**Pintos Project 3: Threads**

담당 교수 :

학번 / 이름 : 20171664 / 이상윤

개발 기간 : 2022-11-06 ~ 2022-11-11

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

thread와 관련된 기능을 구현한다. alarm clock, priority scheduling, MLFQ scheduler 을 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

핀토스는 timer\_sleep() 함수를 통해 스레드를 sleep시킨다. 이 함수를 호출한 스레드는 while문 내부로 진입하고 계속해서 thread\_yield() 함수를 호출하여 계속해서 ready-running state를 반복하고, 이는 비효율적이다. 이를 ready-running state를 계속해서 반복하는 것이 아닌 시간이 다되면 깨우도록 한다.

* 1. Priority Scheduling

핀토스는 기존에 FIFO를 이용한 round-robin 스케줄링을 사용한다. 따라서 우선순위에 관계없이 ready\_list에 먼저 들어온 프로그램이 먼저 실행된다. 이를 우선순위가 높은 작업은 먼저 실행될 수 있도록 한다. 이 때, priority aging도 구현하여 starvation이 일어나지 않도록 한다.

* 1. Advanced Scheduler (추가구현을 한 경우)

Multi-Level-Feedback-Queue(MLFQ) scheduler를 구현한다. priority가 가장 높은 queue에서 스레드가 선택될 수 있다. 이때, priority는 여러가지 요소에 따라 aging되도록 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

timer\_sleep()을 통해 blocked 된 상태의 스레드들은 별도의 리스트에 따로 저장한다. 그리고 대기중인 스레드들 중 깨어나야할 시간이 가장 이른 시간 또한 따로 저장한다. 매 tick 마다 가장 이른 시간이 되었는지 검사하고, 가장 이른 시간이 지났다면 리스트에가서 깨워야할 스레드를 깨운다.

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

ready list에 높은 priority를 가진 스레드가 들어온다면 priority가 높은 스레드가 실행되야한다. Running thread는 ready list에 들어가고, ready list에 가장 높은 priority를 가진 스레드가 실행된다.

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

* load\_avg : 시스템 당 하나만 존재한다. 1분동안 수행가능한 스레드의 평균 개수이다. 크면 클수록 priority는 천천히 증가한다. 매초마다 새로 계산된다. (loag\_avg = (59/60)\*load\_avg + (1/60)\*read\_threads)
* recent\_cpu : thread 당 고유한 값이다. 최근에 얼마나 많은 cpu time을 사용했는가를 의미한다. 클수록 priority는 낮아진다. 매 1 tick마다 기본적으로 1이 올라가고, 매초마다 새로 계산된다.(recent\_cpu = (2\*load\_avg)/(2\*load\_avg + 1)\*recent\_cpu + nice)
* nice : thread 당 고유한 값이다. nice가 클수록 다른 스레드에게 양보하는 정도가 크다. 클수록 priority는 낮아진다.

Priority = PRI\_MAX - (recent\_cpu/4)-(nice\*2) 로 계산 가능하다. 4 tick 마다 새로 계산된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2022-11-06 ~ 2022-11-06 alarm clock 구현

2022-11-07 ~ 2022-11-07 priority scheduling 구현

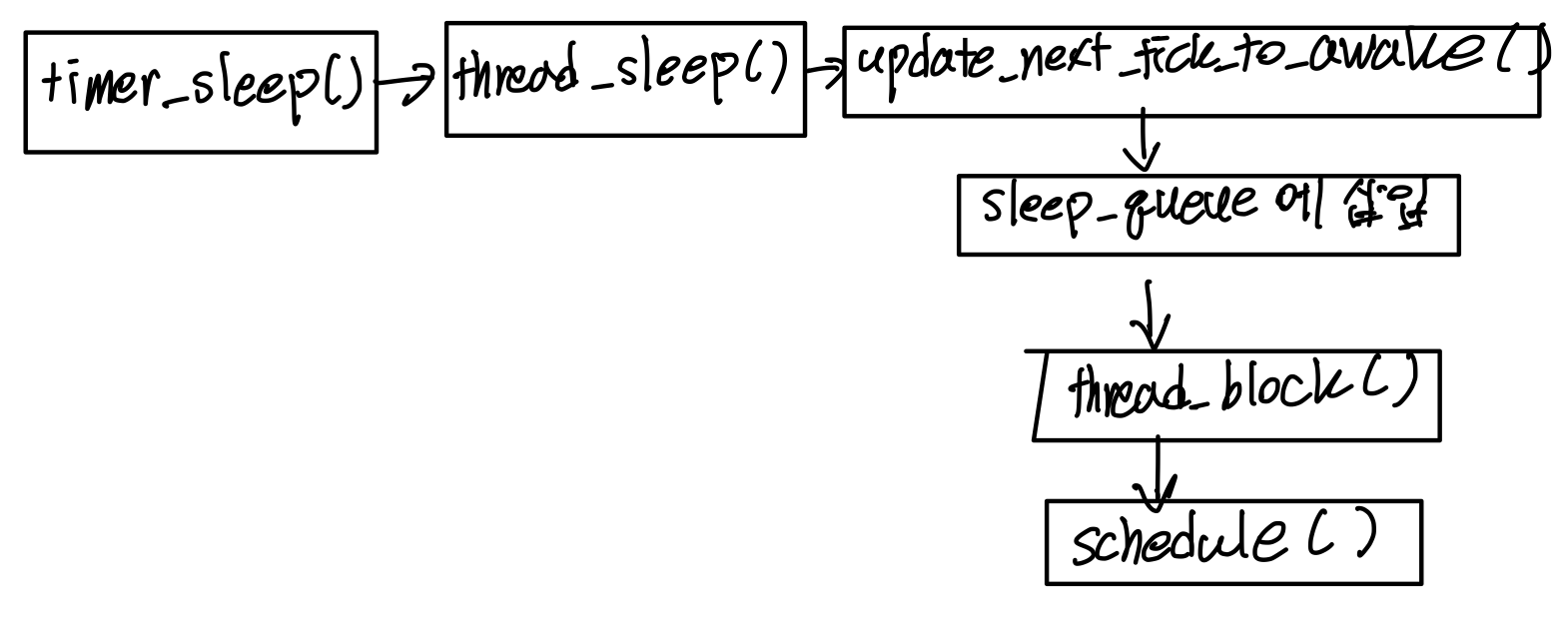
2022-11-08 ~ 2022-11-09 priority scheduling aging 구현

