**Pintos Project 3: Threads**

담당 교수 : 박성용 교수님

조 / 조원 : 정진원 / 20161643

개발 기간 :11/20~12/08

1. **개발 목표**

Pintos의 thread에 대한 이해를 바탕으로 busy wait 방식으로 작동하던 기존의 alarm clock을 수정하고, 동기화가 제대로 이루어지도록 한다. 또한 Round-Robin 방식으로 작동하던 scheduler를 thread의 priority를 고려하는 방식으로 수정한다. 추가적으로 BSD scheduler를 구현한다.

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

Timer\_sleep()에서 busy waiting으로 구현되어 thread\_yield()를 부르는 방식을 sleep / wakeup 기반으로 바꾸어 cpu의 낭비를 줄일 수 있다.

* 1. **Priority Scheduling**

기존의 pintos 스케쥴러가 새로운 thread가 들어오면 ready list의 마지막에 추가하던 방식을 priority에 따라 thread가 스케쥴링 되도록 수정하여, priority가 높은 thread가 우선적으로 수행 될 수 있게 한다.

* 1. **Advanced Scheduler (추가구현을 한 경우)**

Thread마다 다른 scheduling need를 balance 할 수 있는 general purpose scheduler를 구현한다. 4.4 BSD 스케쥴러와 유사한 Multilevel feedback queue를 구현하여 thread의 scheduling need에 부합하는 priority를 부여하고, priority가 높은 thread가 우선적으로 수행 될 수 있도록 한다.

* 1. **개발 내용**

1. **Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.**

Timer\_interrupt가 일어난 경우, sleep queue에 깨어날 thread가 있는지 확인하여 깨어나야 할 thread가 있다면 thread\_unblock 함수를 호출해준다. Thread\_unblock 함수에서는 interrupt를 disable 한 뒤 전달된 thread를 ready\_list에 priority에 따라 삽입해주고 thread의 상태를 ready 상태로 바꿔준다. 마지막으로 interrupt를 set 한다.

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

Ready list에 현재 수행중인 thread보다 높은 priority를 가진 새로운 thread가 추가된 경우, current thread는 processor를 즉시 새로운 thread에게 yield해야한다.

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

Priority의 경우 숫자가 클수록 우선순위가 높으며 thread 생성시 31로 초기화 된다. Priority의 경우 시간이 경과함에 따라 다음과 같은 식 priority = PRI\_MAX – (recent\_cpu / 4) – (nice \* 2)을 통해 갱신된다. Nice는 -20~20사이의 값으로 양수이면 우선순위를 감소시키고, 음수이면 증가시킨다. 초기값은 0이다. Recent\_cpu의 경우 recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1) \* recent\_cpu + nice로 결정되며 최근에 얼마나 많은 cpu time을 사용했는가를 표현한다. Time interrupt가 발생하면 running 상태의 thread의 recent\_cpu 값이 1 증가하며, 모든 thread의 recent cpu 값은 매 초마다 위의 식을 통해 다시 계산된다. Load\_avg의 경우 load\_avg = (59/60) \* load\_avg + (1/60) \* ready\_threads로 계산되며 Ready 상태에 있는 thread의 수를 exponentially weighted moving average를 통해 계산한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

11/20 ~ 11/24 프로그램 요구사항 파악

11/24 ~11/26 Alarm test 구현

11/27 ~ 12/01 Priority scheduling 구현

12/02 ~ 12/05 BSD scheduler 구현

12/05 ~ 12/08 보고서 작성

* 1. **개발 방법**
     1. **Alarm Clock**

src/thread/thread.h의 thread 구조체 안에 thread가 깨어날 시간을 저장하는 변수 wakeup\_time을 선언해준다. 기존의 busy-waiting 방식을 개선하기 위해 src/devices /timer.c에 sleep state의 thread를 저장하는 sleep\_queue를 list 형태로 전역변수로 선언해주고, timer\_sleep() 함수에서 busy waiting 대신, thread에 wakeup\_time을 설정해주고, sleep\_queue에 삽입한다. timer\_interrupt() 함수에서는 일어날 시간이 된 thread를 깨워주는 코드를 추가해준다.

