**Pintos Project 3: Threads**

담당 교수 :김영재 교수님

학번/이름:20211188 방수윤

개발 기간 :11/20-11/23

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

1. **Alarm clock**

PintOS의 기본 스케줄러는 **busy-waiting** 방식으로 동작하며, 이는 자원을 비효율적으로 사용하는 방식이라 할 수 있겠음. 이를 개선하기 위해 스레드가 특정 시간 동안 \*\*효율적으로 대기(sleep)\*\*할 수 있도록 **tick 기반 스케줄링**을 구현함. tick 기반 방식에서는 스레드가 깨어나야 할 기준 tick 값을 계산하여 스레드를 \*\*대기 상태(blocked)\*\*로 전환하며, 이를 관리하기 위해 \*\*대기 리스트(waiters list)\*\*를 사용함. 이후, 시스템 tick이 증가하면 대기 리스트를 확인하여 깨어날 시점이 된 스레드들을 \*\*준비 상태(ready)\*\*로 전환함. 이를 통해 busy-waiting으로 인해 발생하는 CPU 자원 낭비를 제거하고, 스케줄링의 효율성을 높임.

1. **Priority scheduling**

PintOS의 초기 스케줄러는 **Round-Robin 방식**으로 동작하며, 이는 ready\_list의 끝에 새 스레드를 삽입하는 단순한 방식이라 할 수 있겠음. 이로 인해 우선순위가 높은 스레드가 있어도 적절히 처리되지 않는 문제가 있음. 이를 해결하기 위해 \*\*우선순위 기반 스케줄링(Priority Scheduling)\*\*을 구현함. 스레드가 생성되거나 준비 상태로 전환될 때 우선순위를 평가하며, 현재 실행 중인 스레드보다 높은 우선순위를 가진 스레드가 들어오면 preemption하여 즉시 실행되도록 설계함. 하지만, 단순한 우선순위 스케줄링은 낮은 우선순위의 스레드가 실행되지 못하는 \*\*기아 상태(starvation)\*\*를 유발할 가능성이 있음. 이러한 문제를 방지하기 위해 **priority aging** 기법을 도입함. priority aging은 스레드가 대기 상태에 있는 시간이 길어질수록 우선순위를 점진적으로 증가시켜줌으로써, 결국 낮은 우선순위의 스레드도 실행될 기회를 갖도록 보장함. 이를 통해 우선순위 스케줄링의 효율성과 공정성을 동시에 달성할 수 있도록 개선하였음.

1. **Advanced scheduling**

BSD Scheduler(Multi-Level Feedback Queue; MLFQ) 기반 스케줄링을 구현함. MLFQ는 nice 값(스레드의 자발적 양보 정도), recent\_cpu(최근 실행 시간), \*\*load\_avg(시스템 부하 평균)\*\*를 활용하여 스레드의 우선순위를 동적으로 계산함.

* Nice 값은 스레드의 자발적 양보 정도를 나타내며, 값이 높을수록 우선순위가 낮아지도록 설계되었음.
* Recent\_cpu는 스레드가 최근에 얼마나 자주 실행되었는지를 나타내며, 값이 높아질수록 우선순위가 낮아짐.
* Load\_avg는 전체 시스템의 부하를 나타내며, recent\_cpu 값 계산에 영향을 미침.

이 과정에서 소수점 연산이 필요한데, PintOS는 이를 기본적으로 지원하지 않음. 따라서 고정소수점 연산 방식을 활용하여 정확한 계산이 이루어지도록 구현하였음. 이러한 방식은 PintOS의 스케줄링이 동적으로 변화하는 부하 상태에서도 공정하고 효율적으로 작동할 수 있도록 보장함.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

기본적으로 PintOS의 스케줄러는 busy-waiting 방식을 사용하여 thread를 대기 상태로 유지함. 이는 CPU를 불필요하게 점유하여 자원을 낭비하는 비효율적인 방식임. 따라서 tick 기반 스케줄링으로 전환하여 대기 상태의 thread를 효율적으로 관리할 필요가 있음.

tick 기반의 Alarm Clock 구현을 통해 thread가 필요한 시간만 대기 상태로 유지되고, 불필요한 CPU 자원 낭비가 감소함. 이를 통해 스케줄링의 효율성과 시스템 전체 성능이 개선될 것으로 기대됨.

