1. 프로젝트 1

이 과제에서는 최소한으로 동작하는 스레드 시스템을 제공합니다. 여러분의 과제는 이 시스템의 기능을 확장하여 동기화 문제에 대한 더 나은 이해를 얻는 것입니다.
이 과제를 수행하기 위해 주로 threads 디렉토리에서 작업하며, 일부 작업은 devices 디렉토리에서 진행됩니다. 컴파일은 threads 디렉토리에서 이루어져야 하며, 필요시 파일을 추가할 수 있습니다.

1-1. 프로젝트 요구사항

1-1-1. 알람 시계

devices/timer.c에 정의된 **timer_sleep()** 함수를 재구현하십시오. 제공된 구현은 busy waiting를 사용합니다. 즉, 현재 시간을 확인하고 thread_yield()를 호출하는 루프를 반복하여 일정 시간이 지나기를 기다립니다. busy waiting를 피하는 방식으로 재구현해야 합니다.

- void timer_sleep (int64_t ticks) 함수 설명

주어진 시간만큼 스레드의 실행을 일시 중단합니다. 시스템이 다른 작업으로 바쁘지 않은 경우, 스레드는 정확히 해당 시간에 깨어나지 않아도 됩니다. 올바른 시간이 지나면 준비 큐에 다시 넣어야 합니다.

timer_sleep()은 실시간으로 동작하는 스레드에 유용합니다. 예를 들어, 커서를 1초마다 깜박이게 할 때 사용됩니다.

timer_sleep()의 인수는 타이머 tick으로 표현되며 실제 초와 같은 단위가 아닙니다. 초당 TIMER_FREQ 타이머 틱이 발생하며, 이 값은 devices/timer.h에 매크로로 정의되어 있습니다. 기본값은 100입니다. 이 값을 변경하는 것은 권장하지 않습니다.

특정 밀리초, 마이크로초 또는 나노초 동안 대기하는 timer_msleep(), timer_usleep(), timer_nsleep() 함수가 별도로 존재하지만, 이들은 필요시 자동으로 timer_sleep()을 호출합니다. 이 함수들을 수정할 필요는 없습니다.

지연이 너무 짧거나 길다고 생각되면 pintos의 -r 옵션에 대한 설명을 다시 읽어보십시오.

1-1-2. 우선순위 스케줄링

Pintos에서 우선순위 스케줄링을 구현하십시오. 주요 요구 조건은 아래와 같습니다.
- 실행 중인 스레드보다 더 높은 우선순위를 가진 스레드가 준비 목록에 추가되면, 현재 실행 중인 스레드는 즉시 프로세서를 새 스레드에게 양보해야 합니다.
- 스레드가 락, 세마포어 또는 조건 변수에서 대기하게 되면 대기 중인 스레드 중 가장 높은 우선순위를 가진 스레드가 깨어나야 합니다.
- 스레드는 언제든 자신의 우선순위를 높이거나 낮출 수 있지만, 우선순위를 낮추어 더 이상 가장 높은 우선순위를 가지지 않으면 즉시 CPU를 양보해야 합니다.
- 스레드가 자신의 우선순위를 확인하고 수정할 수 있는 다음 함수들을 구현하십시오. 이 함수들의 뼈대는 **threads/thread.c**에 있습니다.

함수: void thread_set_priority (int new_priority)

현재 스레드의 우선순위를 new_priority로 설정합니다. 만약 현재 스레드가 더 이

상 가장 높은 우선순위를 가지지 않게 되면, CPU를 양보합니다.

함수: int thread_get_priority (void)

현재 스레드의 우선순위를 반환합니다. 우선순위 기부가 발생한 경우, 기부된 더 높은 우선순위를 반환합니다.

- **우선순위 기부**(priority donation)를 구현하십시오. 세부 설명은 아래에 있습니다.

스레드의 우선순위는 PRI_MIN(0)~PRI_MAX(63)까지 있습니다. 낮은 숫자가 더 낮은 우선순위를 나타내며, 우선순위 0이 가장 낮고, 우선순위 63이 가장 높습니다. 스레드의 초기 우선순위는 thread_create() 함수의 인수로 전달되며, 특별한 이유가 없으면 PRI_DEFAULT(31)을 사용하십시오. PRI_~ 매크로는 threads/thread.h에 정의되어 있으며, 이 값들을 변경해서는 안 됩니다.

