**Pintos Project 3: Threads**

담당 교수 :박성용 교수님

조 / 조원 :20191597 서윤혁

개발 기간 :3주

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

핀토스는 기본적으로 RR 스케줄러로 구현되어 있다. 우리는 이걸 priority 스케줄러로 구현하게 된다. 이때 Alarm Clock과 추가적으로 BSD 스케줄러를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock

기존에는 timer\_sleep()함수에서 busy waiting 방식으로 구현되어 있어서 매우 비효율적이였다. 따라서 blocked state를 저장할 수 있는 queue를 추가해서 tick을 체크 후 시간이 지나면 blocked queue에 넣어준다. 그 후 wake up 시간을 설정해서 해당 시간이 지나면 다시 ready queue에 넣어준다.

* 1. Priority Scheduling

현재 핀토스는 RR 방식으로 스케줄러가 구현되어 있다. 따라서 새로운 thread가 yield나 unblock되었을 때 ready queue의 마지막으로 들어가게 된다. 즉 다시말해 priority가 전혀 고려되지 않았다.  
하지만 이제는 priority에 따라서 ready queue가 정렬되고 항상 우선순위가 높은 thread들이 RUNNING 상태가 된다. 하지만 이렇게 되면 우선순위가 낮은 쓰레드들은 항상 양보를 하게되어 실행될 기회가 없는 starvation 문제가 발생하는데 이를 해결하기위해 시간이 지남에 따라 우선순위가 높아지는 aging도 구현했다.

* 1. Advanced Scheduler (추가구현을 한 경우)

BSD는 priority aging을 할 때 우선순위를 매틱마다 그냥 올려주는 것이 아닌 계산식에 따라서 우선순위를 계산하게 된다.

이때 우선순위도 매 4틱마다 업데이트를해준다.

따라서 스케줄링을 할 때 여러 변수들을 같이 고려하여 우선순위를 산정하게 된다.

(recent\_cpu, load\_avg, nice 값들을 참조)

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

일단 blocked 상태인 스레드들을 관리할 blocked list를 만들어준다.

그리고 각 스레드에 특정시간이 지나면 깨우는 wake\_up\_time을 설정해준다.

Timer.c의 timer\_interrupt()는 매틱마다 호줄되는데 여기서 blocked 상태에 있는 모든 스레드들을 순회해주면서 wake\_up\_time과 tick 값을 비교해주면서 시간이 지난애들은 blocked list에서 제거하고 thread\_unblock() 함수를 호출해서 깨운다.

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

현재 실행중인 스레드보다 더 높은 우선순위의 스레드가 ready list로 들어오면 현재 스레드와 스위칭되게된다.

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

계산식

priority = PRI\_MAX – (recent\_cpu / 4) – (nice \* 2)

recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1) \* recent\_cpu + nice

load\_avg = (59/60)\*load\_avg+(1/60)\*ready threads

recent\_cpu: CPU time의 근사한 값이다. 초기값은 0이고 timer\_interrupt가 호출될 때마다 1씩 증가한다.

load\_avg: CPU 부하 평균으로 현재 시스템에서 실행 가능한 스레드의 수를 고려하여 계산합 값이다.

nice: 우선순위와 관련된 값으로 -20에서 20사이의 값을 가지고 default 값은 0이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11-08 ~ 11-09: alarm-clock, priority scheduling  
11-15 ~ 11-17: Advanced Scheduler(BSD), 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

1. Alarm Clock

Blocked 상태의 스레드들을 모아놓을 blocked\_list를 만들어주고 thread\_init()에서 초기화를 해준다.

그리고 timer.c의 timer\_sleep()에서 thread\_sleep()함수를 호출하는데  
thread\_sleep() 함수는 스레드를 block시키고 blocked\_list에 넣어준다. 이때 wake\_up\_time도 같이 설정해줌.

그리고나서 timer\_interrupt()에서는 check\_wake\_up\_thread() 함수를 호출하는데

이 함수는 매틱마다 wake\_up\_time과 tick을 비교하면서 시간이 지났는지 안지났는지 체크를하고 시간이 지난 스레드들은 ready\_list로 넣어준다.

