Operating System

Pintos Project #02

Alarm clock 의 개선

컴퓨터 과학과

201710957

이유진

PINTOS PROJECT #02

INDEX

프로젝트 내용

수정한 소스코드 및 테스트 결과

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 interrupt 가 발생할 때마다 (즉 timer interrupt handler 함수 timer_interrupt()에서) timer tick 이 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

```
void init_alarms();
void thread_check();
bool less_cmp(const struct list_elem * a, const struct list_elem *b, void *aux UNUSED);

bool use_alarmlist = false;

struct list alarms;
struct alarm
{
  int64_t expiration; //expiration time
  struct thread *th;  //threads which is blocked.
  struct list_elem elem;
};
```

timer_sleep()에 의해 block 된 스레드들이 대기할 수 있는 리스트인 alarms 와, alarm 이라는 자료구조를 선언하였다.

alarm 은 ticks timer tick 이상이 경과하였는지 체크하기위한 int 형 변수 expiration, block 된 thread 의 포인터를 저장하기위한 struct thread 포인터변수 소, element 들을 연결할 변수 elem 을 선언한다.

그리고 list 가 초기화가 진행이 되었는지 확인하는 bool 타입 use_alarmlist 를 선언하고 false 로 초기화한다.

timer_sleep() 변경

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

  int64_t end = start + ticks;

  struct alarm *temp;
  temp = (struct alarm*)malloc(sizeof(struct alarm));
  temp->expiration = end;
  temp->th = thread_current();                          // 0.set temp



  list_insert_ordered(&alarms, &temp -> elem,less_cmp, (void*)NULL);// 1. push temp at alarms(sorted alarm list)

  enum intr_level old_level = intr_disable();                          // must be off the itr_off

  thread_block();                                       // 2. call thread_block() fsuntion

  intr_set_level(old_level);                            // set intr_on!!


/*
  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
*/

}
```

start 에 현재 timer_ticks()를 통해, 현재의 ticks 를 저장하고, 변수 end 에 현재 ticks 와(start)에 경과해야할 시간(tiks)를 더해 end 에 저장한다.

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

temp 의 expiration 를 end 로 초기화 하고, temp 의 th 은 현재 실행중인 thread 의 주소 값을 넣는다. (thread_current() 함수가 현재 실행중인 스레드의 주소 값을 반환)

alarm list 에 list_insert_ordered 함수를 이용하여 alarms 에 insert 한다. insert ordered 를 사용했기 때문에, alarms 는 정렬되어있다. 이때 정렬 기준은 less_cmp 함수를 이용하고 다음페이지에서 설명하겠다.

thread_block 은 인터럽트를 꺼져있어야 하기 때문에, old_level 에 현상태를 저장하면서 인터럽트를 끄고 스레드를 block 한다. 그 후에 다시 인터럽트 상태를 복구한다.

less_cmp

```
bool less_cmp(const struct list_elem *a,const struct list_elem *b, void *aux UNUSED){
  struct alarm *aa = list_entry(a, struct alarm, elem); //Converts pointer to list element LIST_ELEM into a pointer
  struct alarm *bb = list_entry(b, struct alarm ,elem);

  if(aa->expiration < bb->expiration)
    return true;
  else
    return false;
}
```

list_insert_ordered 에서 사용한 비교 함수이다. list_elem 형식의 a,b 파라미터를 이용하여 a < b 이면 true 를 반환하는 함수이다.

이때 비교는 expiration 를 기준으로 해야한다. 따라서 alarm 의 포인터타입의 aa 와 bb 에 list_entry 를 이용해 비교할 alarm 포인터를 담는다. aa 와 bb 에 담긴 expiration 에 담긴 값을 비교하여 bool 값을 반환한다.

timer_interrupt

```
/* Timer interrupt handler. */
static void
timer_interrupt (struct intr_frame *args UNUSED)
{
  ticks++;
  thread_check();
  thread_tick ();


}
```

thread 들을 리스트에 담은 후 매 번 timer interrupt 가 발생할 때마다 timer tick 이 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()

```
void
thread_check(){

  if(list_empty(&alarms)==false && use_alarmlist == true){ // list is not empty
    struct alarm *temp;
    enum intr_level old_level;



    struct list_elem *e;
    for(e = list_begin(&alarms) ; e!= list_end(&alarms) ; e = list_next(e) ){
      temp = list_entry(e, struct alarm, elem);

      if(temp->expiration <= timer_ticks() ){

        old_level = intr_disable();
        thread_unblock(temp->th);
        intr_set_level(old_level);
        list_remove(e);
      }

    }

  }
```

이 함수에서는 list 가 비어 있지 않고, alams 가 초기화 되어있을 경우 에만 실행해야 하기 때문에, if 문을 이용하여 조건이 부합 할 때만 진행하도록 하였다.

alarm 리스트에 들어있는 모든 element 들의 expiration 을 확인하여, timer_ticks() 의 반환값보다 작거나 같다면 그 스레드를 unblock 해줘야한다. thread_unblock 은 thread_block 과 같게 인터럽트가 꺼져있어야 하므로, 동일한 방법으로 인터럽트 상태를 임시 저장한후 인터럽트를 잠시 끄고 thread 를 unblock 한 후 상태를 복원한다.

thread 를 unblock 하고나면, unblock 한 리스트를 alarm 에서 제거해야 하기 때문에 list_remove 를 이용하여 대기열에서 삭제하였다.

run_action 수정

```
static void
run_actions (char **argv)
{
  init_alarms(); //init alarms !!!

