**Pintos Project 3 : Threads**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 김영재 교수님

조 / 조원 : 50조, 20171672 이정원, 20171698 최아연

개발 기간 : 2019/11/16 – 2019/12/08

**프로젝트 제목 : Pintos Project 3 Threads (3)**

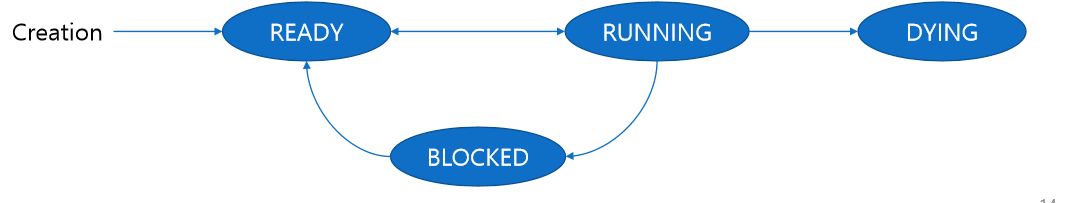
**제출일 : 2019년 12월 8일**

**참여 조원 : 이정원, 최아연**

1. **개발 목표**

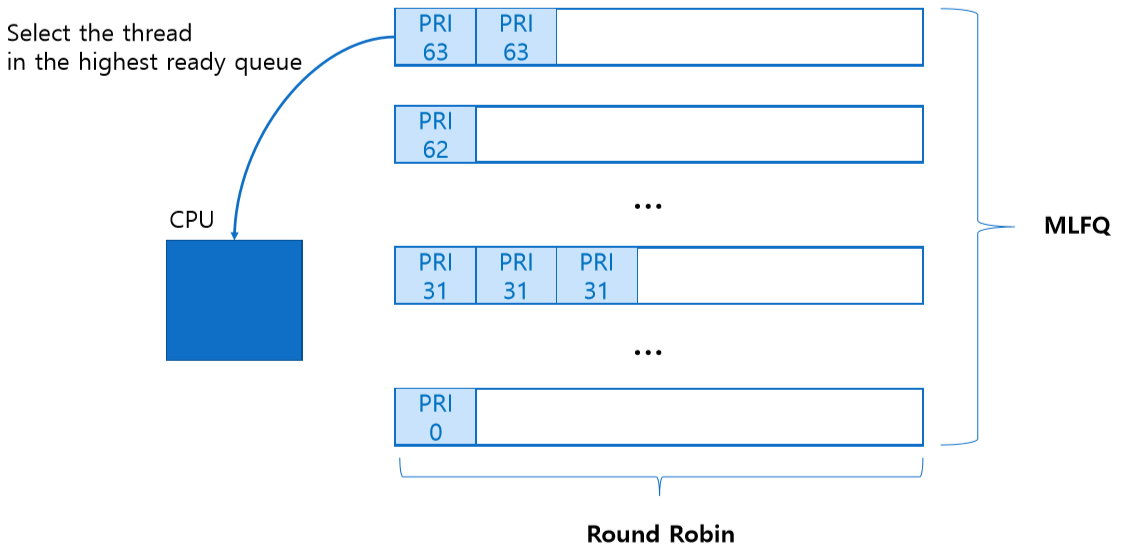
* Project 1, 2에서 사용자의 명령을 수행하는 대부분의 기능을 구현했다. Pintos의 경우 기본적으로 round-robin 방식으로 scheduling을 실행하는데, 이는 thread의 우선 순위를 반영하지 못한다.
* 따라서 이번 project 3에서는 thread의 우선순위를 반영하는 process scheduling을 구현한다. 이를 위해 thread와 여러 synchronization 기법이 활용된다.
* Semaphore, lock등의 기법을 통해 synchronization을 구현하고, 추가적으로 BSD scheduler을 구현한다. BSD scheduling을 위해선 MLFQ나 MLRQ의 기법이 사용된다.

