**Pintos Project 3: Threads**

담당 교수 : 김영재 교수님

조 / 조원 : 20211606 / 한석기

개발 기간 : 23.10.30 ~ 23.11.19

1. **개발 목표**

그저 round-robin 방식의 scheduler에서 우선순위를 부여하고 scheduling하는 scheduler를 구현한다. 그 과정에서 이전에 비효율적으로 구현되어 있던 timer를 Alarm clock 구현으로 효율적으로 바꿔준다. scheduler를 구현 시, 여러 개의 우선순위가 존재하는 큐를 이용하는 MLFQ를 추가적으로 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1) Alarm Clock

timer에서 RUNNING State, READY State를 반복하는 Busy waiting 방식으로 되어 있던 것을 중간에 BLOCKED State를 두어 시간이 되면 Ready Queue에 삽입해주는 것을 구현해 이전의 비효율적인 방식을 개선할 수 있다.

2) Priority Scheduling

Round-robin 방식으로 Ready Queue에서 배정된 시간이 지나면 맨 앞 RUNNING thread를 pop해서 Ready Queue에 push하고 Ready Queue의 front는 RUNNING State가 되고 이런 단순한 구조에서 Threads에 Priority를 부여해 우선순위에 따라 RUNNING thread 및 Ready Queue 내의 threads 순서가 정해진다. 따라서, 우선순위에 따른 Scheduler를 만들 수 있다.

3) Advanced Scheduler (추가구현을 한 경우)

MLFQ(Multi Level Feedback Queue)를 이용해 Advanced Scheduler를 구현한다. Priority aging을 통해 Priority를 지속적으로 업데이트해 starvation을 해결하고 여러 큐를 통해 우선순위를 관리한다. 각 큐마다 우선순위가 있으며 큐 내에서는 Round-robin 방식으로 Scheduling한다.

* 1. **개발 내용**

1) Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술

RUNNING State와 READY State 사이에 BLOCKED State를 두어 RUNNING State에서 배정된 CPU Time을 소모했을 때, Sleep으로 READY State가 아닌 BLOCKED State로 설정하고 BLOCKED State를 관리하기 위한 List를 하나 만든다. Sleep을 wake하기 위한 timer도 thread 구조체 안에 만들어준다. thread가 BLOCKED State가 되어 BLOCKED State List에 들어갈 때, timer를 저장한다. 그리고 tick 마다 호출되는 timer\_interrupt 함수에서 BLOCKED State List를 전부 확인해, wake할 thread를 찾고 READY Queue에 넣는다.

2) Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야 하는지 서술

RUNNING thread와 Context switch가 일어나야 한다. 따라서 thread가 wake될 때, yield할 때, create될 때, lib/kernel/list.c의 list\_insert\_ordered 함수를 이용한다. 이 과정에서 priority는 compare함수에 따른다. 따라서 Ready list의 맨 앞은 Running thread인데 priority에 따라 schedule이 돼, Running thread보다 들어온 thread가 더 높은 우선순위를 가지고 있으면 CPU Control을 들어온 thread가 갖게 된다

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

- nice: -20 ~ 20 사이의 값으로 작을수록 우선순위가 높다.

- recent\_cpu: thread가 최근까지 사용한 CPU Time이다. 값이 클수록 CPU를 많이 사용했으므로 우선순위가 낮아진다.

- load\_avg: Ready State에 있는 thread의 평균 개수이다.

- ready\_threads: IDLE thread 제외, READY거나 RUNNING State인 thread 수이다.

\*식

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

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

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

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

23.10.30 ~ 23.11.05: 메뉴얼 및 ppt 자료 읽기

23.11.06 ~ 23.11.12: Alarm clock + Priority Scheduling 구현하기

23.11.13 ~ 23.11.16: Advanced Scheduler 구현하기

23.11.17 ~ 23.11.19: 보고서 작성

* 1. **개발 방법**

1) Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술

Blocked 상태의 thread를 저장할 sleep\_list list 자료구조를 추가해야 한다. 그리고 thread 구조체 내에 sleep 시간을 나타내는 sleep\_timer를 선언하고 sleep\_list를 순회할 때, sleep\_timer로 확인한다. timer\_sleep() 함수에서 blocked 상태를 추가하는 작업을 하고 timer\_interrupt() 함수에서 sleep\_list를 순회하는 작업을 한다. 그리고 sleep\_lsit를 초기화하는 작업은 timer\_init()에서 진행한다.

2) Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야 하는지 서술

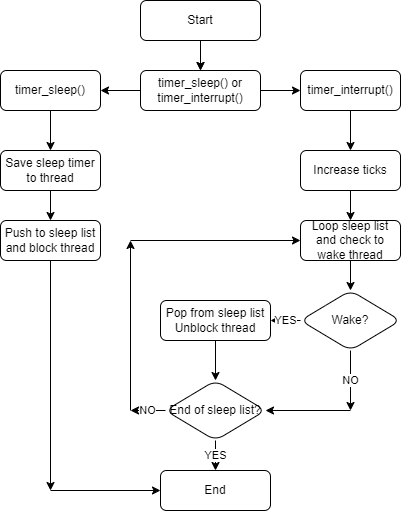
sema\_up() 함수에서 sema 구조체의 waiters 리스트를 정의한 compare 함수에 따라 sort하고 가장 앞에 있는 thread를 unblock하는 과정을 한다. thread\_create(), thread\_yield(), thread\_unblock()에서 scheduling 하는 과정을 수정해준다. priority에 따라 scheduling할 수 있게 compare함수를 정의한다. thread\_aging()을 구현해 priority를 계산한다.

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

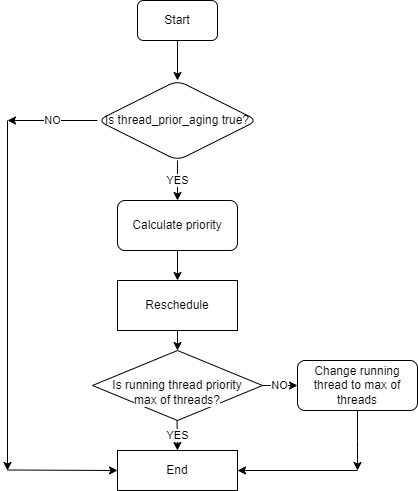
Fixed-Point Real Arithmetic 계산이 필요하다. 따라서 float\_add\_int, float\_sub\_float, float\_div\_int, int\_mul\_float, float\_mul\_float, float\_div\_float함수를 구현해 FRACTION\_DEVISOR를 이용한 계산을 한다. thread\_aging에서 해당 계산들을 하는 함수를 호출하고 계산하는 함수는 따로 thread.c에 만들어준다.

1. **연구 결과**
   1. **Flow Chart**

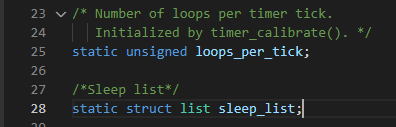
1) Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술

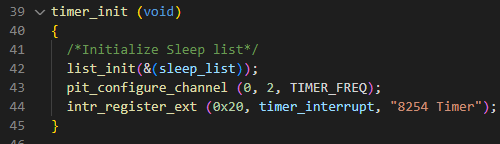
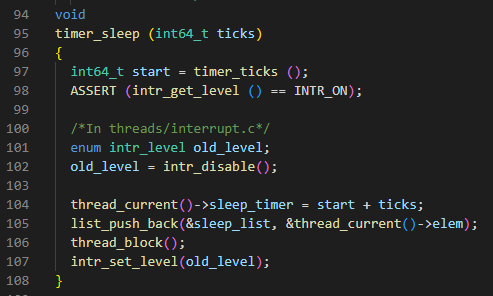