  /* An action. */
  struct action
    {
      char *name;                       /* Action name. */
      int argc;                         /* # of args, including action name. */
      void (*function) (char **argv);   /* Function to execute action. */
    };

  /* Table of supported actions. */
  static const struct action actions[] =
```

위의 테스트를 수행하려면, 테스트를 수행할때마다 시작 전에 alarm 을 비어있는 리스트로 초기화를 시켜줘야 한다. 따라서 run_action 함수에서 init_alarms()를 실행하여 매번 리스트를 초기화 시켰다.

<테스트 결과>

alarm-multiple

```
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ rm tests/threads/alarm-negative.output
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ make tests/threads/alarm-negative.result
pintos -v -k -T 60 --bochs  -- -q  run alarm-negative < /dev/null 2> tests/threads/alarm-negative.errors >
 tests/threads/alarm-negative.output
perl -I../.. ../../tests/threads/alarm-negative.ck tests/threads/alarm-negative tests/threads/alarm-negati
ve.result
pass tests/threads/alarm-negative
pintos@pintos-VirtualBox:~/pintos/src/threads/build$
```

alarm-negative

```
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ rm tests/threads/alarm-multiple.output
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ make tests/threads/alarm-multiple.result
pintos -v -k -T 60 --bochs  -- -q  run alarm-multiple < /dev/null 2> tests/threads/alarm-multiple.errors >
 tests/threads/alarm-multiple.output
perl -I../.. ../../tests/threads/alarm-multiple.ck tests/threads/alarm-multiple tests/threads/alarm-multip
le.result
pass tests/threads/alarm-multiple
```

alarm-single

```
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ make tests/threads/alarm-single.result
pintos -v -k -T 60 --bochs  -- -q  run alarm-single < /dev/null 2> tests/threads/alarm-single.errors > tes
ts/threads/alarm-single.output
perl -I../.. ../../tests/threads/alarm-single.ck tests/threads/alarm-single tests/threads/alarm-single.res
ult
pass tests/threads/alarm-single
pintos@pintos-VirtualBox:~/pintos/src/threads/build$
```

alarm-zero

```
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ rm tests/threads/alarm-zero.output
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ make tests/threads/alarm-zero.result
pintos -v -k -T 60 --bochs  -- -q  run alarm-zero < /dev/null 2> tests/threads/alarm-zero.errors > tests/t
hreads/alarm-zero.output
perl -I../.. ../../tests/threads/alarm-zero.ck tests/threads/alarm-zero tests/threads/alarm-zero.result
pass tests/threads/alarm-zero
pintos@pintos-VirtualBox:~/pintos/src/threads/build$
```

alarm-simultaneous

```
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ rm tests/threads/alarm-simultaneous.output
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ make tests/threads/alarm-simultaneous.result
pintos -v -k -T 60 --bochs  -- -q  run alarm-simultaneous < /dev/null 2> tests/threads/alarm-simultaneous.
errors > tests/threads/alarm-simultaneous.output
perl -I../.. ../../tests/threads/alarm-simultaneous.ck tests/threads/alarm-simultaneous tests/threads/alar
m-simultaneous.result
pass tests/threads/alarm-simultaneous
pintos@pintos-VirtualBox:~/pintos/src/threads/build$
```

alarm-priority

```
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ rm tests/threads/alarm-priority.output
pintos@pintos-VirtualBox:~/pintos/src/threads/build$ make tests/threads/alarm-priority.result
pintos -v -k -T 60 --bochs  -- -q  run alarm-priority < /dev/null 2> tests/threads/alarm-priority.errors >
 tests/threads/alarm-priority.output
perl -I../.. ../../tests/threads/alarm-priority.ck tests/threads/alarm-priority tests/threads/alarm-priori
ty.result
FAIL tests/threads/alarm-priority
Test output failed to match any acceptable form.

Acceptable output:
  (alarm-priority) begin
  (alarm-priority) Thread priority 30 woke up.
  (alarm-priority) Thread priority 29 woke up.
  (alarm-priority) Thread priority 28 woke up.
  (alarm-priority) Thread priority 27 woke up.
  (alarm-priority) Thread priority 26 woke up.
  (alarm-priority) Thread priority 25 woke up.
  (alarm-priority) Thread priority 24 woke up.
  (alarm-priority) Thread priority 23 woke up.
  (alarm-priority) Thread priority 22 woke up.
  (alarm-priority) Thread priority 21 woke up.
  (alarm-priority) end
Differences in `diff -u' format:
  (alarm-priority) begin
- (alarm-priority) Thread priority 30 woke up.
- (alarm-priority) Thread priority 29 woke up.
- (alarm-priority) Thread priority 28 woke up.
- (alarm-priority) Thread priority 27 woke up.
- (alarm-priority) Thread priority 26 woke up.
  (alarm-priority) Thread priority 25 woke up.
  (alarm-priority) Thread priority 24 woke up.
- (alarm-priority) Thread priority 23 woke up.
  (alarm-priority) Thread priority 22 woke up.
+ (alarm-priority) Thread priority 29 woke up.
+ (alarm-priority) Thread priority 26 woke up.
+ (alarm-priority) Thread priority 30 woke up.
+ (alarm-priority) Thread priority 23 woke up.
+ (alarm-priority) Thread priority 27 woke up.
  (alarm-priority) Thread priority 21 woke up.
+ (alarm-priority) Thread priority 28 woke up.
  (alarm-priority) end
pintos@pintos-VirtualBox:~/pintos/src/threads/build$
```

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