**Pintos Project 3: Threads**

담당 교수 : 박성용 교수님

학번 / 이름 : 20191300 / 남유정

개발 기간 : 11/17~12/3

1. **개발 목표**

Project 1, 2에서 userprogram에서 프로세스가 돌아가도록 했다면, Project 3에서는 스레드에 관련된 기능을 구현했다. 스레드란, 프로세스가 할당받은 리소스를 이용하는 단위를 말한다. alarm-clock, priority-scheduling, BSD scheduler을 구현하는 것을 개발 목표로 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Alarm Clock

thread가 일정 시간 이상 실행되지 않으면, timer\_sleep()함수가 실행되어 thread를 sleep시킨다. 일정 시간 이하로 실행 될 경우, while문을 이용해 시간이 지날 때까지 계속 돌며, thread\_yield()를 호출한다. 이 때, 계속 thread가 running state와 ready state를 오가는 비효율적인 상태이므로, 이를 개선시킨다.

* 1. Priority Scheduling

RR 스케줄링을 이용한 상태에서 우선 순위 스케줄링을 도입한다. 기존의 thread\_yield()혹은 thread\_unblock이 호출되면 무조건 ready list의 맨 끝에 들어가던 기존의 방식에서, 우선순위가 높은 새로운 thread를 먼저 실행시키도록 한다. 또한, starvation이 생겨 실행되지 않는 thread가 있지 않도록 priority aging을 진행한다.

* 1. Advanced Scheduler (추가구현을 한 경우)

BSD 스케줄링을 구현한다. multi-level feedback queue를 이용해, 짧은 작업을 먼저 실행시키고, 응답시간을 최적화하도록 한다. 따라서, ready queue에서 각 우선 순위마다의 ready queue를 갖도록 하고, schedule()이 호출되면 우선순위가 가장 높은 queue의 thread가 실행되도록 한다. priority외에도 nice, recent\_cpu, load\_avg 를 통해 구현할 수 있다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

block state의 thread를 관리하기 위한 sleep list를 새로 만들어 thread를 모두 저장하고, tick을 1씩 늘려가며 sleep list 내부의 thread 중 wake up time이 된 thread를 wake up시켜 list에서 빼고 unblock state로 바꿔주고 ready queue에 넣어 깨워준다.

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

ready list의 뒤에 더 높은 priority의 thread가 삽입되면, 해당 thread가 먼저 실행되어야 한다. 돌던 현재 thread를 ready state로 바꾸고, ready list에 priority 순으로 thread를 정렬한다. ready list에서 가장 높은 priority를 갖는 thread를 실행시킨다.

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

thread에 nice와 recent\_cpu를 추가한다. recent\_cpu는 thread의 cpu 시간을 계산한다. time\_interrupt()가 실행되면, recent\_cpu는 +1되고, 매 tick마다 thread들의 recent\_cpu는 다시 계산된다. recent\_cpu는 아래의 식으 ㄹ따른다.

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

load\_avg는 ready state에 있는 thread의 수의 평균이다. 매 tick마다 update되며, 그 식은 아래와 같다.

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

이를 따라 priority는 아래의 수식을 따른다.

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

pintos의 소수점 연산은 fixed point 연산을 따르도록 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

11/17~12/3

11/17~11/21 alarm clock 구현

11/22~11/28 priority scheduling 구현

11/29~12/2 advanced scheduling 구현

12/3 디버깅 및 보고서 작성

* 1. **개발 방법**

timer.c에 전역 변수로 struct list sleep을 선언하여, block state에 있는 list를 받는다. while문을 통해 busy waiting을 하고 있는 timer\_sleep()함수를 수정한다. wakeuptime을 timer\_sleep()에서 계산하여 sleep에 들어 있는 block state의 thread를 깨워 줄 시간을 찾는다. timer\_interrupt()에서 thread\_wakeup()함수를 통해 wakeuptime의 시간이 지난 thread를 깨우는 코드를 추가한다.

