**Pintos Project 3: Threads**

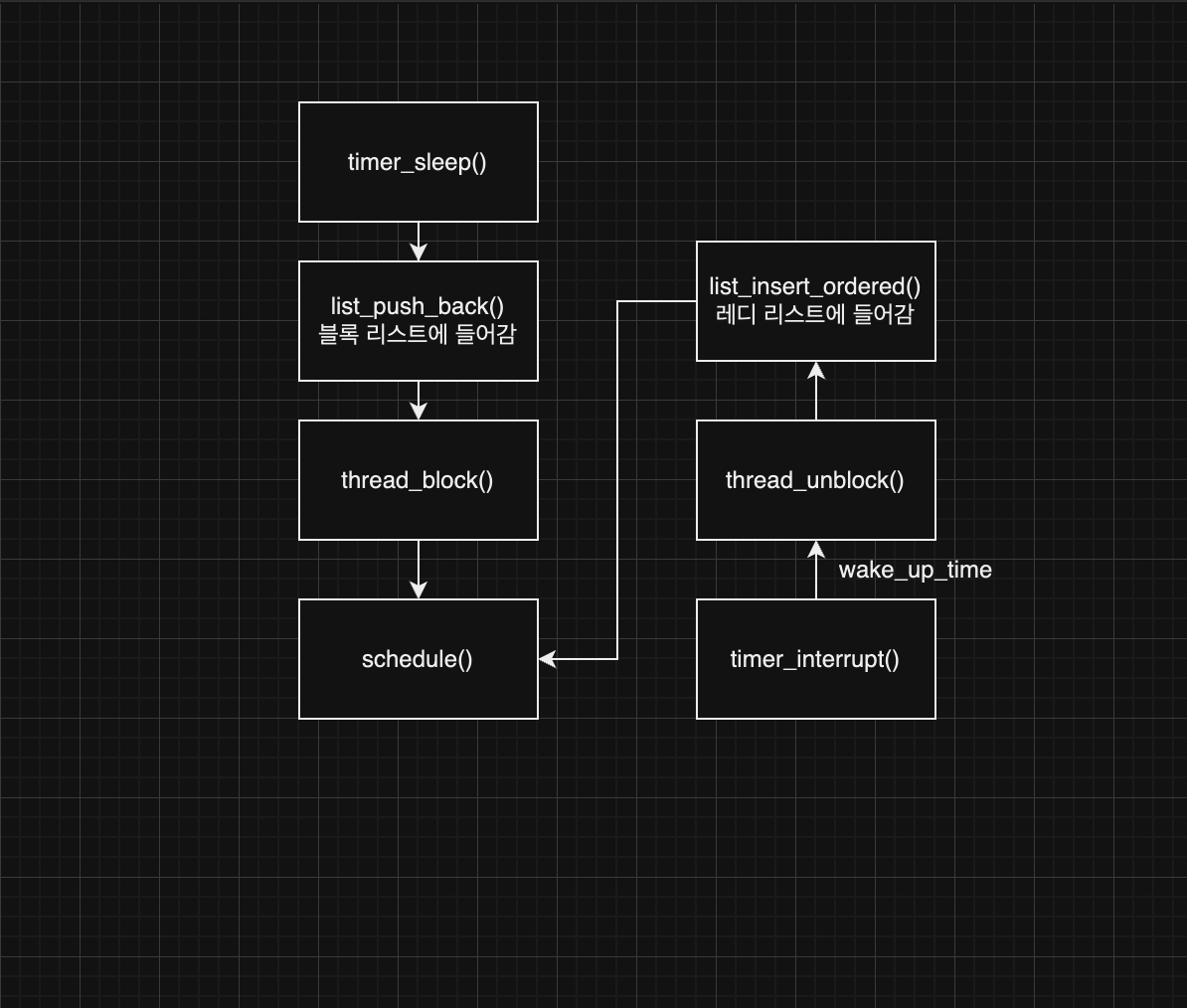
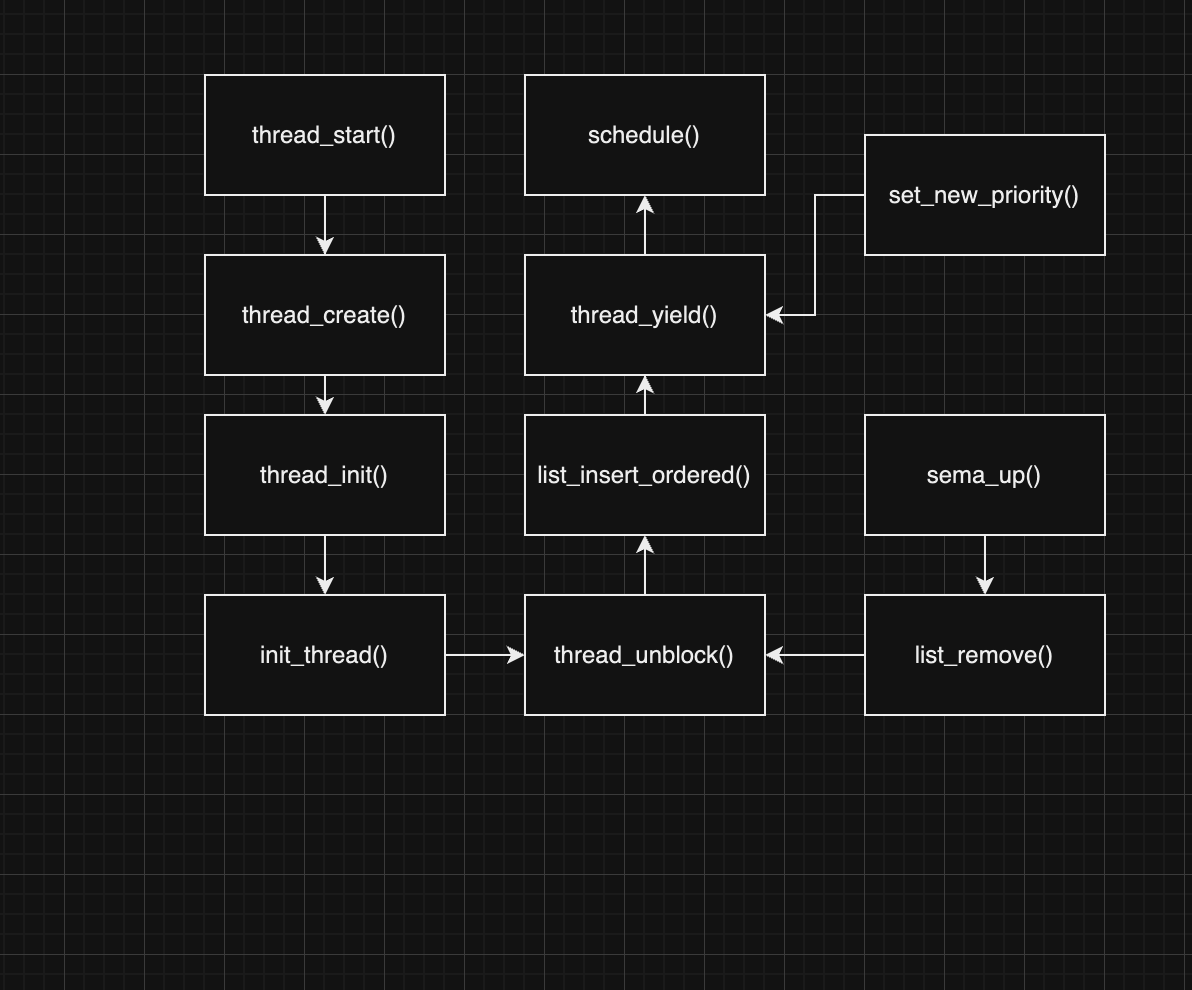
담당 교수 : 박성용 교수님

