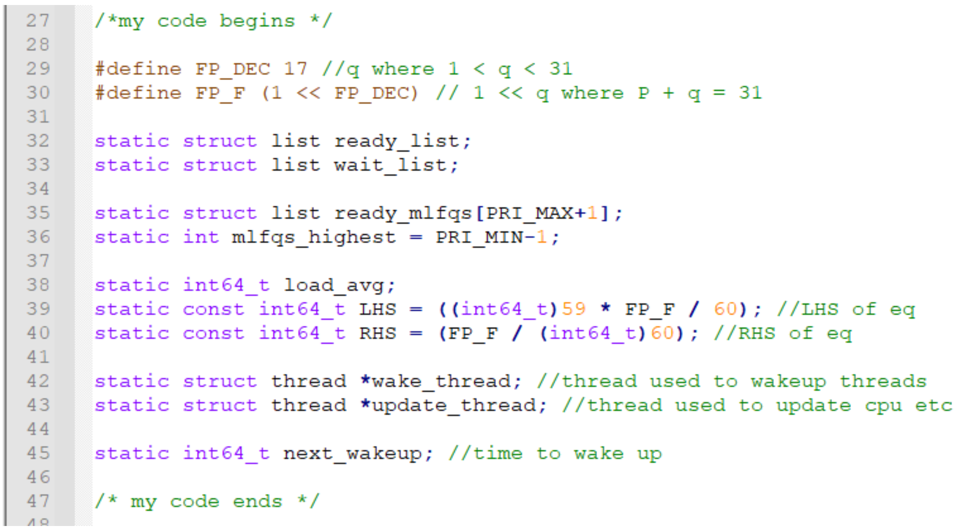
Code

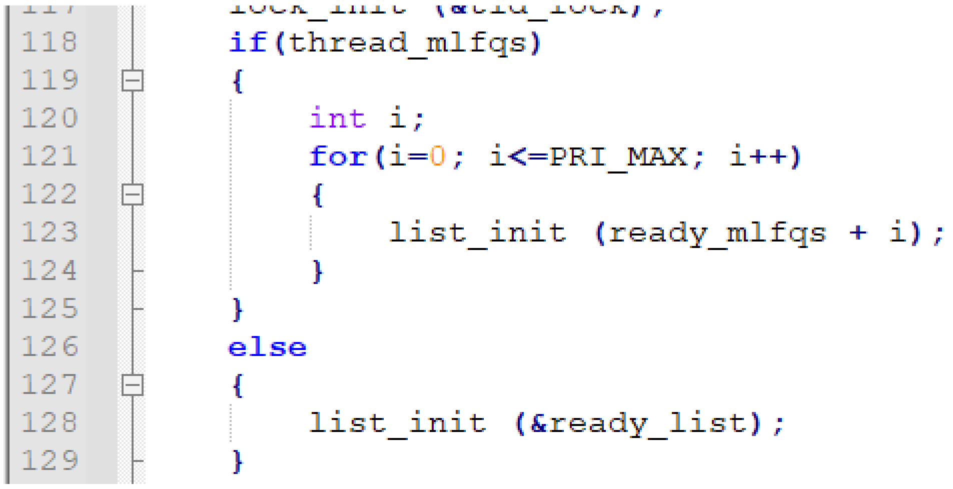
Static



```c
/*my code begins */

#define FP_DEC 17 //q where 1 < q < 31
#define FP_F (1 << FP_DEC) // 1 << q where P + q = 31

static struct list ready_list;
static struct list wait_list;

static struct list ready_mlfqs[PRI_MAX+1];
static int mlfqs_highest = PRI_MIN-1;

static int64_t load_avg;
static const int64_t LHS = ((int64_t)59 * FP_F / 60); //LHS of eq
static const int64_t RHS = (FP_F / (int64_t)60); //RHS of eq

static struct thread *wake_thread; //thread used to wakeup threads
static struct thread *update_thread; //thread used to update cpu etc

static int64_t next_wakeup; //time to wake up

/* my code ends */
```

Setting initial values of FP points used 17 for middle value. Ready list and wait list from p1 but added mlfqs queue and a counter for the highest value initialized at the lowest prio. Wake up thread and update thread used for respective purposes and next_wakeup from p1.

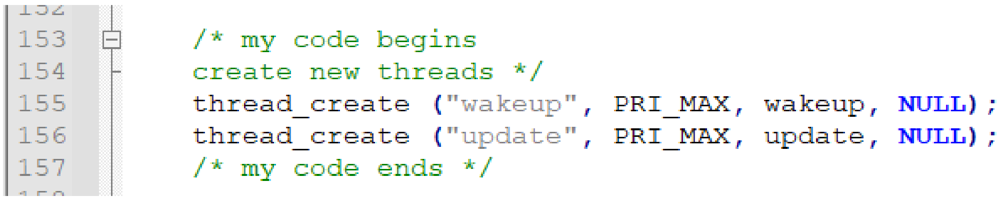
Thread_init



```c
  lock_init (&tid_lock);
  if(thread_mlfqs)
  {
      int i;
      for(i=0; i<=PRI_MAX; i++)
      {
          list_init (ready_mlfqs + i);
      }
  }
  else
  {
      list_init (&ready_list);
  }
```

