**Pintos Project 3: Threads**

담당 교수 : 박성용 교수님

학번 / 이름 : 20181202 김수미

개발 기간 : 2021.11.15 ~ 2021.12.5

1. **개발 목표**

이번 프로젝트에서는 운영체제의 thread 관리에 사용되는 Alarm Clock, Priority Scheduling, Advanced Scheduler(BSD Scheduler) 기능을 개선 및 구현하는 것을 목표로 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1) Alarm Clock  
Thread는 사용되지 않을 때 sleep 상태에 놓이게 되는데, 현재 코드 상에서는 timer ticks의 시간 만큼 thread를 sleep 상태에 놓여있게 하기 위해서 계속 할당된 시간이 지났는지의 여부를 while문을 통해 체크한다. 이 과정에서 thread는 running state와 ready state를 반복적으로 오가는 비효율적인 작업을 하기 때문에, thread를 block 시키는 방식을 이용하여 이를 해결해줘야 한다. 아직 timer ticks의 시간이 지나지 않은 thread는 blocked state로 전환하여 blocked state의 thread만을 저장하는 queue에 보관하고, wake up time이 되면 해당 thread를 ready queue로 옮겨주면 된다.

2) Priority Scheduling  
현재 코드 상에서는 thread\_yield 혹은 thread\_unblock 함수가 호출되면 current thread 혹은 unblocked thread가 priority를 고려하지 않고 무조건 ready queue의 맨 뒤에 삽입된다. Priority scheduling 기능을 구현하면 thread의 priority 값을 고려하여 new thread의 priority가 current thread의 priority 보다 높은 경우 즉시 new thread에 CPU를 yield 시켜줄 수 있어야 하고, 그렇지 않은 경우에는 ready queue에 기존 방식대로 thread를 삽입하되 ready list 안에서는 thread 들이 priority가 높은 순서대로 정렬되어 있어야 한다. Priority는 0부터 63까지의 값을 가질 수 있으며 default 값은 31 이고, 숫자의 값이 클수록 높은 priority 이다. Priority scheduling 기능을 사용하는 경우 priority 값이 낮은 thread는 계속 순위에서 밀려 영원히 실행되지 않는 문제가 발생할 수 있으므로 ‘aging’ 기법을 함께 사용해줘야 한다. Aging 이란 thread가 ready queue에 있는 시간이 길어질수록 priority 값을 점차 증가시키는 기법이다. thread\_prior\_aging 변수를 사용하여 구현할 수 있다.

3) Advanced Scheduler  
마지막으로 **Advanced Scheduler(BSD Scheduler)** 기능을 구현해야 한다. BSD Scheduler 란 mutil-level feedback queue 또는 multi-level ready queue를 사용하는 scheduler로 Priority 에 따라 여러 개의 Ready Queue가 존재하며 각 ready queue는 round robin policy를 따른다. BSD Scheduler를 사용하면 average response time이 줄어든다는 장점이 있다.

* 1. **개발 내용**

1) Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술하시오.  
앞서 이야기했듯이 blocked 상태의 thread들을 저장하는 queue를 만들어 보관할 수 있으며, timer가 지날 때 마다 queue에 저장되어 있는 thread들의 wake up time을 검사하여 wake up 이 필요한 thread는 현재 queue에서 꺼내 ready queue로 옮겨 주면 된다. 즉 blocked 된 thread를 ready queue로 옮기는 작업이 해당 thread를 깨우는 것이라고 할 수 있다.

2) Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술하시오.  
현재 running thread의 CPU를 더 높은 priority를 가진 new thread에게 yield 해야 한다. running thead의 실행을 중단시키고 ready list에 넣은 다음, ready list에서 new thread를 꺼내 실행시키면 된다.

3) Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술하시오.  
nice, recent\_cpu, load\_avg 의 세가지 주요 변수가 필요하며 thread\_set\_priority, thread\_get\_nice, thrad\_set\_nice, thrad\_get\_recent\_cpu, thead\_get\_load\_avg 등의 함수를 작성해야 한다.

