OS Project1 Final Report

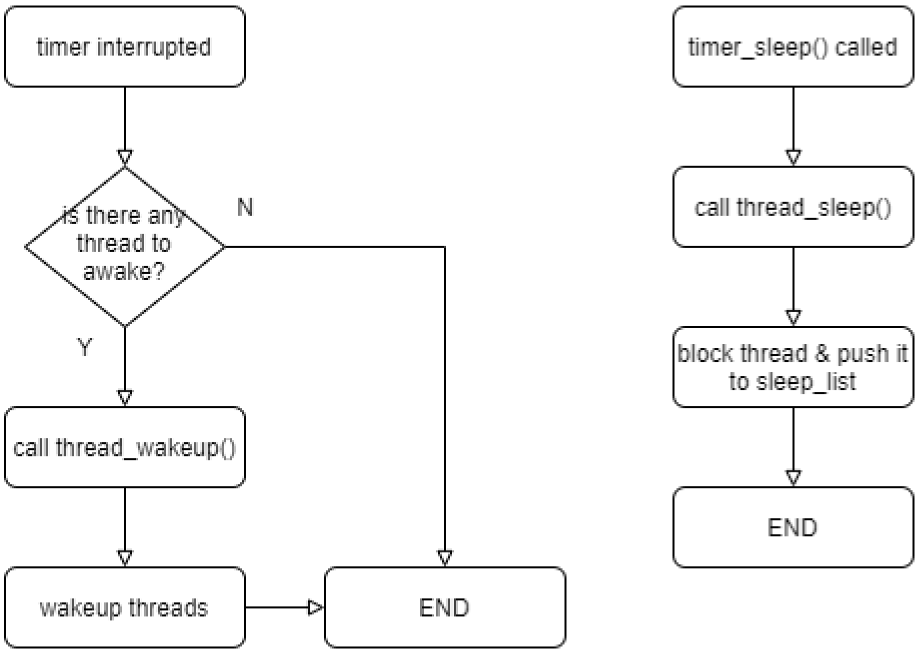
letsgo_seuljongbin
20180038 박형규, 20180480 성창환

1. Introduction

OS Project1을 통해 Pintos Alarm Clock, Priority Scheduling, Advanced Scheduler를 모두 구현해 보았다. 이번 Final Report에서는 각 항목별로 Flow Chart, 추가하거나 수정한 Function & Data Structure, Implementation에 대해 자세히 작성하였다. 특히 Implementation 부분에서는 Design Report와 비교하여 작성하였는데 대부분 Design Report와 동일하지만 Design Report에서 언급되지 않았거나 바뀐 부분이 있다면 해당 부분을 언급하였다.

2. Alarm Clock

2.1. Flow Chart

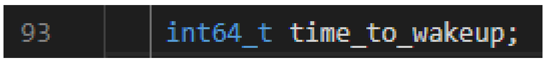


2.2. Added or modified Function & Data Structure

- Modified Data Structure
  struct thread, static int64_t first_time_to_wakeup, static struct list sleep_list
- Added Function
  get_first_time_to_wakeup(), thread_sleep(), thread_wakeup()
- Modified Function
  thread_init(), timer_sleep(), timer_interrupt()

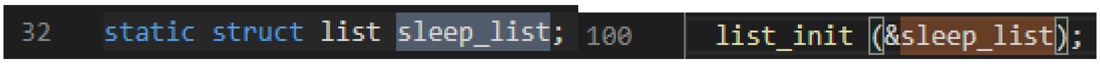
2.3. Implementation

우선 struct thread에서 thread를 깨워야 할 tick을 저장할 변수 time_to_wakeup을 선언했다.



```c
	int64_t time_to_wakeup;
```

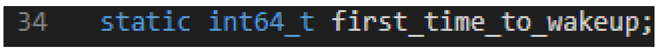
다음으로 sleep상태의 thread들을 관리할 sleep_list를 선언하고 thread_init()에서 초기화 시켰다.



```c
static struct list sleep_list;
```

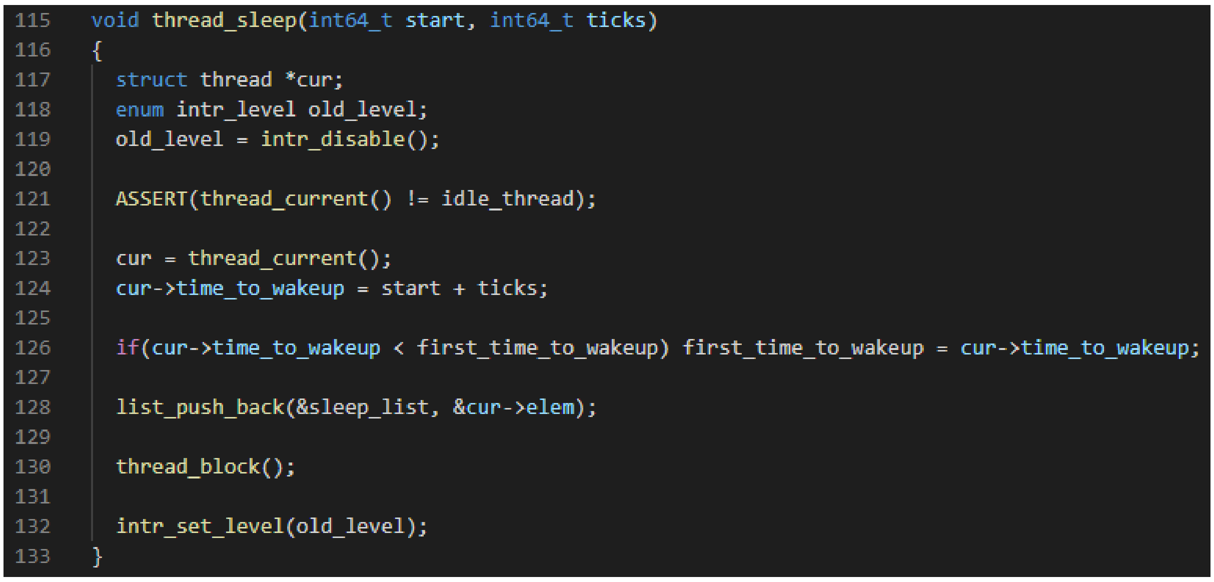
```c
	list_init (&sleep_list);
```

다음으로 sleep_list에 있는 thread들 중 가장 먼저 깨워야 하는 thread가 일어나야 할 시각을 저장하는 전역변수 first_time_to_wakeup을 선언했다.



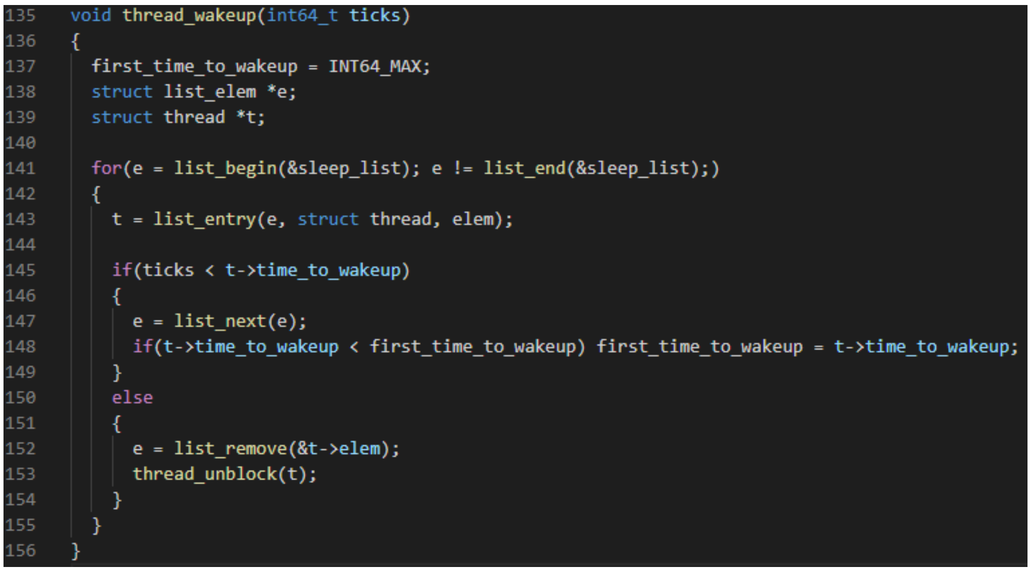
```c
static int64_t first_time_to_wakeup;
```

다음으로 thread를 sleep시키는 함수 thread_sleep()을 구현했다. thread_sleep()은 인자로 현재 tick(start)과 몇 tick동안 sleep시킬지(ticks)를 입력 받아서 current thread를 sleep시킨다. 이때, interrupt를 차단하고, time_to_wakeup과 first_time_to_wakeup에 언제 이 thread가 wakeup될지 저장한 다음 sleep_list에 push back해준다. 그런 다음 thread_block()을 해서 BLOCKED로 설정한다.



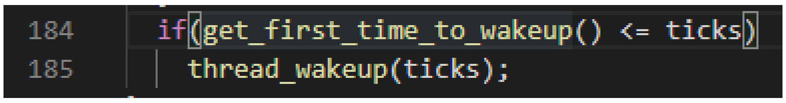
```c
void thread_sleep(int64_t start, int64_t ticks)
{
  struct thread *cur;
  enum intr_level old_level;
  old_level = intr_disable();

  ASSERT(thread_current() != idle_thread);

  cur = thread_current();
  cur->time_to_wakeup = start + ticks;

  if(cur->time_to_wakeup < first_time_to_wakeup) first_time_to_wakeup = cur->time_to_wakeup;

  list_push_back(&sleep_list, &cur->elem);

  thread_block();

  intr_set_level(old_level);
}
```

다음으로 sleep상태의 thread를 깨우는 함수 thread_wakeup을 구현했다. Sleep_list를 처음부터 끝까지 순회하면서 깨워야되는 thread들을 sleep_list에서 제거하고 unblock해준다. 아직 깨울 시간이 되지 않은 thread들은 넘어가고 first_time_to_wakeup을 다음에 깨워야하는 tick으로 설정하도록 구현했다.

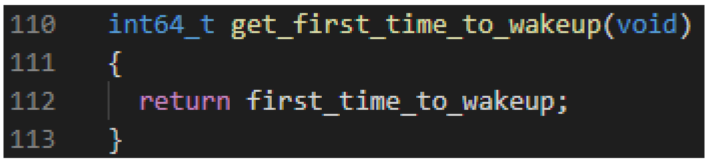


```c
void thread_wakeup(int64_t ticks)
{
  first_time_to_wakeup = INT64_MAX;
  struct list_elem *e;
  struct thread *t;

  for(e = list_begin(&sleep_list); e != list_end(&sleep_list);)
  {
    t = list_entry(e, struct thread, elem);

    if(ticks < t->time_to_wakeup)
    {
      e = list_next(e);
      if(t->time_to_wakeup < first_time_to_wakeup) first_time_to_wakeup = t->time_to_wakeup;
    }
    else
    {
      e = list_remove(&t->elem);
      thread_unblock(t);
    }
  }
}
```

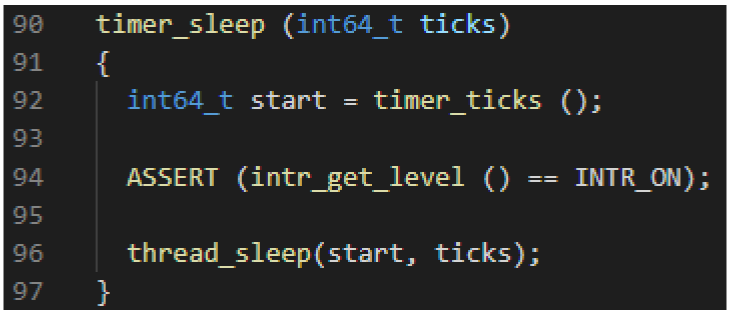
다음으로 thread_wakeup이 실행되게 하기위해 timer_interrupt를 수정했다.


```
  if(get_first_time_to_wakeup() <= ticks)
    thread_wakeup(ticks);
```

이때, 위와 같이 first_time_to_wakeup 값을 현재 tick과 비교하기 위해서 first_time_to_wakeup을 리턴해주는 get_first_time_to_wakeup()을 추가로 구현해 주었다.


```
int64_t get_first_time_to_wakeup(void)
{
  return first_time_to_wakeup;
}
```

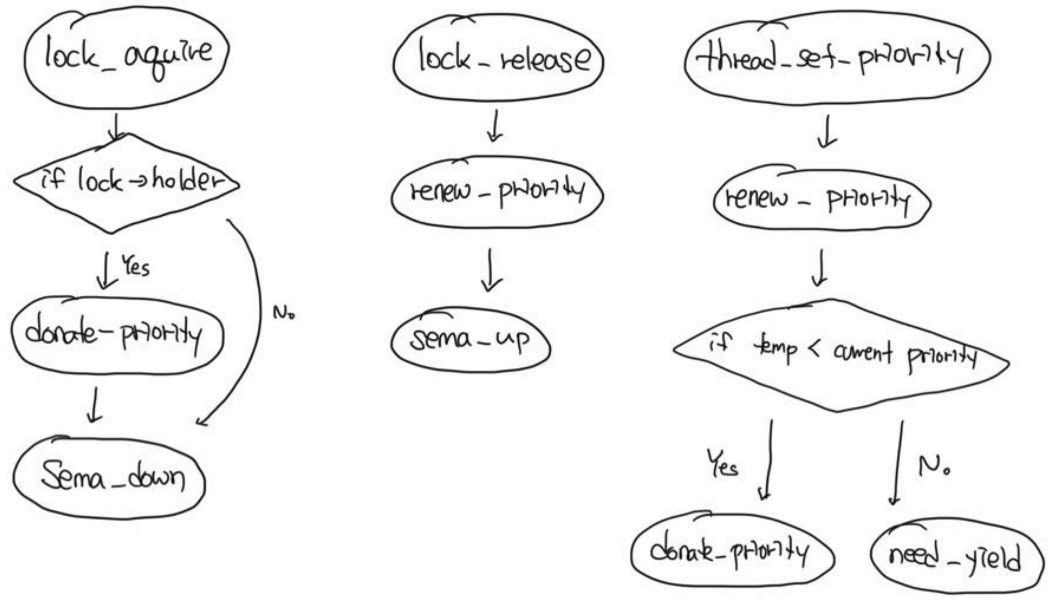
마지막으로 기본 pintos의 timer_sleep()에 구현되어 있던 busy waiting방식 코드를 제거하고, thread_sleep()을 호출하도록 수정했다.


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

  ASSERT (intr_get_level () == INTR_ON);

  thread_sleep(start, ticks);
}
```

3. Priority Scheduling

3.1. Flow Chart

<Priority Donation Flow Chart>



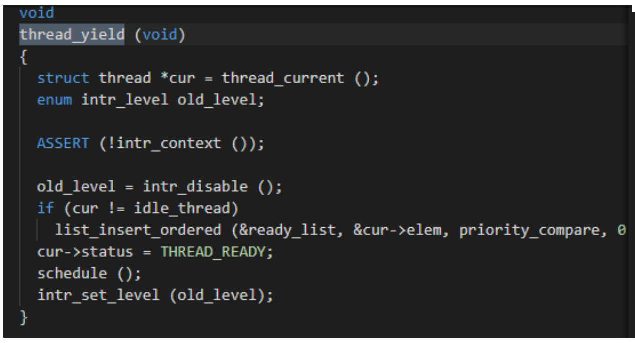
3.2. Added or modified Function & Data Structure

- Modified Data Structure
  struct thread
- Added Function
  priority_compare(), need_yield(), sema_priority_compare(), donate_priority(), renew_priority()
- Modified Function
  thread_yield(), thread_unblock(), thread_create(), thread_set_priority(), sema_down(), sema_up(), cond_wait(), cond_signal(), lock_aquire(), lock_release(), thread_set_priority(), thread_get_priority()

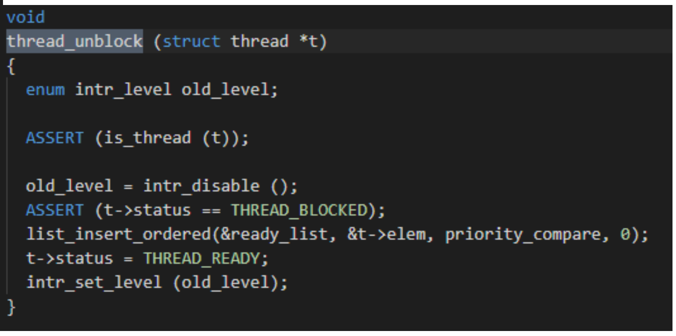
3.3. Implementation

- Priority Scheduling & Synchronization

Priority Scheduling은 Design Report에서 구현한 방법과 같이 구현하였다. 기존의 방법에서 thread_yield() 함수와 thread_unblock()함수에서 ready_list에 thread를 insert할 때 사용하던 방법인 list_push_back() 대신에 list_insert_ordered() 함수를 사용하여 ready_list 내부에서 thread의 우선순위를 기준으로 정렬되도록 하였다.

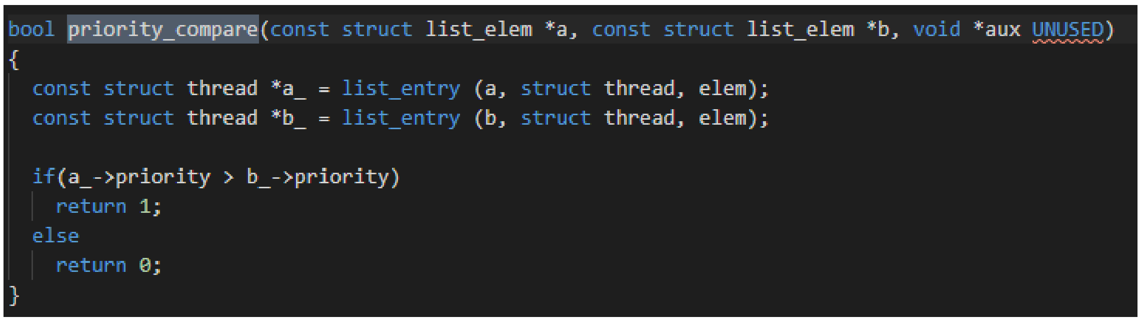


```c
void
thread_yield (void)
{
  struct thread *cur = thread_current ();
  enum intr_level old_level;

  ASSERT (!intr_context ());

  old_level = intr_disable ();
  if (cur != idle_thread)
    list_insert_ordered (&ready_list, &cur->elem, priority_compare, 0
  cur->status = THREAD_READY;
  schedule ();
  intr_set_level (old_level);
}
```



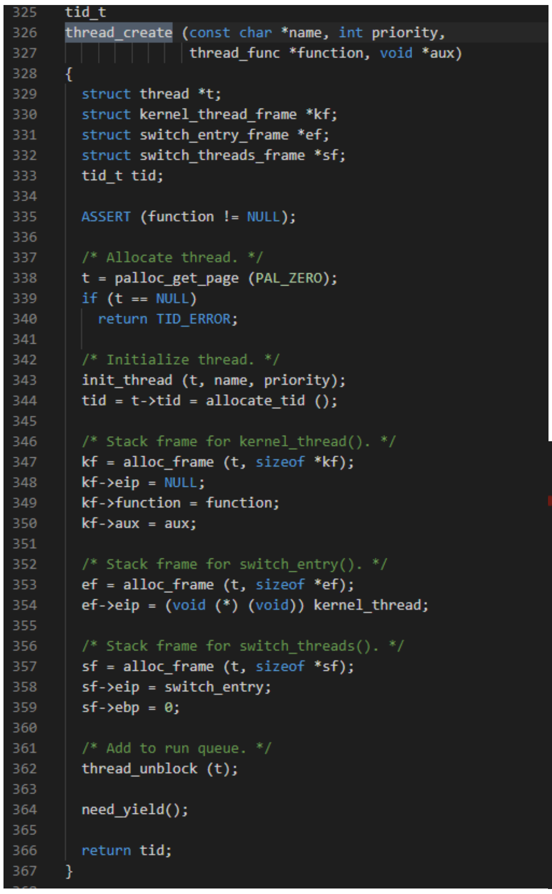
```c
void
thread_unblock (struct thread *t)
{
  enum intr_level old_level;

  ASSERT (is_thread (t));

  old_level = intr_disable ();
  ASSERT (t->status == THREAD_BLOCKED);
  list_insert_ordered(&ready_list, &t->elem, priority_compare, 0);
  t->status = THREAD_READY;
  intr_set_level (old_level);
}
```

이 때, list_insert_ordered() 함수의 인자로 list에 ordered하게 element를 집어넣어야 하므로 thread간의 priority를 비교하는 함수가 필요하다. 따라서 priority_compare() 함수를 구현하였다.

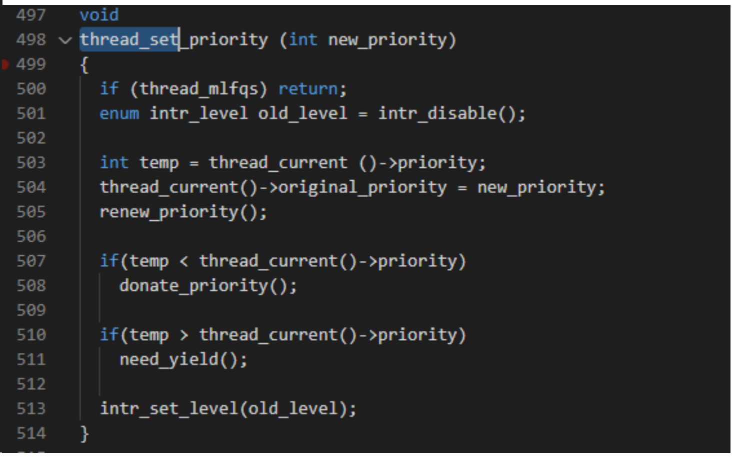


