**Pintos Project 3: Threads**

담당 교수 : 김영재 교수님

학번 / 이름 : 20171663 이도훈

개발 기간 : 10/27 – 11/6

1. **개발 목표**

프로젝트 2에서 구현한 것에 이어서 thread와 관련된 alarm clock, priority scheduling, advanced scheduler를 추가 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Alarm Clock

Thread가 특정 시간 동안 작동하지 않은 경우 sleep 시킨다.

* 1. Priority Scheduling

기존 thread에서는 round-robin 방식을 사용하여 scheduling 하다보니 우선순위에 관계없이 current thread 등은 ready list 마지막에 삽입 되었다. Priority scheduling을 통해 priority에 맞게 삽입되게 한다.

이러한 priority scheduling의 starvation 문제를 해결하기 위해 aging을 사용한다. 시간이 지나면 priority가 상승하게 된다.

* 1. Advanced Scheduler (추가구현을 한 경우)

BSD scheduler를 구현한다. MLFQ를 사용하여 각 priority마다 queue를 가지게 한다. 함수가 호출 되면 가장 높은 priority를 가진 queue에서 thread가 선택된다. 이때 round-robin 방식을 사용한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

Blocked 상태의 thread를 관리하는 list를 새로 만들어서 blocked된 thread를 모두 저장한다. 그 이후 매 tick마다 thread를 검사하여 wake up time이 된 thread를 list에서 제거하고 상태를 업데이트하고 ready queue에 넣어준다.

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

높은 priority를 가진 thread가 들어오면 그 thread가 먼저 실행되어야 한다. 따라서 current thread를 ready 상태로 바꾸고 ready list에 priority 순으로 정렬되도록 running thread를 넣고 높은 priority 순으로 실행한다.

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

Nice와 recent\_cpu, 그리고 load\_avg가 필요하다.

Thread가 초기에 생성되면 nice와 recent\_cpu 값이 0으로 초기화된다. Child thread의 경우에는 parent thread의 값을 받는다. 이때 nice는 -20~20의 값을 갖게 된다. 양의 nice 값은 priority 값을 감소시키므로 nice 값이 변경 되면 thread의 priority를 다시 계산해야한다.

Recent\_cpu는 thread의 CPU time을 추정한다. Timer interrupt가 일어나면 running thread의 recent\_cpu 값이 1씩 증가하고 매초마다 모든 running, ready, blocked thread의 recent\_cpu가 다시 *recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1 ) \* recent\_cpu + nice* 의 식을 통해 다시 계산된다.

Load\_avg는 전역변수로 ready thread의 평균을 측정한다. 처음 시스템이 실행되면 0으로 초기화 되어있다가 매초마다

*load\_avg* = (59/60) \* *load\_avg* + (1/60) \* *ready\_threads* 의 식을 통해 다시 계산된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

10/28 – 11/1 alarm clock 구현

11/1 – 11/3 priority scheduling 구현

11/3 – 11/6 advanced scheduling 구현

11/6 – 11/6 보고서 작성

* 1. **개발 방법**

Timer.c에서 alarm clock을 위해 timer\_sleep 함수 코드를 수정해서 thread의 wakeuptime을 계산하고 blocked thread를 관리한다. Timer\_interrupt에서 wakeup시간이 된 thread를 깨워준다. Scheduling을 위해 timer\_interrupt에서 priority aging과 multi level feedback queue에 대한 처리를 해준다. Tick이 증가하면 thread의 recent\_cpu 값 역시 1씩 증가한다. TIMER\_FREQ만큼 tick이 흘렀으면 모든 thread의 recent\_cpu와 nice값을 update한다. Tick이 4만큼 지나면 모든 process의 priority를 update한다.

