**Pintos Project 3: Threads**

담당 교수 : 김영재 교수님

조 / 조원 : 개별 / 20201614 이민석

개발 기간 : 2024.11.08 ~ 11.24

1. **개발 목표**

* PintOS의 기존 scheduler인 Round-Robin 방식 및 thread의 synchronization 기능을 확장하고 개선하여, thread를 효율적으로 관리하고 scheduling 하는 것이 목표이다. 우선 busy-waiting 방식으로 구현된 것을 Sleep-Awake 방식으로 CPU자원이 낭비되지 않도록 해야 한다. 또한 thread의 Priority 기반 scheduling을 구현하며, 추가적으로 4.4BSD scheduler를 기반으로 한 MLFQ와 유사한 scheduler를 구현해야 한다. 이 과정에서, floating point 연산을 활용하여 정확한 priority 계산을 하게 된다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Alarm Clock
      * 기존 Busy-Waiting은 프로그램이 다른 작업을 수행하며 기다리는 대신, 특정 조건이 충족될 때까지 반복적으로 확인하면서 기다리는 것을 의미한다. 현재 ‘timer\_sleep()’함수는 현재 timer 시간과 ticks를 비교하여 아직 ticks에 도달하지 않았다면 다른 thread에게 CPU를 양보하는데, 이 과정에서 불필요한 Context switching이 많이 일어나게 된다.
      * 따라서 Sleep-Awake 방식을 채택하여, ready\_list에 thread가 깨어날 시각인 ticks에 도달한 경우에만 추가하도록 한다. 이는 sleep상태인 thread가 ticks에 도달하지 않았다면 ready\_list에 추가하지 않아, 아직 ticks에 도달하지 않은 thread가 깨어 CPU를 점유하는 경우가 발생하지 않도록 한다.
   3. Priority Scheduling
      * 기존 PintOS는 Round-Robin scheduling으로 구현되어 있다. 그러나 thread의 Priority에 따라 높은 thread가 우선적으로 CPU를 점유할 수 있도록 구현해야 한다. 이를 위해 priority에 따라 ready\_list에 thread를 추가해야 하며, 만약 실행 중인 thread의 priority보다 higher priority를 가진 thread가 ready list에 추가된다면, 현재 실행 중인 thread는 바로 CPU를 양보해야 한다. 이러한 preemptive한 기능도 구현해야 한다. 또한 starvation을 해결하기 위해 priority aging도 구현해야 한다.
   4. Advanced Scheduler (추가구현을 한 경우)
      * 2번에서 구현한 priority scheduler는 실행 순서를 오직 priority로만 비교하기에 priority가 낮은 thread는 CPU를 점유하기 어렵고, Average response time 또한 커질 가능성이 있다. 따라서 MLFQ scheduling 방식과 유사하게 Priority를 실시간으로 조절하여 advanced scheduler를 구현해야 한다. Pintos를 실행할 때, ‘-mlfqs’옵션을 넣어 advanced scheduler를 적용할 수 있다.
   5. **개발 내용**
2. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.
   * + Blocked 상태의 thread는 특정 조건이 만족되었을 때 Ready 상태로 전환된다. 이는 Blocked 상태의 thread를 관리하는 queue인 ‘sleep\_list’를 이용하여 관리할 수 있다. Thread 구조체에 깨어날 시간을 저장할 변수(wakeup\_tick)를 추가하고, 매 tick마다 ‘timer\_interrupt()’가 호출되어 ‘sleep\_list’의 각 thread가 깨어날 시간이 되었는지 확인한다. 깨어날 시간이 된 thread는 ‘sleep\_list’에서 제거되고, ‘ready\_list’에 삽입된다.
3. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.
   * + Priority Scheduling에 따르면 Ready list에 있는 thread 중 가장 높은 priority를 가진 thread가 항상 실행되어야 한다. 따라서 Running thread보다 높은 priority를 가진 thread가 ready list에 추가될 경우, 현재 실행중인 thread와 context switching이 일어나야 한다. 이를 위해 Ready list에 삽입된 thread를 priority에 따라 내림차순으로 정렬한다. 그리고 Preemption하게 Ready list의 맨 앞 thread(priority가 가장 큰 thread)와 running thread간의 priority를 비교한다. 이는 Running thread의 priority가 낮아질 때 또는 Ready list가 수정될 때마다 확인해야 한다.
4. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)
   * + 이번 프로젝트의 Advanced Scheduler는 MLFQ(Multi-Level Feedback Queue) Scheduler를 기반으로 하며 0~63까지의 값을 가지는 Priority를 바탕으로 thread의 우선순위를 고려한다. 또한 thread가 최근동안 얼마나 많은 CPU시간을 사용했는지 계산해야 한다. 이는 thread 구조체에 “recent\_cpu” 변수를 추가하여 추적할 수 있다. 최근 CPU 사용량이 많을수록 우선순위가 낮아진다. 마지막으로 thread의 ‘nice’ value를 바탕으로 다른 thread에게 CPU를 양보하려는 경향을 계산해야 한다. Nice value가 높을수록 priority는 낮아진다. 추가적으로 system의 부하 정도를 확인해야 한다. 이는 ‘load\_avg’ 변수를 통해 지속적으로 체크한다. Load average가 높을수록 recent\_cpu 값은 천천히 감소하게 된다.
     + 아래는 위 과정에서 필요한 식이다.

