**Pintos Project 3: Threads**

담당 교수 : 김영재 교수님

조 / 조원 : 김효림

개발 기간 : 2024.11.19 ~ 2024.11.23

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

Pintos manual의 첫 번째 프로젝트인 thread에 대해, alarm clock, priority scheduling(+aging), advanced scheduler(MLFQS)를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

Busy-waiting 방식으로 되어 있는 pintos의 스케줄러 방식으로 동작하고 있다. 비효율 적인 이 방식을 일정 시간 동안 작동하지 않는 thread를 sleep으로 상태를 변환해 주는 방식으로 바꿔준다.

* 1. Priority Scheduling

RR 방식으로 구현된 스케줄러를 priority를 고려하여 스케줄링을 진행할 수 있도록 수정한다. 높은 priority를 가지는 thread에 대해 preempt 할 수 있도록 수정한다. 이후 starvation 문제를 해결하기 위해 priority aging 기법을 적용할 수 있도록 한다.

* 1. Advanced Scheduler (추가구현을 한 경우)

MLFQ 방식의 스케줄러를 구현한다. Priority 단계에서 aging 구현을 위해 priority나 load-avg, 그리고 recent\_cpu를 구하는 함수를 이미 구현한 상태이기 때문에, nice value에 대한 함수를 작성하여 구현할 수 있다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

Thread에 대해 특정 시간(timer tick)이 지났다면, thread\_unblock 함수를 호출하여 해당 thread를 깨운다. 이 thread들은 sleeping\_thread\_list에 저장되어 있으며, 해당 리스트를 돌며 wake를 시켜주고 sleep list에서 삭제한다.

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

running thread보다 높은 priority를 가지는 thread가 들어올 경우, 현재 돌아가고 있는 thread를 yield하고 새로운 thread가 cpu를 점유할 수 있도록 한다. 이때 원래 실행 중이던 thread는 ready\_list로 돌아간다.

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

- 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

- nice : nice value

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

1. 11/19 : Alarm Clock
2. 11/20 : Priority 기본 구현
3. 11/22 : Priority aging 구현
4. 11/23 : Advanced Scheduler 구현 및 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
  + Alarm Clock

[device/timer.c]의 timer\_sleep() 함수를 수정하여 busy wait 방식의 스케줄링을 바꿔 준다. 우선 해당 파일 내부에 struct list sleeping\_thread\_list를 선언하여 관리할 수 있는 리스트 배열을 선언하고, timer\_init() 함수에서 list\_init 함수를 이용하여 초기화 해준다. 그 후 timer\_sleep() 함수를 수정하여 wake\_time을 계산하고, list\_push\_back 함수를 이용해 sleeping\_thread\_list에 추가하는 방식으로 코드를 구현한다. Timer interrupt 함수에서는 특정 time이 지났을 경우 list에서 삭제해 준다(이후 해당 thread는 ready list로 옮겨진다). 이 함수는 매 ticks 마다 호출되어 리스트를 순회하게 된다.

* + Priority Scheduling

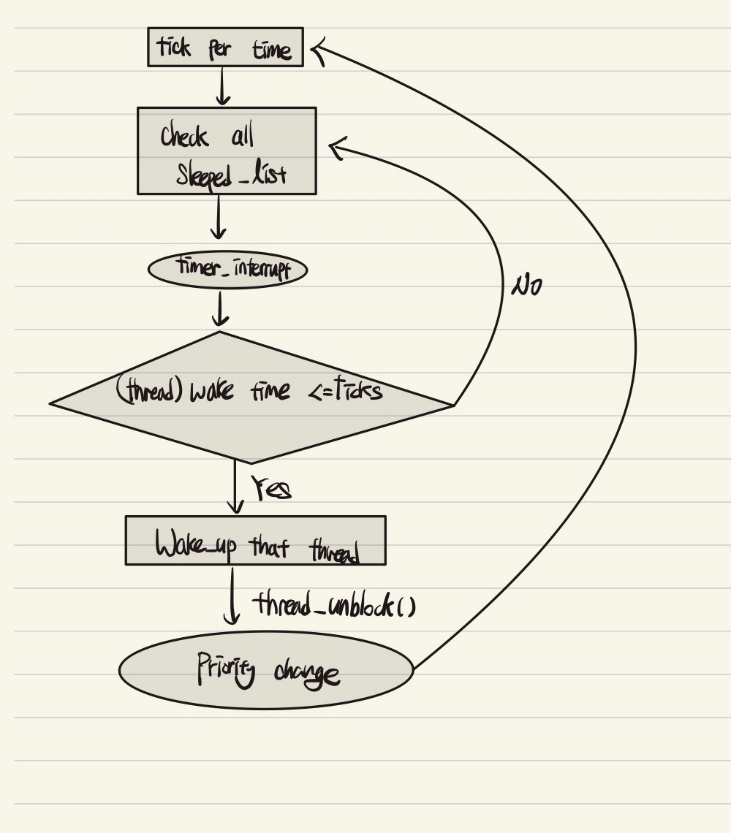
Thread.c에 정의되어 있는 thread\_set\_priority 함수와 thread\_get\_priority 함수를 수정해 준다. 또한 thread\_yield 함수와 thread\_unblock 함수에서 기존에 list\_push\_back 함수로 구현되어 있던 ready list 삽입 방식을 list\_insert\_ordered를 통해 priority에 따라 ready list가 정렬될 수 있도록 한다. 이때 우선 순위 비교를 위해 bool compare\_priority 함수를 정의하여 사용했다. 이후 명세서에 따라 aging에 필요한 flag를 각 부분에 추가하여 aging thread를 처리하는 case를 따로 구현하였다. 이 부분을 제대로 처리하기 위해 구현되어 있지 않았던 thread\_get\_load\_avg, thread\_get\_recent\_cpu 함수를 수정하였고, update\_cpu, thread\_get\_max\_priority, update\_load\_avg, update\_priority 함수를 추가적으로 구현하였다. 각 함수에 대한 설명은 연구 결과 부분에서 설명하도록 하겠다.