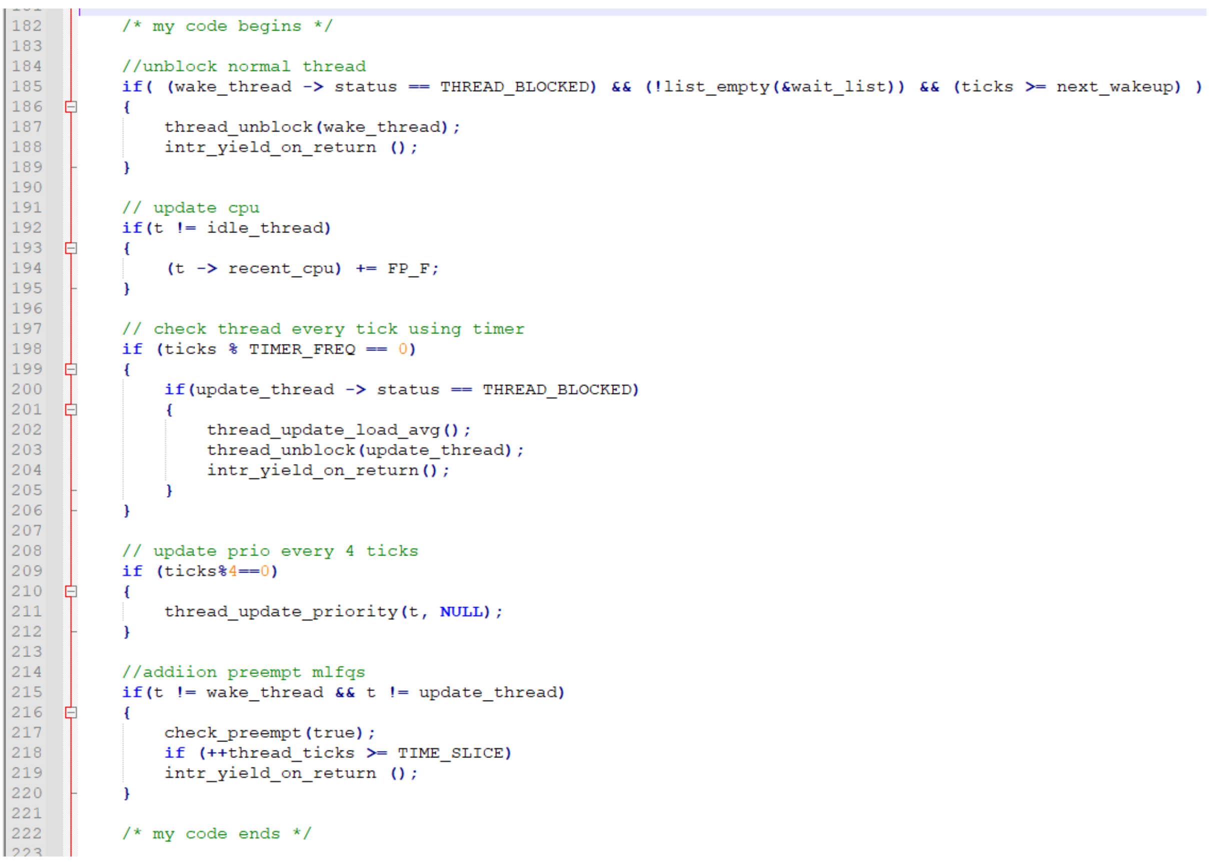
Check and run of thread type (mlfq vs normal) mlfq initialises queue.

Thread_start



```c
  /* my code begins
  create new threads */
  thread_create ("wakeup", PRI_MAX, wakeup, NULL);
  thread_create ("update", PRI_MAX, update, NULL);
  /* my code ends */
```

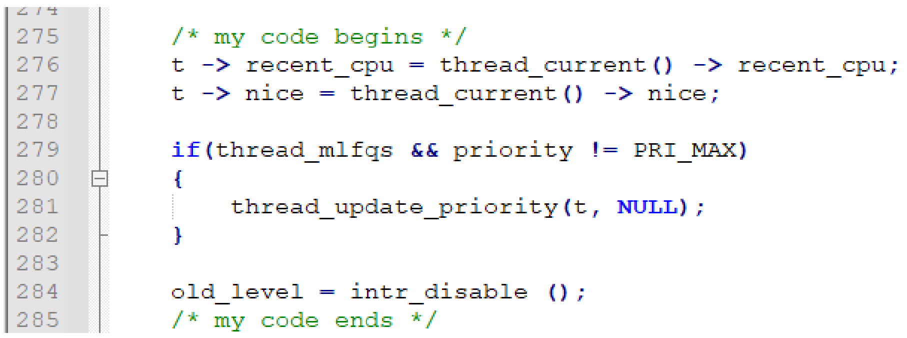
Thread_Tick



```
/* my code begins */

//unblock normal thread
if( (wake_thread -> status == THREAD_BLOCKED) && (!list_empty(&wait_list)) && (ticks >= next_wakeup) )
{
    thread_unblock(wake_thread);
    intr_yield_on_return ();
}

// update cpu
if(t != idle_thread)
{
    (t -> recent_cpu) += FP_F;
}

// check thread every tick using timer
if (ticks % TIMER_FREQ == 0)
{
    if(update_thread -> status == THREAD_BLOCKED)
    {
        thread_update_load_avg();
        thread_unblock(update_thread);
        intr_yield_on_return();
    }
}

// update prio every 4 ticks
if (ticks%4==0)
{
    thread_update_priority(t, NULL);
}

//addiion preempt mlfqs
if(t != wake_thread && t != update_thread)
{
    check_preempt(true);
    if (++thread_ticks >= TIME_SLICE)
    intr_yield_on_return ();
}

/* my code ends */
```

Unblock and wake, update CPU, update load average every tick, update priority every timeslice. Check priority position of normal threads.