1. **개발 범위 및 내용**
   1. **개발 범위**

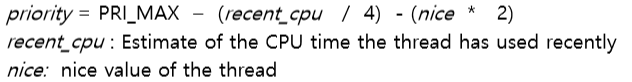
* **Alarm clock**
* 기존 Pintos의 timer\_sleep()은 비효율적인 방식인 busy waiting 기법으로 구현되어 있다. 이 방식은 특정 thread가 계속 일정부분을 반복하며 대기하는 상태로, 자원 낭비가 발생한다. 이 cpu 자원 낭비를 없애는 방식으로 코드를 수정한다.
* 

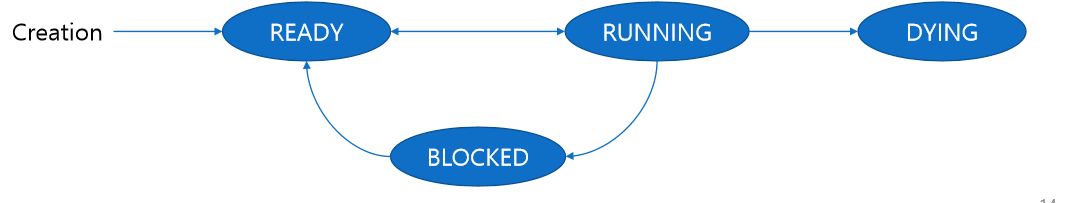
기존의 busy waiting 방식이 아니 위의 방식으로 thread가 실행되도록 변형한다.

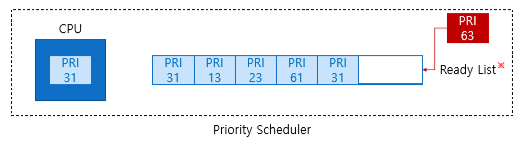
* **Priority scheduling**
* 기존의 pintos는 ready list의 맨 끝에 새로운 thread를 추가한다. 이는 thread의 prioirity를 고려하지 않는 방식이다. 따라서 이번 프로젝트에서는 각각의 thread가 priority를 갖도록 수정을 해야한다.
* 만약 priority가 낮은 thread가 ready list에 있다면, 이 thread는 우선순위가 낮기 때문에 실행되지 못할 수도 있다. 이를 위해선, priority aging을 실시해야 한다. Priority aging이란 ready list에 있는 thread의 priority를 시간이 지남에 따라 증가시키는 것을 말한다.
* Thread\_prior\_aging이 참이라면, 해당 thread의 priority를 증가시킨다.
* **BSD Scheduler**
* 위에서 구현한 priority-aging을 바탕으로 pintos manual을 활용해 BSD scheduler를 구현한다.
* 각각의 priority마다 자신의 ready queue를 가지고 있고, schedule()이 호출될 때, 가장 높은 priority queue의 thread가 선택된다. 각각의 ready queue는 round-robin 방식으로 scheduling된다.



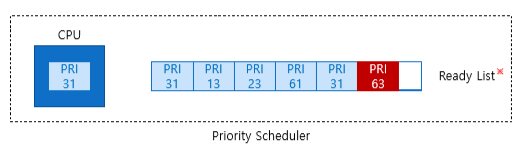
* 이를 위해선 niceness와 prioirity를 활용해 새로운 priority를 구해야한다. 구하는 방식은 다음과 같다.



* 1. **개발 내용**
* **Alarm clock**
* timer\_sleep()함수를 busy waiting 방식이 아닌 방식으로 수정한다. Wake 할 시간이 된 thread가 있는지 계속 확인한다. 만약 아니라면 해당 thread를 block 시켜 sleep 상태로 만들고, 만약 wake up할 시간이 된 thread가 있다면, 해당 thread에 interrupt를 발생시켜 깨워준다. 그 후 ready queue에 삽입하는 방법을 사용한다.
* **Priority scheduling**
* Thread는 64 level의 우선순위를 가진다. 우리는 이를 1개의 ready queue를 통해 구현할 것이다. Thread.c에 선언되어 있는 ready\_list라는 list를 활용한다.
* Thread.c의 함수들을 변형하여, 기존의 round robin 방식이 아닌, 우선순위에 따라 scheduling을 실시하도록 변형한다.



