**Pintos Project 3 : Thread**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070]운영체제

담당 교수 : 소정민 교수님

조 / 조원 : 41조 20130859 송민규, 20130956 장동욱

개발 기간 : 2018.11.16~12.05

**프로젝트 제목 : pintos project 3 thread**

**제출일 : 2018.12.6**

**참여 조원 : 20130859 송민규, 20130956 장동욱**

1. **개발 목표**

* **현재까지 개발되어 있는 pintos는 scheduling이 round-robin 방식으로 동작하있다. 이를 각 thread 별로 우선순위를 부여하고, 높은 순서대로 선택되어 수행되도록 수정할 것이다. 또한 각 thread들이 부여받은 우선순위에 고정되어 있지않고, 점차 시간이 지남에 따라 우선순위가 변할 수 있게 aging 기법을 도입할 것이다. 마지막으로 pintos의 timer\_sleep을 수정해서 busy waiting이 아닌 방법으로 부하가 적은 방향으로 수정할 것이다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* **Alarm Clock**

**현재 Pintos의 timer\_sleep()은 busy waiting 방식으로 구현되어 있다. Thread가 RUNNING 상태와 READY 상태를 반복하며 설정한 시간이 지나가기를 기다리도록 구현되어 있다. 이러한 방식은 불필요한 thread가 ready 상태가 되어 scheduling 되는 문제가 있다. 이런 불필요한 scheduling이 없도록 수정해서 timer가 적당한 시점에 sleep한 작업을 다시 scheduling시킬 수 있도록 수정할 것이다. 이를 위해 devices/timer.c, threads/thread.c, threads/thread.h를 수정한다.**

* **Priority**

**기존의 Pintos는 thread가 각각 일정한 burst time동안 수행되고 다시 큐로 돌아가는 방식을 취하고 있다. 이보다 더 cpu time을 효율적으로 운용하기 위해, thread의 생성시에 우선순위를 부여하고, scheduling이 될 때마다 높은 우선순위의 thread가 실행되도록 수정한다. 이를 위해 threads/thread.c, threads/thread.h, threads/synch.c를 수정한다.**

* **Aging & BSD scheduling**

**단순히 thread별 priority를 부여하게 되면 starvation 문제가 발생한다. 이는 높은 우선순위를 부여받은 thread만 수행되고 낮은 우선순위의 thread는 영영 실행되지 않을수도 있는 문제가 생긴다. 이를 막기위해 점차 우선순위가 변화하게 되는 BSD scheduling을 도입한다. 이를 위해 threads/thread.h, threads/thread.c, threads/synch.h, threads/synch.c를 수정한다.**

* 1. **개발 내용**
* **Interrupt**

**Pintos의 기능들은 thread를 기반으로 수행된다. 어떤 thread가 점유하고 있을때, interrupt가 발생하면 동작의 관심을 interrupt가 발생한 곳으로 돌릴 수 있다. 특히 HW에서 발생한 사건들을 다룰때 interrupt driven으로 작성한다. 본 프로젝트의 timer\_sleep() 문제도 timer에서 ‘매 tick이 지나가는’ 사건이 기본이기 때문에, interrupt를 이용하여 ‘매 tick이 지나갈 때마다’ 필요한 동작을 수행하도록 한다. 또한 추후 필요한 aging에서도 이를 이용한다.**

* **Priority & Scheduling**

**Ready queue에 FIFO한 방법으로 들어와서 일정 시간동안 점유하고 다시 ready queue로 돌아가는 방식을 round-robin 방식이라고 한다. 이는 기존의 pintos가 취하고 있는 방식이다. 하지만 급하게 수행되어야 하는 thread도 일정시간 이상 기다려야하는 비효율성이 있기 때문에, 우선순위대로 정렬되는 queue를 만들어서 일정시간동안 수행되도록 한다.**

* **Aging & BSD scheduler**

**위와 같은 방식으로 만들면 높은 우선순위의 thread가 계속 발생하면 낮은 우선순위의 thread가 영영 수행되지 못 할 수도 있는 문제가 발생한다. 따라서 시간이 점차 지나감에 따라 priority가 변화하게 만든다. 이를 aging이라 한다. 또한 aging에 따라 우선순위가 다양하게 변화되어 scheduling 되는 queue를 Multi Level Feedback Queue라고 한다. Aging이 되는 방식으로는 manual에서 설명하는 BSD scheduler를 적용한다. 이는 현재 대기, 동작하고 있는 작업의 수, thread의 cpu 사용시간 값, 임의로 부여한 수, 등을 조합하여 priority가 수정되도록 하는 방식이다.**

