**Pintos Project 3: Threads**

담당 교수 : 박성용 교수님

학번 / 이름 : 20181662 / 이건영

개발 기간 : 2022.

1. **개발 목표**

Pintos의 쓰레드들이 우선 순위에 의해 스케줄링 되도록 구현하고, 추가적으로 BSD 스케줄러를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Alarm Clock

쓰레드를 지정된 시간 만큼 blocked 상태로 만든 후 다시 ready 상태로 만들 수 있다.

* 1. Priority Scheduling

각 쓰레드들이 priority 값에 의해 스케줄링 된다.

* 1. Advanced Scheduler

nice 값과 recent\_cpu 값에 의해 쓰레드들의 priority가 조정되며 효율적인 스케줄링이 가능하게 한다.

* 1. **개발 내용**

timer\_sleep() 함수로 인해 쓰레드가 blocked 상태로 바뀔 때, 쓰레드가 깨어나야 할 시간을 구조체 내에 저장한 후, 해당 시간이 되었을 때 thread\_unblock() 함수로 깨우면 된다.

만약 ready\_list에 들어온 새로운 쓰레드의 priority 값이 현재 running 쓰레드의 priority 값보다 높다면, 새로운 쓰레드가 cpu를 점유해야 하므로 thread\_yield() 함수를 사용하여 새로운 쓰레드가 running 상태가 될 수 있도록 한다.

Advanced Scheduler에서 priority 계산에 필요한 요소는 load\_avg, recent\_cpu, nice 값이 있다.

load\_avg는 현재 ready 상태인 쓰레드들의 수에 비례하여 증가하는 값으로, 현재 시스템의 load average 값을 나타낸다. 전체 프로그램에서 하나의 값을 사용한다.

recent\_cpu는 각 쓰레드마다 존재하는 값으로, 각 쓰레드의 CPU time을 평가하기 위한 값이다.

nice 값은 recent\_cpu 값과 priority 값을 계산할 때 가중치의 역할을 하며, nice가 0보다 클 경우 결과적으로 priority를 낮추는 역할을 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

Alarm clock : 22.11.01~22.11.03

Priority with aging : 22.11.04~22.11.07

BSD Scheduler : 22.11.08~22.11.10

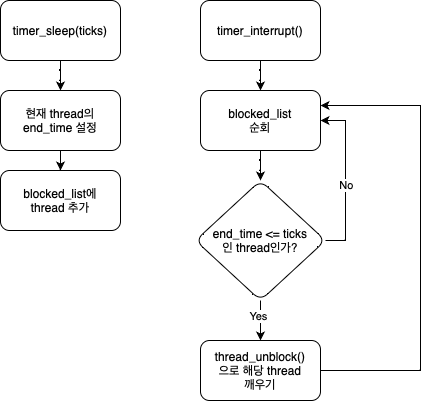
* 1. **개발 방법**

Alarm clock을 구현하기 위해서는 timer.c 파일의 함수들을 수정하여야 한다. 기존의 timer\_sleep() 함수는 쓰레드를 지속적으로 running과 ready 상태를 반복하도록 하는 방식으로 다소 비효율적이었다. 이를 개선하기 위해 우선 struct thread 구조체에 end\_time 이라는 변수를 추가하여 쓰레드가 깨어나야 할 시간을 저장하고, timer.c의 전역변수로 blocked\_list를 추가하여 sleep 상태에 있는 쓰레드들을 관리한다. 또 timer\_sleep() 함수에서 쓰레드의 end\_time을 설정한 후 blocked\_list에 추가하고 thread\_block() 함수로 쓰레드를 blocked 상태로 만든다. 이후 timer\_interrupt 함수에서 지속적으로 blocked\_list의 쓰레드들의 end\_time을 체크하며 깨어나야 할 쓰레드들을 thread\_unblock() 함수로 깨워준다.

