**Pintos 프로젝트1. Pintos Thread**

**(설계 프로젝트 수행 결과)**

**과목명 : [CSE4070-01] 운영체제**

**담당교수 : 서강대학교 컴퓨터공학과 박성용**

**조원 : 11조 권나영 박상욱**

**개발기간 : 2016. 11. 20. -2016. 11. 27.**

**최 종 보 고 서**

**프로젝트 제목: Pintos 프로젝트1. Pintos Thread**

**제출일: 2016. 11. 30 .**

**참여조원: 11조 권나영 박상욱**

**I. 개발 목표**

Alarm clock을 구현하여 특정 Thread가 일정 부분을 계속 반복 수행하면서 대기하는 Busy waiting문제를 고치고, 프로세스마다 우선순위를 부여하여 우선순위가 높은 프로세스가 CPU에 먼저 할당되도록 하는 Priority Scheduling을 구현한다.

**II. 개발 범위 및 내용**

**가. 개발 범위**

1) Alarm clock

 ‘devices/timer.c’ 파일 안에 있는 timer\_sleep() 함수 안에서 Busy Waiting문제(Thread가 일정 부분을 계속 반복 수행하면서 대기하는 상태로, CPU time낭비를 초래)를 아래와 같은 방식으로 수정한다. While문을 반복적으로 수행하며 시간을 체크하는 방법 대신, 시간 체크 후 설정된 시간이 되지 않았으면 Thread를 BLOCK 상태로 만들고 추후 시간이 되었을 때(wakeup 시간) READY 상태로 만들어 Ready Queue(ready\_list)에 삽입해주는 방법을 사용하여 해결하도록 한다.

 BLOCK 상태로 들어간 Thread를 저장해주기 위한 새로운 Queue(sleeping\_list)를 생성하고, Thread 저장 시 wake up할 시간을 저장하여 그 시간을 주기적으로 확인한다. Wake up할 시간이 다된 thread가 존재할 경우 다시 Ready Queue(ready\_list)로 삽입하는 방법을 사용하도록 한다.

2) Priority Scheduling

기존의 pintos에서는 새로운 thread가 입력되면 Ready Queue(ready\_list)의 맨 끝에 추가하는 구조와 달리,Priority scheduling은 각 프로세스마다 우선순위를 부여하여 우선순위가 높은 프로세스(thread)부터 수행하는 scheduling 기법이다. 각 thread에 우선순위를 부여하여 scheduling이 우선순위에 의해 수행하도록 수정하도록 한다. priority가 낮은 process들은 ready queue에서 turn을 계속해서 놓치는 starvation될 수 있다는 단점을 갖고 있다. 따라서 특정 시간이 지난 후에는 priority를 올려주는 aging기법을 구현한다.

3) Fixed-Point Real Arithmetic

BSD Scheduler를 구현하기 위해서는 bsd scheduler만의 계산방식으로 우선순위를 계산해야 하는데,계산식에 들어가는 recent cpu and load avg는 실수이다. Pintos의 kernel에서는 floating-point계산을 지원하지 않기 때문에 실수에 대한 사칙연산에 대해 구현해야한다.

-load\_avg = (59/60)\*load\_avg + (1/60)\*ready\_threads (ready threads 는 idle thread를 뺀,실행 중이거나 기다리고 있는 thread들의 개수).

-recent\_cpu = (2\*load\_avg)/(2\*load\_avg + 1) \* recent\_cpu + nice. Priority.

두 개의 floating point사칙연산과 float과 int사이의 변환에 대한 함수를 만들어준다.

4)BSD Scheduler

기존의 1개의ready\_list를사용하는대신 priority 개수만큼 multi-level의 list를만들어서 스케쥴링한다. 기존의 ready\_list에 적용되는 스케쥴링 함수들을 64 level의 list에 알맞게 연산을 수정한다.

**나. 개발 내용**

**1. Alarm Clock**

- timer.c

Busy waiting 문제를 해결하기 위해 주어진 함수인timer\_sleep()을 수정한다. 일정한 주기(tick)마다의 interrupt발생을 위한 timer\_interrupt()를 수정한다.

- thread.c

Thread를 재우기 위한 현재 thread를 wake\_up\_tick까지 block시키고 thread를 sleeping\_list라는 리스트에 넣어주는 thread\_sleep()이라는 함수를 만든다.

