**Pintos Project 3: Threads**

담당 교수 :박성용 교수님

학번 / 이름 :20191613/윤상현

개발 기간 :

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술.

Alarm clock, priority scheduling, bsd scheduler를 구현할 것입니다.

Alarm clock : thread\_yield의 동작을 좀더 효율적으로 (지금은 busy waiting)

Priority schedule : 지금 단순 rr에서 우선순위를 추가

Bsd scheduler: 더 나아가, mlfq 방식의 스케쥴러를 구현

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Alarm Clock

현재는 간단한 rr 방식의 스케쥴이 구현되어 있고, 각 스레드마다 반복문을 실행하는 방식으로 타이머가 구현되어있습니다. Block 상태와 그 큐를 추가함으로써 좀 더 효율적인 수행을 할 수 있습니다.

* 1. Priority Scheduling

단순히 rr 방식으로 구동하는 현재 방식에서, 우선도에 따라 cpu를 양보하도록 하여 좀더 원하는 방식으로 프로그램이 효율적으로 돌아가게 할 수 있습니다. 우선도가 낮은 일이 아예 실행되지 않는 경우를 위하여, aging 방법을 사용할 수 있습니다.

* 1. Advanced Scheduler (추가구현을 한 경우)

단순 우선순위만을 고려할 경우, 우선순위에서 밀려나는 일들은 cpu를 거의 쓰지 못하는데, 이러한 부분을 추가로 관리할 수 있게 됩니다. 또한, nice, recent\_cpu, load\_avg를 기반으로 우선순위를 적절히 최적화 해주기 때문에 더 효율적인 구동이 가능합니다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.

지금처럼, 스레드 내부에서 while문을 돌리는 것이 아니라, 매 틱마다 호출되는

timer\_interrupt에서, block\_queue에서 깨어날 시간이 된 스레드가 있는지 여부를

체크할 수 있습니다. 있다면 그 스레드를 레디 큐로 옮겨 주면 깨울 수 있습니다.

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

priority schedule에서는 즉시 cpu를 더 높은 thread에게 양보해 줍니다. 그 후에, 원래 방식에서는, 그 우선도와 상관없이 원래 cpu를 쓰던 스레드는 레디큐의 가장 뒤로 보냈지만, 지금은 우선도 높은 순으로 리스트를 유지하기 위하여 적절한 위치에 삽입해 줍니다.

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

Nice: 양수면 우선도 내리고, 음수면 우선도를 올립니다. (-20 ~20)

Priority: 우선도입니다. 4틱마다 갱신됩니다. (0~63)

* + - *priority* = PRI\_MAX – (*recent\_cpu* / 4) - (*nice* \* 2)

Recent\_cpu: 최근에 cpu를 쓴 시간이고, 매 초마다 갱신됩니다. Running 상태인 thread는 매틱마다 +1을 얻습니다. 각 스레드마다 값을 가집니다.

recent\_cpu = (2 \* load\_avg) / (2 \* load\_avg + 1 ) \* recent\_cpu + nice

load\_avg: 레디 큐에 있는 스레드의 평균 수에 대한 값입니다. 0으로 시작해서 매초 갱신됩니다. 모든 스레드가 1개의 값을 공유합니다.

load\_avg = (59/60) \* load\_avg + (1/60) \* ready\_threads

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

먼저, alarm clock관련하여 thread\_yield를 좀더 효율적으로 수정한 이후, 우선순위 기반

스케쥴링을 구현합니다. 그 후에 sema관련하여 수정을 합니다.마지막으로, 추가 구현인 bsd scheduler를 구현합니다. 소수점 연산관련 함수를 구현한 이후에, mlfqs 관련 함수들을 구현합니다.

* 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
* Devices/timer.c

mlfq방식을 위하여, timer\_interrupt()에서 1틱, 4틱, 1초마다 recent\_cpu, priority, load\_avg, recent\_cpu를 갱신해주는 부분을 추가합니다.

* Thread.c
* Thread\_sleep: busywating을 대신해서 스레드를 재우는 함수입니다.
* Thread\_wkae: 잠든 스레드를 다시 readylist로 옮겨주는 함수입니다.

부동소수점 연관 함수인 mul\_f, div\_f, f\_to\_i

Thrad\_check(preemtion이 일어날지 여부 체크), thread\_sort(리스트에 순서에 맞게 삽입)

, thread\_aging(레디큐의 priority +1),

Update\_priority, update\_recent\_cpu, update\_load\_avg., compare\_priority함수를 작성했습니다.

thread.h

recent\_cpu, prirority, load\_avg ,F(2^14 매크로) 추가.

