**Pintos Project 3: Threads**

담당 교수 : 김영재

조 / 조원 : 원종윤

개발 기간 : 23.11.15-23.11.19

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

1. 기존 Alarm clock의 비효율적 작동 방식 개선.
2. 기존 thread scheduling 방식을 priority에 따라 작동하도록 수정.
3. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock
  2. Priority Scheduling
  3. Advanced Scheduler (추가구현을 한 경우)

1. Alarm clock의 작동 구조를 개선하여 thread가 while loop을 통해 자원을 낭비하는 대신 block\_list에 대기한 후 scheduling되도록 한다.
2. Priority scheduling 방식을 구현하여 각 thread마다 할당 받은 priority가 가장 높은 thread를 schedule 되도록 한다.
   1. **개발 내용**

* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.
2. Ready list에 running thread보다 높은 priority를 가진 thread가 들어올 경우 priority scheduling에 따르면 어떻게 해야하는지 서술.
3. Advanced Scheduler에서 priority 계산에 필요한 각 요소를 서술. (추가구현을 한 경우)
4. Timer interrupt 함수가 호출될 때마다 block\_list를 순회하여 wait\_time이 ticks보다 작은 경우 list에서 제거한 후 thread\_unblock을 호출한다.
5. Thread\_create의 경우를 예시로 들어보면, 새로운 thread t->priority가 현재 thread priority보다 높을 때 thread\_yield를 호출한다. 이후 기존 thread를 read\_list에 삽입하고 schedule을 호출한다.
6. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11.15-11.17 alarm clock을 개선한다.

11.17-11.19 priority shceduling을 구현한다.

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수

timer.c thread.c thread.h init.c synch.c 를 수정해야 한다.

Struct thread에 wake\_time을 추가해야 한다.

Static struct list block\_list를 추가해야 한다.

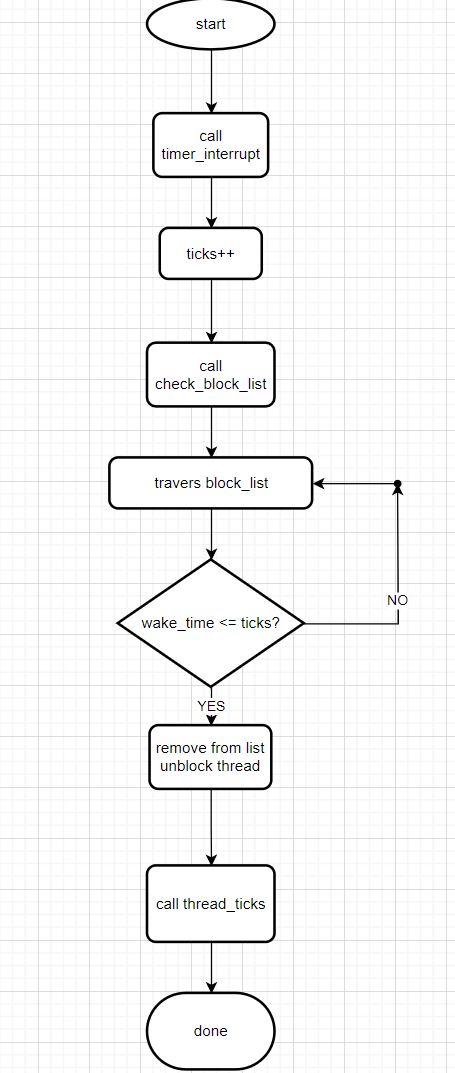
Thread\_init에 block\_list를 list\_init하도록 수정해야 한다.

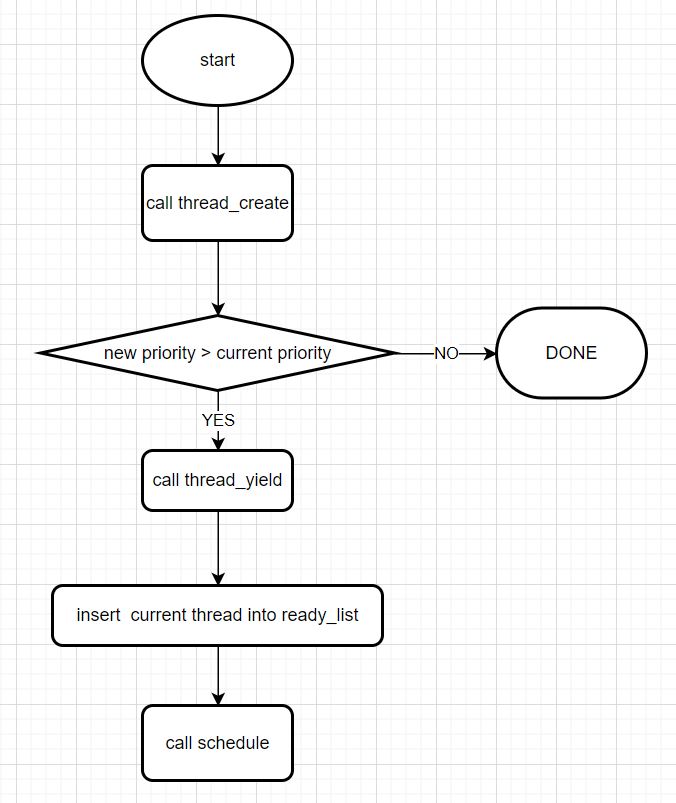
Thread\_create, thread\_block, thread\_unblock, thread\_yield, thread\_set\_priority을 수정해야 한다.