또한,thread\_wake\_up()함수를이용하여wake\_up\_tick이지났을때thread를unblock해준다.sleeping\_list를정렬된꼴로만들기위해list\_insert\_ordered()함수를사용했고그함수의paramter로cmp\_sleeping\_thread()함수를이용한다.

**2. Priority Scheduling**

 - init.c : Priority scheduling aging 관련하여 설정 코드를 추가한다.

- thread.c :thread를Ready\_list에 추가할 때 우선순위개념이 더해진 scheduling을 수행하도록 한다.이 때 thread들 간의 priority를 비교하는 cmp\_priority\_thread()함수를 이용한다.Priority scheduling을 위한 recent cpu calculation, load average calculation,priority calculation과 aging 구현 코드를 추가한다. 적절한 시기가 되면(적절한 명령이 들어오면 aging flag를 1로 set되는데 이것으로 aging의 시기를 판단), ready\_list에 있는 thread의 priority를 증가시켜 주는 thread\_aging()함수를 만들어서 aging을 구현한다.

**3. Fixed point**

pintos메뉴얼에 제시한 그대로 구현하였다. Lib/kernel/arith\_fp.h에 아래의 함수들의 prototype을 선언하였다.그리고 Make.buildfile을 수정하여 같이 build될 수 있도록 하였다.

#defineFP\_SIGN1

#defineFP\_INTEGER17

#defineFP\_DECIMAL14

#defineNUM\_1 (1<< FP\_DECIMAL)

int

int\_to\_FP(int integer){

return integer \* NUM\_1;

}

int

FP\_to\_int(int decimal){

return decimal / NUM\_1;

}

int

add\_FP\_to\_FP(intdec\_a,intdec\_b){

returndec\_a+dec\_b;

}

int

mul\_FP\_to\_FP(intdec\_a,intdec\_b){

return (int)((long long)dec\_a \* dec\_b / NUM\_1);

}

int

sub\_FP\_to\_FP(intdec\_a,intdec\_b){

returndec\_a-dec\_b;

}

int

div\_FP\_to\_FP(intdec\_a,intdec\_b){

return (int)((long long)dec\_a \* NUM\_1 / dec\_b);

}

int\_to\_FP(int integer) :int값을 2^14만큼 곱해줘 17.14 fixed point 값으로 변환하는 함수이다.

FP\_to\_int(int decimal) : 17.14 fixed point 값을 2^14만큼 나눠줘 실수부를 버리고 int값으로 변환하는 함수이다.

add\_FP\_to\_FP(): 17.14 fixed point값끼리의 덧셈,

mul\_FP\_to\_FP() :17.14 fixed point값끼리의곱셈,

sub\_FP\_to\_FP() :17.14 fixed point값끼리의뺄셈,

div\_FP\_to\_FP() :17.14 fixed point값끼리의나눗셈을구현한함수이다.

**4. BSD Scheduling**

 - thread.c : 각 thread들의 priority연산에 필요한 load\_avg(Ready 상태의 thread 수의 평균을 추정한 값), recent\_cpu(Thread의 최근 CPU 사용 시간을 표현.), nice(우선순위를 낮추어, 다른 Thread들이 보다 CPU를 사용할 수 있게 하기 위한 값)을 계산하기위해만든 fixed point calculation 함수를 이용하여 priority를계산하고, 4 tick 마다 시스템 내의 모든 thread의 우선순위를 재계산하여 scheduling 한다.

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

- 11월 22일 ~ 11월 23일 : proj#1 ppt를 분석하고 alarm clock까지 구현하여 alarm clock에 관련된 test를 모두 통과하였다.

- 11월 24일 ~ 11월 25일 :기본적인 구현을 다 하고 priority scheduling을 어떻게 해야할 지 구상하고 priority scheduling까지 구상하였다.

- 11월 26일 ~ 11월 28일 :aging을 구현하고 mlfqs에 쓰일 fixed point arithmetic, Mlfqs를 구현하였다.

**나. 개발 방법**

csprp1과 cspro2서버에서 vi editor를 가지고 강의자료와 pintos매뉴얼을 참고하여 개발한다.

개발 환경은 linux 컴파일 환경은 GCC이다.