* 1. Priority Scheduling

PintOS의 기본 스케줄러는 Round-Robin 방식을 사용하며, thread의 우선순위를 고려하지 않음. 따라서 높은 우선순위를 가진 thread가 빠르게 실행되지 못하거나, 낮은 우선순위 thread의 기아 상태(starvation)가 발생할 수 있음. 이를 해결하기 위해 우선순위 기반 스케줄링이 필요함.

Priority Scheduling을 통해 높은 우선순위의 thread가 빠르게 실행되며, 스케줄링의 효율성이 향상됨. 또한, Priority Aging을 적용하면 낮은 우선순위 thread도 일정 시간 이후 실행 기회를 확보하여 공정성을 보장함. 이를 통해 효율성과 공정성을 동시에 달성할 수 있음.

* 1. Advanced Scheduler (추가구현을 한 경우)

기본 Priority Scheduling은 정적인 우선순위만 고려하므로, 동적으로 부하가 변하는 환경에서는 비효율적일 수 있음. 이를 해결하기 위해 동적인 우선순위 계산이 가능한 MLFQ 스케줄링이 필요함. MLFQ는 thread의 recent\_cpu(최근 실행 시간), load\_avg(시스템 부하 평균), nice 값을 고려하여 thread 우선순위를 동적으로 관리함.

MLFQ를 통해 시스템 부하와 thread 상태에 따라 우선순위를 동적으로 조정할 수 있음. 이를 통해 thread가 공정하게 CPU를 점유하며, 높은 부하 상태에서도 스케줄러가 효율적으로 동작함. 특히 소수점 연산 로직을 추가로 구현하면 정확한 우선순위 계산이 가능해져, 스케줄링의 신뢰성과 성능이 크게 향상됨.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

Blocked 상태의 스레드는 특정 조건이 충족되었을 때 Ready 상태로 전환하여 깨울 수 있을 것이다. 이를 위해 sema\_up() 함수를 사용하여 대기 중인 스레드를 Ready 리스트로 이동시킬 수 있도록 구현했다. 이 과정에서 thread\_unblock() 함수가 호출되어 해당 스레드를 Ready 리스트에 추가했고, 추가된 스레드는 스케줄러에 의해 실행될 수 있었다. 이러한 방식으로 Blocked 상태의 스레드를 효율적으로 처리할 수 있도록 했다.

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

Ready 리스트에 현재 실행 중인 스레드보다 높은 우선순위를 가진 스레드가 들어올 경우 \*\*preemption(선점)\*\*을 수행할 수 있을 것이다. 이를 위해 thread\_yield() 함수를 호출하여 현재 실행 중인 스레드가 CPU를 반환하도록 구현했다. 이후 Ready 리스트에 있는 스레드 중 우선순위가 가장 높은 스레드를 선택하여 실행되도록 했다. 이 방식으로 높은 우선순위의 작업이 신속하게 처리될 수 있도록 했다.

1. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)
   1. Recent CPU (최근 CPU 사용 시간)
   * 각 스레드가 CPU를 얼마나 사용했는지를 나타내는 값이다. CPU를 많이 사용한 스레드는 우선순위가 낮아지도록 계산했다.
   1. Load Average (시스템 부하 평균)
   * 시스템의 전체 CPU 부하를 나타내는 값으로, 시스템 부하가 높을수록 스레드의 우선순위가 낮아지도록 설계했다.
   1. Nice 값 (양보 정도)

->사용자가 CPU 양보를 설정할 수 있는 값이다. 높은 Nice 값을 가진 스레드는 우선순위가 낮아지도록 구현했다.