↓

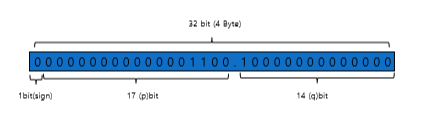


* **Priority aging**

priority = PRI\_MAX - (recent\_cpu / 4) - (nice \* 2)

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

* Aging을 구현하기 위해선 위에서 언급한 수식을 구해야한다. 이를 위해선 floting point가 아닌 fixed point 연산을 구현해야 한다.



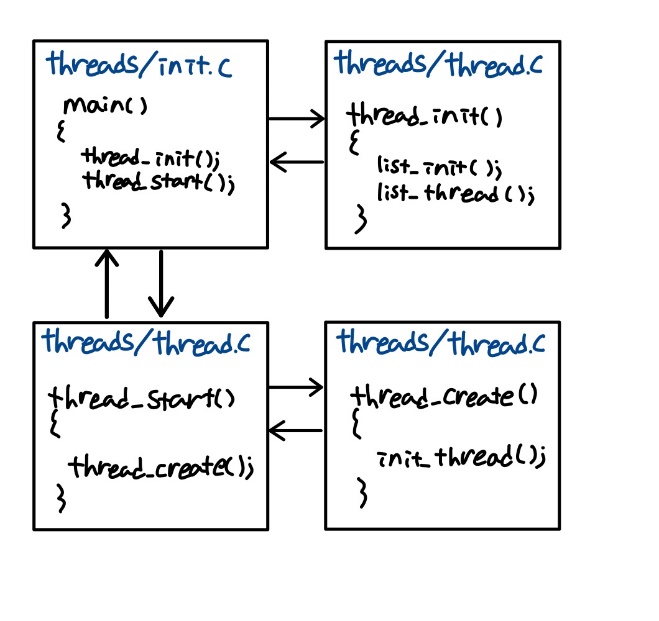
* Fixed point 형식이고, 계산결과가 실수로 나오는 것들은 모두 위의 형식으로 변환해야 한다. 이를 위해 thread.c에 함수들을 추가 구현한다. 또한, Aging check를 위해 threads/intit.c에 aging 플래그를 추가해준다.
* **BSD Scheduler**
* Mlfqs test를 통과하려면 thread.c의 여러 함수를 수정 혹은 구현 해야한다. Threads의 우선순위를 계속 구하며, page aging도 진행해준다. 또한, 일정 주기마다 우선순위를 재계산해, thread를 scheduling해야한다.
* 또한 niceness를 구하는 함수를 구현해야한다. 각각의 thread는 -20에서 20사이의 nice 값을 가지는데, 양수값은 우선순위가 낮고, 0은 우선순위에 영향을 끼치지 못한다.
* Thread\_get\_nice()와 thread\_set\_nice()함수를 구현해, 이 nice 값을 구한다..

