**Pintos Project 3: Threads**

담당 교수 : 김영재 교수님

조 / 조원 : 김희진

개발 기간 : 11.11~11.17

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

Busy waiting방식보다 더 효율적으로 thread들이 CPU를 이용할 수 있도록 alarm clock을 설정하고 priority scheduler를 구현한다. 또한, 추가적으로 advanced scheduler(BSD Scheduler)도 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

앞서 언급했듯이 현재 pintOS의 thread는 RUNNING state와 READY state를 왕복하며 busy waiting을 하는 상태이다. 이는 CPU를 상당히 낭비하는 방식이다.

이번 프로젝트에서는 thread의 BLOCK state를 이용해 CPU를 더 효율적으로 이용하는 방식을 구현할 것이다.

* 1. Priority Scheduling

현재 pintOS에서는 일정시간마다 thread를 실행시키는 round-robin scheduling을 사용하고 있다. 그렇기 때문에 priority에 따라 thread를 실행시켜줄 수 없다.

이번 프로젝트에서는 Thread에 priority를 부여하여 priority가 높은 순서대로 실행될 수 있게 할 것이다. 이때 낮은 priority를 가진 thread가 실행되지 않는 starvation을 막기위해서, thread가 생성된 후 시간에 비례하게 priority가 증가하는 thread aging을 구현할 것이다.

* 1. Advanced Scheduler (추가구현을 한 경우)

앞서 구현한 priority scheduler는 실행을 priority에만 맡기기 때문에 thread aging을 구현한다 하더라도 average response time이 커질 가능성이 있다. Advanced scheduler는 average response time을 줄이기 위해서 mlfq scheduler방식을 도입한다. 이는 priority에 따라 여러 개의 ready queue를 두고 priority는 실시간으로 조절(feedback)해줄 것이다. 각각의 ready queue에서는 round robin방식으로 scheduling을 해줄 것이다.

Priority에 따라 여러 개의 ready queue를 두고 ready queue안에서 round robin 방식으로 scheduling을 해주는 것은 priority scheduling에서 해줬으므로 우리는 priority를 실시간으로 조절해주는 것에 초점을 맞출것이다. 이때 사용되는 nice, priority, recent\_cpu, load\_avg계산에는 부동소수점 계산이 필요하므로 이에 유의해야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

Block상태의 thread를 list에 저장할 것이다. 이후, 이 tick이 증가할 때마다 list들 순회하며 일어날 시간이 되었는지 확인해주며 일어날 시간이 된 thread는 list에서 제거하고 unblock시켜준다.

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

Ready list에 running thread보다 높은 priority를 가진 thread가 들어오면, running thread를 ready state로 바꾸고 ready list에 priority에 따라 내림차순 정렬해서 넣어준다. 그리고 이때 새로 들어온 thread도 priority에 따라 내림차순 정렬해서 ready list에 넣는다. 그 후, ready list에서 priority가 가장 높은 thread부터 실행시킨다.

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

Priority를 계산하는 식은 다음과 같으며 매 4 ticks마다 계산해준다.  
priority = PRI\_MAX – (recent\_cpu/4) - (nice\*2)  
여기서 PRI\_MAX는 가장 큰 priority를 말하며 pintOS에서는 63에 해당한다. 이 식은 많은 CPU time을 가지는 thread는 다음 scheduling 때, 낮은 priority를 가지게됨을 의미한다. Priority를 계산하는 과정에서 필요한 recent\_cpu와 nice는 무엇을 의미하는지 살펴보자

* Recent\_cpu

Thread가 지금까지 사용한 CPU시간을 의미하며 실수이다. 이를 계산하는 식은 다음과 같다.

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

RUNNING state에있는 thread의 recent\_cpu 값은 매 tick마다 1만큼 증가된다. 또한, 모든 thread의 recent\_cpu값은 매초마다 업데이트되는데 이때 1sec는 TIMER\_FREQ(tick 단위)로 정의되어 있다. 초기값은 0이며 자식 thread는 부모 thread의 recent\_cpu값을 가진다.

* Load\_avg

Recent\_cpu계산에 필요한 값이다. READY state에 있는 thread 수의 평균을 의미하며 실수이다. 이를 계산하는 식은 다음과 같으며 ready\_threads는 idle thread를 제외하고 READY나 RUNNING state에 있는 thread수를 의미한다.