우선순위 스케줄링의 문제점 중 하나는 "우선순위 역전(priority inversion)"입니다. 예를 들어, 높은 우선순위(H), 중간 우선순위(M), 낮은 우선순위(L)를 가진 스레드가 있을 때, H가 L을 기다려야 하는 상황이 발생할 수 있습니다(예: L이 락을 보유하고 있을 때). 이때 M이 준비 리스트에 있다면, H는 CPU를 얻지 못하고 계속 대기하게 됩니다. 이를 부분적으로 해결하는 방법은 H가 L에게 자신의 우선순위를 "기부"하고, L이 락을 해제할 때 우선순위를 반환받는 것입니다. 우선순위 기부(priority donation)를 구현할 때에는 우선순위 기부가 필요한 다양한 상황을 고려해야 합니다. 하나의 스레드에 여러 개의 우선순위가 기부되는 다중 기부도 처리해야 합니다. 또한, 중첩된 기부도 처리해야 합니다. 예를 들어, H가 M이 보유한 락을 기다리고, M이 L이 보유한 락을 기다리는 경우, M과 L 모두 H의 우선순위로 상승해야 합니다. 필요하다면 중첩된 우선순위 기부의 깊이를 제한할 수 있습니다(8단계 정도). 락에 대한 우선순위 기부 역시 구현해야 합니다. Pintos의 다른 synchronization 구조에 대해서는 우선순위 기부를 구현할 필요가 없습니다. 그러나 모든 경우에 우선순위 스케줄링을 구현해야 합니다.

1-2. 배경

1-2-1. 스레드 이해하기

첫 번째 단계는 초기 스레드 시스템의 코드를 읽고 이해하는 것입니다. Pintos는 이미 스레드 생성 및 완료, 스레드 간 전환을 위한 간단한 스케줄러, 기초 동기화 도구(세마포어, 락, 조건 변수, 최적화 장벽)를 구현하고 있습니다.

이 코드의 일부는 다소 복잡해 보일 수 있습니다. 컴파일된 파일 소스 코드를 읽어보고, 원하는 곳에 printf() 호출을 추가하여 재컴파일하고 실행함으로써 무엇이 어떻게 진행되는지 확인할 수 있습니다. 또한, 커널을 디버거에서 실행하여 주요 지점에 중단점을 설정하고, 단일 단계로 코드를 실행하면서 데이터를 조사할 수 있습니다.

스레드 생성은 스케줄링될 새로운 context를 생성하는 것입니다. 이 context에서 실행할 함수를 thread_create() 함수에 인수로 제공합니다. 스레드가 처음으로 스케줄링되고 실행될 때, 이 함수가 실행되고, 함수가 반환되면 스레드가 종료됩니다. 따라서 각 스레드는 Pintos 내에서 실행되는 작은 프로그램처럼 동작하며, thread_create()에 전달된 함수는 C언어의 main()과 같은 역할을 합니다.

한 번에 정확히 하나의 스레드만 실행되며, 나머지는 비활성 상태가 됩니다. 스케줄러가 다음에 실행할 스레드를 결정합니다. 주어진 시점에 ready 스레드가 없으면, idle()에서 구현된 특수한 "idle" 스레드가 실행됩니다. 동기화 도구는 한 스레드가 다른 스레드가 작업을 완료할 때까지 기다려야 할 때 컨텍스트 전환을 강제할 수 있습니다.

컨텍스트 전환의 현재 실행 중인 스레드의 상태를 저장하고, 전환할 스레드의 상태를 복원하는 메커니즘은 threads/switch.S에 있으며, 이는 80x86 어셈블리 코드로 작성되어 있습니다(이 코드를 이해할 필요는 없습니다.).
schedule()에 중단점을 설정하고, 거기서부터 단일 단계로 실행을 시작할 수 있습니다. 각 스레드의 주소와 상태, 그리고 각 스레드의 호출 스택에 어떤 프로시저가 있는지 추적해 보세요. 한 스레드가 switch_threads()를 호출하면 다른 스레드가 실행되며, 새로운 스레드가 처음으로 실행하는 작업이 switch_threads()에서 반환하는 것임을 알 수 있습니다. switch_threads()가 호출되는 방식과 반환되는 방식이 서로 다른 이유와 방법을 이해하면 스레드 시스템을 이해할 수 있습니다.
참고: Pintos에서는 각 스레드에 약 4KB 크기의 고정된 크기의 실행 스택이 할당됩니다. 커널은 스택 오버플로우를 감지하려고 하지만, 이를 완벽하게 할 수는 없습니다. 만약 크기가 큰 데이터 구조를 동적 지역 변수로 선언하면(예: int buf[1000];), 커널 패닉과 같은 이상한 문제가 발생할 수 있습니다.

