**Pintos Project 3: Threads**

담당 교수 : 박 성용

학번 / 이름 : 20171665 / 이 선호

개발 기간 : 2022.10.29 ~ 2022.11.10

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

이전 pintos의 round robbin 방법의 thread의 스케줄링을 수정하여 우선순위에 따라 최우선으로 실행되어야 하는 thread를 ready queue에서 먼저 뽑아 실행할 수 있도록 구현한다. 또한 정해진 일정 시간동안 thread를 잠재우는 과정에서 thread가 ready와 running state를 오가는 비효율성을 수정하고, 추가적으로 MLFQ(Multi-Level Feedback Queue)와 thread의 starvation 문제를 보완하는 advanced scheduler를 구현하는 것을 목표로 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock

thread를 일정 시간동안 계속 재우는 역할을 하는 timer\_sleep 함수에서 기존에는 일정 시간에 도달할 때까지 매번 thread\_yield 함수를 호출하므로 thread가 ready state와 running state를 번갈아 오가서 비효율적이며, 이는 busy-waiting 방식의 일종으로 볼 수 있다. 그래서 일정 시간동안 잠재워야 하는 thread를 별도의 queue에 넣어서 정해진 시간이 지난 thread를 queue에서 뽑아 ready queue로 삽입하는 방식으로 개선할 필요성이 존재한다.

* 1. Priority Scheduling

기존에는 round robbin 방법처럼 가장 먼저 ready queue에 삽입된 thread가 우선적으로 ready queue에서 뽑히는 FIFO로 구현이 되어 있으나, 이러한 방식은 우선순위가 가장 높은 thread를 먼저 실행할 수 없다. 그래서 현재 실행 중이거나 block state에서 ready state로 전이되는 thread에 관해 ready queue의 맨 끝에 삽입하는 것보다는 우선순위를 고려하여 정렬된 상태에서의 적절한 위치에 삽입해야 한다.

또한 우선순위가 낮은 thread가 CPU 사용 권한을 얻지 못하는 starvation을 예방하기 위해 tick이 증가함에 따라 thread의 priority도 증가하는 priority aging을 구현한다.

* 1. Advanced Scheduler (추가구현을 한 경우)

각 우선순위마다 ready queue를 가지는 MLFQ를 사용하여 BSD scheduler를 구현하며, schedule 함수가 호출될 때 가장 높은 우선순위의 ready queue에서 FIFO 방식으로 thread를 뽑아 running state로 전이한다. 또한 마찬가지로 우선순위가 낮은 thread가 CPU 사용 권한을 얻지 못하는 starvation을 막기 위해 nice, recent\_cpu, load\_avg 세 가지의 상태값을 활용하여 우선순위를 일정 주기마다 계산한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

일정 시간 이후에 blocked 된 thread를 ready queue로 넣어야 하므로 정해진 시간만큼 blocked state인 thread를 잠재워야 하는데, 앞서 언급한 바처럼 busy waiting 방식은 비효율적이므로 잠재워야 하는 state를 따로 저장할 수 있는 sleep queue를 새로 만들어서 thread마다 정해진 시간이 지났을 때 해당 queue에서 thread를 뽑아 ready queue로 넣을 수 있다. 언제 어떠한 thread를 깨워야 하는지 알아야 하므로 sleep queue에 thread를 넣을 때 각 thread 마다 언제까지 잠재워야 하는지 그 데이터를 같이 저장한다. 또한 timer\_wakeup 함수를 추가로 구현해서 timer\_interrupt에 의해 매 tick 마다 timer\_wakeup 함수를 호출하여 현재 tick 시간이 thread가 잠재워져 있어야 하는 시간보다 크거나 같으면 thread를 sleep queue에서 제거하여 ready queue로 삽입한다.

1. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.

