Pintos: Threads Project

Vijay Kumar
CS 3204 TA

Introduction to Pintos

- Simple OS for the 80x86 architecture
- Capable of running on real hardware
- We use bochs, qemu to run Pintos
- Supports kernel threads, user programs and file system
- In the projects, strengthen support for these + implement support for VM
Development Environment

- Use the machines in McB 124 for the projects
- Alternately, log on to one of the machines in McB 124 remotely using SSH
  
  `ssh -X yourlogin@rlogin.cs.vt.edu`
  
  or
  
  `ssh -Y yourlogin@rlogin.cs.vt.edu` (for trusted X11 forwarding)

- Use CVS
  - for managing and merging code written by the team members
  - keeping track of multiple versions of files

CVS Setup

- Start by choosing a code keeper for your group
- Keeper creates repository on 'fortran.cslab'
- Summary of commands to setup CVS
  
  `ssh fortran`
  
  `cd /home/cs3204`
  
  `mkdir Proj-keeper_pid`
  
  `setfacl --set u::rwx,g::---,o::--- Proj-keeper_pid`
  
  # for all other group members do:

  `setfacl -m u:member-pid:rwx Proj-keeper_pid`
  
  `setfacl -d --set u::rwx,g::---,o::--- Proj-keeper_pid`
  
  # for all group members, including the keeper, do:

  `setfacl -d -m u:member_pid:rwx Proj-keeper_pid`

  `cvs -d /home/cs3204/Proj-keeper_pid init`

  `cd /home/courses/cs3204/pintos/pintos`

  `cvs -d /home/cs3204/Proj-keeper_pid import -m "Imported sources" pintos foobar start`
Using CVS

import

Development machine in McB124

checkout

commit

fortran.cslab containing Repository

Other useful CVS commands
- diff
- add
- remove
- update

Getting started with Pintos

- Set env variable CVS_RSH to /usr/bin/ssh
  
  ```
  export CVS_RSH=/usr/bin/ssh
  ```

- Check out a copy of the repository to directory ‘dir’
  
  ```
  cvs -d :ext:your_pid@fortran:/home/cs3204/Proj-keeper_pid checkout -d dir pintos
  ```

- Add ~cs3204/bin to path
  
  ```
  export PATH=~cs3204/bin:$PATH
  ```

- Build pintos
  
  ```
  cd dir/src/threads
  make
  cd build
  pintos run alarm-multiple
  ```
Project 1 Overview

- Extend the functionality of a minimally functional thread system
- Implement
  - Alarm Clock
  - Priority Scheduling
  - Advanced Scheduler

Pintos Thread System

```c
struct thread
{
    tid_t tid;               /* Thread identifier. */
    enum thread_status status; /* Thread state. */
    char name[16];           /* Name (for debugging purposes). */
    uint8_t *stack;          /* Saved stack pointer. */
    int priority;            /* Priority. */
    /* Shared between thread.c and synch.c. */
    struct list_elem elem;   /* List element. */
#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;       /* Page directory. */
#endif
    unsigned magic;          /* Detects stack overflow. */
};
```
Pintos Thread System (contd…)

- Read threads/thread.c and threads/synch.c to understand
  - How the switching between threads occur
  - How the scheduler works
  - How the various synchronizations primitives work

Alarm Clock

- Reimplement timer_sleep() in devices/timer.c without busy waiting
  /* Suspends execution for approximately TICKS timer ticks. */
  void timer_sleep (int64_t ticks){
    int64_t start = timer_ticks ();
    ASSERT (intr_get_level () == INTR_ON);
    while (timer_elapsed (start) < ticks)
      thread_yield ();
  }

- Implementation details
  - Remove thread from ready list and put it back after sufficient ticks have elapsed
Priority Scheduler

- Ready thread with highest priority gets the processor
- When a thread is added to the ready list that has a higher priority than the currently running thread, immediately yield the processor to the new thread
- When threads are waiting for a lock, semaphore or a condition variable, the highest priority waiting thread should be woken up first
- Implementation details
  - compare priority of the thread being added to the ready list with that of the running thread
  - select next thread to run based on priorities
  - compare priorities of waiting threads when releasing locks, semaphores, condition variables

