**Pintos Project 3: Threads**

담당 교수 : 김영재

조 / 조원 : 신지원

개발 기간 : 2024.11.20 ~11.26

1. **개발 목표**

프로젝트 1, 2에서 구현한 프로그램이 더욱 효율적인 환경에서 실행될 수 있도록 thread 와 관련된 기능을 구현한다. 구현은 alarm clock, priority scheduling, advanced scheduler이며 세 부분으로 나뉜다.

1. **개발 범위 및 내용**
   1. **개발 범위**

**1. Alarm clock**

지금까지 userprog 을 구현하는데 집중하였기 때문에, thread 와 관련된 기능은 기본적으로만 구현되어 있었다. Process 별로 block 함수가 호출 될 때 정해진 시간동안 running과 ready state 을 왕복하는 busy waiting 방식으로 구현되어 있었고, 이는 비효율적인 방식이다. 따라서 blocked process 를 위와 같이 처리해주기 보다 list 내에서 관리하여 효율성을 높이고자 하였다.

**2. Priority Scheduling**

1번에서 언급한 대로, thread는 기본적인 기능만 구현되어 있었기 때문에 thread 의 scheduling 또한 단순한 round-robin 으로 구현되어 있었다. 즉, 다시 말해 프로세스를 순차적으로 실행시킬 뿐 priority 는 고려되지 않고 구현된 것이다.

따라서 priority scheduling 을 구현하여 우선순위가 고려하여 thread list 에 삽입하여 우선순위가 높으면 먼저 실행될 수 있도록 기존 scheduling 을 향상시키고자 한다. 하지만 이렇게만 구현할 경우, 수업시간에 배운 대로 너무 오랫동안 실행되지 않은 thread 에게 starvation이 발생할 수 있기 때문에, 시간에 따라 priority가 점차 증가하는 aging까지 구현하고자 한다.

**3. Advanced Scehduler**

BSD Scheduler 을 구현하고자 한다. 기본적으로 priority scheduling 을 채택하고 있기 때문에 nice와 recent-cpu, load\_avg 을 고려하여 priority 를 계산하도록 구현해야 한다. 또한 시간에 따라 계속해서 update하도록 하여 더욱 효율적인 scheduler 을 구현하고자 한다.

* 1. **개발 내용**

1. **Blocked 상태의 스레드를 어떻게 깨울 수 있는지 서술.**

Blocked 상태의 스레드를 깨우기 위해서 list 를 사용하였다. blocked\_threads 로 선언하였다. thread block 함수가 호출되면 해당 time 을 thread 에 저장하고 thread 를 blocked\_threads 리스트에 추가한다. 따라서 timer interrupt 가 호출될 때마다 앞선 리스트를 순회하면서 현 시점에 해당 thread 들을 깨울지에 대해 결정한다. 만약 깨워야 한다면, blocked thread list 에서 해당 thread 를 제거하고 다시 ready queue 에 넣는다.

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

현재 실행중인 thread 의 우선순위보다 더 높은 우선순위를 가진 thread 가 ready list 에 들어오게 된다면, 당연히 기존의 thread 를 멈추고 ready 상태로 전환한 뒤, 새로 들어온 thread 를 실행시켜야 한다. 이때 ready 상태로 전환하기 위해서는 ready list 에 우선순위를 고려하여 삽입해야 한다. 그 뒤, ready list 에 있던 thread 중 우선순위가장 높은 thread 를 running 상태로 변환하여 실행한다.

이 과정에서 thread 의 우선순위를 고려하여 thread 를 삽입/정렬 하며, priority aging 값 등의 설정을 고려하며 ready list 를 업데이트 한다.

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

Advanced Scheduler 는 priorty 를 계산하여 list 를 정렬해야 하기 때문에 계산식이 필요하다.

텍스트, 폰트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

recent\_cpu와 nice 값은 우선순위에 영향을 미친다. recent\_cpu가 클수록 우선순위는 낮아지고 nice 값이 클수록 스레드가 더 낮은 우선순위를 가지며, 이는 CPU 점유를 덜 요구한다는 의미다. 기본적으로 모든 스레드는 nice = 0에서 시작하며, 새로 생성된 스레드는 부모 스레드의 nice 값을 상속받는다.

텍스트, 폰트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

따라서 recent\_cpu 를 계산하기 위해선 load\_avg 이 필요하다. 실행 중인 스레드의 recent\_cpu는 타이머 인터럽트마다 1씩 증가하며, 이는 스레드가 얼마나 CPU를 점유했는지 추적한다.