1. Priority Scheduling

스레드들을 우선순위대로 ready\_list에 넣어줘야함.

그리고 thread를 실행시키는 모든곳에서 priority를 보도록 코드를 수정해야된다.

thread.c의 thread\_create(), thread\_yield(), thread\_unblock()에서 thread가 priority에 따라스케줄링 되도록 수정해준다.

Ready\_list에 우선순위로 정렬되도록 넣어야하므로 list\_insert\_ordered 함수를 사용할 것이고, 해당 파라미터에 정렬을 해줄 함수 인자로 priority\_scheduling()함수를 넣어줌.

Priority\_scheduling은 우선순위가 높은 애를 리턴하도록 구현했다.

Thread\_set\_priority(), thread\_get\_priority()를 통해 스레드의 우선순위를 수정하거나, 스레드의 우선순위를 가져올 수 있는 함수를 정의하였다.

Thread\_aging함수를 구현하여 매틱마다 스레드들의 우선순위를 높여주도록 하였다.

1. Advanced Scheduler

Fixed-point 계산을 위해 thread.c에 해당 계산을 하는 함수들을 추가해줬다.

Thread 구조체에 nice, recent\_cpu 값을 추가해주었고, thread.c에 static 변수로 load\_avg를 추가해주었다.

그리고 위에서 구현해주었던 thread\_aging함수를 재활용하였는데, 이제 priority를 계산식에 따라 계산하여 매틱마다 업데이트 하도록하였다.

우선 recent\_cpu, load\_avg, priority를 계산하기 위한 함수들을 각각 cal\_recent\_cpu, cal\_load\_avg, cal\_priority를 추가해주었고, 이를 이용해서 thread\_aging함수에서 priority를 계산한다.

그리고 기존에 있는 thread\_get\_nice(), thread\_set\_nice(), thread\_get\_recent\_cpu, thread\_get\_load\_avg()를 구현해준다.

1. **연구 결과**
   1. **Flow Chart**

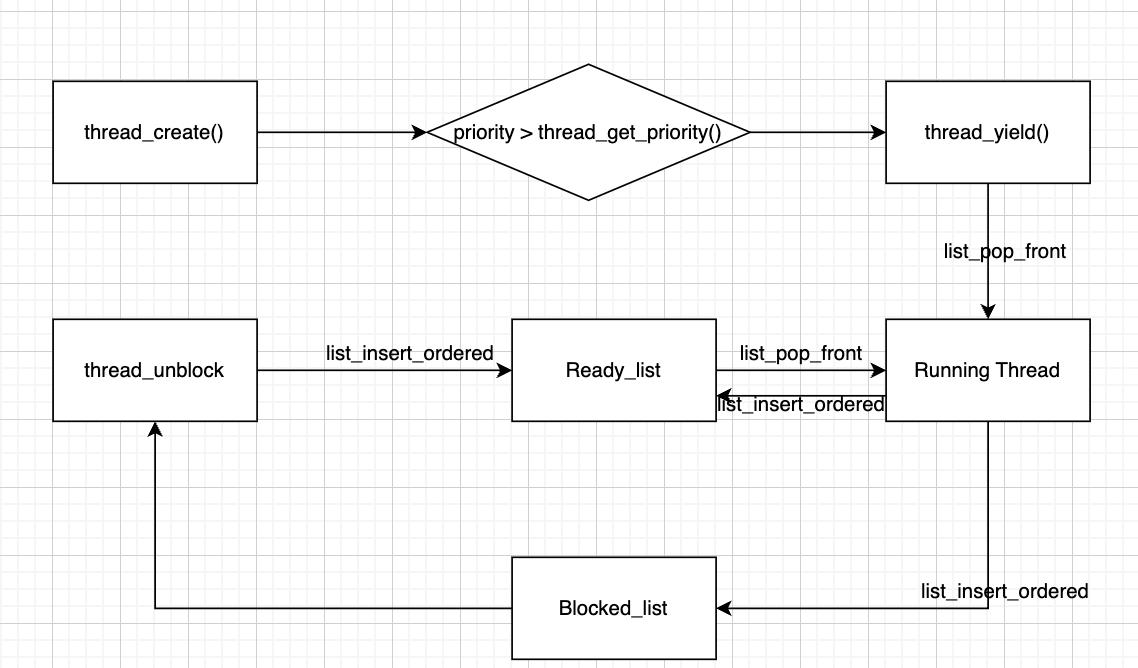
* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)

1. Alarm Clock

**도표, 라인, 그래프, 평면도이(가) 표시된 사진

자동 생성된 설명**

1. Priority Scheduling



* 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

일단 blocked된 쓰레드가 꺠어날 수 있도록 시간 설정을 하는 wake\_up\_time을 thread 구조체에 추가해주었다.