기존에는 ready queue의 끝에 thread를 삽입해서 ready queue에서 실행할 thread를 뽑을 때 queue의 앞에 있는 thread를 뽑았지만, priority scheduling을 위해 우선순위가 가장 큰 thread를 먼저 실행해야 하므로 thread를 ready queue에 삽입할 때 queue에서 기다리고 있는 thread가 정렬되어 있는 상태에서 적절한 위치에 삽입되어야 한다. 그래서 list\_insert\_ordered 함수를 이용해 thread 삽입 시 우선순위가 낮아지는 순으로 ready queue의 원소를 정렬하며, thread를 우선순위에 따라 정렬 시 직접 구현한 thread\_compare\_priority 함수를 사용한다.

1. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)

BSD scheduler에서 priority를 다시 계산할 때 nice, recent\_cpu, load\_avg 세 가지의 요소를 사용한다.

nice는 -20 이상 20 이하의 범위에서 값을 갖으며, thread의 nice 값이 높을수록 thread의 우선순위를 낮춰서 다른 thread가 CPU를 점유할 수 있도록 한다. Thread의 nice 값이 변경되면 thread의 우선순위도 다시 계산된다.

recent\_cpu는 해당 thread가 CPU를 얼마나 사용했는지를 나타내는 값이다. Thread가 running 상태일 때는 timer\_interrupt 발생 시, 즉 1 tick이 지날 때마다 값이 1씩 증가한다. 한 thread가 CPU를 이전에 많이 사용했을수록 해당 thread의 우선순위는 낮아지게 되며, 이는 마찬가지로 다른 우선순위가 낮은 thread의 starvation 현상을 막기 위한 것을 해석할 수 있다. 또한 running state에서 1 tick 뿐만이 아니라 모든 state에서 1초가 지날 때마다 recent\_cpu가 다시 계산되어야 하는데, 그 식은 아래와 같다.

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

load\_avg는 전역변수로 관리되는데, 지난 시간까지 ready state였던 thread 개수의 평균을 뜻한다. 처음에는 0으로 초기화되어 있지만 매 초가 지날 때마다 다음 식에 의해 값이 업데이트된다.

load\_avg = (59/60) \* load\_avg + (1/60) \* ready\_threads

최종적으로 각 thread의 우선순위인 priority는 0(PRI\_MIN)에서 63(PRI\_MAX)까지의 값을 지닐 수 있으며,매 4 tick 마다 다음과 같은 식에 의해 재계산된다.

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

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2022. 10. 28 ~ 2022. 10. 29 : alarm clock 구현

2022. 10. 30 ~ 2022. 11. 03 : priority scheduling 구현

2022. 11. 04 ~ 2022. 11. 06 : BSD scheduler 구현

2022. 11. 07 ~ 2022. 11. 10 : 보고서 작성 및 에러 수정

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수

Alarm clock에서 blocked state인 thread를 일정 시간이 지난 후에 깨우는 작업을 처리하기 위해 기존의 busy waiting 방식의 코드는 지우고 thread 마다 잠자야 하는 시간을 저장할 수 있도록 thread 구조체에 wakeup\_tick이라는 멤버 변수를 추가한다. 또한 잠재워야 하는 blocked state인 thread만 따로 관리할 수 있는 sleep queue를 추가한다. timer\_interrupt 함수에서 매 tick 마다 정해진 시간이 지난 thread를 깨워주기 위해 sleep queue를 탐색하면서 현재 시간과 각 thread의 시작 시간과 wakeup\_tick 시간의 합을 비교하여 sleep queue에서 제거하고 ready queue로 옮겨야 하는 thread를 탐색한다. 이를 이번 프로젝트에서는 timer\_wakeup이라는 함수를 구현하여 호출하는 방식을 택했다.