**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**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**
      * 11.7 ~ 11.8: 프로젝트 3에 대한 전반적 이해 및 Alarm clock 구현
      * 11.10 ~ 11.12: Priority scheduling 구현
      * 11.16 ~ 11.20: Advanced Scheduler 구현 및 테스트 진행
   2. **개발 방법**

* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명

1. Alarm Clock(Blocked 상태의 스레드를 어떻게 깨우고 관리할 것인가)
   * + “threads/thread.h”: thread 구조체에 일어날 시각인 ticks를 저장할 wakeup\_tick을 추가한다.
     + “threads/thread.c”: 아직 깨어날 시간에 도달하지 않은 thread들을 깨어날 시간이 이른 순서대로 저장하는 sleep\_list(extern)를 선언하고 “init\_thread()’에서 초기화한다.
     + “devices/timer.c”: “timer\_sleep()”함수에서 잠든 thread는 sleep\_list에 삽입하여 관리하도록 하고, sleep\_list에 깨어날 시간이 이른 thread가 앞 순서에 위치하도록 정렬해야 한다. 이를 위해 “cmp\_wakeup\_tick()”함수를 선언하여 “list\_insert\_ordered()”를 통해 추가한다. 또한 매 tick마다 호출되는 ‘timer\_interrupt()’함수를 이용하여 깨울 thread가 있는지 확인하는 로직을 내부에 구현한다. 만약 깨어날 시간이 된 thread는 sleep\_list에서 제거한 후, “thread\_unblock()”함수를 통해 ready\_list에 추가되게 된다.
2. Preemptive Priority scheduling
   * + “threads/thread.c”: “thread\_yield()”, “thread\_unblock()”함수에서 ready\_list에 thread를 삽입할 때 priority가 높은 thread부터 앞쪽에 위치하도록 정렬한다. 이 때 (1)과 유사한 방법으로 “cmp\_priority()”함수를 구현한 후, “list\_insert\_ordered()”를 사용하여 삽입할 수 있다. 또한 현재 실행중인 thread와 ready\_list에 있는 thread의 priority를 비교해야 한다. 계산 결과로 priority가 더 높은 thread가 CPU를 점유할 수 있도록 “priority\_preempt()”함수를 구현하고 실행중인 thread의 priority가 변하거나 ready\_list에 thread가 추가될 때 호출해야 한다. 추가적으로 낮은 priority를 가진 thread가 starvation에 빠지지 않도록, 매 tick마다 모든 thread의 priority를 1씩 높이는 “thread\_aging()”함수를 구현해야 한다.
     + “threads/synch.c”: lock, semaphore, condition variable을 기다리며 대기하고 있는 thread가 여러 개인 경우, priority가 가장 높은 thread가 먼저 깨어나야 한다. 따라서 lock이 걸린 대기 list인 waiters도 priority 기준으로 내림차순 정렬로 변경한다. 우선 “sema\_down()’함수에서 priority가 높은 thread가 앞 순서에 위치하도록 waiters에 thread를 추가해야 한다. 이는 ready\_list에 우선순위를 고려하여 추가한 방법과 유사하다. “cond\_wait()’함수의 ‘cond->waiters’를 정렬할 때는 ‘cmp\_sema\_priority()’를 따로 구현하여 사용해야 한다. 인자로 전달되는 elem의 thread에 바로 접근할 수 없기 때문이다. “sema\_up()”, “cond\_signal()”에서 thread를 깨우기 전 waiters목록을 다시 한번 정렬한다. “sema\_up()” 내부에선 “thread\_unblock()”함수가 호출되면서 ready\_list에 새 thread가 추가되기에, “priority\_preempt()”를 호출한다.
3. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)
   * + “threads/init.c”: pinots 실행 시, “-mlfqs”옵션을 넣어주면 advanced scheduler를 적용하여 실행할 수 있도록 “thread\_mlfqs” 값을 true로 바꿔준다. 따라서 “threads/thread.h”에 정의된 “thread\_mlfqs”를 extern 변수로 선언해야 한다.
     + “threads/thread.h”: thread 구조체 안에 nice value 값을 저장하는 “nice”, 최근 CPU 점유량을 저장하는 값인 “recent\_cpu”를 추가적으로 선언한다. 또한 시스템에 하나의 값으로 존재하는 “load\_avg”변수를 선언한다. 추가한 변수들은 적절히 초기화도 해야 한다. 이 때 priority, nice는 정수 값을, recent\_cpu, load\_avg는 실수 값을 가진다. 따라서 우리가 직접 부동 소수점 연산을 구현해야 한다. 다음의 식을 연산할 수 있는 함수들을 구현한다.
     + 추가적으로텍스트, 스크린샷, 폰트이(가) 표시된 사진