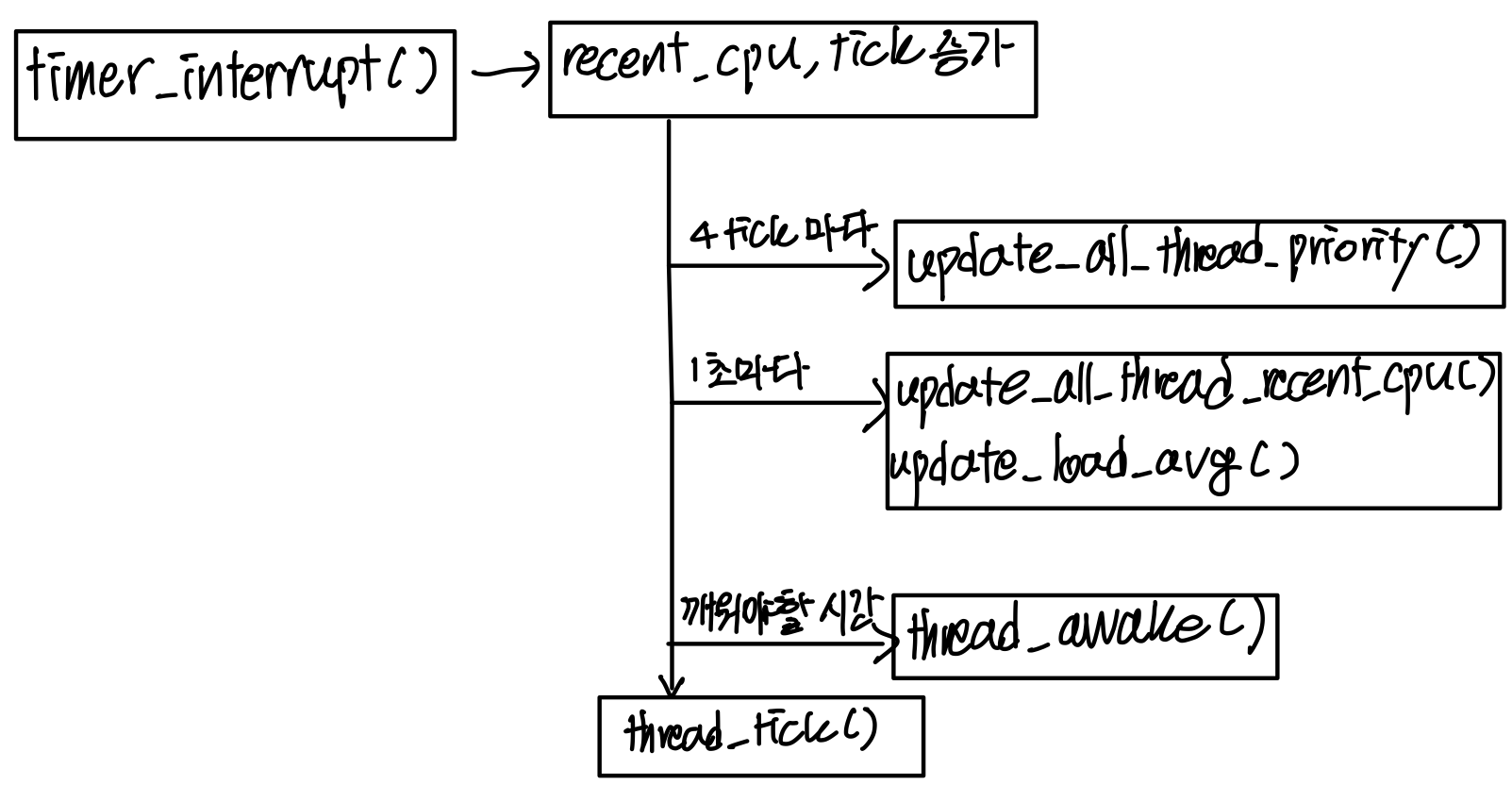
2022-11-10 ~ 2022-11-10 mlfq, 버그 수정

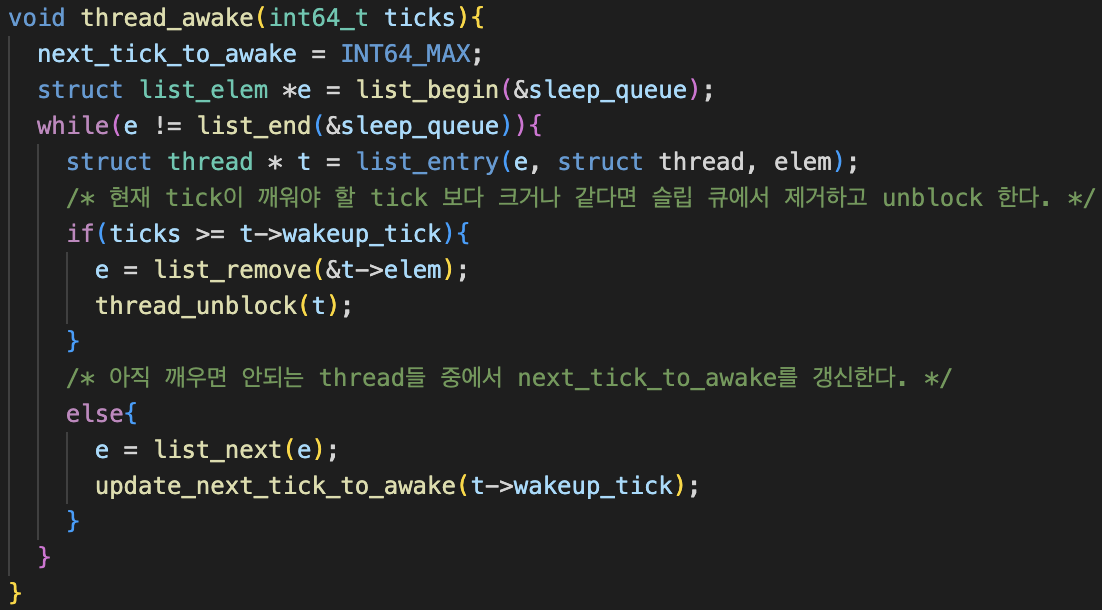
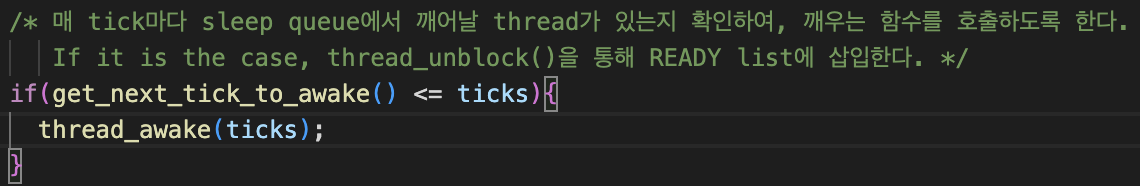
2022-11-11 ~ 2022-11-11 보고서 작성

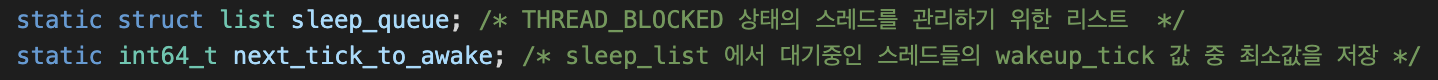
* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수

1. Alarm clock을 구현하기 위해 timer.c 의 timer\_sleep(), timer\_interrupt() 함수를 수정해야 한다. thread.h에 timer\_sleep()으로 THREAD\_BLOCKED 된 상태의 스레드를 관리하기 위한 리스트를 선언하고, 깨워야할 가장 이른 시간을 저장하는 변수를 선언하고, thread 구조체에 는 깨어나야할 tick을 저장하게끔 한다. 그리고 thread.c에 실행중인 스레드를 슬립으로 만드는 함수, 슬립된 스레드를 깨우는 함수, 깨워야할 가장 이른 시간을 저장하는 변수를 업데이트하는 함수를 작성한다.
2. Priority scheduling을 구현하기 위해 synch.c 의 세마포어 조작함수를 수정해야한다. 핀토스 메뉴얼에 따르면, 세마포어를 기다리는 가장 priority가 높은 스레드가 먼저 unblocked 되야하기 때문이다. thread.c의 thread\_create(), thread\_unblock(), thread\_yield(), thread\_set\_priority()를 수정한다.
3. Priority scheduling aging을 구현하기 위해 thread.h에 thread\_prior\_aging 변수를 선언하고, thread 구조체에 recent\_cpu, nice 를 저장하도록 한다. 그리고 thread.c에 load\_avg 변수를 선언하고, thread\_init()에서는 thread 구조체에 추가한 변수들을 초기화 하도록 수정하고, priority를 계산하기 위한 여러 함수를 선언한다. 이때 fixed-point 연산을 하기 위해 fixed-point.c, fixed-point.h를 작성하고 Makefile.build에 이를 추가한다.
4. BDS scheduler를 구현하기 위해 thread\_mlfqs 가 true일 때, thread\_set\_priority()가 실행되지 않도록 한다.
5. **연구 결과**
   1. **Flow Chart**

