**Pintos Project 3: Threads**

담당 교수 : 문의현 교수님

학번 / 이름 : 20181264/ 노영현

개발 기간 : 11/28 ~ 12/5

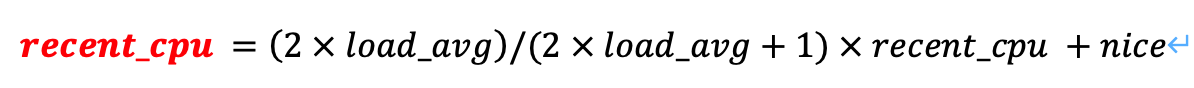
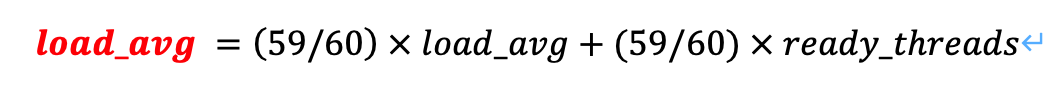
1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

이번 프로젝트3 threads 프로젝트에서는 프로세스를 cpu에 옮기고 실행시키는 과정에서 필요한 thread관리를 개발한다. thread관리를 위해서 Alarm clock, Priority scheduling, BSD scheduler 기능을 개발한다. 위 3가지 기능으로 thread관리에 성공한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