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

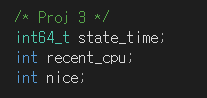
* 2019년 12월 4일 : 코드분석 및 구현해야 할 항목 파악
* 2019년 12월 5일 : 프로젝트 구현
* 2019년 12월 6일 : 추가구현
* 2019년 12월 7일 : 보고서 작성
  1. **개발 방법**
* **Alarm Clock**
* Threads/thread.h의 thread 구조체에 int64\_t state\_time(기상시간)을 선언한다.
* Devices/timer.c의 timer\_sleep함수를 수정한다. Interrupt를 끄고, thread를 실행한 후 다시 interrupt를 set하도록 수정한다.
* Devices/timer.h에 sleep\_thread 리스트 선언한다. 이 리스트는 timer.c의 timer\_init() 함수에서 초기화된다.
* Devices/timer.c의 timer\_interrupt() 함수를 수정한다. Wake up 할 시간이 된 thread를 찾아서 깨우는 함수이다. Sleep\_thread를 탐색하며, state\_time을 확인하고, wake up 시간이 되었으면, wake up 시킨다.
* **Priority scheduling**
* Threads/thread.c에 list이 element 두개를 비교해주는 priority\_less\_func()함수를 선언한다. 이 함수는 A가 B보다 작을 때 true를, 아닐 경우는 false를 return한다.
* Threads/thread.c의 thread\_yield() 함수를 수정한다. 현재의 thread를 다시 scheduling 하도록 한다.
* Threads/thread.c의 thread\_unblock() 함수를 수정한다. Block된 thread를 unblock시켜, ready to run state로 변환한다. 이 함수는 thread를 선점하지는 않는다.
* Threads/thread.c의 thread\_create()함수에, 생성된 thread의 우선순위가 현재 실행되고 있는 thread의 우선순위보다 크면 다시 스케줄링해야 하므로, thread\_yield()함수를 호출하는 부분을 추가한다.
* Threads/thread.c의 thread\_set\_priority() 함수를 수정한다. 현재 실행중인 thread의 우선순위는 제일 높으므로, 새로운 priority를 대입했을 때, 새로운 priority가 최대 priority보다 작아지면 재 scheduling시킨다.
* Threads/synch.c의 sema\_up()함수를 수정한다. Sema는 우선순위가 없어서 sema\_up()을 실행하는 순간 priority 정렬이 깨진다. 이를 수정하기 위해, 대기하고 있는 리스트에서 제일 priority가 큰 thread를 찾고 list elem과 thread를 업데이트한다. 그 후. thread\_yield()를 실행시켜, 재 scheduling을 한다.
* **Priority aging**
* Aging을 위해 필요한 실수 연산을 구현한다. Threads/thread.c에 i\_sub\_f(), i\_mul\_f(), f\_add\_i(), f\_mul\_f(), f\_div\_f(), f\_add\_f(), f\_sub\_f(), f\_div\_i()함수를 구현한다.
* Threads/thread.h의 thread 구조체에 recent\_cpu, nice를 선언해주고, 위에서 구현한 연산함수로 해당 값을 구한다.
* 우선 threads/thread.c의 thread\_init에 해당부분을 초기화하는 부분을 추가한다. init\_thread()함수 실행되고 있는 함수의 해당 값을 받아오는 값도 추가한다.
* 구현된 연산함수로, threads/thread.c의 thread\_get\_nice(), thread\_get\_load\_avg(), thread\_get\_recent\_cpu() 함수도 구현한다.
* threads/thread.c의 update\_load\_avg()는 ready state 일 때, update\_recent\_cpu()는 running state일 때를 나타내도록 수정한다.
* devices/timer.c의 time\_interrupt함수도 aging을 반영해서, interrupt하도록 수정한다.
* **BSD Scheduler**
* thread\_mlfqs가 참일 경우 thread\_set\_priority()가 수행되지 않아야 하므로 threads/thread.c의 thread\_set\_priority() 함수를 수정한다..
* threads/thread.c의 thread\_set\_nice() 함수를 구현해 준다. 현재 실행중인 thread의 nice 값을 set하는 함수이다.
* devices/timer.c의 timer\_interrupt()함수에 mlfq flag를 추가한다**.**
  1. **연구원 역할 분담**
* 이정원: 자료조사, 프로젝트 구현
* 최아연: 자료조사, 보고서 작성

