Introduction to Pintos

Prof. Jin-Soo Kim (jinsoookim@skku.edu)
TAs – Jae-Hun Lee (ljhnimking@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Welcome to Pintos!

What is Pintos?

- An instructional operating system
- Developed by Ben Pfaff @ Stanford U.
- A real, bootable OS for 80x86 architecture
  - Run on a regular IBM-compatible PC or an x86 simulator
- The original structure and form was inspired by the Nachos instructional OS from UC Berkeley (Java-based)
- A few of the sources files are derived from code used in the MIT’s advanced operating systems course
- Written in C language (with minimal assembly code)
What is Bochs?

- Open-source IA-32 emulator
- Simulates a complete Intel x86 computer in software
  - Interprets every instruction from power-up to reboot
  - Has device models for all of the standard PC peripherals: keyboard, mouse, VGA card/monitor, disks, timer, network, ...
  - Supports many different host platforms: x86, PowerPC, Alpha, Sun, and MIPS
- Runs most popular x86 Oses:
  - Windows 95/98/NT/2000/XP/Vista, Linux, BSDs, ...
- Written in C++
- Emulation, not virtualization
Bochs (2)

- Linux + Bochs
  - We will run Pintos using Bochs on Linux
  - Bochs makes it easy to develop and debug Pintos projects
Setting Up (1)

- **Install Linux distribution on your machine**
  - Debian, Fedora, Ubuntu, or whatever you like

- **Install development tools**
  - Including gcc, make, perl, gdb, and so on
  - GCC >= 4.0, binutils >= 2.13

- **Install development libraries, (for Bochs)**
  - Install X windows development libraries, if needed
    - For Debian, install xorg-dev package
  - Install curses development libraries, if needed
    - For Debian, install libncurses5-dev package
  - There could be additional libraries to install
Setting Up (2)

- Install GCC, G++ version 4.1
  - In Ubuntu 10.04,
    - `sudo apt-get install gcc-4.1 g++-4.1`
    - `rm /usr/bin/gcc`
    - `rm /usr/bin/g++`
    - `ln -s /usr/bin/gcc-4.1 /usr/bin/gcc`
    - `ln -s /usr/bin/g++-4.1 /usr/bin/g++`
Setting Up (3)

- **Install GCC, G++ version 4.1**
  - In Ubuntu 10.10,
    - Add to the end of `/etc/apt/sources.list`
      - Deb [http://mirrors.us.kernel.org/ubuntu/](http://mirrors.us.kernel.org/ubuntu/) lucid main universe
    - `sudo apt-get update`
    - `sudo apt-get install gcc-4.1=4.1.2-27ubuntu1 gcc-4.1-base=4.1.2-27ubuntu1 cpp-4.1=4.1.2-27ubuntu1 g++-4.1`
    - `rm /usr/bin/gcc`
    - `rm /usr/bin/g++`
    - `ln -s /usr/bin/gcc-4.1 /usr/bin/gcc`
    - `ln -s /usr/bin/g++-4.1 /usr/bin/g++`
Setting Up (4)

- **Install GCC, G++ version 4.1**
  - In Ubuntu 11.04,
    - `wget ftp://ftp.nluug.nl/mirror/languages/gcc/releases/gcc-4.1.2/gcc-core-4.1.2.tar.bz2`
    - `wget ftp://ftp.nluug.nl/mirror/languages/gcc/releases/gcc-4.1.2/gcc-g++-4.1.2.tar.bz2`
    - `tar jxvf gcc-core-4.1.2.tar.bz2`
    - `tar jxvf gcc-g++-4.1.2.tar.bz2`
    - `<download patch from pastebin>`
    - `cd gcc-4.1.2`
    - `patch -p1 < ./FKkiciZz.txt`
    - `./configure --prefix=/usr/local/gcc-4.1 --program-suffix=-4.1`
    - `make`
    - `make install`
    - `setup link`
## Setting Up (5)

### Install Pintos

- Download the Pintos package (pintos.tar.gz)
  - Use this version only

- Untar Pintos
  
  ```
  $ tar xvzf pintos.tar.gz
  ```

- Build Pintos
  
  ```
  $ cd pintos/src/threads
  $ make
  ```
  - This will create the kernel image (kernel.bin) and the final OS disk image (os.dsk = loader.bin + kernel.bin) in .build
### Setting Up (6)

#### Install Bochs

- You need Bochs to run Pintos
- Get the source code from [http://bochs.sourceforge.net](http://bochs.sourceforge.net)
  - Make sure you are downloading v2.2.6 (bochs-2.2.6.tar.gz)
  - You don’t have to untar the source code