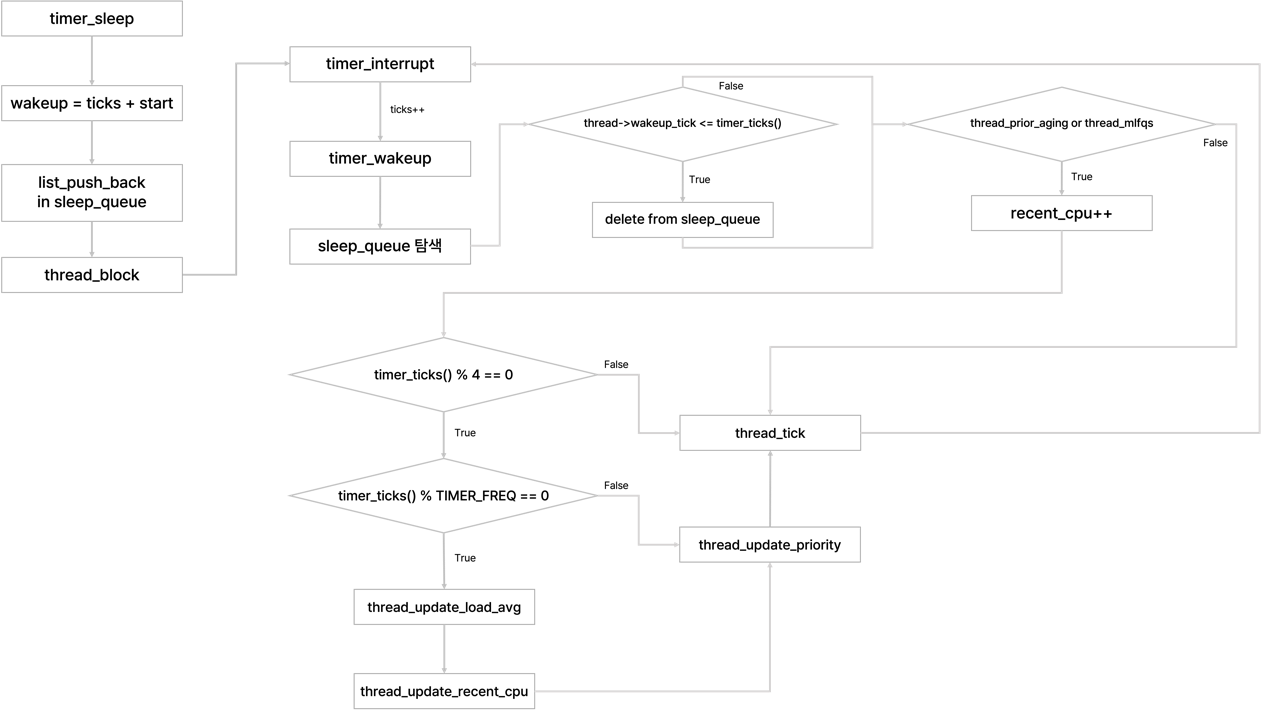
Priority scheduling에 의해 가장 우선순위가 높은 thread가 ready queue에서 먼저 뽑혀서 running state로 전이될 수 있도록 thread의 우선순위 정렬이 이루어질 수 있도록 하고자 thread\_compare\_priority 함수를 추가했다. 이를 가지고 thread가 우선순위에 밀리거나 정해진 실행 시간을 초과하여 CPU 권한을 포기할 때 ready\_list에 thread에 삽입되는데, 이때 기존의 pintos에 존재하는 list\_insert\_ordered 함수를 사용하여 ready\_list에 있는 원소들의 정렬이 보존되도록 했다. 마찬가지로 thread가 blocked state에서 ready state로 전이되면서 ready\_queue에 원소의 정렬이 유지되도록 list\_inser\_ordered 함수를 사용했다. 또한 semaphore를 사용하여 synchronization이 이루어지는 sema\_down과 sema\_up에서도 가장 우선순위가 높은 thread가 대기 중인 sema\_list에서 먼저 실행될 수 있도록 우선순위 정렬에 의해 삽입되거나 제거된다. 설정하고자 하는 priority 값을 인자로 받아서 새로운 priority 값이 기존 thread의 priority보다 작으면 CPU 사용 권한을 내려놓고 다시 스케줄링되기를 기다린다. Thread를 생성하는 thread\_create 함수에서는 현재 실행 중인 thread의 우선순위와 새롭게 생성되는 thread의 우선순위를 비교하여 새로 생성되는 thread의 우선순위가 더 크면 기존에 실행하고 있는 thread는 CPU 권한을 포기하고 새로운 thread가 CPU를 점유할 수 있도록 한다.