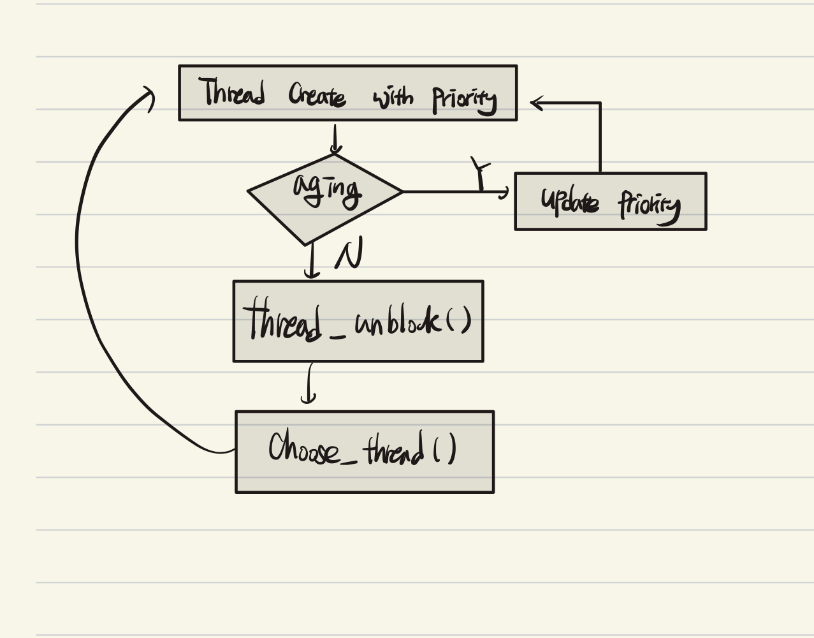
* + Advanced Scheduler (추가구현을 한 경우)

Priority-aging까지 구현한 단계에서, thread\_mlfqs flag에 대해 해당 flag가 true일 때 thread\_set\_priority를 수행하지 않도록 case를 막아준다. 그리고 thread.c에 아직 구현되어 있지 않은 thread\_set\_nice 함수와, thread\_get\_nice 함수를 수정해 준다.

1. **연구 결과**
   1. **Flow Chart**

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





* 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 구현**

struct list sleeping\_thread\_list;

devices/timer.c 파일에 sleep된 thread를 관리하기 위해 sleeping\_thread\_list를 선언하였다. 이후 timer\_init 함수에서 list\_init(&sleeping\_thread\_list);를 통해 해당 리스트를 initialize 하는 부분을 구현하였다. 그 후 과제 명세서에 따라 busy wait 방식의 timer\_sleep 함수를 수정한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Thread가 일어날 wake\_time을 timer\_ticks() + ticks로 설정하고, 현재 실행 중인 thread(현재 스레드)를 sleeping\_thread\_list에 추가한다. 이 함수에서 enum intr\_level을 통해 데이터 손상을 방지한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그 후 timer\_interrupt 함수에서 ticks를 증가시키면서, sleeping\_thread\_list를 돌며 wake time이 ticks보다 큰 경우 sleeping\_thread\_list에서 제거하고 thread\_unblock() 함수를 통해 thread를 깨우는 과정을 진행한다. 아래 thread\_prior\_aging, thread\_mlfqs에 대한 설명은 2번과 3번 내용을 설명할 때 설명하도록 한다.

* 1. **Priority Scheduling 구현**

(명세서에서 aging case를 처리하기 위해 추가한 부분에 대한 코드 캡쳐는 생략했습니다.)

우선 [thread.h] 파일에서 thread 구조체에 다음과 같은 변수들을 추가하고, 구조체 외부에 다음의 함수들을 선언하였다.