Thread_create



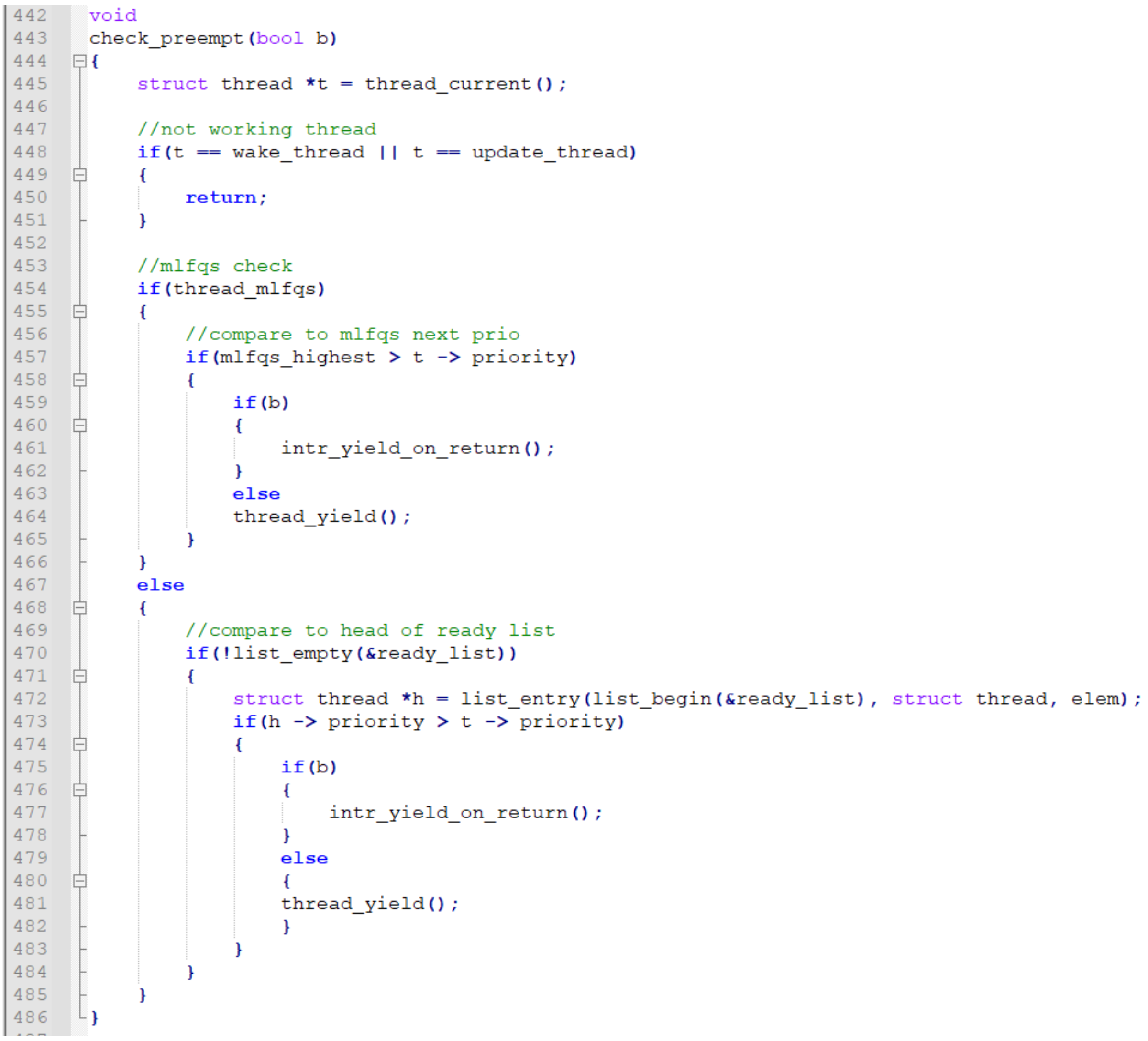
```
/* my code begins */
t -> recent_cpu = thread_current() -> recent_cpu;
t -> nice = thread_current() -> nice;

if(thread_mlfqs && priority != PRI_MAX)
{
    thread_update_priority(t, NULL);
}

old_level = intr_disable ();
/* my code ends */
```

Just make sure things are updated. Checks for new position in queue against other threads further down.

Thread_unblock

Added priority insert

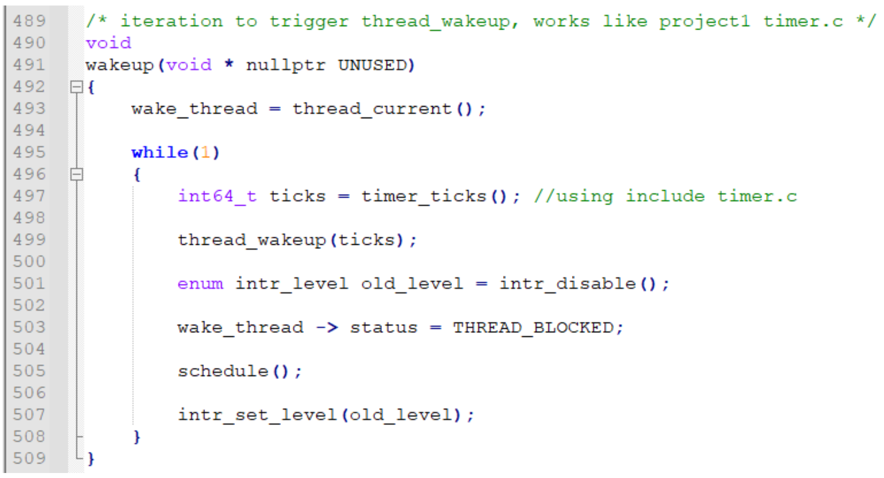
Check_preempt



```c
void
check_preempt(bool b)
{
    struct thread *t = thread_current();

    //not working thread
    if(t == wake_thread || t == update_thread)
    {
        return;
    }

    //mlfqs check
    if(thread_mlfqs)
    {
        //compare to mlfqs next prio
        if(mlfqs_highest > t -> priority)
        {
            if(b)
            {
                intr_yield_on_return();
            }
            else
            thread_yield();
        }
    }
    else
    {
        //compare to head of ready list
        if(!list_empty(&ready_list))
        {
            struct thread *h = list_entry(list_begin(&ready_list), struct thread, elem);
            if(h -> priority > t -> priority)
            {
                if(b)
                {
                    intr_yield_on_return();
                }
                else
                {
                thread_yield();
                }
            }
        }
    }
}
```

Checks normal thread and yields accordingly.

