정글 8기

[WEEK07-11] 정글끝까지

WEEK07 WIL

PROJECT 1: THREADS - Advanced Scheduler

TEAM7 김태훈-장유선-정재혁

CHAPTER 1

Advanced Scheduler
과제 설명

문제 및 해결

Priority Scheduler

- 우선순위가 낮은 스레드가 긴 시간동안 CPU를 점유하지 못하는 starvation 문제

multi-level feedback queue

- Multi-level
- Feedback

과제 목표

MLFQ(Multi-Level Feedback Queue) Scheduler 구현

Why mlfqs?

MULTI-LEVEL

Priority 에 따라 여러 개의
Ready Queue 가 존재한다.

FEEDBACK

Priority 는 실시간으로 조절한다.

CHAPTER 2

Advanced Scheduler KEYWORD

TOP KEYWORDS

#NICENESS

스레드의 상대적인 우선순위를 결정하며,
낮은 값은 더 높은 우선순위를 갖음

#PRIORITY

MLFQS에서 스레드의 Priority는 CPU 사용량과
실행 이력에 따라 동적으로 조정

#RECENT-CPU

스레드가 최근에 CPU를 얼마나 사용했는지를 나타냄
이를 기반으로 스레드의 우선순위가 조절됨

#LOAD-AVERAGE

최근 1분 동안 수행 가능한 스레드의 평균 개수를
추정하여 시스템의 현재 부하 상태를 나타내는 지표

KEYWORDS FORMULA

priority = PRI_MAX – (recent_cpu / 4) – (nice * 2)

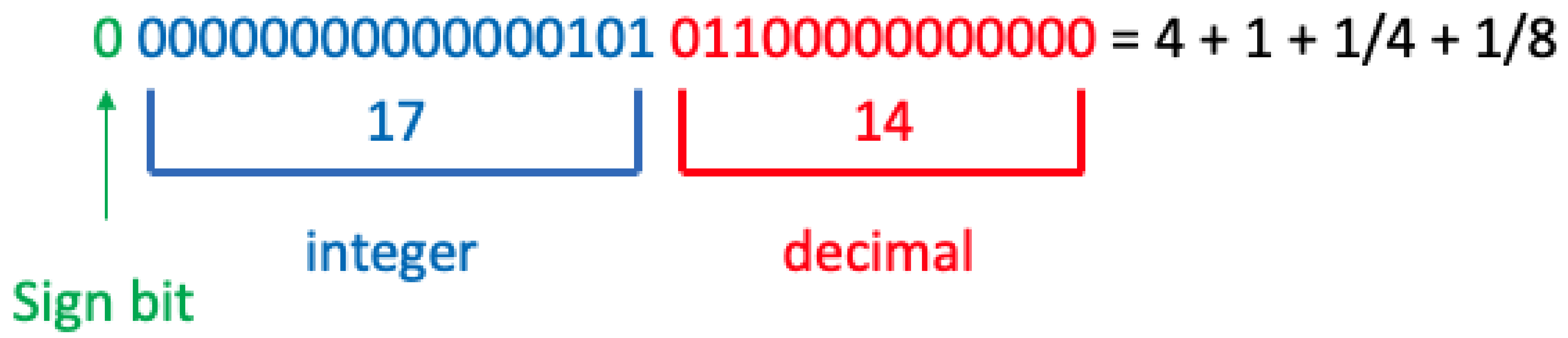
recent_cpu = (2 * load_avg) / (2 * load_avg + 1) * recent_cpu + nice

load_avg = (59/60) * load_avg + (1/60) * ready_threads

FIXED-POINT ARITHMETIC

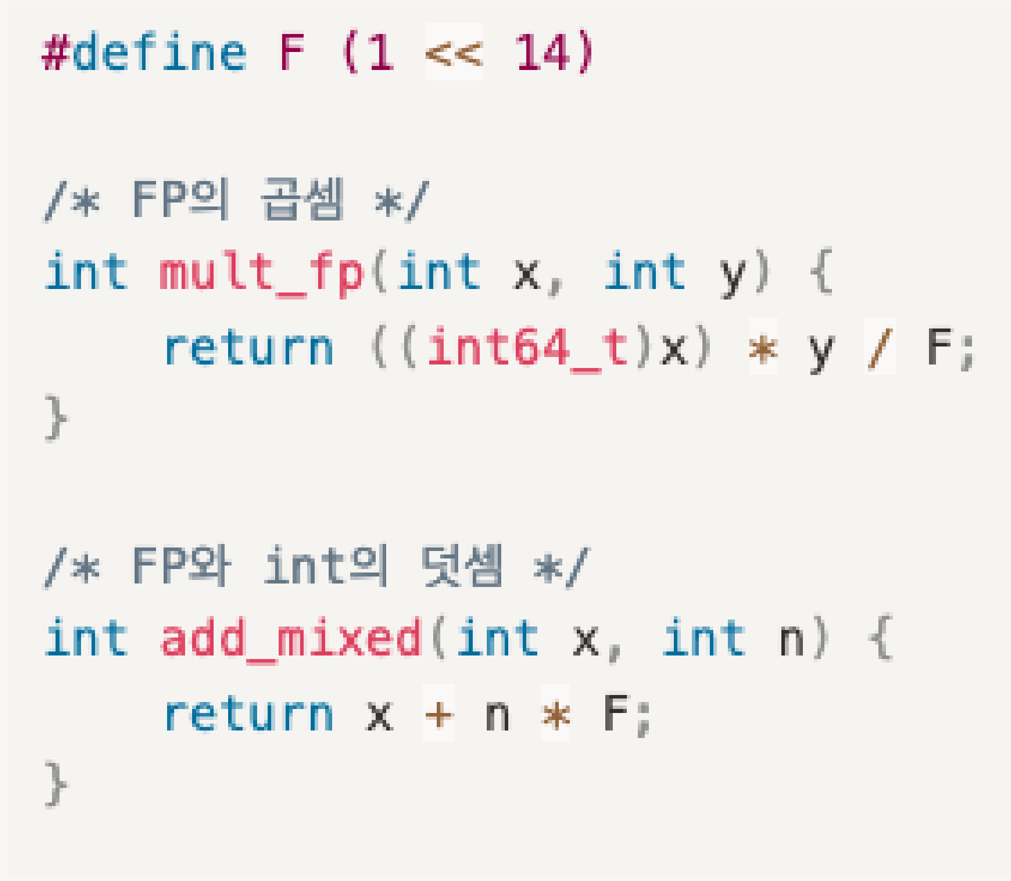
- nice value와 priority를 제외한 값들은 모두 실수 값
- 핀토스는 부동 소수점 연산을 지원하지 않음

