---- GROUP ----

Ahmed Khalil ElZeiny (20010087)

Ahmed Samir Elsayed (20010107)

Moahmed Wael Fathy (20011752)

Youssef Hossam AboElwafa (20012263)

Youssef Mohamed ElMedany (20012293)

ALARM CLOCK

=============

---- DATA STRUCTURES ----

In (thread.h):

Add two new attributes to represent the remaining ticks before waking up the thread.

1. `sleepelem` is the list element for sleeping threads.
2. `remaining_time_to_wake_up` is ticks remaining from waking up, with an initial value 0.

```
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. */

    struct list_elem sleepelem;        /* List element for sleeping threads. */
    int64_t remaining_time_to_wake_up; /* Ticks remaining from waking up. */

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

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

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

---- ALGORITHMS ----

Another list is created to hold the sleeping threads and is initialized with thread init function as well as ready_list and all_list

```
void thread_init(void)
{
  ASSERT(intr_get_level() == INTR_OFF);

  lock_init(&tid_lock);
  list_init(&ready_list);
  list_init(&all_list);
  list_init(&sleeping_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();
}
```

When calling `timer_sleep(int64_t ticks)`function it had a busy waiting so it is removed and replaced with another function `thread_set_sleeping(ticks)` when a thread is put to sleep for a given number of ticks

before (busy waiting):

```
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

after:

```c
void timer_sleep(int64_t ticks)
{

  int64_t start = timer_ticks();

  ASSERT(intr_get_level() == INTR_ON);
  if(ticks<=0){
    return;
  }
  intr_disable();
  thread_set_sleeping(ticks);
  intr_set_level(INTR_ON);
}
```

when calling `thread_set_sleeping(int64_t ticks)` the thread is inserted in descending order w.r.t its priority into the sleeping_list instead of push_back so if there is 2 threads with (remaining_time_to_wake_up) =0

→ then the thread with the highest priority is unblocked first

```c
void thread_set_sleeping(int64_t ticks)
{
  struct thread *cur = thread_current();
  cur->remaining_time_to_wake_up = ticks;
  enum intr_level old_level = intr_disable();
  list_insert_ordered(&sleeping_list, &cur->sleepelem, compare_threads_by_priority_sleeping, NULL);
  intr_set_level(old_level);
  thread_block();
}
```

This while loop in inserted in (thread_tick) that is called in the (timer_interrupt) handler

```
/* Iterate through all sleeping threads in SLEEPING LIST, decrease the
REMAINING TIME TO WAKE UP of these threads by 1. If any of them have a
zero REMAINING TIME TO WAKE UP, wake up these threads. */
struct list_elem *e = list_begin(&sleeping_list);
struct list_elem *temp;

while (e != list_end(&sleeping_list))
{
  struct thread *t = list_entry(e, struct thread, sleepelem);
  temp = e;
  e = list_next(e);

  ASSERT(t->status == THREAD_BLOCKED);

  if (t->remaining_time_to_wake_up > 0)
  {
    t->remaining_time_to_wake_up--;
    if (t->remaining_time_to_wake_up <= 0)
    {
      thread_unblock(t);
      temp = list_remove(temp);
    }
  }
}
```

It iterate through all sleeping threads in sleeping_list , decrease the

REMAINING TIME TO WAKE UP of these threads by 1 with every tick.

If any of them have a zero REMAINING TIME TO WAKE UP, wake up this thread.

---- SYNCHRONIZATION ----

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

With the interrupt disabled during the thread operation, this function is almost "atomic".

```
intr_disable();
```

```
...
```

```
intr_set_level(INTR_ON);
```

---- RATIONALE ----

I have thought about just making all_list to replace sleeping_list, because all alive threads are naturally in all_list, then if this design is taken, there will be no needs to insert put sleeping threads in another place.

But the iteration over all_list to find the sleeping thread and check the (remaining_time_to_wake_up) will be time consuming.

PRIORITY SCHEDULING

==================

---- DATA STRUCTURES ----

>> B1: 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.

>> B2: Explain the data structure used to track priority donation.

>> Use ASCII art to diagram a nested donation. (Alternately, submit a

>> .png file.)

---- ALGORITHMS ----

>> B3: How do you ensure that the highest priority thread waiting for

>> a lock, semaphore, or condition variable wakes up first?

>> B4: Describe the sequence of events when a call to lock_acquire()

>> causes a priority donation. How is nested donation handled?

>> B5: Describe the sequence of events when lock_release() is called

>> on a lock that a higher-priority thread is waiting for.

---- SYNCHRONIZATION ----

>> B6: Describe a potential race in thread_set_priority() and explain

>> how your implementation avoids it. Can you use a lock to avoid

>> this race?

---- RATIONALE ----

>> B7: Why did you choose this design? In what ways is it superior to
>> another design you considered?

ADVANCED SCHEDULER

==================

---- DATA STRUCTURES ----

>> C1: 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.

---- ALGORITHMS ----

>> C2: Suppose threads A, B, and C have nice values 0, 1, and 2. Each

>> has a recent_cpu value of 0. Fill in the table below showing the

>> scheduling decision and the priority and recent_cpu values for each

>> thread after each given number of timer ticks:

```
timer  recent_cpu    priority   thread
ticks   A   B   C   A   B   C   to run
-----  --  --  --  --  --  --   ------
 0
 4
 8
12
16
20
24
28
32
36
```

>> C3: Did any ambiguities in the scheduler specification make values
>> in the table uncertain? If so, what rule did you use to resolve
>> them? Does this match the behavior of your scheduler?

>> C4: How is the way you divided the cost of scheduling between code
>> inside and outside interrupt context likely to affect performance?

---- RATIONALE ----

>> C5: Briefly critique your design, pointing out advantages and
>> disadvantages in your design choices. If you were to have extra
>> time to work on this part of the project, how might you choose to
>> refine or improve your design?

>> C6: The assignment explains arithmetic for fixed-point math in
>> detail, but it leaves it open to you to implement it. Why did you
>> decide to implement it the way you did? If you created an
>> abstraction layer for fixed-point math, that is, an abstract data
>> type and/or a set of functions or macros to manipulate fixed-point
>> numbers, why did you do so? If not, why not?