이러한 요소들을 조합하여 우선순위를 동적으로 계산하고, 효율적인 스케줄링이 이루어질 수 있도록 했다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
  1. 11/20-11/20: alarm clock
  2. 11/21-11/22: priority scheduling
  3. 11/23-11/23: advanced scheduling , 보고서 작성
  4. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명

1. **Alarm Clock**  
devices/timer.c 파일에서 기존 busy-waiting 방식의 스케줄링을 수정해야 한다. 이를 위해 struct list sleep\_list를 선언하여 sleep 상태의 스레드를 관리하며, timer\_init 함수에서 이를 초기화한다. timer\_sleep 함수는 현재 스레드의 wake\_time을 설정하고 sleep\_list에 삽입한 후 스레드를 Blocked 상태로 전환하는 로직을 추가해야 한다. timer\_interrupt 함수에서는 tick 값을 증가시키면서 sleep\_list에서 조건에 맞는 스레드를 Ready 상태로 복귀시켜야 한다.

**2.Priority Scheduling**  
threads/thread.c와 threads/init.c 파일에서 구현을 진행해야 한다. 스레드의 우선순위를 정렬하고 CPU 선점을 구현하기 위해 compare\_priority 함수를 생성하고, 이를 list\_insert\_ordered 함수의 기준 함수로 사용한다. thread\_set\_priority 함수에서는 새로 추가된 스레드가 현재 실행 중인 스레드보다 높은 우선순위를 가질 경우 선점을 수행하도록 수정한다. 또한, Aging을 위한 bool thread\_aging 플래그를 추가하여 우선순위가 낮은 스레드도 시간이 지나면서 처리되도록 한다.

**3. Advanced Scheduler**  
threads/thread.c와 threads/thread.h 파일에서 우선순위 계산에 필요한 recent\_cpu, nice, load\_avg 등을 추가해야 한다. 스레드의 recent CPU 사용량을 갱신하는 update\_recent\_cpu 함수와 시스템의 load average 값을 갱신하는 update\_load\_avg 함수를 구현한다. 이를 기반으로 스레드의 우선순위를 계산하는 update\_thread\_priority 함수도 작성해야 하며, 명세서에 따라 thread\_set\_nice와 thread\_get\_nice 함수를 구현한다. 매 tick마다 모든 스레드의 우선순위를 갱신하도록 mlfqs\_tick 함수를 추가해야 한다.

1. **연구 결과**
   1. **Flow Chart**

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

**1.**

**텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명**

**2.**

**도표, 텍스트, 라인, 평면도이(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**

[devices/time.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

struct list sleep\_a를 선언하여 sleep 상태에 있는 thread를 관리하는 리스트를 생성하고, timer\_init 함수 내에서 list\_init(&sleep\_a)를 호출하여 해당 리스트를 초기화한다. 이는 sleep 상태의 thread를 효율적으로 추적하고 관리하기 위해 필요한 준비 작업이다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

timer\_sleep 함수는 기존 busy waiting 방식의 비효율성을 해결하기 위해, 주어진 tick 동안 thread를 sleep 상태로 만들어 스케줄링을 최적화한다. list\_insert\_ordered를 사용해 thread를 wake-up 시간이 빠른 순서로 sleep\_a에 삽입하고, thread\_block를 호출해 thread를 block 상태로 전환한다.

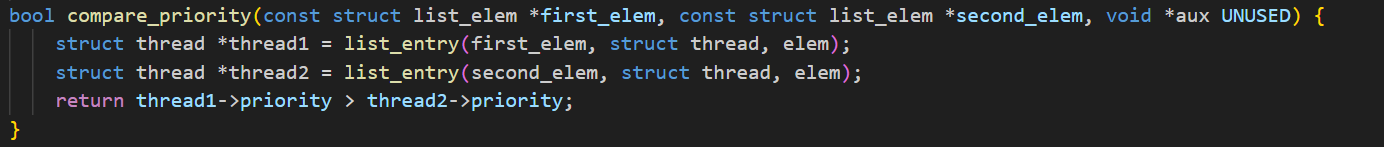
**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

timer\_interrupt 함수는 tick을 증가시키고, 조건에 따라 sleep\_list에서 wake-up 시간이 도래한 thread들을 제거한 후 thread\_unblock을 통해 실행 가능 상태로 전환한다. 추가적으로, thread aging이나 MLFQS 스케줄링이 활성화된 경우 관련 데이터를 갱신한다.