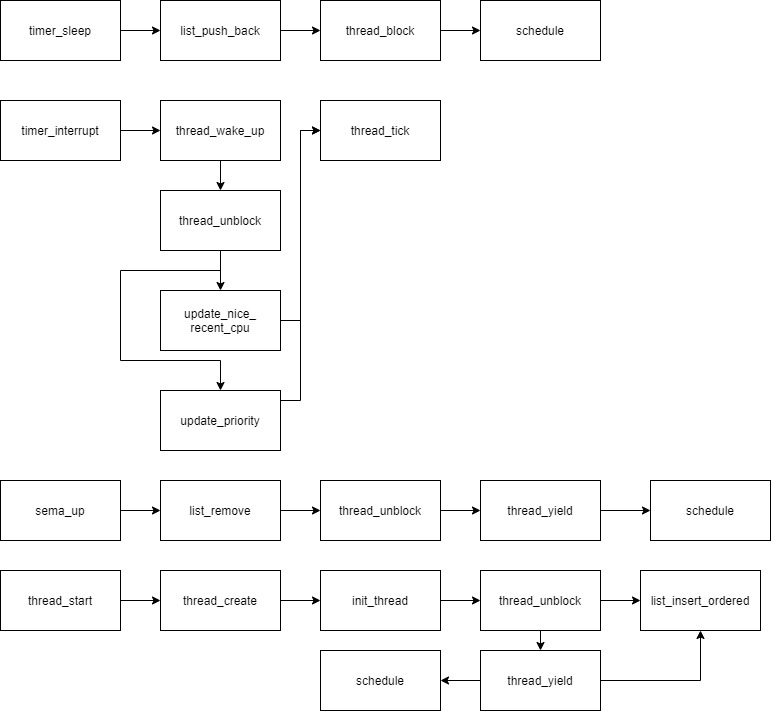
강의자료에서 말한 추가해야할 내용은 thread.h, thread.c, init.c에 추가해준다.

Thread.h에서 기존의 thread 구조체에 alarm clock을 위한 wakeuptime과 scheduling을 위한 recent\_cpu, nice 변수를 추가해준다.

