1. 기존의 timer\_sleep() 함수

void

timer\_sleep (int64\_t ticks)

{

int64\_t start = timer\_ticks();

ASSERT(intr\_get\_level() == INTR\_ON);

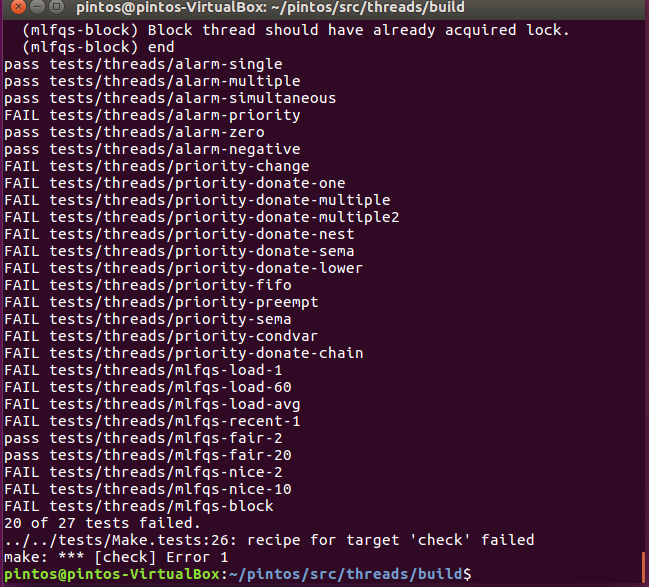
while(timer\_elapesd(start)<ticks)

thread\_yield();

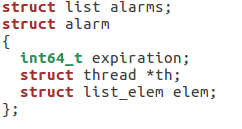
}

기존의 timer\_sleep() 함수는 매개변수로 수행 지연 시간인 ticks를 받고 수행 지연을 시작한 시각을 start 변수에 저장하여, 스레드의 실행 마다 수행 중지 시작 시점부터의 경과시간(timer\_elapesd(start))과 수행 지연 시간(ticks)을 비교하여 수행 지연 시간이 더 크다면 thread\_yield() 함수를 호출하는 busy waiting 방식이다.

기존 timer\_sleep() 함수 일 때 알람실행 결과

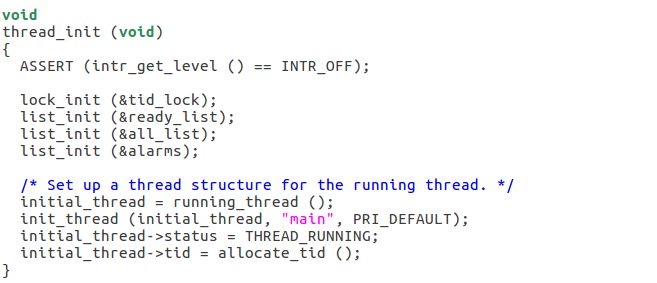


1. 알람을 위한 구조체 선언 (~/pintos/src/threads/thread.h)



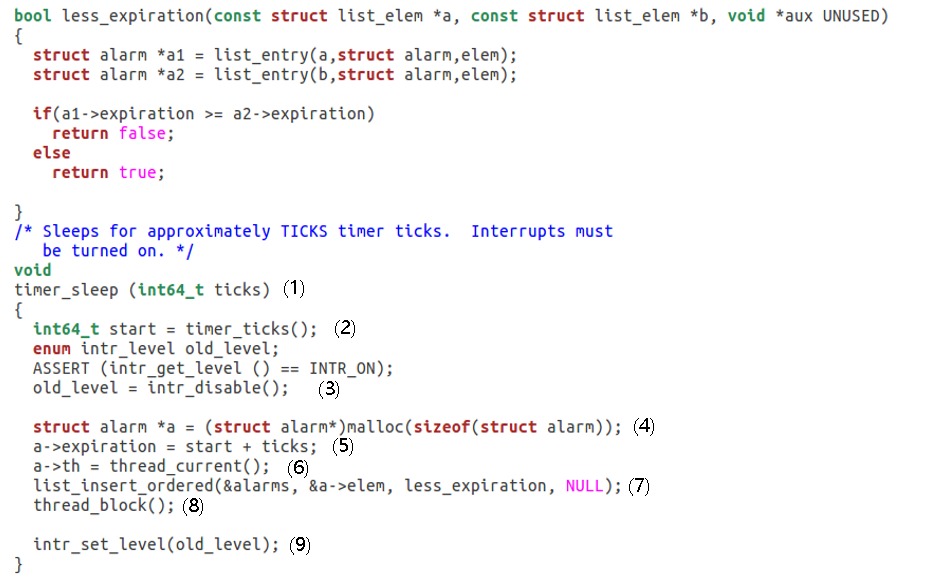
알람 만료시간을 저장할 expiration, 알람을 요청한 스레드를 저장할 th, 알람 리스트를 연결하기 위한 elem을 구조체 멤버 변수로 선언하여 alarm구조체를 선언하고, alarm구조체의 리스트인 alarms를 선언한다.