→ 고정 소수점 연산(fixed-point arithmetic)을 활용



FIXED-POINT MACRO

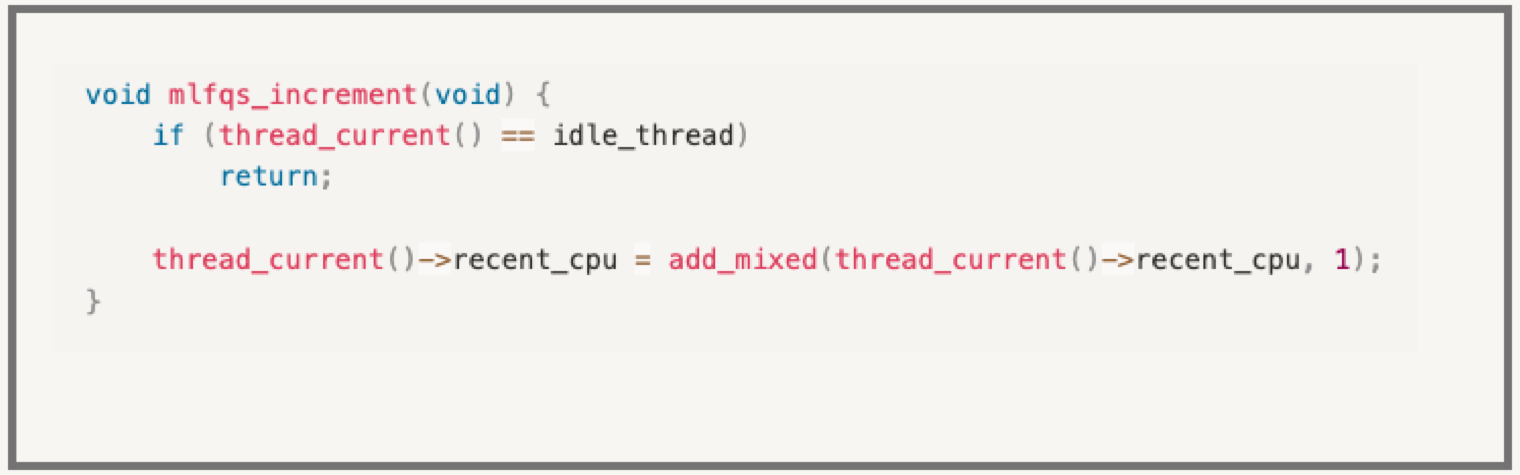
정수를 실수로 전환	n * f
실수를 정수로 전환 (버림)	x / f
실수를 정수로 전환 (반올림)	(x + f / 2) / f if x >= 0, (x - f / 2) / f if x <= 0.
실수의 덧셈	x + y
실수의 뺄셈	x - y
실수와 정수의 덧셈	x + n * f
실수와 정수의 뺄셈	x - n * f
실수의 곱셈	((int64_t) x) * y / f
실수와 정수의 곱셈	x * n
실수의 나눗셈	((int64_t) x) * f / y
실수와 정수의 나눗셈	x / n



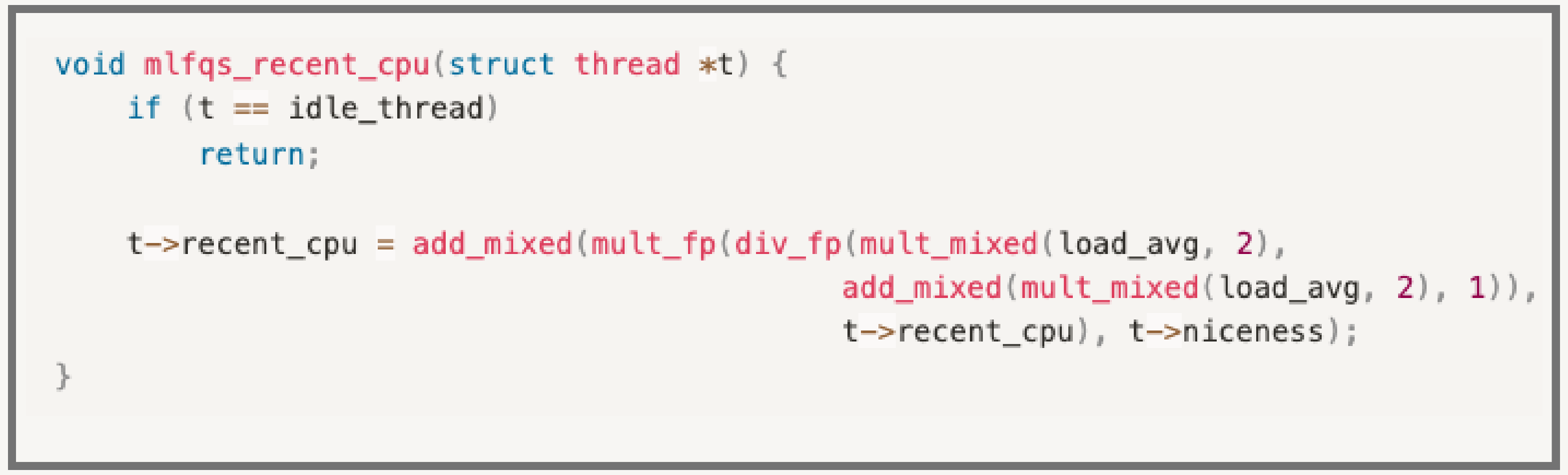
```
#define F (1 << 14)

/* FP의 곱셈 */
int mult_fp(int x, int y) {
    return ((int64_t)x) * y / F;
}

/* FP와 int의 덧셈 */
int add_mixed(int x, int n) {
    return x + n * F;
}
```