1. **연구 결과**
   1. **합성 내용**

* **Thread flowchart**

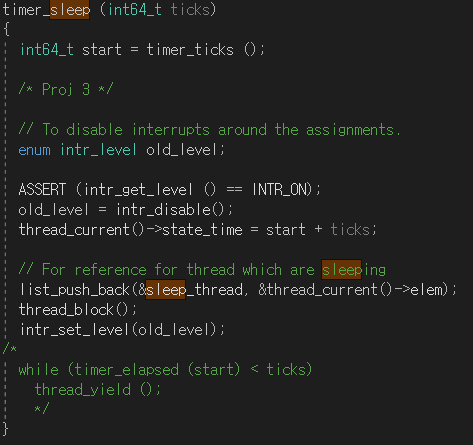


* 1. **제작 내용**
* **Alarm Clock**
* **thread 구조체**



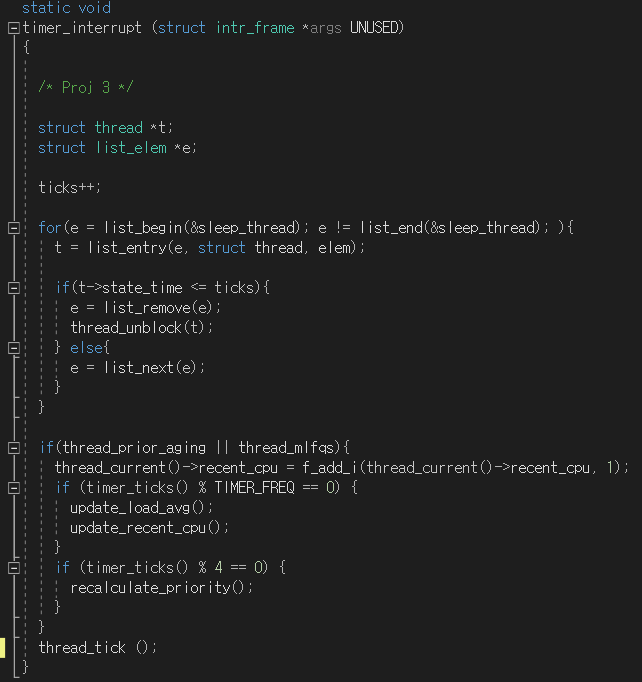
State\_time을 thread 구조체에 추가하였다. 이는 thread가 생성되고, thread\_ticks 함수가 호출될때마다 1씩 증가하고, state\_time이 어느 값을 초과하면 thread의 priority가 1씩 증가한다.

* **timer\_sleep()**

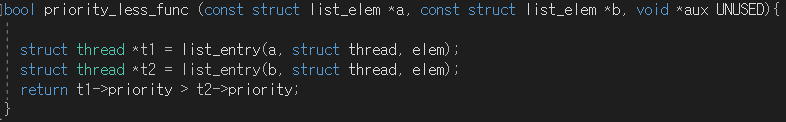
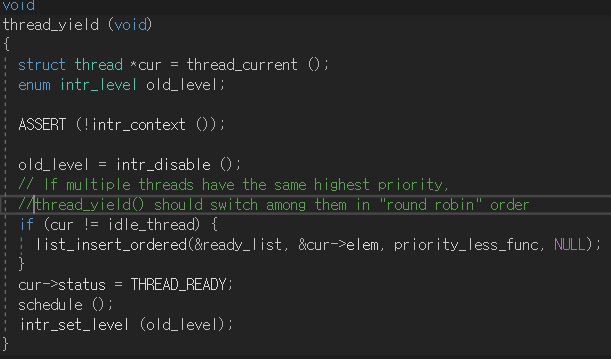


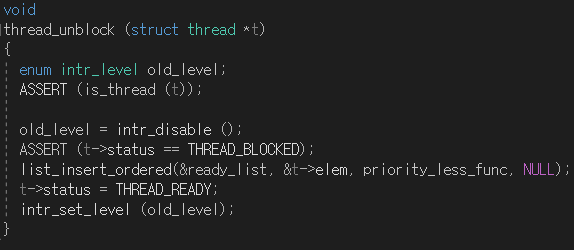
thread를 sleep queue에 포함시키고, block한다. 현재 thread가 ready queue로 이동해야 하는 시간 state\_time을 설정한 뒤, sleep\_thread에 추가하고 thread를 block한다.