```c
bool priority_compare(const struct list_elem *a, const struct list_elem *b, void *aux UNUSED)
{
  const struct thread *a_ = list_entry (a, struct thread, elem);
  const struct thread *b_ = list_entry (b, struct thread, elem);

  if(a_->priority > b_->priority)
    return 1;
  else
    return 0;
}
```

다음으로 ready_list에서 가장 priority가 큰 thread(ready_list의 첫 번째 thread)보다 현재의 thread보다 priority가 높으면 양보를 하도록 해야 하는 부분을 thread_create() 부분의 마지막에 넣어야 한다고 Design Report에 언급하였다. 우리는 이 함수를 need_yield() 함수로 따로 만들어서 구현하였다. thread_create() 함수 아래에 이 부분을 써넣지 않고 따로 만든 이유는 이 부분이 thread_set_priority() 함수 내부에도 필요하기 때문이다. 이 부분은 Design Report의 Priority Donate부분에서 마지막 쯤에 '이 작업에서 current thread가 더이상 highest priority를 가지고 있지 않담녀(다면-오타) thread_yield()함수를 호출하는 작업이 필요하다.' 라고 언급되어 있다. 이 부분이 priority scheduling만을 구현하였을 때는 510, 511번째 줄만 구현되어 있었는데 이후에 Priority Donation을 구현하게 되면서 위 부분을 추가하게 된다. 이는 2.3에서 이어 설명하도록 하겠다.

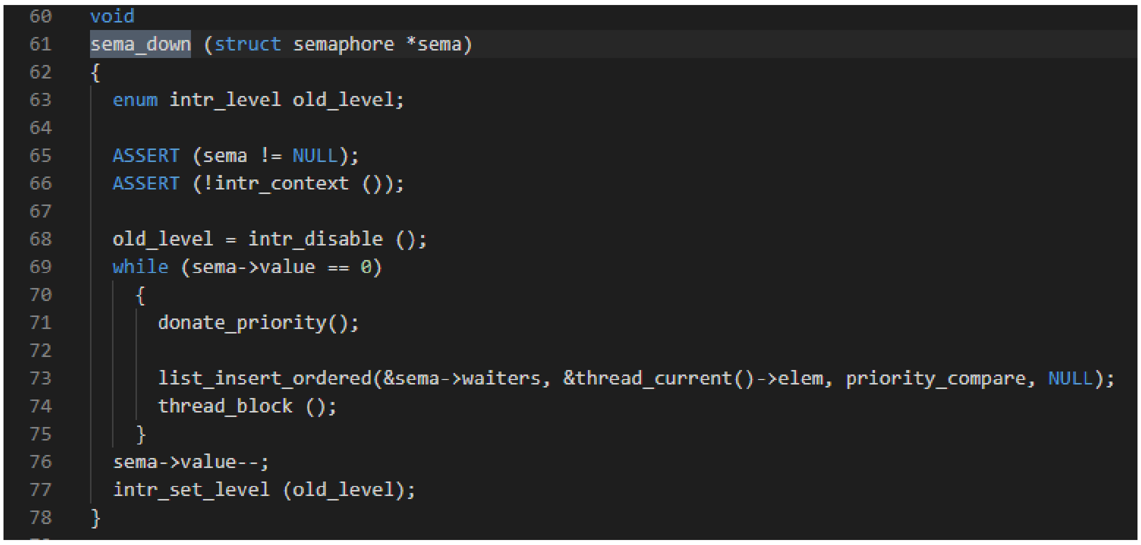


```c
tid_t
thread_create (const char *name, int priority,
               thread_func *function, void *aux)
{
  struct thread *t;
  struct kernel_thread_frame *kf;
  struct switch_entry_frame *ef;
  struct switch_threads_frame *sf;
  tid_t tid;

  ASSERT (function != NULL);

  /* Allocate thread. */
  t = palloc_get_page (PAL_ZERO);
  if (t == NULL)
    return TID_ERROR;

  /* Initialize thread. */
  init_thread (t, name, priority);
  tid = t->tid = allocate_tid ();

  /* Stack frame for kernel_thread(). */
  kf = alloc_frame (t, sizeof *kf);
  kf->eip = NULL;
  kf->function = function;
  kf->aux = aux;

  /* Stack frame for switch_entry(). */
  ef = alloc_frame (t, sizeof *ef);
  ef->eip = (void (*) (void)) kernel_thread;

  /* Stack frame for switch_threads(). */
  sf = alloc_frame (t, sizeof *sf);
  sf->eip = switch_entry;
  sf->ebp = 0;

  /* Add to run queue. */
  thread_unblock (t);

  need_yield();

  return tid;
}
```



```c
void
thread_set_priority (int new_priority)
{
  if (thread_mlfqs) return;
  enum intr_level old_level = intr_disable();

  int temp = thread_current ()->priority;
  thread_current()->original_priority = new_priority;
  renew_priority();

  if(temp < thread_current()->priority)
    donate_priority();

  if(temp > thread_current()->priority)
    need_yield();

  intr_set_level(old_level);
}
```

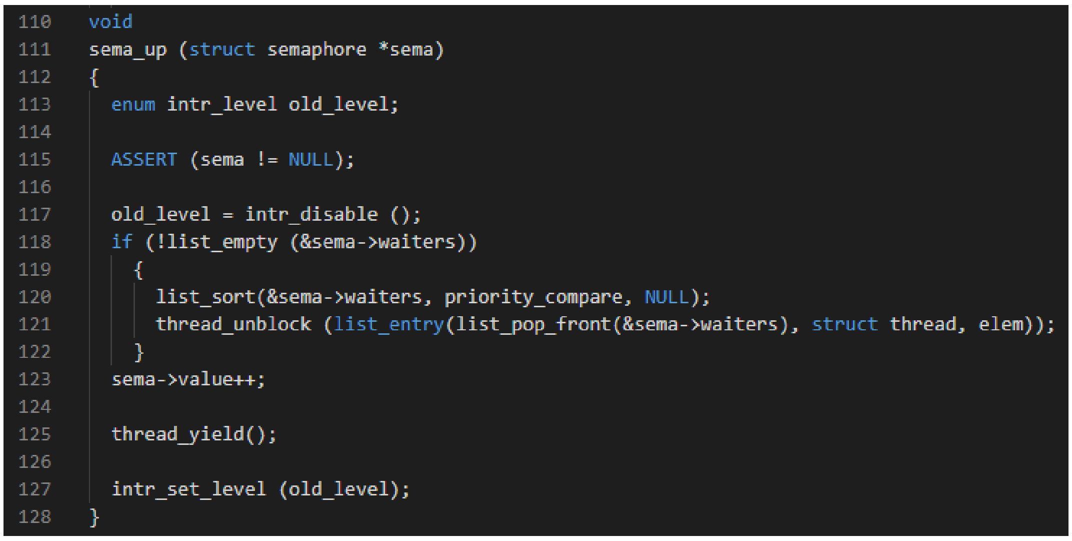
다음으로는 Synchronization을 구현한 방법에 대해 설명하도록 하겠다. Design Report에서는 sema_down() 함수나 sema_up() 함수 둘 중에 하나만 수정해도 된다고 하였고 우리 구현에서는 sema_down()을 수정하였다. 먼저 sema_down() 함수에서는 Design Report에서 언급한 것과 동일하게 waiters list에 element를 insert할 때 list_push_back() 함수 대신에 list_insert_ordered() 함수를 사용하였다. 아래 사진에서 71번째 줄 donate_priority()는 이후 priority donation 부분을 설명할 때 언급하도록 하겠다.



```c
60  void
61  sema_down (struct semaphore *sema)
62  {
63    enum intr_level old_level;
64
65    ASSERT (sema != NULL);
66    ASSERT (!intr_context ());
67
68    old_level = intr_disable ();
69    while (sema->value == 0)
70      {
71        donate_priority();
72
73        list_insert_ordered(&sema->waiters, &thread_current()->elem, priority_compare, NULL);
74        thread_block ();
75      }
76    sema->value--;
77    intr_set_level (old_level);
78  }
```

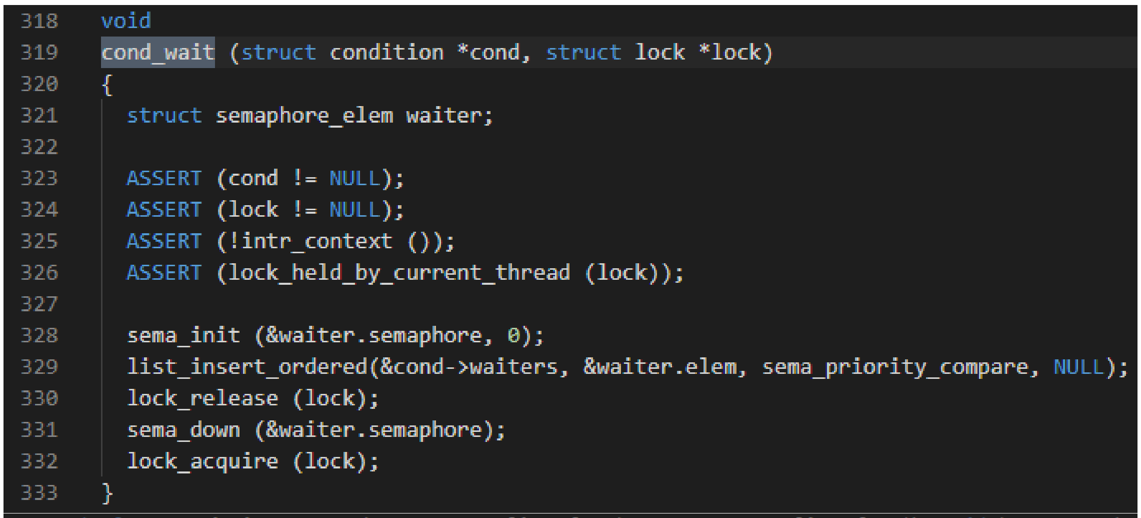
sema_up() 함수를 수정한 부분은 Design Report에 언급되어 있지 않다. 당시 Design Report를 작성할 때에는 미처 생각하지 못한 부분인데, thread가 waiters list에 있는 동안 priority가 변경되었을 경우가 있을 수 있으므로 thread_unblock()을 수행하기 전에 list_sort() 함수를 이용하여 waiters list를 priority 기준으로 정렬하도록 하였다. 또한, sema_up이 하는 역할이 semaphore을 반환하고 동작을 마치므로 thread_yield()

를 추가해 주었다.

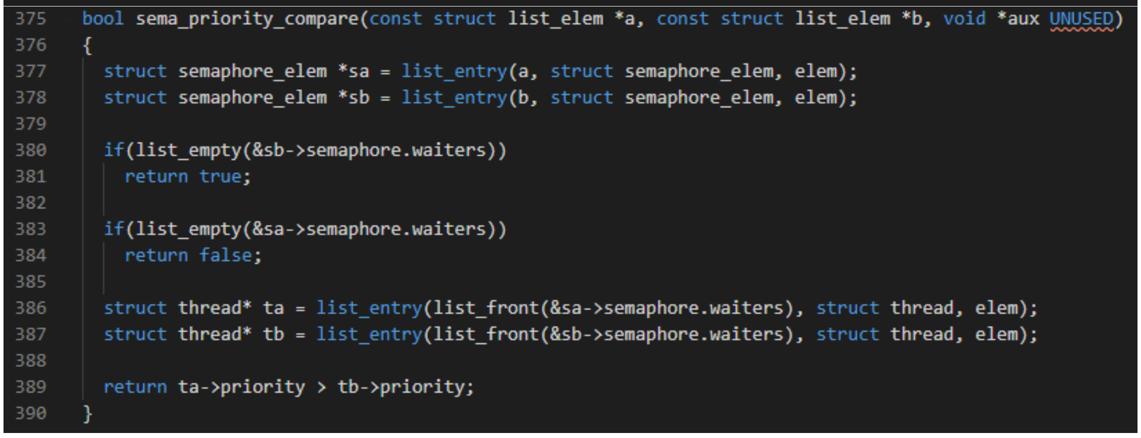


```c
void
sema_up (struct semaphore *sema)
{
  enum intr_level old_level;

  ASSERT (sema != NULL);

  old_level = intr_disable ();
  if (!list_empty (&sema->waiters))
    {
      list_sort(&sema->waiters, priority_compare, NULL);
      thread_unblock (list_entry(list_pop_front(&sema->waiters), struct thread, elem));
    }
  sema->value++;

  thread_yield();

  intr_set_level (old_level);
}
```

다음으로는 semaphore와 비슷하게 condition variable과 관련하여 수정한 내용이다. 이 부분도 Design Report를 작성할 때는 구현해야 한다는 것을 미처 생각하지 못하여 Design Report에는 언급되어 있지 않다. 먼저 cond_wait() 함수에서 condition variable의 waiters list에 element가 priority를 기준으로 삽입되도록 list_insert_ordered() 함수를 이용하였다. 이 때, list_insert_ordered() 함수의 인자로 sema_priority_compare() 함수를 새롭게 만들어 사용하였다. 기존의 priority_compare() 함수를 사용하지 않고 sema_priority_compare() 함수를 따로 만들어 사용한 이유는 condition variable같은 경우 waiters list에 semaphore_element가 삽입되므로 구조체의 구성이 다르기 때문이다.

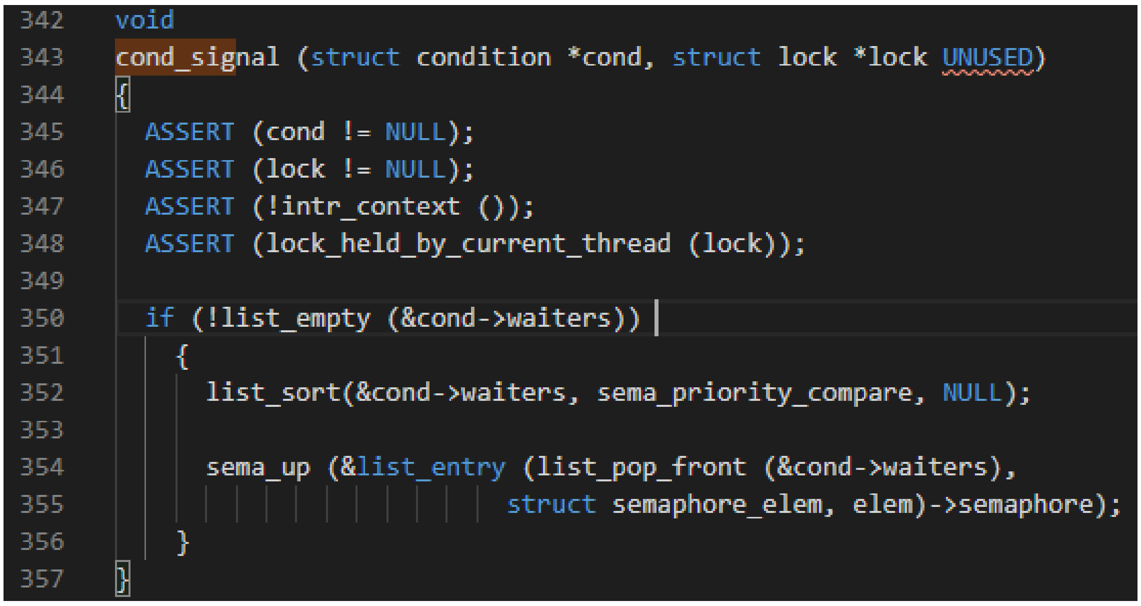


```c
void
cond_wait (struct condition *cond, struct lock *lock)
{
  struct semaphore_elem waiter;

  ASSERT (cond != NULL);
  ASSERT (lock != NULL);
  ASSERT (!intr_context ());
  ASSERT (lock_held_by_current_thread (lock));

  sema_init (&waiter.semaphore, 0);
  list_insert_ordered(&cond->waiters, &waiter.elem, sema_priority_compare, NULL);
  lock_release (lock);
  sema_down (&waiter.semaphore);
  lock_acquire (lock);
}
```



```c
bool sema_priority_compare(const struct list_elem *a, const struct list_elem *b, void *aux UNUSED)
{
  struct semaphore_elem *sa = list_entry(a, struct semaphore_elem, elem);
  struct semaphore_elem *sb = list_entry(b, struct semaphore_elem, elem);

  if(list_empty(&sb->semaphore.waiters))
    return true;

  if(list_empty(&sa->semaphore.waiters))
    return false;

  struct thread* ta = list_entry(list_front(&sa->semaphore.waiters), struct thread, elem);
  struct thread* tb = list_entry(list_front(&sb->semaphore.waiters), struct thread, elem);

  return ta->priority > tb->priority;
}
```

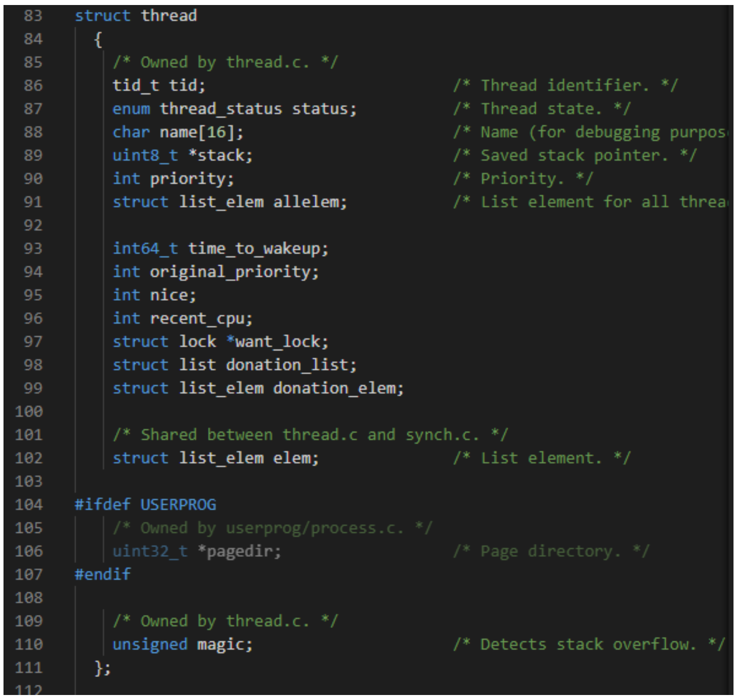
마지막으로 cond_signal() 함수에서 sema_up()을 하기 전에 waiters list에서 대기하던 중에 priority가 변경되었을 가능성이 있다. 따라서 list_sort() 함수를 사용하여 재정렬 시켜주었다.



```c
void
cond_signal (struct condition *cond, struct lock *lock UNUSED)
{
  ASSERT (cond != NULL);
  ASSERT (lock != NULL);
  ASSERT (!intr_context ());
  ASSERT (lock_held_by_current_thread (lock));

  if (!list_empty (&cond->waiters))
    {
      list_sort(&cond->waiters, sema_priority_compare, NULL);

      sema_up (&list_entry (list_pop_front (&cond->waiters),
                          struct semaphore_elem, elem)->semaphore);
    }
}
```

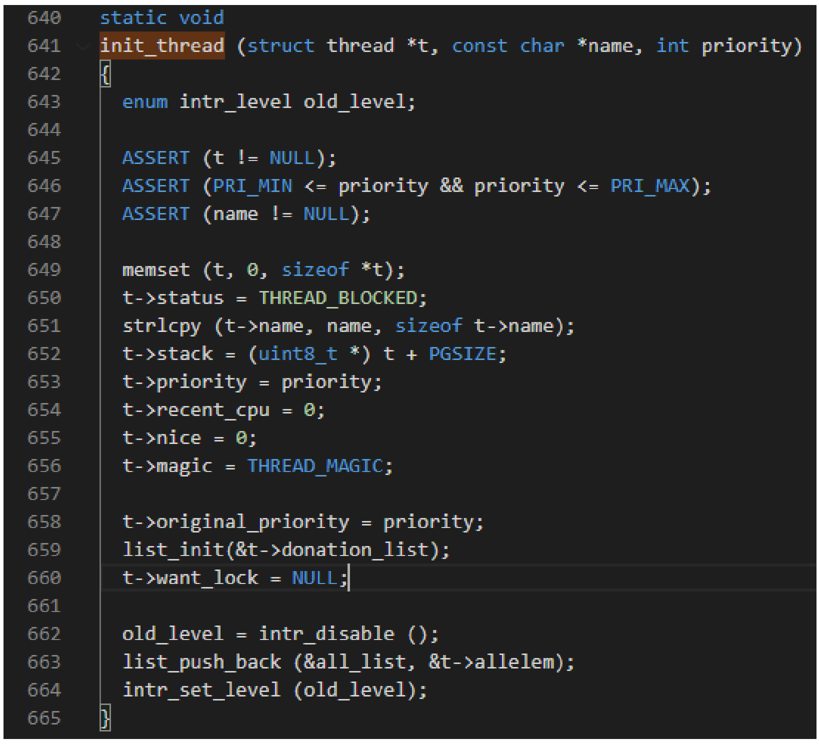
■ Priority Donation 구현

먼저 thread 구조체에 priority donation에 필요한 element들을 추가하였다. original_priority는 donation 이후에 priority를 초기화 시켜 주기 위해 저장한 priority 변경 이전의 priority를 나타내고, want_lock은 acquire하고 싶은 lock을 나타낸다. donation_list와 donation_elem은 multiple donation을 고려하기 위해 도입한 element 들로 donation을 받은 thread 구조체들의 list이다.



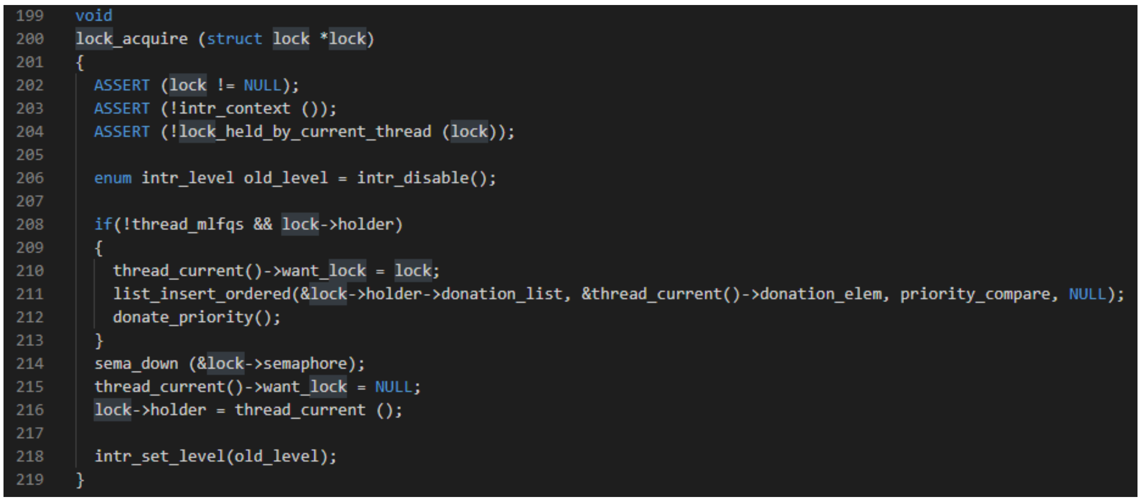
```c
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purpos
    uint8_t *stack;                     /* Saved stack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

    int64_t time_to_wakeup;
    int original_priority;
    int nice;
    int recent_cpu;
    struct lock *want_lock;
    struct list donation_list;
    struct list_elem donation_elem;