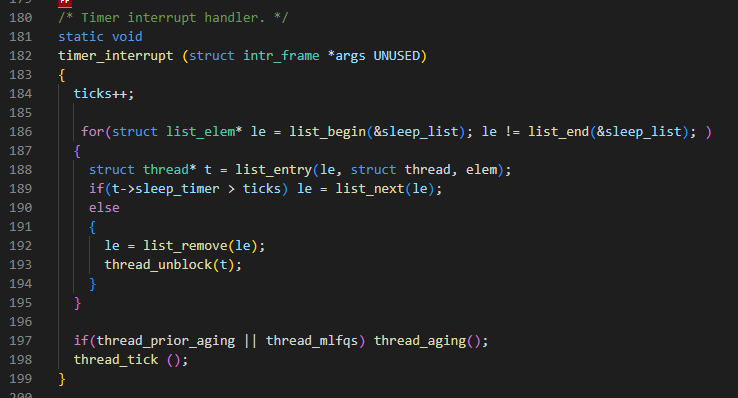
2) Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야 하는지 서술



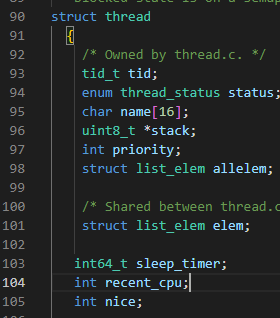
* 1. **제작 내용**



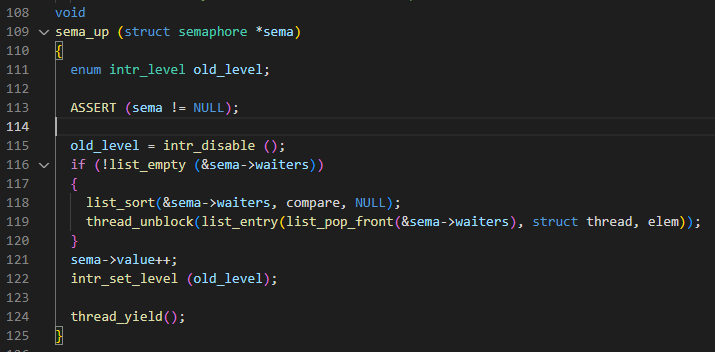




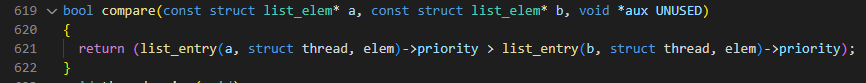
devices/timer.c에 sleep\_list를 선언했고 timer\_sleep, timer\_init, timer\_interrupt 함수를 수정했다. timer\_sleep함수에서 sleep\_list를 활용한 3-state management를 구현하고 timer\_init에서는 sleep\_list를 초기화한다. timer\_interrupt에서는 sleep\_list 내의 wake해야하는 thread를 확인해 wake해준다. 이 과정에서 사용한 list 관련 함수들은 lib/kernel/list.c에 구현되어 있다.



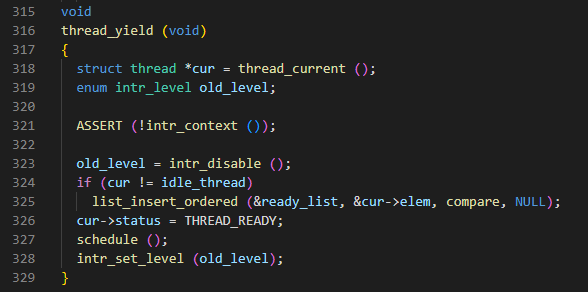
threads/thread.h의 thread 구조체 내에 선언한 변수들이다. sleep\_timer는 각 thread의 wake할 때까지의 시간이고 recent\_cpu는 thread가 최근까지 CPU를 점유한 시간이다. nice는 pintos에서 thread에 부여한 -20 ~ 20 사이의 값으로 우선순위를 정할 때, 반영된다.



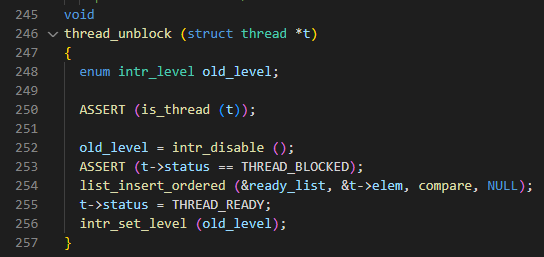
threads/synch.c의 sema\_up 함수에서 lsit를 sort하는 부분을 추가했다. 따라서 우선순위에 따라 맨 앞에 것이 pop된다.



