CS342: Operating Systems Lab

Department of Computer Science and Engineering,
Indian Institute of Technology, Guwahati
North Guwahati, Assam 781 039

Exercise Start

OS Lessons: Aims of the course, Statement of the expectations, Grade determination, Workload expectations
Rating: Easy
Last update: 04 June 2018

This document is written for the students at Indian Institute of Technology, Guwahati (Assam). Students and staff at other universities and institutes may need to adapt and change the arrangements to suit their needs.

Introduction

Welcome to the subject (CS342: OS Labs), and we assure you that you will learn a lot from this subject if you follow the instructions diligently and avoid use of suspicious ways to gain better assessment/grade in the subject. The previous batches of the students have found the subject and the exercises challenging. For best benefits from this subject, you must focus on the learning outcomes and not be too fixated on the grades. Your real payback is the learning.

To support your learning and training, the subject and the exercises are now fully self-paced. All learning is through team activities and team-based works. A student completing this subject with a good level of achievement would have good knowledge and skills as measured on the following benchmarks:

1. Knowledge of the services provided by the modern operating systems to the user programs. The student will understand the nature and purposes of these services together with their limitations and applications.
2. Students will have the ability to understand, interact, modify and organize large parcels of software codes. The students will have an understanding of the need for continuous testing and debugging during the software development phases. Students will also be aware of the advantages of safe software development practices in multiple developer projects.
3. Students will learn and have practical experiences with the tools necessary for software development. The set includes the tools and methodologies for browsing software, baselining the software components, debugging code, reducing vulnerability to programming mistakes.
4. PintOS is an example of a software that has been organized systematically. Try to learn how it has organized directories, files, and functions. The principle of maximum coherence within the files and directories together with the elimination of unnecessary coupling across these storage structures makes it a dream software to learn software engineering paradigms.
5. The PintOS projects provide hands-on learning of test-driven-software-development methodologies. And,
6. Not the least, on completion of this course the students will have experienced the frustrations and surprises of software development exercises. Fortunately the experiences will not be in isolation; but in the supportive company of their classmates.

PintOS

PintOS (Ben Pfaff et al.) [http://pintos-os.org/SIGCSE2009-Pintos.pdf] is a popular software used at many Universities – in India and overseas – to train students in the internal details of the Unix-like operating systems. The primary document that we use for our exercises was originally written at Stanford University [https://web.stanford.edu/class/cs140/projects/pintos/pintos.pdf]. The Pint OS document (we will refer to it as PintDoc) describes a set of four projects to be completed by the student teams under the Stanford model.

The Stanford model does not fit our arrangements, and we have altered the model to fit to our resource constraints and timetable limitations.

The original training model created by Ben Pfaff has four projects: Threads, User Programs, Virtual Memory, and File System. We adopt the model to our situation – we propose weekly sessions by splitting the first two projects into exercises. An IITG student aiming for a double-A grade (AA) must complete an exercise each week. Other students may improve their learning outcomes by progressing on these exercises at a slower pace to match their learning abilities and grade objectives.

The proposed breakup of the exercises (for the year 2018 and maybe later years) is as listed in the table below. Each row defines work for a week or more:

Project	**Exercise**	**Advice**
Start	Form groups, Get login accounts on Progserv, Install PintOS, Setup CVS repositories, Learn CVS commands, Familiarise with PintOS code and PintDoc	PintDoc, Overview
Threads	Timer Alarm	T 01
Threads	Priority Scheduling	T 02
Threads	Advanced Scheduler	T 03
User Programs	Setup Program Stack	UP 01
User Programs	Exit status and Returned values of System Calls	UP 02
User Programs	System calls for file operations	UP 03
User Programs	System calls: `exec()` and `wait()`	UP 04
Virtual Memory	Preliminary setup to level of `User Programs` tests	VM 01 †
Virtual Memory	Any two of: Stack growth, mapped files, Page merge	VM 02 †
Virtual Memory	All test cases	VM 03 †
File Systems	Validating `userprog` test set after Buffer cache + Synchronization	FS 01 †
File Systems	Implementation Section 5.3.2 Indexed and Extensible Files	FS 02 †
File Systems	Implementing Subdirectories and other for all tests pass	FS 03 †
Cases marked by symbol (†) have no additional guide provided for the students.		

