Pintos

- Pintos is a simple operating system framework for the 80x86 architecture.

- It supports kernel threads, loading and running user programs, and a file system, but it implements all of these in a very simple way

- Academic OS written at Stanford; based on earlier “Nachos” from UC Berkeley
- Written in C and runs on regular x86 hardware
- Much smaller and cleaner than most commercial/consumer-oriented operating systems

Used by several institutions

- Stanford
- Virginia Tech
- University of San Francisco
- University of Salzburg
- Linköping Universitet,
- KAIST, Seoul National University, POSTECH

Pintos Features

- Small enough so entire code can be read and understood by students (less than 1mb)
- Runs on real hardware
- Runs and debugs in emulated environment

Project 1 : Threads

- In this assignment, we give you a minimally functional thread system.

- Your job is to extend the functionality of this system to gain a better understanding of synchronization problems

Project 2 : User Programs

- The base code already supports loading and running user programs, but no I/O or interactivity is possible.

- In this project, you will enable programs to interact with the OS via system calls.

Project 3 : Virtual Memory

- Pintos can properly handle multiple threads of execution with proper synchronization , and can load multiple user programs at once. However, the number and size of programs that can run is limited by the machine's main memory size.

- In this assignment, you will remove that limitation.

Project 4 : File systems

- For this last assignment, you will improve the implementation of the file system.

Pintos Geliştirme Ortamı

- Pintos sadece unix/linux sistemler üzerinde geliştirilebiliyor.

- Biz derste önceden hazırlanmış olan , VirtualBox üzerinde çalışan geliştirme ortamını kullanacağız.
- Moodle'da bulunan pintos linkinden virtualbox dosyasını indirip virtualbox'ta çalıştırmamız yeterli.

Pintos on simulator

- Pintos could, theoretically, run on a regular IBM-compatible PC. Unfortunately, it is impractical to supply every student a dedicated PC for use with Pintos. Therefore, we will run Pintos projects in a system simulator, that is, a program that simulates an 80x86 CPU and its peripheral devices accurately enough that unmodified operating systems and software can run under it. In this class we will use the QEMU simulator. Pintos has also been tested with VMware Player. You can also use Bochs.

Building Pintos

- First, cd into the "threads" directory. Then, issue the "make" command. This will create a "build" directory under "threads", populate it with a "Makefile" and a few subdirectories, and then build the kernel inside.

Running Pintos

- cd $PINTOS_HOME/src/threads/build
- make
- pintos run alarm-multiple
- pintos --qemu -- run alarm-multiple

Testing the submission

- To completely test your submission, invoke make check from the project "build" directory. This will build and run each test and print a "pass" or "fail" message for each one. When a test fails, make check also prints some details of the reason for failure. After running all the tests, make check also prints a summary of the test results.
- make SIMULATOR=--qemu check

Synchronization

- If sharing of resources between threads is not handled in a careful, controlled fashion, the result is usually a big mess. This is especially the case in operating system kernels, where faulty sharing can crash the entire machine. Pintos provides several synchronization primitives to help out.

Disabling Interrupts

- The crudest way to do synchronization is to disable interrupts, that is, to temporarily prevent the CPU from responding to interrupts. If interrupts are off, no other thread will preempt the running thread, because thread preemption is driven by the timer interrupt. If interrupts are on, as they normally are, then the running thread may be preempted by another at any time, whether between two C statements or even within the execution of one.

Semaphores

- A semaphore is a nonnegative integer together with two operators that manipulate it atomically, which are :
- "Down" or "P": wait for the value to become positive, then decrement it.
- "Up" or "V": increment the value (and wake up one waiting thread, if any).

Locks

- A lock is like a semaphore with an initial value of 1 . A lock's equivalent of "up" is called "release", and the "down" operation is called "acquire".

  bool lock_held_by_current_thread :

  Returns true if the running thread owns lock, false otherwise. There is no function to test whether an arbitrary thread owns a lock, because the answer could change before the caller could act on it.

Monitors

- A monitor is a higher-level form of synchronization than a semaphore or a lock. A monitor consists of data being synchronized, plus a lock, called the monitor lock, and one or more condition variables.

Proje 1 hakkında

- In this assignment, we give you a minimally functional thread system. Your job is to extend the functionality of this system to gain a better understanding of synchronization problems.
- You will be working primarily in the "threads" directory for this assignment, with some work in the "devices" directory on the side. Compilation should be done in the "threads" directory.