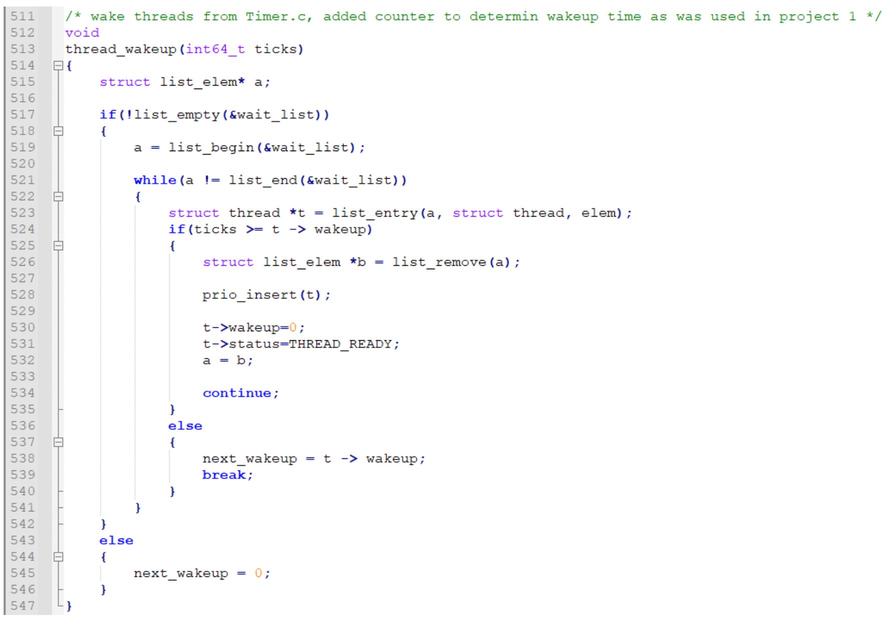
wakeup



```c
/* iteration to trigger thread_wakeup, works like project1 timer.c */
void
wakeup(void * nullptr UNUSED)
{
    wake_thread = thread_current();

    while(1)
    {
        int64_t ticks = timer_ticks(); //using include timer.c

        thread_wakeup(ticks);

        enum intr_level old_level = intr_disable();

        wake_thread -> status = THREAD_BLOCKED;

        schedule();

        intr_set_level(old_level);
    }
}
```

Similar to p1. Iterates and wakes up when needed

Thread_wakeup



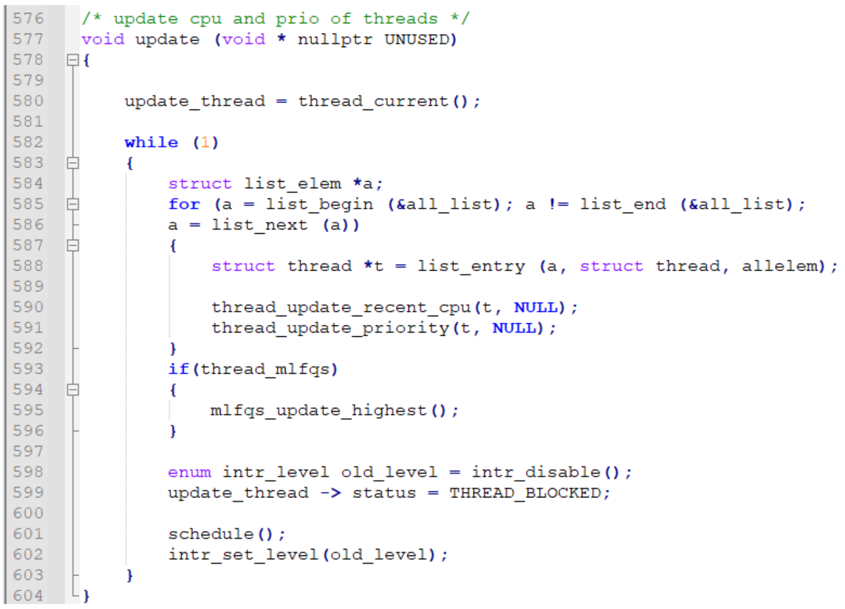
```c
/* wake threads from Timer.c, added counter to determin wakeup time as was used in project 1 */
void
thread_wakeup(int64_t ticks)
{
    struct list_elem* a;

    if(!list_empty(&wait_list))
    {
        a = list_begin(&wait_list);

        while(a != list_end(&wait_list))
        {
            struct thread *t = list_entry(a, struct thread, elem);
            if(ticks >= t -> wakeup)
            {
                struct list_elem *b = list_remove(a);

                prio_insert(t);

                t->wakeup=0;
                t->status=THREAD_READY;
                a = b;

                continue;
            }
            else
            {
                next_wakeup = t -> wakeup;
                break;
            }
        }
    }
    else
    {
        next_wakeup = 0;
    }
}
```

Similar to p1 again but main different is added ticks as argument which is counted in wakeup. Unlike timer.c which counted it as a static.

Thread_sleep

Same as p1

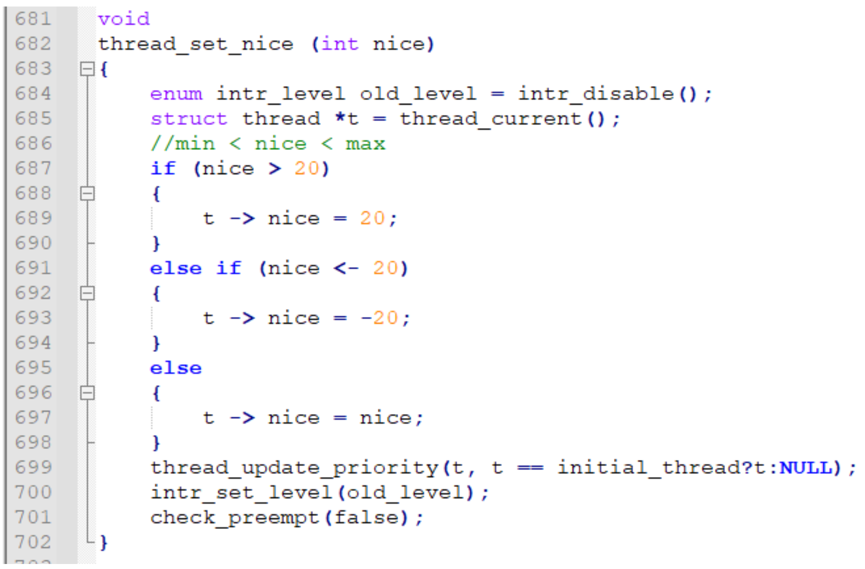
Update



```c
/* update cpu and prio of threads */
void update (void * nullptr UNUSED)
{

    update_thread = thread_current();

    while (1)
    {
        struct list_elem *a;
        for (a = list_begin (&all_list); a != list_end (&all_list);
        a = list_next (a))
        {
            struct thread *t = list_entry (a, struct thread, allelem);

            thread_update_recent_cpu(t, NULL);
            thread_update_priority(t, NULL);
        }
        if(thread_mlfqs)
        {
            mlfqs_update_highest();
        }

        enum intr_level old_level = intr_disable();
        update_thread -> status = THREAD_BLOCKED;

        schedule();
        intr_set_level(old_level);
    }
}
```

Uses update thread to update every thread prio and cpu. In case of mlfq updates the order and then blocks update thread.

