

## Pintos: Threads Project

Slides by: Vijay Kumar  
Updated by Godmar Back

## Introduction to Pintos

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

## Development Environment

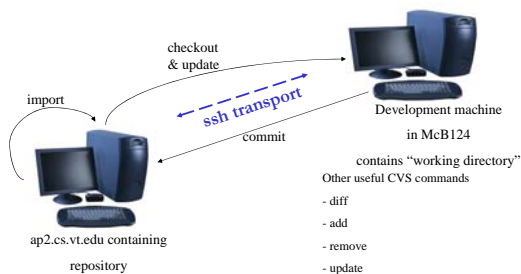
- Use the machines in McB 124 for the projects
- Alternately, log on to the Linux cluster remotely using SSH  
`ssh -Y yourlogin@rlogin.cs.vt.edu` (for trusted X11 forwarding)
- Use CVS
  - for managing and merging code written by the team members
  - keeping track of multiple versions of files

## CVS Setup

- Start by choosing a code keeper for your group
- Keeper creates repository on "ap2"
- Summary of commands to setup CVS

```
ssh ap2
cd /shared/cs3204
mkdir Proj-keeper_pid
setfacl -s 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 /shared/cs3204/Proj-keeper_pid init
cd /home/courses/cs3204/pintos/pintos
cvs -d /shared/cs3204/Proj-keeper_pid import -m "Imported sources" pintos foobar start
```

## Using CVS



## Getting started with Pintos

- Set env variable CVS\_RSH to /usr/bin/ssh

```
export CVS_RSH=/usr/bin/ssh
```

If you don't, it will assume "rsh" which is not a supported service. Connection failures or timeouts will result.
- Check out a copy of the repository to directory 'dir'

```
cvs -d :ext:your_pid@ap2.cs.vt.edu:/shared/cs3204/Proj-keeper_pid checkout -d dir pintos
```
- Add ~/cs3204/bin to path add to .bash\_profile

```
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
    - Including priority inheritance
  - Advanced Scheduler

## Pintos Thread System

```

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. */

  You add more fields here as you need them.
};
  
```

## Pintos Thread System (contd...)

- Read threads/thread.c and threads/synch.c to understand
  - How the switching between threads occur
  - How the provided 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 Based Scheduling

## Priority Inversion

- Strict priority scheduling can lead to a phenomenon called "priority inversion"
- Supplemental reading:
  - What really happened on the Mars Pathfinder? [comp.risks]
- Consider the following example where  $prio(H) > prio(M) > prio(L)$ 
  - H needs a lock currently held by L, so H blocks
  - M that was already on the ready list gets the processor before L
  - H indirectly waits for M
  - (on Path Finder, a watchdog timer noticed that H failed to run for some time, and continuously reset the system)

## 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**

```
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**

```
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

```
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**

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

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

thread_create ("high", PRI_DEFAULT + 2, h_thread_func, &b);
msg ("Low thread should have priority %d. Actual priority: %d.", PRI_DEFAULT + 2, thread_get_priority ());
```

**Medium Priority thread**

```
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

```
static void h_thread_func (void *lock_)
{
  struct lock *lock = lock_;
  lock_acquire (lock);
  ...
}
```


```

## Advanced Scheduler

- Implement Multi Level Feedback Queue Scheduler
- Priority donation not needed in the advanced scheduler – two implementations are not required to coexist
  - Only one is active at a time
- Advanced Scheduler must be chosen only if '-mlfq's' 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.
  - Write a fixed-point layer (header file)

## Typesafe Fixed-Point Layer

```
typedef struct
{
  double re;
  double im;
} complex_t;

static inline complex_t
complex_add(complex_t x, complex_t y)
{
  return (complex_t){x.re + y.re, x.im + y.im};
}

static inline double
complex_real(complex_t x)
{
  return x.re;
}

static inline double
complex_imaginary(complex_t x)
{
  return x.im;
}

static inline double
complex_abs(complex_t x)
{
  return sqrt(x.re * x.re + x.im * x.im);
}
```

## 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  
`pintos-gdb kernel.o`
  - Issue the command  
`debugpintos`
  - All the usual gdb commands can be used: step, next, print, continue, break, clear etc
  - Use the pintos debugging macros described in manual

## 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

## Tips (contd...)

- Include ASSERT's 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 (introduces variability whereas default options run in reproducibility mode)

## Grading & Deadline

- Tests – 50%
  - All group members get the same grade
- Design – 50%
  - data structures, algorithms, synchronization, rationale and coding standards
  - Each group member will submit those individually: you can discuss them in the group, and ask each other questions – but must create write-up individually. Instructions will be posted on the website.
- Due Sep 24, 2007 by 11:59pm

Good Luck!