**Pintos Project 3: Threads**

담당 교수 : 김영재

학번 / 이름 : 20171692 / 조린

개발 기간 : 3주

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

thread가 blocked 상태가 되었을 때 cpu를 계속 점유하지 않도록 block과 unblock을 구현함. 모든 process가 round-robin 방식을 통해 scheduling 되는 것이 아닌 priority를 통한 preemptive scheduling을 구현한다. OS에서 priority를 실시간으로 조정해 convoy-effect가 발생하지 않도록 하는 BSD-scheduler(mlfq)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

프로세스가 block되었을 때 기존에는 cpu를 점유하면서 기다리는 것이 전체적인 수행시간에 좋지 않기 때문에 block되었을 때 cpu를 양보하도록 하는 것이 필요하다.

unblock될 시간을 정해놓고 block되는 프로세스들을 매 time interrupt마다 확인하며 unblock하고 block되어있는 프로세스가 cpu를 차지하지 않고 다른 대기중인 프로세스가 수행될 수 있게 된다.

* 1. Priority Scheduling

모든 프로세스가 동일하게 round-robin을 통해 수행되는 것은 우선적으로 수행되어야 하는 프로세스의 선점이 불가능하게 된다.

프로세스의 priority를 통해 우선도가 높은 프로세스를 먼저 수행할 수 있게 된다.

프로세스보다 낮은 우선도의 lock에 의해 lock이 풀리지 않는 상황을 priority-donation을 통해 처리할 수 있게 된다.

* 1. Advanced Scheduler (추가구현을 한 경우)

프로세스의 우선도가 코드에 의해서만 변경 될 경우 우선도가 낮은 프로세스는 대기시간이 길어져 convoy-effect가 발생할 수 있다.

BSD-scheduler를 통해 OS자체에서 모든 프로세스들의 우선도를 일정 시간마다 조정해 convoy-effect가 발생하지 않도록 하고, 악의적인 프로세스들에 대해서도 대처할 수 있도록 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

매 time interrupt마다 blocked process list를 탐색하며 unblock되어야 할 시간이 되면 ready queue로 올린다. lock을 기다리던 프로세스의 경우 lock을 release할 때 해당 lock을 기다리며 block queue에 있던 프로세스들을 unblock해준다.

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

추가적인 priority의 조정이 없다면 새로 들어온 높은 priority의 thread가 수행되기 시작한다.

새롭게 수행되는 thread가 이전 프로세스가 점유중인 lock을 요청하는 경우 lock을 영원히 반환받을 수 없기 때문에 priority donation을 통해 이전 프로세스들도 수행되게 해 lock을 반환받을 수 있도록 한다.

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

Fixed-point - pintos상에서는 float point연산이 지원되지 않기 때문에 분수와 소수점 이하의 연산을 하기 위해서 fixed-point연산을 정의하고 사용한다.

Nice - thread가 다른 threads에 cpu를 양보하는 정도를 나타낸 변수이다. 낮을 수록 자신이 cpu를 많이 점유하도록 priority가 증가하게 만든다. OS가 주기마다 변경하는 값이 아니고 자체적으로 설정해야 한다.

Recent\_cpu - 각각의 thread가 최근 cpu를 점유한 시간을 추정하는데 쓰인다. 최근에 많이 cpu를 점유하고 있다면 priority가 낮아져 다른 thread들이 cpu를 사용하게 하는 경향이 있다.

Load\_avg - 최근 1초동안 시스템상에서 수행중인 프로세스들의 수를 반영하게 된다. 값이 높을수록 recent\_cpu가 영향을 많이 받게 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

10/26 ~ 10/28 - alarm clock에 필요한 thread sleep, awake함수 구현, devices와 연결.

10/29 - alarm과 관련된 test를 통해 오류 수정

10/30 ~ 11/2 - priority scheduling을 위한 list compare function, preemption구현. lock에 의해 발생하는 문제를 해결하기 위한 priority donation구현

11/3 - priority와 관련된 test를 통해 오류 수정

11/4 ~ 11/5 - BSD-scheduler를 구현하기 위한 fixed-point arithmetic, nice, recent-cpu, load-average 추가 및 일정 주기마다 값을 재설정하도록 구현

11/6 - mlfq와 관련된 test를 통해 오류 수정 및 보고서 작성

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드

기존에 push\_back하던 queue들이 우선도를 기준으로 정렬되게 하도록 insert함수 수정, BSD-scheduler사용 시 계산되어야 하는 변수들을 다루는 함수 추가, 일정 time interrupt마다 변수들을 재설정하는 함수 추가.