조 / 조원 : 20211531 나호영

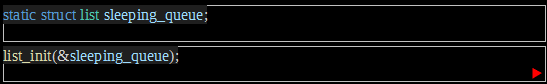
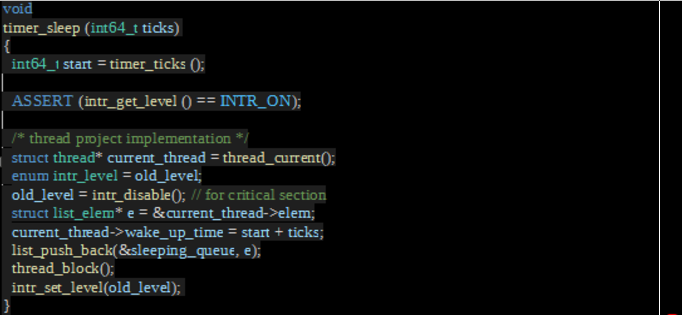
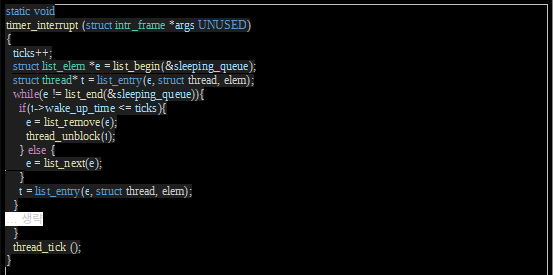
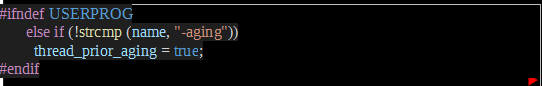
개발 기간 : 2023.11.5 ~ 2323.11.18