thread.h의 thread 구조체에서 wakeuptime을 추가하고, thread.c 의 thread\_init에서 0으로 초기화한다. thread\_yield()와 thread\_unblock() 함수에서 list\_insert\_ordered() 함수를 통해 list를 우선순위 순으로 넣을 수 있도록 코드를 작성한다. 이때, comparison\_priority()함수를 사용하며, 이는 두 priority를 비교하여 처음 값이 작으면 true를 리턴하도록 하여 sorting에 사용되도록 한다. thread\_create() 함수에 현재 priority보다 더 높은 priority의 thread가 생성되면 thread\_yield()를 호출해 높은 priority의 thread를 먼저 수행하도록 코드를 덧붙인다. thrad\_set\_priority()에서 현재 priority를 새로운 priority를 thread에 넣고, priority scheduling이 되도록 코드를 수정한다.

sync.c의 sema\_up() 함수가 priority를 반영하도록 코드를 수정한다. 기존의 들어오는 waiter에 대해 바로 thread를 unblock했던 것을 기다리고 있는 waiters 중 priority가 가장 크 것을 unblock하고, 다시 thread\_yield()를 호출해 스케줄링 하도록 코드를 수정한다.

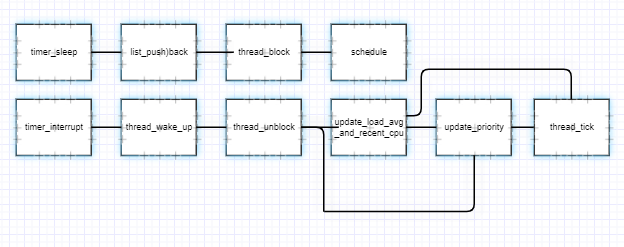
thread.h, initl.c, thread.c에 priority aging에 관련된 부분을 ppt 31페이지에 나온 내용을 참고하여 추가한다.

thread.h,의 thread 구조체에 recent\_cpu와 nice를 추가하고, thread.c의 thread\_init()에서 초기화해준다. int load\_avg의 전역 변수를 선언하고, fixed point 연산을 위해 FRACTION을 define해 주었다. 그리고, nice, load\_avg, recent\_cpu함수와 priority 설정을 위한 함수들을 추가한다.

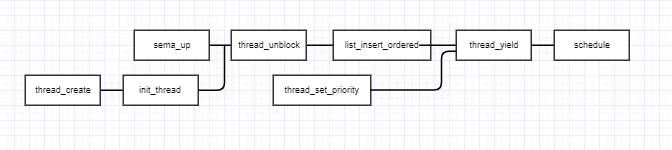
timer.c의 timer\_interrupt함수에서 thread\_prior aging과 thread\_mlfqs 조건을 포함하도록 수정한다. 4번의 tick마다 thread의 우선순위를 업데이트 하도록 하고, 매 tick timer\_FREQ만큼 tick이 흘렀을 때 load\_avg와 recent\_cpu를 업데이트하도록 함수를 호출하게 함수를 수정한다.

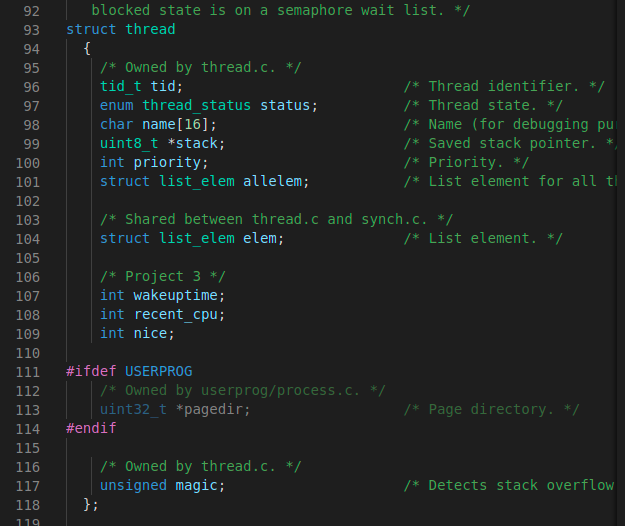
1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)
  + 1. alarm clock