**다. 연구원 역할 분담**

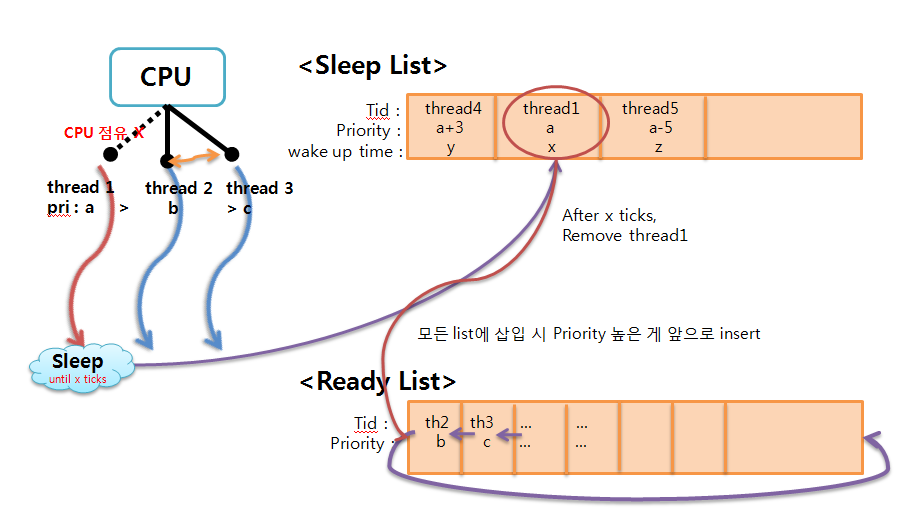
권나영 :매뉴얼 분석, proj#1ppt자료 분석, coding

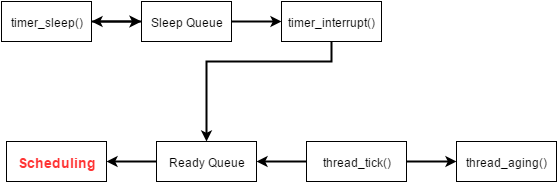
박상욱 :매뉴얼 분석, proj#1ppt자료 분석, coding

**IV. 연구 결과**

**1. 합성 내용:**

**- 설계 목표에 필요한 내용을 조사 분석한 후 그들을 바탕으로 구성한 전체 프로그램 구성도**





**2. 제작 내용:** 개발 결과

2\_1) 자료구조

① Structthread에 wake\_up\_tick 추가한다.

struct thread

{

/\* Owned by thread.c. \*/

tid\_ttid; /\* Thread identifier. \*/

enumthread\_status status; /\* Thread state. \*/

char name[16]; /\* Name (for debugging purposes). \*/

uint8\_t \*stack; /\* Saved stack pointer. \*/

int priority; /\* Priority. \*/

structlist\_elemallelem; /\* List element for all threads list. \*/

/\* Shared between thread.c and synch.c. \*/

structlist\_elemelem; /\* List element. \*/

#ifdef USERPROG

/\* Owned by userprog/process.c. \*/

uint32\_t \*pagedir; /\* Page directory. \*/

#endif

//Added -psu 2016.10.26

struct list child\_tlist; /\* List for child threads. \*/

structlist\_elemchild\_elem; /\* child's INFO. \*/

intis\_loaded; /\* Check for the thread is successfully loaded. \*/

intexit\_status; /\* Save the Status returned by exit. \*/

//modefied to replace busy-waiting. Synchronizing.

struct semaphore sema\_wait;

struct semaphore sema\_load;

struct semaphore sema\_elim;

//Added -kny 2016.11.13

/\*project 2\_2\*/

intfd\_num; //number of filedescriptor which has been used ,즉next\_fd

struct file \*fd\_tab[128];

struct file \*exec\_file;

/\* Owned by thread.c. \*/

unsigned magic; /\* Detects stack overflow. \*/

int64\_t wake\_up\_tick;

};

② thread.c에 sleeping\_list를 추가한다. Thread\_init(void)함수에서 sleeping\_list에 대해 초기화 해줘야한다.

Static struct list sleeping\_list;

2\_2) 알고리즘 또는 구현한 함수

**<timer.c>**

timer\_sleep() : 파라미터로 넘어온 ticks(sleep상태로 있어야할 시간)과 timer\_ticks()의 반환 값, 즉 현재 시간을 더해서 깨야할 시간(wake up time)을 thread\_sleep()의 파라미터로 넘겨준다. 기존의 호출하던 thread\_yield() 대신 thread\_sleep()를 호출한다.

timer\_interrupt() : tick이 증가함에 따라 thead\_tick()을 호출하고 sleeping\_list를 탐색하여 thread의wake\_up\_tick과 tick을 비교하여 둘이 같거나 현재 tick이 크면 list\_pop\_front()를 이용하여 맨 앞에 존재하는 thread를 깨운다.