Check\_block\_list, set\_thread\_block, t\_compare\_priority를 추가해야 한다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성  
  (추가구현에 대해서는 flow chart를 작성하지 않아도 됨)

****

****

* 1. **제작 내용**
* 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
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

1-1) check\_block\_list 함수를 구현했다.

Block\_list를 순회하며 wake\_time이 ticks보다 작거나 같은, 일어날 시간이 된 thread를 block\_list에서 제거하고 unblock 처리하도록 하는 함수다. 이를 timer\_interrupt마다 호출하도록 하여 기존의 while loop를 대체했다.

1-2) set\_thread\_block을 구현했다.

현재 thread를 block list에 삽입하고 wake time을 기록한다.

1\_3) timer\_sleep을 수정했다.

Set\_thread\_block을 통해 thread가 block되기 전 data를 저장하고 thread\_block을 호출하도록 수정했다.

2-1) thread\_create을 수정했다.

새로 생성한 thread의 priority가 기존 thread보다 높은 경우 thread\_yield를 호출하도록 했다.

2-2) thread\_unblock을 수정했다.

기존 list\_insert를 list\_insert\_ordered로 대체했다.

2\_3) t\_compare\_priority를 구현했다.

List\_insert\_ordered의 3rd argument에 전달할 함수로서, 전달받은 list에서 thread를 추출하여 priority의 대소관계를 bool로 반환한다.

2\_4) thread\_yield를 수정했다.

기존 list\_insert를 list\_insert\_ordered로 대체했다.

2-5) thread\_set\_priority를 수정했다.

현재 thread의 priority와 새로 수정할 priority를 비교하여 priority가 작아졌다면 새로 schedule 되도록 thread\_yield를 호출한다.

1. 개발 이슈

3-1) sort

기존 아이디어는, priority에 변화가 있을 때마다 priority\_sort 함수를 호출하여 정렬하고자 하였다. 이에 따라 thread.c에 정렬 함수 sort\_ready\_by\_priority도 구현하였으나, sort 호출 이전의 짧은 간격에 switch가 발생하여 의도한대로 작동하지 않는 이슈가 발생하여, pintos에서 제공하는 list\_insert\_ordered를 사용하도록 수정했다.

3-2) 컴파일 오류

이전 project까지 다루지 않던 timer.c 파일에서 코드를 구현해야 했다. 따라서 timer.c에서 thread.c의 함수나 변수를 사용하고자 include에 추가하였으나 이중 선언 문제가 발생하였다. 이에 따라 구현 방식을 변경하여 thread.c의 변수를 조작하기 위해 thread.h에 조작 함수를 선언하고, timer.c에서 함수만 호출하도록 구현하였다.

3-3) alarm-priority

Max\_priority와 비교하고, max\_priority보다 큰 값을 갖는 thread를 sema\_up에서 처리하는 과정에서, max\_priority 초기 값을 최소값 0으로 설정하니, test case 가운데 priority가 0인 경우가 존재하여 max가 갱신되지 않는 경우가 발생했다. 따라서 초기값을 -1로 수정하였다

* 1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석
* make check 수행 결과를 캡처하여 첨부

simple\_thread\_data는 thread id, 반복 횟수, 출력 lock, 출력 버퍼 위치 pointer를 갖는다.

THREAD\_CNT: 생성 스레드 수

ITER\_CNT: 스레드마다 반복 횟수

Simple\_thread\_func는 스레드가 수행하는 함수로, lock acquire 후 출력 버퍼를 id로 수정하고 lock release 후 yield한다.

Test\_priority\_lifo:

mlfqs인지 확인하고, 현재 thread priority가 DEFAULT 값인지 확인한다.

이후 테스트 메시지를 출력한다.

Output을 할당하고, lock을 초기화한다.