BSD scheduler 구현을 위해 thread 구조체에서 nice, recent\_cpu 멤버 변수를 추가하고, thread.c에 전역변수로 load\_avg를 선언했다. 매 1초마다 load\_avg와 recent\_cpu가 업데이트되도록 thread\_update\_load\_avg, thread\_update\_recent\_cpu 함수를 구현하고, 매 4 tick마다 thread의 우선순위가 업데이트되도록 thread\_update\_priority 함수를 구현하여 위에서 정리한 식에 의해 nice, recent\_cpu, 그리고 priority가 업데이트 될 수 있도록 했다. 단, pintos에서는 기본적으로 floating point 연산을 지원하지 않아서 소수인 값에 관해 별도의 함수를 통해 처리를 해줘야 하는데, 이를 fixed-point.c에 int\_to\_float, float\_to\_int, float\_to\_int\_round, float\_add, float\_add\_int, float\_subtract, float\_subtract\_int, float\_multiply, float\_multiply\_int, float\_divide, float\_divide\_int 함수로 구현했다.

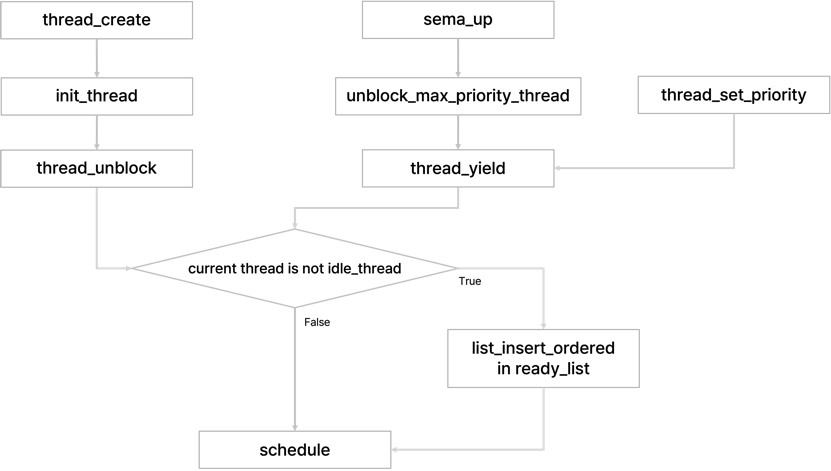
1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)

1) Alarm Clock과 Thread Priority를 사용하는 Advanced Scheduler (BSD Scheduler)