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



* 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

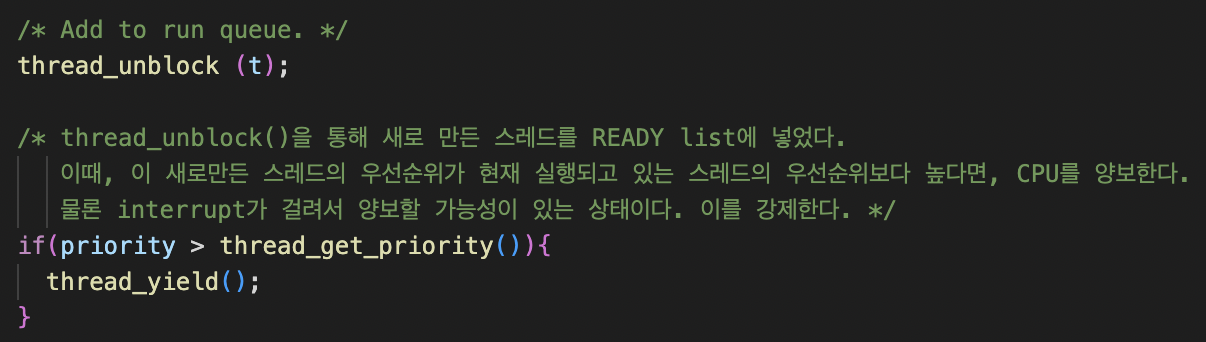
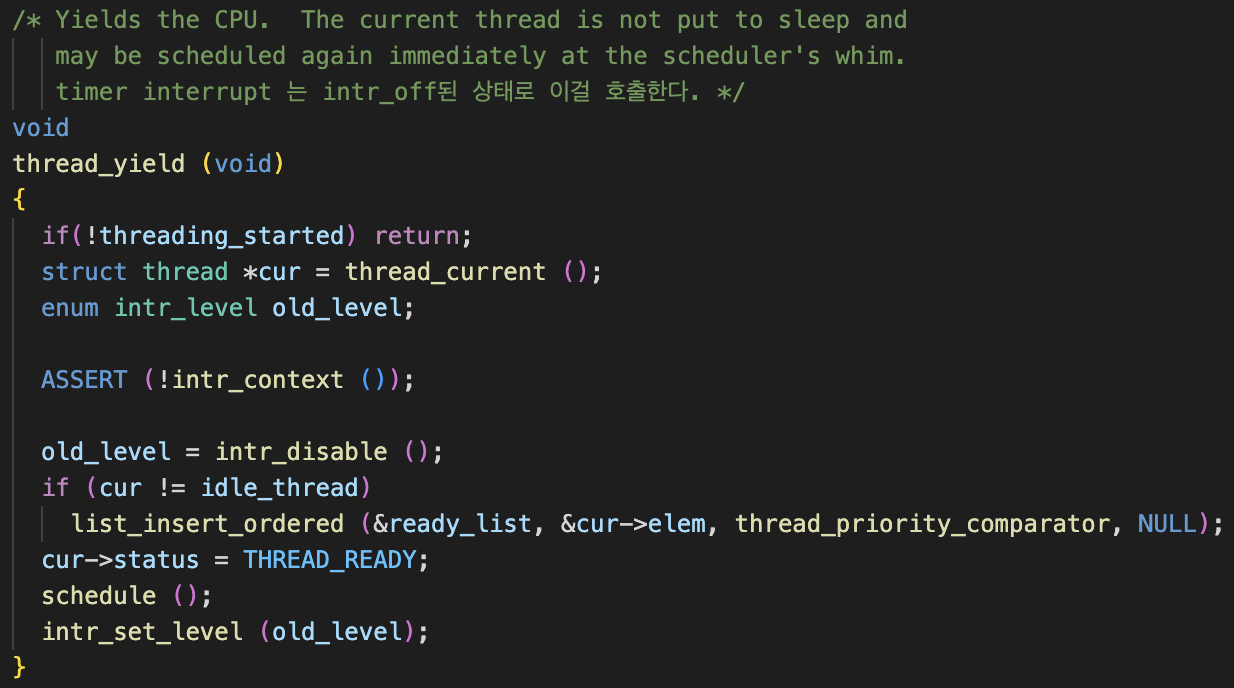
1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술

thread.h에 전역변수로 선언한다. 이는 thread\_init() 에서 초기화된다.