* + 1. **Priority Scheduling**

priority 순서대로 ready queue를 정렬하기 위해, thread의 priority를 비교하는 cmp\_priority() 함수를 src/thread/thread.c에 추가한다. 또한 thread\_yield 함수에서 thread를 ready\_list에 삽입 할 때 priority 순으로 삽입되도록 한다. thread\_unblock() 함수에서도 마찬가지로 순서대로 삽입되도록 수정한다. thread\_create()함수에서는 새로 생성하는 thread의 priority가 수행중인 thread의 priority보다 높은 경우에 scheduling을 다시 해준다. 또한 thread의 priority를 set하는 thread\_set\_priority()함수를 작성하였다. semaphore를 up 시켜주는 경우에도 priority가 높은 waiter를 깨워야 하기 때문에 sema\_up() 함수를 수정해주었다.

또한 aging을 구현하기 위해 src/threads/thread.h과 src/thread/thread.c에 aging의 적용 여부를 나타내는 전역변수 thread\_prior\_aging을 선언해준다. aging은 bsd scheduler와 함께 구현하였다.

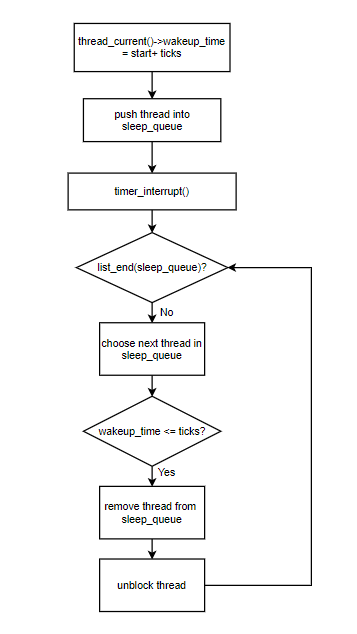
* + 1. **BSD scheduler**

src/threads/thread.c에 BSD scheduler의 load\_avg와 recent\_cpu 계산에 필요한 fixed point연산 함수 float\_arithmetic()과 FRACTION을 define해주어 추가했다. 추가적으로 load\_avg 값을 나타내는 전역변수 load\_avg를 선언하고 bsd scheduler의 사용 여부를 나타내는 전역변수 thread\_mlfqs를 선언했다. 또한 src/threads/thread.c의 다음 함수들을 수정하거나 추가해주었다.

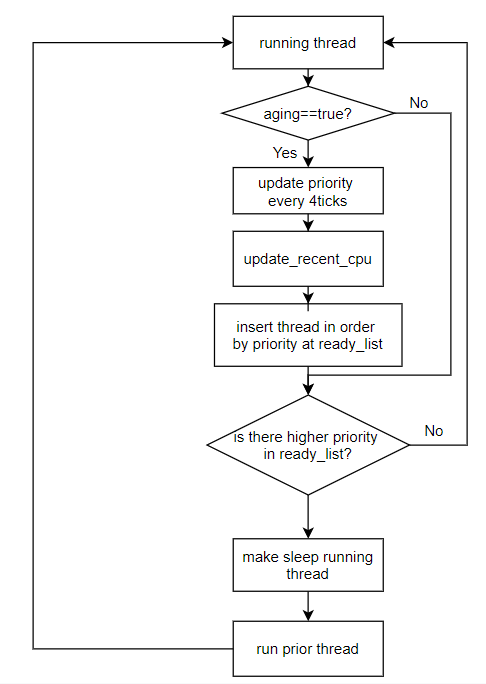
thread\_init()함수에는 thread의 nice와 recent\_cpu를 0으로 초기화 해주는 코드를 추가해준다. init\_thread() 함수에도 recent\_cpu와 nice를 running\_thread()의 recent\_cpu와 nice로 설정해주는 과정을 추가한다. 스케쥴링을 위해 필요한 함수인 ready\_list에 있는 최대 priority를 반환해주는 get\_max\_priority()함수도 추가해준다. thread\_set\_nice()함수에서는 thread의 nice를 설정해주고, nice가 바뀜에 따라 수정되는 priority를 계산해주는 과정을 추가한다. 만약 새로 설정된 priority가 ready\_list에 있는 최대 priority 보다 낮은 경우 thread\_yield를 호출하여 다시 스케쥴링해준다. thread\_get\_nice 함수에서는 현재 thread의 nice값을 반환해준다. thread\_get\_load\_avg와 thread\_get\_recent\_cpu에서는 구해야 하는 값에 100을 곱한 값을 return 해준다. 또한 load\_avg와 recent\_cpu를 주어진 공식에 따라 계산하는 함수 update\_avg\_cpu()를 추가해준다. priority 또한 주어진 공식에 따라 계산해주는 update\_priority함수를 구현해준다. 또한 요구사항에 따라 thread\_mlfqs가 set된 경우 thread\_set\_priority 함수를 disable 해주는 code를 추가한다.

