 thread.h에 extern 선언 extern struct list sleep_list
timer.c에서 thread.h include 해서 사용

sleep_list를 건드리는 동안 또 timer_interrupt가 걸리면 문제가 생길 수 있으니
enum intr_level old_level = intr_disable(); /* 현재 interrupt level저장 및 disable */
    while(.....) {.....  }
int_set_level(old_level) /* 다시 원래 상태로 복구 */
##  참고
enum intr_level {
        INTR_OFF,
        INTR_ON
}
여기까지 해서 test를 하면
make check | grep alarm
 -->다른 test는 성공
 -- alarm-priority test에서 첫 바퀴는 성공
    두 번 째 바퀴에서 실패
    원인은 동시에 여러 thread가 깨어 났을 때 thread의 우선 순위에 따라 실행 되어야 하나
       현재 algorhythm에는 포함 되어 있지 않음

(( 해결 ))
original thread_unblock() 함수  - Hint 참고
 /* KNU-COMP312 HINT (Priority Scheduling):
      thread_unblock() changes the thread's state from BLOCKED to READY
      and adds it to the ready list.

      To support priority scheduling, the ready list should be ordered by thread priority,
      so that the highest-priority thread runs first.

      Consider inserting the thread in sorted order, or sorting the list after insertion.

      Likewise, for priority scheduling to work correctly,
      you may also need to apply similar logic in thread_yield() and next_thread_to_run(). */

thread 구조체에 있는 elem은 어느 list와 관련 있을까?

thread.c  next_thread_to_run (void) {
        if( list_empty(&ready_list))
                return idle_thread;
        else
                return list_entry( list_pop_front ( &ready_list ), struct thread, elem );
}
이 코드를 참조하면 thread구조체의 elem은 ready_list와 관련 있고. ready_list중 제일 앞의 thread부터 실행

시킨다.

따라서 우리는 ready_list에 thread를 넣을 때 우선 순위 대로 넣으면 되겠다는 결론임.
그리고 list를 다룰 때 node는 이미 만들어 져 있는 elem을 이용하면 된다.

과정1) thread.c thread_unblock() 함수를 수정 unblock시 우선 순서대로 ready_list에 추가
list_push_back(....)을 제거하고
list_insert_ordered를 사용

list_insert_ordered(&ready_list, &t->elem, thread_priority_more, null);

당연히 thread.c 및 h에를 추가

```
bool thread_priority_more(const struct list_elem *a,
                          const struct list_elem *b,
                          void *aux UNUSED)
{
        const struct thread *t_a = list_entry(a, struct thread, elem);
        const struct thread *t_b = list_entry(b, struct thread, elem);
        return t_a->priority > t_b->priority;  /* priority : thread 구조체 안의 변수 */
}
```

과정 2) preemption 구현
깬 thread가 현재 실행중인 thread보다 우선 순위가 높으면
-> 현재 실행 중인 thread를 중지시키고 대기열에 넣음 thread_yield()
thread_yield에서 대기열에 넣을 때도 list_insert_ordered를 사용해야 한다
2)-1 thread_unblock()에 추가 - unblock된 thread의 priority가 현재 실행 thread보다 높으면
현재 실행 thread를 yield() 함

```
void
thread_unblock (struct thread *t)
{
  enum intr_level old_level;

  ASSERT (is_thread (t));

  old_level = intr_disable ();
  ASSERT (t->status == THREAD_BLOCKED);
  /* KNU-COMP312 HINT (Priority Scheduling):
     thread_unblock() changes the thread's state from BLOCKED to READY
     and adds it to the ready list.

     To support priority scheduling, the ready list should be ordered by thread priority,
     so that the highest-priority thread runs first.

     Consider inserting the thread in sorted order, or sorting the list after insertion.

     Likewise, for priority scheduling to work correctly,
```

```
      you may also need to apply similar logic in thread_yield() and next_thread_to_run(). */
  // list_push_back (&ready_list, &t->elem);
  list_insert_ordered(&ready_list, &t->elem, thread_priority_more, NULL);  //     높은 priority가 앞으로

  t->status = THREAD_READY;

  // Preemption
  if (thread_current() != idle_thread && t->priority > thread_current()->priority)
    thread_yield();

  intr_set_level (old_level);
}
```

2)-2 thread_yield() 수정

- yield된 thread는 앞에 만들어 놓은 thread_priority_more callback function을 이용 정렬해서 다시 ready_list에 넣는다.

```
/* Yields the CPU.  The current thread is not put to sleep and
   may be scheduled again immediately at the scheduler's whim. */
void
thread_yield (void)
{
  struct thread *cur = thread_current ();
  enum intr_level old_level;

  ASSERT (!intr_context ());

  old_level = intr_disable ();

  // if (cur != idle_thread)
  //   list_push_back (&ready_list, &cur->elem);
  if (cur != idle_thread)
    list_insert_ordered (&ready_list, &cur->elem, thread_priority_more, NULL);

  cur->status = THREAD_READY;
  schedule ();
  intr_set_level (old_level);
}
```

자 이제 다시 test 해봄

```
make check | grep alarm
```

☺☺☺☺ 결과는 모두 성공 ☺☺☺