Pintos 프로젝트 1. Pintos Thread

(설계 프로젝트 수행 결과)

과목명 : [CSE4070-01] 운영체제

담당교수 : 서강대학교 컴퓨터공학과 박성용

조원 : 28조 김태훈

개발기간 : 2016. 11. 21. -2016. 11. 25.

최 종 보 고 서

프로젝트 제목: Pintos 프로젝트 1. Pintos Thread
제출일: 2016. 11. 25.
참여조원: 28조 김태훈

I. 개발 목표

Pintos OS에서 thread 간의 scheduling을 담당하는 다양한 scheduler의 scheduling algorithm을 이해하고 이를 통해 기본적인 priority scheduler와 multi-level feedback queue를 이용한 BSD scheduler를 구현한다.

II. 개발 범위 및 내용

가. 개발 범위

현재의 Pintos OS에는 priority를 무시하고 time quantum에만 의존하여 thread를 돌리는 first in first out round robin scheduler가 구현되어 있다. 이를 변경하여 priority에 따라 thread의 scheduling 순서를 바꾸는 preemptive priority scheduler를 구현한다. 하지만 이 scheduler 에는 starvation의 문제가 있다. 이를 해결하기 위해 thread aging option과 multi-level feedback queue option을 추가한다.

나. 개발 내용

1. 우선 현재 busy waiting 방식으로 구현되어 있는 alarm clock을 일정 tick이 지나면 timer interrupt가 sleep 상태에 있는 thread를 깨우는 방식으로 변경한다. 이를 위해 sleep 상태의 thread를 저장하는 새로운 queue와 이 thread를 깨우는 wake_up 함수를 구현한다.

2. 현재 구현되어 있는 round robin scheduler를 priority scheduler로 개선한다. first in first out ready queue를 priority first ordered ready queue로 변경하며, 이를 유지하기 위해 thread creation, semaphore up 등 priority에 따른 thread yield가 필요한 부분 역시 변경한다.

3. Advanced BSD Scheduler를 구현한다. Pintos의 priority는 PRI_MIN(0)부터 PRI_MAX(63)까지 총 64 단계로 나뉘어져 있다. 이를 토대로 64 level의 ready queue를 가지는 multi-level feedback queue를 구현하고 -mlfqs option을 사용하

면 BSD Scheduler를 사용할 수 있도록 한다. load average, recent cpu, priority의 계산 방식은 pintos manual을 참고 하도록 한다. 단, pintos의 kernel은 floating point 연산을 지원하지 않으므로 fixed point 연산을 위한 macro header인 fixed_point.h를 추가한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

11.21-22 : Pintos manual, thread 관련 source code를 분석하고 이를 토대로 alarm-clock과 basic priority scheduler를 구현

11.23-24 : BSD scheduler 구현 및 mlfqs test case pass

11.25 : Project 마무리 및 보고서 작성

나. 개발 방법

Pintos manual과 강의 자료, Operating Systems 교재를 참고하여 관련 사항을 학습한다. alarm clock 구현 시에는 busy waiting이 발생시킬 수 있는 문제를 이해하고 이를 바탕으로 thread wake up function을 어디에 추가해야 할지 파악한다. 프로젝트 목표가 제대로 완수 되었는지 확인하기 위해서 Pintos OS 내부 test tool을 활용한다.

thread.c source code를 자세히 읽어보면서 priority scheduler 구현 과정에서 thread yield를 추가할 부분들을 파악한다. 함부로 thread yield를 추가할 경우 scheduler가 정상적으로 작동하지 않을 수도 있다.

