Assignment 4

PintOS

Instructions:

1. This project is more of understanding the execution and flow of an operating system. Spend enough time during the learning phase.
2. Copying from any sources is strictly discouraged.
3. The header files mentioned are sufficient to implement all the required functionality. You may not require any additional headers.

Part 1: Reading Assignment

Objective: To understand the internals of PINTOS

Pintos is an instructional operating system, complete with documentation and readymade, modular projects that introduce students to the principles of multi programming, scheduling, virtual memory, and filesystems. By allowing students to run their work product on actual hardware, while simultaneously benefiting from debugging and dynamic analysis tools provided in simulated and emulated environments, Pintos increases student engagement. Unlike tailored versions of commercial or open source OS such as Linux, Pintos is designed from the ground up from an educational perspective. It has been used by multiple institutions for a number of years and is available for wider use.

Understanding PINTOS basics from reading material. Building PINTOS executable from source code and run it on virtual machine. Understanding basics of utilities like GNU MAKE,etc Get familiar with source code – files and data structures.

Link : Introduction :http://www.stanford.edu/class/cs140/projects/pintos/pintos_1.html#SEC1
Source code: http://www.stanford.edu/class/cs140/projects/pintos/pintos.tar.gz

Part 2: Add file to Pintos kernel Objective: Learn how to add a file to pintos kernel.

Setup and Installation: Follow the link given below to install pintos on your system.
http://web.stanford.edu/class/cs140/projects/pintos/pintos_12.html

For building and running and running pintos follow sections 1.1.2 and 1.1.3 of the following link. Use qemu or vmplayer to run pintos.
http://web.stanford.edu/class/cs140/projects/pintos/pintos_1.html#SEC1

Create a hello.c file which consists of main function and prints “Hello Pintos” message. Learn how to run this file inside pintos. For this you can look into the pintos/src/tests/threads. There are several test files present in this directory. Also there is tests.c and tests.h file present. See these sources in order to add and execute your hello.c file.

Part 3: Implementation of thread sleep mechanism

Objective: Understand the thread sleep mechanism of pintos and provide an efficient solution to circumvent busy waiting.

Scope: 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.

Function: void timer_sleep (int64_t ticks) { ... } Suspends execution of the calling thread until time has advanced by at least x timer ticks. Unless the system is otherwise idle, the thread need not wake up after exactly x ticks. Just put it on the ready queue after they have waited for the right amount of time. timer_sleep() is useful for threads that operate in realtime, e.g. for blinking the cursor once per second.

The argument to timer_sleep() is expressed in timer ticks, not in milliseconds or any other unit. There are TIMER_FREQ timer ticks per second, where TIMER_FREQ is a macro defined in devices/timer.h. The default value is 100. We don't recommend changing this value, because any change is likely to cause many of the tests to fail. Separate functions timer_msleep(), timer_usleep(), and timer_nsleep() do exist for sleeping a specific number of milliseconds, microseconds, or nanoseconds, respectively, but these will call timer_sleep() automatically when necessary. You do not need to modify them.

All of the tests and related files are in pintos/src/tests.

Please refer section 1.2.1 of the link below for verifying your output of test cases against expected output. http://web.stanford.edu/class/cs140/projects/pintos/pintos_1.html

For this objective, following test cases should pass. Each of the following test case verifies your implementation of timer_sleep. More can be found here:

1. alarm-single
2. alarm-zero
3. alarm-negative
4. alarm-simultaneous

Part 4: Implementation of Priority Scheduling

Objective 1: Implement priority scheduling for threads.

Scope:

- Implement priority scheduling for thread so that a higher priority thread is scheduled before lower priority threads. In the current scenario, priority value is taken but it has not been used anywhere for thread scheduling (See thread structure in src/threads/thread.h).
- During the process of execution, a thread can lower or higher its 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 making way for the higher priority thread in the ready queue.
- If priority of all threads are same, implement scheduling in round robin fashion.

Priority range:

- PRI_MIN – 0 – lowest
- PRI_DEFAULT – 31 – default
- PRI_MAX – 63 – highest

For implementing above changes, please look into the functions defined in thread.c and the thread structure defined in thread.h

Test cases:

- alarm-priority
- priority-change
- priority-fifo

Objective 2: Understand thread synchronization using semaphores in pintos.

Scope:

When threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be awakened first. This behavior is not followed in current implementation as thread priority scheduling is not implemented. Understand the necessary changes needed for accommodating these changes and implement it.

For implementing above changes, look into sync.h, thread.h, thread.c

Test cases

1. priority-sema

Upload format: You will need to submit the entire source code and a report pdf mentioning the approach you have used for meeting each of the objectives mentioned above. The detailed format will be shared later.