Set_Nice Task 2.1



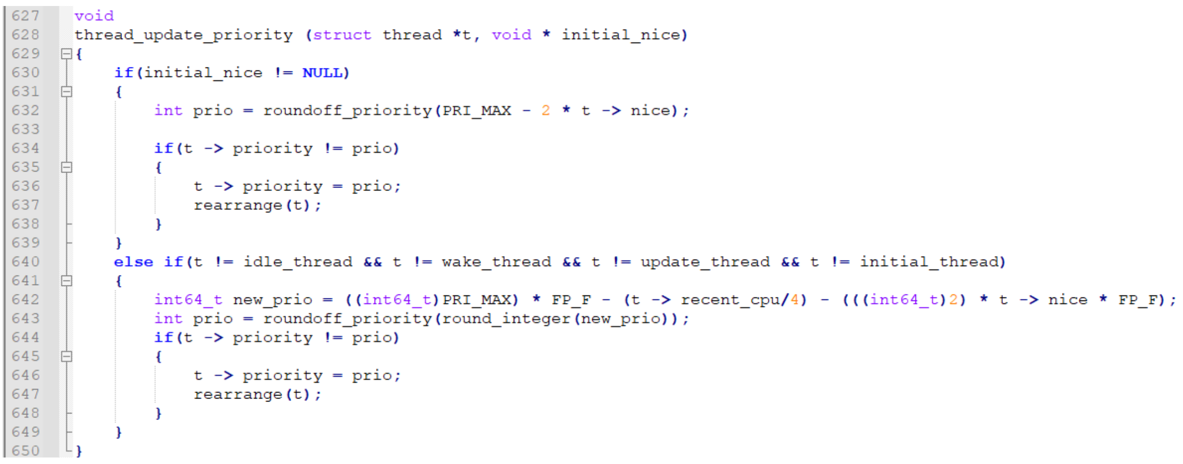
```
void
thread_set_nice (int nice)
{
    enum intr_level old_level = intr_disable();
    struct thread *t = thread_current();
    //min < nice < max
    if (nice > 20)
    {
        t -> nice = 20;
    }
    else if (nice <- 20)
    {
        t -> nice = -20;
    }
    else
    {
        t -> nice = nice;
    }
    thread_update_priority(t, t == initial_thread?t:NULL);
    intr_set_level(old_level);
    check_preempt(false);
}
```

Nice based on max min nice. Onces it sets new value updates priority and check_preempt

Thread_set_priority

Sets new priority and then calls check_preempt to update.

Thread_update_priority Task 2.2

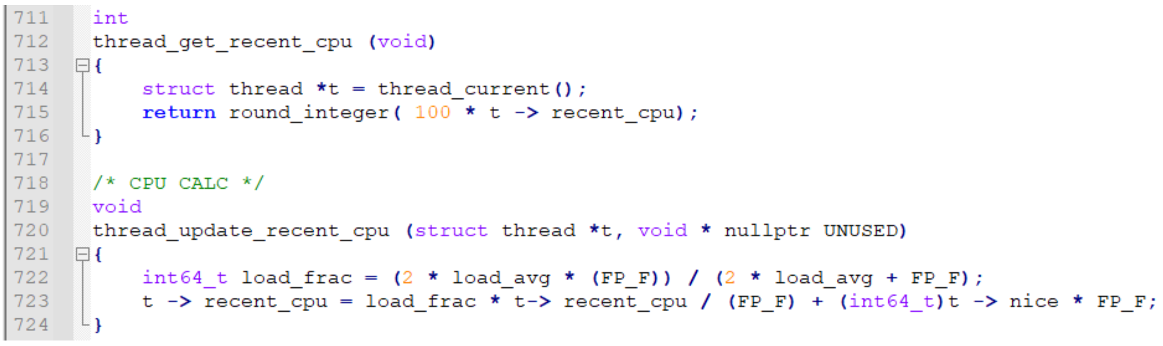


```
void
thread_update_priority (struct thread *t, void * initial_nice)
{
    if(initial_nice != NULL)
    {
        int prio = roundoff_priority(PRI_MAX - 2 * t -> nice);

        if(t -> priority != prio)
        {
            t -> priority = prio;
            rearrange(t);
        }
    }
    else if(t != idle_thread && t != wake_thread && t != update_thread && t != initial_thread)
    {
        int64_t new_prio = ((int64_t)PRI_MAX) * FP_F - (t -> recent_cpu/4) - (((int64_t)2) * t -> nice * FP_F);
        int prio = roundoff_priority(round_integer(new_prio));
        if(t -> priority != prio)
        {
            t -> priority = prio;
            rearrange(t);
        }
    }
}
```

New priority calc seen blow. Using Roundoff_priority to make sure it stays within the mlfqs list and rearrange to make sure order of list is correct.

$$priority = PRI\_MAX - \frac{recent\_cpu}{4} - (nice * 2)$$

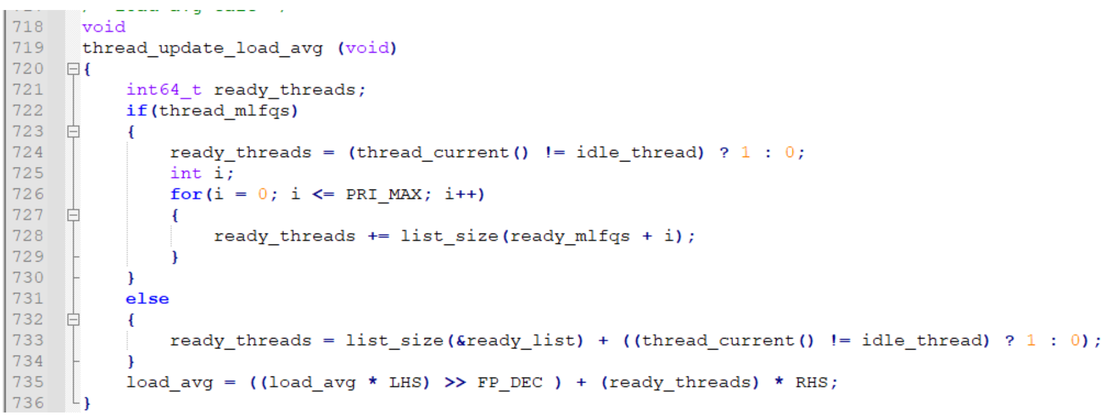
Thread_update_recent_cpu (Task 2.3)



```c
int
thread_get_recent_cpu (void)
{
    struct thread *t = thread_current();
    return round_integer( 100 * t -> recent_cpu);
}

/* CPU CALC */
void
thread_update_recent_cpu (struct thread *t, void * nullptr UNUSED)
{
    int64_t load_frac = (2 * load_avg * (FP_F)) / (2 * load_avg + FP_F);
    t -> recent_cpu = load_frac * t-> recent_cpu / (FP_F) + (int64_t)t -> nice * FP_F;
}
```

CPU calc and get value.