    /* 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. */
  };
```

먼저 init_thread() 함수에서 thread 구조체에 새롭게 추가한 element들을 초기화 시켜주었다.



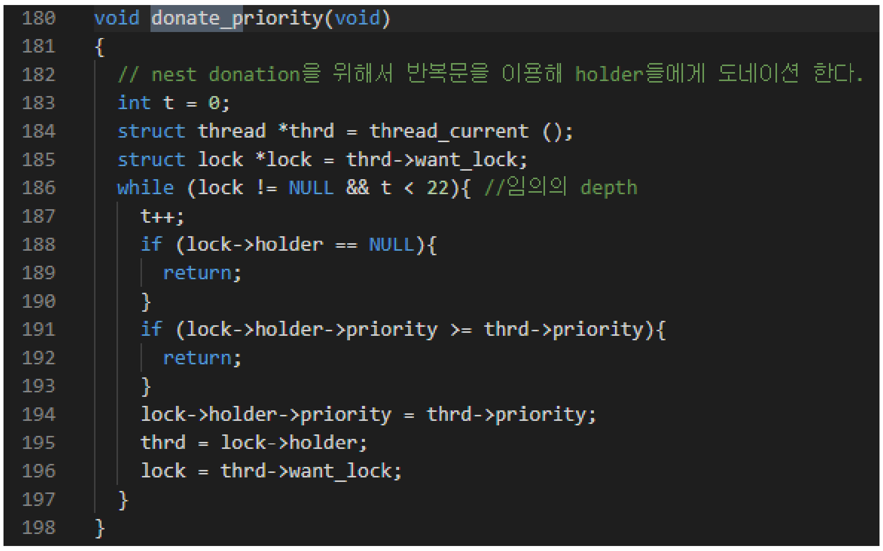
```c
static void
init_thread (struct thread *t, const char *name, int priority)
{
  enum intr_level old_level;

  ASSERT (t != NULL);
  ASSERT (PRI_MIN <= priority && priority <= PRI_MAX);
  ASSERT (name != NULL);

  memset (t, 0, sizeof *t);
  t->status = THREAD_BLOCKED;
  strlcpy (t->name, name, sizeof t->name);
  t->stack = (uint8_t *) t + PGSIZE;
  t->priority = priority;
  t->recent_cpu = 0;
  t->nice = 0;
  t->magic = THREAD_MAGIC;

  t->original_priority = priority;
  list_init(&t->donation_list);
  t->want_lock = NULL;

  old_level = intr_disable ();
  list_push_back (&all_list, &t->allelem);
  intr_set_level (old_level);
}
```

lock_aquire() 함수에서는 인자로 받은 lock의 holder가 이미 존재한다면 현재의 thread의 want_lock 변수에 lock을 저장해 주고, list_insert_ordered() 함수를 이용하여 holder의 donation list에 thread_current의 donation_elem을 저장하고 donate_priority() 함수를 호출하여 priority donation을 수행하였다.



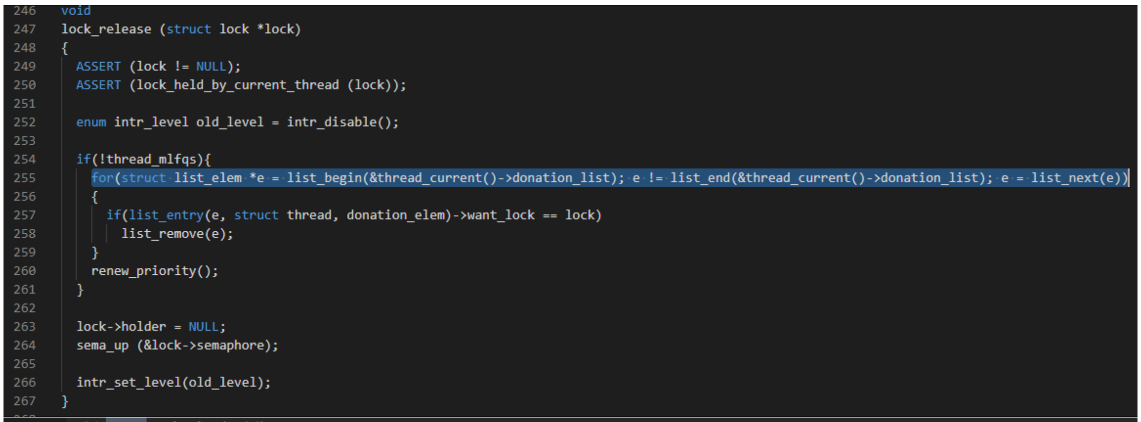
```c
void
lock_acquire (struct lock *lock)
{
  ASSERT (lock != NULL);
  ASSERT (!intr_context ());
  ASSERT (!lock_held_by_current_thread (lock));

  enum intr_level old_level = intr_disable();

  if(!thread_mlfqs && lock->holder)
  {
    thread_current()->want_lock = lock;
    list_insert_ordered(&lock->holder->donation_list, &thread_current()->donation_elem, priority_compare, NULL);
    donate_priority();
  }
  sema_down (&lock->semaphore);
  thread_current()->want_lock = NULL;
  lock->holder = thread_current ();

  intr_set_level(old_level);
}
```

다음으로 priority donation이 직접 실행되는 함수인 donate_priority() 함수이다. 이 함수에서는 처음에 현재 thread가 기다리고 있는 lock에 대해 마치 linked list 형태와 같이 while문을 통해 nested donation을 해결해 준다.

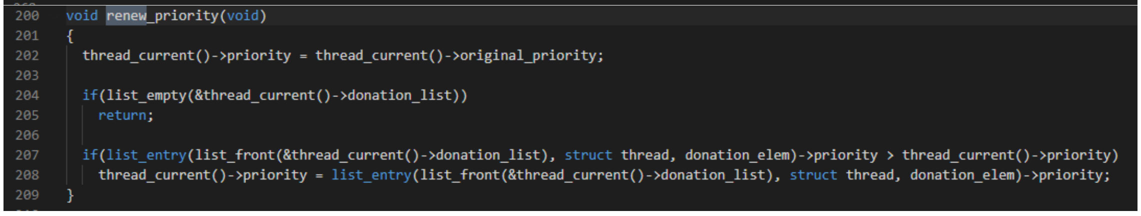


```c
void donate_priority(void)
{
  // nest donation을 위해서 반복문을 이용해 holder들에게 도네이션 한다.
  int t = 0;
  struct thread *thrd = thread_current ();
  struct lock *lock = thrd->want_lock;
  while (lock != NULL && t < 22){ //임의의 depth
    t++;
    if (lock->holder == NULL){
      return;
    }
    if (lock->holder->priority >= thrd->priority){
      return;
    }
    lock->holder->priority = thrd->priority;
    thrd = lock->holder;
    lock = thrd->want_lock;
  }
}
```

다음으로 lock release() 에서는 lock을 해지했을 때 이전의 상태로 돌아가기 위해 필요한 작업들을 추가해 주었다. 원래 Design Report에서는 이를 두 개의 함수로 나누어 작업하려고 했었지만, 첫 번째로 만들려고 했던 donation list에서 해지 할 lock을 보유하고 있는 entry를 삭제하는 함수는 lock_release() 함수에서밖에 쓰이지 않기 때문에 따로 함수로 만들지 않고 직접 코드를 써 넣었다. 그리고 renew_priority() 함수에서는 thread의 priority가 변경되었으므로 donation을 고려해서 priority를 다시 결정하는 작업을 진행하였다. 이는 현재 thread를 donation 받기 전의 original priority로 변경하고, donation list에서 가장 priority가 높은 thread와 현재 thread의 priority를 비교하여 높은 값을 현재 thread의 priority로 설정하는 과정으로 이루어졌다.

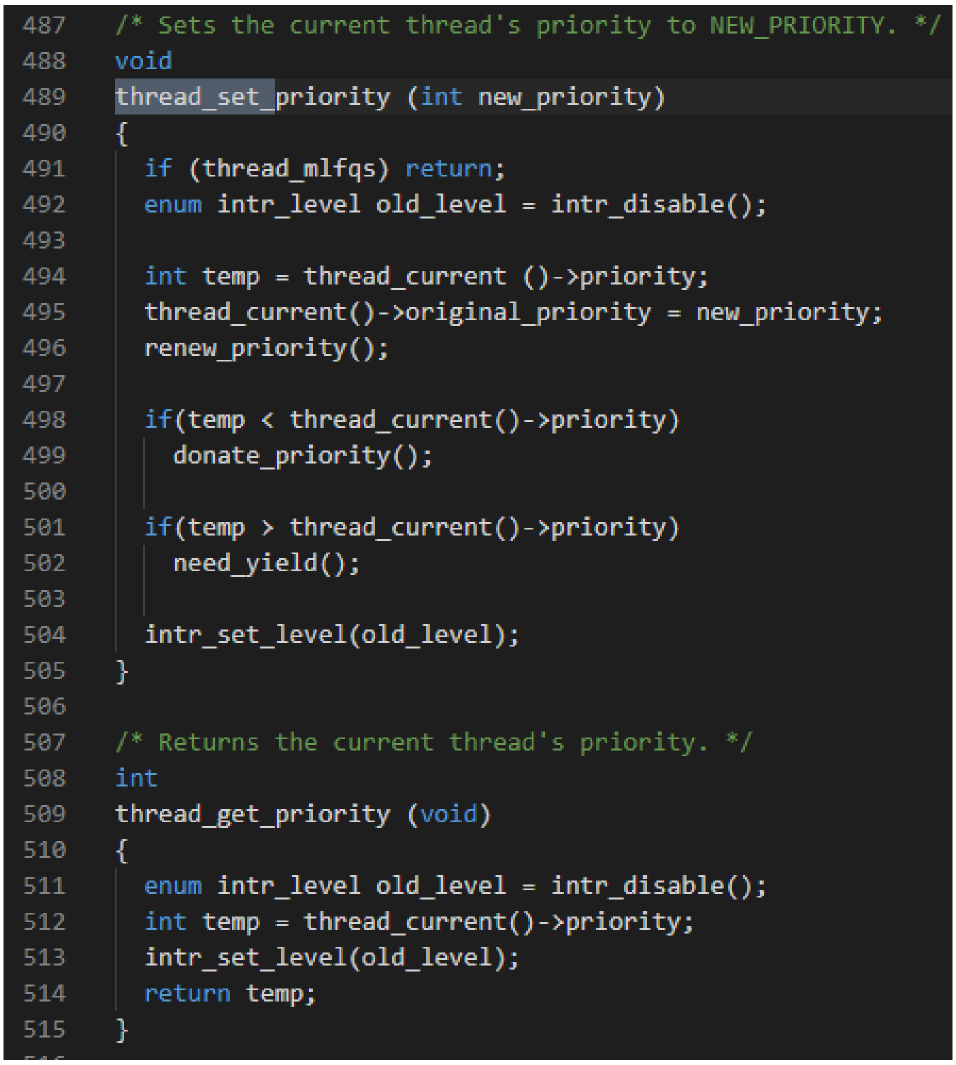


```c
void
lock_release (struct lock *lock)
{
  ASSERT (lock != NULL);
  ASSERT (lock_held_by_current_thread (lock));

  enum intr_level old_level = intr_disable();

  if(!thread_mlfqs){
    for(struct list_elem *e = list_begin(&thread_current()->donation_list); e != list_end(&thread_current()->donation_list); e = list_next(e))
    {
      if(list_entry(e, struct thread, donation_elem)->want_lock == lock)
        list_remove(e);
    }
    renew_priority();
  }

  lock->holder = NULL;
  sema_up (&lock->semaphore);

  intr_set_level(old_level);
}
```



```c
void renew_priority(void)
{
  thread_current()->priority = thread_current()->original_priority;

  if(list_empty(&thread_current()->donation_list))
    return;

  if(list_entry(list_front(&thread_current()->donation_list), struct thread, donation_elem)->priority > thread_current()->priority)
    thread_current()->priority = list_entry(list_front(&thread_current()->donation_list), struct thread, donation_elem)->priority;
}
```

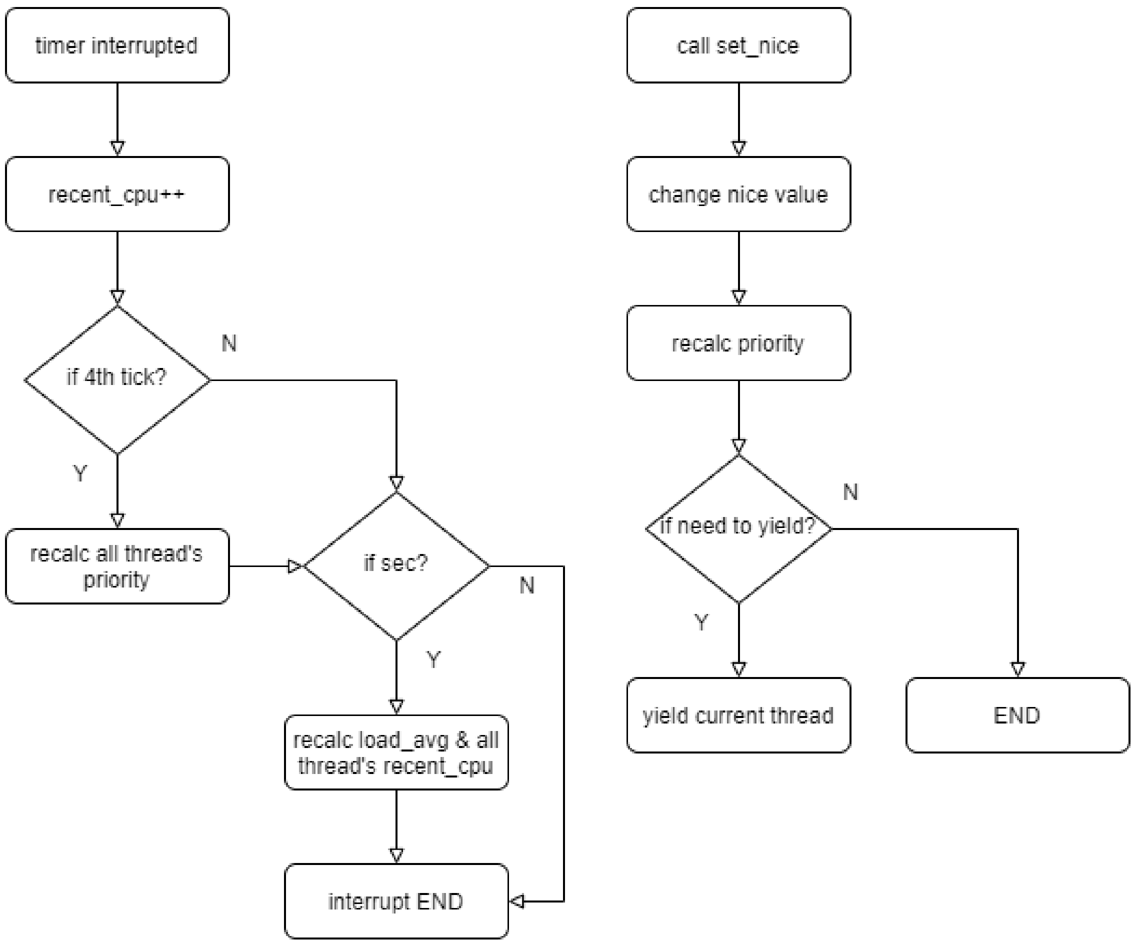
마지막으로 thread_set_priority() 함수와 thread_get_priority() 함수의 수정이 이루어졌다. thread_set_priority() 함수에서는 현재 thread의 priority를 new_priority로 설정하려고 할 때 original_priority를 new priority로 설정하고 renew_priority() 함수를 이용하여 thread_current의 priority를 설정해 주었다. 이는 donation을 반영하였기 때문에 나온 결과이다. 이 때 temp < thread_current()->priority일 경우에는 priority inversion 현상이 발생할 수 있으므로 donate_priority() 함수를 호출하여 이를 해결한다. thread_get_priority() 함수에서는 interrupt disable 과정을 추가해 주어 thread의 priority 값을 return 하도록 하였다.



```
/* Sets the current thread's priority to NEW_PRIORITY. */
void
thread_set_priority (int new_priority)
{
  if (thread_mlfqs) return;
  enum intr_level old_level = intr_disable();

  int temp = thread_current ()->priority;
  thread_current()->original_priority = new_priority;
  renew_priority();

  if(temp < thread_current()->priority)
    donate_priority();

  if(temp > thread_current()->priority)
    need_yield();

  intr_set_level(old_level);
}

/* Returns the current thread's priority. */
int
thread_get_priority (void)
{
  enum intr_level old_level = intr_disable();
  int temp = thread_current()->priority;
  intr_set_level(old_level);
  return temp;
}
```

4. Advanced Scheduler

4.1. Flow Chart



4.2. Added or modified Function & Data Structure

- Modified Data Structure

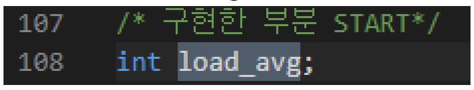
  struct thread, int load_avg
- Added Function

  "fixed_point.h", mlfqs_calc_priority(), mlfqs_increment_recent_cpu(), mlfqs_recalc_per_Sec(), mlfqs_recalc_all_priority()
- Modified Function

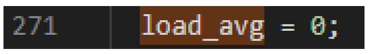
  init_thread(), thread_start(), thread_set_priority(), thread_set_nice(), thread_get_nice(), thread_get_load_avg(), thread_get_recent_cpu(), timer_interrupt(), lock_acquire(), lock_release()

4.3. Implementation

우선 변수 load_avg를 선언하고 thread_start()에서 초기화 시켰다.

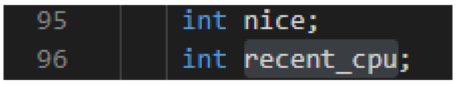


```
/* 구현한 부분 START*/
int load_avg;
```

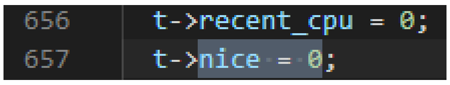


```
  load_avg = 0;
```

다음으로는 sturct thread에 뱐수 nice와 recent_cpu를 추가하고 init_thread()에서 초기화 시켰다.

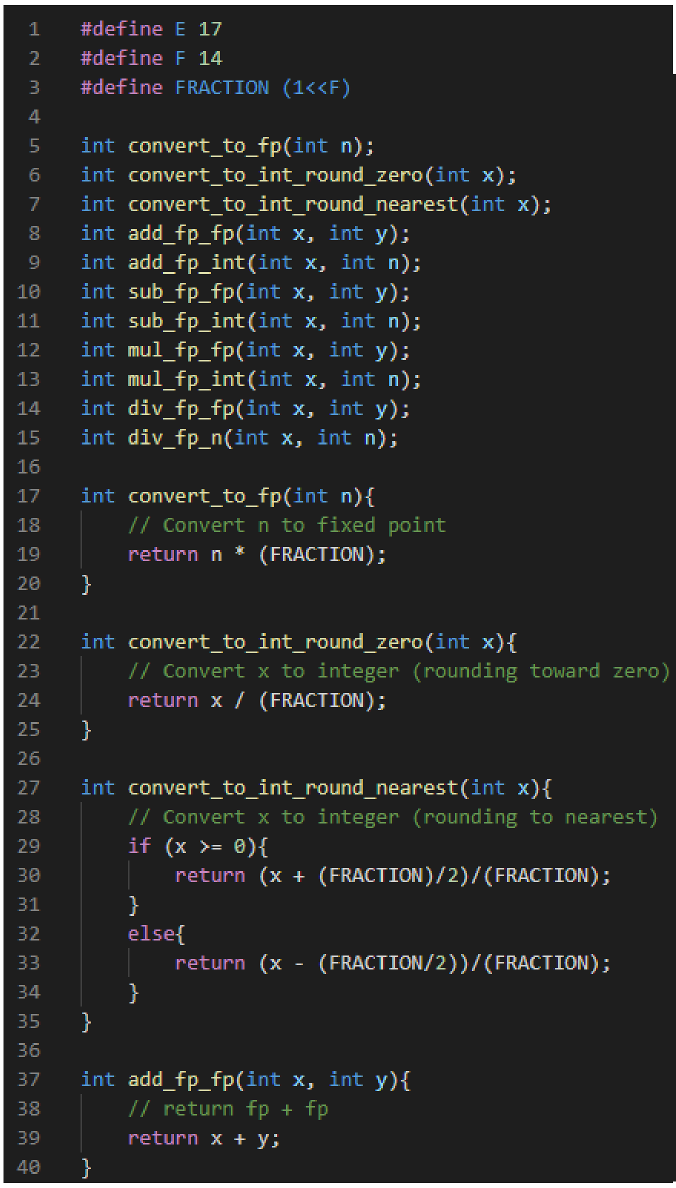


```c
    int nice;
    int recent_cpu;
```

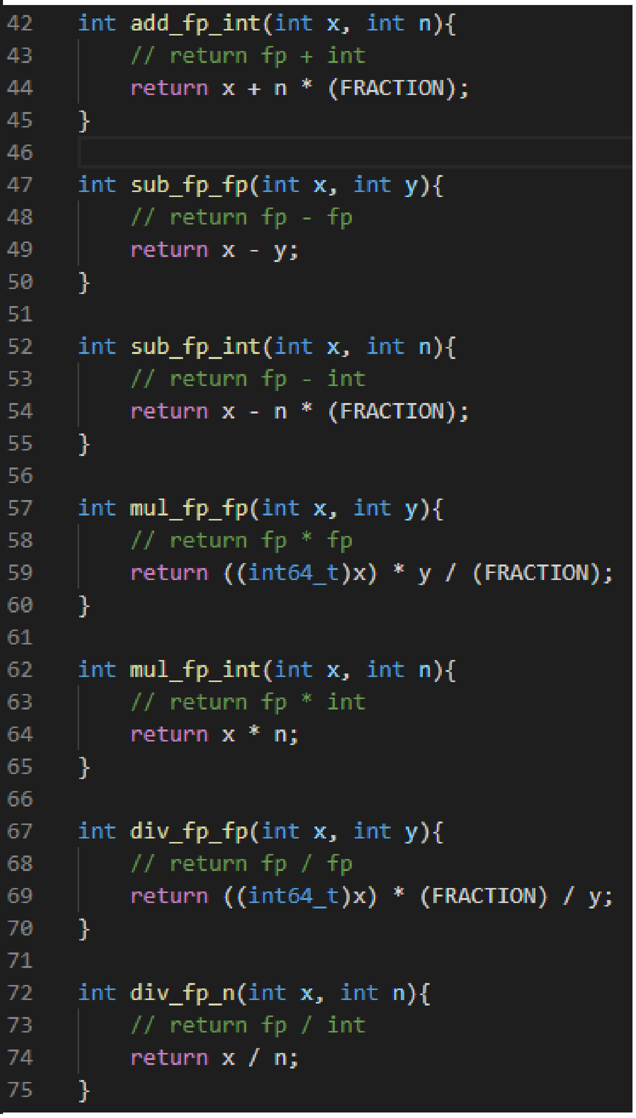