1-2-2. 소스 파일

threads 디렉토리에 있는 파일에 대한 간략한 개요입니다.
- loader.S / loader.h (확인 X)
커널 로더. PC BIOS가 메모리에 로드하는 512바이트의 코드 및 데이터로 어셈블되며, 디스크에서 커널을 찾아 메모리에 로드한 후 start.S의 start()로 점프합니다.
- start.S (확인 X)
80x86 CPU에서 메모리 보호 및 32비트 작업을 위한 기본 설정을 수행합니다. 로더와 달리 이 코드는 커널의 일부입니다.
- kernel.lds.S (확인 X)
커널을 링크하는 데 사용되는 링크 스크립트. 커널의 로드 주소를 설정하고, start.S를 커널 이미지의 시작 부분 근처에 배치합니다.
- init.c / init.h (확인 권장)
커널 초기화, main()을 포함하며 커널의 "메인 프로그램"입니다. 무엇이 초기화되는지 확인하기 위해 main()을 살펴보는 것이 좋습니다. 여기에 직접 초기화 코드를 추가할 수 있습니다.
- thread.c / thread.h (작업 대상 파일)
기본 스레드 지원. 대부분의 작업은 이 파일들에서 이루어집니다. thread.h는 구조체 thread를 정의하며, 이는 네 개의 프로젝트 모두에서 수정할 가능성이 큽니다.
- switch.S / switch.h (확인 X)
스레드를 전환하는 어셈블리어 루틴입니다. 앞서 설명한 내용입니다.
- palloc.c / palloc.h (확인 X)
시스템 메모리를 4KB 페이지 단위로 할당하는 페이지 할당기입니다.
- malloc.c / malloc.h (확인 X)
커널에서 malloc()과 free()의 간단한 구현입니다.
- interrupt.c / interrupt.h (확인 권장)
기본 인터럽트 처리 및 인터럽트의 켜고 끄는 함수들입니다.

- intr-stubs.S / intr-stubs.h (확인 X)
저수준 인터럽트 처리를 위한 어셈블리 코드입니다.
- synch.c / synch.h **(작업 대상 파일)**
기본 동기화 요소들: 세마포어, 락, 조건 변수 및 최적화 배리어입니다. 프로젝트에서 동기화를 위해 이 기능들을 사용할 것입니다.
- io.h (확인 X)
디바이스 디렉토리의 소스 코드에서 사용되는 I/O 포트 접근을 위한 함수들입니다.
- vaddr.h / pte.h (확인 X)
가상 주소와 페이지 테이블 엔트리를 다루는 함수 및 매크로들입니다.
- flags.h (확인 X)
80x86 "flags" 레지스터의 몇 가지 비트를 정의하는 매크로입니다.

devices 디렉토리에 있는 파일에 대한 간략한 개요입니다.
- timer.c / timer.h **(작업 대상 파일)**
기본적으로 초당 100번씩 틱하는 시스템 타이머입니다.
- vga.c / vga.h
VGA 디스플레이 드라이버입니다. 텍스트를 화면에 출력하는 기능을 담당합니다. printf()가 VGA 디스플레이 드라이버를 호출하므로, 직접 호출할 이유가 거의 없습니다.
- serial.c / serial.h
직렬 포트 드라이버입니다. 역시 printf()가 이 코드를 호출하므로, 직접 호출할 필요는 없습니다. 직렬 입력은 입력 레이어로 전달됩니다.
- block.c / block.h
블록 장치에 대한 추상화 계층입니다. 블록 장치는 고정 크기의 블록으로 구성된 랜덤 액세스 디스크와 같은 장치입니다. Pintos는 기본적으로 IDE 디스크와 파티션의 두 가지 블록 장치를 지원합니다.
- ide.c / ide.h
최대 4개의 IDE 디스크에서 섹터를 읽고 쓰는 기능을 지원합니다.
- partition.c / partition.h
디스크의 파티션 구조를 이해하며, 하나의 디스크를 여러 파티션으로 나눠 독립적으로 사용할 수 있도록 합니다.
- kbd.c / kbd.h
키보드 드라이버입니다. 키 입력을 처리하고 이를 입력 레이어로 전달합니다.
- input.c / input.h
입력 레이어입니다. 키보드나 직렬 드라이버에서 전달된 입력 문자를 큐에 저장합니다.
- intq.c / intq.h
커널 스레드와 인터럽트 핸들러가 접근하려는 원형 큐를 관리하는 인터럽트 큐입니다. 키보드와 직렬 드라이버에서 사용됩니다.
- rtc.c / rtc.h
실시간 시계 드라이버로, 현재 날짜와 시간을 확인할 수 있게 합니다. 기본적으로 thread/init.c에서 무작위 수 생성기의 초기 시드를 선택할 때만 사용됩니다.