텍스트, 폰트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

ready\_threads는 READY 상태와 RUNNING 상태의 스레드를 모두 포함하며, 이는 BSD 스케줄러의 계산 방식에 부합합니다. load\_avg는 매초마다 위의 식에 맞추어 갱신된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

11/20 ~ 11/21: 프로젝트 설계 및 매뉴얼 이해

11/21 ~ 11/23: Alarm clock & priority Scheduling 구현

11/24 : Advanced Scehduler 구현

11/25 ~ 11/26 : 에러 해결 및 보고서 작성

* 1. **개발 방법**

**1. Alarm clock**

Alarm clock 을 구현하기 위해 초기 세팅이 필요하다. 우선, thread status 가 block 에서 ready 로 돌아갈 때, time 을 저장할 변수를 선언해야 한다. thread.h 의 thread 구조체 내에 변수를 생성하고 thread.c 에서 이 변수를 초기화 해야 한다. 뿐만 아니라 해당 기능을 수행하는 timer.c 에서 blocked\_threads 리스트를 선언하고 초기화한다.

앞선 A 파트에서 말했던 busy waiting scheduling 에서 thread 의 상태를 반영해주는 scheduling 로 변경하여 thread 가 block 되는 상태를 저장하고 저장한 time 이 호출할 때 잘 반영될 수 있도록 한다.

이후 tick 마다 blocked\_threads 리스트를 순회하며 현재 시점이 blocked 된 thread 를 불러야하는 시점인지 확인한다. 만약 현재의 시점보다 저장된 time 이 같거나 클 경우, unblock 시키고 blocked\_threads 리스트에서 제거하고 이외의 경우엔 다음 번째 thread 를 체크하도록 한다. 이는 매 tick 마다 동작되어야 하기 때문에 timer interrupt 함수에서 구현한다.

**2. Priority Scheduling**

thread 의 switching 의 발생을 고려해야 priority scheduling 을 구현한다. 단순히 구현된기존 thread\_yield 함수를 list.c 에 정의되어 있는 list\_insert\_ordered() 함수를 이용해 priorty 를 고려하여 내림차순으로 thread 를 정렬한다. 이때, 비교를 위해 myPriorityCompare 함수를 정의하고 thread.h 에 추가하였다.

**3. Advanced Scehduler**

BSD Scheduler 를 구현하기 위해서는 CPU 가중치와 nice 값을 선언해야 하기 때문에 각각 recentCPU, nice 를 선언하였다. 따라서 이를 thread\_init, init\_thread 함수에서 초기화해주었다.

recentCPU 는 각각의 thread 의 값을 선언한다면, load\_avg 는 시스템의 값을 선언하기 때문에 이는 전역변수에 선언해주었다.

1. **연구 결과**
   1. **Flow Chart**

**1. Alarm clock**

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

**2. Priority Scheduling**

도표, 라인, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

/\* 사선으로 꺾인 화살표(삼각형으로 된 것)는 무시해주세요! \*/

* 1. **제작 내용**

**1. Alarm clock**

1-1) 초기화

가장 먼저 thread.h 에서 alarm 시간을 저장할 변수를 선언을 해야 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그 뒤에 thread.c 에서 초기화 해주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

1-2) thread 깨우기

time 과 관련된 파일인 deviecs/timer.c 에서 timer\_sleep(), timer\_interrupt() 내부를 구체화하여 시간에 맞게 alarm 이 울릴 수 있도록 설정해주었다. 그러기 위해서는 blocked 된 thread 들을 모아주는 변수가 필요했다. 따라서 timer.c 상단에 선언해주고 timer\_sleep, timer\_interrupt 함수에서 이 리스트를 관리하며 깨울 thread 가 있는지 확인해주었다.

텍스트, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

timer\_interrupt 함수에서는 while 문 안에 조건문을 추가하였다. 이는 깨울 시간인지 아닌지를 파악하는 데 사용된다. 만약 깨울 시간이라면( 그 시간 이상이라면) bloked\_threads 리스트에서 삭제하고 thread 의 block 상태를 해제하였다.

**2. Priority Scheduling**

2-1) MLFQ 와 구분

Pintos 는 우선순위가 고정되지 않은 MLFQ 로 구현되었기 때문에 이를 구분지어주기 위하여 thread.h, thread.c, init.c 에서 변수를 선언하고 초기화 해주었다. 따라서 thread\_prior\_aging 변수를 통해 priority 를 수행해야 하는지 체크할 수 있다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라일락이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2-2) 우선순위 고려한 yield 구현

단순히 ready\_list 끝에 thread 를 밀어넣은 기존 로직에서 priorty 를 고려한 로직으로 수정해야 한다.