**<synch.c>**

Ready list 나 wait list를 우선순위를 반영하여 만들어야하기 때문에 기존에 썼던 list\_push\_back()함수대신 list\_insert\_ordered()함수를 이용하였다.

|  |
| --- |
| void |
|  | sema\_down (struct semaphore \*sema) |
|  | { |
|  | enumintr\_levelold\_level; |
|  |  |
|  | ASSERT (sema != NULL); |
|  | ASSERT (!intr\_context ()); |
|  |  |
|  | old\_level = intr\_disable (); |
|  | while (sema->value == 0) |
|  | { |
|  | //list\_push\_back (&sema->waiters, &thread\_current ()->elem); |
|  | list\_insert\_ordered(&sema->waiters,&thread\_current ()->elem, cmp\_priority\_thread,0); |
|  | thread\_block (); |
|  | } |
|  | sema->value--; |
|  | intr\_set\_level (old\_level); |
|  | } |

Sema\_down에서 우선순위에 따라 wait list를 만들기 위해 우선순위를 비교하여 정렬된 채로 유지하게 해주는 cmp\_priority\_thread(conststructlist\_elem\* a,conststructlist\_elem\* b,void \*)를이용하였다.이함수는parameter로들어간 thread a 와thread b의 priority를비교하여 a가크면 true를반환하고그반대면 false를반환한다.

|  |  |
| --- | --- |
| void | |
|  | | sema\_up (struct semaphore \*sema) | |
|  | | { | |
|  | | enumintr\_levelold\_level; | |
|  | |  | |
|  | | ASSERT (sema != NULL); | |
|  | |  | |
|  | | old\_level = intr\_disable (); | |
|  | | if (!list\_empty (&sema->waiters)) | |
|  | | thread\_unblock (list\_entry (list\_pop\_front (&sema->waiters), | |
|  | | struct thread, elem)); | |
|  | | sema->value++; | |
|  | | intr\_set\_level (old\_level); | |
|  | | #ifndef USERPROG | |
|  | | thread\_yield(); | |
|  | | #endif | |
|  | | } | |
| void | |
|  | | thread\_unblock (struct thread \*t) |
|  | | { |
|  | | enumintr\_levelold\_level; |
|  | |  |
|  | | ASSERT (is\_thread (t)); |
|  | |  |
|  | | old\_level = intr\_disable (); |
|  | | ASSERT (t->status == THREAD\_BLOCKED); |
|  | | //list\_push\_back (&ready\_list, &t->elem); |
|  | | list\_insert\_ordered(&ready\_list,&t->elem,cmp\_priority\_thread,0); //Change for priortity scheduling. |
|  | | t->status = THREAD\_READY; |
|  | | intr\_set\_level (old\_level); |
|  | | } |
| void | |
|  | | thread\_yield (void) |
|  | | { |
|  | | struct thread \*cur = thread\_current (); |
|  | | enumintr\_levelold\_level; |
|  | |  |
|  | | ASSERT (!intr\_context ()); |
|  | |  |
|  | | old\_level = intr\_disable (); |
|  | | if (cur != idle\_thread) |
|  | | //list\_push\_back (&ready\_list, &cur->elem); |
|  | | list\_insert\_ordered(&ready\_list,&cur->elem, cmp\_priority\_thread,0); //Change for priority schedule. |
|  | | cur->status = THREAD\_READY; |
|  | | schedule (); |
|  | | intr\_set\_level (old\_level); |
|  | | } |

Sema\_up에서 우선순위에 따라 ready list를 만들기 위해 우선순위를 비교하여 정렬된 채로 유지하게 해주는 cmp\_priority\_thread(conststructlist\_elem\* a,conststructlist\_elem\* b,void \*)를이용하였다.