       자동 생성된 설명 특정 thread의 priority, recent\_cpu를 계산하는 함수, 시스템의 load\_avg를 계산하는 함수를 구현해야 한다. 매 1tick마다 Running thread의 recent\_cpu 값을 1씩 늘려야 하고, 4tick마다 모든 thread의 priority를 재계산 해야 하며, 1초마다 모든 thread의 recent\_cpu와 load\_avg값을 재계산 해야 한다. 이러한 것들을 계산해주는 함수들도 구현해야 한다.
     + “devices/timer.c”: 위에 내용을 바탕으로 해당하는 시간 주기마다 “timer\_interrupt()”함수에서 “thread\_mlfqs == true”일때만 priority, recent\_cpu, load\_avg 값의 조정되도록 한다.
4. **연구 결과**
   1. **Flow Chart**
5. 텍스트, 도표, 스크린샷이(가) 표시된 사진

   자동 생성된 설명Alarm Clock(Blocked 상태의 스레드를 어떻게 깨우고 관리할 것인가)
6. **텍스트, 도표, 라인, 폰트이(가) 표시된 사진

   자동 생성된 설명**Preemptive Priority scheduling
   1. **제작 내용**

* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  1. Alarm Clock(Blocked 상태의 스레드를 어떻게 깨우고 관리할 것인가)
     + 텍스트, 스크린샷, 폰트이(가) 표시된 사진

       자동 생성된 설명“threads/thread.h”의 thread 구조체에 일어날 시각인 ticks를 저장할 ”int64\_t wakeup\_tick”을 추가한다.

“threads/thread.c”에 아직 깨어날 시간에 도달하지 않은 thread들을 깨어날 시간이 이른 순서대로 저장하는 sleep\_list(extern)를 선언한다.