**폰트, 텍스트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명**

Wake는 이전 alarm clock을 위해 선언해 둔 변수이고, recent\_cpu와 nice는 말 그대로 thread의 recent\_cpu time과 nice value를저장하기 위해 선언한 변수이다. 각 변수들은 thread\_init이나, init\_thread 함수에서 초기화하였다.

[thread.c]에서는, 우선 priority에 따라 ready list를 저장하기 위해 thread\_yield()와 thread\_unblock() 함수를 수정하였다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

Ready\_list에 thread를 넣을 때 list\_push\_back 함수를 이용해 맨 뒷 부분에 thread를 넣어준 것과 달리, priority에 따라 정렬하기 위해 list\_insert\_ordered 함수로 바꿔주었다. 이때 비교 연산을 위해 아래와 같은 함수를 추가로 구현해 주었다. 또, thread\_create 부분에서 생성하는 thread의 priority가 현재 thread의 priority보다 클 경우 thread\_yield를 불러와 다시 스케줄링하도록 조정하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그리고 thread\_set\_priority와 thread\_get\_priority 함수를 수정하여 현재 priority와 인자로 받은 new\_priority를 비교하여, 이전 priority가 new\_priority보다 크다면 thread\_yield를 통해 다시 스케줄링을 진행할 수 있도록 한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Sema\_up 함수를 수정해서, waiting 중인 thread 중 가장 큰 priority를 가진 thread를 찾아 해당 thread를 unblock할 수 있도록 코드를 수정하였다.

우선 이렇게만 구현했을 때 prior의 기본 test는 통과했지만, aging은 통과하지 못했다. 이를 위해서는 recent\_cpu와 load\_avg에 대한 연산이 필요했다. 각 식은 위에서 언급했던 대로

-recent cpu : (2 \* load\_avg) / (2 \* load\_avg + 1) \* recent\_cpu + nice

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

이다. 이 연산을 위해서는 정수와 실수, 실수와 실수의 연산이 필요한데, pintos는 소수점 계산을 지원하지 않기 때문에 [thread.h]에 FRACTION을 (1<<14), 즉 2^14로 define 한 후 고정 소수점 연산에 활용하였다. 매뉴얼에서는 연산을 위해 헤더를 추가하는 것을 권장했지만, thread.c 내부에서 연산하면서 적용하기만 했다. 또한 load\_avg가 정의되어 있지 않기 때문에, [thread.c] 파일에 static int load\_avg를 선언해 주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위 함수들은 코드에서 요구하는 대로 작성해 주었다. 각 연산에 부동 소수점 연산을 적용하기 위해 FRACITON을 사용하였다. 그 후 이전에 [thread.h]에 새로 선언해 주었던 함수들을 작성하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Update\_cpu 함수는 모든 thread의 cpu값을 갱신하고, update\_load\_avg 함수는 load\_avg 값을 갱신한다. 또 update\_priority() 함수는 thread들의 priority를 갱신한다. 이때 가장 큰 priority를 가지는 thread와의 비교과 필요하기 때문에, thread\_get\_max\_priority 함수를 구현하였다. 이와 비교해서 thread\_yield()가 발생할 경우, update\_priority는 interrupt 도중에 진행되기 때문에 intr\_yield\_on\_return() 함수를 사용하였다.

* 1. **Advanced Scheduler 구현**

앞선 prior-aging 문제 해결을 위해 recent\_cpu나 load\_avg를 update 해주는 함수, 그리고 priority를 갱신하는 함수를 이미 구현해 둔 상태기 때문에, 추가적으로 구현해야 할 함수는 thread\_set\_nice() 함수와 thread\_get\_nice() 함수 뿐이다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그 전에 우선 timer\_interrupt에 mlfqs에 대한 예외 케이스를 추가하였다. Aging에 대한 예외 케이스는 이전 단계에서 추가해 둔 상태기 때문에, thread\_mlfqs를 if 문 조건에 추가해 주기만 했다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Thread\_set\_nice 함수는 인자로 받은 nice 값을 현재 thread의 nice 값으로 set해 줄 수 있도록 작성하였다. 그리고 nice 값을 반영하여 thread의 priority를 update해 주었다. 현재 priority가 max priority보다 작으면 thread\_yield를 호출하도록 했다.

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

priority-lifo.c 코드는 우선순위 스케줄링 테스트를 진행하는 코드로, multi thread를 생성하고 LIFO 스케줄링 방식에 따라 스케줄링이 진행되는지 확인하기 위한 코드이다. 총 16개의 스레드를 생성하고, 마지막에 생성된 가장 thread가 가장 높은 priority를 갖게 된다. 따라서 15 thread 부터 실행된 후 나머지 스레드들이 순차적으로 실행되는 걸 확인할 수 있다.

텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* make check 수행 결과를 캡처하여 첨부

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명