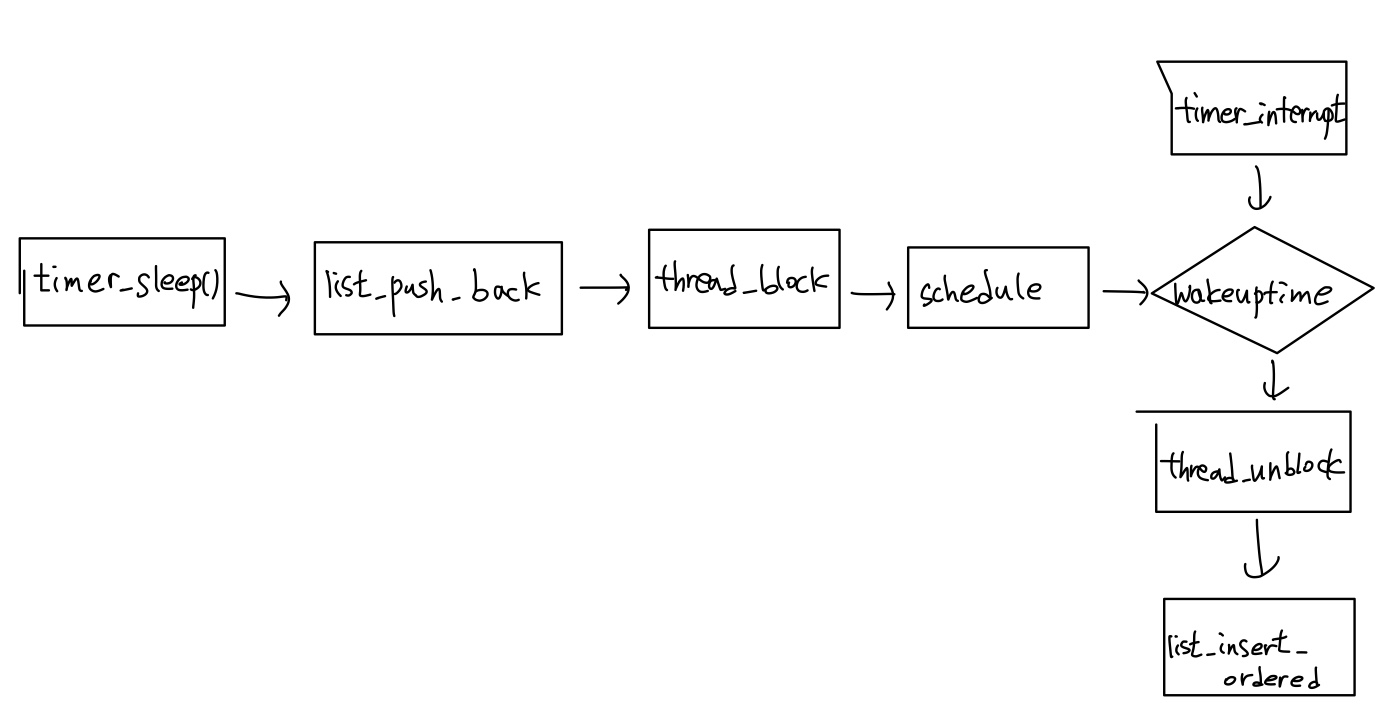
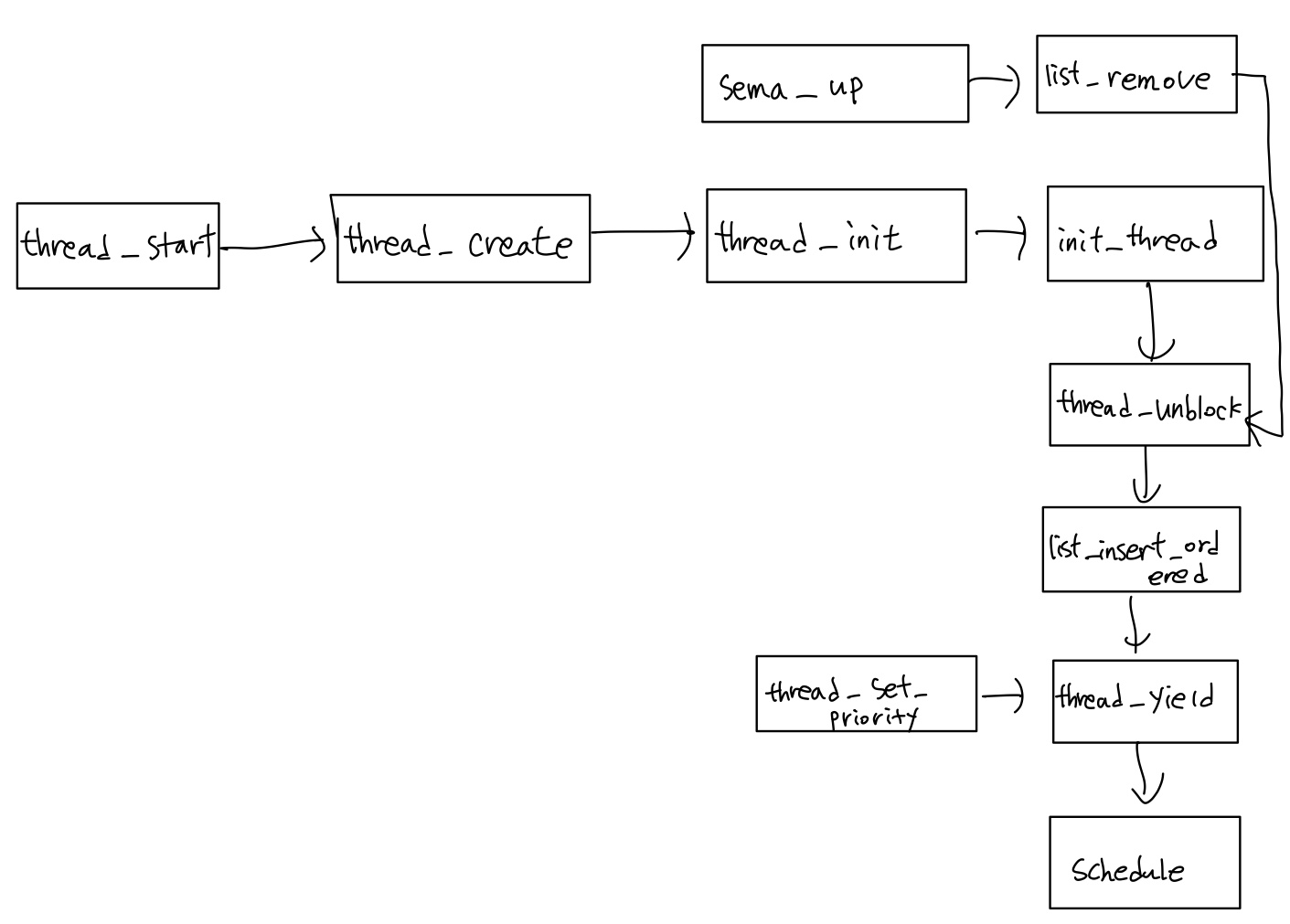
* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock  
     threads는 running state와 ready state 2가지 상태로 구분될 수 있다. thread가 사용되지 않아 sleep 상태에 있을 때 ready state에 있게 되고, 사용되고 있을 때 running state에 있게 된다. 현재 pintos에서 원래 구현되어 있는 코드 상으로는 정해진 시간 동안 thread를 sleep 시키기 위해서 계속 정해진 시간이 되었는 지를 while문을 사용하여 확인한다. 그렇게 되면 while문 내부에서 thread는 계속 ready state와 running state를 iterate하게 된다. 이는 굉장히 비효율적이기 때문에 시간이 지나지 않은 thread들을 blocked\_queue에 옮겨놓고, 시간이 지나면 ready\_queue로 옮겨주는 방식을 새로 개발하여 위 비효율성을 개선할 수 있다.
  2. Priority Scheduling  
     thread가 yield되거나 unblock되어 ready\_queue에 들어가는 경우 현재는 priority를 전혀 고려하지 않고 실행되고 있다. 현재는 우선순위가 높은 thread가 새로 생성되어도 실행되지 않고, thread들이 실행되는 순서들에도 우선순위가 전혀 고려되어 있지 않다. 이를 해결하기 위해서 priority scheduling을 개발한다. 우선 thread priority의 default값은 31이고, 최소값은0 최대값은 63을 가진다. 여기서 우선순위는 priority값과 비례한다. 우선 thread가 yield됐을 때 새로운 thread의 우선순위가 현재 사용되고 있는 thread의 우선순위보다 높은 경우 즉시 사용되게 한다. ready queue에 넣어줄 때에는 우선순위를 반영하여 순서에 맞게 ready queue에 넣어준다. 마지막으로 우선순위가 낮은 thread는 ready queue에 굉장히 오래 있는 starvation 현상을 해결하기 위해서 aging 기법 또한 개발한다. aging 기법이란 시간이 지날수록 ready queue에서 기다리고 있는 thread들의 우선순위가 높아지는 것을 의미한다.
  3. Advanced Scheduler (추가구현을 한 경우)  