- Install Bochs
  - Must patch the Bochs source code for Pintos (Patches are available in pintos/src/misc)
  - Use the installation script provided by Pintos (pintos/src/misc/bochs-2.2.6-build.sh)
  - The script will untar, patch, configure, compile, and install Bochs
  - You need to be a superuser (root) to install Bochs in the system directory (e.g., /usr/local)
Setting Up (7)

- Install Bochs (cont’d)
  - Running the script:

```
file(F) edit(E) view(V) print(T) save(S) exit(Q)
gkm2164@ubuntu:~$ cd OSProject/
gkm2164@ubuntu:~/OSProject$ cd pintos/src/misc
gkm2164@ubuntu:~/OSProject/pintos/src/misc$ ls
bochs-2.2.6-big-endian.patch  bochs-2.2.6-paranoia.patch
bochs-2.2.6-build.sh          bochs-2.2.6-solaris-link.patch
bochs-2.2.6-gdbstub-ENN.patch bochs-2.2.6-solaris-tty.patch
bochs-2.2.6-jitter.patch      bochs-2.2.6-triple-fault.patch
bochs-2.2.6-ms-extensions.patch gcc-3.3.6-cross-howto
bochs-2.2.6-page-fault-segv.patch gdb-macros

gkm2164@ubuntu:~/OSProject/pintos/src/misc$ ./bochs-2.2.6-build.sh
usage: env SRCDIR=<srcdir> PINTOSDIR=<srcdir> DSTDIR=<dstdir> sh ./bochs-2.2.6-build.sh
where <srcdir> contains bochs-2.2.6.tar.gz
and <pintosdir> is the root of the pintos source tree
and <dstdir> is the installation prefix (e.g. /usr/local)
gkm2164@ubuntu:~/OSProject/pintos/src/misc$ sudo env SRCDIR=/home/gkm2164/ PINTOSDIR=/home/gkm2164/OSProject/pintos DSTDIR=/usr/local sh ./bochs-2.2.6-build.sh
[sudo] password for gkm2164:
gkm2164@ubuntu:~/OSProject/pintos/src/misc$ 
```
Setting Up (8)

- Install Bochs at Ubuntu 10.04
  - `sudo apt-get install`
    - `patch`
    - `diff`
    - `g++`
    - `xorg-dev`
    - `ncurses-dev`
Setting Up (9)

- Test Bochs

```sh
$ bochs ; Put $DSTDIR/bin into your PATH
```
Setting Up (10)

- Setting pintos-gdb

```bash
$ vim pintos/src/utils/pintos-gdb
```

```bash
#!/bin/sh

# Path to GDB macros file. Customize for your site.
GDBMACROS=/home/gkm2164/OSProject/pintos/src/misc/gdb-macros

# Choose correct GDB.
if command -v i386-elf-gdb >/dev/null 2>&1; then
    GDB=i386-elf-gdb
else
    GDB=gdb
fi

# Run GDB.
if test -f "$GDBMACROS"; then
    exec $GDB -x "$GDBMACROS" "$@
else
    echo "*** $GDBMACROS does not exist ***"
    echo "*** Pintos GDB macros will not be available ***"
    exec $GDB "$@
fi
```

Your credit will become inactive in 30 days. This is an automated email.

- Jin-Soo Kim (jinsookim@skku.edu)
### Setting Up (11)

#### Run Pintos

$ cd pintos/src/threads

$ ./utils/pintos run alarm-multiple

Execution of 'alarm-multiple' complete.
A Tour of Pintos (1)

- Projects
  - Project 1: Threads
    - pintos/src/threads
  - Project 2: User programs
    - pintos/src/userprog
  - Project 3: Virtual memory
    - pintos/src/vm
  - Project 4: File system
    - pintos/src/filesys

- Use “make” command in each of project directories
A Tour of Pintos (2)

- Interesting files in the ./build directory
  - kernel.o:
    - The object file for the entire kernel
    - Used for debugging
  - kernel.bin:
    - The memory image of the kernel
  - loader.bin:
    - The memory image of the kernel loader (512 bytes)
    - Reads the kernel from disk into memory and starts it up
  - os.dsk:
    - Disk image for the kernel (loader.bin + kernel.bin)
    - Used as a “virtual disk” by the simulator
A Tour of Pintos (3)

- Running Pintos
  - Add "pintos/src/utils" to $PATH and run "pintos"
    
    $ export PATH="~/pintos/src/utils:$PATH"
    
    $ pintos [option] -- [argument]
  
  - Option
    - Configure the simulator or the virtual hardware
  
  - Argument
    - Each argument is passed to the Pintos kernel verbatim
    - 'pintos run alarm-multiple' instructs the kernel to run alarm-multiple
  
  - Pintos script
    - Parse command line, find disks, prepare arguments, run the simulator (Bochs)
A Tour of Pintos (4)