- speaker.c / speaker.h
PC 스피커에서 음을 발생시키는 드라이버입니다.
- pit.c / pit.h
8254 프로그래머블 인터럽트 타이머를 설정하는 코드입니다. 이 코드는 devices/timer.c와 devices/speaker.c 모두에서 사용됩니다. 각각의 장치는 PIT의 출력 채널 중 하나를 사용하기 때문입니다.

마지막으로 lib와 lib/kernel에는 유용한 라이브러리 루틴이 포함되어 있습니다.
- ctype.h / inttypes.h / limits.h / stdarg.h / stdbool.h / stddef.h / stdint.h / stdio.c / stdio.h / stdlib.c / stdlib.h / string.c / string.h
표준 C 라이브러리의 일부입니다.
- debug.c / debug.h
디버깅을 돕는 함수 및 매크로들입니다. 자세한 내용은 디버깅 도구를 참조하세요.
- random.c / random.h
의사 난수 생성기입니다. 특정 조건을 제외하고는 Pintos 실행 간에 동일한 난수 시퀀스가 생성됩니다.
- round.h
라운딩에 대한 매크로들이 정의되어 있습니다.
- syscall-nr.h
시스템 호출 번호가 정의되어 있습니다.
- kernel/list.c / kernel/list.h
이중 연결 리스트 자료구조의 구현. Pintos 코드 전반에서 사용됩니다.
- kernel/bitmap.c / kernel/bitmap.h
비트맵 구현.
- kernel/hash.c / kernel/hash.h
해시 테이블 구현.
- kernel/console.c / kernel/console.h / kernel/stdio.h
printf() 및 몇 가지 함수들 구현.

1-2-3. 동기화

적절한 동기화는 이 문제들을 해결하는 중요한 요소입니다. 동기화 문제는 인터럽트를 끄는 것으로 쉽게 해결할 수 있는 것처럼 보입니다. 인터럽트가 꺼져 있으면 동시 실행이 없으므로 race condition이 발생하지 않습니다. 그러나 모든 동기화 문제를 이 방법으로 해결하면 안 됩니다. 대신 세마포어, 락, 조건 변수를 사용해 동기화 문제를 해결해야 합니다. 동기화 원리에 대해 잘 모르겠다면 threads/synch.c의 주석을 참조하십시오.

Pintos 프로젝트에서 인터럽트를 비활성화하여 해결하는 것이 최선인 문제는 커널 스레드와 인터럽트 핸들러 간에 공유되는 데이터를 조정하는 경우뿐입니다. 인터럽트 핸들러는 잠들 수 없으므로 락을 획득할 수 없습니다. 따라서 커널 스레드와 인터럽트 핸들러 간에 공유되는 데이터는 인터럽트를 비활성화하여 보호해야 합니다.

이 프로젝트에서는 인터럽트 핸들러에서 일부 스레드 상태에만 접근합니다. 알람 시계의 경우 타이머 인터럽트가 잠든 스레드를 깨워야 합니다.

인터럽트를 비활성화할 때는 가능한 한 적은 코드에서만 수행해야 하며, 그렇지 않으면 타이머 틱이나 입력 이벤트와 같은 중요한 것을 놓칠 수 있습니다. 또한, 인터럽트를 비활성화하면 인터럽트 처리 지연이 증가해 시스템 성능이 저하될 수 있습니다.

synch.c에 있는 기본 동기화 도구들은 인터럽트를 비활성화하여 구현되어 있습니다. 여기에서 인터럽트가 비활성화된 상태에서 실행되는 코드를 늘려야 할 수도 있지만, 가능한 최소화해야 합니다.

인터럽트를 비활성화하는 것은 디버깅에 유용할 수 있습니다. 코드가 중단되지 않도록 하고 싶은 경우 사용할 수 있지만, 프로젝트를 제출하기 전에 반드시 디버깅 코드를 완전히 제거해야 합니다.

busy waiting을 하지 않도록 주의하십시오. thread_yield()를 호출하는 반복문은 busy waiting의 한 형태입니다.