     BSD scheduler란 multi-level feedback queue를 사용한 스케쥴러이다. BSE scheduler 는 여러 개의 큐를 사용하여 thread들을 scheduling한다. 각각의 큐들을 Round Robin 알고리즘을 따라서 큐들을 스케쥴링 한다. 큐들 사이에도 우선 순위가 존재한다. multi-level feedback queue를 사용할 경우 ready\_queue를 하나만 사용할 때 보다 thread들이 기다리는 시간이 줄어 효율성이 훨씬 올라가게 된다.
  4. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.  
   sleep 상태에 있는 thread들은 block\_queue에 저장되어 있다. 시간이 지날 때마다 block\_queue에 저장되어 있는 thread들의 고유한 wtime을 검사하여 만약 해당 thread를 깨워야 한다면 block\_queue에서 꺼내 ready\_queue에 넣어준다.
2. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.  
   현재 running되고 있는 thread보다 높은 priority를 가진 thread가 들어온 다면 현재 running 되고 있는 thread를 중단 시키고 다시 ready\_queue에 넣어준다. 그리고 새로 들어온 thread를 바로 run 시켜준다. 위 과정은 새로운 thread가 들어오자마자 진행된다.
3. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)  
   우선 순위를 계산 하기 위해서는 nice, load\_avg, recent\_cpu 3개의 변수가 필요하다. 우선 nice 변수의 경우, -20~20 사이의 값을 가지며 양수일 때와 음수일 때에 따라서 영향을 미치는 범위가 달라진다. 양수일 때는 thread의 priority값을 감소시킨다. nice의 default값은 0이며, 만약 parent thread가 있을 때는 parent thread의 nice값을 가져온다.   
   recent\_cpu 변수는 해당 thread가 얼마나 cpu를 사용하였는지를 나타내는 변수이다. recent\_cpu또한 default값은 0이며, 만약 parent thread가 있을 때는 parent thread의 recent\_cpu 값을 가져온다. time interrupt가 발생할 때마다 현재 thread의 recent\_cpu값이 1씩 늘어나는 방식으로 thread가 얼마나 cpu를 사용하였는지를 계산한다. 또한 time interrupt가 발생할 때마다 존재하는 모든 thread들의 recent\_cpu값이 아래 식을 통하여 갱신된다.  
     
