**Pintos Project 3: Threads**

담당 교수 : 문의현

학번 / 이름 :

개발 기간

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

alarm clock과 priority scheduling을 구현하도록 한다. alarm clock을 구현 시에, 새로운 ready\_list 외의 새로운 큐를 만들어 ready와 running을 오가며 계속 기다리는 비효율성을 해결해야 한다. priority scheduling은 구현 시에, 현재 우선순위를 고려하지 않는 round robin을 priority에 따라 schedule하도록 해야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

Alarm Clock은 timer\_sleep의 비효율성을 해결하기 위해 필요한 개발 항목이다. 원래의 timer\_sleep은 스레드를 그저 yield 하여 양보를 할 뿐, block 상태로 들어가는 것이 아니다. 따라서 계속 스케쥴링 되어서 run-state로 올라오길 반복하며 기다리게 되고, 이는 비효율적인 cpu scheduling이라 할 수 있다. context change의 오버헤드가 들어가기 때문이다. 이를 위해 alarm clock을 개발하면, 스레드를 그저 재우고, 일정 시간이 지나면 깨우면 된다. 따라서 비효율성이 해결된다.

* 1. Priority Scheduling

Priority scheduling은 현재 스케쥴링 방법이 round-robin인 핀토스에서, 그 방법이 priority를 고려하는 round-robin이 되도록 고치는 것이다. 즉, 일정 시간만큼 스레드가 cpu를 점유하도록 하되, priority를 1순위로 고려하여 스케줄링하고, 이를 위해 필요한 경우 preemption까지 진행할 수 있도록 만드는 것이다. 이렇게 하면, priority를 통해 스케쥴링을 관리할 수 있다.

* 1. Advanced Scheduler (추가구현을 한 경우)

X

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

blocked 된 상태의 스레드는 일정시간마다 발생하는 timer interrupt를 이용할 수 있다. 즉, timer interrupt에 호출되는 timer interrupt handler인 timer\_interrupt() 함수에서 각 스레드가 기다려야 할 ticks 만큼 시간이 지났는지를 확인하는 함수를 구현하면 된다. 그 경우에 스레드를 unblock 하여 깨우고, ready\_list에 넣어줄 수 있다.

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

ready list에 running thread보다 높은 우선순위의 thread가 들어온 경우, 현재 스레드가 cpu 점유권을 바로 내주어야 한다. 이를 위해서, ready\_list에 그 스레드(높은 스레드)가 우선순위에 따라 정렬된 상태로 들어가도록 하고, 들어왔을 때, yield()하도록 해야 한다. 이렇게 높은 우선순위가 ready\_list에 들어와 scheduling에서 고려해야 하는 경우는 여러 경우가 있는데, 새 스레드가 생성될 시, 우선순위가 변경될 시, sema\_wait\_list에서 나올 시, sleep 상태에서 깨어날 시 등이 있다.

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

X

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

구현 일정은 다음과 같이 짤 수 있다.

1일 차: 과제 명세서와 과제 강의를 보며 프로젝트 이해 및 개발 목표 작성

2일 차: 과제 명세서와 pintos document를 보며 구체적인 구현 내용 작성

3일 차: alarm clock 구현

4일 차: priority scheduling 구현, make 시 나오는 컴파일 오류 디버깅, grade 시 발생하는 디버깅

5일 차: test case를 보며 디버깅, 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

II. B. 1)을 개발하기 위해서 다음 소스코드를 수정해야 한다. (thread.h, thread.c, timer.c) 자료구조에 있어서는, struct thread 자료구조를 수정해야 하는데, wait\_time을 저장하여 언제 깨어나야 할 지를 저장하도록 한다. 또한, sleep\_list의 자료구조 선언도 필요해진다. thread\_sleep()을 구현하여 추가하여야 하고, timer\_sleep()의 내용을 수정하고, timer\_interrupt() 함수를 수정하고, thread\_alarm()을 구현하여 추가하여야 한다.

II. B. 2)를 구현하기 위해서는 다음 소스코드를 수정해야 한다. (thread.c, synch.c) (aging 관련하여 과제 조건을 충족하기 위해, thread.h, init.c, thread.c를 또한 수정해야 한다.) 추가하는 자료구조는 없지만, 함수를 추가하고 수정해야 한다. thread.c 안에 thread의 priority를 내림차순으로 비교해주는 함수 thread\_less\_func를 구현하여 비교함수로 list를 정렬할 수 있도록 해야 한다. 그 후, list에 우선순위에 따라 정렬된 상태로 스레드가 들어갈 수 있도록 list\_push\_back()으로 구현된 부분을 list\_insert\_ordered()로 고쳐준다. (이는 thread\_unblock, thread\_yield, sema\_down 등의 수정을 가져온다.) 마지막으로, thread\_preemption()을 구현하고, unblock이나 새로운 스레드의 생성 등으로 인해 우선순위 확인 및 preemption이 필요한 경우에 쓸 수 있도록 한다. (이는 sema\_up, thread\_set\_priority, thread\_create 등의 수정을 가져온다.)

1. **연구 결과**
   1. **Flow Chart**

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

(죄송합니다. 시간 관계상 작성하지 못했습니다.)

