

운영체제

1분반

Pintos Project - Alarm clock의 개선

이름: 한규현

학년: 4

학번: 201810896

학과: 융합전자공학전공

1. 프로젝트 개요

pintos 커널의 timer_sleep() 함수를 개선한다.

2. 프로젝트 내용 - Alarm clock의 개선

2.1 문제 정의

아래는 'devices/timer.c'에 정의되어 있는 timer_sleep() 함수이다.

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

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

현재 timer_sleep()은 busy waiting 방식으로 구현되어 있다. 'while'문이 돌면서 timer_elapsed(start)가 ticks보다 작을 때마다 thread_yield()를 호출한다. (thread_yield()는 현재 실행 중인 스레드가 다른 스레드에게 실행을 양보하도록 하는 함수이다.) 이런 방식은 스레드가 실행 가능한 시간을 낭비하며 시스템 전체의 비효율을 초래하는 알고리즘이다.

2.2 문제 해결

문제를 해결하기 위해 timer_sleep()을 호출한 스레드가 현재 시각 기준으로 ticks 시간이 경과할 때까지 block 시키는 방식으로 수정한다. block된 스레드가 대기할 수 있는 리스트를 만들고, 매번 timer interrupt가 발생할 때마다 timer tick이 ticks 이상 지났는지 검사하는 방식으로 구현한다.

2.3 파일 수정 목록

아래는 수정한 파일에 대한 위치와 이름, 수정내용을 정리한 표이다.

수정 파일	수정내용
threads/thread.h	thread_sleep() 함수 프로토타입 선언 추가
	thread_awake() 함수 프로토타입 선언 추가
threads/thread.c	alarms 더블 링크 리스트 추가 및 초기화를 위한 thread_init 수정
	alarm 구조체 추가
	thread_sleep() 함수 추가
	thread_awake() 함수 추가
devices/timer.c	timer_sleep() 함수 수정
	timer_interrupt() 함수 수정

2.4 자료구조

threads/thread.c에 다음과 같은 코드를 추가하였다.

```
static struct list alarms;
```

```
thread_init (void)
{
    //...생략
    list_init (&alarms);
    //...생략
};
```

핀토스에서는 스레드를 관리하는 리스트는 ready list와 all list 두 개만 존재한다. 따라서 threads.c에 스레드들이 요청한 알람을 유지하는 linked list를 선언하고, thread_init에서 선언된 alarms 리스트를 초기화하는 코드를 추가했다. alarms 리스트를 통해 알람 thread를 관리한다.

```
struct alarm
{
    int64_t expiration;
    struct thread * th;
    struct list_elem elem;
};
```

알람을 위한 구조체를 선언하였다. expiration은 알람 만료 시간, *th는 알람을 요청한 스레드, elem은 알람 리스트 연결 필드에 대한 정보를 가진다.

2.5 수정 코드 설명

2.5.1 thread_sleep() - threads/thread.c

```
void
thread_sleep (int64_t ticks)
{
    struct thread *cur;
    enum intr_level old_level;

    old_level = intr_disable();
    cur = thread_current();

    ASSERT(cur != idle_thread);

    struct alarm a;
    a.expiration = ticks;
    a.th = cur;

    list_push_back(&alarms, &a.elem);

    thread_block();

    intr_set_level(old_level);
}
```

thread_sleep() 함수에 주요 기능은 sleep 해야 하는 thread를 alarms 리스트에 추가하고 해당 thread를 thread_block() 호출을 통해 block 상태로 만든다.

‘*cur’라는 thread 구조체를 선언하고 interrupt가 off 상태에서 ‘cur’ 변수에는 현재 thread에 대한 정보를 담도록 했다. idle thread의 경우 sleep에 빠지면 안되기 때문에 ASSERT(cur != idle_thread); 코드를 작성해 block을 방지했다.

alarm 구조체 ‘a’를 선언해 ‘a.expiration’에 알람 만료 시간을 담고, ‘a.th’에 현재 스레드가 알람을 요청했다는 정보를 담는다. 이후 list_push_back()을 사용해 alarms 리스트에 해당 정보를 추가하고 thread_block()을 통해 thread를 block 상태로 만든다. 이 과정에서 thread는 running -> waiting으로 전환되고, interrupt는 on 상태가 되게 된다.