   마지막으로 load\_avg변수는 평균적으로 몇개의 thread가 ready상태에 있는지를 나타내는 변수이다. OS가 시작될 때 0으로 초기화 되며, 평균적인 값을 나타내는 것이므로 매초 아래 식을 통해서 갱신된다.  
     
   위 식에서 ready\_Threads는 현재 ready상태에 있는 thread의 개수를 의미한다.   
   위 3개의 변수를 전부 구했다면 마지막으로 priority를 아래 식을 통하여 계산한다.  
   텍스트이(가) 표시된 사진

   자동 생성된 설명
4. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성  
  11/28 ~ 11/29 : Alarm Clock  
  11/30 ~ 12/2 : Priority Scheduling  
  12/3 ~ 12/4 : Advanced Scheduler  
  12/5 : 버그 수정 및 보고서 작성
  1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명  
  - threads/thread.h : 각각의 고유한 thread가 얼마나 오래동안 block\_queue에 담겨져 있어야 하는지를 나타내는 wtime 변수롤 새로 추가한다. 또한 priority scheduling에서 aging 기법이 작동할지 안할지를 결정해주는 thread\_prior\_aging 변수를 선언해준다. 또한 BSD scheduler 구현에 필요한 load\_avg변수와 FRACTION을 선언해준다. 또한 thread 구조체 내부에 BSD scheduler 구현에 필요한 nice 변수와 recent\_cpu 변수를 추가한다.  
  - threads/thread.c : thread\_init 함수에서 wtime을 0으로 초기화 해준다. 또한 thread\_init 함수에서 nice, recent\_cpu 값 또한 0으로 초기화 해준다. 두 변수는 parent thread 가 있는 경우 값을 0으로 초기화 하는 것이 아니라 부모 쓰레드에게 상속 받는데 이를 init\_thread함수에서 구현한다. thread\_unblock함수와 thread\_yield 함수에서 thread를 ready\_list에 추가할 때 무조건 push\_back 하는 것이 아닌 순서에 맞게 push 될 수 있도록 list\_insert\_ordered 함수를 사용해서 push 해준다. 또한 thread\_create 함수에서 thread가 생성될 때 만약 현재 running 하고 있는 thread보다 우선 순위가 높을 경우 바로 실행될 수 있도록 함수를 수정한다. 또한 thread의 priority를 업데이트 하는 thread\_Set\_priority 함수를 완성해준다. thread\_get\_nice, thread\_set\_nice 함수는 각각 current thread의 값을 return 되도록 함수를 작성한다. thread\_get\_recent\_cpu와 thread\_get\_load\_avg 함수는 위에서 작성해놓은 수식에 따라서 함수를 작성한다.
* devices/timer.c : blocked 상태의 thread들을 저장하는 bqueue 리스트를 선언한다. timer\_init함수에서 list\_init함수를 사용해서 리스트를 초기화한다. timer.c 파일에서는 기존에 thread를 yield하는 방식을 blocked큐에 넣고 이를 확인해 ready\_queue에 넣는 방식으로 변경한다. time\_interrupt 함수 내부에서 매 초 thread들의 priority값을 갱신하는 코드를 작성한다. priority 갱신은 nice, recent\_cpu 를 활용하여 갱신해준다.
* threads/synch.c : sema를 기다리고 있는 thread들 중 priority가 가장 큰 thread를 찾아 sema\_up 처리를 해준다.