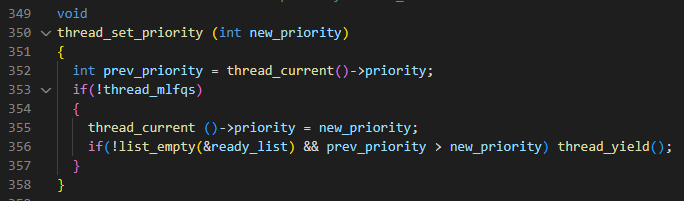
threads/thread.c 안에 구현한 compare함수다. 이 compare함수에 따라 list에 들어갈 수 있고 정렬될 수 있다.



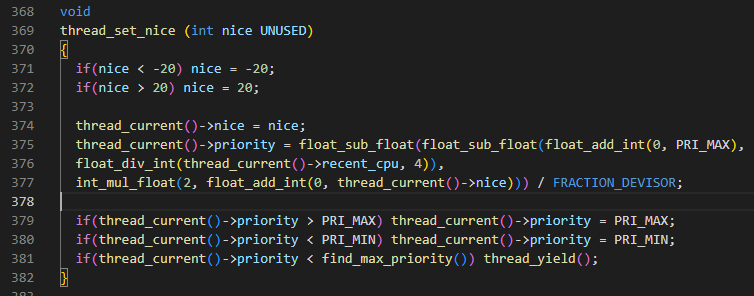
threads/thread.c의 thread\_yield 함수다. list\_insert\_ordered함수를 이용해 list에 우선순위대로 삽입한다.



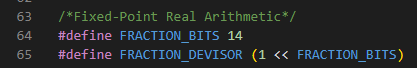
threads/thread.c의 thread\_unblock 함수다. 여기서도 list\_insert\_ordered함수를 이용한다.

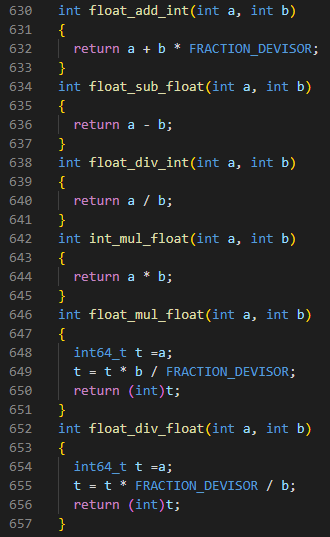


threads/thread.c의 thread\_set\_priority함수다. 현재 thread의 priority를 새로운 priority로 바꾸고 ready\_list가 비지 않았고 이전 우선순위보다 작다면 thread\_yield함수를 통해 다시 우선순위에 따라 정렬하는 작업을 한다.

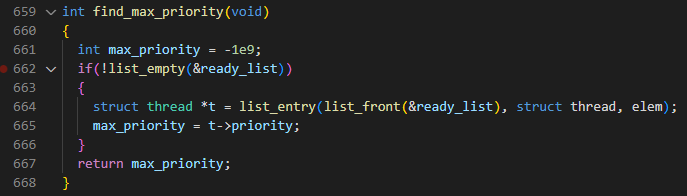


threads/thread.c의 thread\_set\_nice함수다. thread의 nice를 설정하고 priority를 계산한다. 이 과정에서 아래의 Fixed-Point Real Arithmetic이 사용된다.





각각의 계산을 위한 함수를 위와 같이 구현한다.



그리고 제일 큰 priority를 찾기 위한 함수를 구현한다. ready\_list를 돌며 가장 큰 priority를 찾는다.