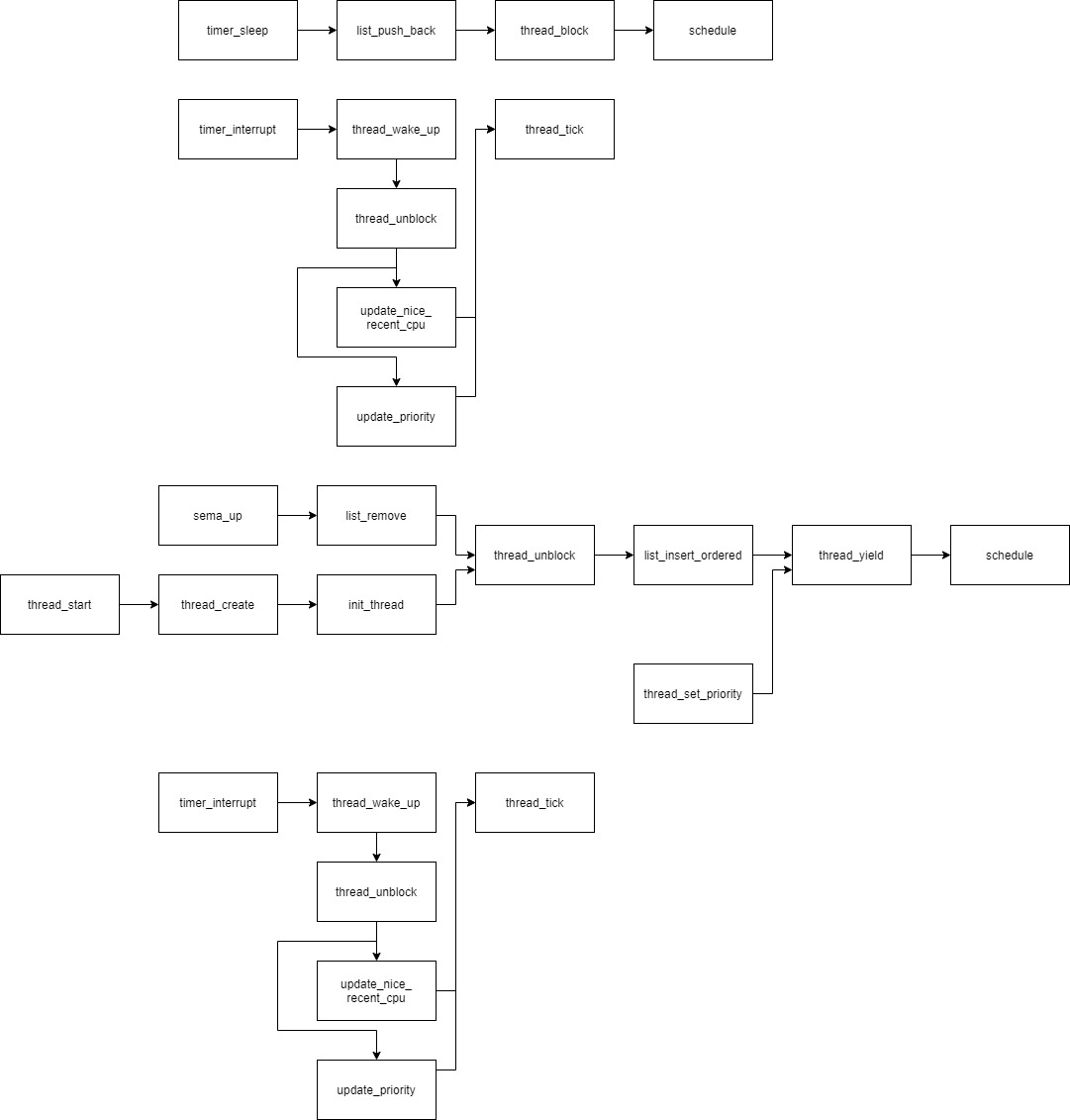
Thread.c에서 priority 순서대로 list를 추가할 수 있도록 thread\_yield와 thread\_unblock함수를 수정한다. Thread\_create에서는 current thread와 새로운 thread의 priority를 비교해서 새로운 thread의 priority가 더 클 경우 rescheduling하도록 코드를 변경한다. 또한 thread\_set\_priority를 통해 thread의 priority를 변경할 수 있도록 한다. 추가적으로 recent\_cpu, nice 값을 계산할 수 있도록 함수를 짠다.

Synch.c에서 sema\_up을 list에서 priority 가장 큰 것을 찾아서 sema\_up 작업을 하고 rescheduling하도록 코드를 변경한다.

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



* + - * Priority scheduling

****

* 1. **제작 내용**

1. Alarm Clock
2. threads/thread.h

텍스트이(가) 표시된 사진

자동 생성된 설명

thread 구조체에 thread의 wake up time을 저장할 wakeuptime 변수를 새로 생성한다.

1. threads/thread.c

텍스트이(가) 표시된 사진

자동 생성된 설명

Thread 구조체에 새로 추가한 변수 wakeuptime을 0으로 초기화한다.

1. devices/timer.c

텍스트이(가) 표시된 사진

자동 생성된 설명

새로 선언한 sleep\_list를 초기화한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Current thread의 wakeuptime을 ticks를 이용한 연산을 통해 구하고 sleep\_list에 추가한다. 그 이후 thread\_block을 통해 thread를 blocked 시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

timer\_interrupt 함수는 ticks을 증가시키는 함수이다. 따라서 thread\_wake\_up함수를 호출하여 ticks이 변할때마다 wakeuptime이 된 thread를 찾아서 깨워준다.

Ticks를 증가시키는 함수로 ticks가 바뀔때마다 해당하는 thread를 찾아서 thread\_wake\_up을 통해 깨운다.

텍스트이(가) 표시된 사진

자동 생성된 설명 thread\_wake\_up함수는 다음과 같다.

1. priority scheduling & advanced scheduling
   * 1. threads/thread.h

텍스트이(가) 표시된 사진

자동 생성된 설명

priority aging을 위해 boolean type의 변수를 선언한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Thread 구조체에 nice와 recent\_cpu를 추가로 선언한다.