**2.priority scheduling**

[threads/thread.c]



compare\_priority() 함수는 두 thread의 우선순위를 비교하여, list\_insert\_ordered()에서 정렬 기준 함수로 동작할 수 있도록 작성되었다. 이를 통해 우선순위에 따라 thread를 정렬된 상태로 삽입할 수 있다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

thread\_yield() 함수는 현재 실행 중인 thread를 ready list에 우선순위에 따라 정렬된 상태로 삽입하고, 다른 thread가 실행될 수 있도록 스케줄링한다. idle thread는 제외하며, ready list의 적절한 위치를 찾아 삽입한 후 스케줄링을 진행한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

thread\_unblock() 함수는 BLOCKED 상태의 thread를 READY 상태로 변경하고, ready list에 우선순위에 따라 정렬된 위치에 삽입한다. ready list에 삽입 시, 다른 thread보다 높은 우선순위를 가진 경우 적절한 위치를 찾아 삽입한다. 작업 후 인터럽트 레벨을 복원한다.

[threads/synh.c]

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

sema\_up() 함수는 semaphore의 value를 증가시키고, waiters list에서 우선순위가 가장 높은 thread를 선택해 READY 상태로 전환한다. waiters list는 compare\_priority 함수를 사용해 정렬되며, 가장 높은 우선순위를 가진 thread를 pop하여 thread\_unblock()을 호출한다. 이후, 현재 thread보다 우선순위가 높은 thread가 있는 경우 thread\_yield()를 호출하여 양보한다. 작업이 끝나면 인터럽트 레벨을 복원한다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

sema\_down() 함수는 semaphore의 value가 0일 때, 현재 thread를 waiters list에 우선순위에 따라 정렬된 상태로 삽입하고 BLOCKED 상태로 전환한다. semaphore의 value가 양수일 경우 value를 감소시키고, 작업이 끝난 후 이전의 인터럽트 상태를 복원한다. 이를 통해 semaphore를 사용한 thread 간의 동기화를 구현한다.

**3.advanced scheduler**

[threads/thread.h]

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

구조체 thread에 각 값을 저장하도록 추가했다.

[threads/thread.c]

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

자동 생성된 설명

update\_loadmean() 함수는 현재 시스템의 load average를 계산하여 업데이트하는 역할을 수행한다. 현재 ready\_list에 있는 thread의 수를 계산하고, 실행 중인 thread가 idle\_thread가 아니라면 active thread의 수를 증가시킨다. 그 후, 주어진 공식 (59 \* load\_avg + active\_threads \* FRACTION) / 60을 이용해 새로운 load average를 계산한다. 이 함수는 Multi-Level Feedback Queue Scheduler의 일부로, 시스템의 작업 부하를 반영하기 위해 사용된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

update\_cpu() 함수는 Multi-Level Feedback Queue Scheduler에서 각 thread의 recent\_cpu 값을 갱신하는 역할을 한다. 함수는 all\_list에 포함된 모든 thread를 순회하며, idle\_thread를 제외한 나머지 thread의 recent\_cpu 값을 업데이트한다. 이를 위해 먼저 load\_avg 값을 두 배로 늘려 계산한 double\_load\_avg를 사용하며, 이를 기반으로 scaling factor를 구한다. 이후 recent\_cpu는 scaling factor와 thread의 이전 recent\_cpu 값, 그리고 nice 값을 활용한 공식에 따라 갱신된다. 이를 통해 각 thread의 CPU 사용량을 반영한 스케줄링이 가능하도록 한다.

