**Pintos Project 3: Threads**

담당 교수 : 김영재 교수님 / 1분반

조 / 조원 : 컴퓨터공학과 20211507 김동건

개발 기간 : 2024년 11월 18일 ~ 2024년 11월 24일

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술  
  지금의 스케쥴러는 기존엔 레지스터를 계속 바꾸어 끼우는 라운드 로빈 방식으로 되어 있는데, 이를 alarm clock을 이용하여 효율적으로 구현한다. 또, 우선순위 스케쥴링과 MLFQ를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock  
     기존엔 Running, ready state를 계속 왕복하는 Busy waiting 방식으로 되어 있어 CPU가 쓸데없이 바쁘다. 따라서 상당히 비효율적인데, 이를 tick과 sleep\_list를 이용하여 효율을 높일 것이다.
  2. Priority Scheduling  
     기존 Pintos에서는 round-robin scheduling을 사용하고 있다. 따라서 우선순위가 고려되지 않는다. 이번 프로젝트에서는 스레드들에 priority를 부여하여 높은 것 먼저 실행되게 구현할 것이다. 그러나 그런 방식을 도입하면, 우선순위가 낮은 스레드들은 CPU에 배정받지 못하는 기아 현상이 발생하기 때문에 시간에 비례하여 우선순위가 증가하는 thread\_aging을 구현하여 이를 막을 것이다.
  3. Advanced Scheduler (추가구현을 한 경우)  
     BSD Scheduler를 구현한다. 이 과정에서 부동소수점 연산을 지원하지 않아, 필요한 몇 가지 함수들을 추가하여 이를 지원하게 바꿀 것이다. 또한, 이 과정에서 필요한 nice value, load\_avg 등 다양한 값을 추가하여 구현한다. 실제 MLFQ에서는 64개의 큐를 이용하지만, 1개의 큐를 이용하면서도 우선순위, nice, 최근 cpu 사용 시간 등의 변수를 이용하여 MLFQ와 유사하게 스케쥴링 될 수 있도록 구현한다.
  4. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.  
   Blocked 상태의 스레드들을 관리하는 sleep\_list를 만들고, tick이 커짐에 따라 list를 순회하며 깨울 시간이 된 스레드들이 있는지 확인한다. 물론, list에 삽입할 때 깨울 시간을 기록한다. 깨울 시간이 된 스레드들은 ready\_list에 넣어준다.
2. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.  
   Ready list에 실행중인 스레드보다 더 높은 스레드가 들어오면, 실행중이던 스레드를 yield한다. 기존 스레드는 ready\_list 맨 뒤에 삽입하는 것이 아닌, 정렬된 위치에 삽입한다. 새로 들어온 스레드가 더 높은 스레드라면 그 스레드가 맨 앞에 위치할 것이고, 그 스레드가 실행될 것이다.
3. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)  
   \* recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1 ) \* recent\_cpu + nice : 각 스레드가 최근 사용한 CPU 시간을 의미한다.  
   \* load\_avg = (59/60) \* load\_avg + (1/60) \* ready\_threads : ready state에 있는 thread의 평균 개수이다.   
   \* nice : -20 ~ 20의 값을 가진다. 이는 priority와는 다르게 값이 작을수록 더 우선순위가 높다.   
   \* priority = PRI\_MAX - (recent\_cpu/4) - (nice \* 2): 우선순위를 나타내는 변수이다. 최근 CPU 사용 시간과 nice값이 클수록 우선순위가 낮아지는 것을 확인할 수 있다.
4. **추진 일정 및 개발 방법**
   1. **추진 일정**