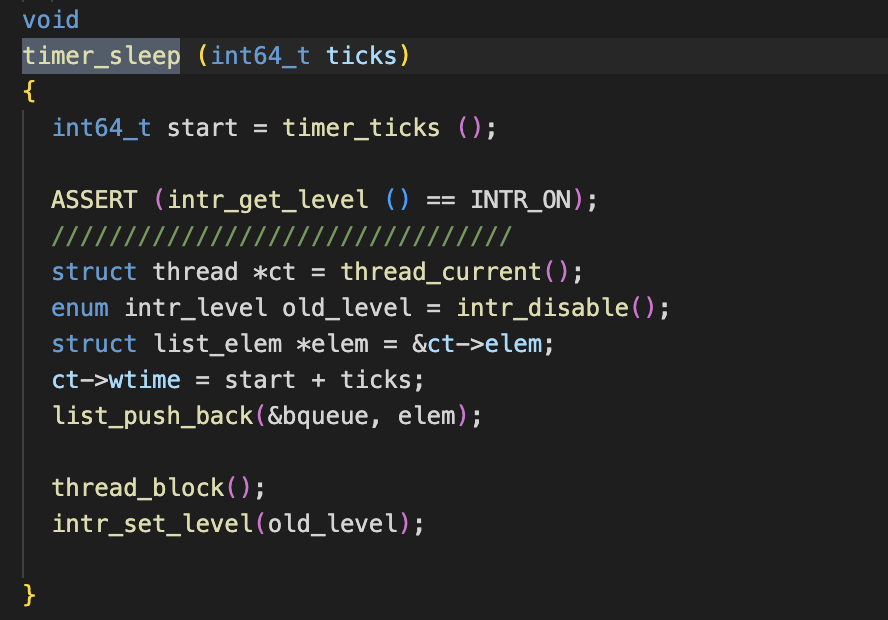
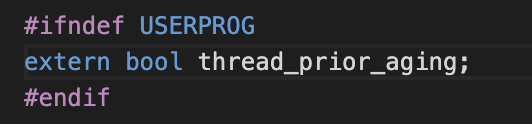
1. **연구 결과**
   1. **Flow Chart**

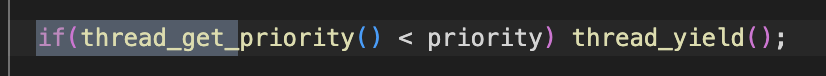
* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)  
  <aging>  
    
  <priority, BSD scheduler>  
  
  1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명  
  1) ALARM CLOCK  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  thread.h 파일의 thread 구조체 내부에 wtime이라는 변수를 추가한다. wtime에 대한 설명은 3-B 에 있다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  thread\_init 함수에서 현재 thread의 wtime값을 0으로 초기화해준다.  
  텍스트이(가) 표시된 사진

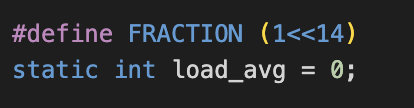
  자동 생성된 설명  
  timer.c에서 blocked state의 thread들을 저장하는 bqueue를 선언한다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  timer\_init 함수에서 bqueue를 list\_init함수를 이용해서 초기화한다.  
    
  timer\_sleep 함수에서 현재 thread의 wtime을 새로 설정해준 뒤, blocked queue에 삽입한 다음 thread\_block() 함수를 호출해 alarm 방식을 변경해준다.  
  2) Priority Scheduling  
    
  thread.h 내부에서 thread\_prior\_aging 함수를 선언한다.  
  텍스트, 표지판, 닫기이(가) 표시된 사진

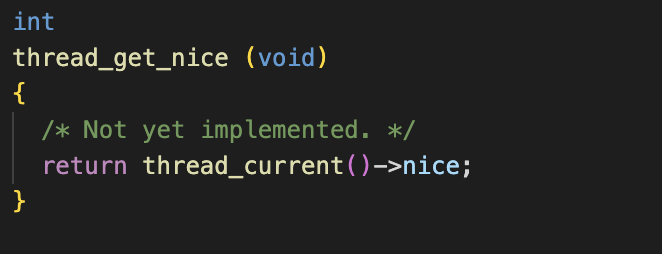
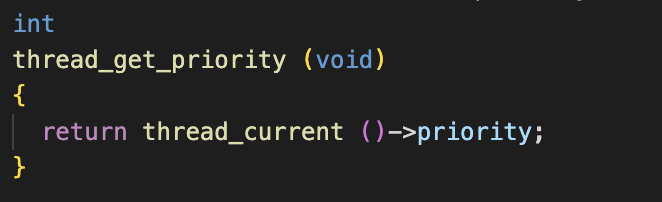
  자동 생성된 설명  
  init.c 파일에서 name에 aging이 있을 경우 thread\_prior\_aging 변수에 true값을 할당한다.  
    