마지막으로 src/devices/timer.c에 timer\_interrupt함수에 시간이 경과함에 따라 priority, load\_avg, recent\_cpu를 갱신해주는 과정을 추가한다.

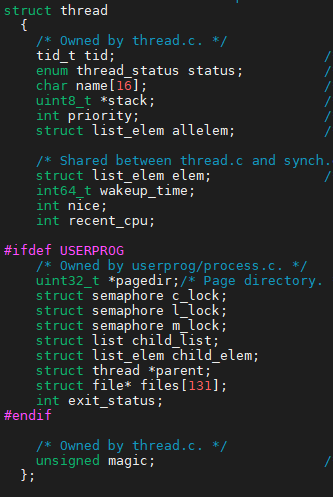
1. **연구 결과**
   1. **Flow Chart**
      1. **Alarm Clock**



* + 1. **Priority Scheduling**



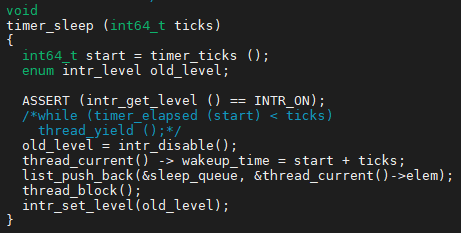
* 1. **제작 내용**
     1. **Alarm Clock**



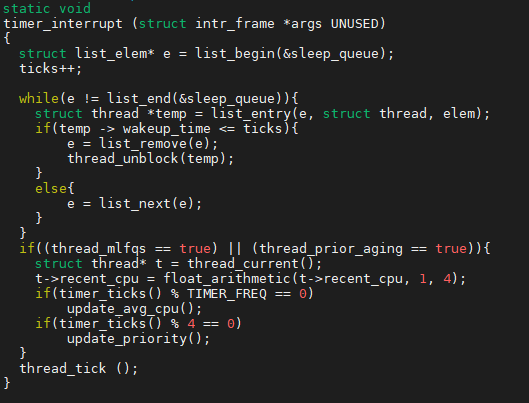
thread 구조체 안에 thread가 깨어날 시간을 저장하는 변수 wakeup\_time을 선언해준다.



Sleep state의 thread를 저장하는 sleep\_queue를 list 형태로 전역변수로 선언해준다.

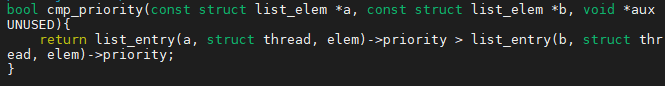


Busy-waiting 방식을 개선하기 위해 src/devices/timer.c에 timer\_sleep() 함수에서 먼저 현재 interrupt를 끄고 현재 interrupt 상태를 return 해주는 intr\_disable() 함수를 이용하여 interrupt를 disable 해준다. 이후 thread에 wakeup\_time을 설정해주고, sleep\_queue에 삽입하는 code를 작성한다. 이후 인자에 따라 interrupt를 끄거나 키는 intr\_set\_level() 함수를 이용하여 interrupt가 일어날 수 있도록 해준다.