FEEDBACK 계산 함수

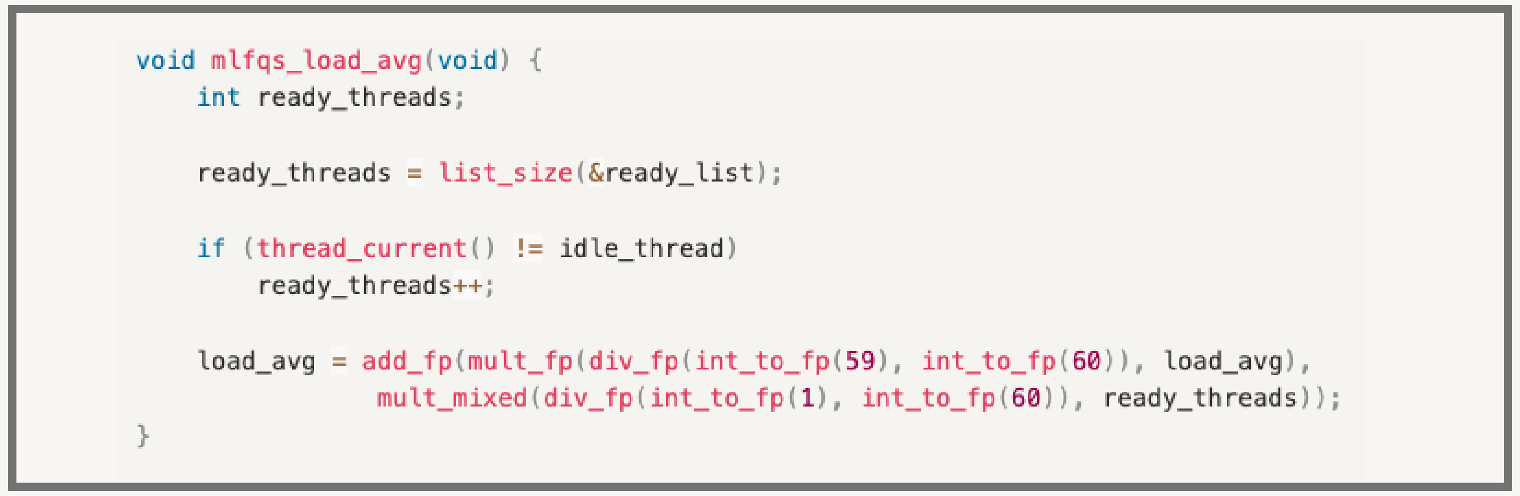
FIXED-POINT ARITHMETIC 적용



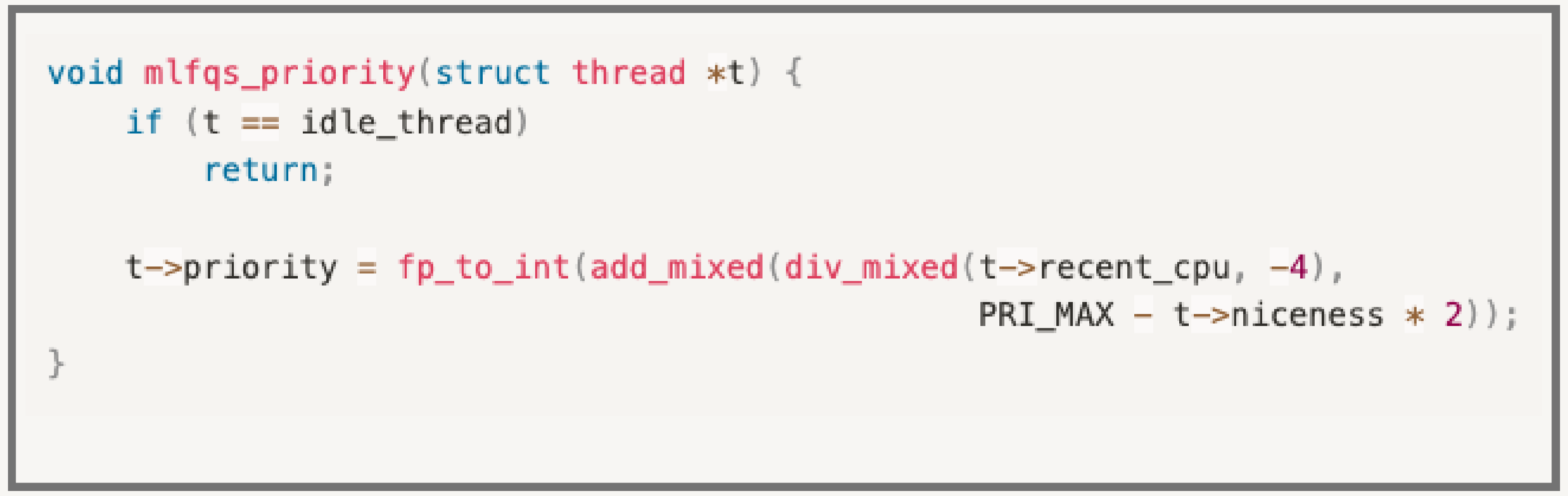
```c
void mlfqs_increment(void) {
    if (thread_current() == idle_thread)
        return;

    thread_current()->recent_cpu = add_mixed(thread_current()->recent_cpu, 1);
}
```



```c
void mlfqs_recent_cpu(struct thread *t) {
    if (t == idle_thread)
        return;

    t->recent_cpu = add_mixed(mult_fp(div_fp(mult_mixed(load_avg, 2),
                                             add_mixed(mult_mixed(load_avg, 2), 1)),
                                             t->recent_cpu), t->niceness);
}
```



```c
void mlfqs_load_avg(void) {
    int ready_threads;

    ready_threads = list_size(&ready_list);

    if (thread_current() != idle_thread)
        ready_threads++;

    load_avg = add_fp(mult_fp(div_fp(int_to_fp(59), int_to_fp(60)), load_avg),
                mult_mixed(div_fp(int_to_fp(1), int_to_fp(60)), ready_threads));
}
```