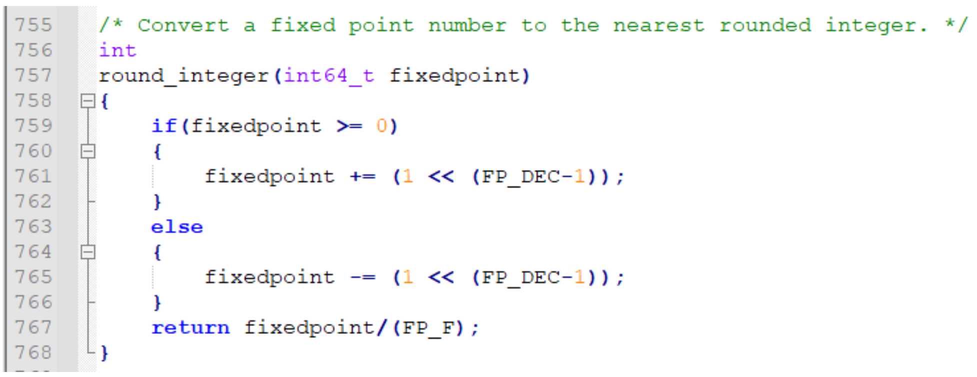
Thread_update_load_avg (Task 2.4)



```c
void
thread_update_load_avg (void)
{
    int64_t ready_threads;
    if(thread_mlfqs)
    {
        ready_threads = (thread_current() != idle_thread) ? 1 : 0;
        int i;
        for(i = 0; i <= PRI_MAX; i++)
        {
            ready_threads += list_size(ready_mlfqs + i);
        }
    }
    else
    {
        ready_threads = list_size(&ready_list) + ((thread_current() != idle_thread) ? 1 : 0);
    }
    load_avg = ((load_avg * LHS) >> FP_DEC ) + (ready_threads) * RHS;
}
```

New load average.

Round_Integer (CONVSERION)



```c
/* Convert a fixed point number to the nearest rounded integer. */
int
round_integer(int64_t fixedpoint)
{
    if(fixedpoint >= 0)
    {
        fixedpoint += (1 << (FP_DEC-1));
    }
    else
    {
        fixedpoint -= (1 << (FP_DEC-1));
    }
    return fixedpoint/(FP_F);
}
```

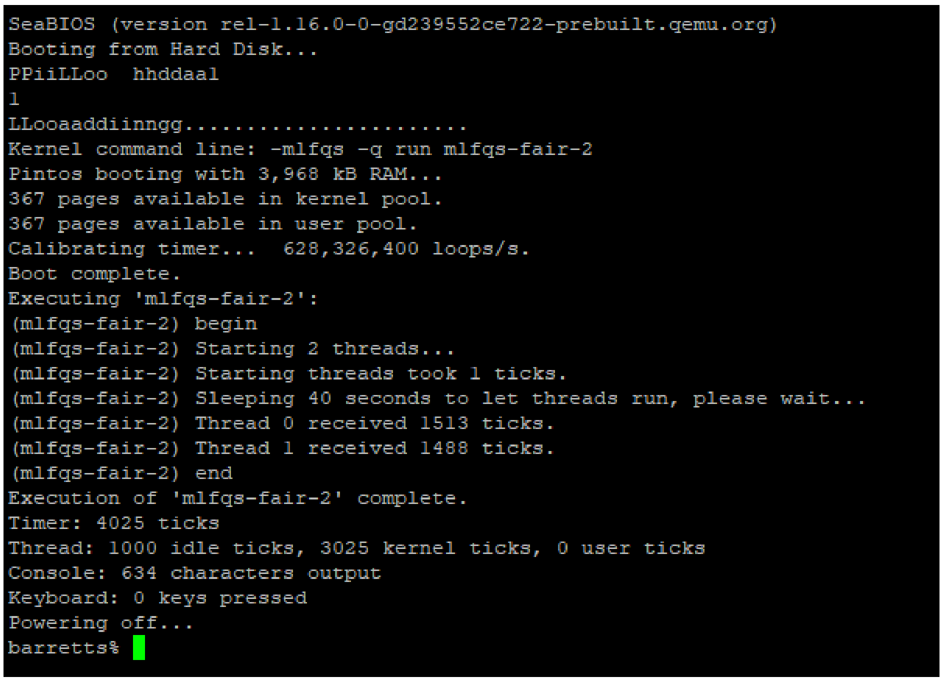
Converts fixed point to an integer.

