Operating System

Pintos Project #02

Alarm clock 의 개선

컴퓨터 과학과

201710957

이유진

pintos project #02

INDEX

**프로젝트 내용**

[<문제정의> 3](#_Toc9455428)

[<해결> 3](#_Toc9455429)

[<테스트> 3](#_Toc9455430)

**수정한 소스코드 및 테스트 결과**

[<수정한 소스코드> 4](#_Toc9455431)

[자료구조 생성 및 함수 선언 4](#_Toc9455432)

[timer\_sleep() 변경 5](#_Toc9455433)

[less\_cmp 6](#_Toc9455434)

[timer\_interrupt 6](#_Toc9455435)

[thread\_check() 7](#_Toc9455436)

[run\_action 수정 8](#_Toc9455437)

[<테스트 결과> 9](#_Toc9455438)

[alarm-multiple 9](#_Toc9455439)

[alarm-negative 9](#_Toc9455440)

[alarm-single 9](#_Toc9455441)

[alarm-zero 9](#_Toc9455442)

[alarm-simultaneous 9](#_Toc9455443)

[alarm-priority 10](#_Toc9455444)

프로젝트 내용

# <문제정의>

‘devices/timer.c’에 정의되어 있는 timer\_sleep() 함수를 개선한다.

현재의 timer\_sleep()은 기능적으로는 정상 동작하나**, busy waiting** 방식으로 구현되어 있다. 이를 busy waiting 없이 수행하도록 수정하여 성능을 개선한다

# <해결>

void timer\_sleep (int64 t ticks)를 호출한 쓰레드를 현재 시각 기준으로 ticks 시간이 경과할 때까지 block 시키도록 수정한다 (ticks 는 timer tick 단위로 표시). 이후 ticks timer tick 이상이 경과하면 해당 thread 를 ready 상태로 전이시키면 된다. 이를 위해 timer\_sleep()에 의해 block 된 스레드들이 대기할 수 있는 리스트를 만들고, 매 번 timer interrupt가 발생할 때마다 (즉 timer interrupt handler 함수 timer\_interrupt()에서) timer tick이 ticks 이상 지났는지 검사하는 방식으로 구현한다.

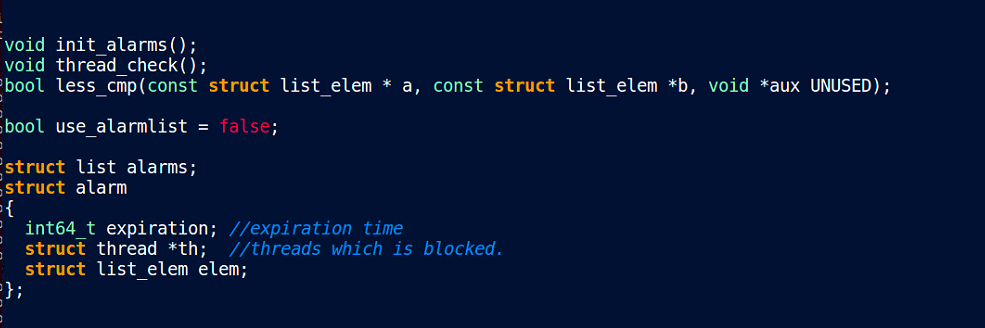
# <테스트>

구현된 timer\_sleep() 함수가 제대로 동작하는지 여부는 테스트 alarm-single, alarm-multiple, alarm-simultaneous, alarm-priority, alarm-zero, alarm-negative 를 사용하여 시험한다 (이 테스트들은 모두 timer\_sleep() 함수를 사용하는 프로그램들이다. 따라서 원본 pintos 커널에서의 이 테스트들의 결과와 timer\_sleep() 함수가 수정된 후의 결과가 동일하여야 한다)