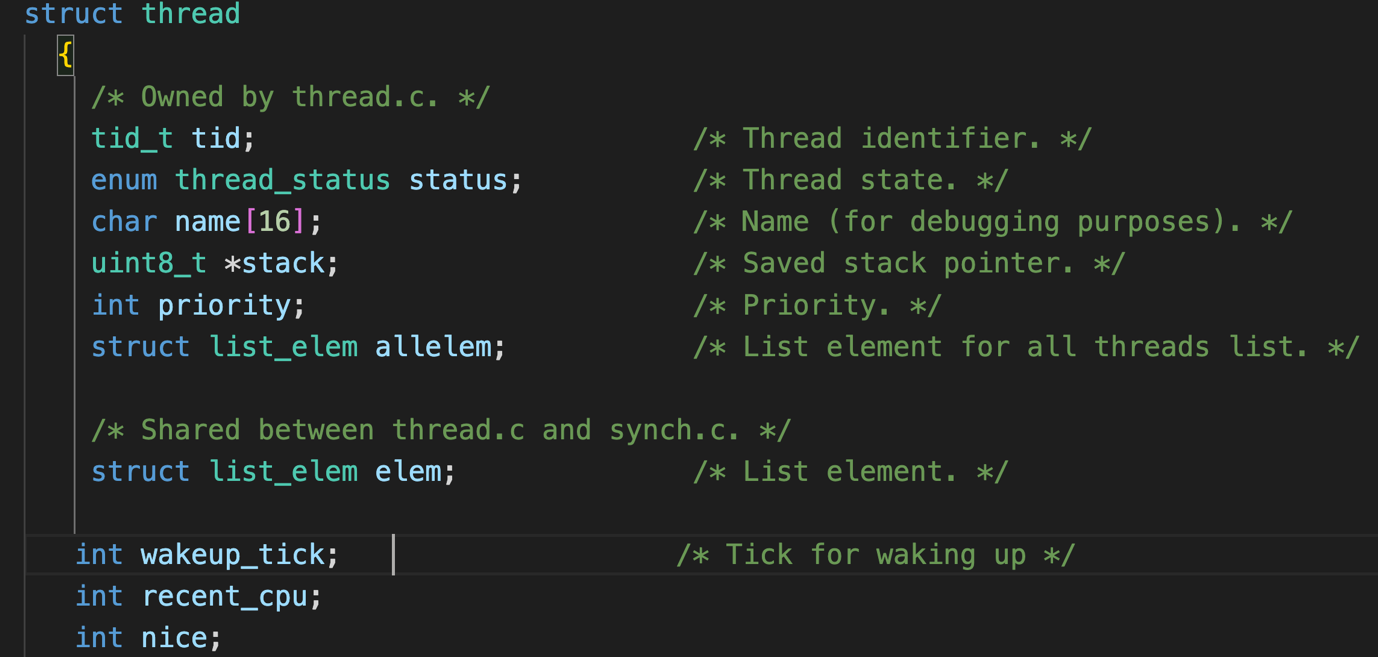


2) Priority Scheduling와 Synchorinaztion



* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

(1) threads/thread.h



thread의 구조체에 alarm clock에 필요한 wakeup\_tick과 BSD scheduler 구현을 위한 recent\_cpu, nice 멤버 변수를 추가로 선언한다. wakeup\_tick은 blocked 상태인 thread를 sleep\_queue에 얼마나 잠재워야 하는지 그 시간을 의미한다. recent\_cpu와 nice는 thread의 우선순위를 조절하는 역할을 하며, 둘 다 값이 커지면 thread의 우선순위를 낮아지게 한다.

(2) devices/timer.c



텍스트이(가) 표시된 사진

자동 생성된 설명

Alarm clock을 위한 sleep\_queue를 전역 변수로 선언하고, timer\_init 함수에 sleep\_queue list를 초기화하고자 list\_init을 사용했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

timer\_sleep 함수는 thread를 sleep\_queue에 넣어서 현재 시간부터 ticks 만큼 잠재우기 위한 역할을 한다. 현재 시간을 불러오는 timer\_ticks 함수로 구해 start 변수에 업데이트하고, 매개변수로 온 ticks 시간만큼 start에 더하여 현재 thread인 current\_thread의 멤버 변수인 wakeup\_tick에 업데이트 한다. list\_push\_back을 사용하여 sleep\_queue에 current\_thread를 넣어주고, thread\_block을 통해 thread를 block 상태로 만든다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

timer\_interrupt에서 매 tick마다 timer\_wakeup 함수로 현재까지 sleep\_queue에서 잠재워져 있는 thread 중에서 시간이 지나 깨워야 할 thread를 깨우는데, sleep\_queue에서 탐색하면서 thread의 wakeup\_tick보다 현재 시간이 크거나 같으면 sleep queue에서 해당 thread를 제외하고 unblock해서 ready 상태로 변경해준다. thread\_prior\_aging 또는 thread\_mlfqs이 true이면 recent\_cpu에 1을 더해준다. 주기가 1초가 된 시점이면 load\_avg와 recent\_cpu를 업데이트하고, 별개로 4 tick이 지난 시점이면 thread의 priority를 업데이트한다.

(3) synch.c

텍스트이(가) 표시된 사진

자동 생성된 설명

thread의 synchronization을 위한 semaphore에서 sema\_down을 할 시 가장 우선순위가 높은 순서대로 정렬되어 semaphore를 위해 기다리는 waiters list의 원소를 정렬해줘야 하므로 list\_push\_back 대신 list\_insert\_ordered와 후술할 thread\_compare\_priority 함수를 사용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

sema\_up을 할 때도 마찬가지로 semaphore를 위해 기다리는 thread가 waiters list에 존재할 경우, unblock\_max\_priority\_thread 함수를 이용해 waiter list에서 가장 우선순위가 높은 thread를 block 상태에서 ready 상태로 변경한 후 현재 CPU를 점유하는 thread의 CPU 사용 권한을 thread\_yield 함수를 통해 포기한다.