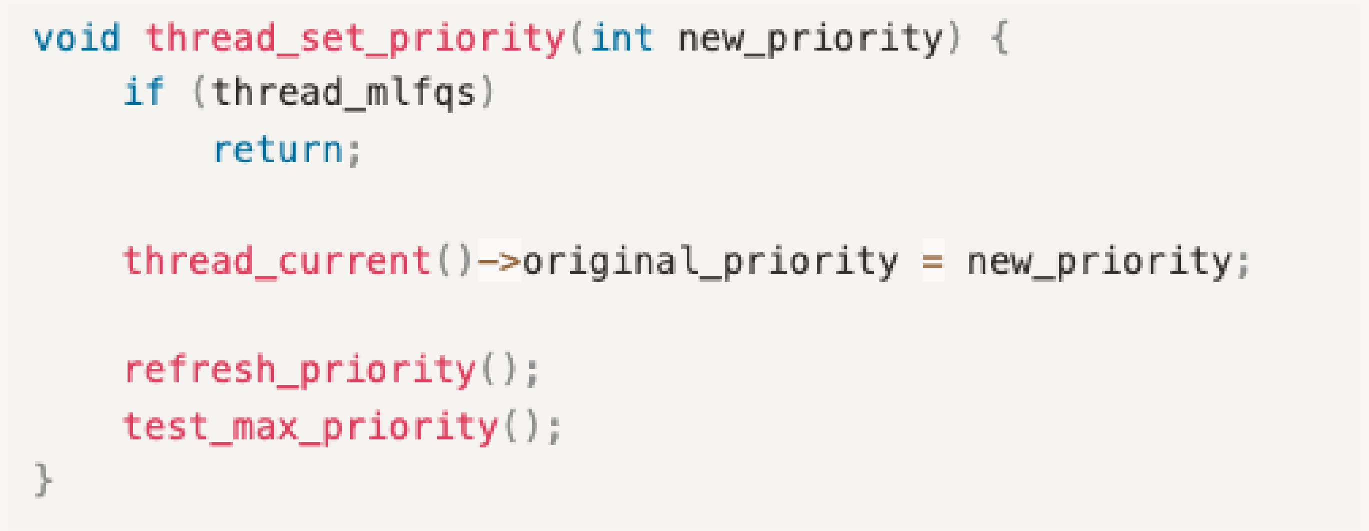
```c
void mlfqs_priority(struct thread *t) {
    if (t == idle_thread)
        return;

    t->priority = fp_to_int(add_mixed(div_mixed(t->recent_cpu, -4),
                                      PRI_MAX - t->niceness * 2));
}
```