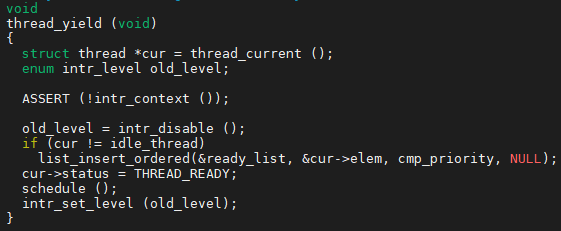


timer\_interrupt() 함수에서는 sleep\_queue를 돌면서 wakeup\_time이 ticks 보다 작아서 일어날 시간이 된 thread를 깨워주는 코드를 작성하였다.

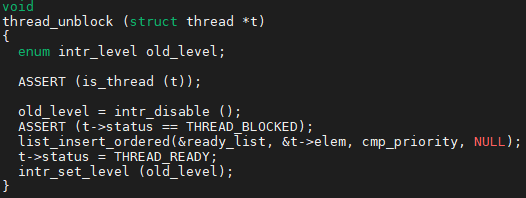
* + 1. **Priority Scheduling**



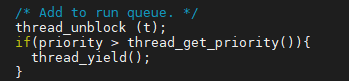
priority 순서대로 ready queue를 정렬하기 위해, thread의 priority를 비교하여 cmp\_priority 함수에서 인자로 전달받은 list\_elem의 priority를 비교한 bool 값을 return 해준다.



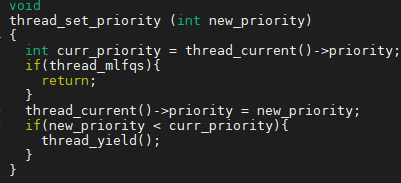
Thread\_yield 함수에서 thread를 ready\_list에 삽입 할 때 list\_insert\_ordered 함수와 앞서 구현한 cmp\_priority 함수를 이용하여 current thread의 element가 ready\_list에 priority 순으로 삽입되도록 한다.



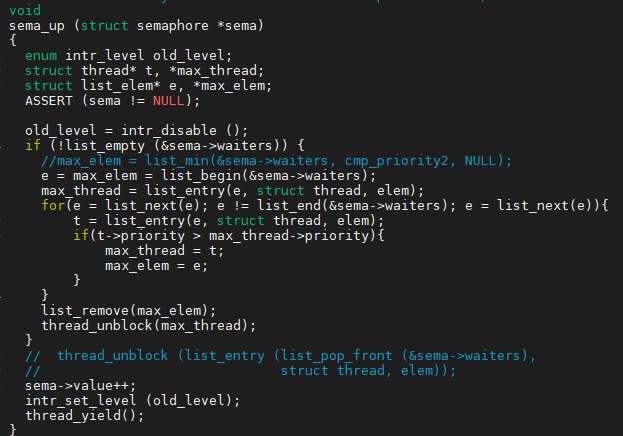
thread\_unblock() 함수에서도 block 되어있던 thread를 ready 상태로 만들어서 ready list에 삽입하는 것이기 때문에 list\_insert\_ordered 함수를 이용하여 순서대로 삽입되도록 수정한다. 이때 interrupt는 intr\_disable(), intr\_set\_level() 함수를 이용하여 비활성화 했다가 다시 복구시킨다.



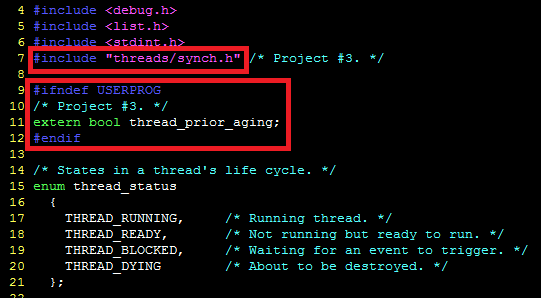
thread\_create()함수에서는 새로 생성하는 thread의 priority가 수행중인 thread의 priority보다 높은 경우에 scheduling을 다시 해주기 위해 생성된 thread의 priority가 thread\_get\_priority() 함수가 return 해주는 현재 running 상태인 thread의 priority 보다 높은 경우 thread\_yield를 통해 reschedule 해준다.