- Project testing
  
  $ make check
  
  $ make grade

FAIL tests/threads/alarm-single
FAIL tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
FAIL tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
FAIL tests/threads/priority-change
FAIL tests/threads/priority-donate-one
FAIL tests/threads/priority-donate-multiple
FAIL tests/threads/priority-donate-multiple2
FAIL tests/threads/priority-donate-nest
FAIL tests/threads/priority-donate-sema
FAIL tests/threads/priority-donate-lower
FAIL tests/threads/priority-fifo
FAIL tests/threads/priority-preempt
FAIL tests/threads/priority-sema
FAIL tests/threads/priority-condvar
FAIL tests/threads/priority-donate-chain
FAIL tests/threads/mlfqs-load-1
FAIL tests/threads/mlfqs-load-60
FAIL tests/threads/mlfqs-load-avg
FAIL tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
FAIL tests/threads/mlfqs-nice-2
FAIL tests/threads/mlfqs-nice-10
FAIL tests/threads/mlfqs-block

22 of 27 tests failed.
make: *** [check] Error 1
A Tour of Pintos (5)

- **Useful tools**
  - **gdb**: The GNU project debugger
    - Allows to see what’s going on inside another program while it executes
    - Refer to Appendix E.5: GDB
  - **Tags**
    - An index to the functions and global variables
    - Powerful when it is combined with vi editor
    - Refer to Appendix F.1: Tags
  - **CVS**: Version-control system
    - Useful for version controls and concurrent development
    - Refer to Appendix F.3: CVS
    - But we use Subversion!!
A Tour of Pintos (6)

- **Tips**
  - Read the project specification carefully
  - Before starting your project, read the document template too!
    - It may give you useful tips
  - Study the test cases in pintos/src/tests used by "make check"
    - One C program for each test case (*.c)
    - One Perl script to check whether your implementation is correct or not (*.ck)
    - Study the correct output stored in the perl script
  - Do it incrementally
    - Otherwise, it can be totally messed up
System Startup

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
System Startup (1)

- Overview
  - BIOS
  - Boot loader
  - Kernel initialization
System Startup (2)

- **The BIOS**
  - The CPU initializes itself and then begins to execute an instruction at a fixed location (\texttt{0xffff ffff 0})
  - Those instructions are supplied from ROM and make the CPU jump into the BIOS
  - The BIOS finds a boot device and loads its first sector into memory
    - Starting from physical address \texttt{0x0000 7c00}
    - The first sector contains the Pintos’ loader (threads/loader.S)
  - The BIOS transfers control to the loader
The boot loader

- Enables memory accesses beyond first 1MB
  - For historical reasons, this initialization is required

- Asks the BIOS for the PC’s memory size
  - Again for historical reasons, the function we use can only detect up to 64MB of RAM (This is the limit that Pintos can support)
  - The memory size is stored in the loader and the kernel can read the information after it boots

- Creates a basic page table
  - This page table maps the 64MB at the base (starting at virtual address 0) directly to identical physical address
  - It also maps the same physical memory starting at virtual address LOADER_PHYS_BASE (0xc000 0000)
The boot loader (cont’d)

- Turns on protected mode and paging
  - Interrupts are still disabled

- Loads the kernel from disk
  - Assumptions:
    » The kernel is stored starting from the second sector of the first IDE disk
    » The BIOS has already set up the IDE controller
  - The loader loads the kernel starting at physical address LOADER_KERN_BASE (0x0010 0000)

- Jumps to the kernel entry point
  - main() in src/threads/init.c
  - Set up using the linker script (threads/kernel1.lds.S)
System Startup (5)

- Kernel initialization
  - Clears BSS and get machine’s RAM size
  - Initializes threads system
  - Initializes VGA, serial port, and console
    - To print a startup message to the console
  - Greets user and reading kernel command line
    - “Kernel command line: “
  - Initializes memory system
  - Initializes random number generator and interrupt system
  - Starts thread scheduler and enables interrupts
  - Initializes file system
Project Policies

Jin-Soo Kim (jinsockim@skku.edu)  
Computer Systems Laboratory  
Sungkyunkwan University  
http://csl.skku.edu
Project Schedule

- **Project 0**
  - Warming-up project (2 weeks, ~9/19)

- **Project 1**
  - Threads (3 weeks, ~10/10)

- **Project 2**
  - User programs (4 weeks, ~11/7)

- **Project 3**
  - Virtual memory (4 weeks, ~12/5)

- **This schedule is subject to change**
Team project (except Project 0)

- Two members in a team
- You must work in teams in the “real world”
- Communicate with colleagues (team members)
  - Communication problems are natural
  - It’s a good chance to get to know each other
  - How to divide work among team members?
  - What have you done?
  - What answers you need from others?
  - You must document your work!
  - You should clearly state the contribution of each team member in your project report
    (And this should be agreed upon among team members)
Project Policy (2)