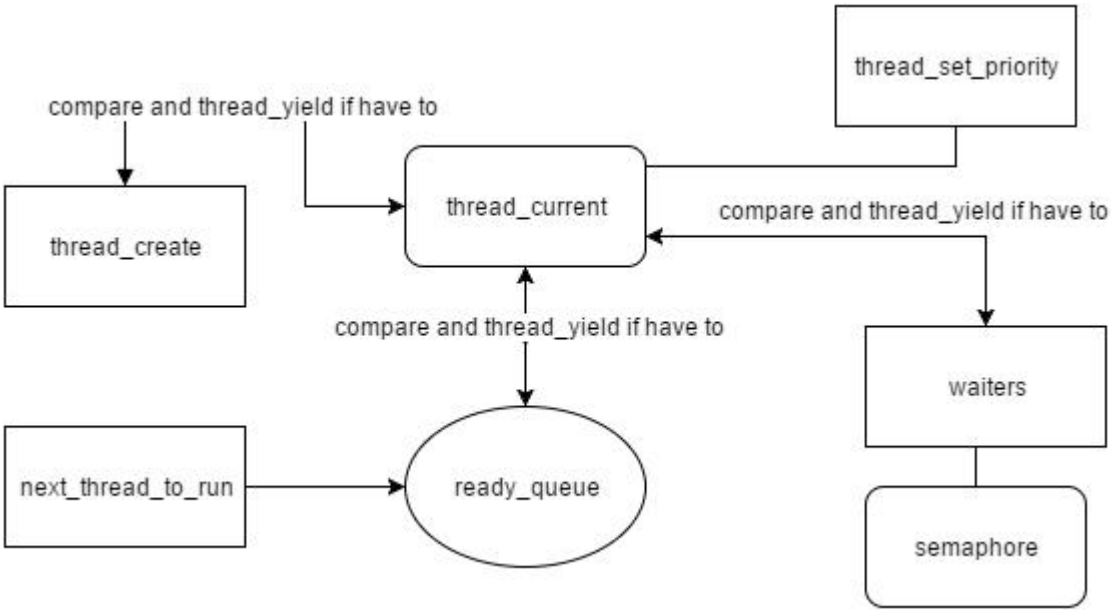
BSD scheduler 구현을 위한 multi level queue의 형태를 고려해본다. 실제로 64개의 ready queue로 나눌지, 아니면 하나의 queue를 사용하면서 64개의 level을 추상적으로 구현할지 분석한 뒤 BSD scheduler 구현을 시작한다. 전자의 경우 ready queue 내부에서 thread의 이동이 힘들어 질 수 있다. 후자의 경우를 선택하면 별도의 ready queue를 추가하고 관련 함수를 변경하지 않고도 BSD scheduler를 구현할 수 있다. scheduler의 내구성을 위해 후자를 선택하는 편이 나을 듯하다.

다. 연구원 역할 분담

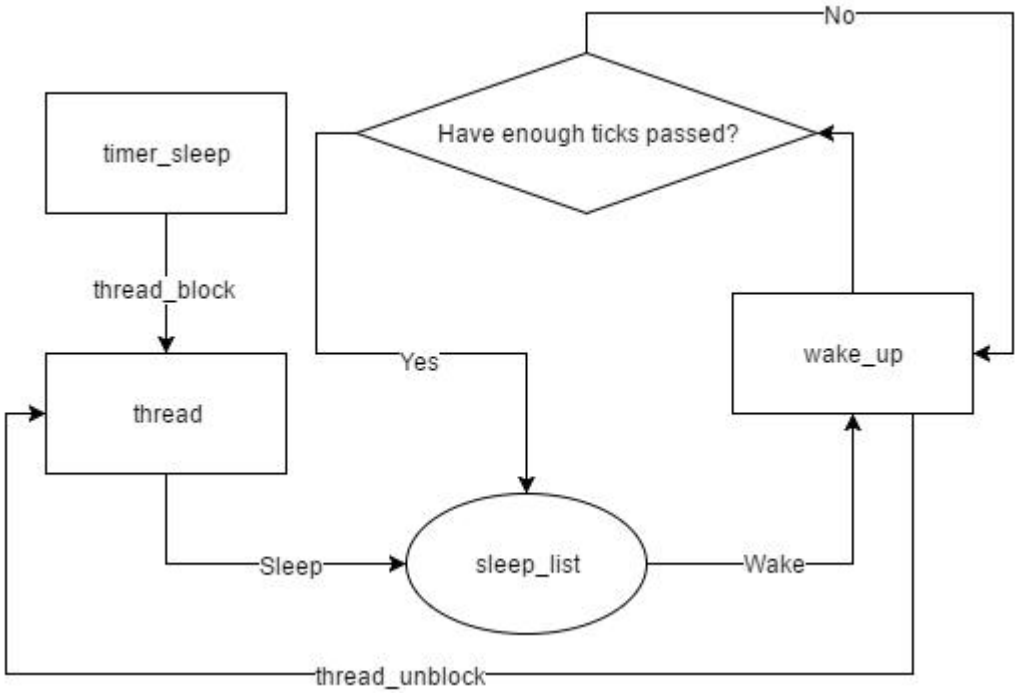
혼자서 진행하는 프로젝트이므로 충분한 시간을 두고 철저한 계획 하에 프로젝트를 진행한다. 일정 부분 이상 진행되지 않으면 중간 결과를 확인 할 수 없으므로 구현 사항에 우선 순위를 정하여 그 순서대로 프로젝트를 진행 한다.