```c
  t->recent_cpu = 0;
  t->nice = 0;
```

다음으로 fixed point 연산을 위한 함수들을 APPENDIX에서 주어진 수식을 따라 "fixed_point.h"에 다음과 같이 구현했다.

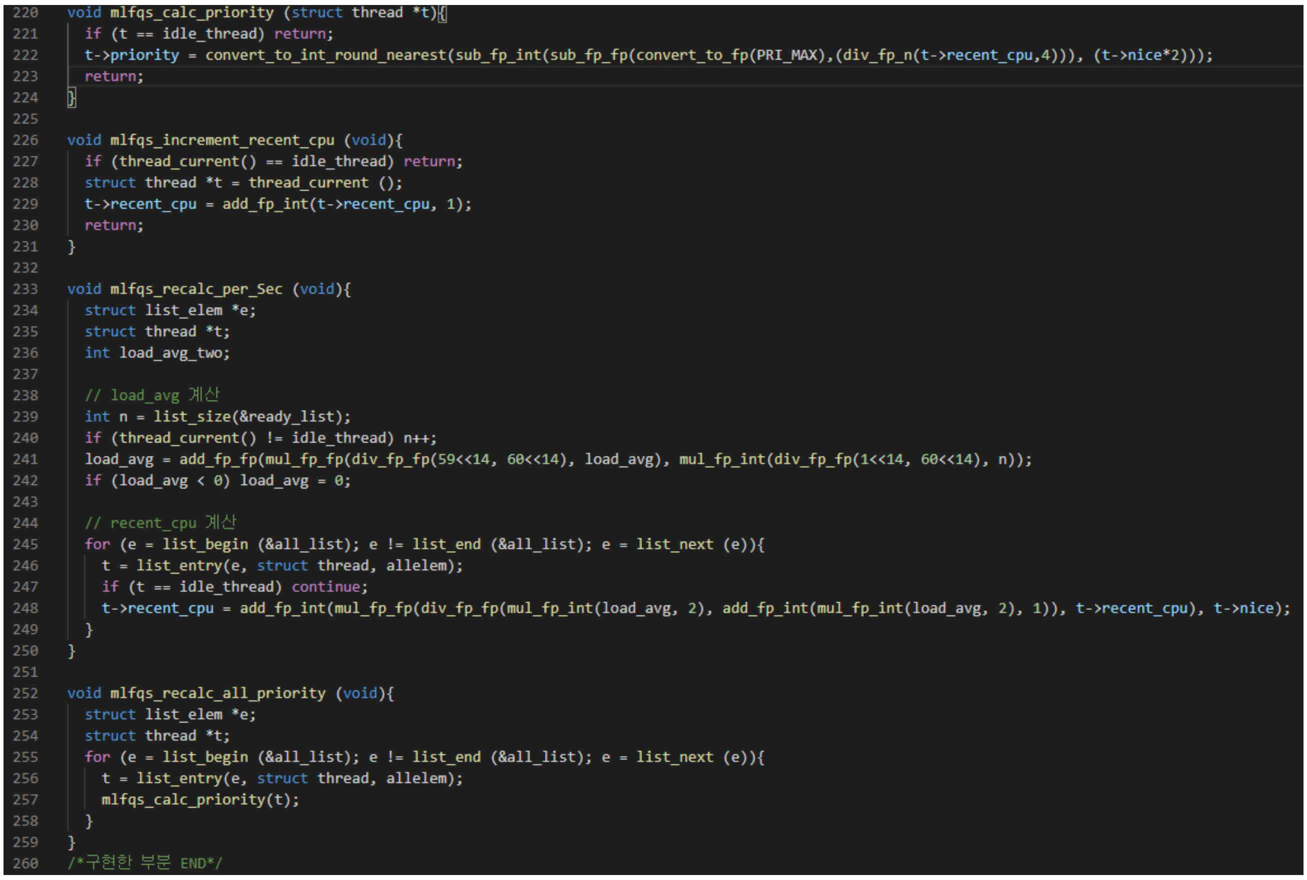


```c
#define E 17
#define F 14
#define FRACTION (1<<F)

int convert_to_fp(int n);
int convert_to_int_round_zero(int x);
int convert_to_int_round_nearest(int x);
int add_fp_fp(int x, int y);
int add_fp_int(int x, int n);
int sub_fp_fp(int x, int y);
int sub_fp_int(int x, int n);
int mul_fp_fp(int x, int y);
int mul_fp_int(int x, int n);
int div_fp_fp(int x, int y);
int div_fp_n(int x, int n);

int convert_to_fp(int n){
    // Convert n to fixed point
    return n * (FRACTION);
}

int convert_to_int_round_zero(int x){
    // Convert x to integer (rounding toward zero)
    return x / (FRACTION);
}

int convert_to_int_round_nearest(int x){
    // Convert x to integer (rounding to nearest)
    if (x >= 0){
        return (x + (FRACTION)/2)/(FRACTION);
    }
    else{
        return (x - (FRACTION/2))/(FRACTION);
    }
}

int add_fp_fp(int x, int y){
    // return fp + fp
    return x + y;
}
```



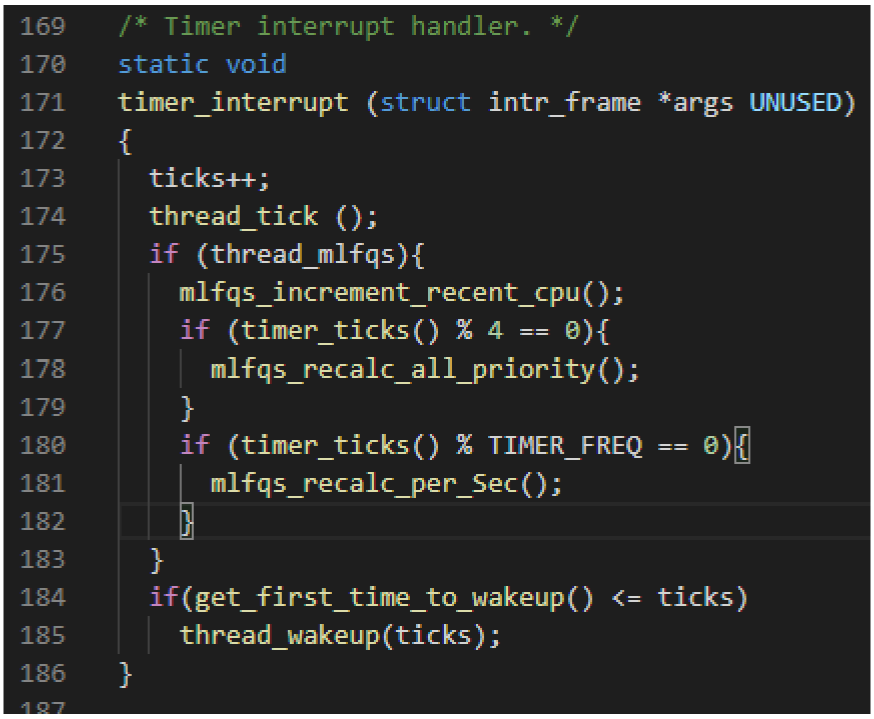
```c
int add_fp_int(int x, int n){
    // return fp + int
    return x + n * (FRACTION);
}

int sub_fp_fp(int x, int y){
    // return fp - fp
    return x - y;
}

int sub_fp_int(int x, int n){
    // return fp - int
    return x - n * (FRACTION);
}

int mul_fp_fp(int x, int y){
    // return fp * fp
    return ((int64_t)x) * y / (FRACTION);
}

int mul_fp_int(int x, int n){
    // return fp * int
    return x * n;
}

int div_fp_fp(int x, int y){
    // return fp / fp
    return ((int64_t)x) * (FRACTION) / y;
}

int div_fp_n(int x, int n){
    // return fp / int
    return x / n;
}
```

다음으로 load_avg, recent_cpu, priority의 계산을 위한 함수들을 구현했다. 아래 사진에 보이듯이 mlfqs_calc_priority(), mlfqs_increment_recent_cpu(), mlfqs_recalc_per_Sec(), mlfqs_recalc_all_priority()이렇게 총 4개의 함수를 구현했다. 우선 design report에 작성했던 것처럼 모든 스레드의 priority는 매 4tick마다 recalculate 된다. 그리고 load_avg와 모든 스레드의 recent_cpu는 timer_ticks()%TIMER_FREQ==0, 즉 1초마다 업데이트 된다. timer_interrupt()에서 이를 간편하게 실행시키기 위해서 두개의 함수 mlfqs_recalc_all_priority()와 mlfqs_recalc_per_Sec()로 분리해서 구현했다. mlfqs_calc_priority()에서는 모든 스레드의 priority를 재계산해주기 위해서 all_list를 순회하면서 mlfqs_calc_priority()를 호출하도록 구현했다. mlfqs_calc_priority()는 thread_set_nice()에서 priority의 재계산을 할 때 사용하기 위해서 별도의 함수로 분리했다. 다음으로 매 timer interrupt마다 current thread의 recent_cpu를 증가시켜 주어야 하기 때문에 mlfqs_increment_recent_cpu()를 구현했다. 이 4개의 함수에는 "fixed_point.h"에 구현된 함수들이 사용되었고, 계산 공식은 design report에도 적어 두었던 것처럼 APPENDIX에서 제공된 계산식을 따랐다.



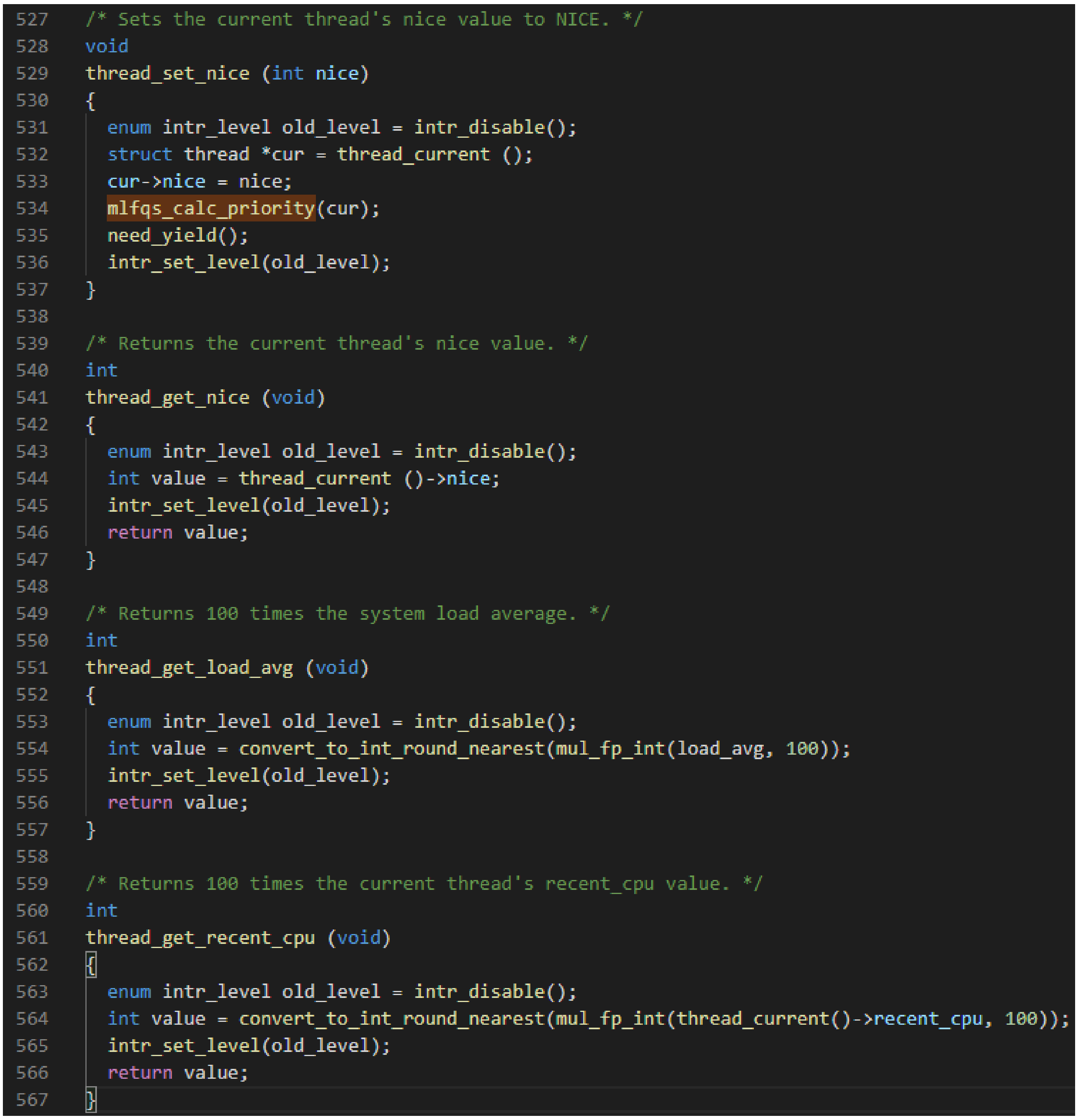
```c
void mlfqs_calc_priority (struct thread *t){
  if (t == idle_thread) return;
  t->priority = convert_to_int_round_nearest(sub_fp_int(sub_fp_fp(convert_to_fp(PRI_MAX),(div_fp_n(t->recent_cpu,4))), (t->nice*2)));
  return;
}

void mlfqs_increment_recent_cpu (void){
  if (thread_current() == idle_thread) return;
  struct thread *t = thread_current ();
  t->recent_cpu = add_fp_int(t->recent_cpu, 1);
  return;
}

void mlfqs_recalc_per_Sec (void){
  struct list_elem *e;
  struct thread *t;
  int load_avg_two;

  // load_avg 계산
  int n = list_size(&ready_list);
  if (thread_current() != idle_thread) n++;
  load_avg = add_fp_fp(mul_fp_fp(div_fp_fp(59<<14, 60<<14), load_avg), mul_fp_int(div_fp_fp(1<<14, 60<<14), n));
  if (load_avg < 0) load_avg = 0;

  // recent_cpu 계산
  for (e = list_begin (&all_list); e != list_end (&all_list); e = list_next (e)){
    t = list_entry(e, struct thread, allelem);
    if (t == idle_thread) continue;
    t->recent_cpu = add_fp_int(mul_fp_fp(div_fp_fp(mul_fp_int(load_avg, 2), add_fp_int(mul_fp_int(load_avg, 2), 1)), t->recent_cpu), t->nice);
  }
}

void mlfqs_recalc_all_priority (void){
  struct list_elem *e;
  struct thread *t;
  for (e = list_begin (&all_list); e != list_end (&all_list); e = list_next (e)){
    t = list_entry(e, struct thread, allelem);
    mlfqs_calc_priority(t);
  }
}
/*구현한 부분 END*/
```

그리고 나서 timer_interrupt()에서 mlfqs함수들을 이용해서 적절한 타이밍에 값들을 재계산하도록 구현했다.



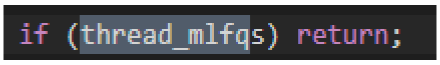
```c
/* Timer interrupt handler. */
static void
timer_interrupt (struct intr_frame *args UNUSED)
{
  ticks++;
  thread_tick ();
  if (thread_mlfqs){
    mlfqs_increment_recent_cpu();
    if (timer_ticks() % 4 == 0){
      mlfqs_recalc_all_priority();
    }
    if (timer_ticks() % TIMER_FREQ == 0){
      mlfqs_recalc_per_Sec();
    }
  }
  if(get_first_time_to_wakeup() <= ticks)
    thread_wakeup(ticks);
}
```

다음으로는 구현되지 않았던 4개의 set/get 함수들을 구현했다. get함수들은 project에서 요구하는 형식에 맞추어서(load_avg와 recent_cpu의 경우에는 100을 곱해서 return) 구현했다. 그리고 thread_set_nice()의 경우에, APPENDIX에서 요구된 것처럼 nice값을 설정한 다음 바로 mlfqs_calc_priority()를 이용해서 priority를 재계산 해준다음 need_yield()를 호출해서 yield해야하는지 검사하도록 구현했다.



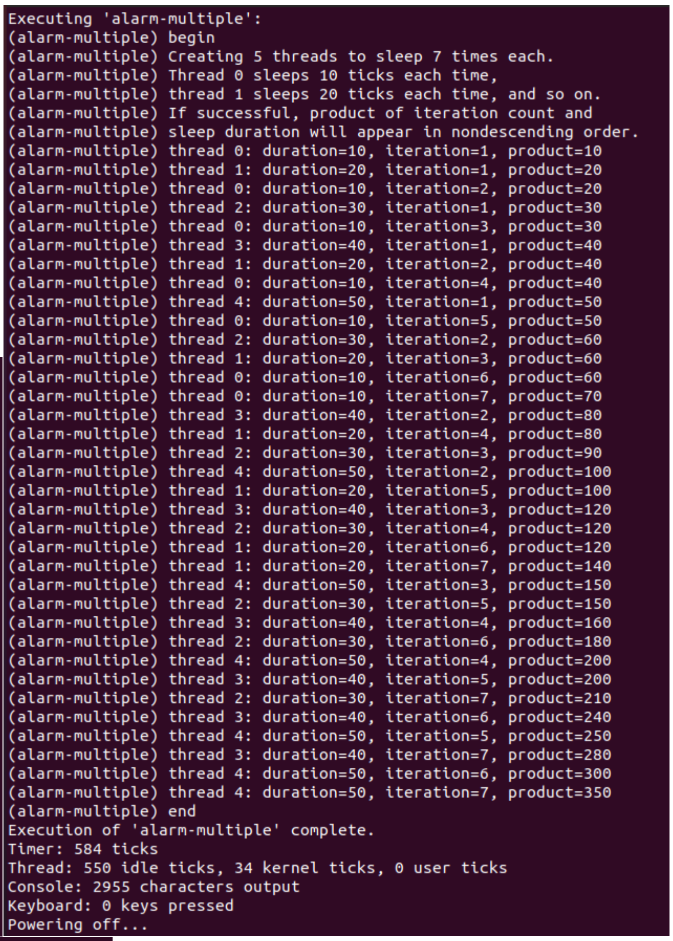
```c
/* Sets the current thread's nice value to NICE. */
void
thread_set_nice (int nice)
{
  enum intr_level old_level = intr_disable();
  struct thread *cur = thread_current ();
  cur->nice = nice;
  mlfqs_calc_priority(cur);
  need_yield();
  intr_set_level(old_level);
}

/* Returns the current thread's nice value. */
int
thread_get_nice (void)
{
  enum intr_level old_level = intr_disable();
  int value = thread_current ()->nice;
  intr_set_level(old_level);
  return value;
}

/* Returns 100 times the system load average. */
int
thread_get_load_avg (void)
{
  enum intr_level old_level = intr_disable();
  int value = convert_to_int_round_nearest(mul_fp_int(load_avg, 100));
  intr_set_level(old_level);
  return value;
}

/* Returns 100 times the current thread's recent_cpu value. */
int
thread_get_recent_cpu (void)
{
  enum intr_level old_level = intr_disable();
  int value = convert_to_int_round_nearest(mul_fp_int(thread_current()->recent_cpu, 100));
  intr_set_level(old_level);
  return value;
}
```

마지막으로 mlfqs가 실행되면 thread_mlfqs값이 true로 설정된다. thread_mlfqs를 이용해서 thread_set_priority(), lock_acquire(), lock_release() 세 함수에서 priority donation 관련 코드들이 동작하지 않도록 구현했다.

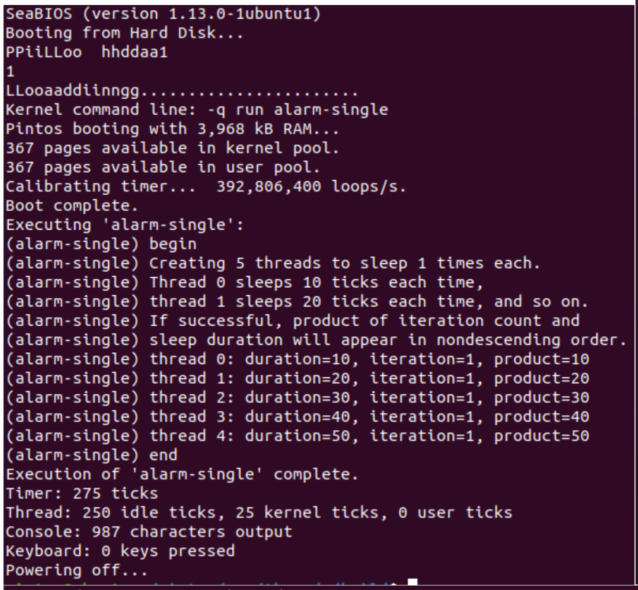