단순히 yield 함수만을 수정하기 보다 만약 thread 가 생성될 때 현재 진행되는 thread 의 우선순위보다 더 크다면, 현재 thread 를 yield 로 전환해야 하기 때문에 thread\_create 함수에서 위 로직을 추가하였다. 따라서 thread 를 setting 하는 함수와 get priority 를 get 하는 함수도 수정하여 우선순위를 고려할 수 있도록 구현하였다.

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그 뒤에 앞선 함수에서 호출되는 thread.c 의 thread\_yield 함수를 수정하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

내장되어있는 list\_insert\_ordered 함수를 사용하여 순위를 결정지어 READY 상태로 전환하였다. list\_insert\_ordered 의 비교 함수 포인터를 사용하기 때문에 이 포인터의 영역에 우선 순위를 고려할 수 있는 함수를 넣어주어야 한다. 따라서 myPriorityCompare 함수를 선언하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이는 위처럼 현재와 다음 thread 를 비교하는 함수를 구현하였고, 이 함수는 앞서 언급한 것과 같이 interrupt 함수 뿐만 아니라 unblock 함수에서도 사용되었다.

2-3) synchronize 구현

본 과제에서는 우선순위를 고려하는 것이기 때문에 동기화하는 과정에서도 우선순위에 대한 고려가 필요하다. 따라서 synch.c 의 파일 내부의 sema\_up, sema\_down 함수에서도 이를 구현해주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위 로직은 리스트의 첫 번째 요소부터 돌며 우선순위를 고려하여 우선순위가 높다면 unblock 처리 하는 과정을 수행한다. 이 로직이 끝나고 난 뒤엔 thread\_yield() 를 호출하여 로직에서 처리한 순위가 즉시 반영될 수 있도록 구현한다. 사실 코드 위쪽에 있는 주석을 보았을 때 기존에 구현한 myPriorityCompare 함수를 재사용하거나 semaPriortyCompare 함수를 semaphore 에 맞게 구현하여 사용하고자 하였지만, semaphore\_elem 구조체가 sema\_up, sema\_down 과 같은 파일에 정의되어 있어 구조를 바꾸어주어야 해 단순히 sema\_up, sema\_down 함수 자체에 구현하였다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Sema\_down 또한 비슷하게 구현되었다. 다만, 현재 스레드의 우선순위가 높을 경우를 체크하는 것이 sema\_up 과 차이가 있다.

**3. Advanced Scehduler**

3-1) 초기화

BSD 스케줄러를 구현하기 위하여 몇 가지 초기화 작업을 해주었다. 우선 앞서 구현한 alarm\_tick 과 같이 thread.h, thread.c 에서 변수를 초기화 해주었다.

* 텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 핑크이(가) 표시된 사진

자동 생성된 설명

또한 thread.c 에서 fixed point 연산을 위한 함수들을 #define 처리 해주었다.

텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

정의한 nice 와 recentCPU 를 불러오는 함수도 수정하여 fixed point 연산을 고려한 nice, recentCPU 가 구현될 수 있도록 하였다.

thread\_prior\_aging과 thread\_mlfqs flag가 set된 경우 시간에 맞추어 recentCPU 를 호출해야 한다.

따라서 아래와 같이 구현하였고, 호출하는 함수를 추가로 구현해 주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

recent\_cpu와 load\_avg의 경우 매 초마다 새롭게 갱신되어야 한다. 이를 위해서 갱신되는 함수를 구현하였고, 이는 thread 의 프로퍼티들을 사용해야 하기 때문에 threads 파일 내에 있는 thread.c 에서 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이는 앞서 B 개발방법 – 3번 문항에서 작성한 식을 고려하여 구현되었다.

* 1. **시험 및 평가 내용**

**priority-lifo.c 코드 및 priority-lifo 테스트 결과 분석**

* 1. priority-lifo.c 코드 분석

priority-lifo.c 함수는 스레드의 실행 순서를 기록할 배열 output과 op를 동적메모리로 할당하고 락(lock)을 초기화하여 스레드 간 동기화를 설정한다. 현재 스레드의 우선순위를 모든 새로 생성되는 스레드보다 높게 설정하고 있다 또한 현재 스레드의 우선순위를 모든 새로 생성되는 스레드보다 높게 설정하고, 이 설정을 통해 새로 생성된 스레드가 즉시 실행되지 않고, 대기 상태로 유지되도록 한다. 따라서 락이 특정 스레드에 의해 보유된 상태로 유지되면 동기화 문제가 발생했음을 알린다.

* 1. priority-lifo 테스트 결과

**텍스트, 스크린샷, 문서, 패턴이(가) 표시된 사진

자동 생성된 설명**

**make check 수행 결과**

* 텍스트, 스크린샷, 문서, 패턴이(가) 표시된 사진

  자동 생성된 설명