* + - 텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

      자동 생성된 설명“threads/thread.c”의 “init\_thread()’에서 sleep\_list를 초기화한다.
    - “devices/timer.c”에서 “timer\_sleep()”함수 내부이다. 기존 busy-waiting으로 구현된 것을 모두 지우고, thread의 wakeup\_tick을 활용하여 block 상태의 thread는 sleep\_list에 삽입하였다. 이 때 “cmp\_wakeup\_tick()” 비교함수를 통해 wakeup\_tick 순서대로 sleep\_list를 정렬하여 관리한다. 또한 처음 부분에 interrupt를 비활성화 하여 timing과 관련된 race condition을 방지하고 마지막 부분에 이를 원래 상태로 복원한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + - “devices/timer.c”에서 “timer\_interrupt()”함수 내부이다. “timer\_sleep()”과 마찬가지로 ‘intr\_disable()’를 호출하여 interrupt를 비활성화하고 마지막에 복원한다. 반복문을 통해 sleep\_list를 순회하며 깨어날 시간이 된 thread들을 sleep\_list에서 제거한다. “thread\_unblock(thread\_ptr)”를 호출하여 해당 thread를 Ready\_list에 추가한다. 또한 wakeup\_tick에 따라 오름차순으로 thread를 sleep\_list에 저장했기에, 아직 깨어날 시간이 되지 않은 thread가 있다면 그 뒤의 thread는 확인할 필요가 없다.
    - “devices/timer.c”에서 “cmp\_wakeup\_tick()”함수이다. Wakeup\_tick값을 기준으로 thread들을 정렬할 수 있도록 하며, “list\_insert\_ordered()”에 사용된다. wakeup\_tick 값이텍스트, 스크린샷, 폰트이(가) 표시된 사진

      자동 생성된 설명 작은 thread가 list 앞쪽에 오도록 정렬하게끔 한다.
  1. Preemptive Priority scheduling

**텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명**

* + - “threads/thread.c”에 구현된 “thread\_yield()”, “thread\_unblock()”의 내부이다. ready\_list에 thread를 삽입할 때, priority가 높은 thread가 list의 앞쪽에 위치하도록 정렬한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* + - “threads/thread.c”에 구현된 “cmp\_priority()”이다. ready\_list에 thread가 추가될 때, **텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

      자동 생성된 설명**priority가 높은 순서대로 정렬될 수 있도록 도와주는 함수이다.
    - “threads/thread.c”에 구현된 “priority\_preempt()”함수이다. ready\_list의 thread중 priority가 가장 높은 thread는 가장 앞쪽에 위치하는데, 이 thread가 Running 상태의 thread보다 priority가 높을 경우 양보해야 한다. 이를 구현한 함수이다. 이는 ‘thread\_yield()’를 호출하여 해결한다.

**폰트, 텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

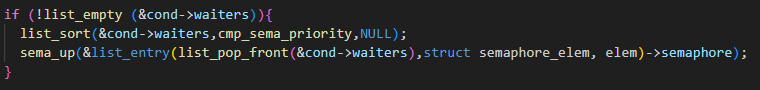
* + - “threads/thread.c”의 “thread\_create()”에서 thread가 unblocked 되면, 이 때 현재 실행 중인 thread와 ready\_list에 있는 thread와 priority를 비교해야 한다. 또한 “thread\_set\_priority()”에서 실행중인 thread의 priority를 변경하므로, “priority\_preempt()”를 호출해야 한다.’

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - **텍스트, 스크린샷, 폰트이(가) 표시된 사진

      자동 생성된 설명**“threads/synch.c”의 “sema\_down()”과 “cond\_wait()”의 일부이다. waiters에 새 thread를 삽입할 때, priority가 높은 thread가 앞쪽에 위치하도록 정렬해야 한다. 이는 새로 구현하는 “cmp\_sema\_priority()”함수 또는 기존 “cmp\_priority()”함수와 “list\_insert\_ordered()”를 이용하여 적용할 수 있다.
    - “threads/synch.c”의 “sema\_up()”과 “cond\_signal()”의 일부이다. waiters에서 thread를 깨우기 전, waiters 목록을 새로 정렬하며, unblock()함수가 호출될 때, 적절히 “priority\_preempt()”를 호출해야 한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - **텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