수정한 소스코드 및 테스트 결과

# <수정한 소스코드>

## 자료구조 생성 및 함수 선언

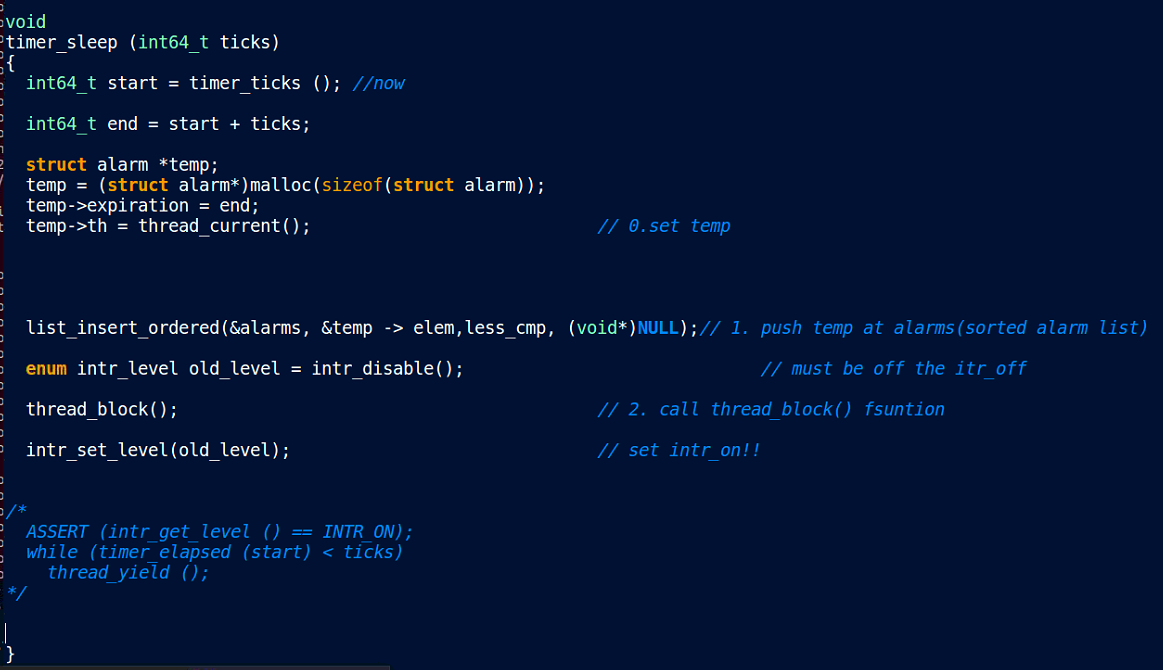


timer\_sleep()에 의해 block 된 스레드들이 대기할 수 있는 리스트인 alarms와, alarm 이라는 자료구조를 선언하였다.

alarm 은 ticks timer tick 이상이 경과하였는지 체크하기위한 int 형 변수expiration, block 된 thread의 포인터를 저장하기위한 struct thread포인터변수 소, element들을 연결할 변수 elem 을 선언한다.

그리고 list가 초기화가 진행이 되었는지 확인하는 bool 타입 use\_alarmlist를 선언 하고 false 로 초기화한다.

## timer\_sleep() 변경



start 에 현재 timer\_ticks()를 통해, 현재의 ticks를 저장하고, 변수 end 에 현재 ticks와(start)에 경과해야할 시간(tiks)를 더해 end 에 저장한다.

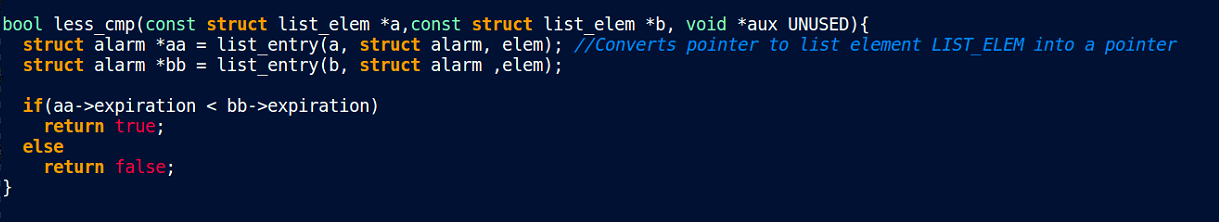
struct alarm 포인터 변수 temp를 선언한후, temp 에 malloc 을 이용하여 alarm 을 할당한다.

temp의 expiration를 end 로 초기화 하고, temp의 th은 현재 실행중인 thread의 주소 값을 넣는다. (thread\_current() 함수가 현재 실행중인 스레드의 주소 값을 반환)

alarm list에 list\_insert\_ordered 함수를 이용하여 alarms 에 insert 한다. insert ordered를 사용했기 때문에, alarms는 정렬되어있다. 이때 정렬 기준은 less\_cmp함수를 이용하고 다음페이지에서 설명하겠다.

thread\_block은 인터럽트를 꺼져있어야 하기 때문에, old\_level에 현상태를 저장하면서 인터럽트를 끄고 스레드를 block 한다. 그 후에 다시 인터럽트 상태를 복구한다.