CHAPTER 3

구현

Thread Set Priority

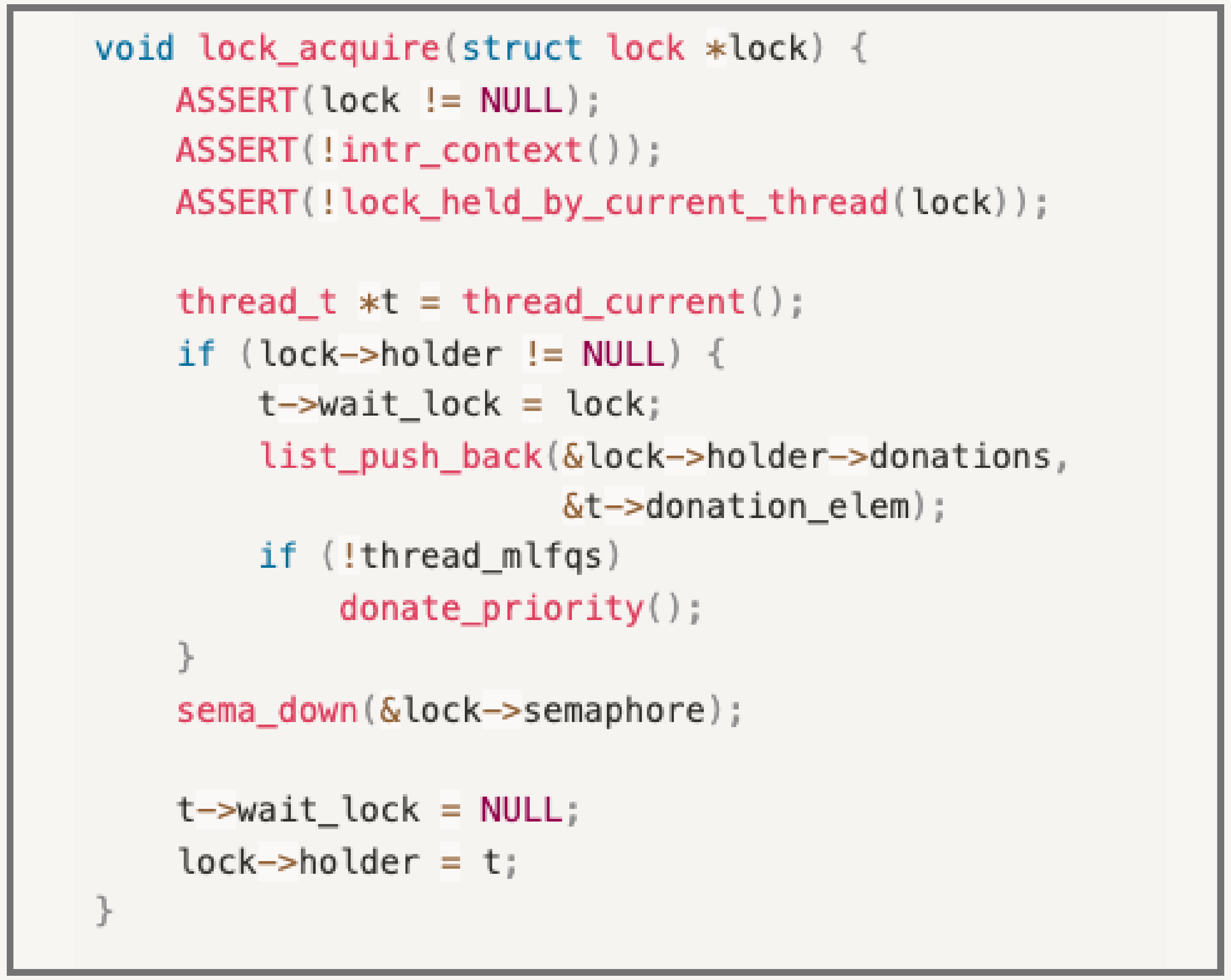
mlfqs 일 때 우선순위를 임의로 변경할 수 없도록 수정



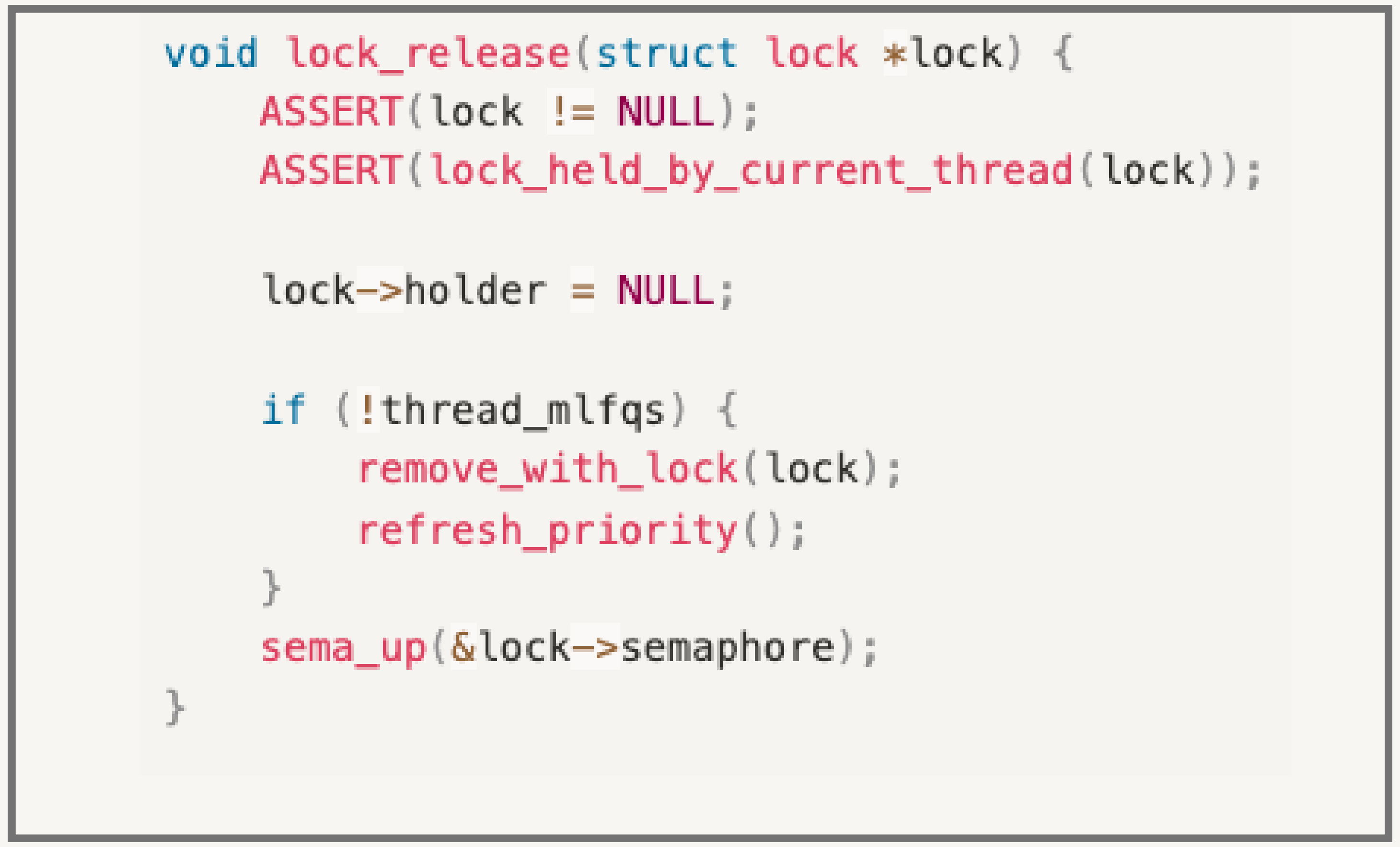
```c
void thread_set_priority(int new_priority) {
    if (thread_mlfqs)
        return;

    thread_current()->original_priority = new_priority;

    refresh_priority();
    test_max_priority();
}
```

Lock Acquire & Lock Release

mlfqs 스케줄러 사용시 donation 사용 금지

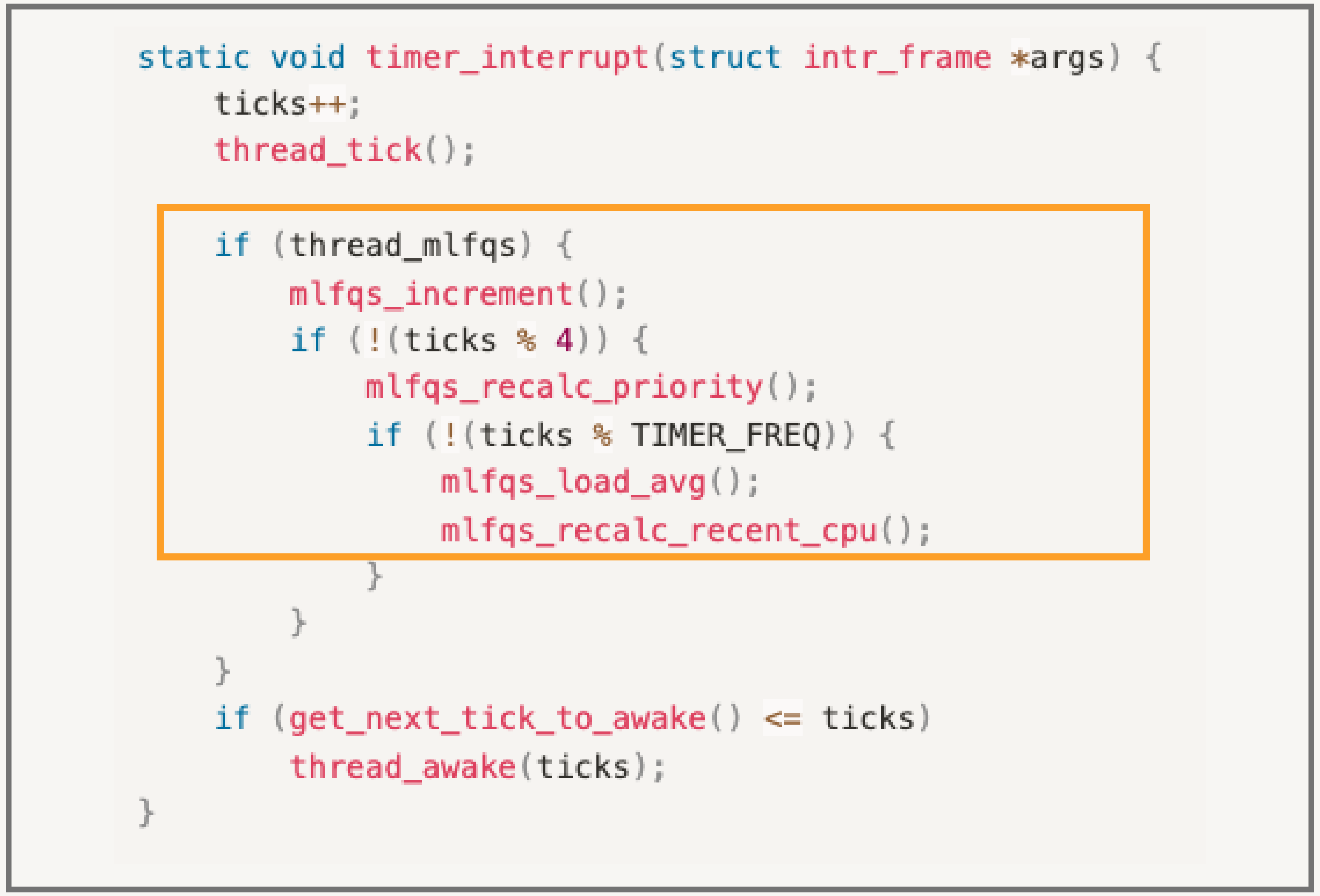


```
void lock_acquire(struct lock *lock) {
    ASSERT(lock != NULL);
    ASSERT(!intr_context());
    ASSERT(!lock_held_by_current_thread(lock));

    thread_t *t = thread_current();
    if (lock->holder != NULL) {
        t->wait_lock = lock;
        list_push_back(&lock->holder->donations,
                       &t->donation_elem);
        if (!thread_mlfqs)
            donate_priority();
    }
    sema_down(&lock->semaphore);

    t->wait_lock = NULL;
    lock->holder = t;
}
```