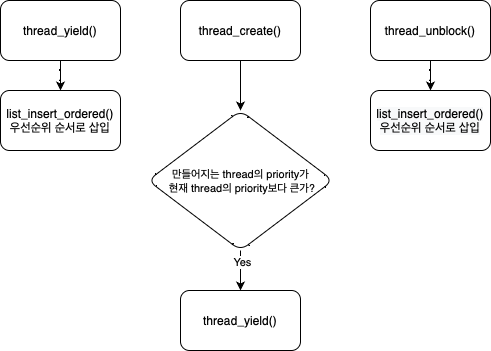
priority를 구현하기 위해서는 ready\_list의 쓰레드들을 priority 순으로 정렬하여 schedule() 함수에서 실행시킬 다음 쓰레드를 고를 때 우선순위가 높은 쓰레드가 선택되도록 하여야 한다. 이를 위해 list\_push\_back() 함수로 ready\_list에 쓰레드를 추가하던 방식에서 list\_insert\_ordered() 함수를 사용하여 priority 순으로 쓰레드가 추가되도록 수정하였다. 또한 semaphore를 사용하여 waiters 목록에 쓰레드가 추가 될 때에도 이를 고려하여 추가되도록 하고, sema\_up 시 waiters list를 한번 list\_sort() 함수를 사용하여 정렬하도록 하여 priority 순으로 깨어나도록 구현하였다.

BSD 스케줄러의 경우 struct thread 구조체에 nice와 recent\_cpu 값을 추가하고, thread.c에 static 전역변수로 load\_avg를 추가하였다. 또 각 값을 반환하는 thread\_get\_~~() 함수와 값을 설정하는 thread\_set\_~~() 함수를 구현하였다. 마지막으로 update\_~~() 함수를 사용하여 지정된 시간마다 해당 값들이 갱신되도록 구현하였다.

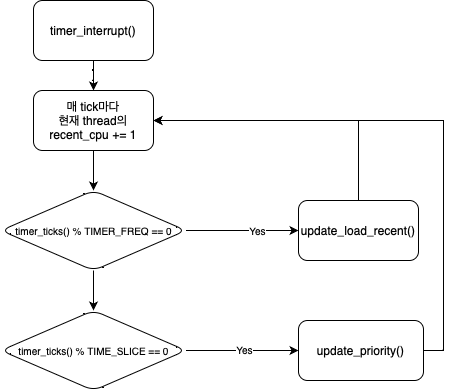
1. **연구 결과**
   1. **Flow Chart**

****

**<Alarm Clock>**

****

**<Priority Scheduling>**

****

**<Priority aging & BSD Scheduler>**

* 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

우선 timer.c에서 수정한 내용이다.

텍스트, 표지판, 점수판이(가) 표시된 사진

자동 생성된 설명

BLOCKED 상태의 쓰레드를 관리하기 위한 blocked\_list를 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음으로는 timer\_sleep() 함수를 수정하였다. 함수가 호출될 당시의 시간을 start 변수에 저장하고 이를 쓰레드를 잠재울 시간인 ticks와 더해 현재 쓰레드의 end\_time을 설정하였다. 이후 해당 쓰레드를 blocked\_list에 추가하고 thread\_block() 함수를 통해 쓰레드를 BLOCKED 상태로 만들었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 timer\_interrupt() 함수에서 blocked\_list를 순회하며 각 쓰레드들의 end\_time이 지났는지 확인하여 깨어나야 할 쓰레드들을 thread\_unblock() 함수를 통해 깨워준다.

다음으로는 priority 부분이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

ready\_list에 쓰레드를 추가하는 부분을 list\_insert\_ordered 함수로 바꾸었다. 이 때 세 번째 인자인 compare\_pri 함수는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같이 두 쓰레드의 priority를 비교하여 새로 들어갈 쓰레드의 위치를 찾도록 도와준다. 그러나 위의 구현만으로는 priority-sema 테스트를 통과할 수 없었기에, synch.c 역시 수정하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

sema\_down으로 waiters 리스트에 쓰레드를 추가할 때에도 priority 순서대로 들어가도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 sema\_up 호출 시 waiters 리스트를 list\_sort를 통해 정렬하도록 하였다. 이후 thread\_pri\_reload() 함수를 통해 새롭게 스케줄링을 할 수 있도록 하였다.