Ⅳ. 연구 결과

1. **합성 내용:** 전체적인 흐름도는 다음과 같다.



Alarm Clock의 흐름도는 다음과 같다.



2. 제작 내용:

(1) Alarm Clock : timer_sleep에서는 busy waiting 대신 thread block을 사용하였다. 이후 thread에 wake up 해야 할 tick을 저장한 채로 이를 sleep_list로 추가하였다. wake_up 함수는 timer interrupt가 발생할 때마다 sleep_list를 확인하여 wake up 시켜줘야 할 thread가 있다면 이를 unblock 하고 ready queue에 추가한다.

(2) Priority Scheduling:

Priority Scheduling을 위해서 총 5개의 함수를 수정하였다. 먼저 sema_up 의 waiters list를 first in first out queue에서 high priority first out list로 변경하였다. 이를 위해 list_push_back을 list_insert_ordered로 변경하였다. 이후 sema_down에서 waiters list로부터 나온 thread의 priority가 현재 thread의 priority보다 높을 경우 thread_yield를 수행하였다.

thread_unblock 에서도 unblock된 thread를 ready queue에 다시 넣을 때 list_push_back이 아닌 list_insert_ordered를 사용하였다. thread_create에서 새로 생성된 thread의 priority가 현재 thread의 priority보다 높을 경우 thread_yield를 수행하였다. 마지막으로 thread_set_priority에 의해 새로 set된 현재 thread의 priority가 ready queue의 맨 앞에 있는 thread의 priority 보다 낮을 경우 thread_yield를 수행하였다. 이를 통해 Priority Scheduler를 구현했으나 starvation problem이 발생할 수 있는 한계는 남아있었다. 이를 위해 aging option을 추가하였으며, 근본적인 해결은 BSD Scheduler를 구현하여 해결하였다.

(3) BSD Scheduler: BSD Scheduler의 구현에는 별도의 ready queue를 만들지 않고 기존의 ready queue를 그대로 활용하여 추상적인 multi level feedback queue를 구현하였다. 이를 위한 새로운 priority 계산법은 다음과 같다.

priority = PRI_MAX - (recent_cpu/4) - (nice * 2)
recent_cpu = load_avg/(load_avg + 1) * recent_cpu + nice
load_avg = (59/60) * load_avq + (1/60) * ready_threads

단 floating point system이 없는 pintos kernel에서는 floating point arithmetic이 불가능 하다. 이를 위해 fixed point arithmetic을 한 새로운 macro 함수를 추가하여 이를 사용하였다.

```
#define FRAC_BASE 14
#define F (1 << FRAC_BASE)

#define int_to_fix(n) ((n) * F)
#define fix_to_int_zero(x) ((x) / F)
#define fix_to_int_nearest(x) (((x) >= 0 ? ((x) + (F/2)) : ((x) - (F/2)))/F)
#define fix_add(x,y) ((x) + (y))
#define fix_sub(x,y) ((x) - (y))
#define fix_mul(x,y) ((((int64_t)(x)) * (y)) / F)
#define fix_div(x,y) ((((int64_t)(x)) * F) / (y))
```

이를 이용한 BSD Scheduler의 priority calculation은 다음과 같은 방식으로 이루어졌으며, 이를 threads/thread_tick()에 추가하였다.

```
#ifndef USERPROG
  if(thread_mlfqs==true)
  {
      old_level=intr_disable();
      if(t!=idle_thread)
          t->recent_cpu+=int_to_fix(1);
      if(timer_ticks() % TIMER_FREQ==0)
      {
          mlfqs_load_avg_calculation();
          for(e=list_begin(&all_list);e!=list_end(&all_list);e=e->next)
          {
              cur=list_entry(e,struct thread,allelem);
              if(cur!=idle_thread)
                  mlfqs_recent_cpu_calculation(cur);
          }
      }
      if(timer_ticks() % 4==0)
      {
          for(e=list_begin(&all_list);e!=list_end(&all_list);e=e->next)
          {
              cur=list_entry(e,struct thread,allelem);
              if(cur!=idle_thread)
                  mlfqs_priority_calculation(cur);
          }
      }
      intr_set_level(old_level);
  }
#endif
```