Wake\_tick추가(일어날 시간) block\_list추가(sleep 으로 인해 block된 들리스트)

* synch.c

sema\_down부분에서 priority순으로 waiter에 들어가도록 수정

* 그리고, aging 부분 구현을 위하여 thread.c, thread.h, init.c에 -aging, prior\_aging 관련 구문들을 추가해줍니다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성
* 1텍스트이(가) 표시된 사진

  자동 생성된 설명
* 2
* 
* 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
* Devices/timer.c

mlfq방식을 위하여, timer\_interrupt()에서 1틱, 4틱, 1초마다 recent\_cpu, priority, load\_avg, recent\_cpu를 갱신해주는 부분을 추가합니다. (Ticks%4, ticks%TIME\_FREQ로 판별함.)

* Thread.c

Thread\_sleep: busywating을 대신해서 스레드를 재우는 함수입니다. 얼만큼 재울지 ticks를 받아서, 현재틱 + ticks 를 thread의 wake\_tick에 저장해놓고, readylist에서 해당 스레드를 제거합니다. 그후, block\_list에 추가합니다.

Thread\_wake: wake\_tick이 됬거나 지난 스레드가 있는 지 여부를 block\_list를 훑어서 검사하고 있다면, block\_list에서 빼주고 thread\_unblock을 호출하여 스레드를 깨워줍니다.

부동소수점 연관 함수인

mul\_f,div\_f, f\_to\_i:나머지 연산은 그냥 그 때 그 때 F를 곱하고 나누어주었습니다.

Thrad\_check: running\_thread 와 ready\_thread의 맨 앞요소(이미 정렬되어 있으므로 가장 우선도 높은 스레드임)의 우선도를 비교하여, preemtion이 일어나야 하면, thread\_yield를 호출합니다.

thread\_sort: list 를 for문으로 돌리면서 지금 삽입할 스레드의 우선도보다 작은 것이 나오면 그 앞에 삽입해 줍니다. (우선도 같은 건 x)

thread\_aging(레디큐의 priority +1),해줍니다.

Update\_priority: priority 계산 공식에 따라서 priority를 업데이트합니다. 그, 후 list\_sort함수를 사용하여 readylist를 정렬해줍니다.

update\_recent\_cpu: 1틱마다idle 이 아니 러닝 스레드의 값을 +1, 1초마다 모든 스레드의 값을 업데이트 해줍니다.

update\_load\_avg: 1초마다 load\_avg의 값을 업데이트 해줍니다.

compare\_priority: list\_sort에 들어갈 비교함수인데, 우선도가 높은 순으로 정렬하는 내용입니다.

Set\_priority: priority를 바꾸고, preemtion이 일어나는지 여부를 thread\_check로 체크해줍니다. Mlfqs가 켜져있으면 즉시 종료합니다.

, get\_priority: 현재 스레드의 우선도를 반환합니다.

set\_nice: 현재 스레드의 nice를 바꾸고, 그에따른 우선도도 바꾸고, 그에 따라서 thread\_check로 preemtion여부를 확인합니다.

get\_nice: nice를 반환합니다.

get\_load\_avg: load\_avg를 반환합니다.

get\_recent\_cpu: recent\_cpu를 반환합니다.

* thread.h

recent\_cpu, prirority, load\_avg ,F(2^14 매크로) 추가.

F는 부동소수점 연산에쓰이는 2^14 == 16384 입니다.

* synch.c

sema\_down부분에서 priority순으로 waiter에 들어가도록 수정(thread\_sort함수 사용)

* 그리고, aging 부분 구현을 위하여 thread.c, thread.h, init.c에 -aging, prior\_aging 관련 구문들을 추가해줍니다.
  1. **시험 및 평가 내용**
* priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석

15~0이 순서대로 각 16번씩 반복되어 출력된 모습.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

메인 스레드는 우선도가 pri\_default + 17이었다가, 생성이끝난후는 default가 됨.

0~15번 스레드는 각각 id가 0~15이고 우선도는 pri\_default + (1~16)이다.

우선도가 제일 높은 `15번 실행->lock에 waiter가 없으므로 그냥 id를 op에 16번 쓰고 yield실행.

14,13,12,,,,,2,1,0번째 스레드까지 동일하게 실행됨.

텍스트이(가) 표시된 사진

자동 생성된 설명 그 후 다시 메인 스레드가 실행됨. 후 op를 출력함 🡪 15,.14,13,,,2,1,0순으로 각각 16번씩 출력되어야 함. 🡪 제대로 출력되어 있습니다.

* make check 수행 결과를 캡처하여 첨부
* 텍스트이(가) 표시된 사진

  자동 생성된 설명