먼저 ***nice*** 변수는 -20 에서 20 범위 사이의 값을 가지며 nice 값이 양수일 때 이는 해당 thread의 priority 값을 감소시킨다. Thread가 처음 생성되었을 때 nice 값은 0으로 초기화되며, parent thread가 있는 경우 parent thread의 nice 값을 inherit 받는다.  
current thread의 nice value를 return 하는 ***thread\_get\_nice*** 함수와, 인자로 받은 nice 값을 current thread의 nice value로 설정한 다음, 해당 thread의 priority를 다시 계산하는 ***thread\_set\_nice*** 함수를 작성해야 한다.

recent\_cpu 변수는 특정 thread가 소모한 CPU time을 나타내는 변수로, 더 recent 한 CPU time이 덜 recent 한 CPU Time보다 더 크게 측정된다. Thred가 처음 생성되었을 때 recent\_cpu 값은 0으로 초기화 되고, parent thread가 있다면 그 값을 inherit 받는다. Time interrupt 가 invoke 될 때 마다 running state에 놓여있는 thread의 recent\_cpu 값은 1씩 증가한다. Thread가 어느 상태(running/ready/blocked)에 있던 간에 상관없이 매 초 마다 모든 thread의 recent\_cpu 값은 아래와 같은 식을 통해 갱신된다.

current thread의 recent\_cpu에 100을 곱한 값을 return 해주는 ***thred\_get\_recent\_cpu*** 함수를 작성해야 한다.

load\_avg 변수는 ready 상태에 있는 thread의 개수의 평균을 나타내는 변수이다.  
전역변수로 선언해서 사용해야 하며 system이 처음 boot 되었을 때 0 으로 값이 초기화된다. load\_avg 값 역시 매 초마다 값이 아래와 같은 식을 통해 갱신된다.

ready\_threads 는 ready 또는 running 상태에 있는 thred의 개수를 나타내는 변수이다.

current system의 load\_avg 값에 100을 곱한 값을 return 해주는 ***thread\_get\_load\_avg*** 함수를 작성해야 한다.

위에서 설명한 세가지 변수를 이용해 priority 값을 계산할 수 있다. 계산 식은 아래와 같다.  
PRI\_MAX는 thread priority가 가질 수 있는(0부터 63사이의 값) 최댓값인 63이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

**11/15~11/20** : Alarm Clock 기능 구현

**11/21~11/27** : Priority Scheduling 기능 구현

**11/28~12/05** : Advanced Scheduler 기능 구현 + 보고서 작성

* 1. **개발 방법**

1) Alarm Clock

1. **threads/thread.h** : 먼저 thread가 blocked 상태에 얼마 동안 놓여있어야 하는지를 저장하기 위해 thread 구조체에 정수형 필드 wakeuptime을 추가한다.
2. **threads/thread.c** : 추가한 wakeuptime 필드는 thread.c의 thread\_init 함수에서 값을 0으로 초기화 해줘야 한다.
3. **devices/timer.c** : blocked state의 thread 들을 저장하는 blocked\_queue 리스트를 생성한 다음 timer\_init 함수에서 해당 리스트를 초기화 해주고, timer\_sleep 함수와 timer\_interrupt 함수를 수정하여 기존의 thread\_yield 방식을 thread\_block 방식으로 변환할 것이다. 자세한 것은 IV.B 제작내용 부분에서 살펴보도록 하자.