-thread_tick() 함수에 BSD Scheduler 관련 내용을 추가

```
507 void
508 mlfqs_recent_cpu_calculation(struct thread* t)
509 {
510     int64_t temp;
511     int64_t recent_cpu;
512 
513     temp=fix_div(load_average*2 , load_average*2 + int_to_fix(1));
514     recent_cpu=fix_mul(temp,t->recent_cpu) + int_to_fix(t->nice);
515     t->recent_cpu=recent_cpu;
516 }
517 void
518 mlfqs_load_avg_calculation(void)
519 {
520     int64_t temp;
521     int ready_threads;
522     int64_t load_avg;
523 
524     temp=int_to_fix(59) / 60;
525 
526     ready_threads=list_size(&ready_list);
527     if(thread_current()!=idle_thread)
528         ready_threads++;
529     load_avg=fix_mul(temp,load_average) + (int_to_fix(1)/60) * ready_threads;
530     load_average=load_avg;
531 }
532 void
533 mlfqs_priority_calculation(struct thread* t)
534 {
535     int priority;
536 
537     priority=PRI_MAX-fix_to_int_nearest(t->recent_cpu/4)-(t->nice)*2;
538     if(priority < PRI_MIN)
539         t->priority=PRI_MIN;
540     if(priority > PRI_MAX)
541         t->priority=PRI_MAX;
542     else
543         t->priority=priority;
544 }
```

- BSD Scheduler priority calculation functions

(4) trial and error: fixed point arithmetic을 이용한 사칙연산 과정에서 multiplication을 수행할 시 overflow가 빈번하게 발생하였다. 이를 해결하기 위해 가능하면 division을 먼저 수행하도록 하였으며 coefficient를 먼저 계산한 뒤 이를 각 variable에 multiply 하는 등의 과정을 통해 overflow의 발생을 막았다.

3. 시험 및 평가 내용:

Project 결과에 대한 평가는 make check의 pass/fail을 기준으로 진행하였다.
평가는 cspro1.sogang.ac.kr에서 진행하였으며 simulator로 qemu를 사용하였다.

그 결과는 다음과 같다.

총 21개의 test를 모두 pass 하였다.

(1) 다음 6개의 test를 통해 생산성을 test하였다.

```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/mlfqs-load-1
```

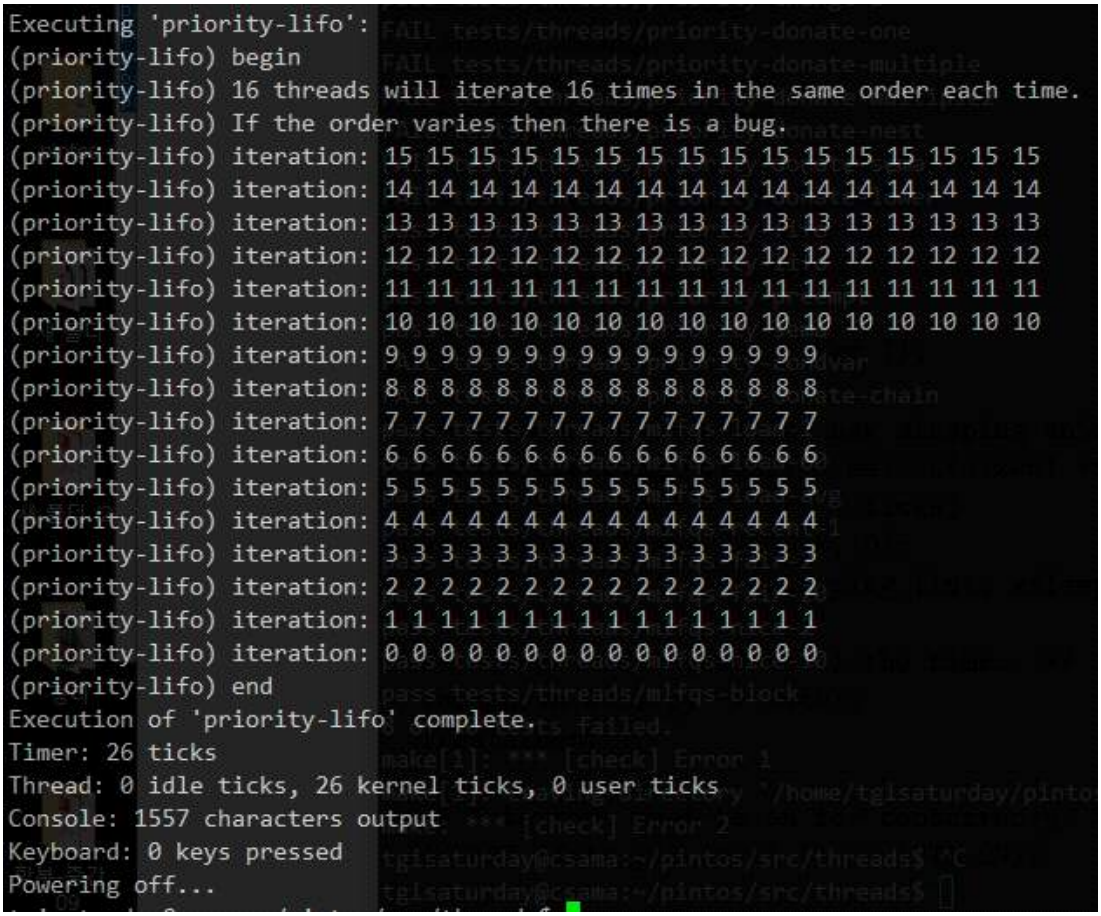
alarm clock과 priority scheduling을 문제없이 수행하여 의도한 결과를 출력하여 생산성을 만족하였다.