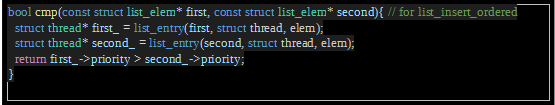
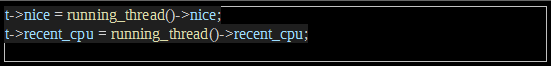
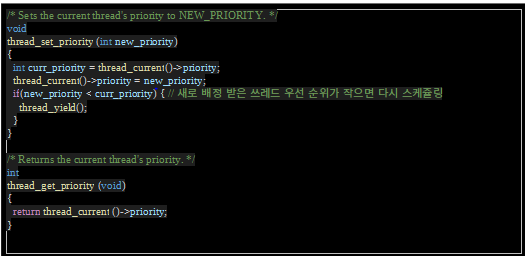
1. **개발 목표**

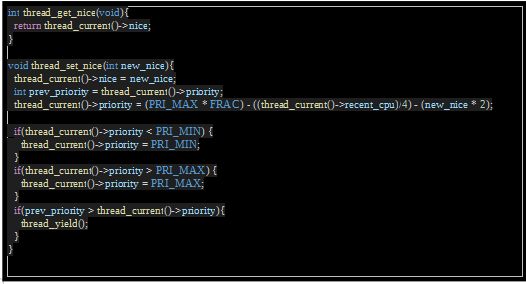
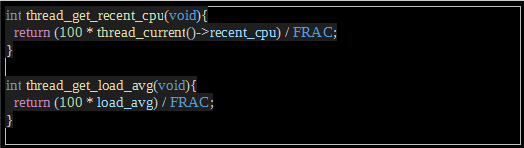
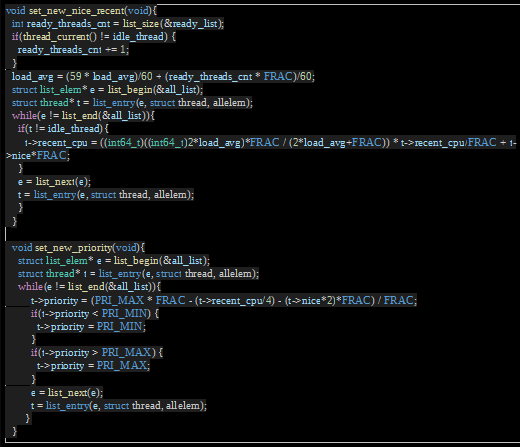
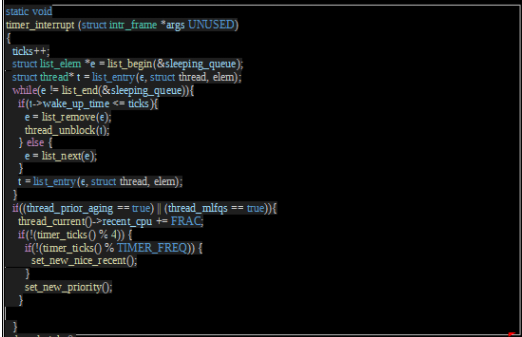
* 운영체제의 쓰레드 관리를 위한 Alarm Clock, Priority Scheduling과 추가적으로 Advanced Scheduler(BSD Scheduler)를 구현한다.