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

recent\_cpu와 마찬가지로 매초마다 업데이트 되어야 하며 초기값은 0이다.

* Nice

-20부터 20 사이의 값을 가진다. 새로운 nice값을 설정할 때마다 priority를 다시 계산해야 한다. 초기값은 0이며 자식 thread는 부모 thread의 recent\_cpu값을 가진다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

- 11.10~11.11: pintOS 매뉴얼 분석

- 11.12 : alarm clock, priority scheduling 구현

- 11.13 : Advanced scheduler 구현

- 11.14~15 : error 수정

- 11.16~17 : 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. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

-우선 threads/thread.c에서 block상태의 thread를 저장해줄 sleep\_list를 선언하고 초기화한다.

-threads/thread.h에서 thread가 일어날 시간을 저장하기 위해 thread구조체에 wakeup\_tick 변수를 추가한다. Threads/thread.c의 init\_thread와 thread\_init에서 이 변수를 초기화한다.

-threads/thread.c에서 Block될 thread의 wakeup\_tick을 설정하고 thread를 sleep\_list에 넣고 block시켜줄 thread\_sleep 함수를 구현한다.  
Sleep\_list에 있는 thread들을 순회하며 일어날 때가 된 thread를 깨우고(unblock시키고) sleep\_list에서 제거시키는 함수 thread\_wake\_up 함수를 구현한다. 매 tick마다 이 함수를 call한다.

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

-Ready list에 running thread보다 높은 priority를 가진 thread가 들어오는 경우는 block된 thread가 unblock된 경우, thread yield가 호출된 경우이다. 그래서 list.c에 정의되어 있는 list\_insert\_ordered() 함수를 이용해 thread\_unblock(), thread\_yield() 함수에서 ready\_list에 priority 내림차순으로 thread가 들어가게 한다. 이때 priority 비교를 위해서 threads/thread.c에 cmp\_thread\_priority 함수를 정의하고 thread.h에도 추가한다.

-mlfq 방식과 구분을 위해서 ppt 32page의 수정사항들을 추가한다.

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

-Thread구조체(threads/thread.h)에 recent\_cpu와 nice 변수를 그리고 threads/thread.c에 전역변수로 load\_avg를 추가한다.  
threads/thread.c의 thread\_init에서 위 변수들을 초기화 해주고 init\_thread에서 자식 thread의 위 변수들이 부모 thread의 값을 받게 한다.

-threads/thread.c에 매 tick마다 recent\_cpu를 증가시켜줄 increase\_recent\_cpu함수와 매 초마다 recent\_cpu와 load\_avg를 업데이트 할 thread\_set\_recent\_cpu와   
thread\_set\_load\_avg 함수를 추가한다. 또한, 새로운 nice값을 설정해줄 thread\_set\_nice함수도 구현한다.

-Recent\_cpu와 load\_avg는 실수이며 load\_avg와 priority는 정수이므로 이들을 계산하기위해 fixed\_point.h 헤더파일을 추가했다.

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

**텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명**

* 1. **텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

     자동 생성된 설명**Priority Scheduler

**텍스트, 도표, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 라인이(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
  1. Alarm Clock

-threads/thread.h

Thread 구조체에 thread마다 깨워줄 tick을 저장하는 wakeup\_tick변수를 int64\_t타입으로 선언한다.

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

-threads/thread.c

Thread\_init과 init\_thread에서 wake\_upt tick을 초기화한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

Block된 thread를 저장할 sleep list를 선언하고 thread\_init에서 초기화한다.



폰트, 텍스트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

-device/timer.c

Timer\_sleep함수에서 thread를 block시키고 sleep\_list에 넣어줄 thread\_sleep을 call하고 인자로는 ticks+start를 넣어준다. 이는 thread의 wakeup\_tick에 저장될 값이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Timer\_interrupt함수에서 일어날 시간이 된 thread를 깨워주는 thread\_wake\_up함수를 call 해준다. 이때, 현재 +1된 tick값에 대해 깨워줘야 하므로 tick++뒷 부분에서 call한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-threads/threads.c

Thread\_sleep은 intr\_disable()을 이용해 interrupt를 꺼주고 이전 interrupt 상태를 old\_level변수에 담아준다. 그리고 현재 thread가 idle\_thread가 아닐 때, wakeup\_tick에 인자로 받은 즉, start+ticks의 값을 넣어준다. 현재 thread를 sleep\_list에 넣은 후, thread를 block state로 바꿔준 후 intr\_set\_level을 이용해 이전의 interrupt 상태로 돌린다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Thread\_wake\_up함수는 sleep list를 돌면서 해당 thread의 wakeup\_tick이 현재 ticks이하면 일어날 수 있으므로 list에서 지워주고 해당 thread를 unblock시킨다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-threads/thread.h

새롭게 선언한 두 함수들을 추가한다.