2) Priority Scheduling

1. **threads/thread.h** : Priority scheduling에서 Aging 기법 구현에 필요한 bool 타입 변수 thread\_prior\_aging을 선언해준다(강의자료 참고).
2. **threads/init.c** : thread.h에서 선언한 thread\_prior\_aging 변수에 true값을 할당하여 aging이 작동할 수 있도록 설정해준다(강의자료 참고).
3. **threads/thead.c** : thread.c 에서도 마찬가지로 bool 타입 변수 thread\_prior\_aging을 선언해준다(강의자료 참고).  
   그 다음 thread\_unblock 함수와 thread\_yield 함수에서 기존에 사용하던 list\_push\_back 함수를 list\_insert\_ordered 함수로 바꿔 새로운 thread가 ready\_list에 추가될 때 무조건 맨 뒤에 삽입되는게 아닌, priority 값에 따라서 우선순위의 순서대로 ready\_list에 추가될 수 있도록 해준다.  
   그리고 thread\_create 함수를 수정하여 새로 생성하는 thread의 priority가 현재 실행중인 thread의 priority보다 더 높은 경우 현재 thread의 CPU를 새 thread에 yield 할 수 있도록 처리해준다.  
   현재 thread의 priority값을 새로운 priority 값으로 수정하는데 필요한 thread\_set\_priority 함수를 완성해 주어야 한다. 현재 thread의 priority 값을 인수로 받은 값으로 변경해주는 간단한 내용만 구현하면 된다. 다만 새로 배정받은 priority의 값이 기존 값보다 더 작다면 thread\_yield 함수를 호출하여 다시 스케줄링 할 수 있도록 해 준다.