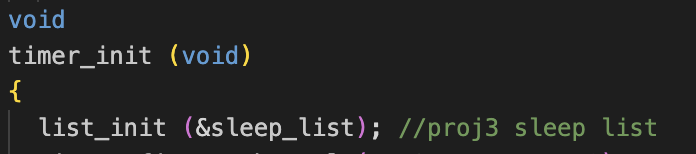
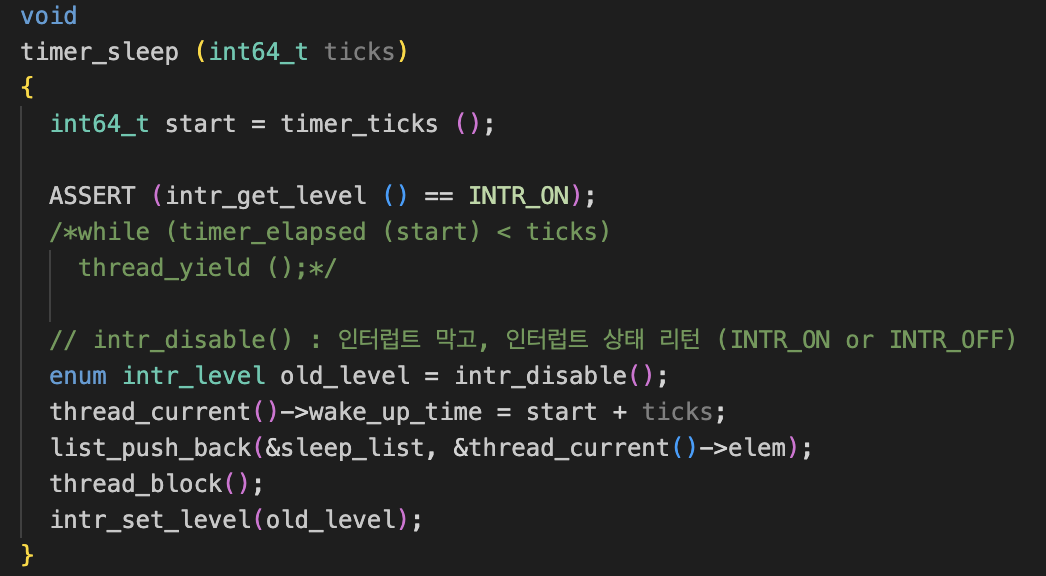
* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성  
  24.11.18 ~ 24.11.18 : 과제 명세서 및 pintos 메뉴얼 공부   
  24.11.19 ~ 24.11.23 : Alarm Clock + Priority Sheduling + Aging 구현   
  24.11.24 ~ 24.11.24 : Advanced Scheduler 구현 + 보고서 작성
  1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수  
      
    II. B.의 1번 항목인 Block 상태인 스레드를 깨우기 위해 sleep\_list를 추가해 주어야 한다. 이제, 현재 실행중인 스레드를 sleep\_list에 추가할 때, wake\_up\_time을 기록하기 위해 timer\_sleep함수 수정이 필요하다. 또한, timer\_interrupt에서 tick을 증가시키기 전에 증가시키면 깨워야 할 스레드를 깨워주고, tick을 증가시킨다. wake\_up\_time은 thread 구조체에 추가시켜야 한다.  
      
    다음으로, II. B.의 2번 항목인 우선순위가 더 높은 스레드가 들어왔을 때를 위해 구현할 것을 살펴보자. thread\_yield에서 현재 실행중인 스레드를 정렬된 순서로 ready\_list에 삽입한다. 또, seam\_up 함수에서 리스트를 정렬하는 코드를 추가한다. 이때 list\_sort 정렬 함수에 넘길 비교 함수를 구현해준다.  
      
    마지막으로, II. B.의 3번 항목인 Advanced Scheduler를 구현하기 위해서 thread에 nice, recent\_cpu 변수 2개를 추가하고, load\_avg 변수를 thread.c에 전역으로 선언한다. 제일 높은 우선순위를 찾는 함수, load\_avg값과 recent\_cpu값을 update하는 함수, 우선순위를 update하는 함수를 작성한다. 추가한 값들은 init 과정에서 적절히 초기화해준다. 마지막으로, 기존에 구현되어 있지 않았던 thread\_get\_load\_avg, thread\_get\_recent\_cpu, thread\_get\_nice, thread\_set\_nice 함수를 구현한다.

