**Pintos Project 3: Threads**

담당 교수 : 박성용 교수

학번 / 이름 : 20181589 / 강민석

개발 기간 : 10/28 ~ 11/10

1. **개발 목표**

* 현재 핀토스에서는 round-robin의 기본 scheduler 만을 사용한다. 그렇기 때문에 각 프로세스나 thread의 priority를 고려하지 않는다. 그러므로 주어진 thread의 시스템을 고려해서 Alarm Clock, Priority Scheduling, Advanced Scheduler (BSD Scheduler)를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Alarm Clock
      * 기존의 timer\_sleep() 함수는 계속해서 RUNNING과 READY state를 반복하는 busy waiting 식으로 구현이 되어 있다. 이 방식은 비효율적이기 때문에, while문을 반복적으로 수행하면서 시간을 체크하지 않고, 현재 수행 시간을 체크해서 Thread 를 BLOCKED 상태로 만든다. 그리고 그 뒤에 시간이 되었을 때 READY 상태로 만들어 Ready Queue에 삽입하는 방법을 사용한다
   3. Priority Scheduling
      * 현재의 핀토스는 round-robin 스케쥴링을 이용한다. 그래서 thread\_yield()나 thread\_unblock()이 호출되면, 현재 thread나 unblocked thread는 priority에 관련 없이 ready list의 끝에 insert된다.
      * 이를 수정해서 priority를 고려하는 complex 스케쥴러를 만드는 것이 목표이다. 만약 higher priority를 가지고 있는 새로운 thread가 들어온다면, 현재 thread는 즉시 그 thread에게 CPU를 양보해 주어야 한다.
      * Starvation을 해결하기 위해 priority aging 또한 구현한다.
   4. Advanced Scheduler (추가구현을 한 경우)
      * BSD 스케쥴러에서는 각 priority마다 자신의 ready queue를 가지는 Multi-Level Feedback Queue(MLFQ)를 사용한다. 특히, 이번 프로젝트에서는 64개의 priority마다 MLFQ를 각각 가지고 있다. 그리고 각 queue는 round-robin 정책을 따른다.
   5. **개발 내용**
2. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.
   * Wakeup time이 되지 않았을 때 ready state에 보내는 것이 아니라 blocked state로 갈 수 있도록 한다. 그러므로 이 blocked state를 관리하기 위한 queue를 생성한다.
   * 먼저, thread 구조체에 wake\_timer라는 변수를 설정을 하고, blocked state를 관리하기 위한 list\_sleeping queue를 만들어준다.
   * list\_sleeping에 해당 thread를 넣을 때, 일어나야 할 시간인 wake\_timer를 같이 저장해 준다. 그리고 시간이 경과하면 해당 thread를 깨우고 다시 ready queue에 넣어준다. timer\_interrupt()에서 매 tick마다 호출되면서 tick 값을 올려주는데, 여기서 list\_sleeping을 순회하며 일어나야 할 thread를 찾아서 ready queue에 넣는다. 만약 아직 시간이 되지 않았으면 그냥 넘어간다.
3. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.
   * 만약에 ready list에 running thread보다 높은 priority를 가진 thread가 들어온 경우에, 현재 실행중인 thread와 switch가 일어난다. 이를 구현하기 위해서는 ready list를 우선순위에 따라 push해야 하기 때문에, list\_push\_back() 대신에 list\_insert\_ordered() 함수를 사용한다.
   * Priority의 범위는 0부터 63까지이다. Default priority value는 31이고, 더 낮은 숫자는 더 낮은 우선순위를 의미한다.
4. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)
   * BSD Scheduler는 64개의 ready queue를 가지고, 초기값은 thread\_create()에서 31의 default 값으로 설정된다. 그 후 매 4tick마다 시스템에 있는 모든 thread의 priority가 계산된다. 계산 식은 다음과 같다.
   * priority = PRI\_MAX – (recent\_cpu / 4) – (nice \* 2)
   * recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1) \* recent\_cpu + nice
   * load\_avg = (59/60)\*load\_avg+(1/60)\*ready threads
     + recent\_cpu: 최근에 thread가 수행하면서 소모한 CPU Time의 근사이다. 최근에 사용한 소모한 CPU Time일수록 더 높은 가중치를 받게 된다. Thread가 새로 처음에 생성될 때는 0으로 초기화 되어 있고, 새로 생성된 thread가 아니라면 parent thread를 상속한다. Time interrupt가 호출될 때마다, RUNNING state에 있는 thread의 recent\_cpu 값은 1 커진다.
     + load\_avg: READY state와 RUNNING state의 thread의 개수인 ready\_threads로 계산된다.
     + nice: 각 thread는 priority와 관련된 nice value를 가지는데, 범위는 -20 ~ 20이다.
5. **추진 일정 및 개발 방법**
   1. **추진 일정**