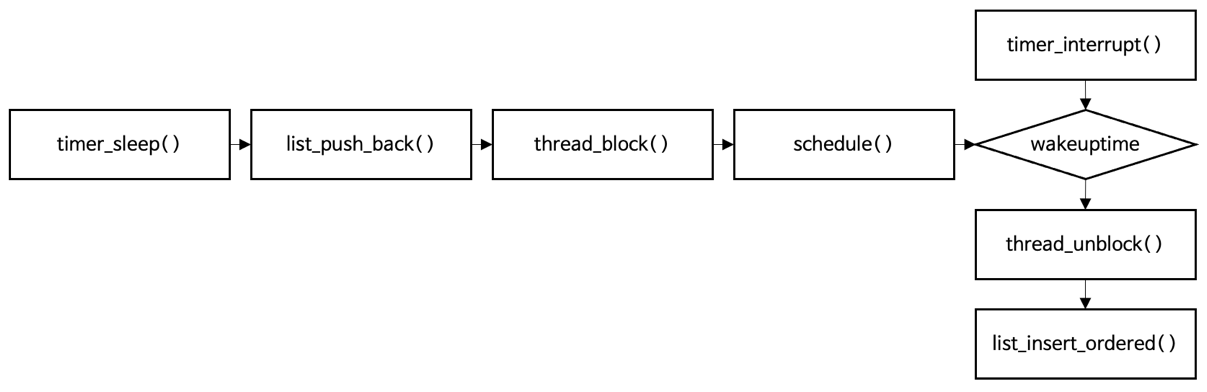
3) Advanced Scheduler

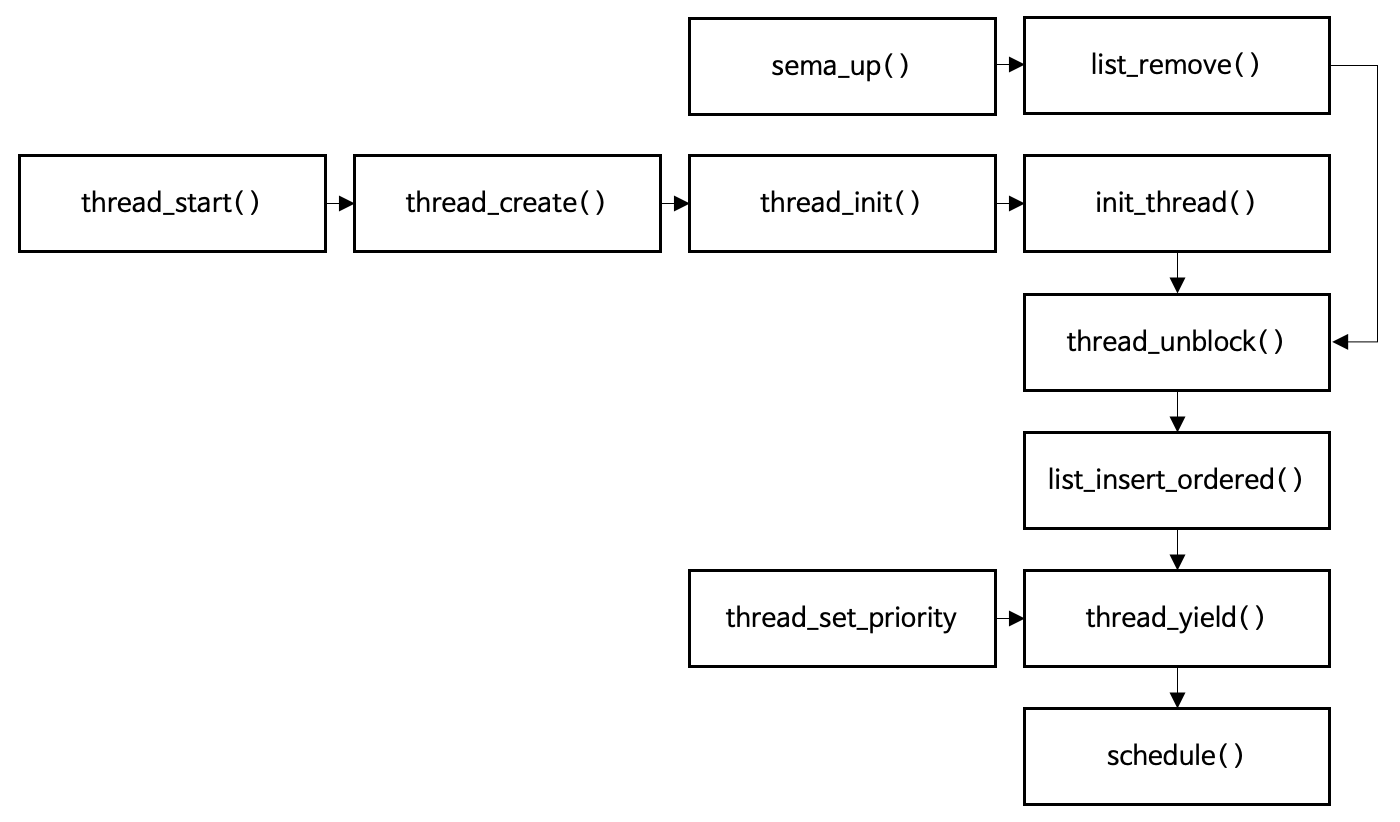
1. **threads/thread.h** : 먼저 BSD Scheduler 구현에 필요한 load\_avg 변수와 fixed-point real arithmetic 연산에 필요한 FRACTION 매크로를 thread.h 파일에 선언해준다. 그 다음 thread.h 파일의 thread 구조체 안에 정수형 필드 nice와 recent\_cpu를 추가해준다.
2. **threads/thread.c** : thread 구조체 안에 추가해준 nice, recent\_cpu 필드는 thread\_init 함수 안에서 값을 0으로 초기화해준다. parent\_thread가 있는 경우 parent\_thread의 nice, recent\_cpu 값을 inherit 받아야 하므로 그 부분을 init\_thread 함수 안에 구현해 주면 된다.

그 다음 앞에서 작성해야 한다고 이야기했던 함수들을 차례로 작성해 주어야 한다. 먼저 thread\_get\_nice 함수의 경우 current thread의 nice 값을 return 해주도록 작성하면 되고, thread\_set\_nice 함수의 경우 current thread의 nice 값을 인수로 받은 값으로 변경해주도록 작성하면 되므로 크게 복잡하지 않다. 단 thread\_set\_nice 함수에서는 nice 값이 변경 되었으므로 thread의 priority 값을 새로 계산해 주는 부분을 포함해야 한다. 이 때 위에서 살펴보았던 계산 식을 사용하면 된다.