Threads/synch.c에서 priority donation이 가능하도록 lock과 관련된 함수들 수정.

* + 수정하거나 추가해야 하는 자료구조

thread가 sleep할 때 다시 깨어날 시간을 알 수 있도록 하는 변수 추가

thread가 nice와 recent\_cpu를 저장할 수 있도록 변수 추가.

* + 수정하거나 추가해야 하는 함수

Devices/timer.c에서 매 time interrupt발생 시 blocked list를 탐색하도록하는 함수를 추가.

sleep할 때 해당 thread를 block하도록 하는 함수 추가.

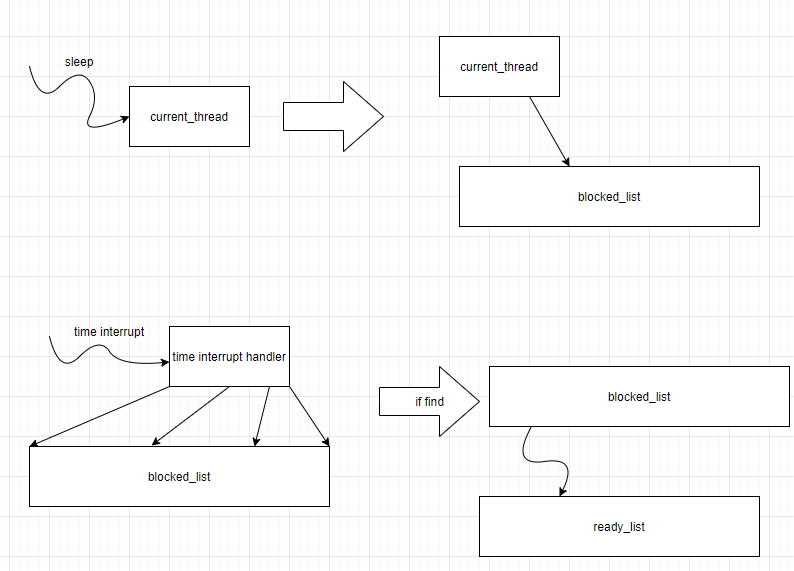
Threads/thread.c에서 priority가 변경될 때 마다 가장 높은 우선도를 가진 thread를 수행하도록 하게 하는 preemption함수 추가

기존에 push\_back하던 queue들이 우선도를 기준으로 정렬되게 하는 compare함수 추가

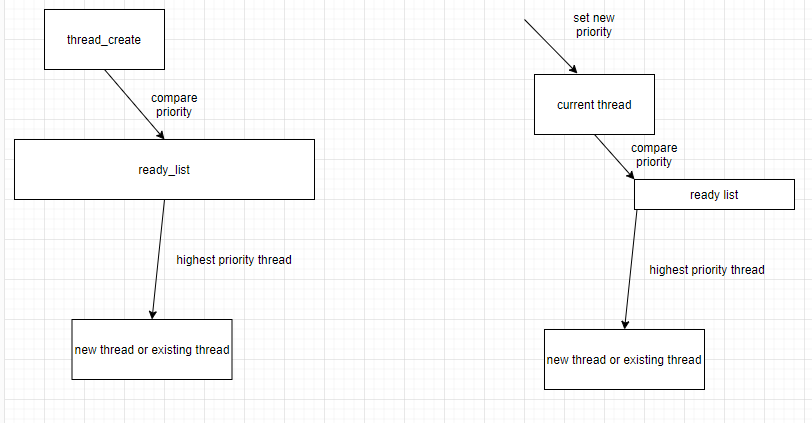
1. **연구 결과**
   1. **Flow Chart**

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

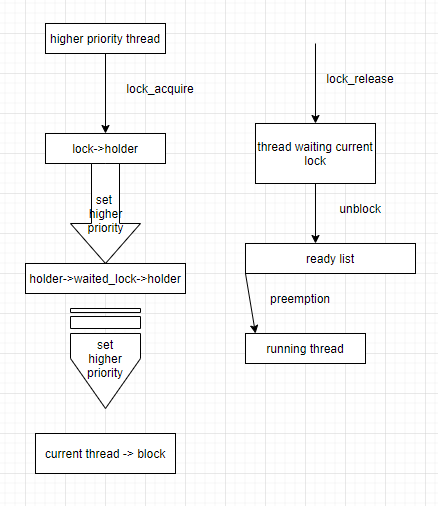
1. alarm clock



2. priority scheduling



3. priority donation



* 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

sleep명령 수행시 block이 끝나는 시간을 저장한 뒤 block\_list에 추가.