```c
if (thread_mlfqs) return;
```

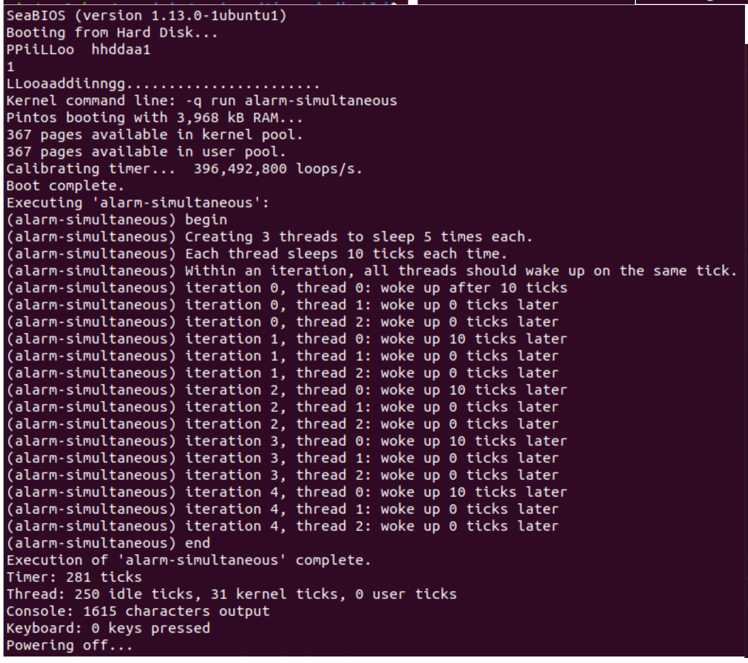
5. Result Verification



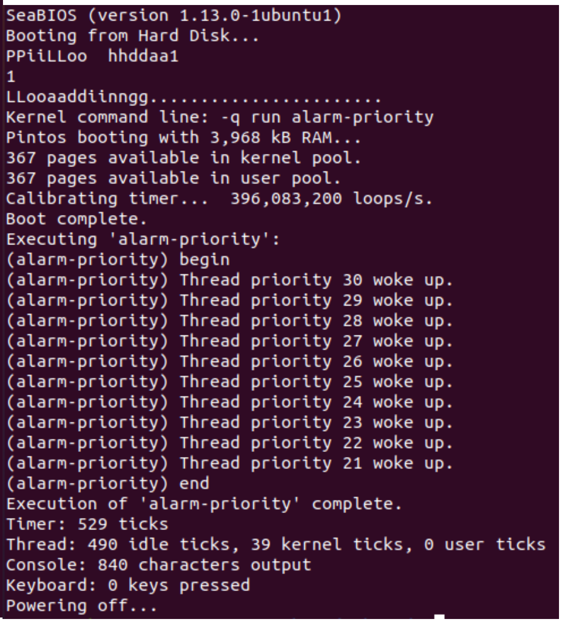
```
Executing 'alarm-multiple':
(alarm-multiple) begin
(alarm-multiple) Creating 5 threads to sleep 7 times each.
(alarm-multiple) Thread 0 sleeps 10 ticks each time,
(alarm-multiple) thread 1 sleeps 20 ticks each time, and so on.
(alarm-multiple) If successful, product of iteration count and
(alarm-multiple) sleep duration will appear in nondescending order.
(alarm-multiple) thread 0: duration=10, iteration=1, product=10
(alarm-multiple) thread 1: duration=20, iteration=1, product=20
(alarm-multiple) thread 0: duration=10, iteration=2, product=20
(alarm-multiple) thread 2: duration=30, iteration=1, product=30
(alarm-multiple) thread 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 3: duration=40, iteration=1, product=40
(alarm-multiple) thread 1: duration=20, iteration=2, product=40
(alarm-multiple) thread 0: duration=10, iteration=4, product=40
(alarm-multiple) thread 4: duration=50, iteration=1, product=50
(alarm-multiple) thread 0: duration=10, iteration=5, product=50
(alarm-multiple) thread 2: duration=30, iteration=2, product=60
(alarm-multiple) thread 1: duration=20, iteration=3, product=60
(alarm-multiple) thread 0: duration=10, iteration=6, product=60
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 584 ticks
Thread: 550 idle ticks, 34 kernel ticks, 0 user ticks
Console: 2955 characters output
Keyboard: 0 keys pressed
Powering off...
```



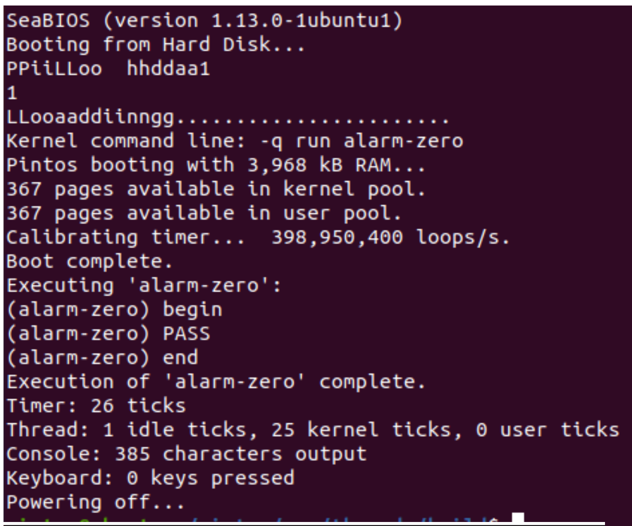
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run alarm-single
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  392,806,400 loops/s.
Boot complete.
Executing 'alarm-single':
(alarm-single) begin
(alarm-single) Creating 5 threads to sleep 1 times each.
(alarm-single) Thread 0 sleeps 10 ticks each time,
(alarm-single) thread 1 sleeps 20 ticks each time, and so on.
(alarm-single) If successful, product of iteration count and
(alarm-single) sleep duration will appear in nondescending order.
(alarm-single) thread 0: duration=10, iteration=1, product=10
(alarm-single) thread 1: duration=20, iteration=1, product=20
(alarm-single) thread 2: duration=30, iteration=1, product=30
(alarm-single) thread 3: duration=40, iteration=1, product=40
(alarm-single) thread 4: duration=50, iteration=1, product=50
(alarm-single) end
Execution of 'alarm-single' complete.
Timer: 275 ticks
Thread: 250 idle ticks, 25 kernel ticks, 0 user ticks
Console: 987 characters output
Keyboard: 0 keys pressed
Powering off...
```



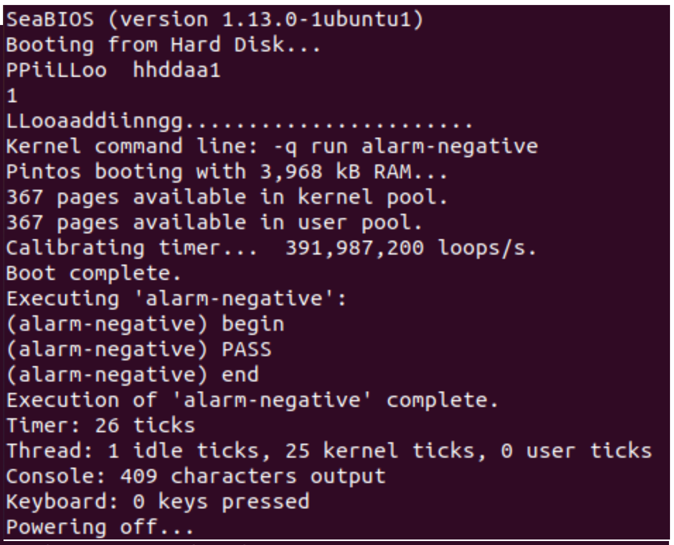
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run alarm-simultaneous
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  396,492,800 loops/s.
Boot complete.
Executing 'alarm-simultaneous':
(alarm-simultaneous) begin
(alarm-simultaneous) Creating 3 threads to sleep 5 times each.
(alarm-simultaneous) Each thread sleeps 10 ticks each time.
(alarm-simultaneous) Within an iteration, all threads should wake up on the same tick.
(alarm-simultaneous) iteration 0, thread 0: woke up after 10 ticks
(alarm-simultaneous) iteration 0, thread 1: woke up 0 ticks later
(alarm-simultaneous) iteration 0, thread 2: woke up 0 ticks later
(alarm-simultaneous) iteration 1, thread 0: woke up 10 ticks later
(alarm-simultaneous) iteration 1, thread 1: woke up 0 ticks later
(alarm-simultaneous) iteration 1, thread 2: woke up 0 ticks later
(alarm-simultaneous) iteration 2, thread 0: woke up 10 ticks later
(alarm-simultaneous) iteration 2, thread 1: woke up 0 ticks later
(alarm-simultaneous) iteration 2, thread 2: woke up 0 ticks later
(alarm-simultaneous) iteration 3, thread 0: woke up 10 ticks later
(alarm-simultaneous) iteration 3, thread 1: woke up 0 ticks later
(alarm-simultaneous) iteration 3, thread 2: woke up 0 ticks later
(alarm-simultaneous) iteration 4, thread 0: woke up 10 ticks later
(alarm-simultaneous) iteration 4, thread 1: woke up 0 ticks later
(alarm-simultaneous) iteration 4, thread 2: woke up 0 ticks later
(alarm-simultaneous) end
Execution of 'alarm-simultaneous' complete.
Timer: 281 ticks
Thread: 250 idle ticks, 31 kernel ticks, 0 user ticks
Console: 1615 characters output
Keyboard: 0 keys pressed
Powering off...
```



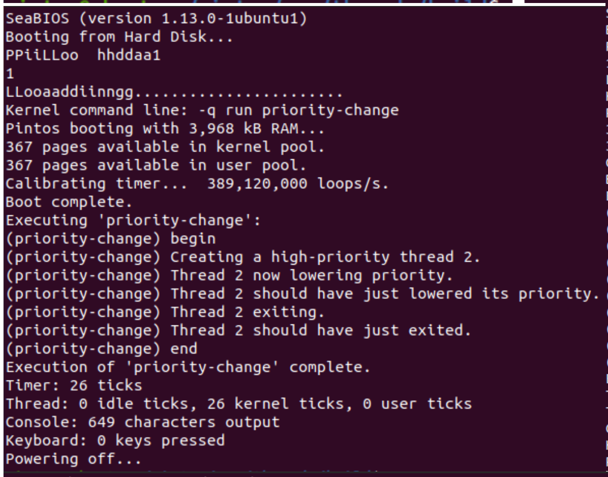
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run alarm-priority
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  396,083,200 loops/s.
Boot complete.
Executing 'alarm-priority':
(alarm-priority) begin
(alarm-priority) Thread priority 30 woke up.
(alarm-priority) Thread priority 29 woke up.
(alarm-priority) Thread priority 28 woke up.
(alarm-priority) Thread priority 27 woke up.
(alarm-priority) Thread priority 26 woke up.
(alarm-priority) Thread priority 25 woke up.
(alarm-priority) Thread priority 24 woke up.
(alarm-priority) Thread priority 23 woke up.
(alarm-priority) Thread priority 22 woke up.
(alarm-priority) Thread priority 21 woke up.
(alarm-priority) end
Execution of 'alarm-priority' complete.
Timer: 529 ticks
Thread: 490 idle ticks, 39 kernel ticks, 0 user ticks
Console: 840 characters output
Keyboard: 0 keys pressed
Powering off...
```



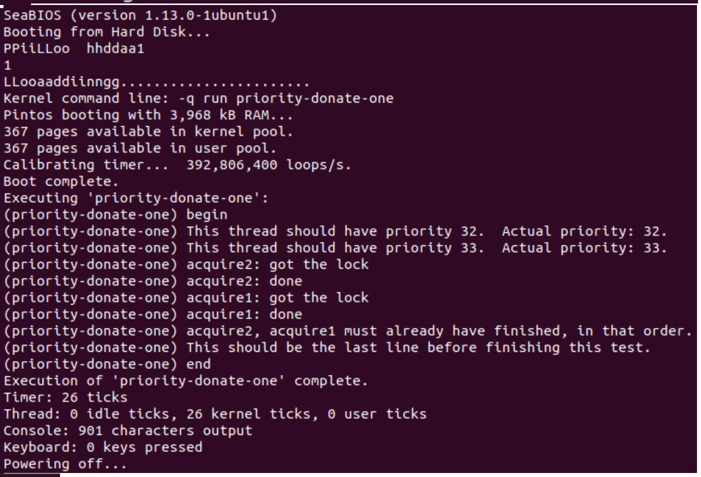
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q run alarm-zero
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,950,400 loops/s.
Boot complete.
Executing 'alarm-zero':
(alarm-zero) begin
(alarm-zero) PASS
(alarm-zero) end
Execution of 'alarm-zero' complete.
Timer: 26 ticks
Thread: 1 idle ticks, 25 kernel ticks, 0 user ticks
Console: 385 characters output
Keyboard: 0 keys pressed
Powering off...
```



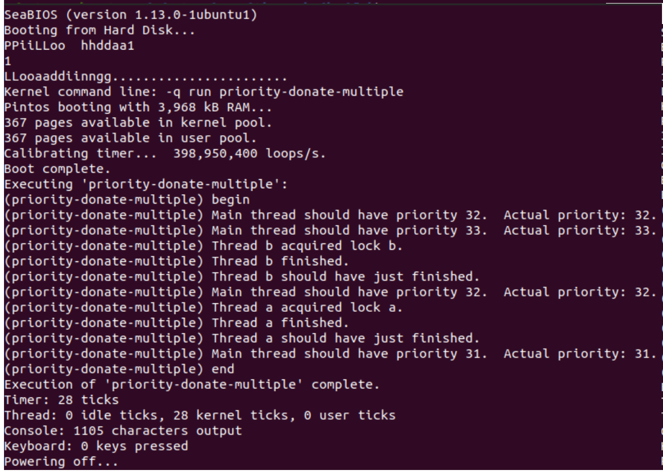
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q run alarm-negative
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  391,987,200 loops/s.
Boot complete.
Executing 'alarm-negative':
(alarm-negative) begin
(alarm-negative) PASS
(alarm-negative) end
Execution of 'alarm-negative' complete.
Timer: 26 ticks
Thread: 1 idle ticks, 25 kernel ticks, 0 user ticks
Console: 409 characters output
Keyboard: 0 keys pressed
Powering off...
```



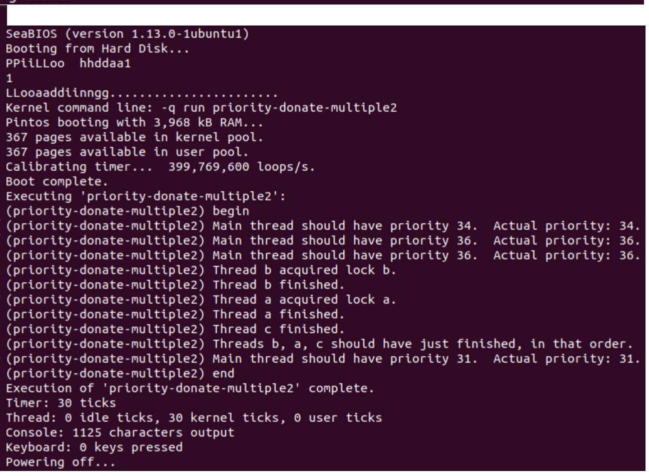
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q run priority-change
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  389,120,000 loops/s.
Boot complete.
Executing 'priority-change':
(priority-change) begin
(priority-change) Creating a high-priority thread 2.
(priority-change) Thread 2 now lowering priority.
(priority-change) Thread 2 should have just lowered its priority.
(priority-change) Thread 2 exiting.
(priority-change) Thread 2 should have just exited.
(priority-change) end
Execution of 'priority-change' complete.
Timer: 26 ticks
Thread: 0 idle ticks, 26 kernel ticks, 0 user ticks
Console: 649 characters output
Keyboard: 0 keys pressed
Powering off...
```



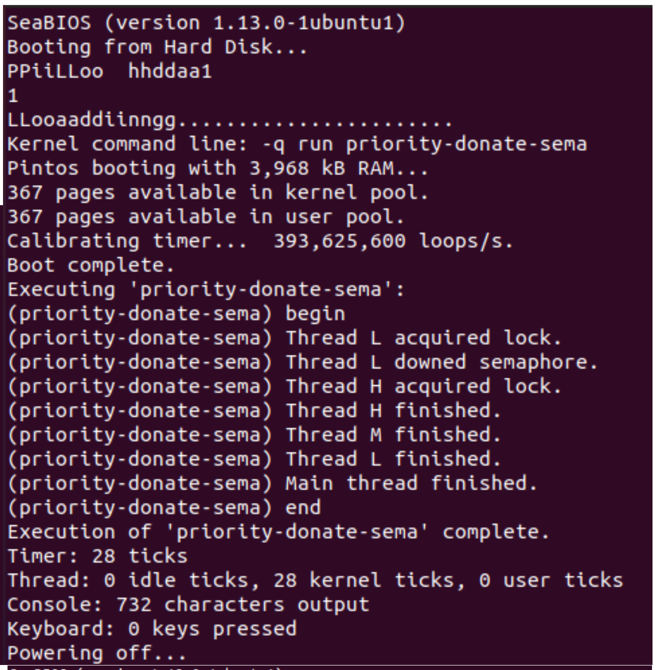
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q run priority-donate-one
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  392,806,400 loops/s.
Boot complete.
Executing 'priority-donate-one':
(priority-donate-one) begin
(priority-donate-one) This thread should have priority 32.  Actual priority: 32.
(priority-donate-one) This thread should have priority 33.  Actual priority: 33.
(priority-donate-one) acquire2: got the lock
(priority-donate-one) acquire2: done
(priority-donate-one) acquire1: got the lock
(priority-donate-one) acquire1: done
(priority-donate-one) acquire2, acquire1 must already have finished, in that order.
(priority-donate-one) This should be the last line before finishing this test.
(priority-donate-one) end
Execution of 'priority-donate-one' complete.
Timer: 26 ticks
Thread: 0 idle ticks, 26 kernel ticks, 0 user ticks
Console: 901 characters output
Keyboard: 0 keys pressed
Powering off...
```



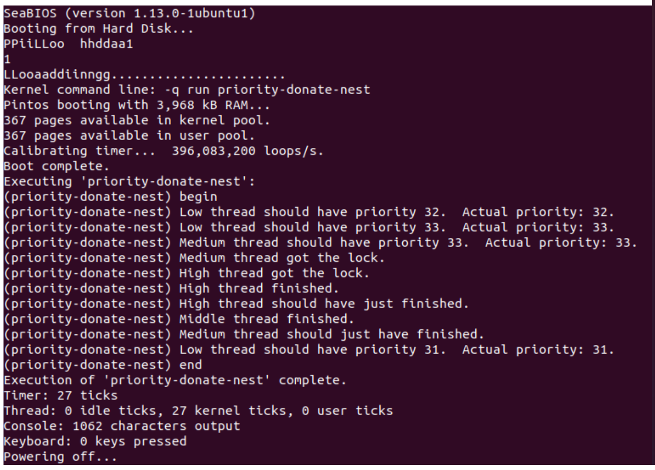
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q run priority-donate-multiple
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,950,400 loops/s.
Boot complete.
Executing 'priority-donate-multiple':
(priority-donate-multiple) begin
(priority-donate-multiple) Main thread should have priority 32.  Actual priority: 32.
(priority-donate-multiple) Main thread should have priority 33.  Actual priority: 33.
(priority-donate-multiple) Thread b acquired lock b.
(priority-donate-multiple) Thread b finished.
(priority-donate-multiple) Thread b should have just finished.
(priority-donate-multiple) Main thread should have priority 32.  Actual priority: 32.
(priority-donate-multiple) Thread a acquired lock a.
(priority-donate-multiple) Thread a finished.
(priority-donate-multiple) Thread a should have just finished.
(priority-donate-multiple) Main thread should have priority 31.  Actual priority: 31.
(priority-donate-multiple) end
Execution of 'priority-donate-multiple' complete.
Timer: 28 ticks
Thread: 0 idle ticks, 28 kernel ticks, 0 user ticks
Console: 1105 characters output
Keyboard: 0 keys pressed
Powering off...
```



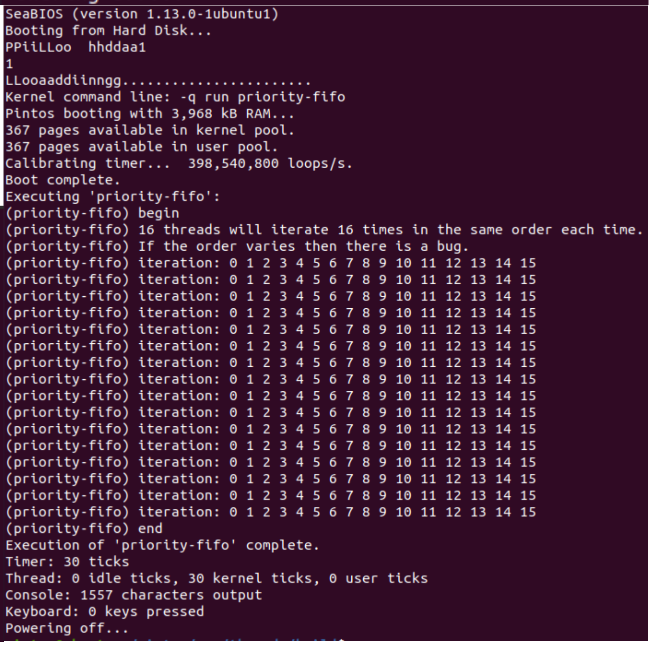
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q run priority-donate-multiple2
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  399,769,600 loops/s.
Boot complete.
Executing 'priority-donate-multiple2':
(priority-donate-multiple2) begin
(priority-donate-multiple2) Main thread should have priority 34.  Actual priority: 34.
(priority-donate-multiple2) Main thread should have priority 36.  Actual priority: 36.
(priority-donate-multiple2) Main thread should have priority 36.  Actual priority: 36.
(priority-donate-multiple2) Thread b acquired lock b.
(priority-donate-multiple2) Thread b finished.
(priority-donate-multiple2) Thread a acquired lock a.
(priority-donate-multiple2) Thread a finished.
(priority-donate-multiple2) Thread c finished.
(priority-donate-multiple2) Threads b, a, c should have just finished, in that order.
(priority-donate-multiple2) Main thread should have priority 31.  Actual priority: 31.
(priority-donate-multiple2) end
Execution of 'priority-donate-multiple2' complete.
Timer: 30 ticks
Thread: 0 idle ticks, 30 kernel ticks, 0 user ticks
Console: 1125 characters output
Keyboard: 0 keys pressed
Powering off...
```