다음으로 thread\_get\_recent\_cpu 함수와 thread\_get\_load\_avg의 경우, 각각 current thread 의 recent\_cpu 값과 load\_avg 값에 100을 곱한 값을 return 해주는 함수이므로 이 역시 작성하기 어렵지 않다. 다만 모든 연산에서 fixed-point real arithmetic을 사용해야 한다는 점을 유의해야 한다.

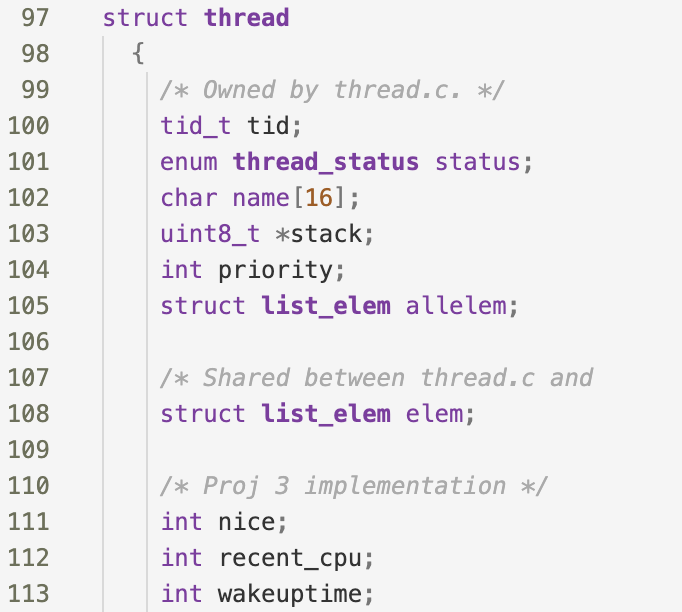
1. **devices/timer.c** : 이제 every tick 마다 호출되는 함수인 timer\_interrupt 함수 내부에 every tick 마다 thread의 priority 값을 갱신해주는 부분을 작성해 주어야 한다. 이를 위해 thread\_prior\_aging 또는 thread\_mlfqs가 true일 때 current thread의 nice, recent\_cpu, priority 값이 계산식에 따라 갱신될 수 있도록 작성해주면 된다.
2. **threads/synch.c** : 마지막으로 sema\_up 함수를 수정해서 priority에 sema\_up을 적용시킬 수 있도록 해준다. Sema를 기다리고 있는 thread list인 waiters에서 priority가 가장 큰 thread를 찾아 sema\_up 처리를 해준 다음, 변동사항을 바탕으로 scheduling 할 수 있도록 thread\_yield 함수를 호출해주면 된다.
3. **연구 결과**
   1. **Flow Chart**

1) Alarm Clock  


2) Priority Scheduling  
****

* 1. **제작 내용**

1) Alarm Clock

**< threads/thread.h >**  
thread 구조체에 정수형 필드 wakeuptime을 추가한다

**< threads/thread.c >**  
텍스트이(가) 표시된 사진

자동 생성된 설명  
thread.c의 thread\_init 함수에서 추가한 wakeuptime 필드값을 0으로 초기화 해준다.