* 1. Priority Scheduling

Mlfq인 경우와 구별하기 위해서 다음과 같은 코드를 추가해준다.

폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명threads/thread.h

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

threads/thread.c

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명 threads/init.c

-threads/thead.c

Ready list에 thread를 넣는 경우는 thread\_unblock()과 thread\_yield()를 호출하는 경우이다. 이때, priority에 따라 내림차순으로 들어갈 수 있게 다음과 같이 코드를 작성한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

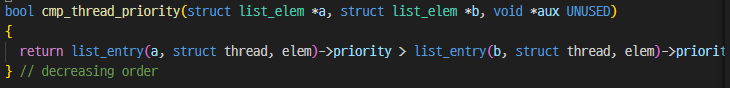
자동 생성된 설명

List\_insert\_ordered는 lib/kernel/list.c에 다음과 같이 선언되어있다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

즉 정렬을 해줄 함수가 필요하며 이를 위해서 thread/thread.c에 priority에 따라 내림차순으로 정렬해줄 bool함수 cmp\_thread\_priority를 추가해 사용했다.



현재 실행중인 thread의 priority가 바뀌는 것은 thread\_create와 thread\_set\_priority 함수에서이다. Thread\_create()에서는 새롭게 생긴 현재 thread의 priority와 ready list에서 가장 앞에있는 즉, ready state의 thread 중 priority가 가장 큰 thread의 priority를 비교해준다. Ready state thread의 priority가 더 크다면 thread\_yield를 호출해 cpu점유권을 넘겨준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Thread\_set\_priority()에서는 이전의 priority와 새롭게 설정한 priority를 비교해주어 이전의 priority가 더 크다면 thread\_yield를 통해 CPU 점유권을 넘겨준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Thread\_get\_priority는 현재 함수의 priority를 반환하는 함수이므로 다음과 같이 처리해준다.

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

-threads/thread.h

새롭게 추가한 함수들을 헤더파일에 추가한다.



-threads/synch.c

Sema\_up함수는 sema->waiters에 있는 block된 thread들을 unblock시켜주는 함수이다. 이 list에 있는 thread들을 unblock할 때에도 priority에 따라 unblock해주어야 하므로 list.c에 있는 list\_sort함수를 이용해 정렬한 후 가장 앞에 있는 thread를 unblock해주었다. 그리고 이 unblock된 thread의 priority가 running thread의 priority보다 크다면 thread\_yield()를 call해 CPU 점유권을 넘긴다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Semaphore로 block상태일 때도 list 내에서 priority변동이 있을 수 있으므로 sema\_down에도 list\_insert\_ordered함수를 이용해 sema->waiters list를 정렬시켰다. 하지만 sema\_up에서 unblock시킬 때 정렬한 후, unblock하므로 이 부분이 없어도 정상 작동될 것 같다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-device/timer.c

매 tick마다 thread의 priority를 aging해준다. 이 부분은 advanced scheduling에서 더 발전시켰으므로 아래에서 자세히 설명한다.

* 1. Advanced Scheduling

-device/timer.c

Thread\_mlfq나 thread\_prior\_aging인 case에 대해서 thread aging을 수행할 것이다. 이때, 매 tick마다 recent\_cpu를 1씩 증가시켜주기 위해 increase\_recent\_cpu()를 call한다. 또한 매초마다 load\_avg와 recent\_cpu를 업데이트 해야한다. 그러므로 timer\_ticks() % TIMER\_FREQ가 0일 때, 즉, 매초마다 이들을 업데이트하는 함수들을 call한다. 이때 recent\_cpu를 업데이트 하는 데에 load\_avg가 필요하므로 load\_avg를 업데이트 하는 함수를 먼저 call한다. 또한 4 tick마다 thread aging함수를 call해 aging을 수행한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-threads/fixed\_point.h

그럼 이제 각각의 함수들을 살펴볼 것인데 그 전에, recent\_cpu와 load\_avg는 실수이며 다른 priority나 nice는 정수이기 때문에 부동소수점을 계산할 헤더파일을 만들었다. 해당 프로젝트를 진행하며 사용한 함수들만 첨부했으며 이 함수들은 매뉴얼을 그대로 따랐기에 자세한 설명은 생략한다.



텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

-threads/thread.c

다음은 running thread가 idle thread가 아닐 때 recent cpu를 1씩 증가시켜주는 함수이다. 이때 recent\_cpu는 real number이므로 int\_to\_fp함수를 사용했다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

다음은 load\_avg를 업데이트하는 thread\_set\_load\_avg()이다. Ready\_threads는 명세서에 기재된 대로 ready or running state은 thread의 개수이되 idle thread를 제외하도록 코드를 짰다. 이때 lib/kernel/list.c에 있는 list\_size함수를 이용했다. Load\_avg계산은 명세서에 나온대로 수행하되 load\_avg는 실수이고 ready\_threads는 정수임에 유의하며 계산했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음은 recent\_cpu를 업데이트하는 thread\_set\_recent\_cpu()이다. Load\_avg와 달리 recent\_cpu는 각 thread마다 갖고 있는 값이므로 all\_list를 순회하며, 즉 모든 thread를 순회하며 명세서에 따라 recent\_cpu를 계산한다. 이때, idle\_thread는 계산하지 않는다는 점과 recent\_cpu, load\_avg는 실수, nice는 정수인점에 유의하며 계산했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음은 4 tick마다 call되는 thrad\_aging()이다. All list를 순회하며 즉, 모든 thread를 순회하며 idle\_thread가 아닌 thread에 대해 명세서에 따라 priority를 계산한다. 이때, priority와, PRI\_MAX 그리고 nice는 정수이며 recent\_cpu는 실수라는 점에 유의하며 계산했다. 또한 계산한 priority가 혹시 최소 그리고 최대 priority 범위를 넘어갈 때 조정해주었다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이제 나머지 요구되는 함수들을 채울 것이다.

Thared\_get\_load\_avg는 load\_avg의 100배의 값을 return하는 함수이고 load\_avg는 실수이므로 fp\_to\_int를 이용해 다음과 같이 코드를 짰다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Thread\_get\_recent\_cpu는 recent\_cpu의 100배의 값을 정수로 return하는 함수이고 recent\_cpu는 실수이므로 fp\_to\_int를 이용해 다음과 같이 코드를 짰다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

다음은 thread\_set\_nice함수로 nice에 인자로 들어온 새로운 값을 넣어주는 함수이다. 이때 인자로 들어온 값이 nice의 범위를 넘어갔을 때 조절한다. 또한 nice값이 갱신될 때마다 priority를 조절하라고 명세서에 나와있으므로 priority를 다시 계산해주고 범위 체크를 한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-threads/thread.h

새롭게 선언한 함수를 헤더파일에 추가한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

결과적으로 priority-lifo.c는 last in first out 즉, 나중에 들어간 우선순위가 높은 thread가 첫번째로 나오는지 test하는 코드이다.

해당 코드는 0번부터 15번까지의 thread를 차례로 생성하는데 각 thread는 PRI\_DEFAULT+1+i(thread 번호)의 priority를 가진다. 즉, 0번 thread는 가장 낮은 priority, 15번 thread는 가장 높은 priority를 가진다. 그 후, thread\_set\_priority(PRI\_DEFAULT)코드를 통해 main thread의 우선순위를 DEFAULT로 설정하고 모든 thread를 run시키고 termination까지 관찰한다. 0번부터 15번 thread까지 차례로 ready list에 들어가지만 우선순위는 반대 방향이기 때문에 15번 thread부터 0번 thread까지 차례로 running되는 것이 올바른 결과이다.

우선순위에 따라 내림차순 정렬되어 thread가 ready list에 들어가도록 코드를 작성했다면 15번 thread부터 내림차순으로 thread가 running할 것이다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위 사진은 priority-lifo test결과이며 15번 thread부터 0번 thread까지 차례로 수행된 결과를 확인할 수 있다.

* make check 수행 결과를 캡처하여 첨부

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명