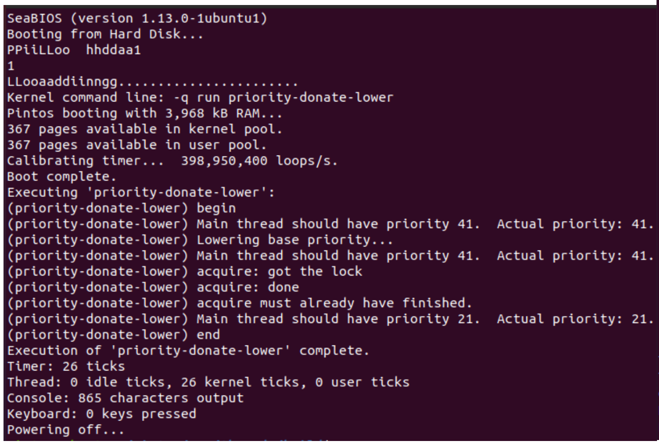
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-donate-sema
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  393,625,600 loops/s.
Boot complete.
Executing 'priority-donate-sema':
(priority-donate-sema) begin
(priority-donate-sema) Thread L acquired lock.
(priority-donate-sema) Thread L downed semaphore.
(priority-donate-sema) Thread H acquired lock.
(priority-donate-sema) Thread H finished.
(priority-donate-sema) Thread M finished.
(priority-donate-sema) Thread L finished.
(priority-donate-sema) Main thread finished.
(priority-donate-sema) end
Execution of 'priority-donate-sema' complete.
Timer: 28 ticks
Thread: 0 idle ticks, 28 kernel ticks, 0 user ticks
Console: 732 characters output
Keyboard: 0 keys pressed
Powering off...
```



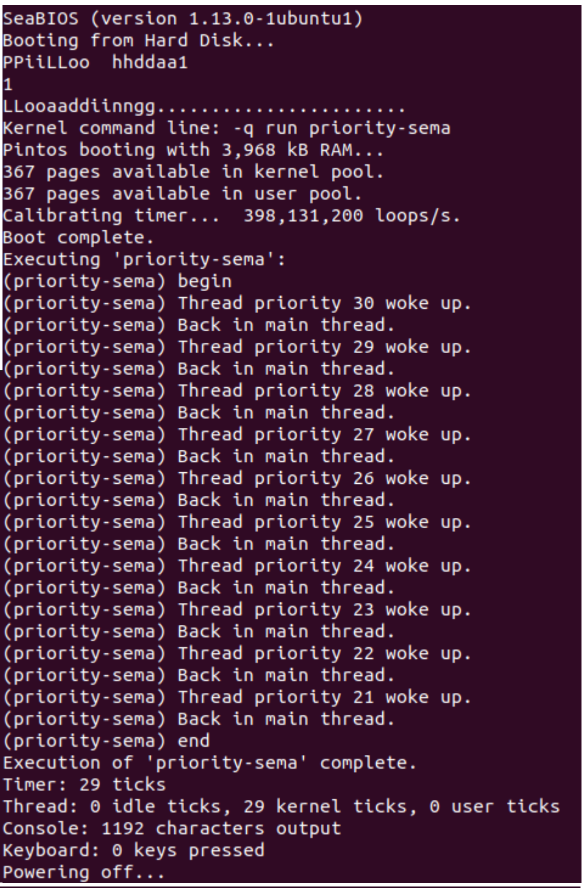
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-donate-nest
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  396,083,200 loops/s.
Boot complete.
Executing 'priority-donate-nest':
(priority-donate-nest) begin
(priority-donate-nest) Low thread should have priority 32.  Actual priority: 32.
(priority-donate-nest) Low thread should have priority 33.  Actual priority: 33.
(priority-donate-nest) Medium thread should have priority 33.  Actual priority: 33.
(priority-donate-nest) Medium thread got the lock.
(priority-donate-nest) High thread got the lock.
(priority-donate-nest) High thread finished.
(priority-donate-nest) High thread should have just finished.
(priority-donate-nest) Middle thread finished.
(priority-donate-nest) Medium thread should just have finished.
(priority-donate-nest) Low thread should have priority 31.  Actual priority: 31.
(priority-donate-nest) end
Execution of 'priority-donate-nest' complete.
Timer: 27 ticks
Thread: 0 idle ticks, 27 kernel ticks, 0 user ticks
Console: 1062 characters output
Keyboard: 0 keys pressed
Powering off...
```



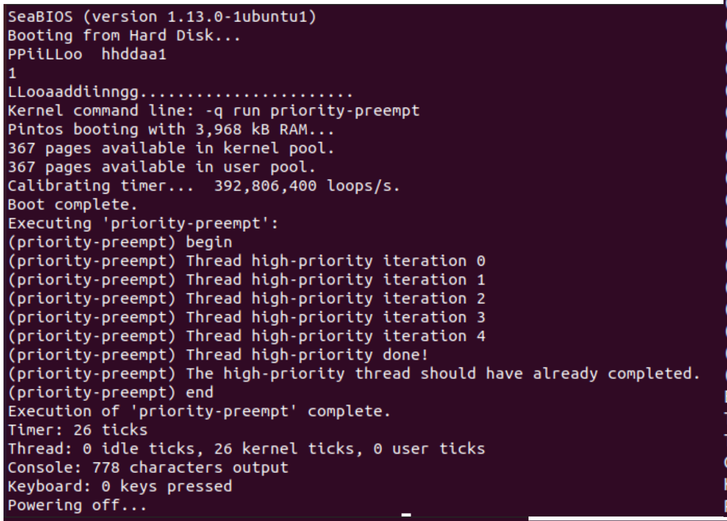
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-fifo
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,540,800 loops/s.
Boot complete.
Executing 'priority-fifo':
(priority-fifo) begin
(priority-fifo) 16 threads will iterate 16 times in the same order each time.
(priority-fifo) If the order varies then there is a bug.
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) end
Execution of 'priority-fifo' complete.
Timer: 30 ticks
Thread: 0 idle ticks, 30 kernel ticks, 0 user ticks
Console: 1557 characters output
Keyboard: 0 keys pressed
Powering off...
```



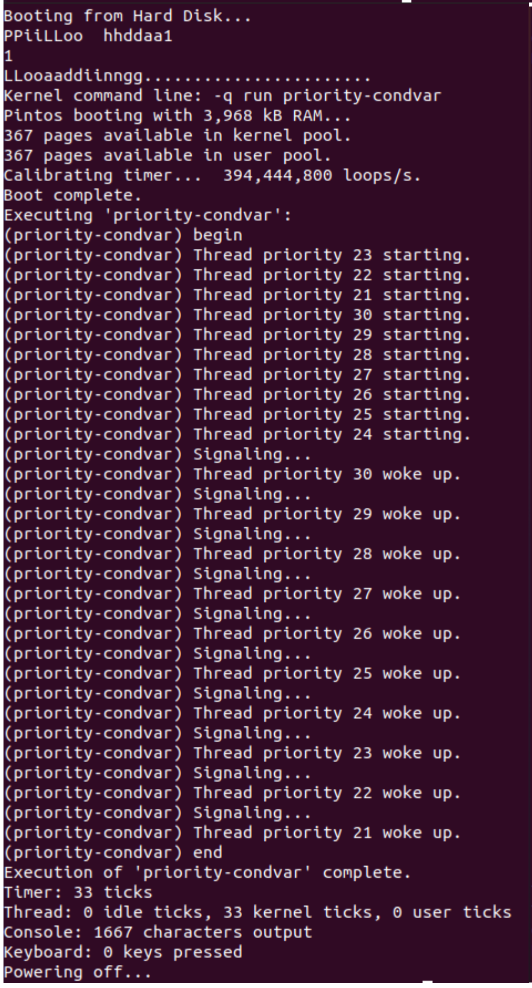
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-donate-lower
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,950,400 loops/s.
Boot complete.
Executing 'priority-donate-lower':
(priority-donate-lower) begin
(priority-donate-lower) Main thread should have priority 41.  Actual priority: 41.
(priority-donate-lower) Lowering base priority...
(priority-donate-lower) Main thread should have priority 41.  Actual priority: 41.
(priority-donate-lower) acquire: got the lock
(priority-donate-lower) acquire: done
(priority-donate-lower) acquire must already have finished.
(priority-donate-lower) Main thread should have priority 21.  Actual priority: 21.
(priority-donate-lower) end
Execution of 'priority-donate-lower' complete.
Timer: 26 ticks
Thread: 0 idle ticks, 26 kernel ticks, 0 user ticks
Console: 865 characters output
Keyboard: 0 keys pressed
Powering off...
```



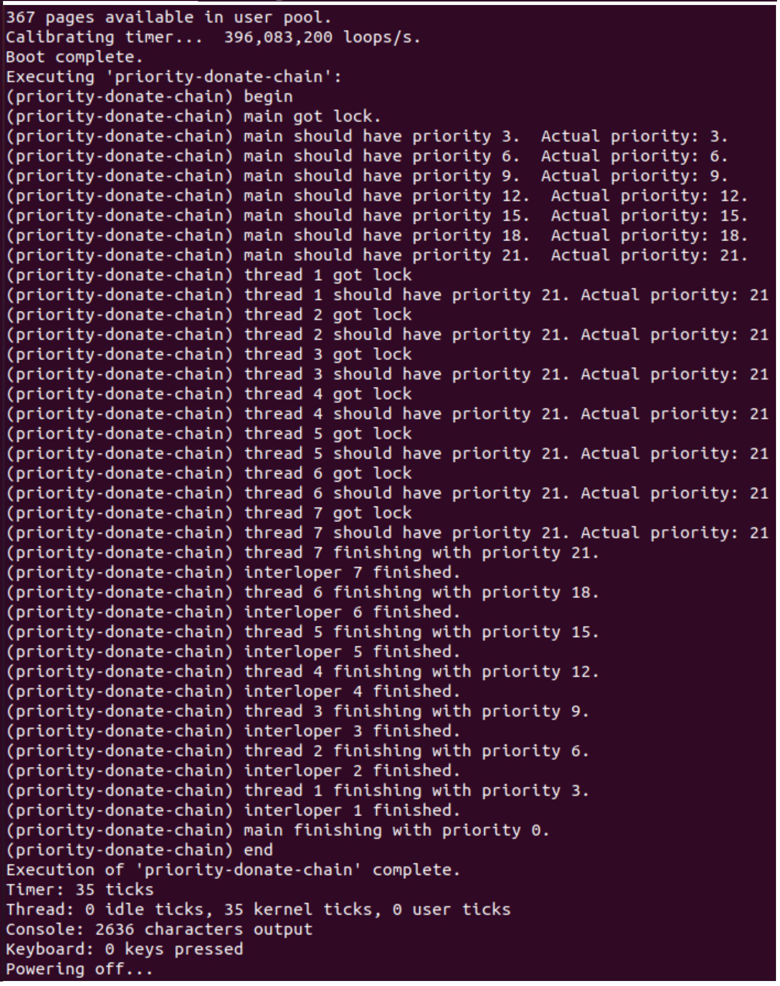
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-sema
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,131,200 loops/s.
Boot complete.
Executing 'priority-sema':
(priority-sema) begin
(priority-sema) Thread priority 30 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 29 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 28 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 27 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 26 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 25 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 24 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 23 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 22 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 21 woke up.
(priority-sema) Back in main thread.
(priority-sema) end
Execution of 'priority-sema' complete.
Timer: 29 ticks
Thread: 0 idle ticks, 29 kernel ticks, 0 user ticks
Console: 1192 characters output
Keyboard: 0 keys pressed
Powering off...
```



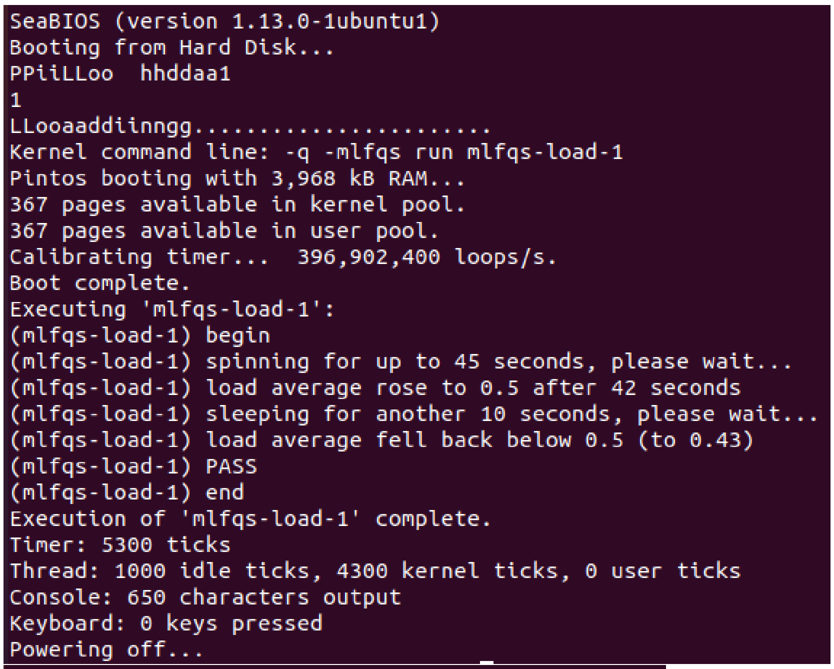
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-preempt
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  392,806,400 loops/s.
Boot complete.
Executing 'priority-preempt':
(priority-preempt) begin
(priority-preempt) Thread high-priority iteration 0
(priority-preempt) Thread high-priority iteration 1
(priority-preempt) Thread high-priority iteration 2
(priority-preempt) Thread high-priority iteration 3
(priority-preempt) Thread high-priority iteration 4
(priority-preempt) Thread high-priority done!
(priority-preempt) The high-priority thread should have already completed.
(priority-preempt) end
Execution of 'priority-preempt' complete.
Timer: 26 ticks
Thread: 0 idle ticks, 26 kernel ticks, 0 user ticks
Console: 778 characters output
Keyboard: 0 keys pressed
Powering off...
```



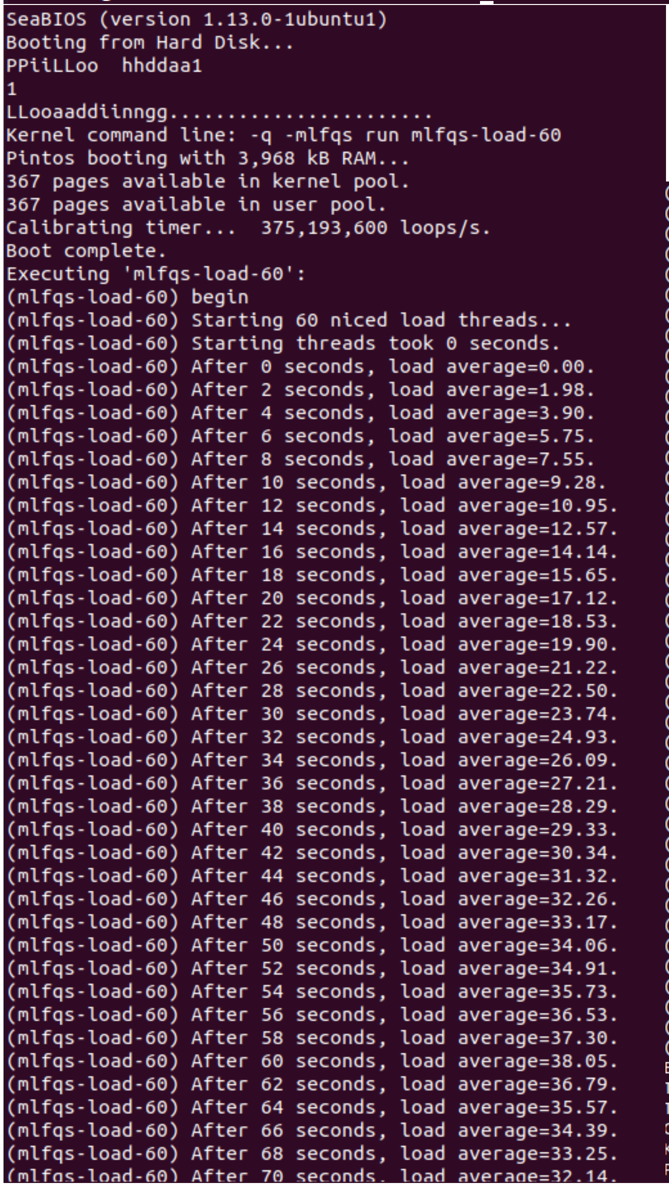
```
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q run priority-condvar
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  394,444,800 loops/s.
Boot complete.
Executing 'priority-condvar':
(priority-condvar) begin
(priority-condvar) Thread priority 23 starting.
(priority-condvar) Thread priority 22 starting.
(priority-condvar) Thread priority 21 starting.
(priority-condvar) Thread priority 30 starting.
(priority-condvar) Thread priority 29 starting.
(priority-condvar) Thread priority 28 starting.
(priority-condvar) Thread priority 27 starting.
(priority-condvar) Thread priority 26 starting.
(priority-condvar) Thread priority 25 starting.
(priority-condvar) Thread priority 24 starting.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 30 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 29 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 28 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 27 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 26 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 25 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 24 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 23 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 22 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 21 woke up.
(priority-condvar) end
Execution of 'priority-condvar' complete.
Timer: 33 ticks
Thread: 0 idle ticks, 33 kernel ticks, 0 user ticks
Console: 1667 characters output
Keyboard: 0 keys pressed
Powering off...
```



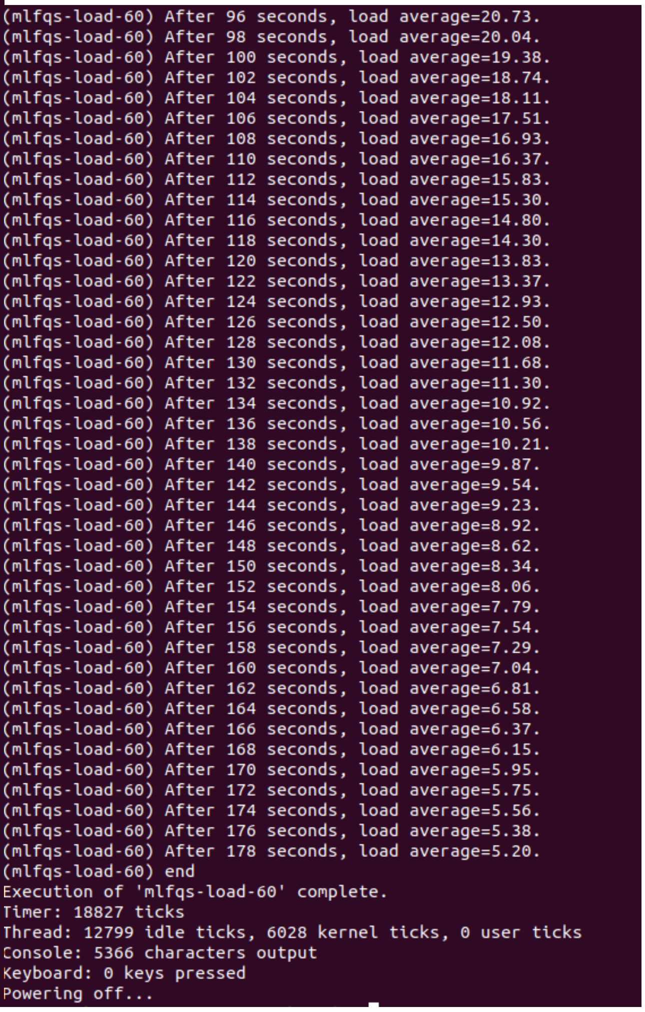
```
367 pages available in user pool.
Calibrating timer...  396,083,200 loops/s.
Boot complete.
Executing 'priority-donate-chain':
(priority-donate-chain) begin
(priority-donate-chain) main got lock.
(priority-donate-chain) main should have priority 3.  Actual priority: 3.
(priority-donate-chain) main should have priority 6.  Actual priority: 6.
(priority-donate-chain) main should have priority 9.  Actual priority: 9.
(priority-donate-chain) main should have priority 12.  Actual priority: 12.
(priority-donate-chain) main should have priority 15.  Actual priority: 15.
(priority-donate-chain) main should have priority 18.  Actual priority: 18.
(priority-donate-chain) main should have priority 21.  Actual priority: 21.
(priority-donate-chain) thread 1 got lock
(priority-donate-chain) thread 1 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 2 got lock
(priority-donate-chain) thread 2 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 3 got lock
(priority-donate-chain) thread 3 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 4 got lock
(priority-donate-chain) thread 4 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 5 got lock
(priority-donate-chain) thread 5 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 6 got lock
(priority-donate-chain) thread 6 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 7 got lock
(priority-donate-chain) thread 7 should have priority 21. Actual priority: 21
(priority-donate-chain) thread 7 finishing with priority 21.
(priority-donate-chain) interloper 7 finished.
(priority-donate-chain) thread 6 finishing with priority 18.
(priority-donate-chain) interloper 6 finished.
(priority-donate-chain) thread 5 finishing with priority 15.
(priority-donate-chain) interloper 5 finished.
(priority-donate-chain) thread 4 finishing with priority 12.
(priority-donate-chain) interloper 4 finished.
(priority-donate-chain) thread 3 finishing with priority 9.
(priority-donate-chain) interloper 3 finished.
(priority-donate-chain) thread 2 finishing with priority 6.
(priority-donate-chain) interloper 2 finished.
(priority-donate-chain) thread 1 finishing with priority 3.
(priority-donate-chain) interloper 1 finished.
(priority-donate-chain) main finishing with priority 0.
(priority-donate-chain) end
Execution of 'priority-donate-chain' complete.
Timer: 35 ticks
Thread: 0 idle ticks, 35 kernel ticks, 0 user ticks
Console: 2636 characters output
Keyboard: 0 keys pressed
Powering off...
```