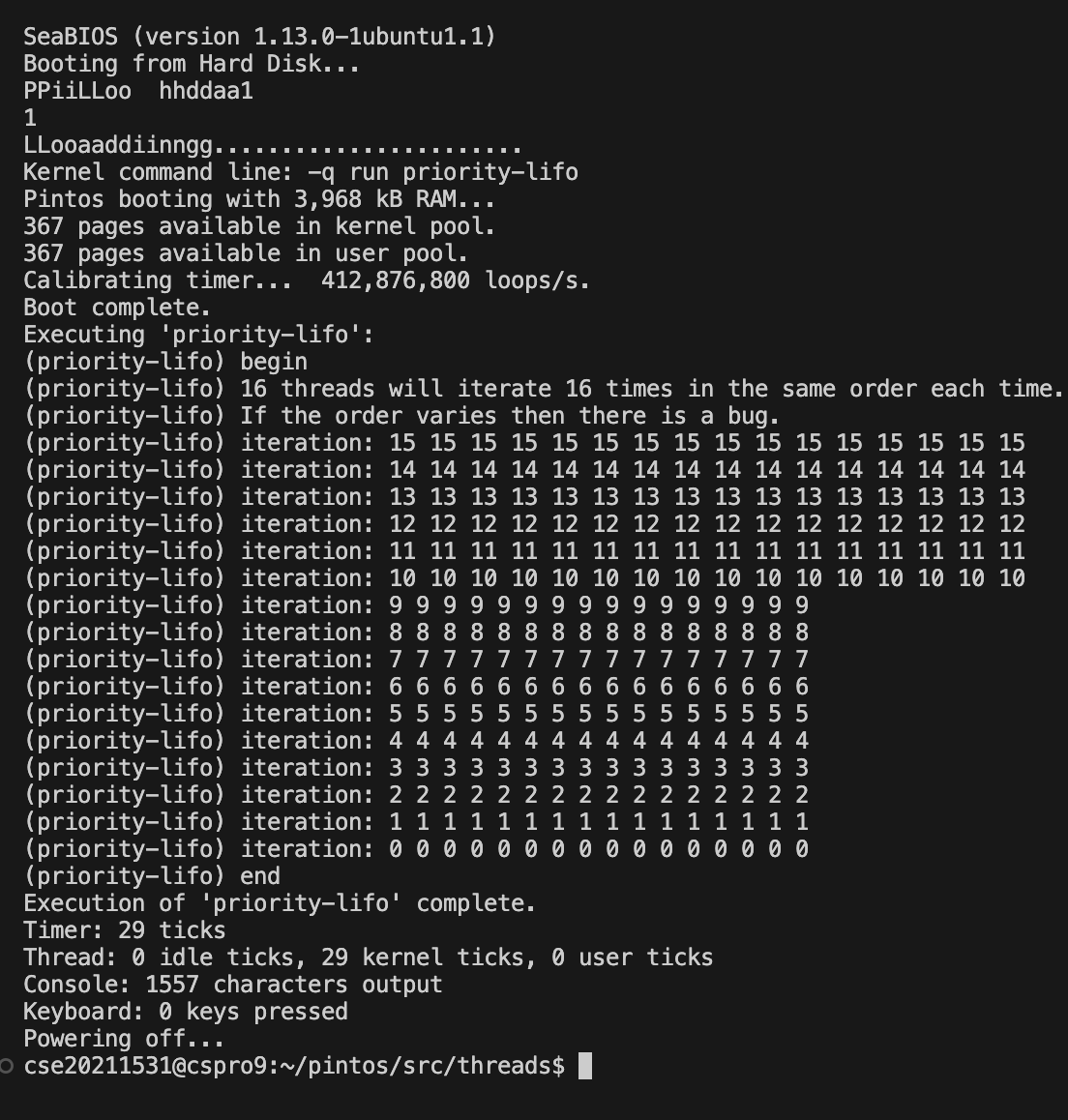
1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Alarm Clock  
      쓰레드가 현재 작동하지 않고 있으면 sleep 상태에 들어가는데 주어진 코드에서는 주어진 타이머 틱만큼 쓰레드를 sleep 상태로 유지하기 위해서 while 문을 통하여 할당된 시간이 지났는지 확인하는 방식을 사용하고 있다. 이 때 쓰레드는 Running 상태와 Ready 상태를 왔다갔다하는 비효율적인 동작을 수행하기 때문에 쓰레드를 Block하여 이를 개선해야 한다. 주어진 타이머 틱만큼 지나지 않은 쓰레드들은 Blocked 상태로 만들어 Block된 쓰레드들을 저장하는 블록 큐에 저장하고 주어진 타이머 틱만큼 시간이 지난 쓰레드들은 레디 큐로 이동시킨다.
   3. Priority Scheduling  
      주어진 코드에서는 Round-Robin 방식을 사용하여 thread\_yield/thread\_unblock 함수가 호출될 때 우선 순위가 고려되지 않고 레디 큐의 맨 뒤에 추가된다. 우선순위를 고려하는 스케쥴링을 구현하기 위해서는, 들어온 쓰레드의 우선순위를 체크하여 만약 작동하고 있는 현재 쓰레드보다 우선순위가 높다면 새로운 쓰레드에게 CPU를 양보해야 한다. 그렇지 않다면 레디 큐에 들어가야 하는데 이 때, 다음에 사용될 쓰레드를 선정하기 위해 레디 큐는 우선순위대로 정렬되어 있기 때문에 새로운 쓰레드의 우선순위에 맞추어 레디 큐에 추가하면 된다. 쓰레드가 레디 큐에 오랜 시간 동안 기다리고 있으면 해당 쓰레드의 우선 순위를 증가시켜주어 우선 순위가 낮은 쓰레드가 영원히 CPU를 얻지 못하는 Starvation을 해결하는 aging을 구현한다.
   4. Advanced Scheduler  
      BSD Scheduler는 MLFQ(multi-level-feedback-queue)나 MLRQ(mult-level-ready-queue)를 스케쥴링 방식으로 이용한 스케쥴러이다. 이를 구현하기 위해 우선 순위에 따라 레디 큐를 여러 개로 나누고 높은 우선 순위 큐에 존재하는 쓰레드들부터 CPU를 얻게 한다. 또한 큐 내부에서는 Round-robin 방식을 사용하게 한다.
   5. **개발 내용**
2. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.  
   sleep 중인 쓰레드들은 Block 상태에 있고 이 쓰레드들은 블록 큐에 저장되어 있다. 먼저 쓰레드를 sleep 시킬 때 깨어날 시간과 함께 블록 큐에 저장한다. 타이머 인터럽트가 발생하면 큐에 저장되어 있는 쓰레드들의 깨어날 시간을 체크하여 깨워야 하는 쓰레드들을 레디 큐로 옮겨주면 된다.
3. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.  