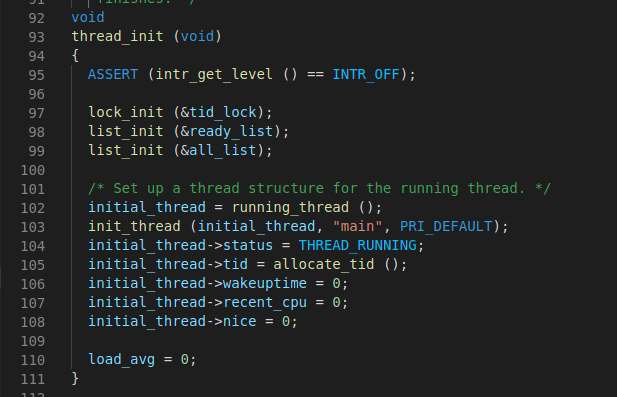
* + 1. priority scheduling



* 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. alarm clock
       1. threads/thread.h

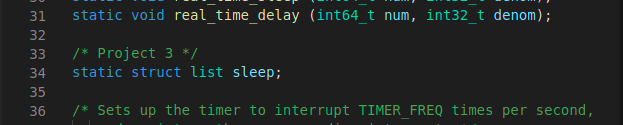
thread 구조체에 thread가 깨어날 시간인 wakeuptime을 선언하였다.

* + - 1. threads/thread.c

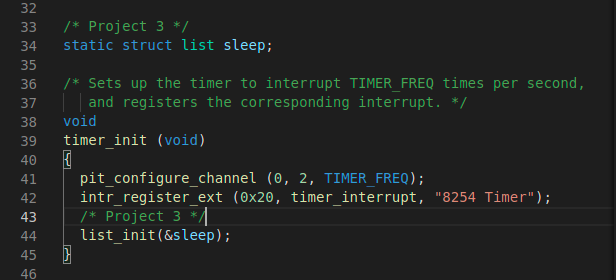


thread\_init()에서 wakeuptime을 0으로 초기화시킨다.

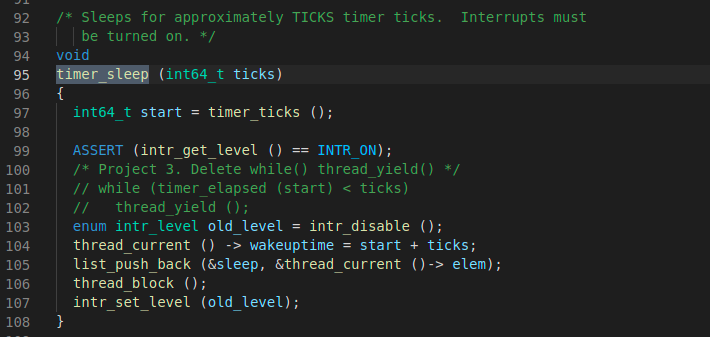
* + - 1. devices/timer.c



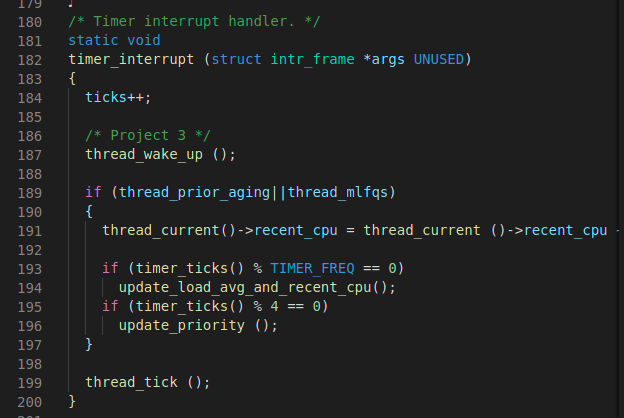
block state에 있는 list를 저장하는 struct list sleep을 선언한다.



timer\_init() 함수에서 선언한 list를 초기화해준다.