Priority_higher and thread_compare

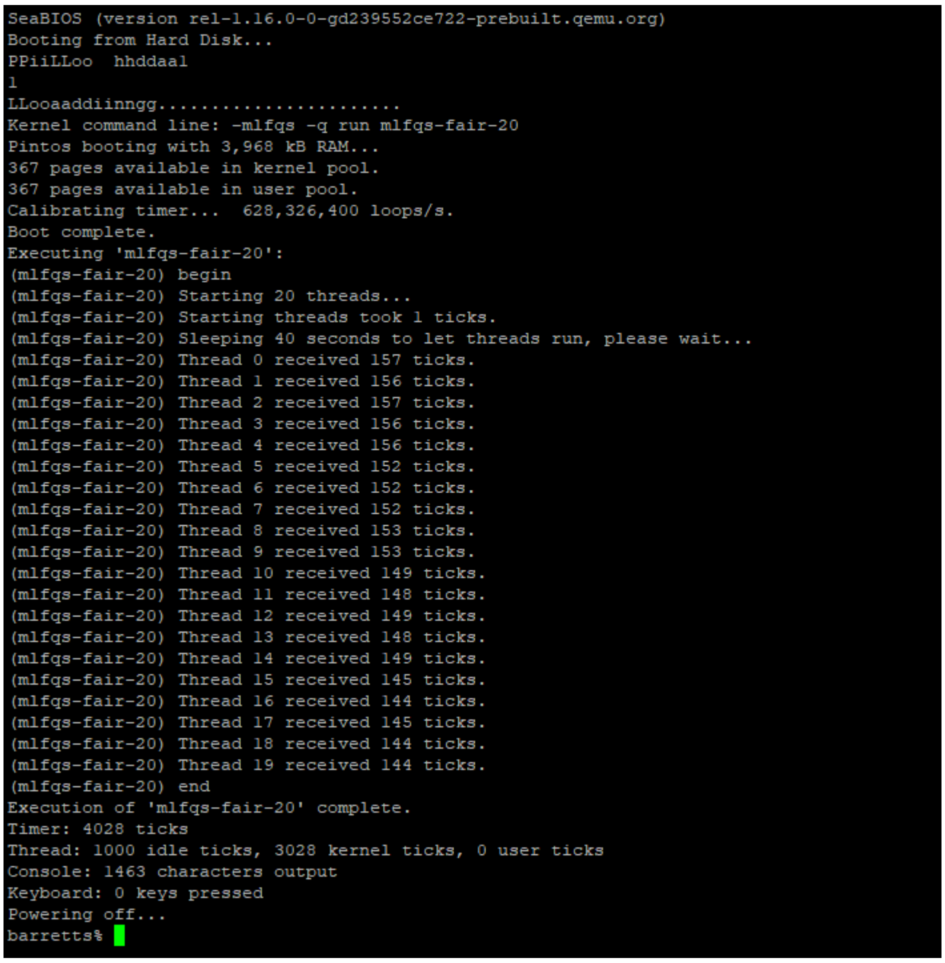
Comparisons of prio and wake time. Used in P1.

Testing

Mlfqs-fair-2

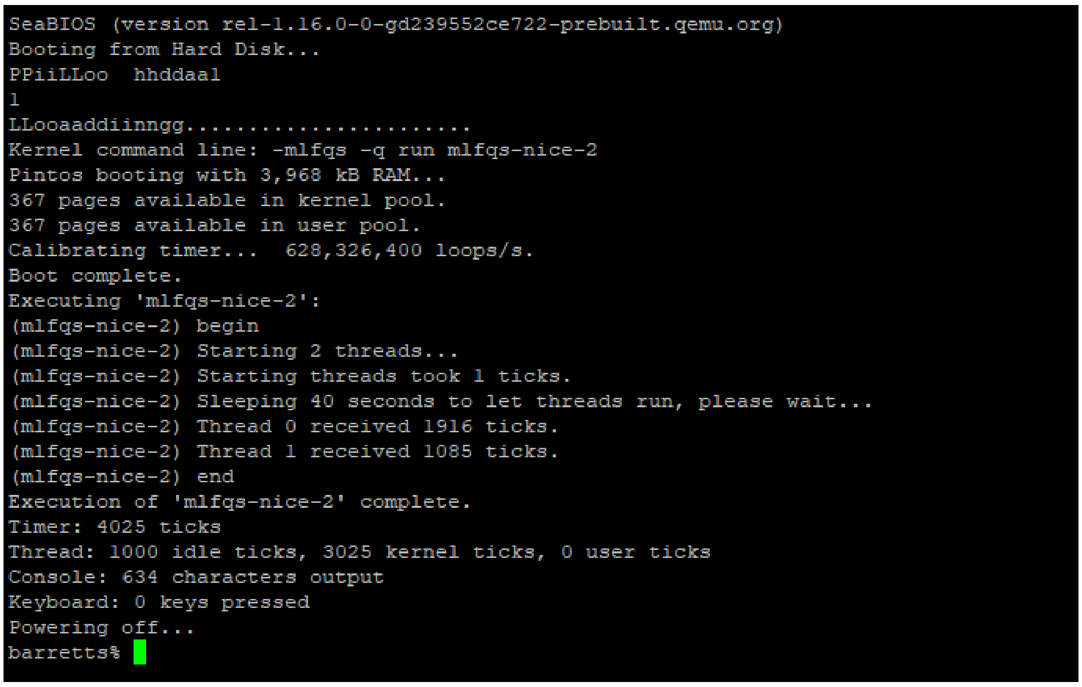

```
SeaBIOS (version rel-1.16.0-0-gd239552ce722-prebuilt.qemu.org)
Booting from Hard Disk...
PPiiLLoo  hhddaal
l
LLooaaddiinngg.......................
Kernel command line: -mlfqs -q run mlfqs-fair-2
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  628,326,400 loops/s.
Boot complete.
Executing 'mlfqs-fair-2':
(mlfqs-fair-2) begin
(mlfqs-fair-2) Starting 2 threads...
(mlfqs-fair-2) Starting threads took 1 ticks.
(mlfqs-fair-2) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-fair-2) Thread 0 received 1513 ticks.
(mlfqs-fair-2) Thread 1 received 1488 ticks.
(mlfqs-fair-2) end
Execution of 'mlfqs-fair-2' complete.
Timer: 4025 ticks
Thread: 1000 idle ticks, 3025 kernel ticks, 0 user ticks
Console: 634 characters output
Keyboard: 0 keys pressed
Powering off...
barretts%
```