* 10. 28 ~ 11. 01: 매뉴얼 분석 및 pintos 코드 이해
* 11. 02 ~ 11. 04: Alarm Clock
* 11. 05 ~ 11. 07: Priority Scheduling
* 11. 08 ~ 11. 10: Advanced Scheduler(BSD)
* 11. 11: 마무리 작업, 보고서 작성
  1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  1. Alarm Clock
  + devices/timer.c: timer\_sleep()에서 busy waiting 방식으로 돼있는 것을 바꿔준다. 일단 timer.c에서 list\_sleeping 이름의 queue를 만들어준다.
  + timer\_init()에서 list\_sleeping을 list\_init으로 초기화해준다.
  + timer\_interrupt()에서 list\_sleeping을 순회하면서 wakeup할 thread가 있는지 확인한다.
  + threads/thread.h에서 struct thread 구조체에 int64\_t wake\_timer 변수를 추가한다.

2) Priority Scheduling

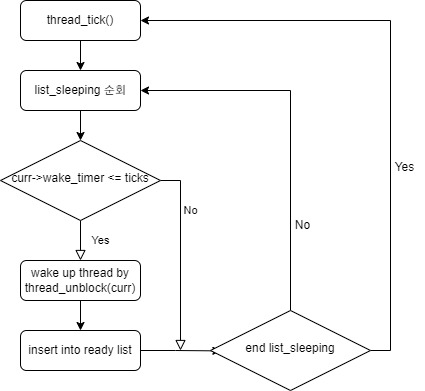
* + PPT 31쪽의 자료구조들을 먼저 추가해 준다.
  + threads/synch.c에서 sema\_up()에 새로운 thread가 들어올 수도 있으므로 list\_sort를 해주어야 한다.
  + threads/thread.c의 thread\_create(), thread\_yield(), thread\_unblock()에서 thread가 priority에 따라서 스케쥴링 되도록 일부 수정해준다.
  + threads/thread.c에서 list\_insert\_ordered를 호출할 때 있는 파라미터로써 넘겨줄 comp\_priority() 함수를 추가한다.
  + threads/thread.c에서 thread\_set\_priority()를 구현해서 current thread의 priority를 새로 업데이트한다. 그리고 thread\_get\_priority()를 구현해서 current thread의 priority를 반환한다.
  + threads/thread.c에서 thread\_aging()을 구현한다.
  + devices/timer.c에서 timer\_interrupt()의 내용을 thread\_prior\_aging 변수에 따라서 수정해준다.

3) Advanced Scheduler (BSD)

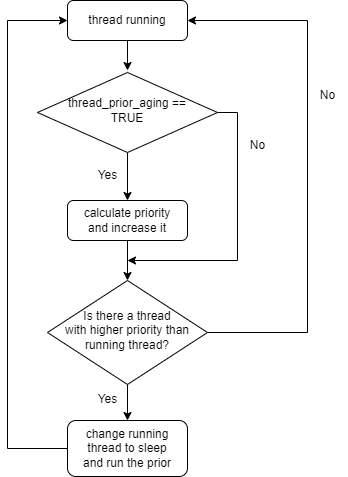
* + Fixed-point format 계산을 위해서 threads/thread.c에 계산 함수를 만든다. (정수-실수), (정수x실수), (실수+정수), (실수x실수), (실수/실수), (실수+실수), (실수-실수), (실수/정수)를 fixed-point format을 이용해 계산한다.
  + devices/timer.c에서 timer\_interrupt()의 내용을 thread\_mlfqs에 따라 수정한다.
  + threads/thread.c에서 nice 변수를 수정하기 위해 thread\_get\_nice(), thread\_set\_nice()를 구현한다. recent\_cpu 계산을 위해서 thread\_get\_recent\_cpu(), calc\_recent\_cpu()를 구현한다. 또한, load\_avg 계산을 위해서 thread\_get\_load\_avg(), calc\_load\_avg()를 구현한다.
  + threads/thread.c에서 nice와 recent\_cpu를 이용해서 priority를 계산하는 update\_priority()를 새로 구현한다.
  + thread/thread.c에서 thread\_aging()을 수정한다.

1. **연구 결과**
   1. **Flow Chart**
      1. Alarm Clock

timer\_sleep()에서 BLOCKED STATE인 list\_sleeping로 들어간다.



* + 1. Priority Aging



* 1. **제작 내용**

1. 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
2. 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명
3. Alarm Clock
   * devices/timer.c: timer\_sleep()에서 busy waiting 방식으로 돼있는 것을 바꿔준다. 일단 timer.c에서 list\_sleeping 이름의 queue를 만들어준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* + timer\_init()에서 list\_sleeping을 list\_init으로 초기화해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + timer\_interrupt()에서 list\_sleeping을 순회하면서 wakeup할 thread가 있는지 확인한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/thread.h에서 struct thread 구조체에 int64\_t wake\_timer 변수를 추가한다.