- Working in teams
  - Do not try to merge all the codes developed independently by each team member just before the deadline
  - Often two changes conflict with each other, requiring lots of debugging
  - Instead, integrate your team’s changes early and often.
  - Understand your requirement first. And then design well before the actual implementation
    → This will save your time considerably.
  - Refer to 2.1.4: Development Suggestions
Project Policy (3)

- Late policy
  - Each team has 5 “slip” days
  - 20% off per day after slip days exhausted
  - No advantage on remaining slip days
  - Save your slip days for rainy days, as the project is getting harder and harder

- For Project 0, there is no slip day.
Project Policy (4)

- Cheating policy

  - “Copying all or part of another person’s work, or using reference material not specifically allowed, are forms of cheating and will not be tolerated.”

  - For a student involved in an incident of cheating, the following policy will apply:
    - You will get 0 points in the particular project and the final grade will be lowered by one grade (e.g., B+ → B)
    - For serious offenses, you will get an F grade and this will be notified to the department chair

  - Share useful information: helping others use systems or tools, helping them with high-level designs or debug their code is NOT cheating!

  - To check cheating, TA see subversion logs & ask
Project Grading (1)

- Presentations in the Lab session (bonus)
- Functionality (70%)
  
  ```
  $ make check
  $ make grade
  ```

- Design & documentation (30%)
  
  - Source code
    - variable name, function name, comments
  
  - Design document
    - Data structure, Algorithm, Synchronization, Rationale
  
  - Refer to Appendix D: Project Documentation

- Demos & oral tests
Project Grading (2)

- Source code
  - comments

```c
NTSTATUS
FatCommonCreate (
    __inout PIRP_CONTEXT IrpContext,
    __inout PIRP Irp
);

/*++
Routine Description:

This is the common routine for creating/opening a file called by both the fsd and fsp threads.

Arguments:

  Irp - Supplies the Irp to process

Return Value:

  NTSTATUS - the return status for the operation

*/
```

```c
DebugTrace( 0, Dbg, "-EaLength
.
Create.EaLength ");

//
// This is here because the Win32 layer can't avoid sending me double
// beginning backslashes.
//

if ((Irsp->FileObject->FileName.Length > sizeof(WCHAR))
    && (Irsp->FileObject->FileName.Buffer[1] == L'\')
    && (Irsp->FileObject->FileName.Buffer[0] == L'\')) {

    Irsp->FileObject->FileName.Length = sizeof(WCHAR);
    RtIMoveMemory( &Irsp->FileObject->FileName.Buffer[0],
                   &Irsp->FileObject->FileName.Buffer[1],
                   Irsp->FileObject->FileName.Length );

    //
    // If there are still two beginning backslashes, the name is bogus.
    //

    if ((Irsp->FileObject->FileName.Length > sizeof(WCHAR))
        &&
```
**Project Grading (2)**

- **Demos & oral tests**
  - Usually done in the next week of the due date
  - Each team should meet the TA offline
  - All team members should be present
  - You may bring your notebook as there could be a problem in running your solution in the TA’s machine
  - TA check subversion logs
  - You should be able to answer any questions on
    - Basic system architecture
    - Design decisions
    - Implementation details
    - ...
Project Grading (3)

- **Individual score**
  - \( f(\text{overall project score, individual contribution}) \)
  - You should specify the followings in your report:
    - The percentage of contribution for each team member
    - The detailed list of specific tasks done by each team member
  - The report should be signed by all team members as a token of acceptance.
  - During demos & oral tests, the percentage of contribution can be adjusted by the instructor.
  - As long as your contribution is \( \geq 40\% \), you will get the full project score.
Project 0: Warming Up
Project 0 (1)

- Set up your own project environment
  - Install Linux
  - Install all the required tools
  - Install Pintos
  - Capture the screen shot of working Pintos
    $ pintos run alarm-multiple
Project 0 (2)

- Add a new test code: print-name
  - Add a new kernel function which prints your name in ASCII text format
  - To run the new function, add a new command "print-name"
    - The following command should run your new function
      $ pintos run print-name
  - Work in the pintos/src/threads and pintos/src/tests/threads directories
  - Be creative when you print your name!
  - Capture the screen shot
Example:
Project 0 (4)

- **Documentation**
  - Specification of your environment
    - Linux distributions, versions of gcc, etc.
  - A screen shot of “alarm-multiple”
  - A screen shot of “print-name”
  - Detailed explanation of how the “print-name” is handled and your name is printed by the kernel

- **Due:**
  - Sep. 19, 11:59PM (NO slip day)
  - Submit via e-mail to sse3044@csl.skku.edu
  - Note: This is an individual project