**<thread.c>**

|  |
| --- |
|  |

thread\_tick (void) : 매 tick마다 timer interrupt에 의해 불려지는 함수이다. Thread\_mlfps이 수행되는 상황에서는 매초마다ready\_threads의 개수와 load\_avg, recent\_cpu를 계산하고, 4초마다 한번씩 priority를 계산한다. Thread에 aging이 필요한 상황일 경우 이 함수 내에서 thread\_aging을 호출하여 aging을 해준다.

thread\_create (const char \*name, int priority, thread\_func \*function, void \*aux) : parameter로 받은 name이라는 이름을 가진 새로운 thread를 생성하고 \*priority를 비교하여 ready\_list에 넣어준다. (\*는 수정된 기능)

thread\_yield (void) : Yields the CPU. The current thread is not put to sleep and may be scheduled again immediately at the scheduler's whim. idle\_thread가 아니면 \*priority를 고려하여 ready\_list에 넣어주는 기능이 추가된다.

thread\_aging(void) : 우선순위 스케쥴링에서 starvation 문제를 막기 위해 aging 방법을 구현한다.

thread\_set\_nice (int nice UNUSED) : 현재 thread의 nice값을 parameter로 넘어온 nice값으로 설정해준다.

thread\_get\_nice (void) : 현재 thread의 nice값을 반환한다.

thread\_get\_load\_avg (void) : load average를 ppt에 나온 식을 이용하여 값을 구하고 반환한다.

thread\_get\_recent\_cpu (void) : thread의 recent\_cpu value를 ppt에 나온 식을 이용하여 값을 구하고 반환한다.

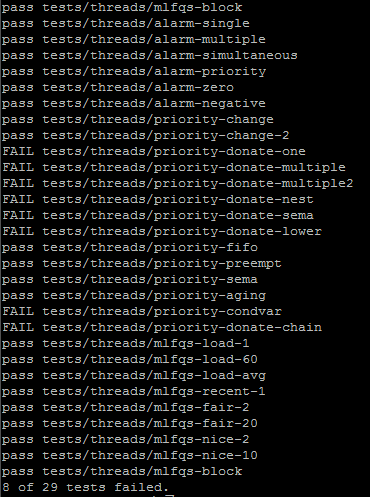
init\_thread (struct thread \*t, const char \*name, int priority) : thread t가 initial\_thread인지 아닌지를 확인하고 initial\_thread이면 nice, recent\_cpu값을 0으로 설정해주고, 아니면 parent의 thread의 nice값과 recent\_cpu값과 같게 설정해주는 것을 추가한다.

**3. 시험 및 평가 내용:**

- 평가 방법에 대한 설명을 기술하라.(수행 sequence 등)

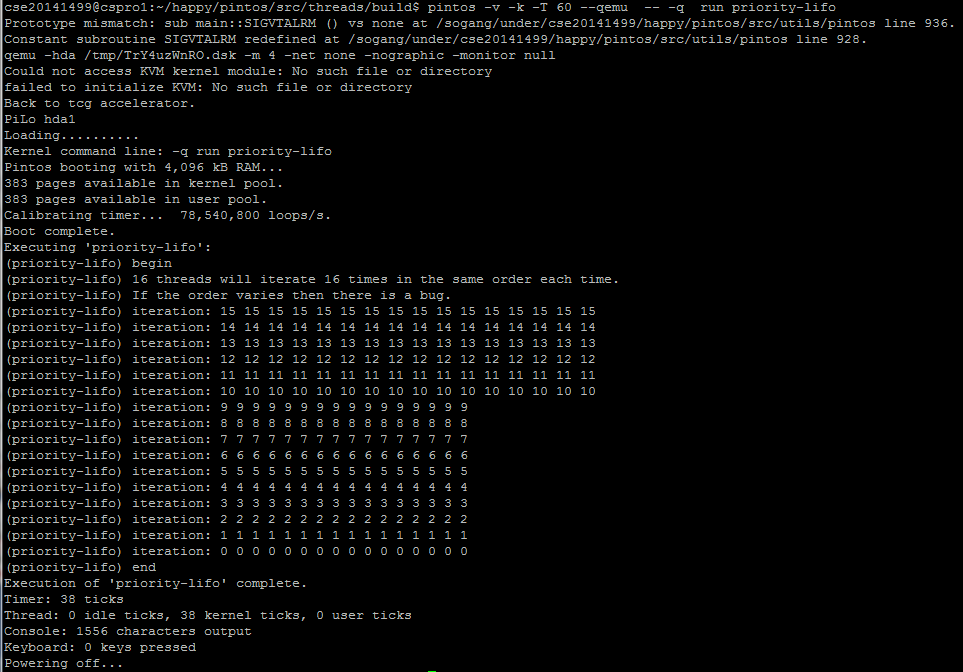
- 자신들의 결과물이 갖는 **보건 및 안정**, **생산성과 내구성**에 대하여 반드시 기술할 것.