* **Fixed Point Arithmetic**

**Pintos는 자체적으로 floating point number를 다룰 수 있는 기능이 없다. 그러나 BSD scheduler는 실수값들을 다룬다. 따라서 이를 보완하기 위해 fixed-point 형식을 사용한다. 매뉴얼에서는 32bit를 lsb 14bit를 소수점 이하 이진정보를 저장하고, 최상위 bit는 sign, 그 이외 17bit는 정수부 이진정보를 담는 방식을 고안해야 한다. 또한 정수-실수, 실수-실수 연산을 다루는 함수를 추가해야 한다.**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **11.16-20 코드, 명세서 탐독 및 자료조사, 11.21-12.3 코드 작성, 12.3-12.5 디버깅 및 보고서 작성**
  1. **개발 방법**
* **Alarm Clock**

**Threads/thread.h 의 thread 구조체에 sleep\_time을 설정할 수 있도록 int형 변수를 추가한다. Devices/timer.c에 sleep을 하고 있는 thread들을 담을 list자료형 변수를 추가한다. 해당 list를 timer interrupt를 등록하는 함수에서 initialize 하도록 만들어 준다. timer\_sleep() 함수의 while loop을 제거하고, 특정시간에 일어날 수 있도록 시각을 설정해서 thread의 sleeptime에 넣어주고, scheduling이 되지 않도록 세팅하고, sleep\_list에 넣어준다. timer\_interrupt가 발생하면 sleep의 단위인 tick이 지나간 거기 때문에, timer\_interrupt() 함수에서 sleep\_list중 일어날 thread들을 찾아 다시 scheduling이 되도록 설정해준다.**