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

    lock->holder = NULL;

    if (!thread_mlfqs) {
        remove_with_lock(lock);
        refresh_priority();
    }
    sema_up(&lock->semaphore);
}
```

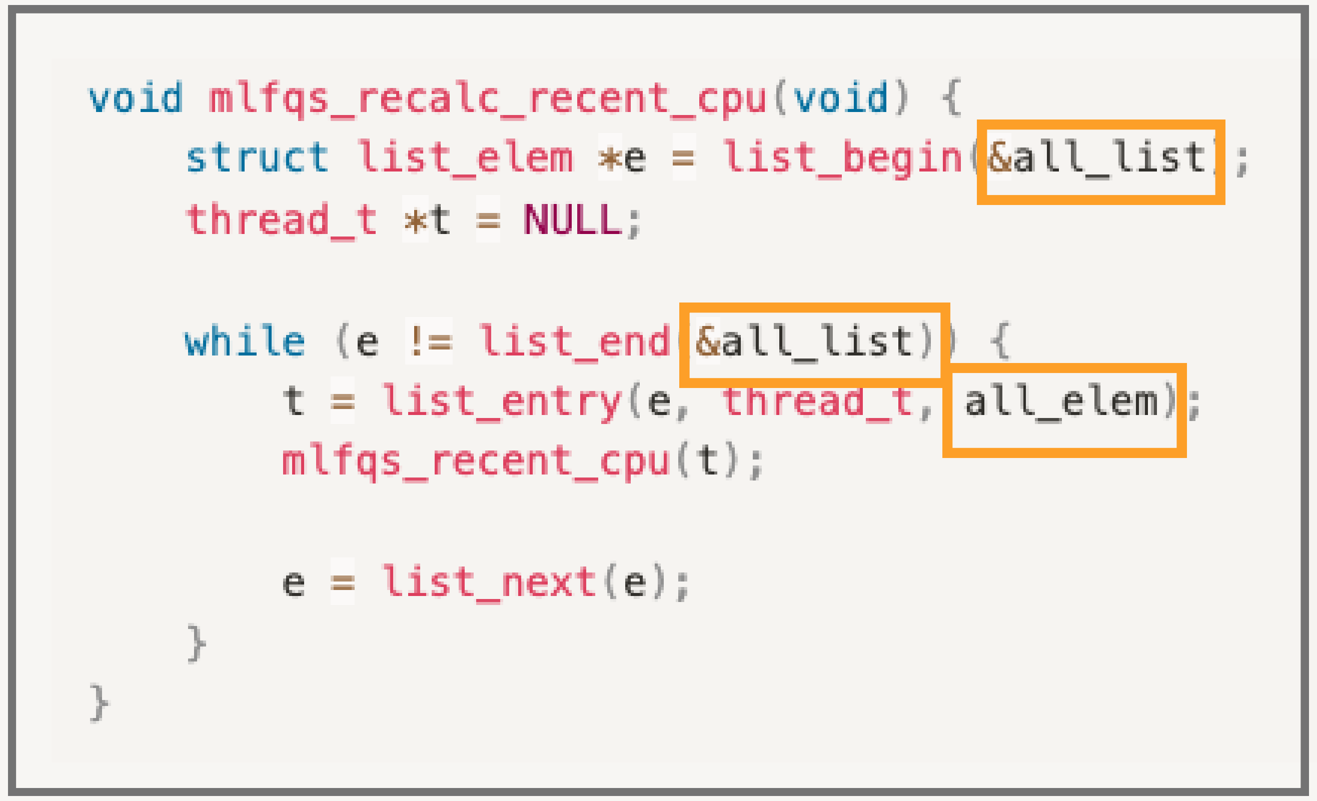
Timer Interrupt



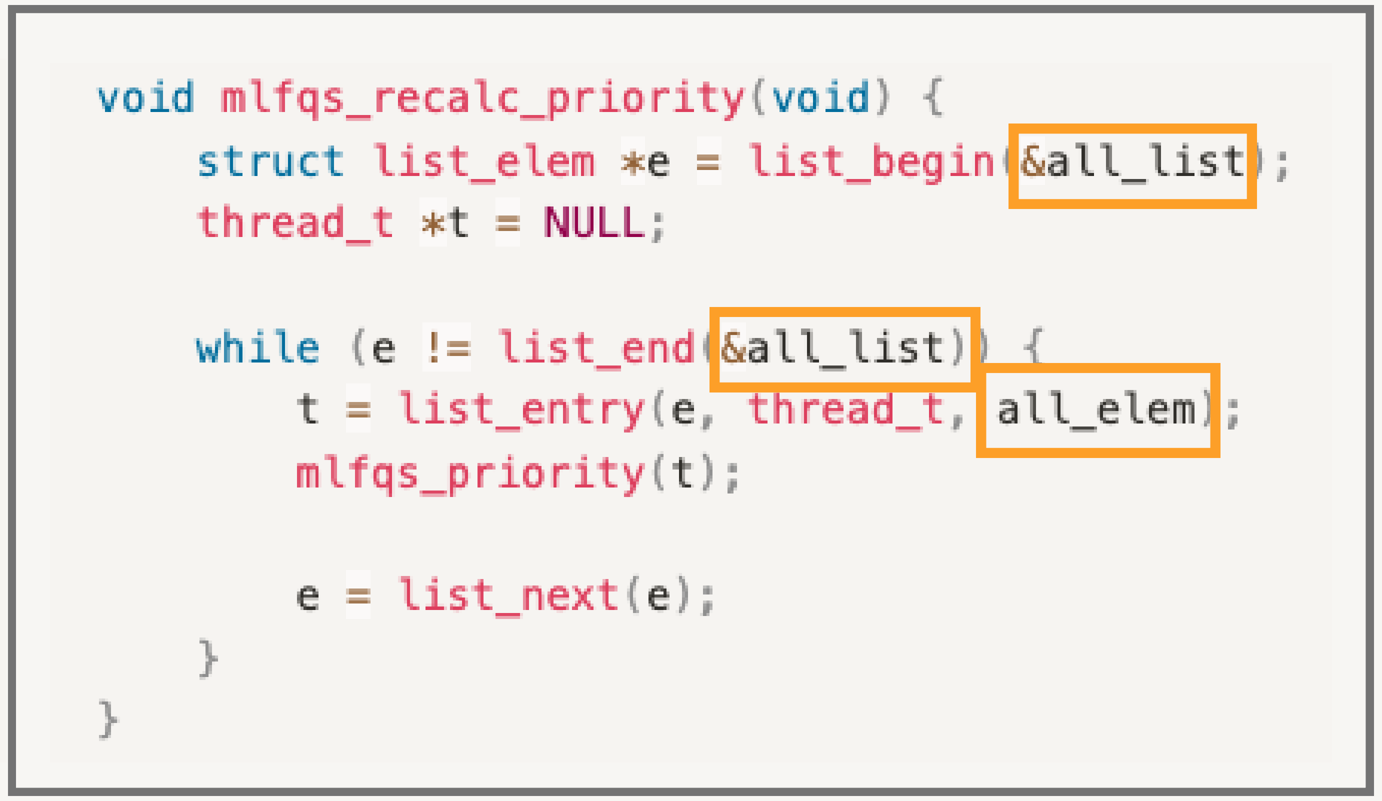
```c
static void timer_interrupt(struct intr_frame *args) {
    ticks++;
    thread_tick();