* + 1. threads/init.c

텍스트이(가) 표시된 사진

자동 생성된 설명

Thread\_prior\_aging의 값을 true로 설정해준다.

* + 1. thread/thread.c

텍스트이(가) 표시된 사진

자동 생성된 설명

Current thread가 idle 하지 않을 경우 ready\_list에 current thread의 element를 priority 순으로 삽입한다. 이후 current thread를 ready 상태로 변경한 후 schedule 함수를 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Ready\_list에 priority 순으로 thread의 element를 삽입한다. 이후 thread를 ready 상태로 바꿔준다.

텍스트, 모니터, 닫기이(가) 표시된 사진

자동 생성된 설명

두 thread의 priority를 비교하여 그 값을 boolean type으로 return한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Current thread의 priority를 수정한다. 새로운 priority가 원래 priority보다 작으면 thread\_yield를 통해서 다시 scheduling한다.

텍스트, 실내, 어두운, 설정이(가) 표시된 사진

자동 생성된 설명

priority aging을 위해 thread\_prior\_aging을 boolean type 변수로 선언한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Thread 구조체에 새로 추가한 nice와 recent\_cpu 변수를 0으로 초기화한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Current thread의 priority를 return해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

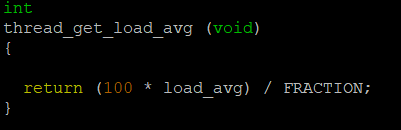
현재 thread의 nice값을 전달받은 nice로 설정하는 함수이다. 따라서 nice값을 변경하고 priority를 새로 계산해준다. 계산식은 아래의 계산식을 이용하였다.

Current thread의 nice를 전달받은 새로운 값으로 변경하는 함수다. Nice 값을 변경하고 priority를 *priority* = PRI\_MAX – (*recent\_cpu* / 4) - (*nice* \* 2) 식을 통해 계산해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Current thread의 nice 값을 return해준다.



current thread의 load average를 계산해서 return해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Current thread의 recent\_cpu를 계산해서 return해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 process의 nice와 recent\_cput 값을 update해준다. Load\_avg는 *load\_avg* = (59/60) \* *load\_avg* + (1/60) \* *ready\_threads* 식을 통해 계산해준다.

그 이후 thread가 idle하지 않다면 recent\_cpu를 *recent\_cpu* = (2 \* load\_avg) / (2 \* load\_avg + 1 ) \* *recent\_cpu* + nice 통해 다시 계산해서 저장한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 process의 priority를 update한다. 계산은 thread\_set\_nice에서 사용한 식으로 동일하게 한다.

* + 1. threads/synch.c

텍스트이(가) 표시된 사진

자동 생성된 설명

Waiters 내부 thread를 탐색해서 가장 높은 priority를 가지고있는 thread를 탐색한다. 이후 list\_remove를 통해 list를 제거하고 unblock thread 상태로 변경한다. 그 이후 yield 함수를 통해 rescheduling한다.

* + 1. devices/timer.c

텍스트이(가) 표시된 사진

자동 생성된 설명

priority aging과 multi level feedback queue에 대한 처리를 해준다. Tick이 증가하면 thread의 recent\_cpu 값 역시 1씩 증가한다. TIMER\_FREQ만큼 tick이 흘렀으면 모든 thread의 recent\_cpu와 nice값을 update한다. Tick이 4만큼 지나면 모든 process의 priority를 update한다.

* 1. **시험 및 평가 내용**

1. priority-lifo

텍스트이(가) 표시된 사진

자동 생성된 설명

Priority-lifo.c의 코드는 동일한 priority를 갖는 thread를 여러 개 만든 후 동일한 round-robin 순서로 실행되도록 하는 코드이다. 따라서 16개의 thread가 LIFO 방식으로 16번 반복되는 모습을 위 실행 결과에서 확인할 수 있다.

1. Make check

텍스트이(가) 표시된 사진

자동 생성된 설명