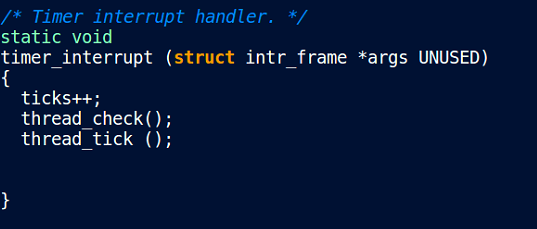
## less\_cmp



list\_insert\_ordered에서 사용한 비교 함수이다. list\_elem 형식의 a,b 파라미터를 이용하여 a < b 이면 true를 반환하는 함수이다.

이때 비교는 expiration 를 기준으로 해야한다. 따라서alarm의 포인터타입의 aa와 bb 에 list\_entry를 이용해 비교할 alarm 포인터를 담는다. aa와 bb에 담긴 expiration 에 담긴 값을 비교하여 bool값을 반환한다.

## timer\_interrupt

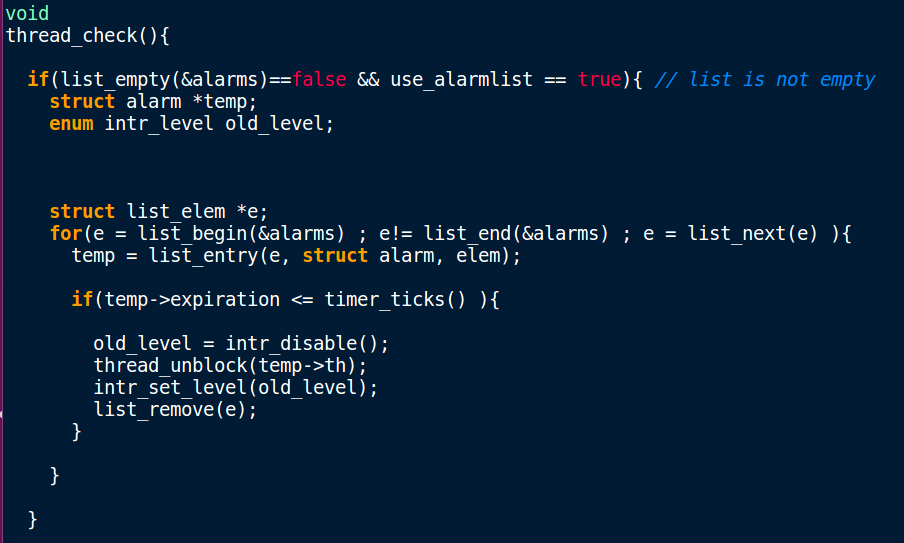


thread들을 리스트에 담은 후 매 번 timer interrupt가 발생할 때마다timer tick이 ticks 이상 지났는지 검사해야 한다.

따라서 직접 만든 thread\_check()함수를 timer\_interrupt 에서 실행하도록 했다.

thread\_check 함수는 다음 장에서 설명하도록 하겠다.

## thread\_check()

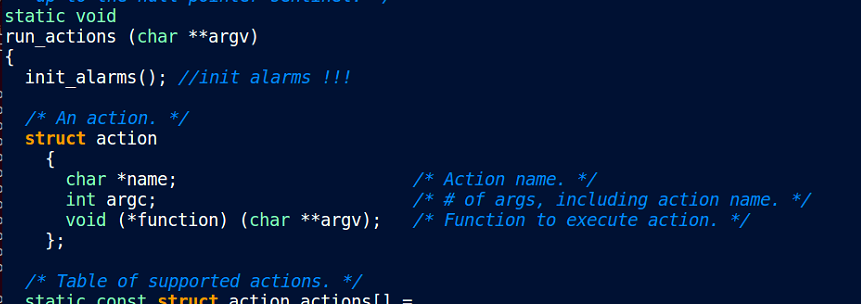


이 함수에서는 list가 비어 있지 않고, alams가 초기화 되어있을 경우 에만 실행해야 하기 때문에, if문을 이용하여 조건이 부합 할 때만 진행하도록 하였다.

alarm 리스트에 들어있는 모든 element들의 expiration을 확인하여, timer\_ticks() 의 반환값보다 작거나 같다면 그 스레드를 unblock 해줘야한다. thread\_unblock은 thread\_block과 같게 인터럽트가 꺼져있어야 하므로, 동일한 방법으로 인터럽트 상태를 임시 저장한후 인터럽트를 잠시 끄고 thread 를 unblock한 후 상태를 복원한다.

thread를 unblock 하고나면, unblock 한 리스트를 alarm에서 제거해야 하기 때문에 list\_remove를 이용하여 대기열에서 삭제하였다.