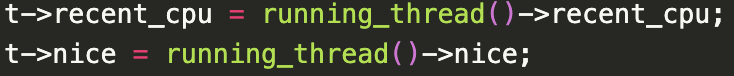
thread\_pri\_reload() 함수는 위와 같이 ready\_list에서 우선순위가 가장 높은 쓰레드와 현재 쓰레드를 비교하여 현재 쓰레드의 priority가 더 높다면 thread\_yield() 함수로 새롭게 스케줄링 하도록 구현하였다.

다음으로는 priority aging과 BSD Scheduler 부분이다. 우선 struct thread에 새로운 변수를 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 init\_thread() 함수에서 해당 값들을 초기화하는 부분을 추가하였다.



다음으로 nice와 load\_avg, recent\_cpu 값을 사용하여 priority를 계산하는 기능을 구현하여야 한다. 이를 위해 fixed-point 계산을 이용하였다. Pintos에서는 17.14 format을 사용한다. 따라서 다음과 같은 값을 정의하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각 값의 계산에는 명세서와 매뉴얼을 참고하여 다음 수식을 사용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

또한 fixed-point arithmetic을 위해 다음의 식을 사용하였다.

테이블이(가) 표시된 사진

자동 생성된 설명

x와 y는 fixed-point이며 n은 정수, f는 위의 FRACTION이다. 위의 수식 중 가까운 정수로 rounding하는 함수인 near\_int() 함수를 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음으로 load\_avg와 recent\_cpu 값을 갱신하는 함수 update\_load\_recent() 함수를 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이를 바탕으로 priority를 계산하는 update\_priority() 함수도 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

만약 priority의 계산 결과, 현재 실행 중인 쓰레드의 priority 값보다 더 큰 값을 지닌 쓰레드가 생겼다면 이를 위해 intr\_yield\_on\_return() 함수를 사용하여 스케줄링 해준다.

마지막으로 이 함수들을 매 시간간격으로 호출해야 하므로 timer\_interrupt() 함수에서 호출하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

명세서에 따라 TIMER\_FREQ 값과 TIME\_SLICE 값을 사용하여 정해진 주기마다 호출되도록 구현하였고, 매 tick 마다 현재 쓰레드의 recent\_cpu 값을 증가시키도록 하였다.

다음으로는 nice 값을 설정하는 thread\_set\_nice() 함수를 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

쓰레드의 nice 값이 변경되면 이에 따라 priority 값도 변경되므로 이를 update한 후 thread\_yield() 함수를 사용해 새로 스케줄링 하는 부분까지 추가하였다.

또한 각 값들을 반환하는 함수들을 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

이 때 값들이 fixed-point 값이면 near\_int() 함수를 사용하여 정수로 변환하였다.

* 1. **시험 및 평가 내용**

priority-lifo 의 경우, priority-fifo와 비교하였을 때 각 쓰레드의 priority 값을 지정하는 방식이 달랐다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

좌측이 lifo, 우측이 fifo의 코드이며 thread\_create 부분에서 priority 값을 지정하는 부분이 다른 것을 확인하였다. 따라서 priority-lifo의 경우 나중에 생성된 쓰레드의 우선순위가 높다고 예측할 수 있었다. 실행 결과는 다음과 같았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

예상하였던 대로 나중에 생성된 쓰레드(=id의 값이 큰 쓰레드)가 priority 값이 높았고, 따라서 매 iteration에서 우선 순위별로 하나의 쓰레드만 실행된 것을 확인하였다.

make check의 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명