Timer.c에 timer\_sleep()함수에서 thread\_sleep()함수를 호출해서 스레드들을 block 상태로 관리하도록 하였다. 이를 위해서 blocked\_list 자료구조를 추가해서 여기에 block된 스레드들을 넣어주었다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그리고 나서 매틱마다 timer.c의 timer\_interrupt() 함수가 호출되는데 여기서 매번 wake\_up\_time과 tick을 비교해서 해당 함수가 다시 ready\_list로 들어갈 지 말지를 결정한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 코드가 구현한 함수인데, 여기서 tick보다 wake\_up\_time이 작아졌다는것은 wake\_up\_time의 시간이 지났다는 뜻으로 여기서 다시 스레드를 unblock해준다.

1. Priority Scheduling

이제 스레드들을 스케줄링할때 라운드로빈 방식이 아니라 우선순위를 기반으로 스케줄링을 하도록 바꾼다.

그래서 thread.c의 thread\_yield() 함수에서 스케줄링할 때 list\_insert\_ordered를 사용한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그리고 thread\_create() 함수에서도 새로운 스레드를 생성할 때 해당 스레드의 우선순위와 현재 running thread의 우선순위를 비교해서 만약 생성되는 스레드의 우선순위가 높다면 다시 thread\_yield()함수를 호출하여 우선순위가 높은 스레드로 스위칭해준다.

그리고 priority\_aging함수에서는 시간이 지날수록 스레드들의 우선순위를 높여주는 코드를 구현하였다.

마지막으로 sema\_up 함수에서 thread\_unblock() 해주는 코드가 있는데 우선순위를 기반으로 작성이 안되어 있어서 우선순위가 높은 애들을 unblock 해주는 방식으로 코드를 수정해주었다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. Advanced Scheduler

마지막으로 BSD scheduler이다.

우선 핀토스에서는 floating-point 계산이 안되므로 fixed-point 계산법을 사용하기 위해서 함수들을 추가해주었다.

그리고 priority를 계산할 때 필요한 recent\_cpu와 nice를 thread.h의 thread 구조체에 선언해주었고, load\_avg도 필요하므로 thread.c에 스태틱 변수로 선언해주었다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그리고 나서 이제 priority를 계산해주는 코드들을 작성하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

각각 recend\_cpu, load\_avg, cal\_priority를 계산해주는 함수들이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명  
  
매틱마다 thread\_tick() 함수에서 thread\_aging 함수를 호출하는데, TIMER\_FREQ 마다 recent\_cpu 값과 load\_avg 값을 갱신해주고, 매 4틱마다 스레드의 priority를 갱신해준다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

마지막으로 nice, load\_avg, recent\_cpu의 값들을 리턴하는 함수들과

Nice 값을 수정할 수 있는 함수도 추가해주었다.

여기서 nice의 값이 수정될 때 priority 값이 수정되므로 비교해서 thread\_yield() 함수를 호출해주는 것을 알 수 있다.

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

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

**Priority-lifo의 경우 0~15 순으로 쓰레드가 들어온다.**

**Priority-lifo.c의 코드를 보면 for문을 돌면서 i의 값만큼 priority를 높여서 thread\_create() 하기 때문에 뒤에 생성되는 스레드일수록 우선순위가 높아지게 되므로 역순으로 실행되는 것이다.**

**따라서 이때 제일 늦게 들어온 스레드인 15가 실행되고 그뒤로 역순으로 실행된다. 이때 ITER\_CNT가 16이므로 모든 스레드들은 16번 실행된 후 종료된다.**

* make check 수행 결과를 캡처하여 첨부

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명