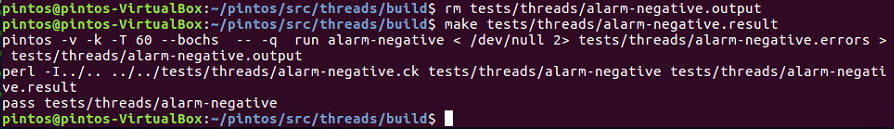
## run\_action 수정



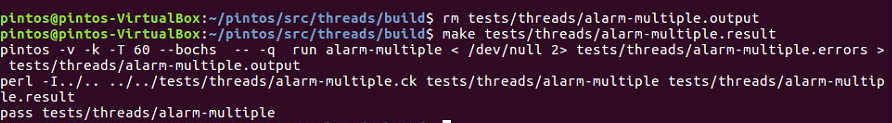
위의 테스트를 수행하려면, 테스트를 수행할때마다 시작 전에alarm 을 비어있는 리스트로 초기화를 시켜줘야 한다. 따라서 run\_action함수에서 init\_alarms()를 실행하여 매번 리스트를 초기화 시켰다.

# <테스트 결과>

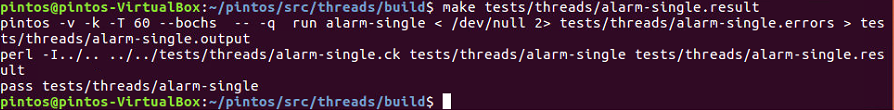
## alarm-multiple



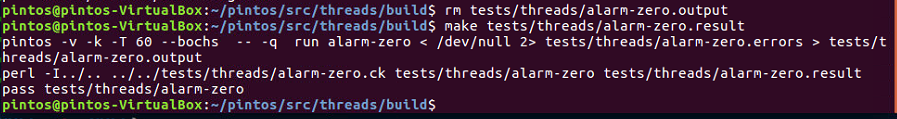
## alarm-negative



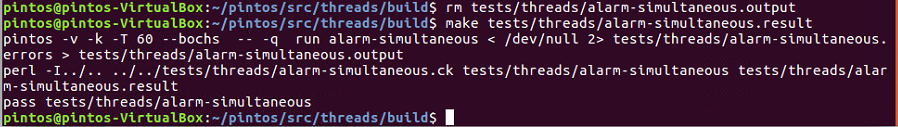
## alarm-single



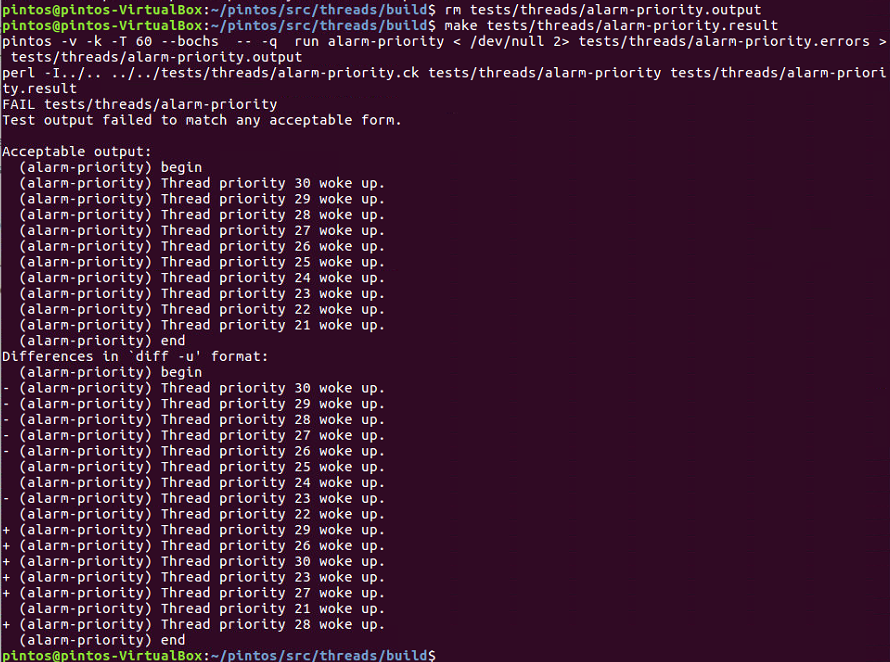
## alarm-zero



## alarm-simultaneous



## alarm-priority



위의 모든 테스트가 통과하였지만 우선순위 스케쥴러를 구현하기 전까지는 priority 관련 테스트는 통과하지 못하기 때문에 alarm 관련 테스트에서 alarm-priority 테스트만 실패하였다.