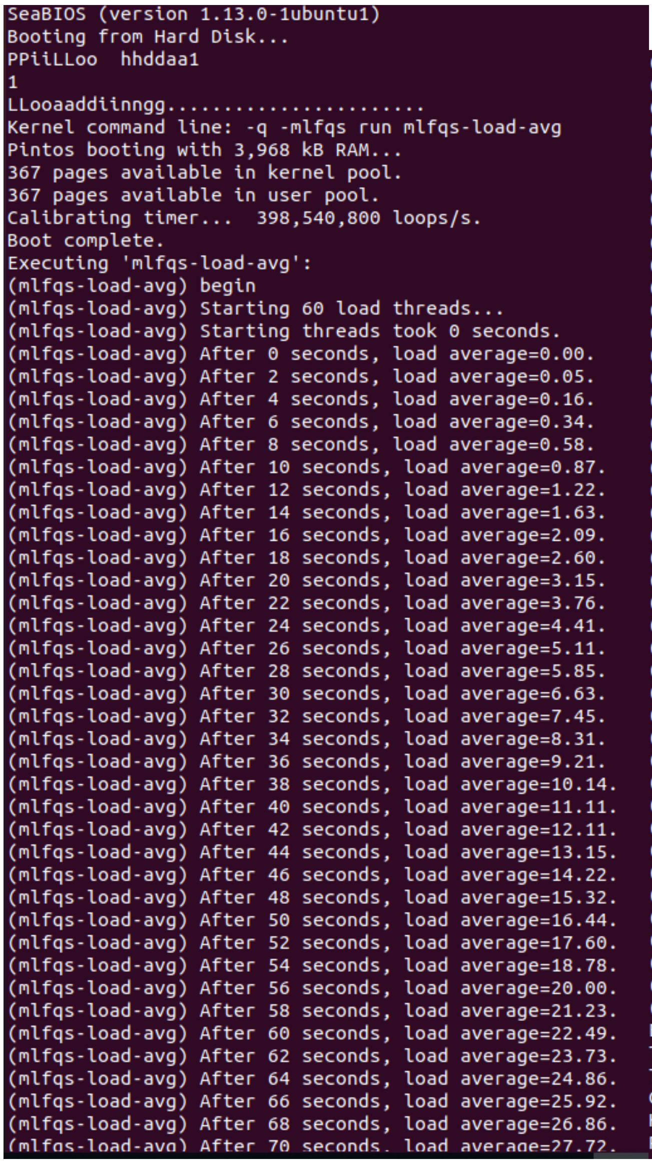
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-load-1
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  396,902,400 loops/s.
Boot complete.
Executing 'mlfqs-load-1':
(mlfqs-load-1) begin
(mlfqs-load-1) spinning for up to 45 seconds, please wait...
(mlfqs-load-1) load average rose to 0.5 after 42 seconds
(mlfqs-load-1) sleeping for another 10 seconds, please wait...
(mlfqs-load-1) load average fell back below 0.5 (to 0.43)
(mlfqs-load-1) PASS
(mlfqs-load-1) end
Execution of 'mlfqs-load-1' complete.
Timer: 5300 ticks
Thread: 1000 idle ticks, 4300 kernel ticks, 0 user ticks
Console: 650 characters output
Keyboard: 0 keys pressed
Powering off...
```



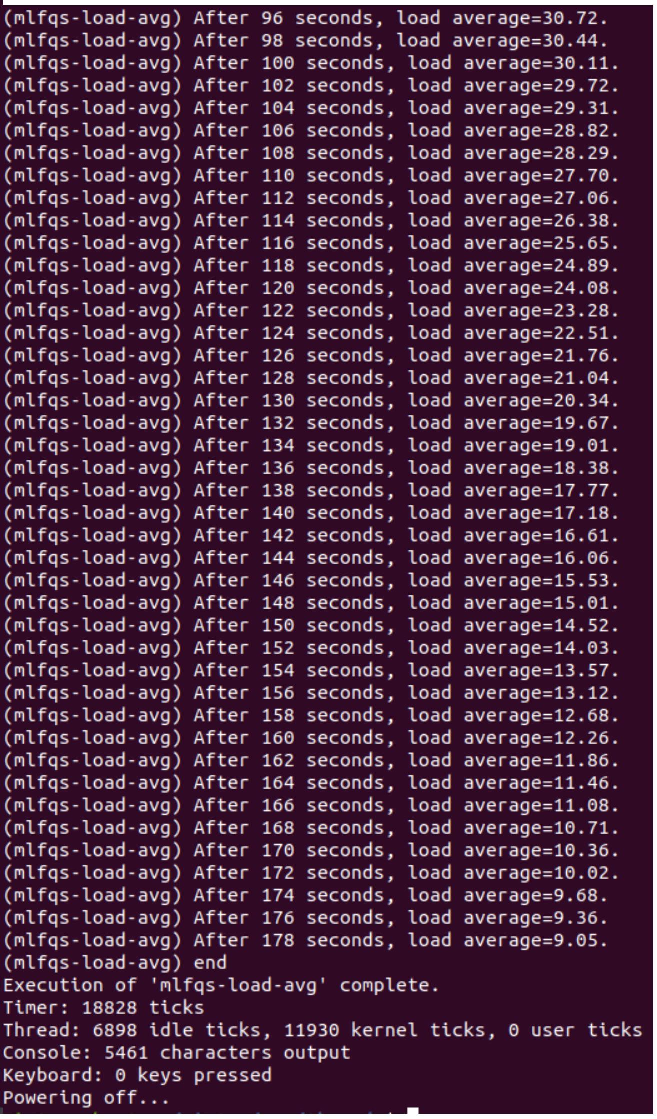
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-load-60
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  375,193,600 loops/s.
Boot complete.
Executing 'mlfqs-load-60':
(mlfqs-load-60) begin
(mlfqs-load-60) Starting 60 niced load threads...
(mlfqs-load-60) Starting threads took 0 seconds.
(mlfqs-load-60) After 0 seconds, load average=0.00.
(mlfqs-load-60) After 2 seconds, load average=1.98.
(mlfqs-load-60) After 4 seconds, load average=3.90.
(mlfqs-load-60) After 6 seconds, load average=5.75.
(mlfqs-load-60) After 8 seconds, load average=7.55.
(mlfqs-load-60) After 10 seconds, load average=9.28.
(mlfqs-load-60) After 12 seconds, load average=10.95.
(mlfqs-load-60) After 14 seconds, load average=12.57.
(mlfqs-load-60) After 16 seconds, load average=14.14.
(mlfqs-load-60) After 18 seconds, load average=15.65.
(mlfqs-load-60) After 20 seconds, load average=17.12.
(mlfqs-load-60) After 22 seconds, load average=18.53.
(mlfqs-load-60) After 24 seconds, load average=19.90.
(mlfqs-load-60) After 26 seconds, load average=21.22.
(mlfqs-load-60) After 28 seconds, load average=22.50.
(mlfqs-load-60) After 30 seconds, load average=23.74.
(mlfqs-load-60) After 32 seconds, load average=24.93.
(mlfqs-load-60) After 34 seconds, load average=26.09.
(mlfqs-load-60) After 36 seconds, load average=27.21.
(mlfqs-load-60) After 38 seconds, load average=28.29.
(mlfqs-load-60) After 40 seconds, load average=29.33.
(mlfqs-load-60) After 42 seconds, load average=30.34.
(mlfqs-load-60) After 44 seconds, load average=31.32.
(mlfqs-load-60) After 46 seconds, load average=32.26.
(mlfqs-load-60) After 48 seconds, load average=33.17.
(mlfqs-load-60) After 50 seconds, load average=34.06.
(mlfqs-load-60) After 52 seconds, load average=34.91.
(mlfqs-load-60) After 54 seconds, load average=35.73.
(mlfqs-load-60) After 56 seconds, load average=36.53.
(mlfqs-load-60) After 58 seconds, load average=37.30.
(mlfqs-load-60) After 60 seconds, load average=38.05.
(mlfqs-load-60) After 62 seconds, load average=36.79.
(mlfqs-load-60) After 64 seconds, load average=35.57.
(mlfqs-load-60) After 66 seconds, load average=34.39.
(mlfqs-load-60) After 68 seconds, load average=33.25.
(mlfqs-load-60) After 70 seconds, load average=32.14.
```



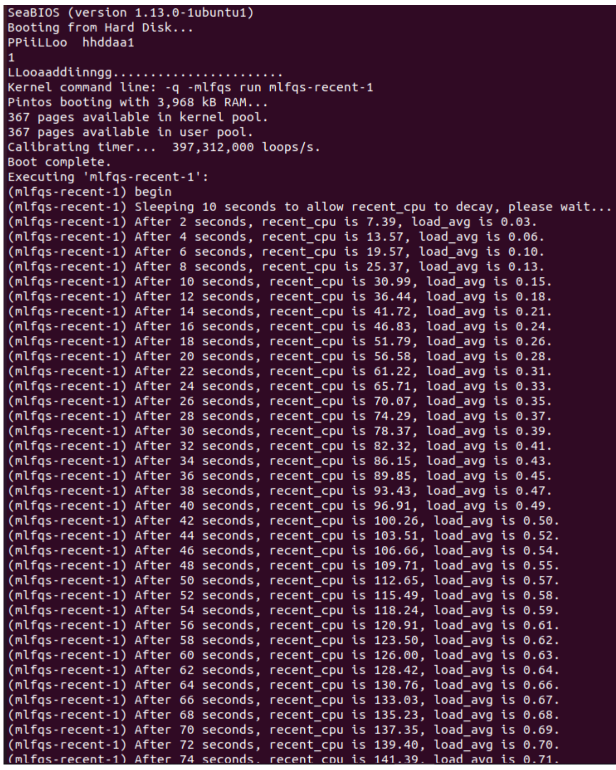
```
(mlfqs-load-60) After 96 seconds, load average=20.73.
(mlfqs-load-60) After 98 seconds, load average=20.04.
(mlfqs-load-60) After 100 seconds, load average=19.38.
(mlfqs-load-60) After 102 seconds, load average=18.74.
(mlfqs-load-60) After 104 seconds, load average=18.11.
(mlfqs-load-60) After 106 seconds, load average=17.51.
(mlfqs-load-60) After 108 seconds, load average=16.93.
(mlfqs-load-60) After 110 seconds, load average=16.37.
(mlfqs-load-60) After 112 seconds, load average=15.83.
(mlfqs-load-60) After 114 seconds, load average=15.30.
(mlfqs-load-60) After 116 seconds, load average=14.80.
(mlfqs-load-60) After 118 seconds, load average=14.30.
(mlfqs-load-60) After 120 seconds, load average=13.83.
(mlfqs-load-60) After 122 seconds, load average=13.37.
(mlfqs-load-60) After 124 seconds, load average=12.93.
(mlfqs-load-60) After 126 seconds, load average=12.50.
(mlfqs-load-60) After 128 seconds, load average=12.08.
(mlfqs-load-60) After 130 seconds, load average=11.68.
(mlfqs-load-60) After 132 seconds, load average=11.30.
(mlfqs-load-60) After 134 seconds, load average=10.92.
(mlfqs-load-60) After 136 seconds, load average=10.56.
(mlfqs-load-60) After 138 seconds, load average=10.21.
(mlfqs-load-60) After 140 seconds, load average=9.87.
(mlfqs-load-60) After 142 seconds, load average=9.54.
(mlfqs-load-60) After 144 seconds, load average=9.23.
(mlfqs-load-60) After 146 seconds, load average=8.92.
(mlfqs-load-60) After 148 seconds, load average=8.62.
(mlfqs-load-60) After 150 seconds, load average=8.34.
(mlfqs-load-60) After 152 seconds, load average=8.06.
(mlfqs-load-60) After 154 seconds, load average=7.79.
(mlfqs-load-60) After 156 seconds, load average=7.54.
(mlfqs-load-60) After 158 seconds, load average=7.29.
(mlfqs-load-60) After 160 seconds, load average=7.04.
(mlfqs-load-60) After 162 seconds, load average=6.81.
(mlfqs-load-60) After 164 seconds, load average=6.58.
(mlfqs-load-60) After 166 seconds, load average=6.37.
(mlfqs-load-60) After 168 seconds, load average=6.15.
(mlfqs-load-60) After 170 seconds, load average=5.95.
(mlfqs-load-60) After 172 seconds, load average=5.75.
(mlfqs-load-60) After 174 seconds, load average=5.56.
(mlfqs-load-60) After 176 seconds, load average=5.38.
(mlfqs-load-60) After 178 seconds, load average=5.20.
(mlfqs-load-60) end
Execution of 'mlfqs-load-60' complete.
Timer: 18827 ticks
Thread: 12799 idle ticks, 6028 kernel ticks, 0 user ticks
Console: 5366 characters output
Keyboard: 0 keys pressed
Powering off...
```



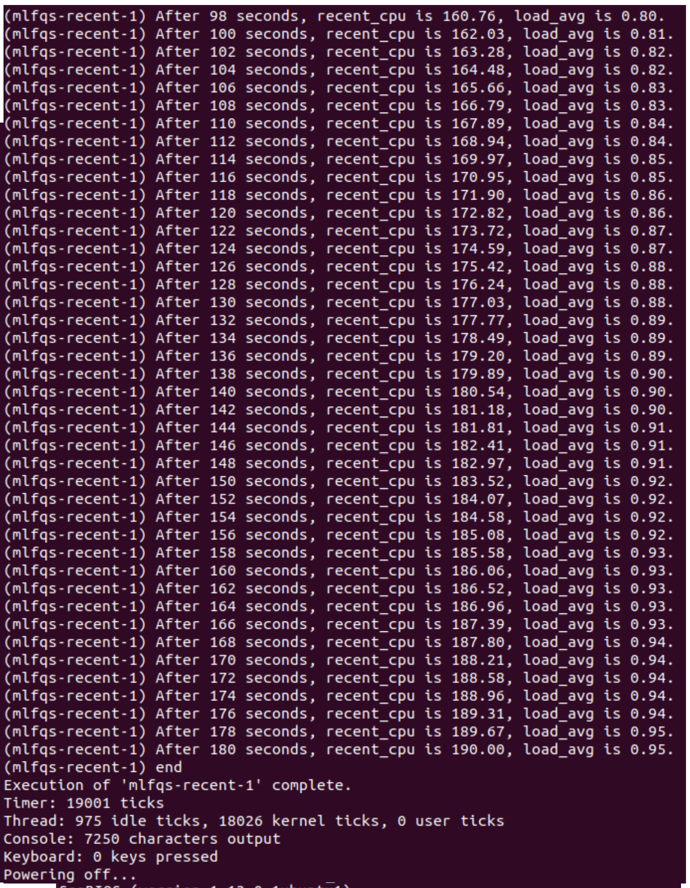
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg.......................
Kernel command line: -q -mlfqs run mlfqs-load-avg
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,540,800 loops/s.
Boot complete.
Executing 'mlfqs-load-avg':
(mlfqs-load-avg) begin
(mlfqs-load-avg) Starting 60 load threads...
(mlfqs-load-avg) Starting threads took 0 seconds.
(mlfqs-load-avg) After 0 seconds, load average=0.00.
(mlfqs-load-avg) After 2 seconds, load average=0.05.
(mlfqs-load-avg) After 4 seconds, load average=0.16.
(mlfqs-load-avg) After 6 seconds, load average=0.34.
(mlfqs-load-avg) After 8 seconds, load average=0.58.
(mlfqs-load-avg) After 10 seconds, load average=0.87.
(mlfqs-load-avg) After 12 seconds, load average=1.22.
(mlfqs-load-avg) After 14 seconds, load average=1.63.
(mlfqs-load-avg) After 16 seconds, load average=2.09.
(mlfqs-load-avg) After 18 seconds, load average=2.60.
(mlfqs-load-avg) After 20 seconds, load average=3.15.
(mlfqs-load-avg) After 22 seconds, load average=3.76.
(mlfqs-load-avg) After 24 seconds, load average=4.41.
(mlfqs-load-avg) After 26 seconds, load average=5.11.
(mlfqs-load-avg) After 28 seconds, load average=5.85.
(mlfqs-load-avg) After 30 seconds, load average=6.63.
(mlfqs-load-avg) After 32 seconds, load average=7.45.
(mlfqs-load-avg) After 34 seconds, load average=8.31.
(mlfqs-load-avg) After 36 seconds, load average=9.21.
(mlfqs-load-avg) After 38 seconds, load average=10.14.
(mlfqs-load-avg) After 40 seconds, load average=11.11.
(mlfqs-load-avg) After 42 seconds, load average=12.11.
(mlfqs-load-avg) After 44 seconds, load average=13.15.
(mlfqs-load-avg) After 46 seconds, load average=14.22.
(mlfqs-load-avg) After 48 seconds, load average=15.32.
(mlfqs-load-avg) After 50 seconds, load average=16.44.
(mlfqs-load-avg) After 52 seconds, load average=17.60.
(mlfqs-load-avg) After 54 seconds, load average=18.78.
(mlfqs-load-avg) After 56 seconds, load average=20.00.
(mlfqs-load-avg) After 58 seconds, load average=21.23.
(mlfqs-load-avg) After 60 seconds, load average=22.49.
(mlfqs-load-avg) After 62 seconds, load average=23.73.
(mlfqs-load-avg) After 64 seconds, load average=24.86.
(mlfqs-load-avg) After 66 seconds, load average=25.92.
(mlfqs-load-avg) After 68 seconds, load average=26.86.
(mlfqs-load-avg) After 70 seconds, load average=27.72.
```



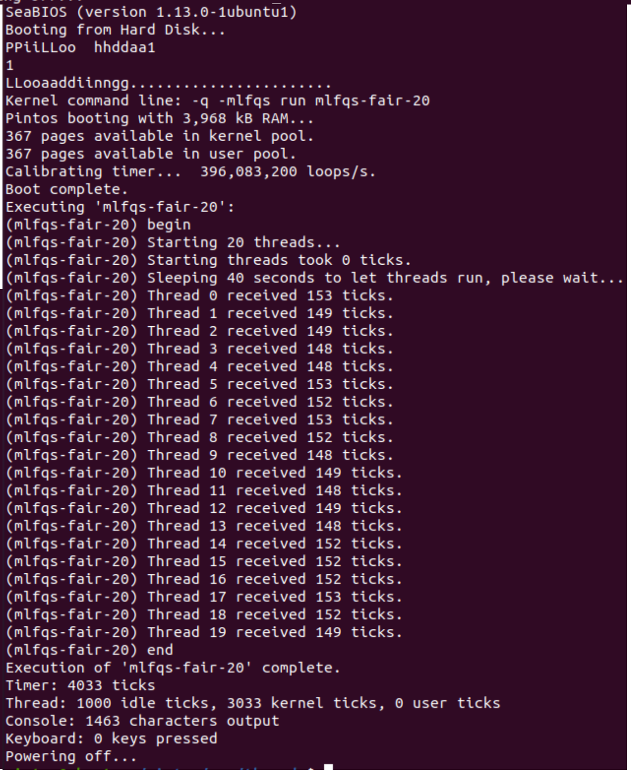
```
(mlfqs-load-avg) After 96 seconds, load average=30.72.
(mlfqs-load-avg) After 98 seconds, load average=30.44.
(mlfqs-load-avg) After 100 seconds, load average=30.11.
(mlfqs-load-avg) After 102 seconds, load average=29.72.
(mlfqs-load-avg) After 104 seconds, load average=29.31.
(mlfqs-load-avg) After 106 seconds, load average=28.82.
(mlfqs-load-avg) After 108 seconds, load average=28.29.
(mlfqs-load-avg) After 110 seconds, load average=27.70.
(mlfqs-load-avg) After 112 seconds, load average=27.06.
(mlfqs-load-avg) After 114 seconds, load average=26.38.
(mlfqs-load-avg) After 116 seconds, load average=25.65.
(mlfqs-load-avg) After 118 seconds, load average=24.89.
(mlfqs-load-avg) After 120 seconds, load average=24.08.
(mlfqs-load-avg) After 122 seconds, load average=23.28.
(mlfqs-load-avg) After 124 seconds, load average=22.51.
(mlfqs-load-avg) After 126 seconds, load average=21.76.
(mlfqs-load-avg) After 128 seconds, load average=21.04.
(mlfqs-load-avg) After 130 seconds, load average=20.34.
(mlfqs-load-avg) After 132 seconds, load average=19.67.
(mlfqs-load-avg) After 134 seconds, load average=19.01.
(mlfqs-load-avg) After 136 seconds, load average=18.38.
(mlfqs-load-avg) After 138 seconds, load average=17.77.
(mlfqs-load-avg) After 140 seconds, load average=17.18.
(mlfqs-load-avg) After 142 seconds, load average=16.61.
(mlfqs-load-avg) After 144 seconds, load average=16.06.
(mlfqs-load-avg) After 146 seconds, load average=15.53.
(mlfqs-load-avg) After 148 seconds, load average=15.01.
(mlfqs-load-avg) After 150 seconds, load average=14.52.
(mlfqs-load-avg) After 152 seconds, load average=14.03.
(mlfqs-load-avg) After 154 seconds, load average=13.57.
(mlfqs-load-avg) After 156 seconds, load average=13.12.
(mlfqs-load-avg) After 158 seconds, load average=12.68.
(mlfqs-load-avg) After 160 seconds, load average=12.26.
(mlfqs-load-avg) After 162 seconds, load average=11.86.
(mlfqs-load-avg) After 164 seconds, load average=11.46.
(mlfqs-load-avg) After 166 seconds, load average=11.08.
(mlfqs-load-avg) After 168 seconds, load average=10.71.
(mlfqs-load-avg) After 170 seconds, load average=10.36.
(mlfqs-load-avg) After 172 seconds, load average=10.02.
(mlfqs-load-avg) After 174 seconds, load average=9.68.
(mlfqs-load-avg) After 176 seconds, load average=9.36.
(mlfqs-load-avg) After 178 seconds, load average=9.05.
(mlfqs-load-avg) end
Execution of 'mlfqs-load-avg' complete.
Timer: 18828 ticks
Thread: 6898 idle ticks, 11930 kernel ticks, 0 user ticks
Console: 5461 characters output
Keyboard: 0 keys pressed
Powering off...
```



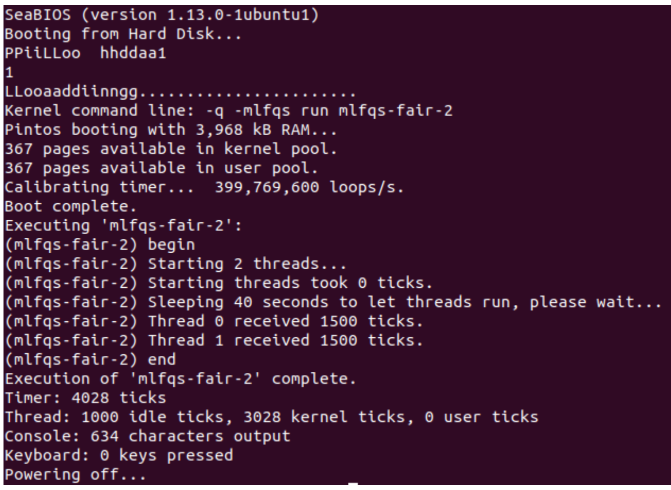
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-recent-1
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  397,312,000 loops/s.
Boot complete.
Executing 'mlfqs-recent-1':
(mlfqs-recent-1) begin
(mlfqs-recent-1) Sleeping 10 seconds to allow recent_cpu to decay, please wait...
(mlfqs-recent-1) After 2 seconds, recent_cpu is 7.39, load_avg is 0.03.
(mlfqs-recent-1) After 4 seconds, recent_cpu is 13.57, load_avg is 0.06.
(mlfqs-recent-1) After 6 seconds, recent_cpu is 19.57, load_avg is 0.10.
(mlfqs-recent-1) After 8 seconds, recent_cpu is 25.37, load_avg is 0.13.
(mlfqs-recent-1) After 10 seconds, recent_cpu is 30.99, load_avg is 0.15.
(mlfqs-recent-1) After 12 seconds, recent_cpu is 36.44, load_avg is 0.18.
(mlfqs-recent-1) After 14 seconds, recent_cpu is 41.72, load_avg is 0.21.
(mlfqs-recent-1) After 16 seconds, recent_cpu is 46.83, load_avg is 0.24.
(mlfqs-recent-1) After 18 seconds, recent_cpu is 51.79, load_avg is 0.26.
(mlfqs-recent-1) After 20 seconds, recent_cpu is 56.58, load_avg is 0.28.
(mlfqs-recent-1) After 22 seconds, recent_cpu is 61.22, load_avg is 0.31.
(mlfqs-recent-1) After 24 seconds, recent_cpu is 65.71, load_avg is 0.33.
(mlfqs-recent-1) After 26 seconds, recent_cpu is 70.07, load_avg is 0.35.
(mlfqs-recent-1) After 28 seconds, recent_cpu is 74.29, load_avg is 0.37.
(mlfqs-recent-1) After 30 seconds, recent_cpu is 78.37, load_avg is 0.39.
(mlfqs-recent-1) After 32 seconds, recent_cpu is 82.32, load_avg is 0.41.
(mlfqs-recent-1) After 34 seconds, recent_cpu is 86.15, load_avg is 0.43.
(mlfqs-recent-1) After 36 seconds, recent_cpu is 89.85, load_avg is 0.45.
(mlfqs-recent-1) After 38 seconds, recent_cpu is 93.43, load_avg is 0.47.
(mlfqs-recent-1) After 40 seconds, recent_cpu is 96.91, load_avg is 0.49.
(mlfqs-recent-1) After 42 seconds, recent_cpu is 100.26, load_avg is 0.50.
(mlfqs-recent-1) After 44 seconds, recent_cpu is 103.51, load_avg is 0.52.
(mlfqs-recent-1) After 46 seconds, recent_cpu is 106.66, load_avg is 0.54.
(mlfqs-recent-1) After 48 seconds, recent_cpu is 109.71, load_avg is 0.55.
(mlfqs-recent-1) After 50 seconds, recent_cpu is 112.65, load_avg is 0.57.
(mlfqs-recent-1) After 52 seconds, recent_cpu is 115.49, load_avg is 0.58.
(mlfqs-recent-1) After 54 seconds, recent_cpu is 118.24, load_avg is 0.59.
(mlfqs-recent-1) After 56 seconds, recent_cpu is 120.91, load_avg is 0.61.
(mlfqs-recent-1) After 58 seconds, recent_cpu is 123.50, load_avg is 0.62.
(mlfqs-recent-1) After 60 seconds, recent_cpu is 126.00, load_avg is 0.63.
(mlfqs-recent-1) After 62 seconds, recent_cpu is 128.42, load_avg is 0.64.
(mlfqs-recent-1) After 64 seconds, recent_cpu is 130.76, load_avg is 0.66.
(mlfqs-recent-1) After 66 seconds, recent_cpu is 133.03, load_avg is 0.67.
(mlfqs-recent-1) After 68 seconds, recent_cpu is 135.23, load_avg is 0.68.
(mlfqs-recent-1) After 70 seconds, recent_cpu is 137.35, load_avg is 0.69.
(mlfqs-recent-1) After 72 seconds, recent_cpu is 139.40, load_avg is 0.70.
(mlfqs-recent-1) After 74 seconds, recent_cpu is 141.39, load_avg is 0.71.
```