* **timer\_interrupt()**

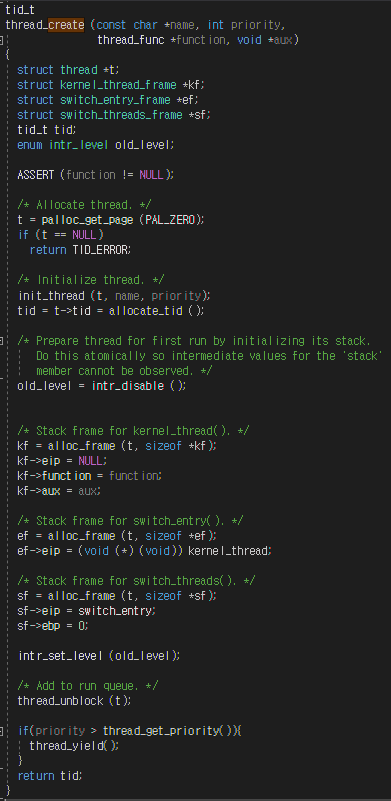
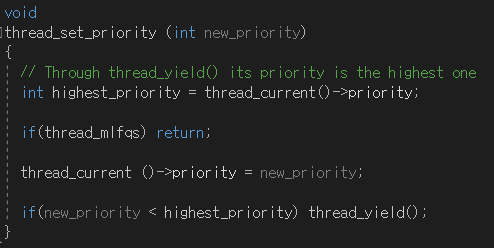
****

Sleep\_thread에 있는 thread를 unblock 시킨 다음 sleep\_thread에서 제거한다. state\_time이 현재 timer\_tick보다 크지 않은 경우엔 전부 sleep queue에서 제거해주고, unblock 시킨다. 또한 추가구현을 위해 if문에 mlfq flag 조건도 추가했다.

* **Priority scheduling**
* **priority\_less\_func()**a와 b의 크기를 비교해 Boolean 값으로 return한다.
* **thread\_yield()**thread를 ready queue에 삽입하고 schedule을 호출해 scheduling한다.
* **thread\_unblock()**