(4) thread.c

텍스트이(가) 표시된 사진

자동 생성된 설명

thread\_unblock 함수에서는 thread를 blocked 상태에서 ready 상태로 전이하면서 thread를 ready\_list에 넣을 때 우선순위가 높은 순으로 정렬됨을 보장하면서 thread가 삽입될 수 있도록 list\_insert\_ordered 함수를 사용했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

thread\_yield 함수에서는 현재 실행 중인 thread가 CPU 사용 권한을 포기하고 running state에서 ready state로 전이되어야 하므로 list\_insert\_ordered 함수를 사용하여 현재 thread를 ready\_list에 넣어주고 새로 scheduling을 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

list\_insert\_ordered를 사용하여 list를 정렬할 때 thread\_compare\_priority 함수를 사용하는데, 앞쪽의 원소가 뒤쪽의 원소보다 우선순위가 더 크면 true를, 그렇지 않으면 false를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

thread\_update\_priority에서는 ready\_list 등 모든 list에 있는 원소들을 while loop로 탐색하면서 priority를 계산하여 업데이트한다. priority 계산 값이 최대 threshold인 PRI\_MAX를 넘으면 PRI\_MAX로 설정하고, 최소 threshold인 PRI\_MIN보다 작으면 PRI\_MIN으로 설정한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

thread\_set\_nice 함수에서는 매개변수로 온 nice 값으로 현재 thread의 nice를 설정하고, 새로운 nice 값에 의해 priority 값을 계산하여 업데이트 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

thread\_get\_load\_avg에서는 floating 연산에 의해 load\_avg에 100을 곱한 값을 반환하고, thread\_update\_load\_avg에서는 현재 실행 중인 thread가 idle하지 않으면 ready\_list에 있는 thread의 개수를 세서 앞서 언급한 load\_avg 계산 식으로 load\_avg 값을 업데이트 한다. 만약 load\_avg가 0보다 작으면 load\_avg를 0으로 업데이트한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

thread\_get\_recent\_cpu에서는 현재 thread의 recent\_cpu에 100을 곱해서 floating point 연산 결과를 반환한다. thread\_update\_recent\_cpu에서는 ready list 등 모든 list에 존재하는 thread 원소를 탐색하면서 idle 상태의 thread가 아니면 recent\_cpu를 위에서 언급한 식에 의해 floating point 연산으로 업데이트 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Pintos에서는 floating 연산이 구현되어 있지 않기 때문에 직접 구현해야 하는데, Pintos 매뉴얼에서 구현된 안내 사항에 따라서 floating 사칙 연산을 수행할 수 있는 함수들을 정의했다. 구체적으로 실수 + 실수, 실수 + 정수, 실수 – 실수, 실수 – 정수, 실수 \* 정수, 실수 \* 실수, 실수 / 정수, 실수 / 실수의 기능을 구현했다.

* 1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석
* make check 수행 결과를 캡처하여 첨부

(1) priority-lifo

priority-lifo.c에서는 0부터 15까지의 우선순위를 지닌 thread를 여러 개 생성하면, ready\_list에서는 우선순위가 높은 순으로 정렬되어 있으므로 priority가 높은 thread가 list에 나중에 삽입되어도 먼저 뽑혀서 실행이 되도록 해야 정상적이라고 체크하는 코드이며, last-in first-out 순으로 잘 실행되는지 확인하는 역할을 한다고 볼 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

priority가 높은 15의 thread가 나중에 ready\_list로 삽입되더라도 가장 먼저 ready\_list에서 뽑혀 실행되고 종료되었다.

(2) make check 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

프로젝트 3에서 주어진 채점 케이스는 모두 다 통과한 것을 볼 수 있다.