      자동 생성된 설명**“threads/synch.c”에 구현된 “cmp\_sema\_priority()”이다. 두 semaphore 안의 waiters 중 제일 높은 priority를 비교하여 cond->waiters를 정렬할 수 있다.
    - 명세서에 적혀있는 project 3의 priority scheduling과 관련된 변수들 및 설정을 구현한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* + - “threads/thread.c”의 “thread\_aging()”함수이다. 모든 thread의 priority를 1씩 늘려, starvation 문제를 방지하는 역할을 하는 함수이다. 이는 “all\_list”를 순회하며 모든 thread를 최신화 하는데, priority값이 MIN~MAX 범위 안의 값이 될 수 있도록 보정 또한 한다.
  1. **텍스트, 스크린샷이(가) 표시된 사진

     자동 생성된 설명텍스트, 스크린샷이(가) 표시된 사진

     자동 생성된 설명**Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)
     + 위에 설명한 것처럼, Advanced Scheduler의 모든 계산에서 부동 소수점 연산이 필요하므로 FP 값을 매크로로 설정하고, 연산에 필요한 함수들을 모두 “threads/thread.c”에 구현한다.

**텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

****

****

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* + - **텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

      자동 생성된 설명**“threads/thread.h”의 thread 구조체에 nice, recent\_cpu 변수를 추가한다. “threads/thread.c”엔 load\_avg를 선언한다. 이에 따른 초기화는 “thread\_init()”, “thread\_start()”에서 진행한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - **텍스트, 스크린샷, 폰트이(가) 표시된 사진

      자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

      자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

      자동 생성된 설명**“threads/thread.c”에 구현된 각각 load\_avg, priority, recent\_cpu를 계산하는 식이다. 위의 내용에 작성된 식을 바탕으로 계산하며, idle 상태의 thread는 priority가 고정이므로 제외하며, 계산 결과의 소수부분은 버린다.
    - **텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

      자동 생성된 설명**
    - 매 1tick마다 Running thread의 recent\_cpu 값이 1 증가하므로 이를 계산하는 함수, 4 tick마다 모든 thread의 priority를 다시 계산해야 하므로 이를 계산하는 함수, 1초 (100tick) 마다 모든 thread의 recent\_cpu 값과 load\_avg 값을 다시 계산해야 한다. 이를 “threads/thread.c”에 구현해야 하며, “devices/timer.c”의 “timer\_interrupt()”에 구현해야 한다.
    - “threads/thread.c”에 선언만 되어있는 “thread\_set\_nice()“, “thread\_get\_nice ()“, “thread\_get\_load\_avg ()“, “thread\_get\_recent\_cpu ()“를 구현해야 한다. 현재 thread의 nice value를 새 값으로 초기화 하거나, 반환하는 함수, 현재 load\_avg \* 100 값을 반환하는 함수, thread의 recent\_cpu \* 100 값을 반환하는 함수가 되도록 구현해야 한다. 이는 강의자료를 바탕으로 구현하였다.
  1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* 우선 “priority-lifo.c”를 간단히 분석하자. 16 threads가 서로 다른 priority 값을 가지고 실행되며, Priority에 따라 LIFO 순서대로 실행되는지 확인하는 코드이다. 이 때, 각 thread는 “PRI\_DEFAULT + 1 + i”로 priority가 설정되며, i는 thread이름(id)값과 같다. 그 후, main thread의 priority를 기본값인 PRI\_DEFAULT로 설정하여 다른 16개의 thread가 먼저 실행되도록 한다.
* 따라서 ready\_list에서 thread는 priority 내림차순 순서로 정렬되어 있기 때문에, 빠져나오는 순서도 이와 같다. 각 반복에서 priority가 높은 thread가 실행을 마칠 때까지, 이하의 priority를 가진 thread는 실행되지 않는다. 따라서 LIFO가 보장됨을 알 수 있다.
* make check 수행 결과를 캡처하여 첨부(평가항목만 test하도록 수정하였음.)

텍스트, 스크린샷, 책, 메뉴이(가) 표시된 사진

자동 생성된 설명