Projeye başlamadan önce okunması gereken kısımlar

- Before you read the description of this project, you should read all of the following sections: 1. Introduction, C. Coding Standards, E. Debugging Tools, and F. Development Tools. You should at least skim the material from A.1 Loading through A.5 Memory Allocation, especially **A.3 Synchronization.** To complete this project you will also need to read B. 4.4BSD Scheduler.

Proje 1

- The first step is to read and understand the code for the initial thread system.
- Kodu anlamak için uygun gördüğünüz yerlere printf() 'ler ekleyebilirsiniz.

Requirements of Project1

1-)Alarm Clock

- Reimplement timer_sleep(), defined in "devices/timer.c". Although a working implementation is provided, it "busy waits," that is, it spins in a loop checking the current time and calling thread_yield() until enough time has gone by. Reimplement it to avoid busy waiting.
- timer_sleep fonksiyonu kendisini çağıran threadi parametre olarak girilen süre kadar uyutur.

2-)Priority Scheduling

2.1-)Implement priority scheduling in Pintos.When a thread is added to the ready list that has a higher priority than the currently running thread, the current thread should immediately yield the processor to the new thread.

-Similarly, when threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be awakened first. A thread may raise or lower its own priority at any time, but lowering its priority such that it no longer has the highest priority must cause it to immediately yield the CPU.

2.2-) priority donation

- One issue with priority scheduling is "priority inversion". Consider high, medium, and low priority threads H, M, and L, respectively. If H needs to wait for L (for instance, for a lock held by L), and M is on the ready list, then H will never get the CPU because the low priority thread will not get any CPU time. A partial fix for this problem is for H to "donate" its priority to L while L is holding the lock, then recall the donation once L releases (and thus H acquires) the lock.
- Implement priority donation.

2.3-)Implement those functions

Finally, implement the following functions that allow a thread to examine and modify its own priority. Skeletons for these functions are provided in "threads/thread.c".

Function: void thread_set_priority (int new_priority)

Sets the current thread's priority to new_priority. If the current thread no longer has the highest priority, yields.

Function: int thread_get_priority (void)

Returns the current thread's priority. In the presence of priority donation, returns the higher (donated) priority.

3-)Advanced Scheduler

- Implement a multilevel feedback queue scheduler similar to the BSD scheduler to reduce the average response time for running jobs on your system.
- Like the priority scheduler, the advanced scheduler chooses the thread to run based on priorities. However, the advanced scheduler does not do priority donation. Thus, we recommend that you have the priority scheduler working, except possibly for priority donation, before you start work on the advanced scheduler.

… advanced scheduler

- By default, the priority scheduler must be active, but we must be able to choose the 4.4BSD scheduler with the "-mlfqs" kernel option. Passing this option sets thread_mlfqs, declared in "threads/thread.h", to true when the options are parsed by parse_options(), which happens early in main().
- When the 4.4BSD scheduler is enabled, threads no longer directly control their own priorities. The priority argument to thread_create() should be ignored, as well as any calls to thread_set_priority(), and thread_get_priority() should return the thread's current priority as set by the scheduler.

BSD scheduler

- This type of scheduler maintains several queues of ready-to-run threads, where each queue holds threads with a different priority. At any given time, the scheduler chooses a thread from the highest-priority non-empty queue. If the highest-priority queue contains multiple threads, then they run in "round robin" order.
- Niceness :if niceness is low thread wants to get more cpu time else wants to get less cpu time
- Thread priority is dynamically determined by the scheduler using a formula
- priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)
- recent_cpu=... ,load_avg= ...

Fixed-Point Real Arithmetic

- In the formulas above, priority, nice, and ready_threads are integers, but recent_cpu and load_avg are real numbers. Unfortunately, Pintos does not support floating-point arithmetic in the kernel, because it would complicate and slow the kernel.
- Real kernels often have the same limitation, for the same reason.This means that calculations on real quantities must be simulated using integers.
- The fundamental idea is to treat the rightmost bits of an integer as representing a fraction.

How much code will I need to write?

- devices/timer.c | 42 +++++-
- threads/fixed-point.h | 120 ++++++++++++++++++
- threads/synch.c | 88 ++++++++++++-
- threads/thread.c | 196 +++++++++++++++++++
  +++++++----
- threads/thread.h | 23 +++
- 5 files changed, 440 insertions(+), 29 deletions(-)

(lines of code)