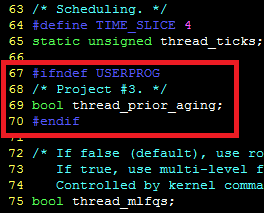
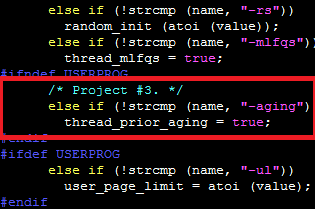


Thread의 priority를 set하는 thread\_set\_priority()함수를 작성하였다. 인자로 전달받은 new\_priority를 thread\_current()로 가져온 현재 수행중인 thread의 priority로 설정해준다. 새로 설정된 thread의 priority가 기존의 priority 보다 낮은 경우, 다른 priority가 높은 thread가 수행되어야 하기 때문에 이를 확인하기 위해 조건문과 thread\_yield() 함수를 이용해주었다.



semaphore를 up 시켜주는 경우에도 priority가 높은 waiter를 먼저 깨워야 하기 때문에 반복문을 이용하여 waiter중 priority가 가장 높은 element를 찾아서 이를 sema🡪waiters list에서 list\_remove() 함수를 이용해 제거하고, 이를 thread\_unblock() 함수로 ready 상태로 만들어주었다.



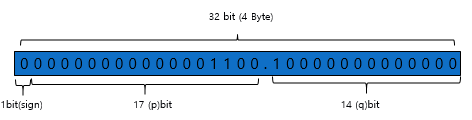


또한 aging을 구현하기 위해 ppt에 나온대로 src/threads/thread.h과 src/thread/thread.c에 aging의 적용 여부를 나타내는 전역변수 thread\_prior\_aging을 선언해주고, src/threads/init.c 에서는 aging 상태를 set하는 조건을 추가해주었다.

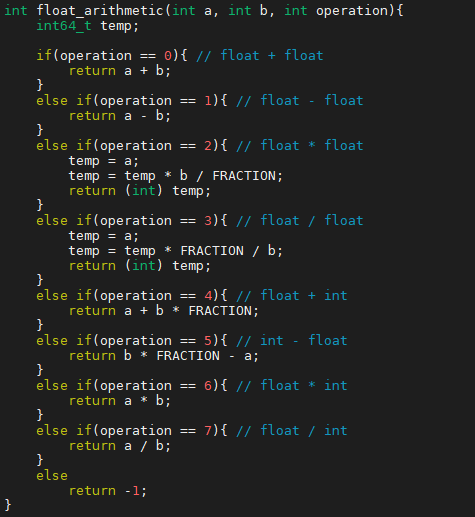
* 1. **BSD scheduler**



fixed point 연산에 필요한 fraction이라는 값을 define하고, load\_avg를 저장하기 위한 전역변수를 선언해주었다.



Fixed point의 수는 위와 같이 나타낼 수 있으며, FRACTION은 14bit 소수 부분을 계산하기 위해 선언한 것이다.



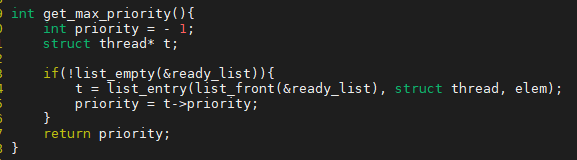
src/threads/thread.c에 BSD scheduler의 load\_avg와 recent\_cpu 계산에 필요한 fixed point연산 함수 float\_arithmetic()을 추가하였다. 주석으로 설명되어있는 총 8가지 경우에 대한 연산을 수행 할 수 있다.



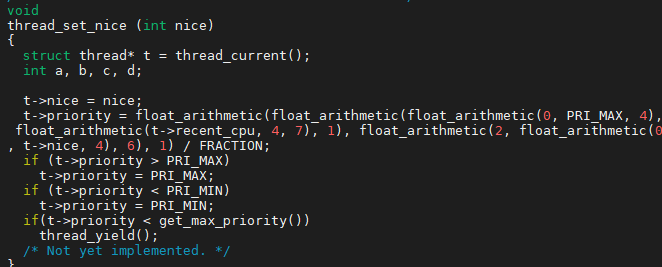
thread\_init()함수에는 thread의 nice와 recent\_cpu를 0으로 초기화 해주는 코드를 추가해준다.



init\_thread() 함수에도 recent\_cpu와 nice를 running\_thread()의 recent\_cpu와 nice로 설정해주는 과정을 추가한다.