   현재 작동 중인 running 쓰레드는 자신보다 우선 순위가 높은 쓰레드가 들어오면 CPU를 들어온 쓰레드에게 양보시키고 레디 큐에 넣으면 된다.
4. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술.  
   먼저 이를 구현하기 위해 필요한 nice, recent\_cpu, load\_avg 요소들에 대해 설명하겠다. nice는 -20 ~ 20 까지의 값을 가지며 양수일 시 해당 쓰레드의 우선 순위를 감소시키고, 0일 시 우선 순위를 건드리지 않고, 음수일 시 우선 순위를 증가시킨다. recent\_cpu는 쓰레드가 사용한 CPU 시간을 나타내며 최근에 사용된 CPU 시간에 더 높은 가중치를 부여한다. 타이머 인터럽트가 발생하면 Running 상태의 쓰레드의 recent\_cpu 값은 1씩 증가되며 모든 쓰레드들은 매 초마다 recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1 ) \* recent\_cpu + nice 식을 통해 새로 계산된다. 마지막으로 recent\_cpu 계산식에 들어가있는 load\_avg는 시스템 전체의 로드 평균을 나타내는데 Ready 상태의 쓰레드들의 개수가 반영된다. 이 역시 마찬가지로 매 초마다 load\_avg = (59/60) \* load\_avg + (1/60) \* ready\_threads 식을 통하여 새로 계산된다. 위의 세가지 요소들을 사용하여 우선 순위가 계산된다. 우선 순위는 매 4틱마다 모든 쓰레드의 우선 순위가 priority = PRI\_MAX – (recent\_cpu / 4) - (nice \* 2) 식을 통하여 새로 계산된다.
5. **추진 일정 및 개발 방법**
   1. **추진 일정**11/5 ~ 11/8 : 프로젝트 이해 및 전체적인 구조 설계  
      11/9 ~ 11/14 : Alarm Clock, Priority Scheduling 구현  
      11/15 ~ 11/18 : Advanced Scheduler 구현 및 보고서 작성
   2. **개발 방법**  
      (1) Alarm Clock  
      - wake\_up\_time : 쓰레드가 깨어날 시간을 체크하기 위한 변수로 thread 구조체에 멤버로 추가한다. 이는 thread\_init 함수에서 0으로 초기화 된다.  
      - sleeping\_queue : 블록된 쓰레드들을 저장하기 위한 큐로 timer\_init 함수에서 초기화 된다. 또한 timer\_sleep 함수에서 블록해야 하는 쓰레드를 이 큐에 삽입한 뒤, thread\_block()을 호출하는 방식으로 변경한다. 또한 timer\_interrupt 함수에서 이 큐를 체크하여 깨워야 하는 쓰레드들을 다시 레디 큐에 넣는 코드를 추가한다.  
        