Mlfqs-fair-20



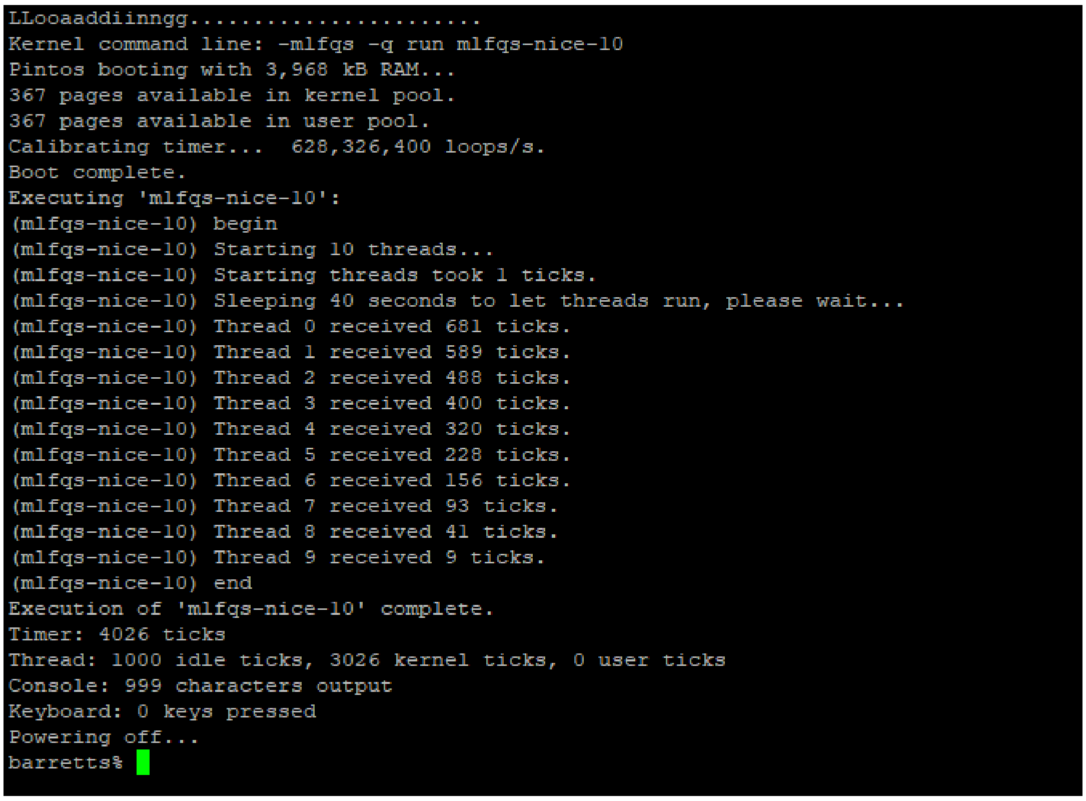
```
SeaBIOS (version rel-1.16.0-0-gd239552ce722-prebuilt.qemu.org)
Booting from Hard Disk...
PPiiLLoo  hhddaal
1
LLooaaddiinngg......................
Kernel command line: -mlfqs -q run mlfqs-fair-20
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  628,326,400 loops/s.
Boot complete.
Executing 'mlfqs-fair-20':
(mlfqs-fair-20) begin
(mlfqs-fair-20) Starting 20 threads...
(mlfqs-fair-20) Starting threads took 1 ticks.
(mlfqs-fair-20) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-fair-20) Thread 0 received 157 ticks.
(mlfqs-fair-20) Thread 1 received 156 ticks.
(mlfqs-fair-20) Thread 2 received 157 ticks.
(mlfqs-fair-20) Thread 3 received 156 ticks.
(mlfqs-fair-20) Thread 4 received 156 ticks.
(mlfqs-fair-20) Thread 5 received 152 ticks.
(mlfqs-fair-20) Thread 6 received 152 ticks.
(mlfqs-fair-20) Thread 7 received 152 ticks.
(mlfqs-fair-20) Thread 8 received 153 ticks.
(mlfqs-fair-20) Thread 9 received 153 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 149 ticks.
(mlfqs-fair-20) Thread 15 received 145 ticks.
(mlfqs-fair-20) Thread 16 received 144 ticks.
(mlfqs-fair-20) Thread 17 received 145 ticks.
(mlfqs-fair-20) Thread 18 received 144 ticks.
(mlfqs-fair-20) Thread 19 received 144 ticks.
(mlfqs-fair-20) end
Execution of 'mlfqs-fair-20' complete.
Timer: 4028 ticks
Thread: 1000 idle ticks, 3028 kernel ticks, 0 user ticks
Console: 1463 characters output
Keyboard: 0 keys pressed
Powering off...
barretts%
```

Mlfqs-nice-2



```
SeaBIOS (version rel-1.16.0-0-gd239552ce722-prebuilt.qemu.org)
Booting from Hard Disk...
PPiiLLoo  hhddaal
1
LLooaaddiinngg......................
Kernel command line: -mlfqs -q 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...  628,326,400 loops/s.
Boot complete.
Executing 'mlfqs-nice-2':
(mlfqs-nice-2) begin
(mlfqs-nice-2) Starting 2 threads...
(mlfqs-nice-2) Starting threads took 1 ticks.
(mlfqs-nice-2) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-nice-2) Thread 0 received 1916 ticks.
(mlfqs-nice-2) Thread 1 received 1085 ticks.
(mlfqs-nice-2) end
Execution of 'mlfqs-nice-2' complete.
Timer: 4025 ticks
Thread: 1000 idle ticks, 3025 kernel ticks, 0 user ticks
Console: 634 characters output
Keyboard: 0 keys pressed
Powering off...
barretts%
```

Mlfqs-nice-10



```
LLooaaddiinngg......................
Kernel command line: -mlfqs -q 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...  628,326,400 loops/s.
Boot complete.
Executing 'mlfqs-nice-10':
(mlfqs-nice-10) begin
(mlfqs-nice-10) Starting 10 threads...
(mlfqs-nice-10) Starting threads took 1 ticks.
(mlfqs-nice-10) Sleeping 40 seconds to let threads run, please wait...
(mlfqs-nice-10) Thread 0 received 681 ticks.
(mlfqs-nice-10) Thread 1 received 589 ticks.
(mlfqs-nice-10) Thread 2 received 488 ticks.
(mlfqs-nice-10) Thread 3 received 400 ticks.
(mlfqs-nice-10) Thread 4 received 320 ticks.
(mlfqs-nice-10) Thread 5 received 228 ticks.
(mlfqs-nice-10) Thread 6 received 156 ticks.
(mlfqs-nice-10) Thread 7 received 93 ticks.
(mlfqs-nice-10) Thread 8 received 41 ticks.
(mlfqs-nice-10) Thread 9 received 9 ticks.
(mlfqs-nice-10) end
Execution of 'mlfqs-nice-10' complete.
Timer: 4026 ticks
Thread: 1000 idle ticks, 3026 kernel ticks, 0 user ticks
Console: 999 characters output
Keyboard: 0 keys pressed
Powering off...
barretts%
```

Number seems slightly off but low enough to taken as rounding or conversion error.