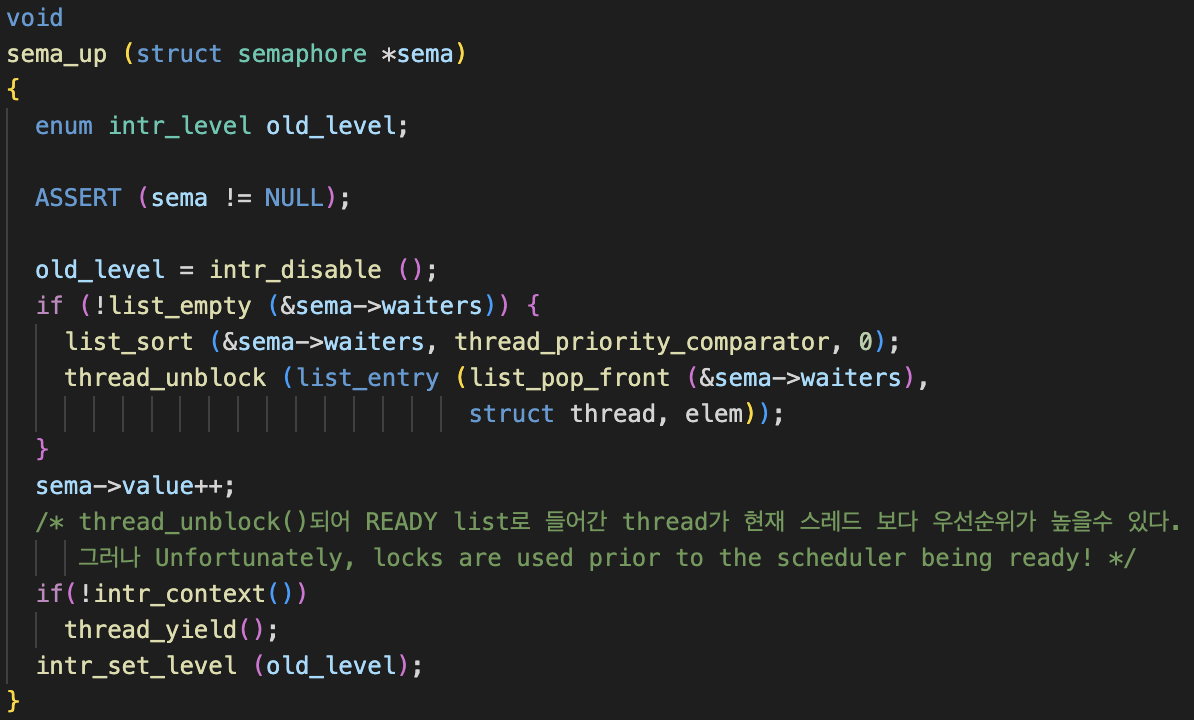
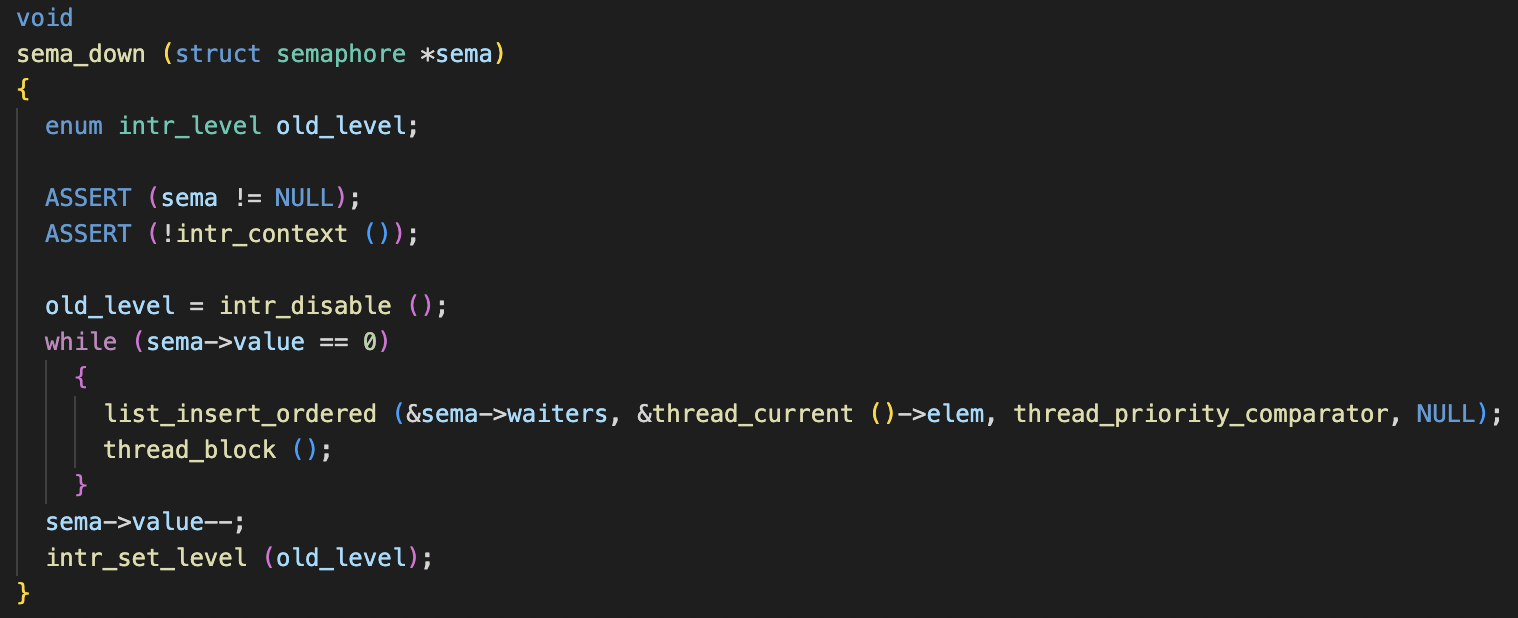
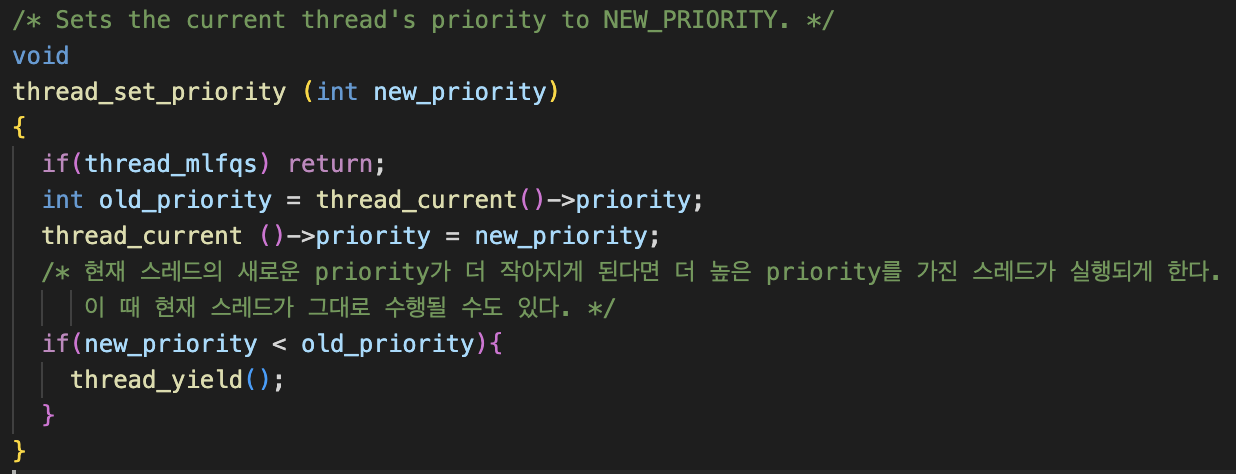
timer\_interrupt() 내부이다. 주석에 서술된 대로 매 tick 마다 깨울 스레드가 있는지 확인한다. 깨울 스레드가 존재한다면 thread\_awake()를 호출한다.

thread\_awake() 의 내부는 위와 같다. sleep\_queue를 순회하면서 next\_tick\_to\_awake를 갱신함과 동시에 깨워야할 스레드를 찾아서 깨운다.