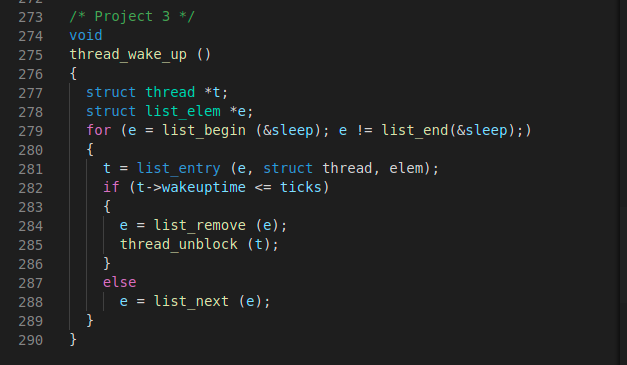


주석 처리된 busy waiting하고 있는 기존의 코드를 지우고, wakeuptime을 계산하고, sleep에 넣어주고, thread\_block함수를 통해 해당 스레드를 block state로 바꾼다.



주기적으로 tick을 증가시키는 함수이고, 매 tick이 갱신될 때마다 thread\_wake\_up) 함수를 불러 wakeuptime에 도달한 thread를 탐색하고, 있다면 깨워준다.

thread\_wake\_up()함수는 아래와 같다.



sleep을 처음부터 끝까지 돌며, wakeuptime보다 tick 수가 적은 리스트의 경우, sleep에서 제거하고 해당 thread의 state를 unblock으로 바꿔준다.

* + 1. priority scheduling & 3. BSD schduling
       1. threads/thread.h

#include “threads/synch.h” /\* Project 3 \*/

/\* Project 3 \*/

//#ifdef USERPROG

extern bool thread\_prior\_aging;

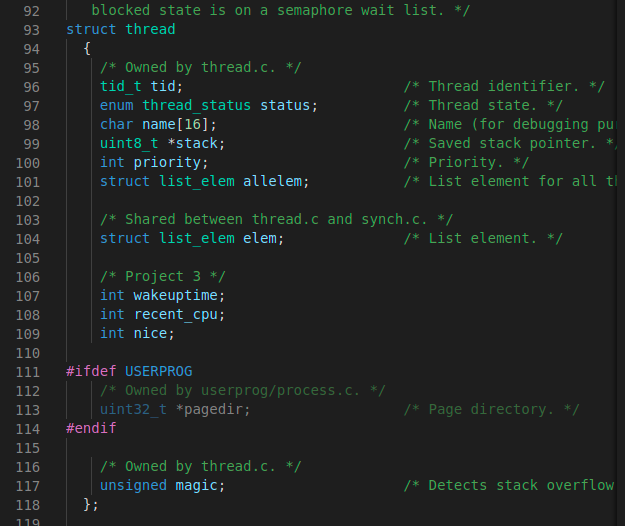
//#endif

static int load\_avg = 0;

#define FRACTION (1<<14)

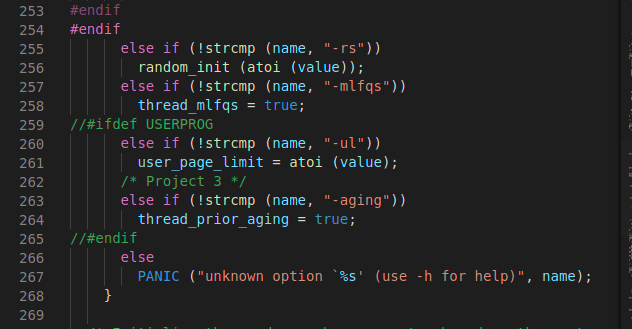
\*\* 캡처가 안되어 소스코드 일부를 typing 하였습니다.

synchronization을 위해 “threads/synch.h”을 include하고, priority aging을 위하여 bool thread\_prior\_aging을 선언하였다. 또한, priority aging을 위해 load\_avg을 전역 변수로 선언하고 0으로 초기화 하며, fixed point 연산을 위해 FRACTION을 선언해주었다.