Priority Inversion

- Priority scheduling leads to priority inversion
- Consider the following example where
  \[
  \text{prio}(H) > \text{prio}(M) > \text{prio}(L)
  \]
  - H needs a lock currently held by L
  - M that was already on the ready list gets the processor before L
  - H indirectly waits for M
Priority Donation

- When a high priority thread H waits on a lock held by a lower priority thread L, donate H’s priority to L and recall the donation once L releases the lock
- Implement priority donation for locks
- Handle the cases of multiple donations and nested donations

Multiple Priority Donations: Example

Low Priority thread

```c
lock_acquire (&a);
lock_acquire (&b);

thread_create ("a", PRI_DEFAULT - 1, a_thread_func, &a);
msg ("Main thread should have priority \%d. Actual priority: \%d.", PRI_DEFAULT - 1, thread_get_priority ());

thread_create ("b", PRI_DEFAULT - 2, b_thread_func, &b);
msg ("Main thread should have priority \%d. Actual priority: \%d.", PRI_DEFAULT - 2, thread_get_priority ());
```

Medium Priority thread

```c
static void a_thread_func (void *lock_)
{
    struct lock *lock = lock_;
    lock_acquire (lock);
    msg ("Thread a acquired lock a.");
    lock_release (lock);
    msg ("Thread a finished.");
}
```

High Priority thread

```c
static void b_thread_func (void *lock_)
{
    struct lock *lock = lock_;
    lock_acquire (lock);
    msg ("Thread b acquired lock b.");
    lock_release (lock);
    msg ("Thread b finished.");
}
Nested Priority Donations:

Example

Low Priority thread

```c
lock_acquire (&a);
locks.a = &a;
locks.b = &b;

thread_create ("medium", PRI_DEFAULT - 1, m_thread_func, &locks);
thread_yield ();
msg ("Low thread should have priority %d.  Actual priority: %d.", PRI_DEFAULT - 1, thread_get_priority ());
```

Medium Priority thread

```c
static void m_thread_func (void *locks_)
{
    struct locks *locks = locks_;  
    lock_acquire (locks->b);
    lock_acquire (locks->a);
    msg ("Medium thread should have priority %d.  Actual priority: %d.", PRI_DEFAULT - 2, thread_get_priority ());
}
```

High Priority thread

```c
static void h_thread_func (void *lock_)
{
    struct lock *lock = lock_;  
    lock_acquire (lock);
    msg ("High thread should have priority %d.  Actual priority: %d.", PRI_DEFAULT - 2, thread_get_priority ());
}
```

Advanced Scheduler

- Implement Multi Level Feedback Queue Scheduler
- Priority Donation not needed in the advanced scheduler
- Advanced Scheduler must be chosen only if ‘–mlfqs’ kernel option is specified
- Read section on 4.4 BSD Scheduler in the Pintos manual for detailed information
- Some of the parameters are real numbers and calculations involving them have to be simulated using integers.
Suggested Order

- Alarm Clock
  - easier to implement compared to the other parts
  - other parts not dependent on this
- Priority Scheduler
  - needed for implementing Priority Donation and Advanced Scheduler
- Priority Donation | Advanced Scheduler
  - these two parts are independent of each other
  - can be implemented in any order but only after Priority Scheduler is ready

Debugging your code

- printf, ASSERT, backtraces, gdb
- Running pintos under gdb
  - Invoke pintos with the gdb option
    pintos --gdb -- run testname
  - On another terminal invoke gdb
    gdb kernel.o
  - Issue the command
    target remote localhost:1234
  - All the usual gdb commands can be used: step, next, print, continue, break, clear etc
Tips

- Read the relevant parts of the Pintos manual
- Read the comments in the source files to understand what a function does and what its prerequisites are
- Be careful with synchronization primitives
  - disable interrupts only when absolutely needed
  - use locks, semaphores and condition variables instead
- Beware of the consequences of the changes you introduce
  - might affect the code that gets executed before the boot time messages are displayed, causing the system to reboot or not boot at all
  - use gdb to debug

Tips (contd…)

- Include ASSERTs to make sure that your code works the way you want it to
- Integrate your team’s code often to avoid surprises
- Use gdb to debug
- Make changes to the test files, if needed
- Test using qemu simulator and the –j option with bochs
Grading & Deadline

- Tests – 50%
- Design – 50%
  - data structures, algorithms, synchronization, rationale and coding standards
- Due February 27, 2006 by 11:59pm