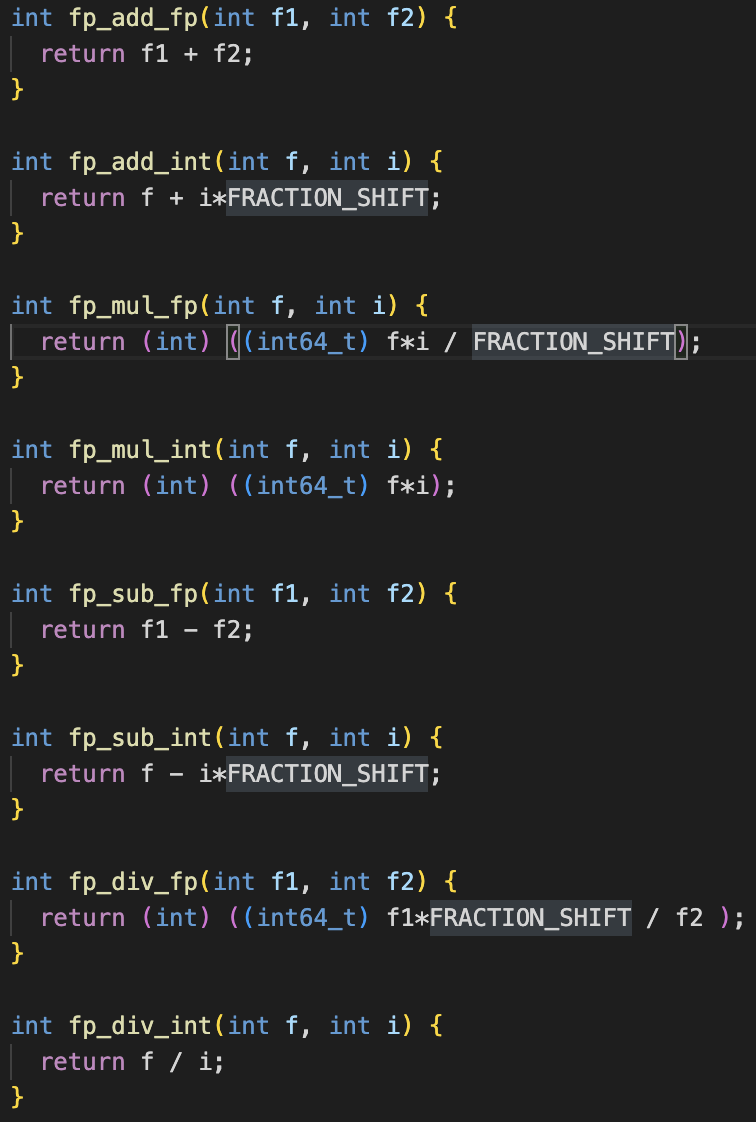
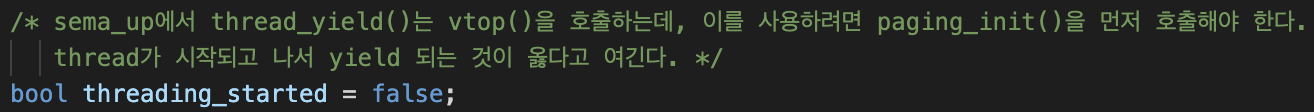
1. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.

기본적으로 intr\_handler()에서 외부 인터럽트인 경우 thread\_yield()를 호출한다. ready\_list에 내림차순으로 정렬하면서 삽입하는 것을 볼 수 있다. 그리고 schedule() 함수를 호출함을 통해 context switch를 수행한다.

우선 thread\_create() 함수의 내부이다. 이때 ready list에 running thread 보다 높은 priority를 가진 thread가 들어갈 수 있다. 이 때 새로만든 스레드의 우선순위가 현재 실행되고 있는 우선순위 보다 높다면 CPU를 양보한다.

Thread.c의 함수들이다. 이 또한 ready list에 지금 돌아가는 스레드보다 더 높은 priority가 있는 경우라고 볼 수 있다.

다음은 semaphore 조작 함수이다. sema\_down() 시 thread\_block()으로 ready list에서 해당 스레드는 사라진다. sema\_up을 통해 thread\_unblock() 시킬때, 핀토스 메뉴얼(2.3.2 priority scheduling FAQ)에 의해 높은 priority가 먼저 unlock 되도록 해야한다. 따라서 sema->waiters에 priority에 대해 내림차순으로 정렬하여 삽입하는 것을 볼 수 있다. 그리고 sema\_up()의 마지막에 thread\_yield()를 통해 ready list로 들어간 thread가 현재 스레드보다 우선순위가 높은 경우를 예외처리한다.