(2) 다음 15개의 test를 통해 내구성을 test 하였다.

```
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-aging
pass tests/threads/priority-change
pass tests/threads/priority-change-2
pass tests/threads/mlfqs-load-60
pass tests/threads/mlfqs-load-avg
pass tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
pass tests/threads/mlfqs-nice-2
pass tests/threads/mlfqs-nice-10
pass tests/threads/mlfqs-block
```

각종 예외 상황을 적절하게 대처하여 Pintos OS의 비정상적인 종료를 방지하였으며 mlfqs 항목의 경우 priority 계산을 위한 관련 수치들이 오차 범위 이내엣서 정상적으로 계산되는지 확인하였다. 이로서 내구성을 만족하였다.

(3)priority-lifo test case 결과 분석



priority-lifo test case는 priority-fifo test case와는 달리 나중에 만들어진 thread의 priority가 더 높도록 설계되어 있다. 따라서 이 test를 수행하게 되면 나중에 만들어진 thread부터 순서대로 종료되어야 하며, priority가 높은 thread가 종료될 때 까지 다른 thread가 run 되면 안된다. 위의 결과는 이를 잘 보여주고 있다.

(4) 보건 및 안정성 test

OS가 마주하는 예외 사항은 다양하다 . 이에 대해 적절하게 대처하지 못하면 사용자의 불편을 초래하게 된다. scheduler가 마주하게 되는 가장 큰 문제는 thread의 starvation이다. Priority가 낮은 thread는 ready queue에 지속적으로 priority가 높은 thread가 들어오게 되면 running state에 계속 도달하지 못하게 된다. 이를 해결하기 위해 aging과 mlfqs BSD Scheduler를 구현하였다. 또한 make check를 반복 수행해도 mlfqs 관련 항목이 timeout 480 이내에 모두 완료되고 오차 범위 내의 결과 값을 출력하였다. 이를 통해 안정성을 확인하였다.

V. 기타

1. **연구 조원 기여도**: 김태훈 (100%)

2. 해당 프로젝트를 진행한 cspro1.sogang.ac.kr은 서버의 낙후화로 다양한 문제점들이 발생하였다. bochs simulator를 사용할 수 없어 qemu simulator만을 사용하여야 했으며, git이 설치되어 있지 않아 source code version control에 어려움이 있었다. 하지만 느린 서버 환경이 다양한 synchronization 문제를 발생시켜 이를 해결하며 Pintos OS 자체의 성능을 향상하는데 도움이 되었다. 특히 mlfqs 관련 항목의 timeout은 480으로 BSD Scheduler의 계산 과정 최적화가 필요했다.