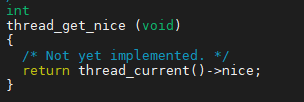
스케쥴링을 하기 위해 ready\_list의 thread들이 가진 최대 priority를 반환해주는 위와 같은 함수를 작성하였다.



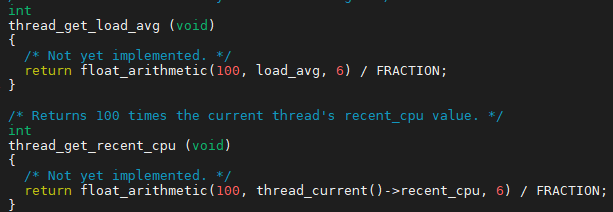
thread\_set\_nice()함수에서는 thread의 nice를 설정해주고, nice가 바뀜에 따라 다음의 공식으로 인하여



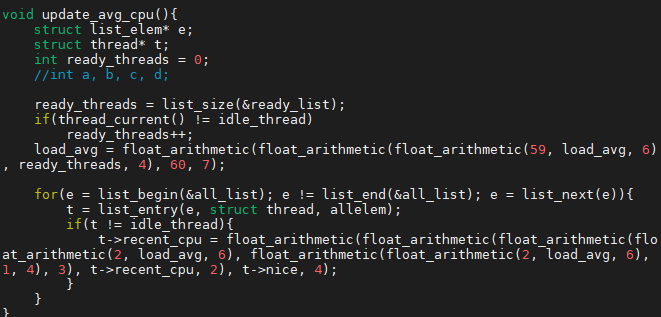
바뀌는 priority를 계산해주는 과정을 추가한다. 계산된 값이 PRI\_MAX와 PRI\_MIN의 범위를 벗어나지 않게 조건문을 추가해준다. 만약 새로 설정된 priority가 get\_max\_priority()로 확인한 ready list에 있는 thread의 최대 priority 보다 낮은 경우 thread\_yield()를 호출하여 다시 스케쥴링해준다.



thread\_get\_nice() 함수에서는 현재 thread의 nice값을 반환해준다.



thread\_get\_load\_avg()와 thread\_get\_recent\_cpu()에서는 프로그램의 요구사항에 따라load\_avg와 thread의 recent cpu 값에 100을 곱한 값을 return 해준다.

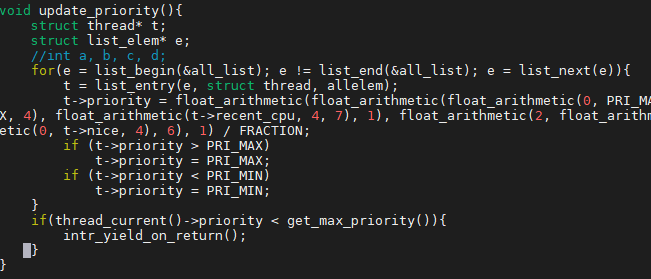


update\_avg\_cpu()라는 추가한 함수에서는 load\_avg와 recent\_cpu를 다음과 같이 주어진 공식에 따라 계산해준다.

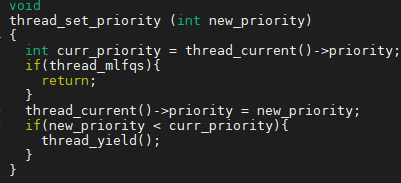




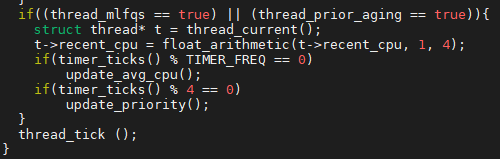
Recent\_cpu 값의 경우 idle 상태가 아닌 모든 thread들에 대해 for문을 이용하여 계산된다.