이때 개발중에 문제가 발생하였다. thread\_yield()는 vtop()을 호출하는데, 이를 사용하려면 paging\_init()을 먼저 호출해야 한다. 그러나 semaphore 함수는 thread\_init() 전에도, paging\_init() 전에도 사용된다. 따라서 thread가 시작되고 나서(=thread\_start()가 호출되고 나서) thread\_yield 되는 것이 옳다고 여긴다.

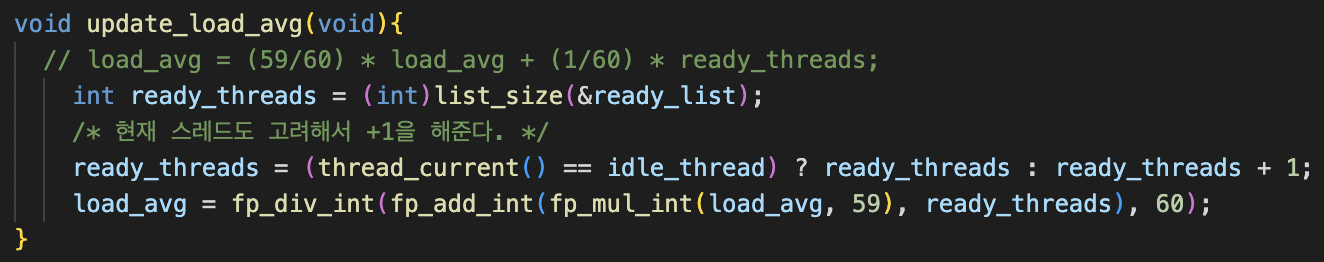
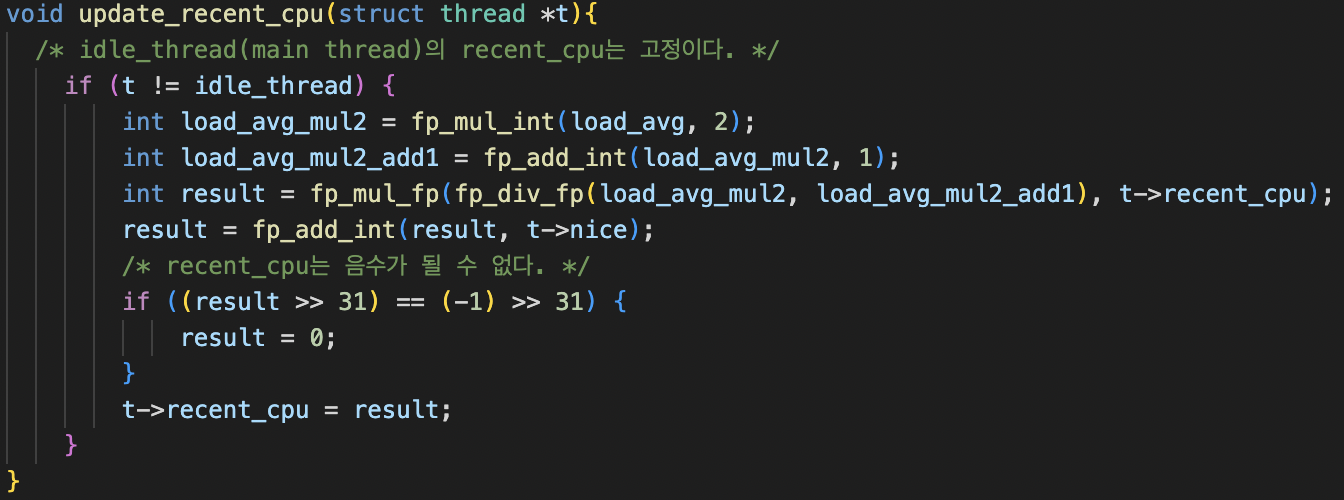
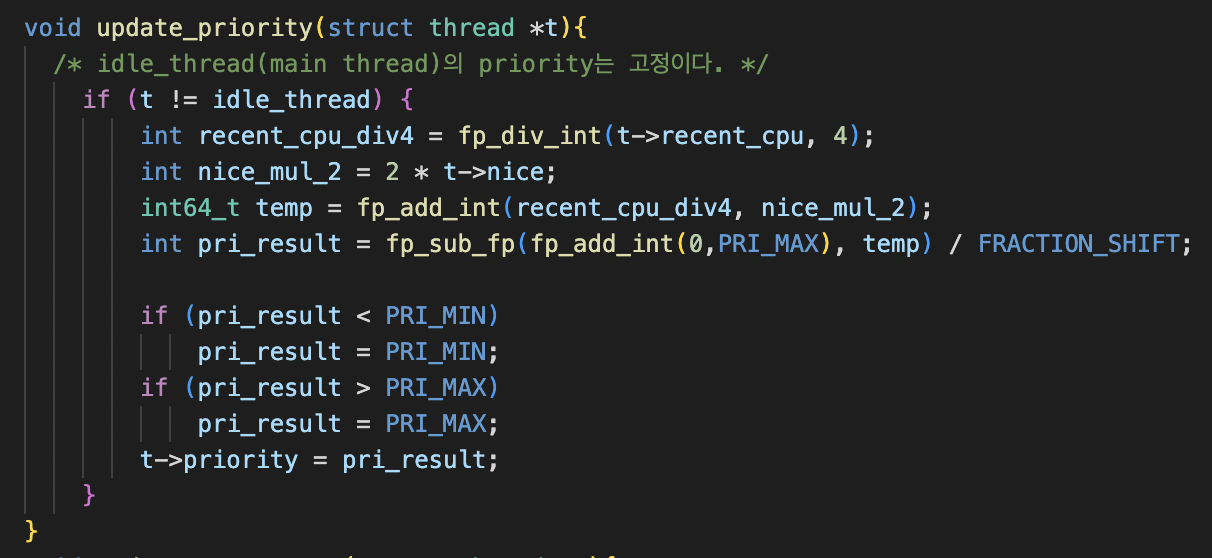
따라서 위와 같은 변수를 등록하여 thread\_start() 되기 전에는 thread\_yield()가 아무일도 하지 않도록 한다.