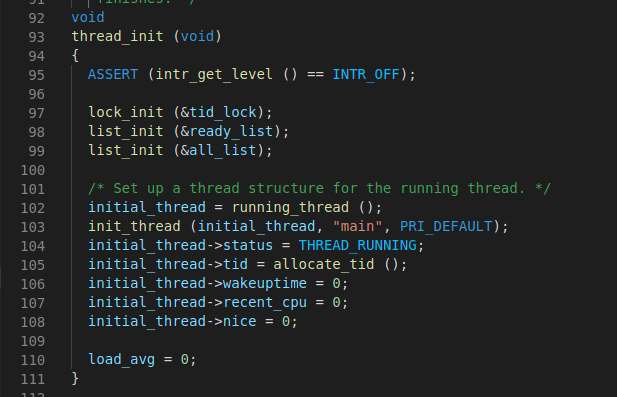
thread 구조체에 recent\_cpu와 nice를 선언해주었다.

* + - 1. threads/init.c

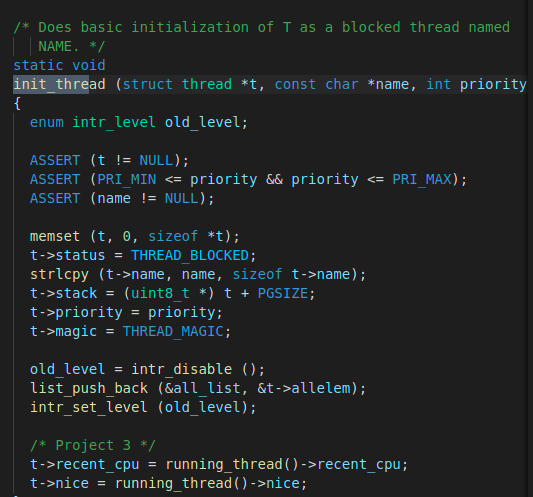


ppt를 따라, ‘-aging’ 옵션이 있다면 thread\_prior\_aging이 true가 되도록 선언해준다.

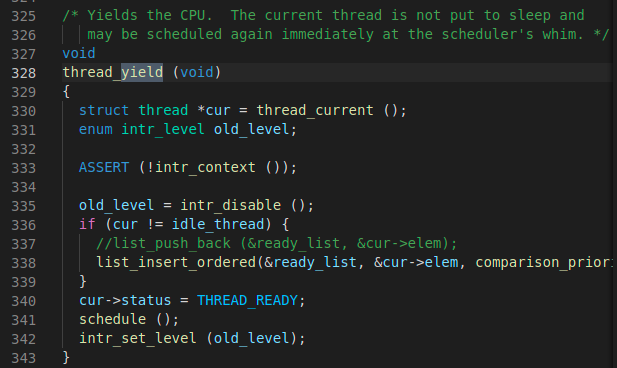
* + - 1. threads/thread.c



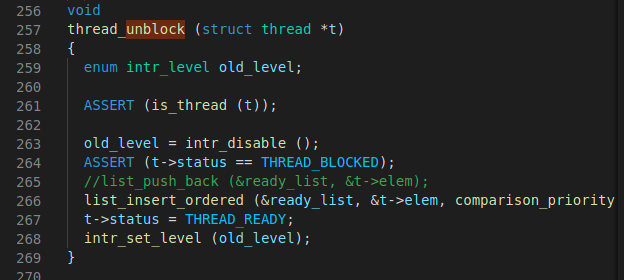
추가로 선언한 recent\_cpu와 nice를 초기화하였다.



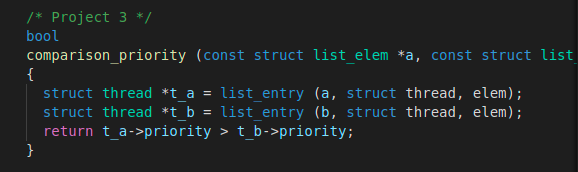
init\_thread()에서 recent\_cpu와 nice를 parent thread, 즉 running\_thread()의 값을 받도록 하였다.