* 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

II. B. 1)을 개발하기 위해, 먼저 thread.h에 wake\_up\_time을 스레드별로 각각 저장할 수 있도록 thread 구조체를 수정해야 한다. 이로써 각각 언제 sleep 상태가 풀려야 할지를 알 수 있다. 또한, thread.c에 static으로 struct list sleep\_list를 선언해주도록 한다. 이를 수정하고 난 후, timer.c의 timer\_sleep에서 thread\_sleep()을 호출하도록 고친다. thread\_sleep()은 thread.c에 구현하는데, 정해진 ticks까지만 스레드가 sleep 상태로 있게 하는 것이다.

thread\_sleep()은 interrupt를 끈 후 진행해야 하고, 다시 원래 상태로 돌려야 함에 주의한다. 호출 전에 interrupt가 켜진 상태였기 때문이다. 그 함수 안에서 wake\_up\_time을 설정한 후, sleep\_list에 넣어주고, thread\_block()을 한다.

이러한 block되고 sleep\_list에서 관리되는 스레드들을 깨우기 위해서는 다음과 같이 구현한다. timer.c의 timer\_interrupt() 함수의 안에서 thread\_alarm()을 호출하도록 한다. 이렇게 timer interrupt handling을 이용할 수 있는 것이다. 각 스레드가 기다려야 할 ticks 만큼 시간이 지났는지를 주기적으로 확인하는 이 함수는 sleep\_list에 있는 모든 스레드의 필요 시간이 지났는지를 확인하고, 지난 경우에 list에서 제거한 후 unblock 해준다. 이로써, ready\_list에서 계속 running 상태로 들어오는 비효율을 막게 된다.

개발 중 발생한 문제는 특별히 없었지만, sleep\_list와 ready\_list에 elem이 공통으로 들어갔다가 빠져나오는 과정이 있기 때문에 list의 관리에서 약간의 난관을 겪었다. priority 관련 문제로 인해 alarm-priority 테스트가 통과하지 못했지만, 문제를 찾아 해결하게 되었다. (priority 관련 이후 서술.)

II. B. 2)를 구현하기 위해서, priority에 따라 스레드가 정렬되어 들어가고 priority 순으로 cpu 점유가 진행되도록 로직을 짜야 한다. priority 순으로 ready\_list 혹은 sema의 wait\_list를 정렬하는 것을 통해 list의 맨 앞에서 필요한 스레드를 뽑는 기본 로직을 수정하지 않고도 priority scheduling을 구현 가능하게 할 수 있다. 여기서 정렬을 위해, 비교함수 thread\_less\_func를 구현하여 내림차순으로 list를 정렬하는 것이 가능하도록 만든다. 그 후, push\_back을 모두 insert\_ordered로 대체해주면 된다. 이로써, priority에 따른 list 정렬과 활용이 가능해진다.

추가적으로 구현해야 될 부분은 preemption과 aging이다. 현재 cpu의 스레드보다 높은 우선순위 스레드가 있는 경우 preemption하는 방식이 필요한데, 일단 preemption 과정을 thread\_preemption()에 구현한다. 그 다음, 높은 우선순위가 들어올 가능성이 있는 경우를 모두 찾아 이 함수를 호출하면 된다. 즉, ready\_list에 들어오게 되는 경우를 모두 조사하면 다음과 같다. semaphore에서 wait\_list에서 빠져나와 unblock되었을 때, 이 스레드가 우선순위가 높을 수 있으므로, 이를 체크해야 한다.(sema\_up) 또한, thread\_set\_priority가 우선순위를 바꿨을 때, ready\_list의 우선순위에 변동이 있음을 의미하고, 테스트 케이스에서 지시되는 것과 같이 preemption이 필요하다. 마지막으로, thread\_create 시, 즉 스레드 생성 시 새로 들어온 스레드의 우선순위가 더 높을 수 있으므로, preemption check 및 진행이 필요하다.

개발 중 내림차순으로 구현해야 할 것을 습관처럼 오름차순으로 구현하여 alarm-priority가 실패하고 모든 priority test가 실패하는 문제가 발생하였었다. 여러 소스코드와 테스트를 살피다가, alarm-priority 테스트 케이스가 거꾸로 출력되는 것을 인지하고 이를 수정할 수 있었다.

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

test\_priority\_lifo()에서는 각 priority가 서로 다른, 점차 증가하는 16개의 스레드를 띄운 후, 각 스레드에서 16번 만큼 iterate를 실행하게 된다. 만약 priority가 낮은 스레드가 먼저 실행되거나 하면서 일정 순서가 섞이는 경우, test case는 통과하지 못하는 것이다.

이 테스트 코드를 실행하여 다음과 같은 결과가 나왔다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 결과를 확인할 때, 스레드의 우선순위 순서에 따라 올바르게 iterate 하고 있다. iteration이 섞이지 않았으므로 test가 통과된 것으로 생각할 수 있고, lifo의 뜻처럼 가장 높은 우선순위가 가장 마지막에 들어갔지만, 가장 먼저 출력되는 것을 통해 테스트의 목적을 다시 확인할 수 있다.

* make check 수행 결과를 캡처하여 첨부

텍스트이(가) 표시된 사진

자동 생성된 설명