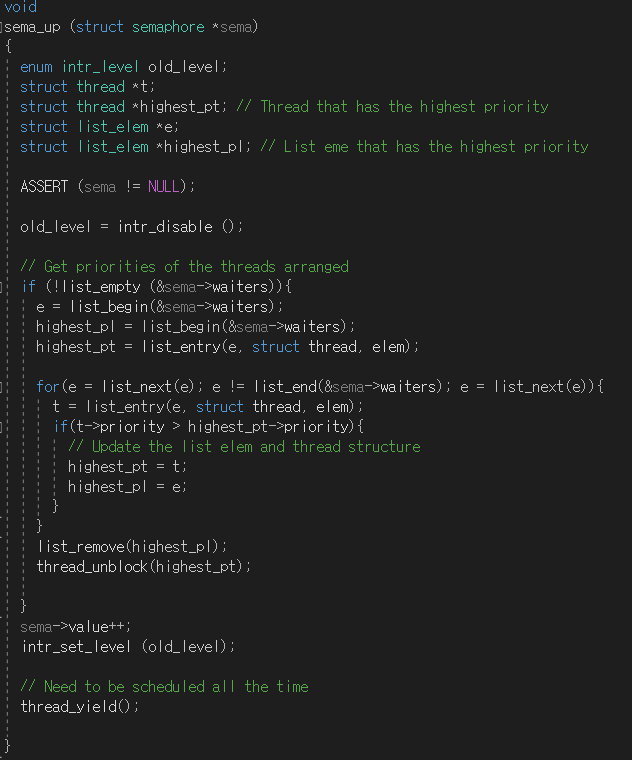


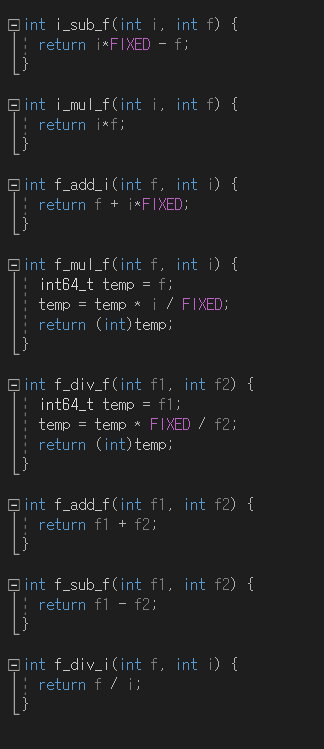
block상태의 thread를 unblock시켜 ready queue에 삽입한다. thread가 list\_push\_back의 형태로 삽입되는데, list\_insert\_ordered를 사용하여 priority scheduling을 실시한다.

* **thread\_create()**thread의 priority가 바뀌는 thread\_set\_priority() 함수와 thread가 생성되는 thread\_create() 함수 뒤에 thread\_yield 함수를 호출해 scheduling을 진행한다.   
  추가구현을 위해 thread\_mlfqs가 참일 경우 thread\_set\_priority()가 수행되지 않도록 했다.
* **thread\_set\_priority()**

priority가 바뀔 경우, thread\_yield() 함수를 호출해 scheduling을 실행한다.

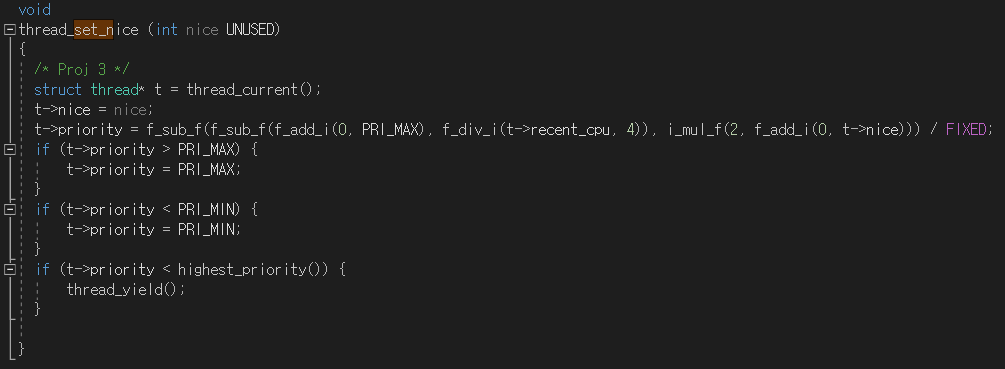
* **sema\_up()**

thread가 unblock됐을 때 sema의 값이 올라가지 않은 상황에서 schedule 함수를 호출하면, 해당 thread가 바로 block이 될 수 있다. 따라서 thread\_unblock()이 호출된 후, sema의 값을 증가시키고, thread\_yield를 호출하여 scheduling을 다시 실시한다.

* **Priority aging**
* **연산함수** 

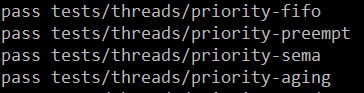
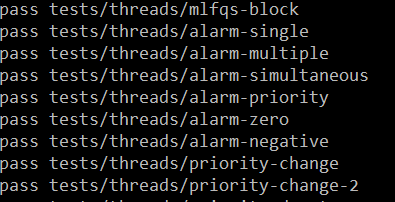
정수-실수, 정수\*실수, 실수+정수, 실수\*실수, 실수/실수, 실수+실수, 실수/실수, 실수/정수의 기능을 수행하는 함수들을 구현했다.