Priority도 마찬가지로 공식에 따라 update해주는 update\_priority함수를 추가해준다. for문을 돌며 모든 thread들에 대하여 priority 값을 계산해주며 priority 범위를 벗어나지 않게 조건문을 추가해주었다. 또한 현재 thread의 priority가 ready list에 있는 최대 priority 보다 작은 경우 interrupt가 끝나는 시점에 thread\_yield()가 수행되어 reschedule 되도록 intr\_yield\_on\_return() 함수를 이용해주었다.



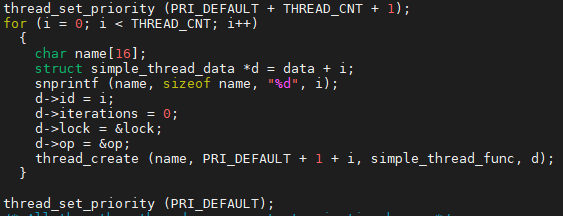
또한 요구사항에 따라 thread\_mlfqs가 set된 경우 thread\_set\_priority 함수를 disable 해주는 code를 추가한다.



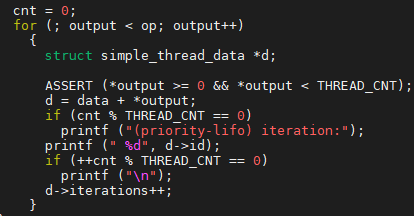
timer\_interrupt함수에 mlfqs == true이거나 aging이 설정된 경우 interrupt 가 발생할 때마다 recent\_cpu 값을 1 증가시킨다. 이후 timer\_ticks()의 값이 timer\_freq의 배수라면 update\_avg\_cpu() 함수를 호출하여 load\_avg와 recent\_cpu를 갱신하고, tick이 4의 배수라면 priority를 갱신해준다. 작업이 끝난 뒤 thread\_tick()함수를 호출하여 tick을 다시 계산해야한다.

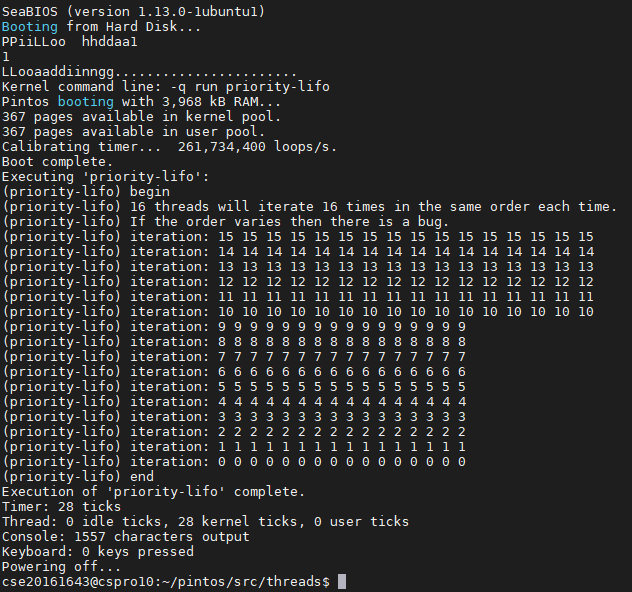
* 1. **시험 및 평가 내용**

1. **priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석**



Priority-lifo.c에서는 먼저 현재 thread의 priority를 가장 높게 설정하여 현재 thread에서thread\_cnt = 16만큼의 thread를 생성한다. 생성한 thread의 priority는 PRI\_DEFAULT+1+i로 나중에 생성된 thread의 우선순위가 더 높다. 이후 현재 thread의 priority를 PRI\_DEFAULT로 낮춰서 더 priority가 높은 생성된 thread들이 수행되어 id를 thread\_cnt번 출력되게 한다.





결과를 보면 위와 같이 후순위에 생성된 thread들이 먼저 수행되어 큰 id가 먼저 thread\_cnt 번 출력되는 것을 볼 수 있다. 이후 priority 순서대로 id가 0이 될 때까지 출력해준다.

1. **make check 수행 결과를 캡처하여 첨부**