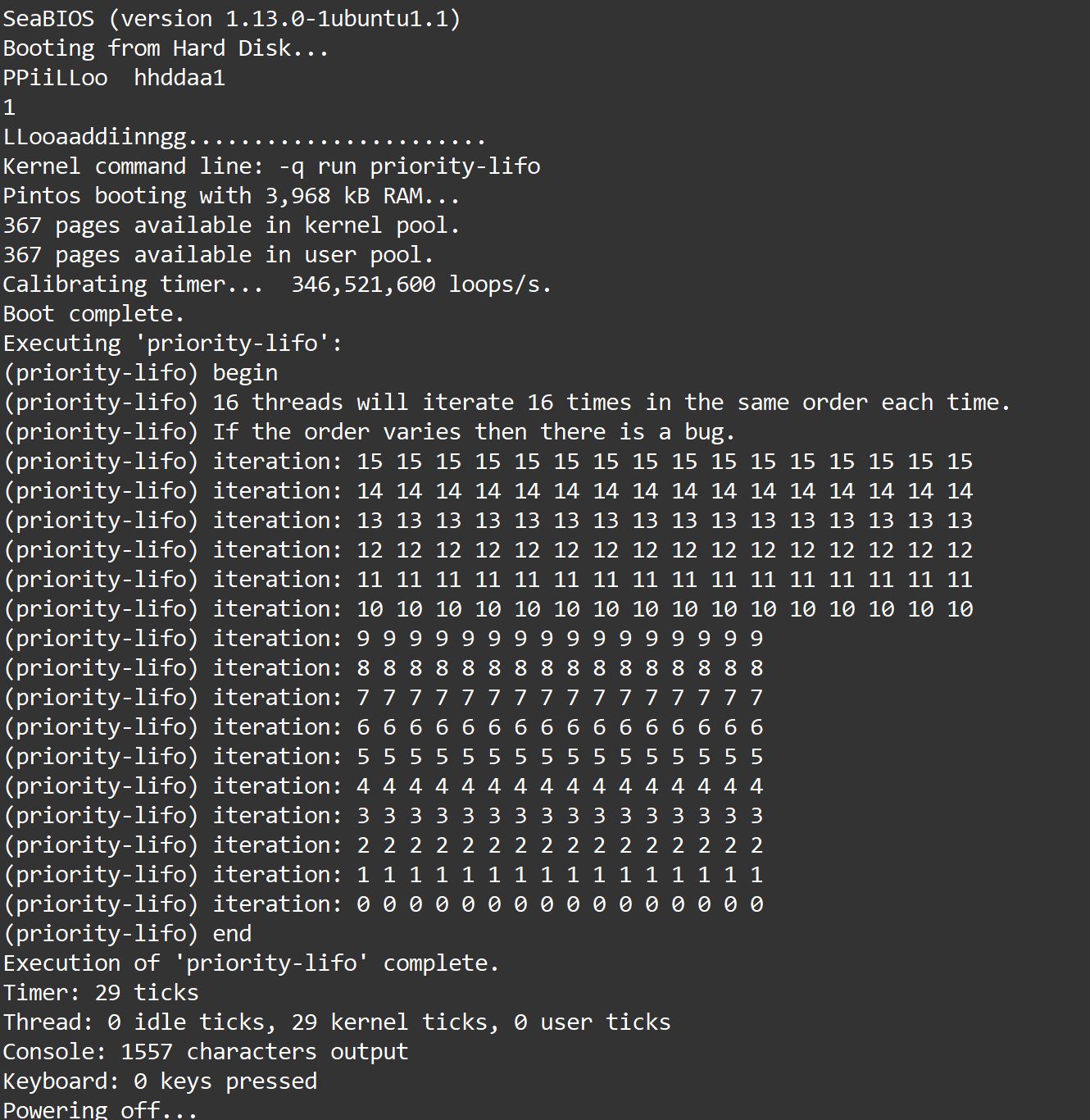
현재 thread priority를 PRI\_DEFAULT + THREAD\_CNT + 1로 수정한다.

이후 TNREAD\_CNT 개수만큼 스레드를 생성하고, data 값을 저장한다.

현재 thread의 priority를 기본 값으로 수정한다.

Lock을 확인하고, 결과를 출력한다.

요약: 같은 priority thread들이 round-robin으로 정상 작동하는지 확인한다.



Round-robin에 의해 정상적으로 작동했다.

main thread를 thread 생성 후 priority를 DEFAULT로 수정하였기에 priority가 15인 thread부터 차례차례 실행된다. 이 때, priority에 따라 ready\_list에서 schedule 되도록 구현하였기에 출력은 위와 같다.