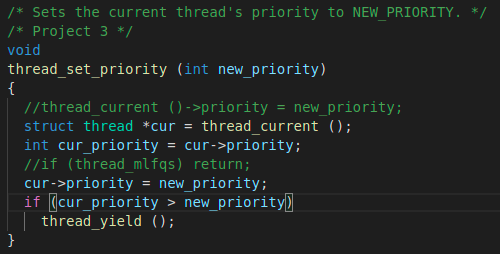
현재 thread가 idle thread가 아닐 때, 이 thread의 element를 read\_list에 priority 순서대로 삽입한다. 그리고 현재 thread의 상태를 ready state로 바꾸고, 스케줄링을 다시 한다.



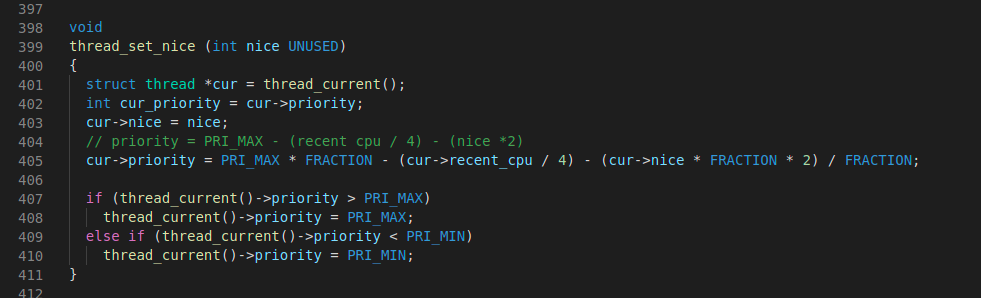
block state에 있는 thread를 unblock해 ready state로 바꾸기 위해 ready\_list에 priority 순서대로 삽입한다. 그리고, status를 ready로 바꿔준다.



list\_insert\_ordered()함수에 사용하는 이 comparison\_priority() 함수는 두 list elem을 받아 대소관계를 Boolean type으로 리턴하는 함수이다.



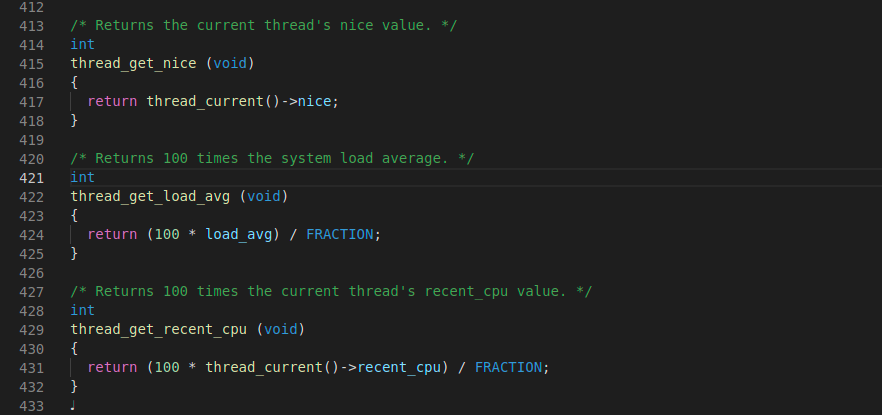
thread\_set\_priority 함수는, 새로운 우선순위로 현재의 우선 순위를 update하고, 만약 새로운 priority보다 이전의 priority가 더 크다면 thread\_yield() 함수를 통해 re-scheduling 한다.