2.5.2 timer_sleep() - devices/timer.c

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

    ASSERT (intr_get_level () == INTR_ON);
    thread_sleep (start + ticks);
}
```

timer_sleep() 함수에 주요 기능은 timer_sleep()을 호출한 thread의 수행 호출 시점부터 ticks 값 만큼 지연하게 하는 역할이다.

‘start’ 변수에 timer_ticks();을 사용해 현재까지 경과한 ticks을 반환 받고 interrupt가 on 상태인지 확인 후, thread_sleep() 함수를 호출해 현재 시점부터 ticks 까지 block 상태가 된다.

2.5.3 thread_awake() - threads/thread.c

```
void
thread_awake(int64_t ticks)
{
    struct alarm *a;
    struct list_elem *e = list_begin(&alarms);

    while (e != list_end(&alarms))
    {
        a = list_entry(e, struct alarm, elem);
        if (a->expiration <= ticks)
        {
            e = list_remove(e);
            thread_unblock(a->th);
        }
        else
        {
            e = list_next(e);
        }
    }
}
```

thread_awake()는 alarms 리스트를 돌면서 모든 알람을 검사하고 block에서 해제되어야 하는 thread를 찾아 ready list 옮기고, thread 상태를 ready로 변경하는 역할을 한다.

e에는 alarms 리스트의 시작 지점을 대입하고, while 문에서 끝까지 도달하도록 반복한다. while이 돌면서 ‘a’에는 현재 alarm의 정보를 가르키게 된다. 만약 ‘a->expiration’이 ‘ticks’

보다 작으면 thread가 block을 해제해도 되는 thread 이므로 list에서 제거하고, thread_unblock(); 호출을 통해 thread를 깨운다.

알람 만료가 된 것이 없는 경우 다음 알람을 검사할 수 있도록 동작한다.

thread_sleep()과 thread_awake()의 경우 추가로 작성한 함수이므로, threads/thread.h에 아래와 같이 프로토타입을 선언했다.

```
void thread_sleep(int64_t ticks);  
void thread_awake(int64_t ticks);
```

2.5.4 timer_interrupt() - devices/timer.c

```
static void  
timer_interrupt (struct intr_frame *args UNUSED)  
{  
    ticks++;  
    thread_tick ();  
    thread_awake(ticks);  
}
```

timer_interrupt()에 기능은 매 tick이 발생할 때마다, interrupt 처리 과정에서 만료된 알람을 검사하고, 만료된 알람이 있는 경우 알람 리스트에서 제거하고 해당 thread를 thread_unblock()을 호출해서 thread를 깨운다.

기존에 코드에서 thread_awake() 함수를 호출함으로써 ticks가 증가할 때마다 깨워야 할 thread 여부 확인 및 unblock을 할 수 있다.

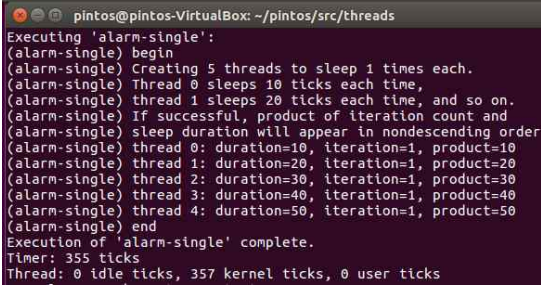
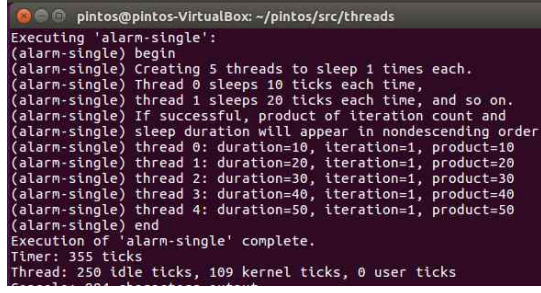
2.6 테스트

2.6.1 make check