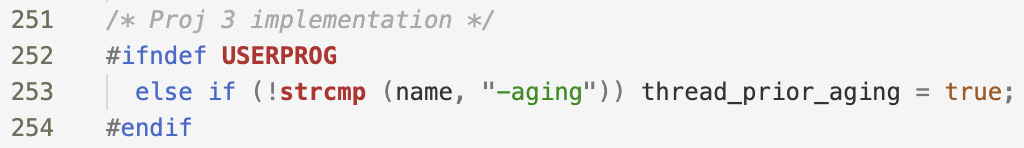
**< device/timer.c >**텍스트이(가) 표시된 사진

자동 생성된 설명  
blocked state의 thread들을 저장하기 위한 blocked\_queue를 선언해준다.  
텍스트이(가) 표시된 사진

자동 생성된 설명  
timer\_init 함수에서 blocked\_queue 리스트를 list\_init 함수를 이용해 초기화 한다.  
텍스트이(가) 표시된 사진

자동 생성된 설명  
timer\_sleep 함수에서 기존의 thread\_yiled( )를 호출하는 부분을 지우고, thread의 wakeup time을 설정하여 blocked\_queue에 삽입한 다음 thread\_block( )을 호출하도록 수정한다.

2) Priority Scheduling  
**< threads/thread.h >  
텍스트이(가) 표시된 사진

자동 생성된 설명**Aging 기법 구현에 필요한 bool 타입 변수 thread\_prior\_aging을 선언해준다. **< threads/init.c >  
**thread.h에서 선언한 thread\_prior\_aging 변수에 true값을 할당하여 aging이 작동할 수 있도록 설정해준다. **< threads/thread.c >**텍스트이(가) 표시된 사진

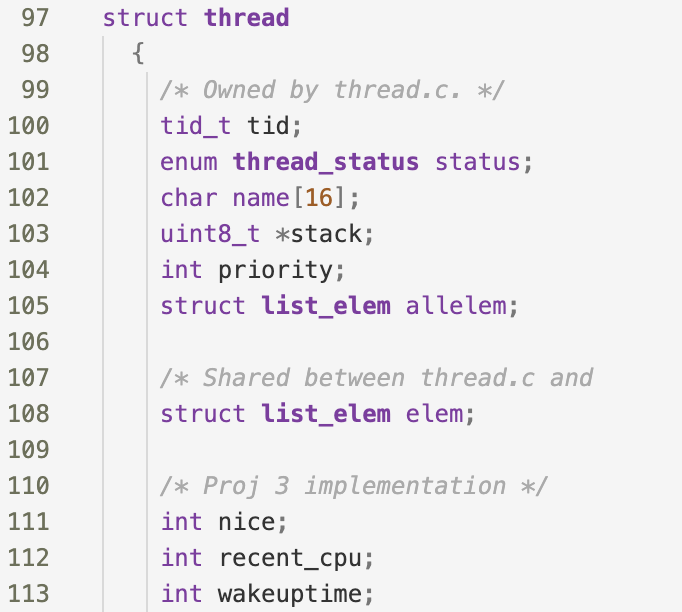
자동 생성된 설명thread.c 에서도 마찬가지로 bool 타입 변수 thread\_prior\_aging을 선언해준다.thread\_create 함수를 수정하여 새로 생성하는 thread의 priority가 현재 실행중인 thread의 priority보다 더 높은 경우 현재 thread의 CPU를 새 thread에 yield 할 수 있도록 처리해준다.텍스트이(가) 표시된 사진

자동 생성된 설명list\_insert\_ordered 함수 사용 시 필요한 비교 함수이다. 두 thread의 priority 값의 대소비교 결과를 return 하며 list\_insert\_ordered 함수의 인자로 넘겨 사용할 수 있다.  
텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명  
thread\_unblock 함수와 thread\_yield 함수에서 기존에 사용하던 list\_push\_back 함수를 list\_insert\_ordered 함수로 바꿔 새로운 thread가 ready\_list에 추가될 때 무조건 맨 뒤에 삽입되는게 아닌, priority 값에 따라서 우선순위의 순서대로 ready\_list에 추가될 수 있도록 해준다.

