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 can run Pintos using Bochs on Linux
  - Bochs makes it easy to develop and debug Pintos projects
Setting Up with Bochs (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 with Bochs (2)

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

- **Install GCC, G++ version 4.1**
  - In Ubuntu 10.10 or later
    - Install GCC
      » [http://csl.skku.edu/SSE3044F12/GCC](http://csl.skku.edu/SSE3044F12/GCC)
    - Apply GCC-4.1 version in pintos
      » [http://csl.skku.edu/SSE3044F12/GCCPintos](http://csl.skku.edu/SSE3044F12/GCCPintos)
Setting Up with Bochs (4)

- Install GCC, G++ version 4.1
  - Using pre-built version of GCC
    - Reference this page
      » [http://csl.skku.edu/SWE3004S13/SSEGCC](http://csl.skku.edu/SWE3004S13/SSEGCC)
### Setting Up with Bochs (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 with Bochs (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 with Bochs (7)

- **Install Bochs (cont’d)**

  - Running the script:

```
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 with Bochs (8)

- Prerequisite of Bochs
  - sudo apt-get install
    - patch
    - diff(diffutils in Ubuntu 12.10)
    - g++
    - xorg-dev
    - ncurses-dev
Setting Up with Bochs (9)

- Test Bochs

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

- Setting pintos-gdb

```
$ 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
```
Setting Up with Bochs (11)

- Run Pintos

```
$ cd pintos/src/threads
$ ../utils/pintos run alarm-multiple
```

```
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=7, product=140
(alarm-multiple) thread 3: duration=40, iteration=5, product=150
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=160
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
```
What is Qemu?

• Quick EMUlator
• Written by Fabrice Bellard
• Supports the emulation of various architectures
  – IA-32, x86-64, MIPS R4000, Sun, ARM, PowerPC, etc..

Qemu + Linux

• We can run Pintos using Qemu on Linux
• Installation of Qemu is very easy!
Setting Up with Qemu (1)

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

- Install QEMU
  - See http://csl.skku.edu/SSE3044F12/QEMU
Setting Up with Qemu (2)

- **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 with Qemu (3)

- Setting Pintos for QEMU
  - Simulator Setting
    - Check Make.vars at ~/pintos/src/threads
    - ‘Simulator = --qemu’
  - Pintos script setting
    - Also see http://csl.skku.edu/SSE3044F12/QEMU
    - Modify ~/pintos/src/utils/pintos
    - You can use any text editor to modify this
  - Run option
    - You have to use --qemu option for pintos
      - Default simulator is bochs
    - ../utils/pintos --qemu -- run alarm-multiple
What is different?

- **Difference between Bochs and Qemu**
  - “Reproducibility” is important issue for debugging
    - Always same result occurs when you run program in same manner
  - Bochs offers reproducibility
    - Same jitter value causes exactly same result
    - But it also provides real time mode
      » By using –r option
  - Qemu doesn’t offer reproducibility
    - Only real time mode is supported
  - Qemu is faster
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

```
xterm
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
```

$
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

- 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 (0xffff ffff)
  - 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 0x0000 7c00
    - The first sector contains the Pintos’ loader (threads/loader.S)
  - The BIOS transfers control to the loader
System Startup (3)

- 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)
System Startup (4)

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/kernel.1ds.S)
System Startup (5)

- Kernel initialization
  - Clears BSS and gets machine’s RAM size
  - Initializes threads system
  - Initializes VGA, serial port, and console
    - To print a startup message to the console
  - Greets user and reads 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 (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Project Schedule

- **Project 0**
  - Warming-up project (2 weeks, 3/7~3/20)

- **Project 1**
  - Threads (2 weeks, 3/21~4/3)

- **Project 2**
  - User programs (3 weeks, 4/11~5/8)

- **Project 3**
  - Virtual memory (3 weeks, 5/16~6/5)

- **This schedule is subject to change**
Project Policy (1)

- Late policy
  - There are 3 “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 (2)

**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 a penalty in the final grade (down to F)
  - For serious offenses, 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 submission server, analyze detail code & ask
Project Grading (1)

- **Functionality (70%)**
  
  $\text{make check}$
  
  $\text{make grade}$

- **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
)

(argc) (r2)(e)() v(0) b=() t=() d()

/*++

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

/*

DebugTrace( 0, Dbg, "->EaLength = %08lx\n", IrpSp->Parameters .Create.EaLength );

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

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

    IrpSp->FileObject->FileName.Length = sizeof(WCHAR);

    RtlMoveMemory( &IrpSp->FileObject->FileName.Buffer[0],
        &IrpSp->FileObject->FileName.Buffer[1],
        IrpSp->FileObject->FileName.Length );

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

    if ((IrpSp->FileObject->FileName.Length > sizeof(WCHAR)) &&
```

---

SWE3004: Operating Systems | Spring 2013 | Jin-Soo Kim (jinsookim@skku.edu)
Project Grading (3)

- Demos & oral tests
  - Usually done in the next week of the due date
  - Everyone should meet the TA offline
  - 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 0: Warming Up
Project 0 (1)

- Set up your own project environment
  - Install Linux
  - Install all the required tools
  - Install Pintos
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
Project 0 (3)

- Add a new test code: print-name
  - Print format
    - (print-name) Course : SWE3004
    - (print-name) ID : 2010000000
    - (print-name) Name : GilDong Hong
  - Capture screenshot
Project 0 (4)

- Example:

```
Bochs x86 emulator, http://bochs.sourceforge.net/

Bochs BIOS - build: 01/25/06
$Revision: 1.160 $ $Date: 2006/01/25 17:51:49 $
Options: apmbios pcibios eltorito

ata0 master: Generic 1234 ATA-2 Hard-Disk (0 MBytes)

Booting from Hard Disk...
Pilo hdal
Loading........
Kernel command line: run print-name
Pintos booting with 4,096 KB RAM...
383 pages available in kernel pool.
383 pages available in user pool.
Calibrating timer... 204,600 loops/s.
Boot complete.
Executing 'print-name':
(print-name) begin
(print-name) Course : SSE3044
(print-name) ID : 2007310048
(print-name) Name : JongSung Lee
(print-name) end
Execution of 'print-name' complete.
```

CTRL + 3rd button enables mouse  |  HD:0-M NUM  |  CAPS  |  SCRL
**Submission (1)**

- **Documentation**
  - 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
  - File format – PDF format
  - File name – “GDHong_2013123456.pdf”

- **Source code**
  - Tar and gzip your Pintos source codes
    
    ```bash
    $ cd pintos
    $(cd src/threads; make clean)
    $ tar cvzf GDHong_2013123456.tar.gz src
    ```
Submission (2)

- **Due**
  - Mar. 20, 11:59PM (NO slip day)
  - Upload your source code and documentation at sys.skku.edu
  - Good luck!