      (2) Priority Scheduling  
      - thread\_prior\_aging : Aging 을 구현하기 위해 필요한 변수로 thread.h와 thread.c에 추가한다. 또한 init.c에서 커널이 -aging 옵션과 함께 호출되었는지 검사하는 코드를 추가하여 만약 그렇다면 이 변수를 true로 변경한다.   
      - list\_inser\_ordered : 기존에는 RR 방식으로 그냥 맨 뒤에 넣는 list\_push\_back 함수 대신에 우선 순위에 따라 추가될 수 있도록 우선순위에 따라 레디 큐에 삽입하는 함수로 이를 thread.c에서 thread\_unblock 함수와 thread\_yield 함수에서 수정해준다. 또한 함수의 파라미터에 넣기 위해 cmp 함수를 작성해 thread.c에 추가한다.  
      - thread\_create 수정 : 새로 들어온 쓰레드의 우선 순위가 현재 쓰레드보다 높은 경우 해당 쓰레드에게 CPU를 yield 할 수 있게 수정한다.  
        
      (3) Advanced Scheduler  
      - nice, recent\_cpu : 우선 순위를 새로 계산하기 위해 필요한 변수들로, thread.h 파일에 thread 구조체 내부에 멤버로 추가한다. 또한 thread\_init 함수에서 0으로 초기화하는 코드를 추가한다. 만약 부모 쓰레드에 의해 생성된 쓰레드인 경우 부모 쓰레드(parent\_thread)의 nice와 recent\_cpu를 상속받을 수 있게 하는 코드를 추가한다.  
      - load\_avg : 시스템 로드 평균을 나타내는 변수로 thread.h 파일에 추가한다   
      - FRACTION : 부동 소수점 연산을 위해 필요한 매크로로, thread.h 파일에 추가한다.  
      - thread\_get\_nice : 현재 쓰레드의 nice 값을 반환하는 함수로 thread.c 파일에 추가한다.  
      - thread\_set\_nice : 현재 쓰레드의 nice 값을 파라미터로 받은 new\_nice로 새로 할당하는 함수로 - thread.c 파일에 추가한다. 이 함수에서는 nice가 변경되기 때문에 새로 우선순위를 계산해주는 코드를 추가한다.  
      - thread\_get\_recent\_cpu : 쓰레드의 recent\_cpu에 100을 곱한 값을 정수로 반올림하여 반환하는 함수로 thread.c에 추가한다.  
      - thread\_get\_load\_avg : load\_avg에 100을 곱한 값을 정수로 반올림하여 반환하는 함수로 thread.c에 추가한다.  
      - set\_new\_nice\_recent : nice와 recent\_cpu를 갱신하기 위해 호출되는 함수로 레디 큐의 쓰레드 수로 load\_avg를 갱신한 뒤 모든 쓰레드들에 대하여 recent\_cpu를 식에 따라 계산하여 갱신한다. thread.c 파일에 추가한다.  
      - set\_new\_priority : 우선 순위를 갱신하기 위해 호출되는 함수로 모든 쓰레드들에 대하여 식에 따라 계산하여 우선순위를 갱신한다. thread.c 파일에 추가한다.  
      - timer\_interrupt 수정 : 매 틱마다 새로 우선 순위를 계산하는 코드를 추가하여 수정한다. thread\_prior\_aging, thread\_mlfqs가 true 인 경우, set\_new\_nice\_recent 와 set\_new\_priority를 각각의 조건문을 추가하여 호출하는 코드를 추가한다.  
      - sema\_up 수정 : 세마포어가 up되기를 기다리고 있는 &sema->waiters 리스트에서 가장 큰 우선 순위를 가지는 쓰레드를 찾아서 해당 쓰레드를 대기 리스트에서 제거하고 thread\_unblock을 호출한다. 그 후 sema->value 값을 올린뒤 thread\_yield()를 호출하여 우선 순위의 역전을 방지한다.
6. **연구 결과**
   1. **Flow Chart  
        
      (1) Alarm Clock  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      (2) Priority Scheduling  
        
