Lab2: Threads

EE 5173 Operating System

TA: Hsueh-Yi, Chen. Advisor: Farn, Wang.

2025/4/9

Lab2 Description

- Minimally functional thread system has been provided. Your job is to extend the functionality of this system to gain a better understanding of synchronization problems.
- Highly recommend to read the reference. (We just selected some content here.)

  https://web.stanford.edu/class/cs140/projects/pintos/pintos_2.html#SEC15

Strongly Recommended To Read

- A.1 Loading
- A.2 Threads
- A.3 Synchronization
- A.4 Interrupt Handling
- A.5 Memory Allocation

https://web.stanford.edu/class/cs140/projects/pintos/pintos_6.html#SEC91

Background

Understanding Threads

- When a thread is created, you are creating a new context to be scheduled. You provide a function to be run in this context as an argument to **thread_create().**

```c
/* Creates a new kernel thread named NAME with the given initial
   PRIORITY, which executes FUNCTION passing AUX as the argument,
   and adds it to the ready queue.  Returns the thread identifier
   for the new thread, or TID_ERROR if creation fails.

   If thread_start() has been called, then the new thread may be
   scheduled before thread_create() returns.  It could even exit
   before thread_create() returns.  Contrariwise, the original
   thread may run for any amount of time before the new thread is
   scheduled.  Use a semaphore or some other form of
   synchronization if you need to ensure ordering.

   The code provided sets the new thread's `priority' member to
   PRIORITY, but no actual priority scheduling is implemented.
   Priority scheduling is the goal of Problem 1-3. */
tid_t
thread_create (const char *name, int priority,
               thread_func *function, void *aux)
{
  struct thread *t;
```

Understanding Threads

- At any given time, exactly one thread runs and the rest, if any, become inactive.
- The scheduler decides which thread to run next. (If no thread is ready to run at any given time, then the special "idle" thread, implemented in idle(), runs.)
- Synchronization primitives can force context switches when one thread needs to wait for another thread to do something.
- The mechanics of a context switch are in threads/switch.S

Understanding Threads

- It is recommended to use a GDB debugger and printf to dynamically trace through a context switch.
- Set a breakpoint on **schedule()** to start out, and single step from there.

Source Files

- threads
- devices
- lib

https://web.stanford.edu/class/cs140/projects/pintos/pintos_2.html#SEC15

Synchronization

1. Synchronization in Pintos

- Proper synchronization is crucial for solving concurrency problems.
- Disable interrupts to solve synchronization problems is tempting but should be avoided.

2. Disabling Interrupts for Synchronization

- Disabling interrupts is useful only for coordinating data between kernel threads and interrupt handlers.
- Interrupt handlers cannot sleep, so they can't acquire locks.
- Data shared between kernel threads and interrupt handlers must be protected by turning off interrupts.

Synchronization

3. Only requires accessing a little bit of thread state from interrupt handlers

- For the alarm clock, the timer interrupt needs to wake up sleeping threads
- In the advanced scheduler, the timer interrupt needs to access a few global and per-thread variables

Disabling interrupts ensures that the timer interrupt does not interfere.

Synchronization

4. Here are some reminders about disabling interrupts

- When disabling interrupts, minimize the code section to avoid losing important events like timer ticks or input.
- Excessive interrupt disabling increases latency, which can make the system feel sluggish.

5. Others

- The synchronization primitives themselves in synch.c are implemented by disabling interrupts. You may need to increase the amount of code that runs with interrupts disabled here, but you should still try to keep it to a minimum.
- Disabling interrupts can be useful for debugging, if you want to make sure that a section of code is not interrupted

LAB

Alarm Clock

- Reimplement the timer_sleep() function (defined in devices/timer.c) to avoid busy waiting.
- Function: void timer_sleep (int64_t ticks)
- Hint: We need a place to store (queue) information about sleeping threads and wake them up by moving them to the ready queue

```
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

Priority Scheduling

- 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**.
- PRI_MIN (0) to PRI_MAX (63), from lowest to highest

Priority Scheduling (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.

Priority Scheduling (priority donation)

- You will need to account for all different situations in which priority donation is required. (Test Driven)
- Be sure to handle multiple donations, in which multiple priorities are donated to a single thread. You must also handle nested donation: if H is waiting on a lock that M holds and M is waiting on a lock that L holds, then both M and L should be boosted to H's priority. (Recursive call)

Priority Scheduling (priority donation)

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.

~~**Threads modify other threads' priorities.**~~

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. BSD Scheduler
- Recommend complete priority scheduler first before start work on it (Advanced scheduler choose the thread to run based on priorities).
- The BSD scheduler is enabled, threads no longer directly control their own priorities

https://web.stanford.edu/class/cs140/projects/pintos/pintos_7.html#SEC131

Advanced Scheduler

- priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)
- recent_cpu = (2*load_avg)/(2*load_avg + 1) * recent_cpu + nice
- load_avg = (59/60)*load_avg + (1/60)*ready_threads

Advanced Scheduler

```
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(-)
```

- You can add a file called “fixed-point.h”
- Recommandation: inline static function

Convert n to fixed point:	n * f
Convert x to integer (rounding toward zero):	x / f
Convert x to integer (rounding to nearest):	(x + f / 2) / f if x >= 0, (x - f / 2) / f if x <= 0.
Add x and y:	x + y
Subtract y from x:	x - y
Add x and n:	x + n * f
Subtract n from x:	x - n * f
Multiply x by y:	((int64_t) x) * y / f
Multiply x by n:	x * n
Divide x by y:	((int64_t) x) * f / y
Divide x by n:	x / n

Grading Policy

- Total - 100%
  - Test Score – 40%
  - Tech Report – 60%

Deadline 2025/4/30 23:59

- Filename (-3%)
- Late submission (-10%), over 1 weeks (-20%) over 2 weeks (Immediate 0 points)
- Plagiarism or All/Most LLM (Immediate 0 points)
- Bonus (Other implementations: Max 10)

Report

- Content format: 12pt front,16pt row height, and align to the left.
- Caption format: 18pt and Bold font.
- Figure: center with single line row height.
- Upload with the file structure format :

```
G[team number]_2.zip (e.g. G01_2.pdf)
|── G[team number]_2.pdf
|── pintos
|    ├ devices
|    ├ …
|
```

Requirement in tech report

- **Please follow the design document questions (https://web.stanford.edu/class/cs140/projects/pintos/threads.tmpl)**
- **Test Results**
    - Provide a snapshot of test results and your **make check** score.
    - If any tests fail, briefly explain the possible reasons.

Thanks for your attention !