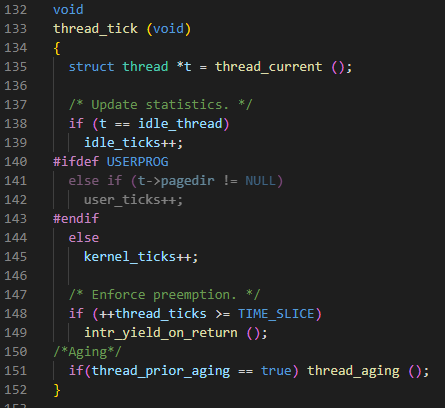




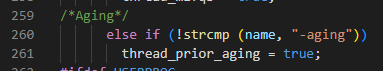
threads/thread.c에 global variable로 load\_avg와 thread\_prior\_aging을 선언한다.



threads/thread.h에는 extern변수로 thread\_prior\_aging을 선언한다.

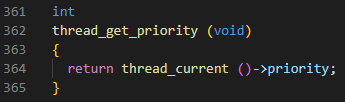


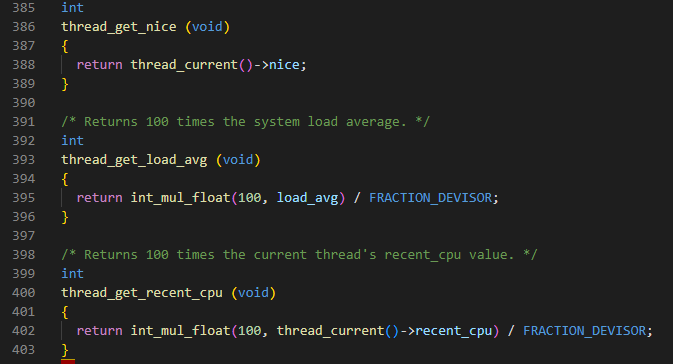
ppt 32쪽에 써져있는대로 thread\_aging을 thread\_tick함수에서 호출해준다.

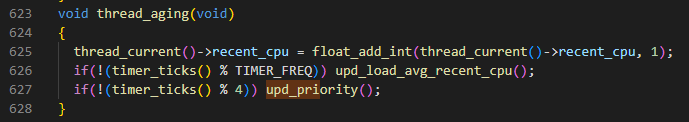


threads/init.c에서 parse\_options 함수에 위와 같이 추가해준다.

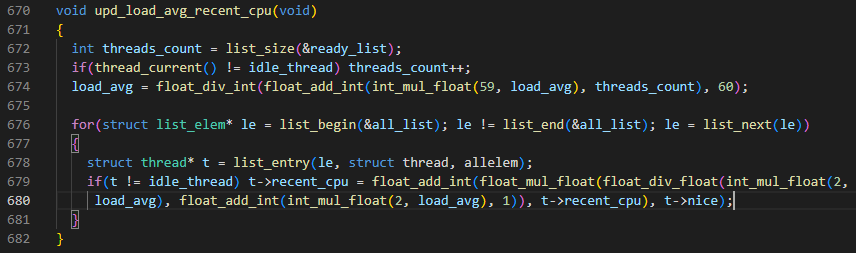
위 과정에서 ppt에는 ifdef USERPROG endif를 주고 했는데 그렇게 하면 USERPROG에만 한정해 해당 과정이 시행되기 때문에 threads에서 test 시, test가 통과하지 못한다. 따라서 ppt 32쪽과 달리, 모든 ifdef USERPROG endif를 제거했다.



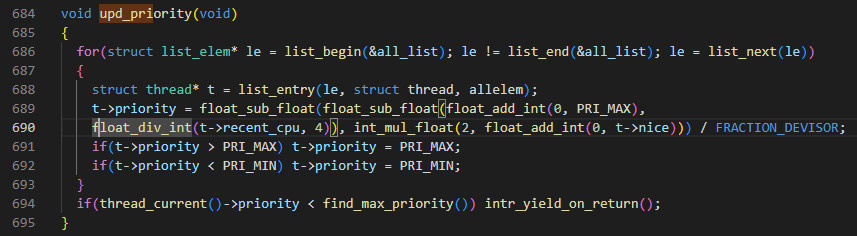


threads/thread.c의 현재 thread의 priority를 얻고 nice를 얻고 load\_avg를 얻고, recent\_cpu를 얻는 함수다.

threads/thread.c에 구현한 thread\_aging함수다. 여기서 recent\_cpu를 1 증가시키고 ticks의 조건에 따라 priority와 load\_avg, recent\_cpu를 update한다.