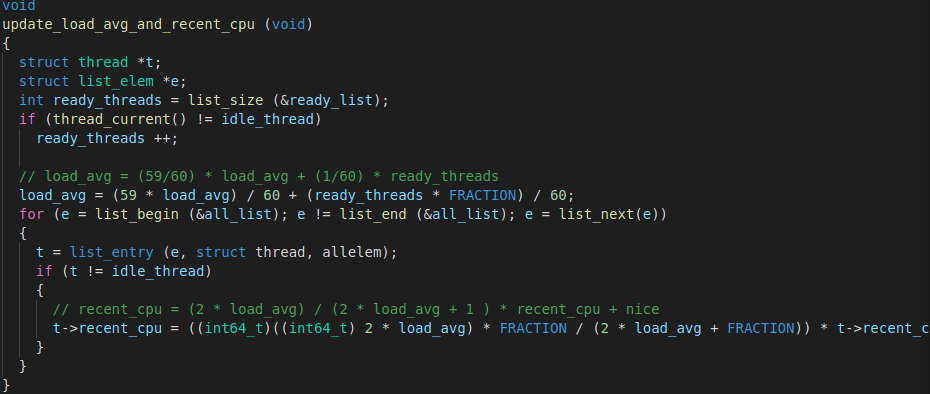
thread\_set\_nice()함수는 입력받은 nice로 현재 thread의 nice를 설정하는 함수이다. nice값을 현재 thread에 할당하고, priority 연산을 진행한다. 이 식은 아래와 같다.

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

fixed point연산에 따라 연산을 진행한다. 연산 후, 계산한 priority값이 priority의 하한선과 상한선을 넘는지 확인하고 넘는 경우, 하한선, 상한선으로 재설정해주도록 한다.



thread\_get\_nice()함수는 현재 thread의 nice를 리턴해 주며, get\_load\_avg()함수는 현재 load average에 100을 곱한 값을 리턴하는 함수이다. get\_recent\_cpu()함수는 현재 thread의 recent\_cpu값에 100을 곱한 값을 리턴하는 함수이다. 모든 연산은 fixed point 연산을 통한다.

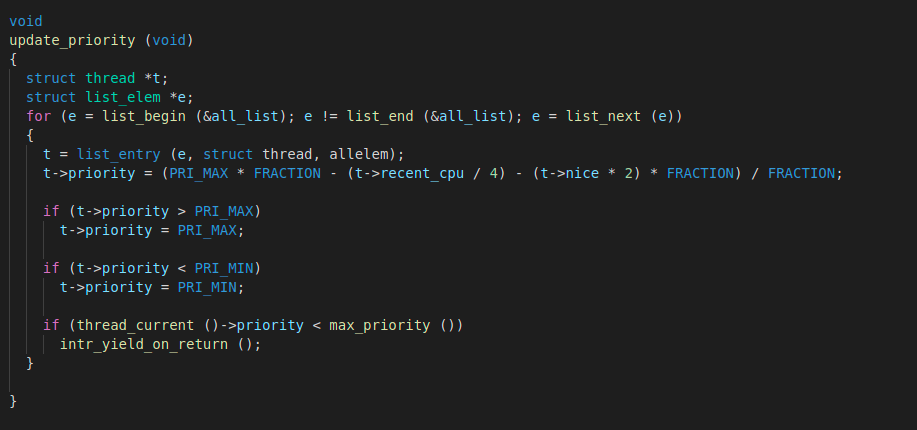


udate\_load\_avg\_and\_recent\_cpu값은 모든 thread의 recent\_cpu와 nice를 갱신하는 함수이다. 현재 thread가 idle thread가 아니라면, ready\_list의 size를 1 늘려주고, 수정한 값을 이용해 load\_avg를 계산한다. 그 식은 아래와 같다.