3) Advanced Scheduler  
**< threads/thread.h >  
텍스트이(가) 표시된 사진

자동 생성된 설명**BSD Scheduler 구현에 필요한 load\_avg 변수와 fixed-point real arithmetic 연산에 필요한 FRACTION 매크로를 thread.h 파일에 선언해준다. ****thread.h 파일의 thread 구조체 안에 정수형 필드 nice와 recent\_cpu를 추가해준다. **< threads/thread.c >**텍스트이(가) 표시된 사진

자동 생성된 설명thread 구조체의 nice, recent\_cpu 필드를 thread\_init 함수에서 0으로 값을 초기화해준다.  
**텍스트이(가) 표시된 사진

자동 생성된 설명**  
parent\_thread가 있는 경우 parent\_thread의 nice, recent\_cpu 값을 inherit 받아야 하므로  
그 부분을 init\_thread 함수 안에 작성해 주었다.텍스트이(가) 표시된 사진

자동 생성된 설명현재 thread의 priority값을 새로운 priority 값으로 수정하는데 필요한 thread\_set\_priority 함수이다. 현재 thread의 priority 값을 인수로 받은 값으로 변경해주며 새로 배정받은 priority의 값이 기존 값보다 작다면 thread\_yield 함수를 호출해 스케줄링 다시 수행한다.텍스트이(가) 표시된 사진

자동 생성된 설명current thread의 nice 값을 return 해주는 함수이다.텍스트이(가) 표시된 사진

자동 생성된 설명current thread의 nice 값을 인수로 받은 값으로 변경해주는 함수이다. Fixed-point real arithmetic을 사용해야 한다. nice 값이 변경 되었으므로 thread의 priority 값을 새로 계산해 주는 부분도 포함되어 있다.텍스트이(가) 표시된 사진

자동 생성된 설명각각 current thread 의 recent\_cpu 값과 load\_avg 값에 100을 곱한 값을 return 해주는 함수이다. Fixed-point real arithmetic을 사용해야 한다.텍스트이(가) 표시된 사진

자동 생성된 설명current thread의 nice, recent\_cpu 값을 계산식에 따라 새롭게 연산해주는 함수이다.텍스트이(가) 표시된 사진

자동 생성된 설명current thread의 priority 값을 계산식에 따라 새롭게 연산해주는 함수이다.  
위 두 함수는 timer.c 파일의 timer\_interrupt 함수에서 사용된다.  
 **< threads/timer.c >**텍스트이(가) 표시된 사진

자동 생성된 설명every tick 마다 thread의 priority 값을 갱신해주는 부분이다. thread\_prior\_aging 또는 thread\_mlfqs가 true일 때 current thread의 nice, recent\_cpu, priority 값이 계산식에 따라 갱신될 수 있도록 작성되었다.

**< threads/synch.c >**텍스트이(가) 표시된 사진

자동 생성된 설명priority에 sema\_up을 적용시킬 수 있도록 작성한 부분이다. Sema를 기다리고 있는 thread list인 waiters에서 priority가 가장 큰 thread를 찾아 sema\_up 처리를 해준 다음, 변동사항을 바탕으로 scheduling 할 수 있도록 thread\_yield 함수를 호출해 주었다.

* 1. **시험 및 평가 내용**

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

텍스트이(가) 표시된 사진

자동 생성된 설명  
priority-lifo.c 는 동일한 priority를 가지는 thread를 여러 개 생성하여 동일한 round-robin 순서 안에서 consistently run 시키는 코드이다. 총 16개의 thread가 lifo(last in first out)의 순서대로 각각 16번씩 반복 실행되어야 정상이다. 실행 결과를 보면 0부터 15까지의 thread 총 16개가 모두 16번 씩 동일한 round-robin 순서 안에서 실행되는 것을 확인할 수 있다.

2) make check 수행 결과  
텍스트, 신문이(가) 표시된 사진

자동 생성된 설명  
직접 test 해야 하는 priority-lifo를 제외한  
모든 21개의 평가에서 통과했음을 확인할 수 있다.