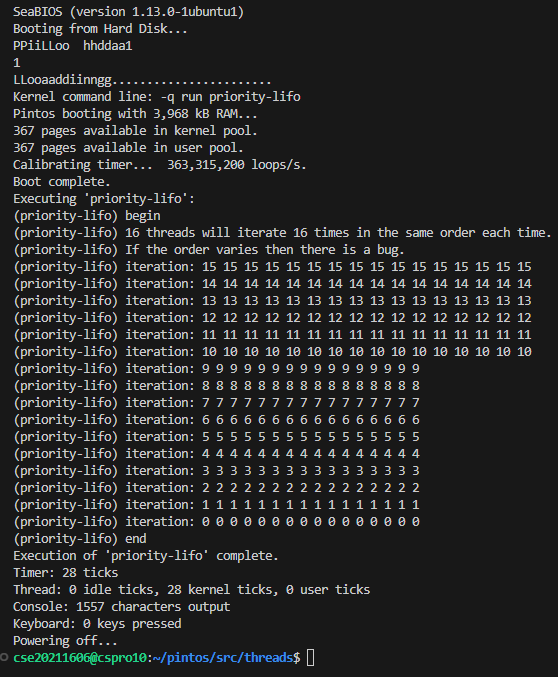


threads/thread.c의 load\_avg., recent\_cpu를 update하는 함수다.

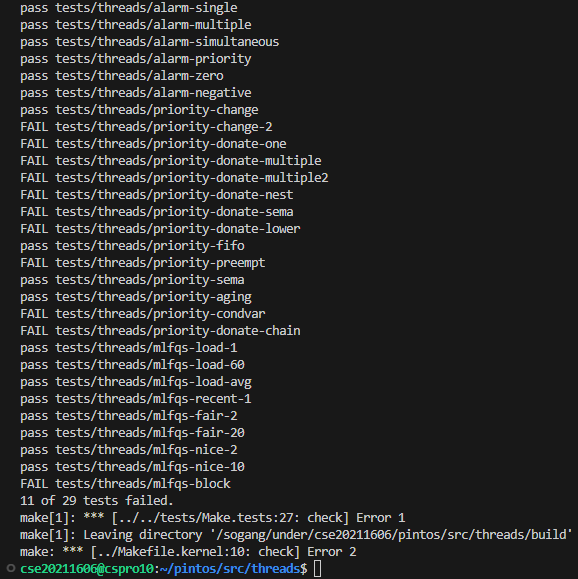


threads/thread.c의 priority를 update하는 함수다.

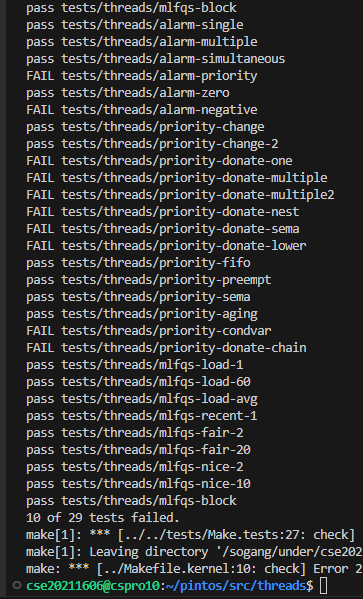
* 1. **시험 및 평가 내용**



priority\_info.c는 같은 0부터 15까지 priority의 thread를 16개 생성하는 작업을 16번 반복한다. 그래서 실행하는데 당연히 우선순위가 높은 15부터 실행이 되고 14, 13, .... 0 순으로 실행된다. 위의 결과는 해당 과정이 잘 실행됐다는 것을 보여준다.



priority-change-2와 priority-preempt와 mlfqs-block는 문자열 밀림으로 FAIL했다.



다시 실행했더니 다른 문자열 밀림 제외 위에서 안됐던 것이 pass했다.