    if (thread_mlfqs) {
        mlfqs_increment();
        if (!(ticks % 4)) {
            mlfqs_recalc_priority();
            if (!(ticks % TIMER_FREQ)) {
                mlfqs_load_avg();
                mlfqs_recalc_recent_cpu();
            }
        }
    }
    if (get_next_tick_to_awake() <= ticks)
        thread_awake(ticks);
}
```

Recalculate Recent CPU & Priority

모든 priority와 recent CPU 재계산하는 함수

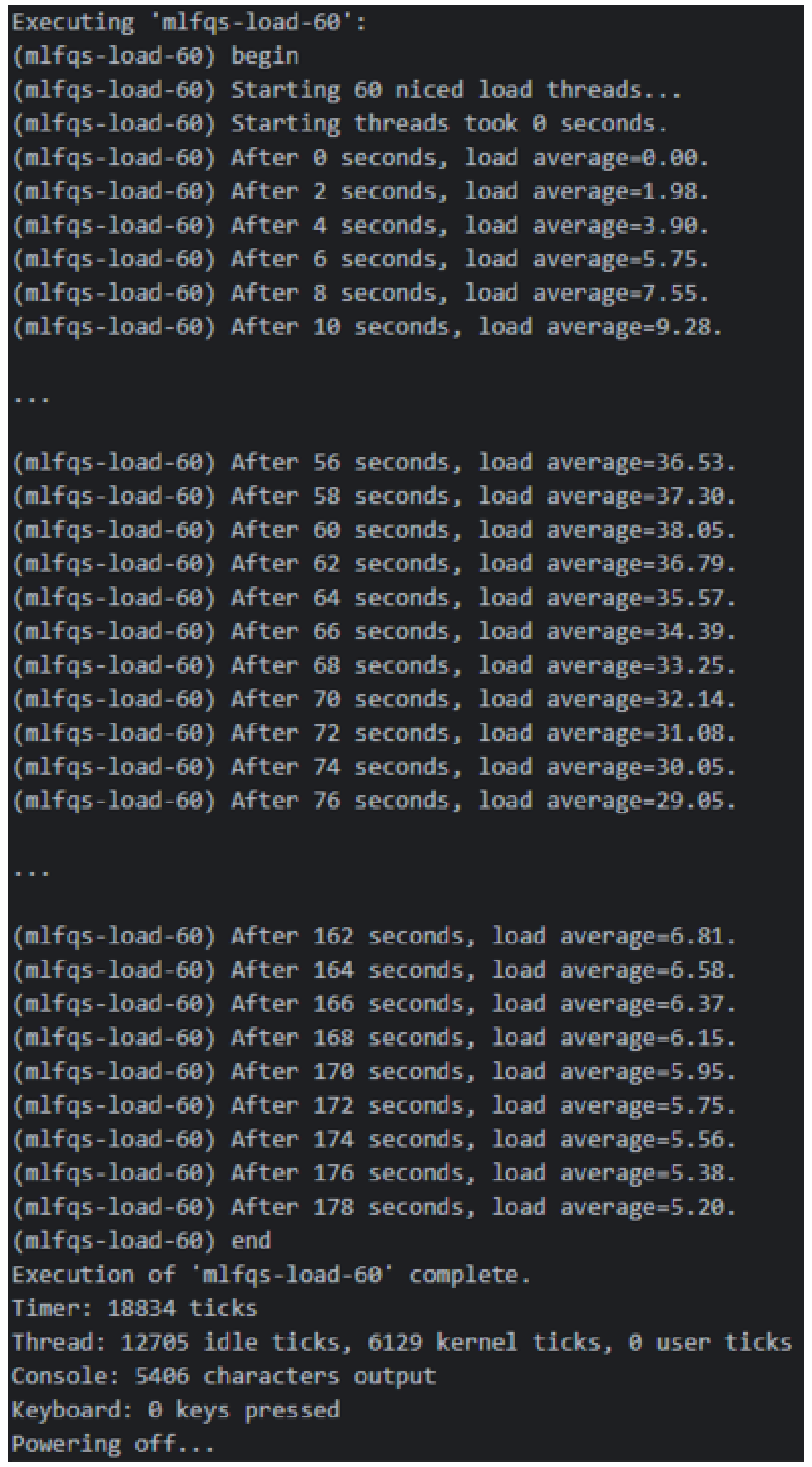