  thread create 함수에서 thread가 생성될 때 현재 run 되고 있는 thread보다 우선순위가 더 높다면 바로 해당 thread를 실행시키는 코드를 추가한다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  struct list\_elem 2개를 인자로 받아 2개의 priority를 비교하는 함수를 cmp함수를 작성한다. 이 함수는 scheduling을 기반으로 ready\_queue를 만들 때 사용된다. 즉, list\_insert\_ordered 함수의 인자로 사용된다.   
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  thread\_unblock 함수와 thread\_yield 함수에서 기존에 사용하던 list\_push\_back함수를 지우고 list\_insert\_ordered 함수를 사용하여 ready\_list에 들어가는 element들이 priority에 따라서 정렬된 상태로 들어가게 만든다.  
  3) Advanced scheduler  
    
  thread.h 파일에 FRACTION 매크로와 load\_avg를 선언한다. FRACTION 매크로는 fixed-point real arithmetic 연산에 사용되는 매크로로 Advanced Scheduler에서 필요한 연산 시에 계속 사용된다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  struct therad 내부에 BSD scheduler 구현을 위해 필요한 nice 변수와 recent\_cpu를 선언한다. 선언한 두 변수는 thread\_init 함수에서 0으로 초기화 해주며, init\_thread 함수에서 만약 parent thread 가 있을 경우 nice와 recent\_cpu 값을 parent thread로부터 inherit 받는 코드를 구현한다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  thread\_set\_priority 함수를 작성한다. thread\_Set\_priority함수는 인자로 받은 new\_priority로 현재 thread의 priority 값을 변경한다. 또한 새로 배정받은 priority 값이 원래 존재하던 priority 값보다 작은 경우 thread\_yield함수를 호출해 scheduling 을 다시 해준다.  
    
    
  thread\_get\_nice 함수는 현재 thread의 nice값을 return 하는 함수이다. thread\_get\_priority함수는 현재 thread의 priority값을 return 하는 함수이다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  thread\_set\_nic함수는 current thread의 nice값을 갱신해주고, 이에 따른 priority 또한 새로 갱신해주는 함수이다. 이 과정에서 FRACTION을 사용하여 Fixed-point real arithmetic 연산이 포함되어 있다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  thread\_get\_load\_avg 함수와 thread\_get\_recent\_cpu 함수는 각각 load avg 와 recent cpu를 반환 하는 함수이다. 여기서 또한 FRACTION을 활용한 Fixed-point real arithmetic을 사용한다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  current\_thread의 nice, recent\_cpu 값을 업데이트 해주는 함수이다. for문을 활용하여 all\_list에 있는 모든 element들을 참조하여 필요한 element의 값들을 업데이트 해준다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  위에서 설명한 n\_nice\_recent\_cpu 함수와 마찬가지로 n\_priority함수는 priority값을 업데이트 해주는 함수이다. 마찬가지로 for문을 활용하여 all\_list에 있는 모든 element들을 참조하여 필요한 element의 값들을 업데이트 해준다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  timer\_interrupt는 기본적으로 매 time\_tik 마다 실행되는 함수이다. time\_interrupt함수를 사용하여 매 time\_tik 마다 nice, recent\_cpu, priority 값이 갱신된다. 위 갱신하는 과정은 작성한 n\_nice\_recent\_cpu함수와 n\_priority함수를 사용하여 갱신한다.  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  sema에서 기다리고 있는 모든 thread들을 for문을 활용하여 참조하면서 그 중 가장 priority가 가장 큰 thread를 찾는다. 해당 thread를 sema\_up 해준 뒤, thread yield함수를 호출한다.  
  1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  priority\_lifo 명령어는 여러 개의 thread를 동일한 round-robin 알고리즘 안에서 실행되는 코드이다. 0~15 번의 thread가 각각 전부 16번씩 반복해서 실행되면 개발이 잘 되었다고 볼 수 있다. 위 실행결과를 보면 전부 16번씩 잘 실행되고 있음을 볼 수 있다.
* make check 수행 결과를 캡처하여 첨부  
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  PASS 해야 하는 항목들에서 전부 PASS 했음을 확인할 수 있다.