        
      **

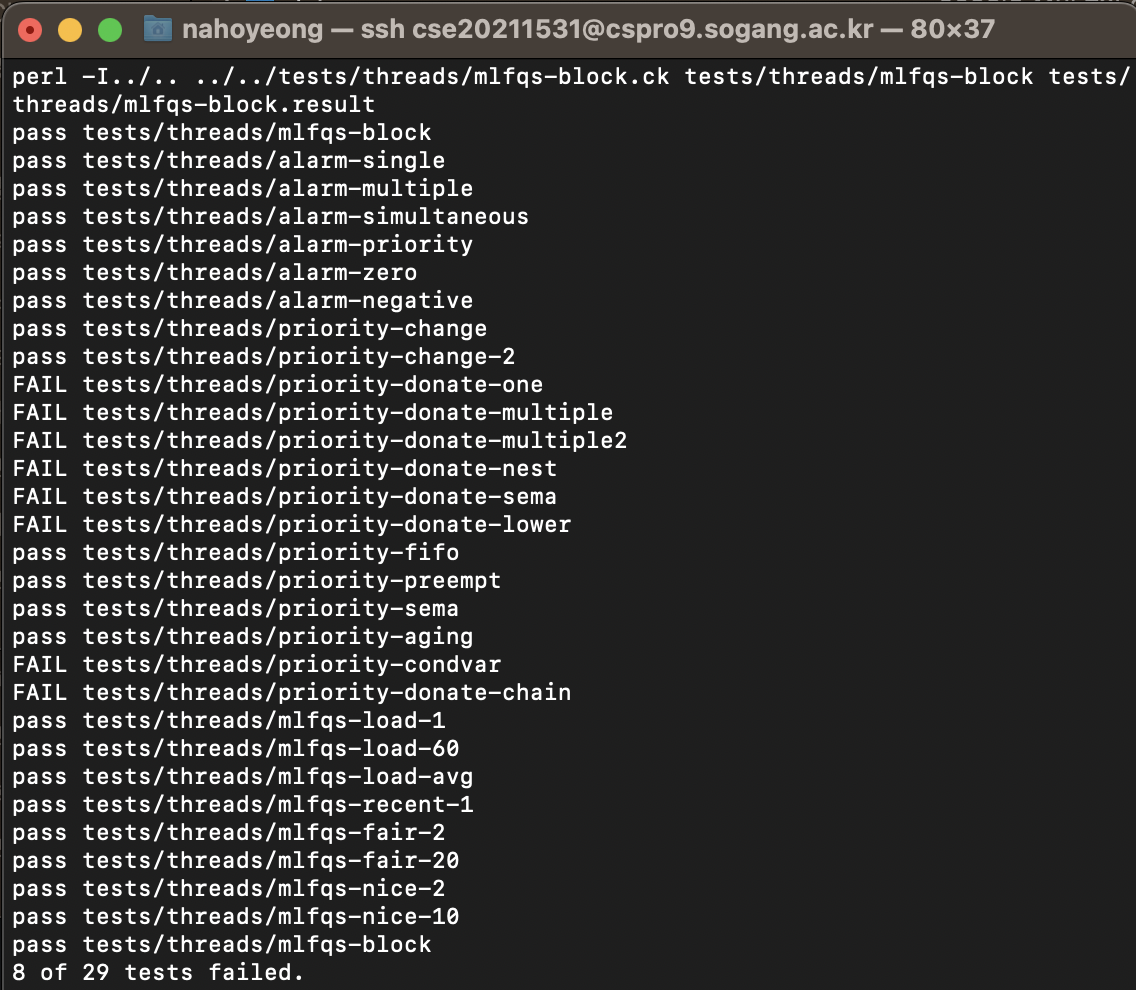
* 1. **제작 내용**

(1) Alarm Clock  
  
  
  
threads/thread.h 의 thread struct에 쓰레드가 일어날 시간을 의미하는 wake\_up\_time 멤버를 추가했다. 또한 threads/thread.c 의 thread\_init 함수에서 wake\_up\_time 을 0으로 초기화하는 코드를 추가했다.  
  
  
  
  
  
device/timer.c 에 전역 변수로 sleep하고 있는 쓰레드들을 저장하기 위한 sleeping\_queue를 추가했다. 또한device/timer.c 의 timer\_init 함수에 sleeping\_queue를 초기화하는 코드를 추가하였다.  
  
  
deveice/timer.c의 timer\_sleep 함수 내에서 기존의 비효율적인 방식을 바꾸었다. 쓰레드에 wake\_up\_time을 start 에 주어진 틱만큼 추가한 값을 할당하고 sleeping\_queue에 list\_push\_back 을 이용하여 블록 큐에 추가한 뒤 thread\_block()을 호출하였다.   
  
또한 timer\_interrput 함수에서 타이머 인터럽트가 발생할 때 블록 큐를 순회하며 wake\_up\_time이 지난 쓰레드들을 큐에서 제거하고 thread\_unblock을 호출에 러닝 큐에 넣는 코드를 추가하였다.  
  
  
  
(2) Priority Scheduling  
  
  
  
  
강의 자료에 설명되었던 대로 threads/thread.h에 Aging을 위해 필요한 플래그 변수 thread\_prior\_aging을 선언하였다. 또한 이 변수를 thread.c에서도 사용할 수 있게 thread.c에서는 bool thread\_prior\_aging; 과 같이 전역 변수로 추가해주었다.  
  
  
  
  
또한 threads/init.c에서 -aging 옵션과 함께 커널이 호출되면 thread\_prior\_aging 플래그를 true로 바꾸는 코드를 추가하였다.  
  
  
  
  
  
  
   
쓰레드가 만들어질 때 해당 쓰레드가 running 쓰레드보다 우선 순위가 높은 경우 CPU를 양보해야 한다.   
이를 위해 threads/thread.c에서 thread\_create 함수에 위 코드를 추가하였다.

  
  
  
  
위 첫번 째 코드는 threads/thread.c의 thread\_unblock 함수에서 기존에 사용하던 list\_push\_back을 대체한 코드이다. 이는 쓰레드가 unblock되면 레디 큐인 ready\_list로 들어가게 되는데 이 때 이 레디 큐를 항상 우선 순위를 기준으로 정렬시키기 위한 코드이다. 두 번째 코드는 threads/thread.c의 thread\_yield 함수에 들어가있는 부분인데 마찬가지로 list\_push\_back 대신에 list\_insert\_ordered를 사용하여 Running 쓰레드가 다시 레디 큐로 들어갈 때 우선 순위에 따라 들어갈 수 있게 한다. 파라미터로 들어가는 cmp 함수는 아래와 같이 구현하여 쓰레드의 멤버 priority를 확인해 이를 기준으로 정렬할 수 있게 한다.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
위는 threads/synch.c에 존재하는 sema\_up 함수이다. 세마포어에 락이 걸려 기다리고 있는 쓰레드들이 waiters에 있는데 이 때 가장 높은 우선순위를 가진 쓰레드가 다음에 세마포어를 걸 수 있게 해야한다. 따라서 위와같이 sema\_up 호출될 때 waiters에서 priority가 가장 높은 쓰레드를 찾고 리스트에서 제거한 뒤에 unblock 시키고 세마포어 값을 올려준뒤 thread\_yield를 호출하는 코드를 추가하였다.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
(3) Advanced Scheduler  
  