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

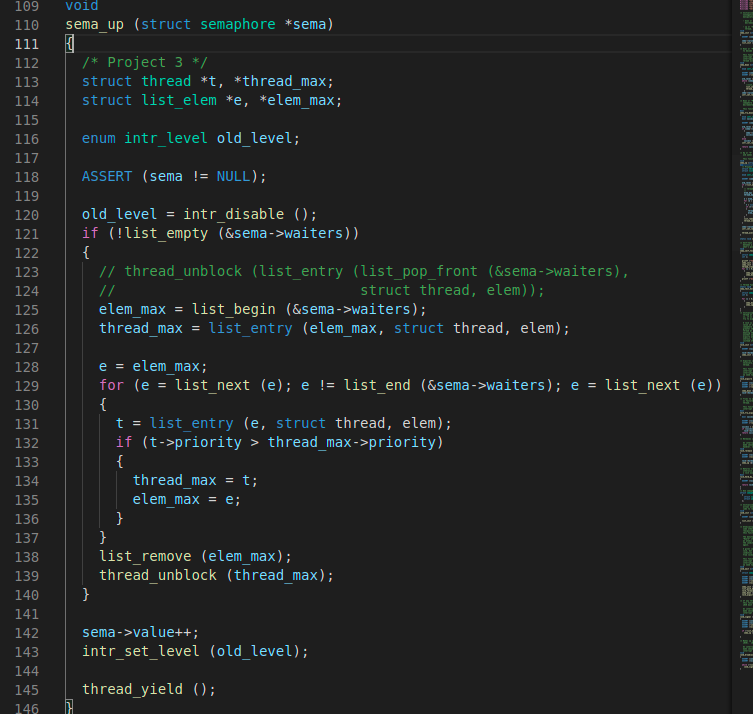
그 다음, all\_list에 있는 모든 thread들을 돌면서 idle thread가 아니라면 해당 thread의 recent\_cpu를 갱신한다. 그 식은 아래와 같다.

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



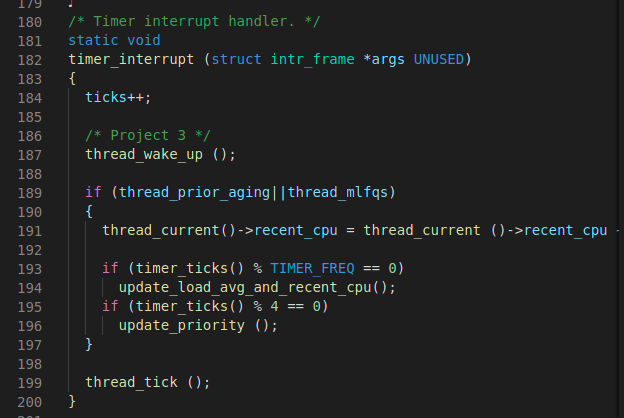
update\_priority() 함수는 모든 프로세스의 priority를 갱신하는 함수이다. all\_list를 돌며 모든 thread의 priority를 게산하고, 상한선 PRI\_MAX나 하한선 PRI\_MIN을 넘는 경우, 각각 PRI\_MAX와 PRI\_MAX로 수정한다.

* + - 1. threads/sync.c



세마포어가 priority를 반영하도록 sema\_up() 함수를 수정한다. semaphore을 기다리고 있는 waiters가 존재하는 동안, 바로 thread\_unblock을 했던 코드를 주석 처리하고, waiters를 돌며 가장 큰 priority를 갖고 있는 thread를 찾아, waiters 리스트에서 제거하고 thread\_unblock함수를 호출해 unblock state로 바꿔준다. 그 후, thread\_yield()를 통해 re-schedule 한다.

* + - 1. devices/timer.c



주기적으로 tick은 증가하고, 깨울 thread가 있는지 확인한다. thread prior aging 혹은 threas\_mlfqs가 true인 경우, 추가 처리가 필요한데, tick이 1 증가할 때마다 그에 상응하게 FRACTION만큼 증가시킨다. 그리고, timers\_ticks가 TIMER\_FREQ만큼 지나면 load\_avg와 recent\_cpu를 갱신하도록 한다. timer\_ticks가 4만큼 지나면, priority를 갱신하도록 한다. 02

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

텍스트이(가) 표시된 사진

자동 생성된 설명

priority-lifo 수행 결과는 위와 같다. priority-lifo.c 코드를 보면, 같은 우선순위의 thrad가 여러 개 생성되고, round robin schduling을 통해 수행되도록 하는 코드이다. 16개의 thread가 last in first out policy대로 16번 반복되면 된다. 실행 결과를 보면 0~15의 thread가 16번씩 실행되는 것을 확인할 수 있다.

* + 1. make check