* **Priority**

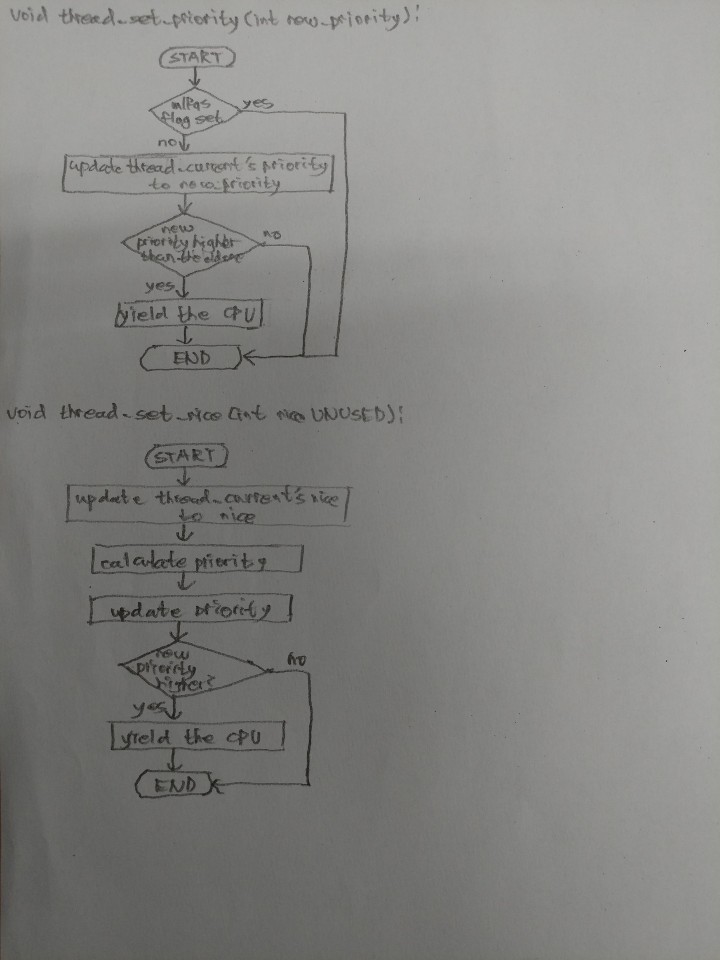
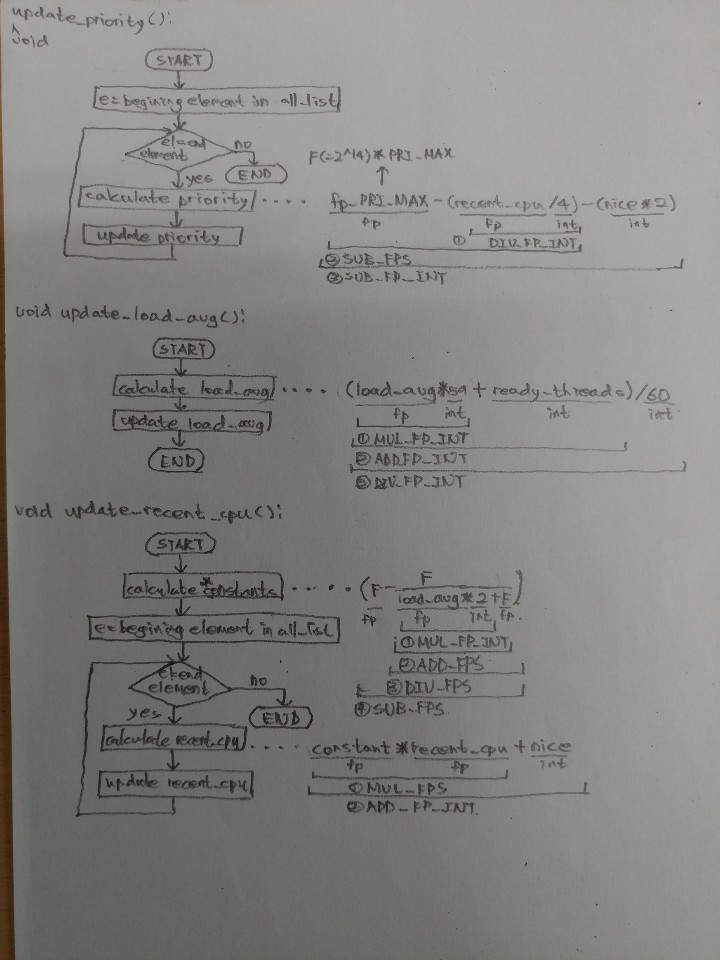
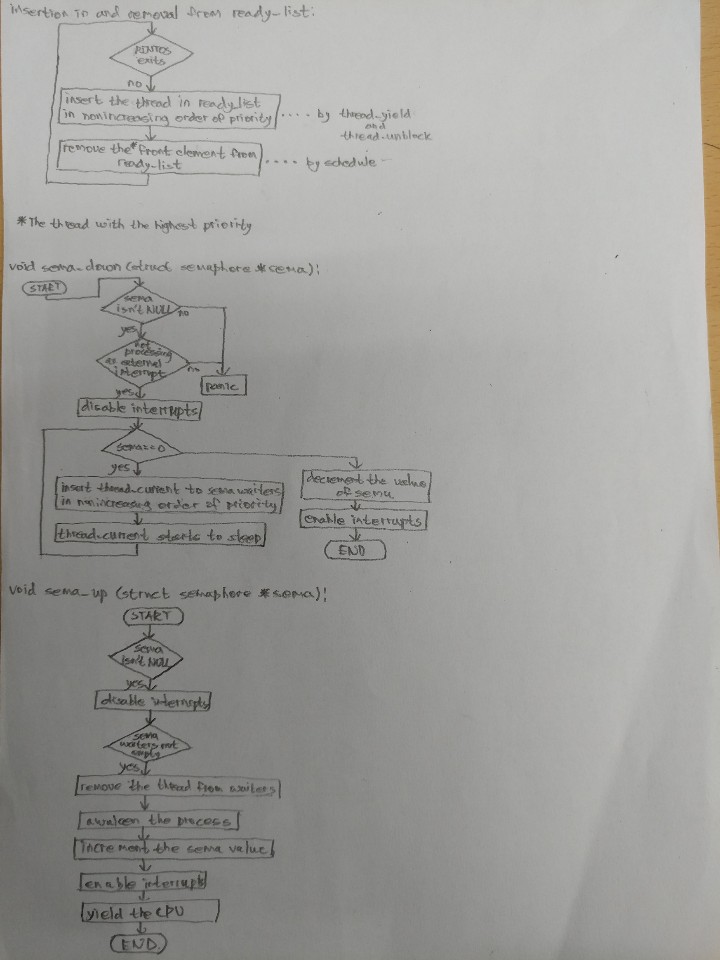
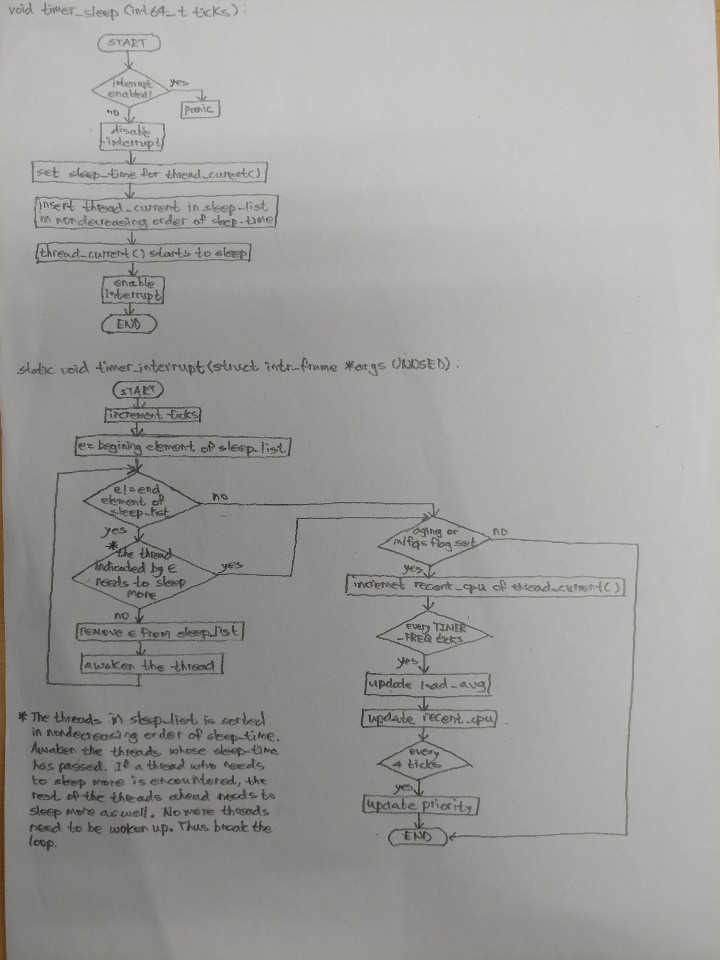
**Priority를 저장하는 변수는 이미 thread.h에 thread구조체 선언에 있다. 기존의 scheduling은 thread\_yield() 함수를 이용해 수행되었다. 이 함수는 ready queue의 마지막에 thread를 넣고 scheduling을 함수이다. Thread\_yield를 priority 기준으로 queue에 넣을 수 있도록 수정하고, thread\_unblock()에서도 이처럼 동작하도록 수정한다. 왜냐하면 blocked thread의 경우 thread\_yield()가 아닌 thread\_unblock()이 호출되어 ready queue로 진입하기 때문이다.**