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

모든 스레드의 우선순위를 업데이트하는 로직을 별도 함수로 분리했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

전체 우선순위를 업데이트한 후, 현재 스레드의 우선순위와 비교하여 CPU 양보 여부를 판단하는 기능을 이 함수에 넣었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

thread\_set\_nice() 함수는 현재 실행 중인 thread의 nice 값을 설정하는 역할을 한다. 먼저 thread\_current()를 통해 현재 thread를 가져온 후, 전달받은 nice 값을 current\_thread의 nice 속성에 할당한다. 이후, updated\_thread\_prio()를 호출하여 우선순위를 다시 계산한다. 만약 시스템에서 가장 높은 우선순위(get\_max\_priority())가 현재 thread의 우선순위보다 높다면, thread\_yield()를 호출하여 CPU를 우선순위가 높은 thread에게 양보하도록 한다. 이를 통해 동적으로 nice 값 변화에 따른 우선순위 변경과 적절한 스케줄링이 이루어진다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

updated\_thread\_prio() 함수는 주어진 thread의 우선순위를 계산하여 갱신하는 역할을 한다. 우선 기본 우선순위(base\_priority)는 최대 우선순위(PRI\_MAX)에서 recent\_cpu 값에 기반한 감산 값을 빼서 결정된다. 여기에 nice 값에 기반한 추가적인 조정값(nice\_adjustment)을 계산하여 적용한다. 최종 우선순위는 계산된 값을 소수점 연산을 보정한 뒤 저장된다.

우선순위는 유효 범위 내에 있어야 하므로, 계산된 값이 PRI\_MAX보다 크면 PRI\_MAX로, PRI\_MIN보다 작으면 PRI\_MIN으로 설정한다. 이를 통해 thread의 우선순위가 시스템 내에서 유효한 범위 내에 항상 유지되도록 보장한다.

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

자동 생성된 설명

get\_max\_priority() 함수는 ready\_list에 있는 thread들 중 가장 높은 우선순위를 가진 thread를 찾는 역할을 수행한다.

초기값으로 highest\_priority를 최소 우선순위 값보다 작은 값으로 설정한다. 그런 다음, ready\_list를 순회하면서 각 thread의 우선순위를 비교한다. 만약 현재 thread의 우선순위가 highest\_priority보다 크다면, highest\_priority를 갱신한다.

최종적으로, ready\_list에서 가장 높은 우선순위를 가진 thread의 우선순위를 반환. 이를 통해 현재 실행 중인 thread가 ready\_list의 thread보다 우선순위가 낮은 경우 스케줄링을 조정하는 데 사용할 수 있다.

[devices/timer.c]

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

timer\_interrupt() 함수는 시스템 타이머 인터럽트를 처리하며, tick 값을 증가시키고 현재 thread의 상태를 갱신한다. sleep\_list를 순회하여 wake\_time이 지난 thread를 찾아 깨운다. thread\_aging이나 thread\_mlfqs가 활성화된 경우, recent\_cpu와 우선순위를 갱신하며, 매 tick마다 필요한 함수들을 호출하여 시스템 상태를 유지한다.

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

priority-lifo는 동일한 우선순위를 가진 여러 스레드를 생성한 후, Round-Robin 방식으로 스케줄링이 제대로 이루어지는지 검증하는 테스트이다. 테스트 결과를 보면 Main thread의 우선순위는 PRI\_DEFAULT로 설정되어 있으며, 15번 스레드부터 실행이 시작된다. 이는 Ready list에서 우선순위에 따라 정렬된 상태로 스레드가 실행됨을 보여준다. 각 스레드는 반복적으로 실행되며, 스케줄러가 정해진 순서대로 모든 스레드를 공평하게 처리하고 있음을 확인할 수 있다.

* 텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

  자동 생성된 설명
* make check 수행 결과를 캡처하여 첨부
* 텍스트, 스크린샷, 디스플레이, 멀티미디어 소프트웨어이(가) 표시된 사진

  자동 생성된 설명