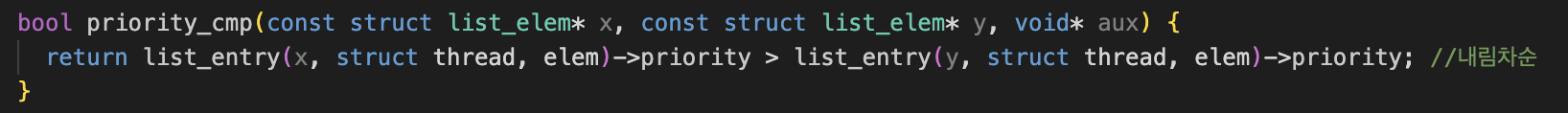
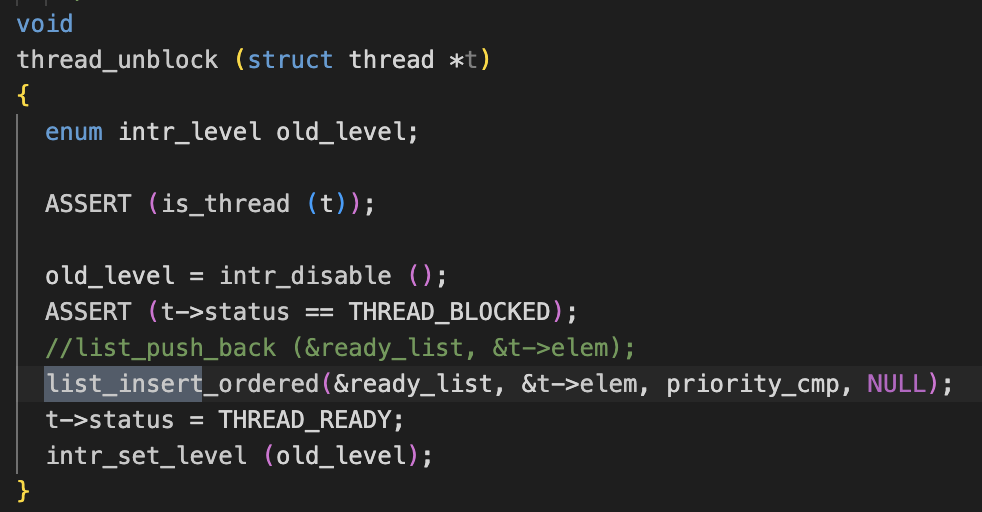
1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)  
    
  Alarm Clock  
  도표, 기술 도면, 평면도, 라인이(가) 표시된 사진

  자동 생성된 설명  
    
    
  Priority Schedule  
  도표, 스케치, 그림, 기술 도면이(가) 표시된 사진

  자동 생성된 설명
  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  
    
  먼저 Alarm Clock을 위한 구현을 살펴보자.  
  텍스트, 폰트, 친필, 스크린샷이(가) 표시된 사진

  자동 생성된 설명timer.c에 sleep되어 있는 list를 관리하는 sleep\_list 자료구조를 선언하였고, 이를 timer\_init에서 초기화 해준다. 또한, thread 구조체 안에 wake\_up\_time을 추가하여 준다.  
    
  기존 while로 구현된 busy waiting을 sleep\_list를 이용하여 바꾸었다. 인터럽트 중에 인터럽트를 막은 뒤, sleep할 스레드의 wake\_up\_time을 기록한 후, sleep\_list에 추가한다. 그리고 스레드를 block후, intr\_set\_level을 통해 상태를 다시 돌려놓는다.
* 텍스트, 소프트웨어, 멀티미디어 소프트웨어, 폰트이(가) 표시된 사진

  자동 생성된 설명  
  timer\_interrupt에서 tick을 증가시키기 전, tick이 증가되면 깨울 시간이 되는 스레드들을 리스트에서 제거하고 깨워준다. 위 과정에서 사용한 list 자료구조와 관련 함수들은 lib/kernel/list.c에 있다.  
    
  두 번째로 우선순위 스케쥴링에 대해 살펴보자.  
  이를 위해, 우선 과제 명세서에서 언급한 추가해야 할 코드들을 모두 추가하였다. aging을 체크하는 조건문을 추가하였고, bool 변수인 thread\_aging을 추가하였다. 또, 우선순위에 따라 정렬을 하는 데에 사용되는 비교함수를 추가하였다.   
    
  이는 thread/synch.c에 추가하였고, 우선순위 내림차순으로 리스트를 정렬하는 데에 필요한 정렬함수이다. 이제 위 정렬함수를 이용하여 기존의 push\_back으로 구현되어 있던 것들을 insert\_ordered로 바꾸어준다.  
    
  텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

  자동 생성된 설명  
  또, sema\_up을 하는 과정에서 thread를 unblock하기 전, 정렬을 하여 가장 우선순위가 큰 것이 unblock시킨 뒤, yield를 통해 양보할 수 있게 수정하였다. 이 과정에서 intr\_context일 때는 yield가 일어나지 못하게 조건문을 추가하였다.  
  텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

  자동 생성된 설명  
    
  세 번째로, Advanced Scheduler를 살펴보자.  
  텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

  자동 생성된 설명 폰트, 스크린샷, 텍스트, 그래픽이(가) 표시된 사진

  자동 생성된 설명  
  이를 구현하기 위한 변수를 thread 구조체에 추가하였고, load\_avg를 thread.c에 전역변수로 추가하였다. 다음으로는 이번 명세서에서 핀토스 커널은 부동소수점 연산을 지원하지 않아, 17.14 format 실수연산을 지원하는 코드를 작성해 주어야 한다고 하였다. 이를 위해 thread.h와 thread.c 에 필요한 연산이 있는 함수들을 모두 작성하여 주었다.  
  텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명  
    
  텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

  자동 생성된 설명  
    
  다음으로, 가장 큰 우선순위를 return하는 함수, load\_avg와 recent\_cpu를 업데이트 하는 함수, 우선순위를 업데이트 하는 함수를 작성하였다.  
  텍스트, 스크린샷, 폰트이(가) 표시된 사진

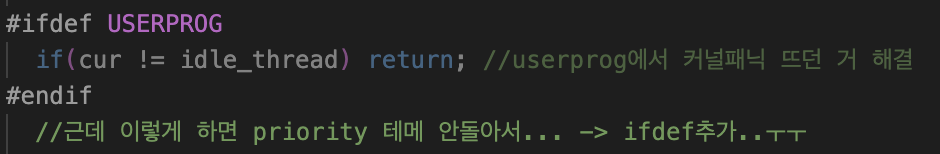
  자동 생성된 설명  
  ready\_list는 이전 구현에서 정렬된 상태로 되어있으므로, front를 리턴해주면 된다. 텍스트, 폰트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명  
  recent\_cpu와 load\_avg값은 명세서에 나와있는 수식을 그대로 구현해 주었다.   
  텍스트, 멀티미디어 소프트웨어, 소프트웨어, 스크린샷이(가) 표시된 사진

  자동 생성된 설명  
  priority update에서 update하는 것 역시 명세서에 나와있는 수식을 그대로 사용하여 구현하였고, 이 과정에서 범위를 벗어나지 않게 보정해 주었다. 그 후, 현재 스레드의 웅선순위보다 높은 우선순위를 가지는 스레드가 생겼다면, yield를 위한 함수를 호출해 주었다.  
    
  텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명  
  thread\_aging에서는 명세서에 나와있는 대로 TIMER\_FREQ마다 load\_avg, recent\_cpu를 업데이트 해줬고, 4틱마다 우선순위를 업데이트 해 주었다.  
    
  그리고, timer\_interrupt에서 aging 혹은 mlfq 테스트 케이스라면 thread\_aging을 호출하여 주었다.  
    
  텍스트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명또한, thread\_set\_nice, thread\_get\_nice, thread\_get\_load\_avg, thread\_get\_recent\_cpu 함수를 구현하였고, get함수들은 주석을 참고하여 값을 올바르게 계산하여 리턴하여 주었다. set\_nice의 경우 update\_priroity 함수와 비슷한 로직으로 작성하였는데, 현재 스레드의 nice를 인자로 받아 설정해주고, update\_priority와 같은 방식으로 priority 값을 정해주었다.  
  위의 구현에서 사용된 min, max는 thread.h의 상단에 매크로로 define하여 사용하였다.  
  텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

  자동 생성된 설명  
    
    
  마지막으로, 코드를 다 작성 후, thread 부분은 모두 잘 돌았지만, userprog에서 커널 패닉이 떴다. 따라서 이를 디버깅해 보니, thread\_yield에서 오류가 발생하였다. 그 중에서도 현재 스레드가 idle\_thread가 아닐 때 USERPROG에서 오류가 났다. 이 오류를 해결하기 위해 여러 로직을 바꾸어 보고 공부도 해보았지만 해결을 할 수 없어, 이를 방지하기 위한 코드를 아래와 같이 thread\_yield에 추가하였다.  
  
  1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석  
  텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명  
  priority-lifo.c 코드를 보면 위와 같다. thread\_create의 두 번째 인자가 priority인데, i가 점점 커지므로 우선순위가 낮은 스레드부터 생성이 된다. 그러나, 코드를 잘 작성하였다면 우선순위가 높은 스레드들부터 실행이 되어야 한다. 이제 실행 결과를 보자. 0 ~ 15의 우선순위를 생성하고, 15부터 실행되는 것이 의도된 답이다.  
  텍스트, 스크린샷, 컴퓨터, 소프트웨어이(가) 표시된 사진

  자동 생성된 설명결과를 보아도, 우선순위가 높은 스레드인 15부터 내림차순으로 실행되는 것을 확인할 수 있다.
* make check 수행 결과를 캡처하여 첨부  
  텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

  자동 생성된 설명스크린샷, 텍스트, 디스플레이, 컴퓨터이(가) 표시된 사진

  자동 생성된 설명이번 프로젝트3에서 요구하는 testcase들과 지난 프로젝트 1, 2에서 요구했던 testcase들 모두 통과하는 것을 확인할 수 있다.