* **Aging & BSD scheduling**

**threads/thread.h의 thread 구조체에 int형 nice 변수를 추가하고 init\_thread() 함수에서 부모 thread의 nice 값을 받도록 설정한다. threads/thread.h의 thread 구조체에 int형 recent\_cpu 변수를 추가하고 init\_thread() 함수에서 부모 thread의 nice 값을 받도록 설정한다. Load\_avg와 recent\_cpu는 int형 자료형이지만, 실제로 다루는건 fixed point의 실수이다. fixed point 실수의 연산을 위해 필요한 기능은 macro로 thread.h에 구현한다. thread.c에 전역변수 load\_avg를 추가한다. 이를 기반으로 update\_load\_avg(), update\_priority(), update\_recent\_cpu() 함수를 thread.c에 구현한다. 위 함수들에 대한 접근을 thread\_aging으로 하도록 만들고, timer\_interrupt가 발생하면 thread\_aging이 호출되도록 만든다. 또한 test코드에서 접근하기 위해 thread\_get\_priority (), thread\_set\_priority (), thread\_get\_nice (), thread\_set\_nice (), thread\_get\_recent\_cpu (), thread\_get\_load\_avg () 함수를 구현한다.**

* 1. **연구원 역할 분담**
* **각자 일정정도 구현하면서 막히는 부분을 같이 풀어보는 식으로 구현한다. 일정정도 진행되면 진행된 것을 한 명의 코드로 취합한 다음, 남은 구현을 마무리하고 결과 검토와 디버깅을 진행한다. 이때 다른 구성원은 보고서 작성을 한다.**

1. **연구 결과**
   1. **합성 내용**

****

* 1. **제작 내용**
* **Alarm Clock**

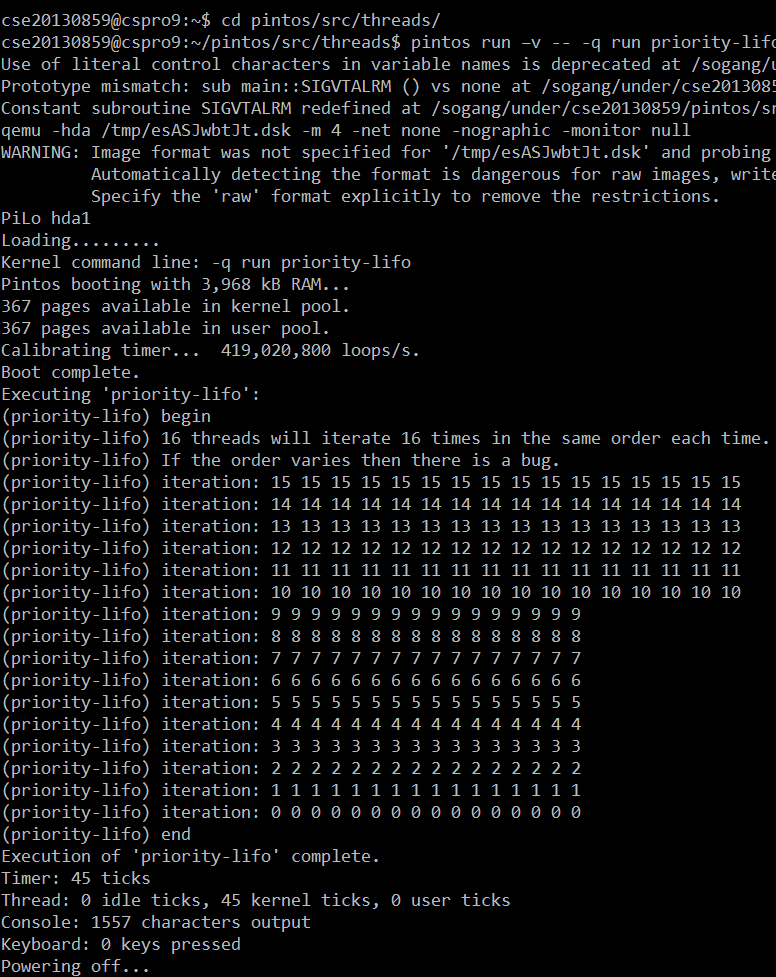
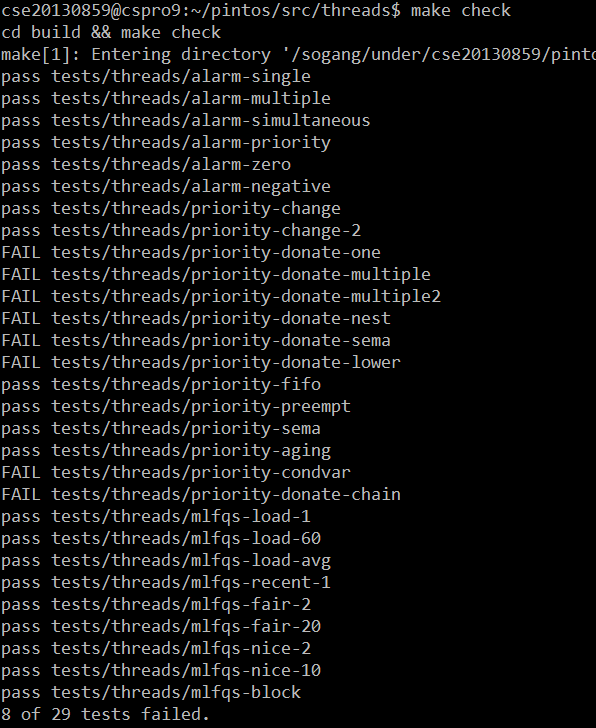
**timer\_sleep에서 sleep\_time은 현재 시간과 인자로 넘어온 tick의 합으로 설정한다. 그래야 미래 일어날 시간이 설정된다. 이를 기준으로 Sleep\_list는 항상 가장 낮은 sleep\_time을 가진 순으로 정렬되어 있어야 한다. 이를 위해 list에서 제공하는 list\_insert\_ordered() 함수를 이용한다. list\_insert\_ordered()에 필요한, 각 thread의 sleep\_time을 비교하는 sleep\_time\_cmp()함수를 thread.c에 구현한다. 그리고 timer\_interrupt()함수는 현재시간보다 이전의 sleep\_time을 가진 thread들을 sleep\_list에서 제거하고 unblock 시킨다.**