1. 알람리스트 초기화 (~/pintos/src/threads/thread.c)



thread를 초기화하는 함수인 thread\_init() 함수에 list\_init(&alarms)를 추가하여 알람리스트를 초기화 한다.

1. timer\_sleep() 함수 개선 (~/pintos/src/devices/timer.c)



1. 매개변수로 수행지연 시간인 ticks를 받는다.
2. start변수에 수행 지연 시각을 저장한다.
3. interrupt를 비활성화 시킨다.
4. 알람 구조체 변수 a를 선언한다.
5. alarm 구조체 멤버 변수 중 expiration은 알람 만료시간을 저장하는 변수이므로 수행 지연을 시작한 시각(start)에 수행지연 시간(ticks)를 더한 값을 저장한다.
6. th에는 알람을 요청한 스레드, 즉 현재 실행중인 스레드를 저장한다.
7. alarm구조체들을 저장하는 list인 alarms에 a를 연결한다. 이때 알람 만료시간에 대해 오름차순으로 정렬하기 위해 list\_insert\_orderd() 함수를 사용하였고, 정렬 방법을 명시해주기 위해 앞에 오는 구조체의 만료시간이 더 크면 false, 더 작으면 true를 반환하는 less\_expiration() 함수를 정의하고, list\_insert\_orderd()함수의 매개변수로 전달한다. list를 연결할 때에는 구조체의 멤버 변수 중 elem을 사용한다.
8. thread\_block()함수를 이용하여 알람을 요청한 스레드를 block 시킨다,.
9. 비활성화 시켰던 interrupt를 원래의 상태로 돌려놓는다.
10. timer\_interrupt() 함수 개선 (~/pintos/src/devices/timer.c)

(2)

(3)

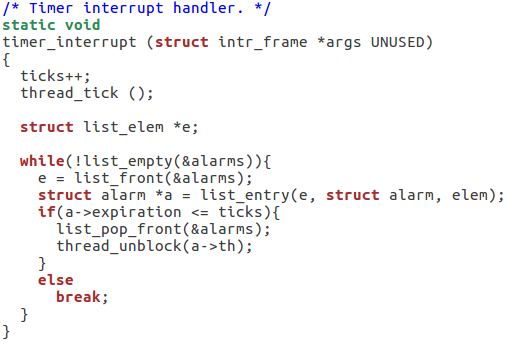
(6)

(5)

(4)

(7)

(1)



1. alarms가 비어있는 리스트가 아니면 반복문을 실행한다.
2. list\_front() 함수를 호출하여 alarms의 첫번째 원소에 접근한다. list\_front() 함수의 반환형은 list\_elem 이므로 해당 구조체 타입의 변수인 e를 선언하고 e에 저장한다.
3. list\_entry() 함수를 호출하여 e로부터 alarms에 맨앞에 연결된 alarm구조체를 변수 a에 저장한다.
4. 만약 시스템이 실행한 시간(ticks)이 구조체 변수 a에 저장된 만료시간보다 작거나 같으면 알람이 만료되었다는 의미이다.
5. 알람이 만료된 경우, list\_pop\_front() 함수를 이용하여 맨앞에 연결된 원소를 삭제한다.
6. 알람이 만료된 경우, 알람을 요청했던 스레드를 thread\_unblock()함수를 이용하여 unblock 시킨다.
7. 그렇지 않은 경우, 반복문을 빠져나온다. 리스트를 끝까지 검사하지 않는 이유는 alarm구조체를 alarms에 연결시킬 때 만료시간의 오름차순으로 연결하였으므로 앞의 스레드가 만료시간이 되지 않았다면 뒤의 스레드도 만료되지 않았다는 의미이기 때문이다.
8. 개선된 alarm clock의 alarm\_test 실행 결과