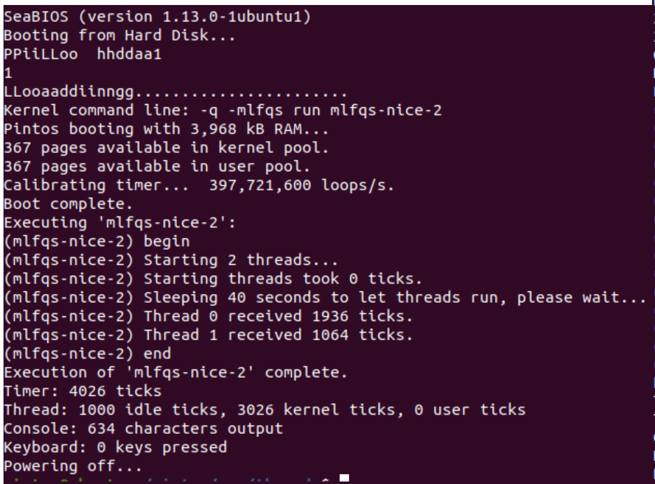
```
(mlfqs-recent-1) After 98 seconds, recent_cpu is 160.76, load_avg is 0.80.
(mlfqs-recent-1) After 100 seconds, recent_cpu is 162.03, load_avg is 0.81.
(mlfqs-recent-1) After 102 seconds, recent_cpu is 163.28, load_avg is 0.82.
(mlfqs-recent-1) After 104 seconds, recent_cpu is 164.48, load_avg is 0.82.
(mlfqs-recent-1) After 106 seconds, recent_cpu is 165.66, load_avg is 0.83.
(mlfqs-recent-1) After 108 seconds, recent_cpu is 166.79, load_avg is 0.83.
(mlfqs-recent-1) After 110 seconds, recent_cpu is 167.89, load_avg is 0.84.
(mlfqs-recent-1) After 112 seconds, recent_cpu is 168.94, load_avg is 0.84.
(mlfqs-recent-1) After 114 seconds, recent_cpu is 169.97, load_avg is 0.85.
(mlfqs-recent-1) After 116 seconds, recent_cpu is 170.95, load_avg is 0.85.
(mlfqs-recent-1) After 118 seconds, recent_cpu is 171.90, load_avg is 0.86.
(mlfqs-recent-1) After 120 seconds, recent_cpu is 172.82, load_avg is 0.86.
(mlfqs-recent-1) After 122 seconds, recent_cpu is 173.72, load_avg is 0.87.
(mlfqs-recent-1) After 124 seconds, recent_cpu is 174.59, load_avg is 0.87.
(mlfqs-recent-1) After 126 seconds, recent_cpu is 175.42, load_avg is 0.88.
(mlfqs-recent-1) After 128 seconds, recent_cpu is 176.24, load_avg is 0.88.
(mlfqs-recent-1) After 130 seconds, recent_cpu is 177.03, load_avg is 0.88.
(mlfqs-recent-1) After 132 seconds, recent_cpu is 177.77, load_avg is 0.89.
(mlfqs-recent-1) After 134 seconds, recent_cpu is 178.49, load_avg is 0.89.
(mlfqs-recent-1) After 136 seconds, recent_cpu is 179.20, load_avg is 0.89.
(mlfqs-recent-1) After 138 seconds, recent_cpu is 179.89, load_avg is 0.90.
(mlfqs-recent-1) After 140 seconds, recent_cpu is 180.54, load_avg is 0.90.
(mlfqs-recent-1) After 142 seconds, recent_cpu is 181.18, load_avg is 0.90.
(mlfqs-recent-1) After 144 seconds, recent_cpu is 181.81, load_avg is 0.91.
(mlfqs-recent-1) After 146 seconds, recent_cpu is 182.41, load_avg is 0.91.
(mlfqs-recent-1) After 148 seconds, recent_cpu is 182.97, load_avg is 0.91.
(mlfqs-recent-1) After 150 seconds, recent_cpu is 183.52, load_avg is 0.92.
(mlfqs-recent-1) After 152 seconds, recent_cpu is 184.07, load_avg is 0.92.
(mlfqs-recent-1) After 154 seconds, recent_cpu is 184.58, load_avg is 0.92.
(mlfqs-recent-1) After 156 seconds, recent_cpu is 185.08, load_avg is 0.92.
(mlfqs-recent-1) After 158 seconds, recent_cpu is 185.58, load_avg is 0.93.
(mlfqs-recent-1) After 160 seconds, recent_cpu is 186.06, load_avg is 0.93.
(mlfqs-recent-1) After 162 seconds, recent_cpu is 186.52, load_avg is 0.93.
(mlfqs-recent-1) After 164 seconds, recent_cpu is 186.96, load_avg is 0.93.
(mlfqs-recent-1) After 166 seconds, recent_cpu is 187.39, load_avg is 0.93.
(mlfqs-recent-1) After 168 seconds, recent_cpu is 187.80, load_avg is 0.94.
(mlfqs-recent-1) After 170 seconds, recent_cpu is 188.21, load_avg is 0.94.
(mlfqs-recent-1) After 172 seconds, recent_cpu is 188.58, load_avg is 0.94.
(mlfqs-recent-1) After 174 seconds, recent_cpu is 188.96, load_avg is 0.94.
(mlfqs-recent-1) After 176 seconds, recent_cpu is 189.31, load_avg is 0.94.
(mlfqs-recent-1) After 178 seconds, recent_cpu is 189.67, load_avg is 0.95.
(mlfqs-recent-1) After 180 seconds, recent_cpu is 190.00, load_avg is 0.95.
(mlfqs-recent-1) end
Execution of 'mlfqs-recent-1' complete.
Timer: 19001 ticks
Thread: 975 idle ticks, 18026 kernel ticks, 0 user ticks
Console: 7250 characters output
Keyboard: 0 keys pressed
Powering off...
```



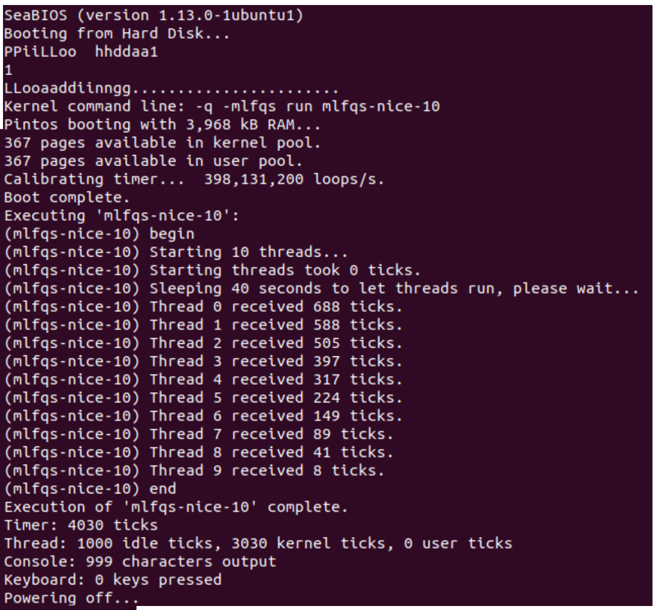
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-fair-20
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  396,083,200 loops/s.
Boot complete.
Executing 'mlfqs-fair-20':
(mlfqs-fair-20) begin
(mlfqs-fair-20) Starting 20 threads...
(mlfqs-fair-20) Starting threads took 0 ticks.
(mlfqs-fair-20) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-fair-20) Thread 0 received 153 ticks.
(mlfqs-fair-20) Thread 1 received 149 ticks.
(mlfqs-fair-20) Thread 2 received 149 ticks.
(mlfqs-fair-20) Thread 3 received 148 ticks.
(mlfqs-fair-20) Thread 4 received 148 ticks.
(mlfqs-fair-20) Thread 5 received 153 ticks.
(mlfqs-fair-20) Thread 6 received 152 ticks.
(mlfqs-fair-20) Thread 7 received 153 ticks.
(mlfqs-fair-20) Thread 8 received 152 ticks.
(mlfqs-fair-20) Thread 9 received 148 ticks.
(mlfqs-fair-20) Thread 10 received 149 ticks.
(mlfqs-fair-20) Thread 11 received 148 ticks.
(mlfqs-fair-20) Thread 12 received 149 ticks.
(mlfqs-fair-20) Thread 13 received 148 ticks.
(mlfqs-fair-20) Thread 14 received 152 ticks.
(mlfqs-fair-20) Thread 15 received 152 ticks.
(mlfqs-fair-20) Thread 16 received 152 ticks.
(mlfqs-fair-20) Thread 17 received 153 ticks.
(mlfqs-fair-20) Thread 18 received 152 ticks.
(mlfqs-fair-20) Thread 19 received 149 ticks.
(mlfqs-fair-20) end
Execution of 'mlfqs-fair-20' complete.
Timer: 4033 ticks
Thread: 1000 idle ticks, 3033 kernel ticks, 0 user ticks
Console: 1463 characters output
Keyboard: 0 keys pressed
Powering off...
```



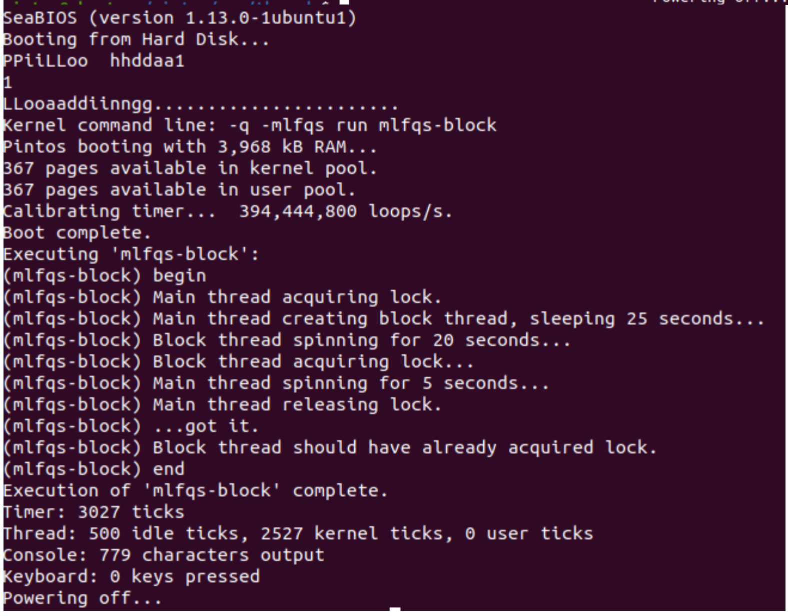
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-fair-2
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  399,769,600 loops/s.
Boot complete.
Executing 'mlfqs-fair-2':
(mlfqs-fair-2) begin
(mlfqs-fair-2) Starting 2 threads...
(mlfqs-fair-2) Starting threads took 0 ticks.
(mlfqs-fair-2) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-fair-2) Thread 0 received 1500 ticks.
(mlfqs-fair-2) Thread 1 received 1500 ticks.
(mlfqs-fair-2) end
Execution of 'mlfqs-fair-2' complete.
Timer: 4028 ticks
Thread: 1000 idle ticks, 3028 kernel ticks, 0 user ticks
Console: 634 characters output
Keyboard: 0 keys pressed
Powering off...
```



```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-nice-2
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  397,721,600 loops/s.
Boot complete.
Executing 'mlfqs-nice-2':
(mlfqs-nice-2) begin
(mlfqs-nice-2) Starting 2 threads...
(mlfqs-nice-2) Starting threads took 0 ticks.
(mlfqs-nice-2) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-nice-2) Thread 0 received 1936 ticks.
(mlfqs-nice-2) Thread 1 received 1064 ticks.
(mlfqs-nice-2) end
Execution of 'mlfqs-nice-2' complete.
Timer: 4026 ticks
Thread: 1000 idle ticks, 3026 kernel ticks, 0 user ticks
Console: 634 characters output
Keyboard: 0 keys pressed
Powering off...
```



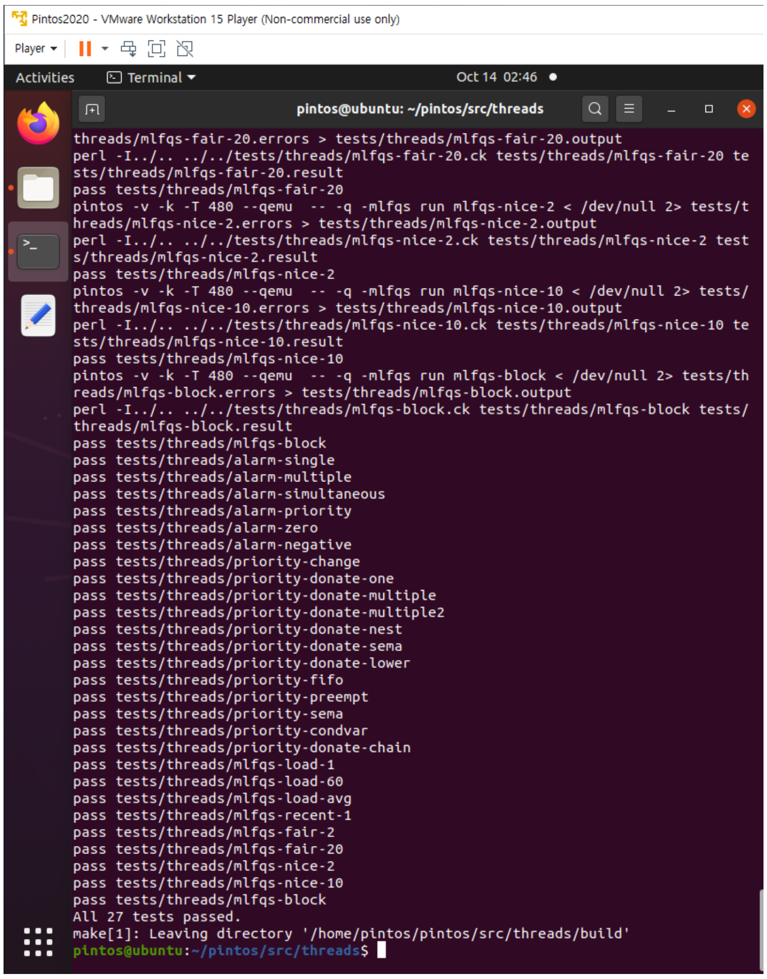
```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-nice-10
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  398,131,200 loops/s.
Boot complete.
Executing 'mlfqs-nice-10':
(mlfqs-nice-10) begin
(mlfqs-nice-10) Starting 10 threads...
(mlfqs-nice-10) Starting threads took 0 ticks.
(mlfqs-nice-10) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-nice-10) Thread 0 received 688 ticks.
(mlfqs-nice-10) Thread 1 received 588 ticks.
(mlfqs-nice-10) Thread 2 received 505 ticks.
(mlfqs-nice-10) Thread 3 received 397 ticks.
(mlfqs-nice-10) Thread 4 received 317 ticks.
(mlfqs-nice-10) Thread 5 received 224 ticks.
(mlfqs-nice-10) Thread 6 received 149 ticks.
(mlfqs-nice-10) Thread 7 received 89 ticks.
(mlfqs-nice-10) Thread 8 received 41 ticks.
(mlfqs-nice-10) Thread 9 received 8 ticks.
(mlfqs-nice-10) end
Execution of 'mlfqs-nice-10' complete.
Timer: 4030 ticks
Thread: 1000 idle ticks, 3030 kernel ticks, 0 user ticks
Console: 999 characters output
Keyboard: 0 keys pressed
Powering off...
```



```
SeaBIOS (version 1.13.0-1ubuntu1)
Booting from Hard Disk...
PPiiLLoo  hhddaa1
1
LLooaaddiinngg......................
Kernel command line: -q -mlfqs run mlfqs-block
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  394,444,800 loops/s.
Boot complete.
Executing 'mlfqs-block':
(mlfqs-block) begin
(mlfqs-block) Main thread acquiring lock.
(mlfqs-block) Main thread creating block thread, sleeping 25 seconds...
(mlfqs-block) Block thread spinning for 20 seconds...
(mlfqs-block) Block thread acquiring lock...
(mlfqs-block) Main thread spinning for 5 seconds...
(mlfqs-block) Main thread releasing lock.
(mlfqs-block) ...got it.
(mlfqs-block) Block thread should have already acquired lock.
(mlfqs-block) end
Execution of 'mlfqs-block' complete.
Timer: 3027 ticks
Thread: 500 idle ticks, 2527 kernel ticks, 0 user ticks
Console: 779 characters output
Keyboard: 0 keys pressed
Powering off...
```

6. Conclusion

이번 과제의 목표인 Alarm Clock, Priority Scheduling, Advanced Scheduler를 모두 성공적으로 구현하였다.


Pintos2020 - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Terminal
Oct 14 02:46
pintos@ubuntu: ~/pintos/src/threads

```
threads/mlfqs-fair-20.errors > tests/threads/mlfqs-fair-20.output
perl -I../.. ../../tests/threads/mlfqs-fair-20.ck tests/threads/mlfqs-fair-20 te
sts/threads/mlfqs-fair-20.result
pass tests/threads/mlfqs-fair-20
pintos -v -k -T 480 --qemu  -- -q -mlfqs run mlfqs-nice-2 < /dev/null 2> tests/t
hreads/mlfqs-nice-2.errors > tests/threads/mlfqs-nice-2.output
perl -I../.. ../../tests/threads/mlfqs-nice-2.ck tests/threads/mlfqs-nice-2 test
s/threads/mlfqs-nice-2.result
pass tests/threads/mlfqs-nice-2
pintos -v -k -T 480 --qemu  -- -q -mlfqs run mlfqs-nice-10 < /dev/null 2> tests/
threads/mlfqs-nice-10.errors > tests/threads/mlfqs-nice-10.output
perl -I../.. ../../tests/threads/mlfqs-nice-10.ck tests/threads/mlfqs-nice-10 te
sts/threads/mlfqs-nice-10.result
pass tests/threads/mlfqs-nice-10
pintos -v -k -T 480 --qemu  -- -q -mlfqs run mlfqs-block < /dev/null 2> tests/th
reads/mlfqs-block.errors > tests/threads/mlfqs-block.output
perl -I../.. ../../tests/threads/mlfqs-block.ck tests/threads/mlfqs-block tests/
threads/mlfqs-block.result
pass tests/threads/mlfqs-block
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
pass tests/threads/priority-donate-chain
pass tests/threads/mlfqs-load-1
pass tests/threads/mlfqs-load-60
pass tests/threads/mlfqs-load-avg
pass tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
pass tests/threads/mlfqs-nice-2
pass tests/threads/mlfqs-nice-10
pass tests/threads/mlfqs-block
All 27 tests passed.
make[1]: Leaving directory '/home/pintos/pintos/src/threads/build'
pintos@ubuntu:~/pintos/src/threads$
```

이번 과제를 진행하면서 처음부터 제공된 함수들 관의 상관관계를 모두 파악해야 했고, 복잡한 함수의 흐름 속에서 새로운 개념(priority donation 과 같은)을 도입하여 구현해야해서 어려웠다. 하지만, 함수의 흐름을 그려보고 구현해야 할 것이 함수들 가운데서 어떻게 동작해야 하는지 논리적인 흐름을 따라 구현하니 마침내 구현에 성공하였다. 이번 과제를 통해 OS가 어떻게 thread들의 실행을 관리하는지, 여러 복잡한 관계의 thread들 가운데 priority donation을 사용하여 어떻게 이를 해결할 수 있는지, mlfqs scheduler를 사용하여 어떻게 thread들을 관리하는지 자세히 알 수 있는 시간이었다. pintOS는 실제로 우리가 사용하는 OS보다 간단한 OS인데도 직접 구현하려고 하니 이렇게 복잡한 것을 보니 실제 우리가 사용하고 있는 OS는 정확히 어떤 구조로 이루어져 있고 pintOS에 비해 추가적으로 어떤 것들이 더 필요한지 궁금해졌다.