1. Priority Scheduling
   * PPT 31쪽의 자료구조들을 먼저 추가해 준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/synch.c에서 sema\_up()에 새로운 thread가 들어올 수도 있으므로 list\_sort를 해주어야 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/thread.c의 thread\_create(), thread\_yield(), thread\_unblock()에서 thread가 priority에 따라서 스케쥴링 되도록 일부 수정해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/thread.c에서 list\_insert\_ordered를 호출할 때 있는 파라미터로써 넘겨줄 comp\_priority() 함수를 추가한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/thread.c에서 thread\_set\_priority()를 구현해서 current thread의 priority를 새로 업데이트한다. 그리고 thread\_get\_priority()를 구현해서 current thread의 priority를 반환한다.

텍스트이(가) 표시된 사진

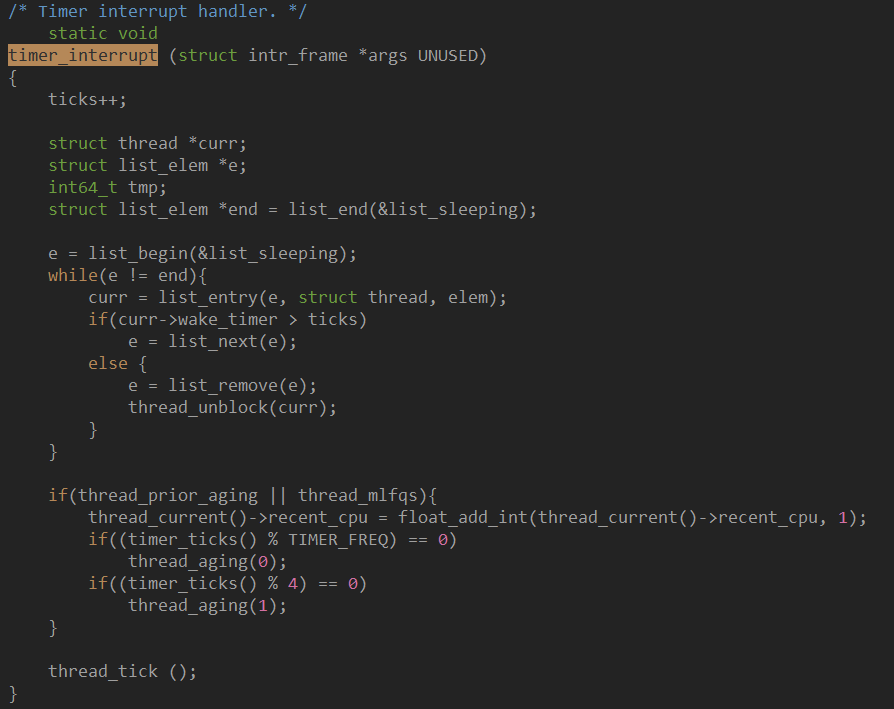
자동 생성된 설명

* + threads/thread.c에서 thread\_aging()을 구현한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + devices/timer.c에서 timer\_interrupt()의 내용을 thread\_prior\_aging 변수에 따라서 수정해준다.
  + devices/timer.c에서 timer\_interrupt()의 내용을 thread\_mlfqs에 따라 수정한다.



1. Advanced Scheduler (BSD)
   * Fixed-point format 계산을 위해서 threads/thread.c에 계산 함수를 만든다. (정수-실수), (정수x실수), (실수+정수), (실수x실수), (실수/실수), (실수+실수), (실수-실수), (실수/정수)를 fixed-point format을 이용해 계산한다.



텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/thread.c에서 nice 변수를 수정하기 위해 thread\_get\_nice(), thread\_set\_nice()를 구현한다. recent\_cpu 계산을 위해서 thread\_get\_recent\_cpu(), calc\_recent\_cpu()를 구현한다. 또한, load\_avg 계산을 위해서 thread\_get\_load\_avg(), calc\_load\_avg()를 구현한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* + threads/thread.c에서 nice와 recent\_cpu를 이용해서 priority를 계산하는 update\_priority()를 새로 구현한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + thread/thread.c에서 thread\_aging()을 수정한다.

****

* 1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석
  + priority\_lifo.c는 같은 priority를 갖는 여러 개의 thread를 생성을 한 다음에, 정상적으로 round-robin으로 돌아가는지를 확인한다. (모든 thread가 같은 priority이므로)
  + priority는 PRI\_DEFAULT + I + 1이다. 따라서 처음 main thread의 자식들은 실행되지 않는다. 그 후 main thread의 priority를 PRI\_DEFAULT 값으로 설정해 priority가 15인 thread부터 순서대로 실행된다. ready list에서 나오는 thread는 우선순위 기준으로 정렬되어 있기 때문에 그 이후에도 우선순위 기준으로 thread가 실행된다. 결과적으로 아무리 반복하여도 같은 output이 나오게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* make check 수행 결과를 캡처하여 첨부

텍스트이(가) 표시된 사진

자동 생성된 설명