  
  
threads/thread.h 파일에서 fixed-point 산술 연산에 필요한 매크로를 FRAC으로 선언하였다. 이 때 공식에 사용되는 값인 (1<<14)를 선언하였다. 또한 시스템 로드 평균을 나타내기 위한 load\_avg 도 0으로 초기화하고 혼동을 방지하기 위하여 static으로 선언하였다.  
  
  
  
  
  
  
  
  
  
첫번 째 코드는 threads/thread.h 파일의 thread struct에 위와 같이 nice와 recent\_cpu 멤버를 추가한 부분이다. 이는 우선 순위를 계산할 시 사용된다. 두번 째 코드는 이를 threads/thread.c의 thread\_init에서 0으로 초기화해주는 코드이다. 세번 째 코드는 threads/thread.c의 init\_thread 함수 내에 있는 코드인데 이는 혹시 부모로부터 생성된 쓰레드인 경우 부모의 nice 와 recent\_cpu를 상속받기 위해 추가한 코드이다.  
  
여기서부턴 threads/thread.c에 새로 구현한 함수들에 대해 설명하겠다.  
  


위는 쓰레드의 멤버 priority에 대한 함수이다. thread\_set\_priority는 받은 우선 순위를 현재 쓰레드에 업데이트하고 우선 순위가 낮으면 스케쥴링을 다시 하는 함수이다. thread\_get\_priority는 현재 쓰레드의 우선순위를 반환한다.  
  
  
  
  
위는 쓰레드의 멤버 nice에 관한 함수이다. thread\_get\_nice 함수는 현재 쓰레드의 nice를 반환한다. thread\_set\_nice 함수는 현재 쓰레드에 새로운 nice를 업데이트하고 nice가 업데이트 되었으므로 새로 우선순위를 계산하여 업데이트하는 함수이다. 밑의 조건문은 계산 이후에 PRI\_MIN이나 PRI\_MAX 범위 밖을 벗어나는 경우를 제어한다. 또한 이전의 우선 순위가 새로 할당받은 우선순위보다 높으면 다시 스케쥴링을 시작하도록 thread\_yield를 호출하는 조건문을 추가하였다.  
  
  
  
  
위의 두 함수는 각각 현재 쓰레드의 recent\_cpu와 load\_avg를 반환하는 함수이다.   
  
  
  
  
  
  
  
  
  
  
  
  
  
위의 두 함수는 timer\_interrupt 함수에서 필요한 함수이다. set\_new\_nice\_recent 함수는 레디 큐에 존재하는 쓰레드들의 개수를 세고 현재 쓰레드가 idle\_thread가 아니라면 1을 더 추가한다. 이를 이용하여 새로운 load\_avg를 구한 뒤 모든 쓰레들에 대하여 recent\_cpu값을 초기화 하는 함수이다. set\_new\_prority는 모든 쓰레드들을 순회하며 새로 우선 순위를 업데이트하는 함수이다. 이어서 threads/timer.c 파일에 timer\_interrput 함수로 설명을 이어가겠다.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
위 함수는 timer inerrput 핸들인데, thread\_prior\_aging과 thread\_mlfqs를 체크하여 참일 경우, 다시 timer\_ticks 함수를 호출하여 틱이 4로 나누어질 경우 우선 순위를 새로 세팅하는 set\_new\_priority를 호출하고 나아가 100(TIMER\_FREQ)로 나누어질 경우 set\_new\_nice\_recent() 를 호출하여 새로운 recent\_cpu를 업데이트하도록 하는 코드를 추가하였다.

* 1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석  
    
  priority-lifo.c 코드를 살펴보면 동일한 우선 순위를 가지는 쓰레드들이 Round-Robin 방식으로 잘 실행되는지 확인하기 위한 테스트 코드이다. 위의 테스트 결과를 분석해보면 총 16개의 쓰레드가 모두 16번씩 반복되어 Round-Robbin의 순서로 각 반복마다 잘 실행되는 것을 확인할 수 있다.
* make check 수행 결과를 캡처하여 첨부

  
  
요구되는 모든 테스트 케이스를 통과한 것을 확인할 수 있다.