**<MAKE CHECK 수행화면 >**

****

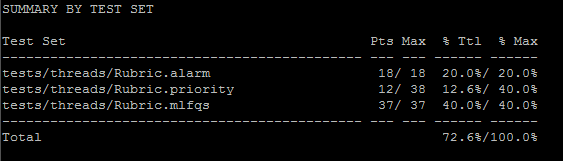
기본구현 13개 이외에도 추가구현에 대한mlfqs test들이 모두 수행되는 것을 알 수 있다.

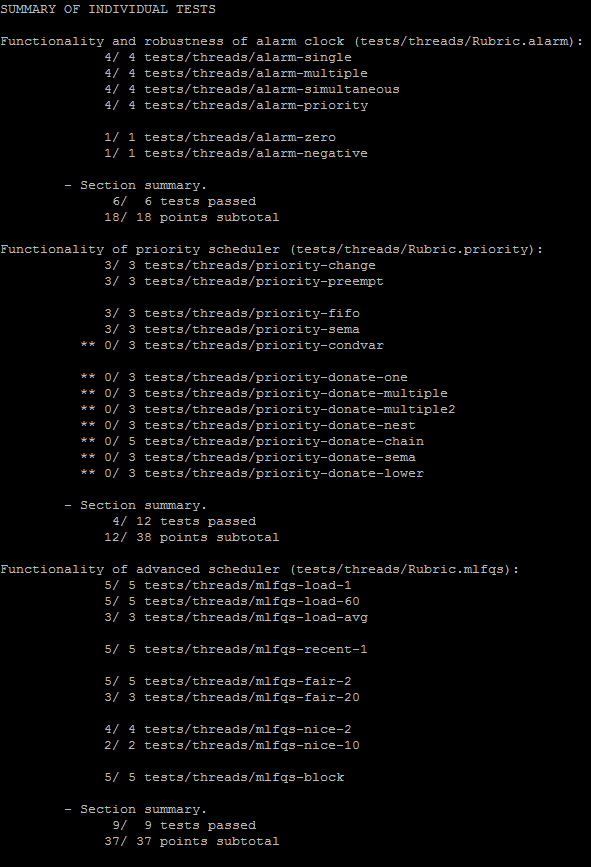
**<Priority-lifo test수행 결과>**



우선, 16\*16개의 array에 16개의 threads는 실행될 때 마다, PRI\_DEFAULT+1부터 16개를 넣고, +16까지 들어가게된다. 하지만 Main이 PRI\_DEFAULT+17이므로 수행되지 못하고 대기하다가, Main 이 default값으로priority를 낮추면 우리 프로그램은 priority가 높은 thread부터 수행되도록 schedule되어 있으므로, 나중에 들어갔지만 제일 우선순위가 높은 thread부터 실행되어 위와 같은 출력결과를 얻는다. 즉, thread들이 LIFO식으로 맨 나중에 생성된 thread부터 먼저 수행되고 맨 처음에 들어간 녀석이 나중에 튀어나오게 되므로 제대로 구현되었다 할 수 있다.

**<MAKE　GRADE 수행화면>**

****

****

**V. 기타**

- 기타 관련 내용을 기술할 것.

**1.연구 조원 기여도**: 기여도를 백분율로 표현할 것.

권나영 50%, 박상욱 50%

**2.**기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

권나영 : thread의 개념을 수업시간에는 배웠지만 마치 동시에 수행되는 것처럼 수행하기위해 schedule되고 수행되는 과정을 직접 구성해보니 좀 더 심화된 프로그램을 짜는 듯 했다. 지금은 pintos의 scheduling 이슈에 맞춰서 했지만, 수업시간에 배운 다른 scheduling 방식도 사용해 보고 비교해 볼 수 있다면 좋겠다는 생각도 했었다. 이전에는 CPU사용에 대한 효율성을 전혀 생각해보지 않았지만, 깊게 생각해보고 프로그래밍을 해보는 계기가 되어서 좋았다.

박상욱