* **Priority**

**Sleep\_time을 비교해 sleep\_list의 순서를 유지했듯이, priority를 비교해 ready\_list를 priority가 높은 순으로 유지해 준다. 이를 위해 list\_insert\_ordered()를 이용하고 비교하는 함수로 pri\_cmp()를 만든다. 이때 pri\_cmp()의 부등호의 방향은 sleep\_time\_cmp()의 반대이다. 이를 thread\_yield()와 thread\_unblock()에 적용한다. 또한 thread가 생성될 때, 새로 생성되는 thread의 priority가 높은 경우에는 rescheduling을 하도록 thread\_yield 함수를 thread\_create에 넣어준다. 또한 sema로 인해 block되어있는 경우 이를 up하는 건 waiter에서 thread순서가 나오기 때문에, down에서 list\_insert\_ordered를 이용해 항상 waiter내 thread가 우선순위가 높은 순서에서 낮은 순서로 정렬되도록 만든다.**

* **Aging & BSD scheduling**

**BSD scheduling의 수식을 구현하기 위해선 fixed point 실수 연산을 해야한다. 이때 주목해야하는 부분은 fixed point는 하위 14bit를 소수점 이하를 표현하는데에 사용하고 있다는 점이다. 즉, int 3은 fixed point로 표현될 때 3\*2^14의 값이 int형 자료형에 들어가는 것으로 표현되는 것이다. 이를 이용해서, 실수-실수, 실수-정수간의 사칙연산을 macro로 만든다. 기본적으론 정수에 2^14를 곱해 실수와 더하거나 빼서 실수결과를 반환하고, 곱하고 나누는건 실수와 실수 사이의 경우 2^14로 한쪽을 나눠서 곱하거나 나누는 식으로 진행한다. Timer\_interrupt에서 tick을 봐서 지정한 시간마다 이를 이용해 구현한 update\_load\_avg(), update\_priority(), update\_recent\_cpu() 함수를 호출한다. 단 이는 명령어 단에서 –aging이, -mlfqs가 들어간 경우에만 동작하도록, init.c의 parsing에서 해당 명령어가 들어간 경우 aging, mlfqs boolean을 setting한다.**

* 1. **시험 및 평가 내용**
* 

1. **기타**
   1. **연구 조원 기여도**

* **송민규(50%), 장동욱(50%)**
  1. **소감**
* **Mlfqs의 구현을 확인하는 testcase와 그 코드 이해가 필요했다. 정답의 가부를 일치여부가 아니라 범위에 포함되는 지의 여부로 표현되고 명확한 기대출력이 명시적으로 .ck 에 작성되어있지 않아, 이를 이해하고 시험해보는데에 시행착오가 있었다.**
* **Fixed point arithmetic을 위해 bit단위 연산을 알아보거나 pintos에서 제공하는 bitmap 기능을 알아봤었다. 그러나 fixed point의 특징적인 부분을 단순화해서 int 32비트 자료형의 성질을 이용해서 단순화해 구현할 수 있었다.**