Submitted by Amin Jafarpour, A01225133 and Nicky Cheng, A01269051

Section 0

Github repo link

https://github.com/amin-jafarpour/COMP-7035-assignment1-part2

Section 1

I use semaphores to put threads to sleep and wake them up. I initialize a counting semaphore to zero for every thread that wants to sleep. And I use the semaphore down function to block the threads that want to sleep. And then, when it is time to wake up a sleeping thread, I use semaphore up function to unblock that thread. I also disable interrupts when I am dealing with global variables to ensure no race condition occurs.

Open output.txt file to see the test results.

Open timer.c file to see the my code snippet along with original code that came with pintOS.

Open snippet.c to view the source code

Source code is also available below:

```
//NOTES:
// 1) Comment out the original `timer_sleep` function in devices/timer.c
// 2) Comment out the original `timer_interrupt` function in devices.timer.c
// 3) Insert `sleeping_threads_init()` line of code at the very end of `timer_init`
function in devices/timer.c
// 4) Insert this code snippet after the last function prototype in the file
devices/timer.c



// A struct that contains the necessary data to wake up a sleeping thread.
struct sleeping_thread
{
  int64_t wake_up_time;
  // A counting semaphore
  struct semaphore sema;
  struct list_elem elem;
};

// A list that contains all threads currently sleepings. Data structure used from
lib/kernel/list.c
static struct list sleeping_threads;

//The new implementation of timer_interrupt function; replaces the existing
timer_interrupt function within devices/timer.c
static void timer_interrupt (struct intr_frame *args UNUSED);

// Contains the code within original timer_interrupt function and will be called by
the new implementation timer_interrupt function.
```

```
static void original_timer_interrupt(struct intr_frame *args UNUSED);

//Takes two sleeping threads and returns true if the first one has lower (a < b)
sleep timer than the second one and false otherwise.
bool compare_wake_up_time (const struct list_elem *a, const struct list_elem *b,
void *aux UNUSED);

//NOTE:call it within timer_init of devices/timer.c
// Initializes the sleeping thread list according to specs in lib/kernel/list.h
void sleeping_threads_init (void);


void timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();
  struct sleeping_thread st;
  st.wake_up_time = start + ticks;
  // Initialize the value of the semphore to 0.
  sema_init(&st.sema, 0);
  enum intr_level old_level = intr_disable ();
  // Insert the current thread into the list of sleeping threads and sort the list
based on thw value of wake up times.
  list_insert_ordered(&sleeping_threads, &st.elem, (list_less_func *)
&compare_wake_up_time, NULL);
  intr_set_level(old_level);

  //MYNOTE: might move to where interrupt is disabled above
  //block the current thread
  // Down or "P" operation on a semaphore.
  // Waits for SEMA's value to become positive and then atomically decrements it.
  // This function may sleep, so it must not be called within an interrupt handler.
  //This function may be called with interrupts disabled, but if it sleeps then the
next scheduled
  //thread will probably turn interrupts back on.
  sema_down(&st.sema);
}


bool compare_wake_up_time (const struct list_elem *a, const struct list_elem *b,
void *aux UNUSED)
{
  struct sleeping_thread *st_a = list_entry(a, struct sleeping_thread, elem);
  struct sleeping_thread *st_b = list_entry(b, struct sleeping_thread, elem);
  return st_a->wake_up_time < st_b->wake_up_time;
}


static void timer_interrupt (struct intr_frame *args UNUSED)
{
  // Run the code of the original timer_interrupt function
  original_timer_interrupt(args);

  enum intr_level old_level = intr_disable ();

  while (!list_empty(&sleeping_threads))
    {
      // Take thread with least amount of sleep timer
      struct list_elem *e = list_front(&sleeping_threads);
```

```
      struct sleeping_thread *st = list_entry(e, struct sleeping_thread, elem);

      //
      if (st->wake_up_time > timer_ticks())
        break;

      // Remove the thread from sleeping thread list
      list_remove(e);

      // Wake up the thread
      // Up or "V" operation on a semaphore.
      // Increments SEMA's value and wakes up one thread of those waiting for SEMA,
if any.
      // This function may be called from an interrupt handler.
      sema_up(&st->sema);
    }

  intr_set_level(old_level);
}


static void original_timer_interrupt(struct intr_frame *args UNUSED)
{
  ticks++;
  thread_tick ();
}


void sleeping_threads_init (void)
{
  list_init(&sleeping_threads);
}
```

Section 2

Video link youtube

https://youtu.be/eEhLmiojRgY

Section 3

Amin Jafarpour, A01225133 : Source code implementation, debugging, video, documentation, Github repo.

Nicky Cheng, A01269051: Report.

Section 4

Initially Amin tried to implement the solution using thread `thread_block` and `thread_unblock` functions, but this approach failed as it resulted in page fault errors. Amin was unable to debug this solution.

So, Amin decided to implement a second solution using semaphores and this second solution works just fine.