또 세마포어는 외부 인터럽트뿐만 아니라 내부 인터럽트에서도 사용된다. 내부 인터럽트가 걸렸을때마다 스케줄링을 다시 하면 문제가 생길 것이다. 따라서 외부 인터럽트가 걸릴때만 스케줄링을 다시하도록 한다.

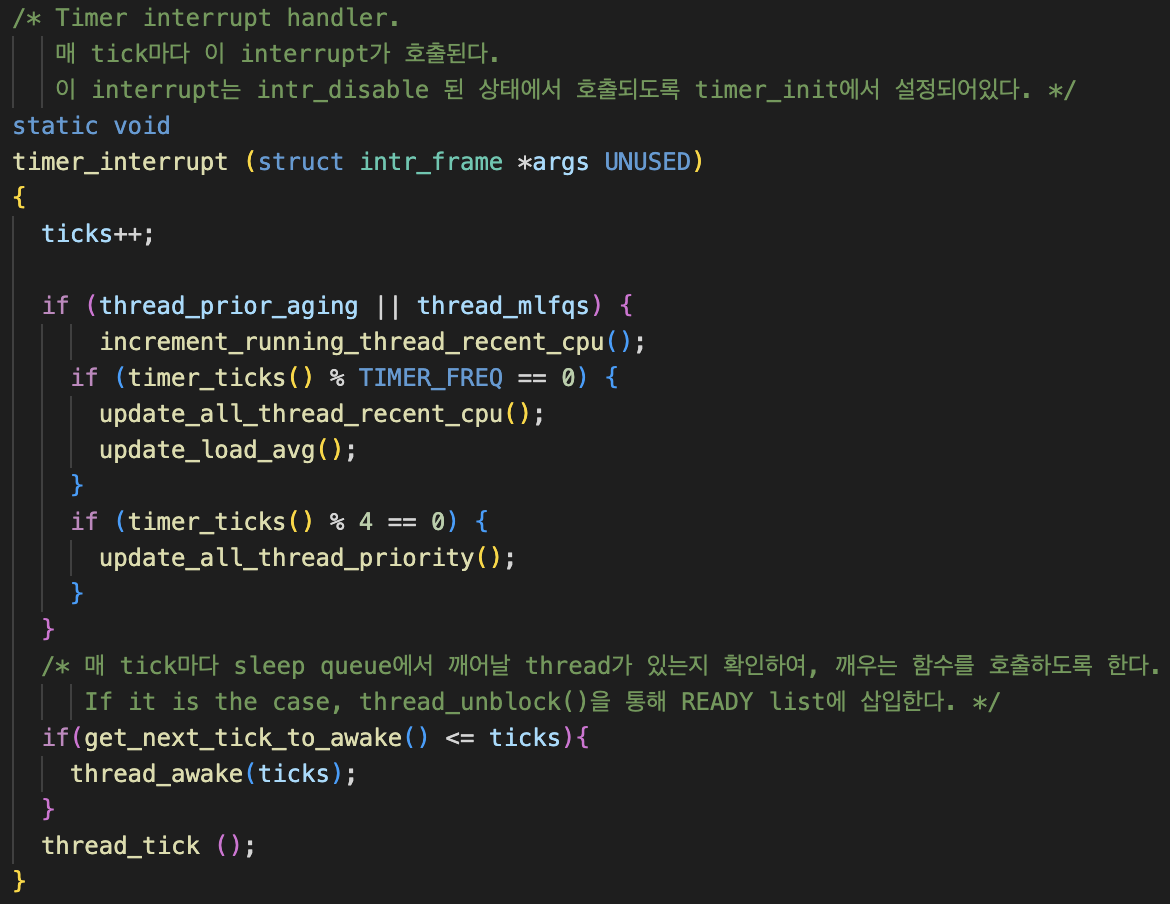
1. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)

우선, 핀토스는 소수연산을 지원하지 않으므로 fixed-point.c, fixed-point.h를 생성하여 고정소수점 연산을 지원하도록 해야 한다.

위에서 설명한 바에 의하면

* load\_avg : 시스템 당 하나만 존재한다. 1분동안 수행가능한 스레드의 평균 개수이다. 크면 클수록 priority는 천천히 증가한다. 매초마다 새로 계산된다. (loag\_avg = (59/60)\*load\_avg + (1/60)\*read\_threads)
* recent\_cpu : thread 당 고유한 값이다. 최근에 얼마나 많은 cpu time을 사용했는가를 의미한다. 클수록 priority는 낮아진다. 매 1 tick마다 기본적으로 1이 올라가고, 매초마다 새로 계산된다.(recent\_cpu = (2\*load\_avg)/(2\*load\_avg + 1)\*recent\_cpu + nice)
* nice : thread 당 고유한 값이다. nice가 클수록 다른 스레드에게 양보하는 정도가 크다. 클수록 priority는 낮아진다.

이고, 이를 이용하여 코드를 작성하였다.

그리고 최종적으로 timer\_interrupt()의 전체 코드는 recent\_cpu, priority, nice의 특성을 이용하여 이와같이 나타내진다.

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