* **BSD Scheduler**
* **thread\_set\_nice()**

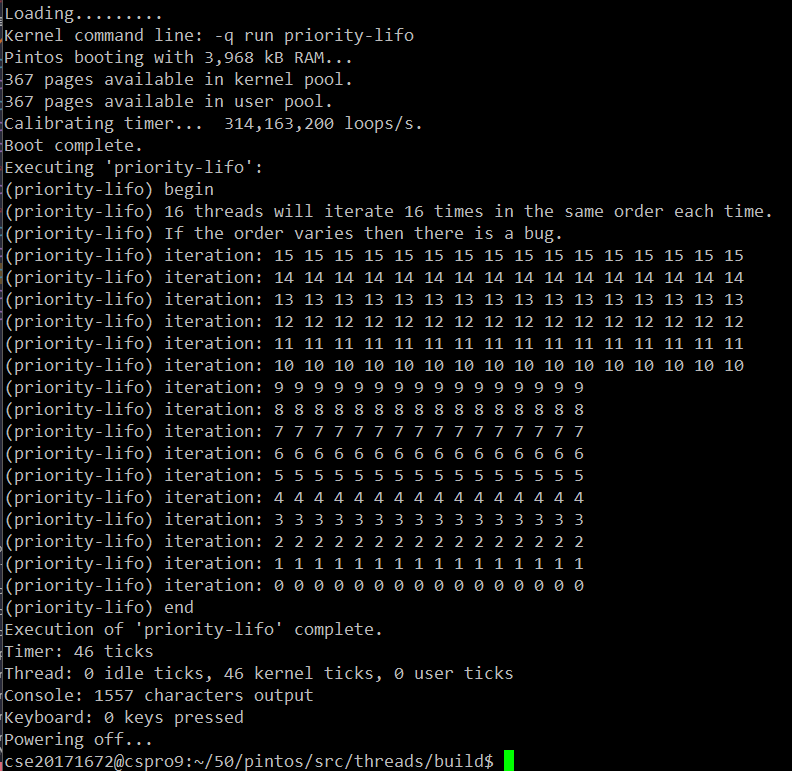


현재 실행중인 thread의 nice값을 설정한다. Nice 값을 구하기 위해 위에서 구현한 연산함수를 사용한다.

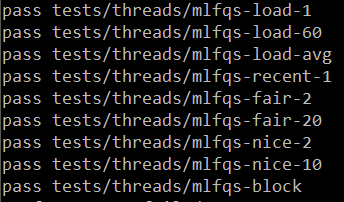
* 1. **시험 및 평가 내용**
* 12 tests (except priority-lifo)



* Priority-lifo



* Additional implementation



1. **기타**
   1. **연구 조원 기여도**

* 이정원: 50%
* 최아연: 50%
  1. **소감**
* Pintos 환경에서 thread와 관련된 project를 수행하며, 이전에 배웠던 내용을 다시 한 번 복습할 수 있었다. 중간고사 이후 약간 잊어버렸던 내용인데 다시 복습하며, 공부할 수 있어서 좋았다. 또한 직접 구현해봄으로서 scheduling 기법들을 체화할 수 있었다.
* 중간고사때 공부한 부분이라, 대체적으로 구현에 어려움은 없었다. 하지만, aging부분이나 중간중간 막히는 부분이 있었는데, 다른 팀과 함께 코드를 분석해보면서 문제를 해결할 수 있었다.
* 추가 구현항목을 구현하는데 많은 어려움이 있었다. MLFQ 기법을 이해하는데도 시간이 오래 소요되었고, 구현하는데도 많은 어려움이 있었다. 하지만 여러 자료들을 찾아보며, 문제점을 분석했고, 문제를 해결할 수 있었다.
* 또한 팀프로젝트였기 때문에, 가독성 좋은 코드를 짜는 것이 중요했다. 평소 코드를 가독성 좋지 않게 짜는 편이었는데, 이 기회를 통해 개발자에 맞는 코딩 스타일을 익히게 되었다.