매 time interrupt마다 block list의 원소를 순차적으로 탐색하며 thread의 block이 끝나는 시간이 되었다면 unblock하도록 thread\_awake함수 구현

Ready list에 있는 thread의 priority가 변경되거나, ready list에 새로운 thread가 추가되는 경우에 current thread와 ready list의 front원소의 priority를 비교해 높은 priority의 thread가 수행될 수 있도록 한다. ready list에 thread를 추가할 때 priority에 따라 정렬된 상태를 유지하도록 list\_insert\_ordered를 통해 추가함.

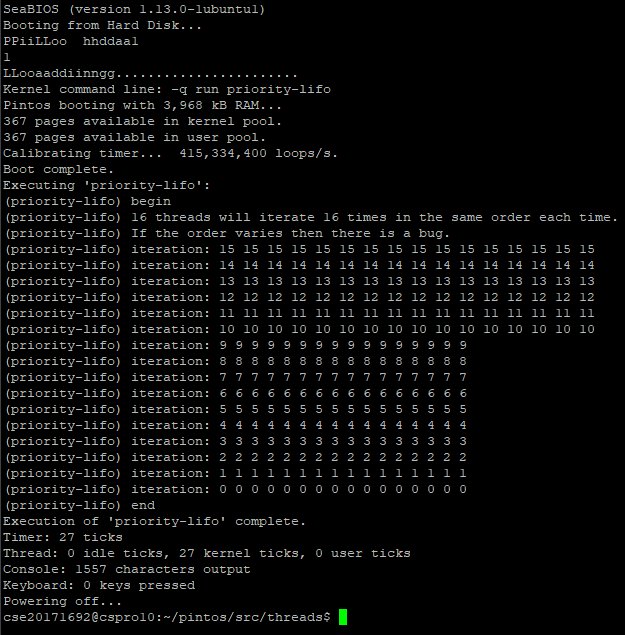
lock을 점유하려 할 때 이미 점유하고 있는 thread가 있다면 lock에 의해 연결되어있는 thread들의 priority를 자신의 priority로 변경함.

lock을 release할 때 priority를 원래대로 돌리기 위해 자신의 원 priority를 저장할 수 있도록 변수를 추가하고, 받아온 priority와 original priority중 가장 높은 priority로 재설정함. release하는 lock을 기다리고 있던 lock들 중 가장 priority가 높은 thread를 ready list로 이동시켜 priority scheduling이 유지되도록 한다.

Fixed-point를 통한 소수점 연산이 가능하도록 manual에 나와있는 fixed\_point연산을 새로운 파일에 추가. priority-aging을 통해 4tick마다 priority 재설정, 1초마다 recent\_cpu, load\_avg 재설정하도록 한다.

Mlfqs와 관련된 test중에서 priority가 제대로 변경되지 않던 문제가 있었는데, 기존의 방법인 -aging option이 없는 경우라 발생하던 문제였다. 따라서 priority\_aging()을 수행하는 조건을 -aging or -mlfqs일 때로 변경했고, 이후 제대로 수행되는 것을 확인했다.

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



priority가 점점 커지게 설정된 16개의 thread를 생성하고, 각각의 thread는 lock을 acquire한 뒤 result에 자신의 id를 저장한뒤 lock을 release하고 cpu를 넘겨주게된다. 현재 priority scheduling이 구현되어있기 때문에 cpu를 넘겨줬어도 방금 전에 수행했던 thread의 priority가 가장 높기 때문에 바로 다음에 수행되게 된다. 따라서 priority\_fifo와 달리 priority가 높은 순서대로 thread가 수행되는 결과가 나오게 된다.

* make check 수행 결과를 캡처하여 첨부