```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
FAIL tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
```

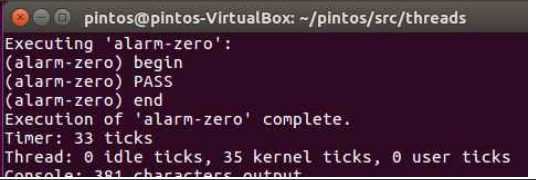
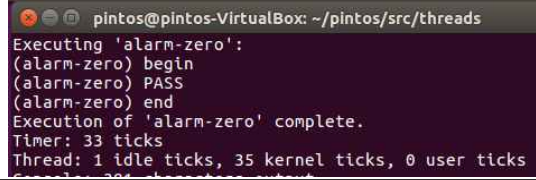
alarm-priority를 제외한 5개 프로그램 모두 테스트 수행 결과가 pass인 것을 확인할 수 있다.

2.6.2 alarm-single

alarm-single	
개선 전(busy-waiting)	개선 후(no-busy-waiting)
 <pre>pintos@pintos-VirtualBox: ~/pintos/src/threads 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: 355 ticks Thread: 0 idle ticks, 357 kernel ticks, 0 user ticks</pre>	 <pre>pintos@pintos-VirtualBox: ~/pintos/src/threads 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: 355 ticks Thread: 250 idle ticks, 109 kernel ticks, 0 user ticks</pre>

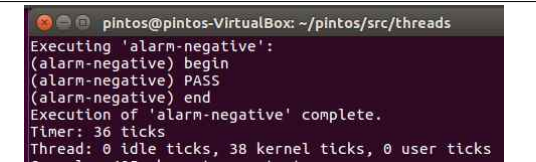
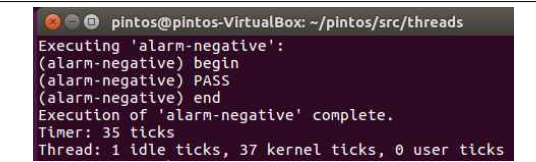
idle tick의 값이 0 -> 250으로 증가하였다.

2.6.5 alarm-zero

alarm-zero	
개선 전(busy-waiting)	개선 후(no-busy-waiting)
 <pre> pintos@pintos-VirtualBox: ~/pintos/src/threads Executing 'alarm-zero': (alarm-zero) begin (alarm-zero) PASS (alarm-zero) end Execution of 'alarm-zero' complete. Timer: 33 ticks Thread: 0 idle ticks, 35 kernel ticks, 0 user ticks Console: 381 characters output </pre>	 <pre> pintos@pintos-VirtualBox: ~/pintos/src/threads Executing 'alarm-zero': (alarm-zero) begin (alarm-zero) PASS (alarm-zero) end Execution of 'alarm-zero' complete. Timer: 33 ticks Thread: 1 idle ticks, 35 kernel ticks, 0 user ticks Console: 381 characters output </pre>

idle tick의 값이 0 -> 1로 증가하였다.

2.6.6 alarm-negative

alarm-negative	
개선 전(busy-waiting)	개선 후(no-busy-waiting)
 <pre> pintos@pintos-VirtualBox: ~/pintos/src/threads Executing 'alarm-negative': (alarm-negative) begin (alarm-negative) PASS (alarm-negative) end Execution of 'alarm-negative' complete. Timer: 36 ticks Thread: 0 idle ticks, 38 kernel ticks, 0 user ticks Console: 381 characters output </pre>	 <pre> pintos@pintos-VirtualBox: ~/pintos/src/threads Executing 'alarm-negative': (alarm-negative) begin (alarm-negative) PASS (alarm-negative) end Execution of 'alarm-negative' complete. Timer: 35 ticks Thread: 1 idle ticks, 37 kernel ticks, 0 user ticks Console: 381 characters output </pre>

idle tick의 값이 0 -> 1로 증가하였다.

끝.