```c
void mlfqs_recalc_recent_cpu(void) {
    struct list_elem *e = list_begin(&all_list);
    thread_t *t = NULL;

    while (e != list_end(&all_list)) {
        t = list_entry(e, thread_t, all_elem);
        mlfqs_recent_cpu(t);

        e = list_next(e);
    }
}
```

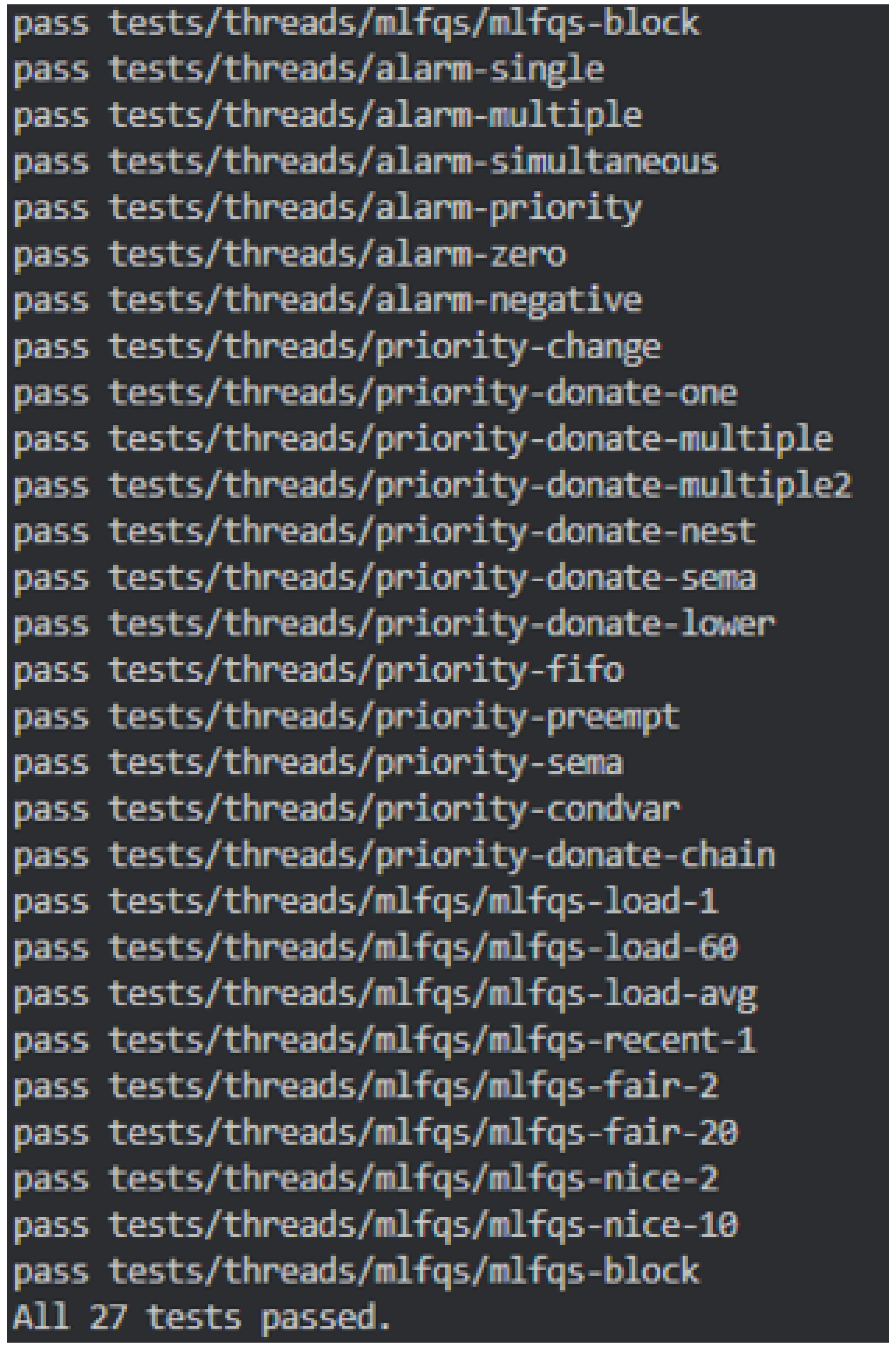


```c
void mlfqs_recalc_priority(void) {
    struct list_elem *e = list_begin(&all_list);
    thread_t *t = NULL;

    while (e != list_end(&all_list)) {
        t = list_entry(e, thread_t, all_elem);
        mlfqs_priority(t);

        e = list_next(e);
    }
}
```

결과



```
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 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 72 seconds, load average=31.08.
(mlfqs-load-60) After 74 seconds, load average=30.05.
(mlfqs-load-60) After 76 seconds, load average=29.05.

...

(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: 18834 ticks
Thread: 12705 idle ticks, 6129 kernel ticks, 0 user ticks
Console: 5406 characters output
Keyboard: 0 keys pressed
Powering off...
```



```
pass tests/threads/mlfqs/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/mlfqs-load-1
pass tests/threads/mlfqs/mlfqs-load-60
pass tests/threads/mlfqs/mlfqs-load-avg
pass tests/threads/mlfqs/mlfqs-recent-1
pass tests/threads/mlfqs/mlfqs-fair-2
pass tests/threads/mlfqs/mlfqs-fair-20
pass tests/threads/mlfqs/mlfqs-nice-2
pass tests/threads/mlfqs/mlfqs-nice-10
pass tests/threads/mlfqs/mlfqs-block
All 27 tests passed.
```

정글 8기

[WEEK07-11] 정글끝까지

감사합니다

TEAM7 김태훈-장유선-정재혁