To ensure that the students work and progress on the exercises listed above at an even pace we make the following non-negotiable arrangement. **No more than one exercise completion will be recorded in any one week.** That is, no more than one exercise will be assessed each week for a team. [*At IITG, each day of an academic week is counted through an arrangement approved by the Academic Senate. For assessment recording, we shall use days of the week as listed in the academic section document:* http://www.iitg.ernet.in/acad/acadCal/academic_calander.htm]

Grading procedures and criteria

The final subject grades will be determined based on the exercises completed by the student teams and recorded by the tutor(s). A student seeking assessment recording for an exercise must show *pass* for all applicable test cases run by a relevant PintOS command `make check` to a course tutor in a lab session. The subject instructor will randomly reassess some of the recorded exercise completions to ensure compliance with the quality standards. All assessments and reassessments are proposed to occur during the scheduled lab hours.

A team that is not able to convince CS342 instructor that they have worked independently on the assessed exercise will be required to repeat the exercise. This would have a consequence on the team's progress. A team asked to repeat an exercise will miss their chance to show a new assessment to their tutor in the following lab session. Please note that team engaged in academic misconduct may be asked to complete more than one exercises; setting them several weeks behind. Academic cheating should not bring a better grade!

Note: The rows in tables below without a grade in the first column still require a meeting with the team's tutor at scheduled weekly time. If you do not report to tutor, it will shift your reporting arrangements by a week. This may affect the best final grade possible to your team.

Grade	Week in which this exercise will be assessed for the first time	Set of exercises to be completed for the grade
	Week 02	Start – Meet tutor and report readiness to begin work
DD	Week 03	Start, T 01
CD	Week 04	Start, T 01, (T 02 or T 03)
CC	Week 05	Start, T 01, T 02, T 03
	Week 06	Start, T 01, T 02, T 03, UP 01
BC	Week 07	Start, T 01, T 02, T 03, UP 01, UP02
	Week 08	Start, T 01, T 02, T 03, UP 01, UP02, UP 03
BB	Week 09	T 01, T 02, T 03, UP 01, UP 02, UP 03, UP 04 (All together)

After completion of the above exercises, the students may choose to do either of these two projects for which no weekly exercises have been created. These exercises are based on PintDoc.

Grade	Week in which this will be assessed for the first time	Either Virtual Memory project	Or File System project
	Week 10	Above + VM 01	Above + FS 01
AB	Week 11	Above + VM 01, VM 02	Above + FS 01, FS 02
AA	Week 12	Above + VM 01 to VM 03	Above + FS 01 to FS 03
AS	Teams creating documents similar to T 01 thru UP 04 for their completed project exercises. Only teams earning AA are eligible.		

Project teams/groups:

In the first week of the semester, the students must form teams of five students. To avoid conflicts and disagreements about the work arrangements within a team, it would help if all team members I the team have similar final grade ambitions from the subject. Teams of 4 members is acceptable but there will be no grade benefits for the smaller teams.

Each member in a team is committing to work about 8 to 10 hours each week for the subject; roughly, 9*13 = 120 hours over the semester for AA level achievements. Expected efforts for grade BB is around 80 hours.

The work that the teams perform would include reading and understanding various available documents, searching for the information to improve their understanding, planning and design of the proposed solutions for the exercises, developing new PintOS code to implement the designed solutions, writing logbook entries to create record of work done by the team, getting the exercise completion records checked and accepted (assessed) by the team's tutor.

It is important that all members of a team do all activities in this subject together (jointly). Your tutor is required to submit the assessment records to the subject instructor within 24 hours of the scheduled lab session each (academic) week. The assessment records received late (that is, assessment records not received by the 24-hour deadline after the end of the lab session in the week) will be considered late and are recorded as submitted in the following week. Remember, *at most one assessment a week* rule!

The first exercise (labelled start) is set for assessment in week 02 of the semester. From that time on you must report your weekly progress in each lab session. Each team must report the work done during the week to their tutor even if no assessment of a new work for a grade is needed.

This year teams are required to work only on the Department of CSE computer progserv and use CVS (as described in PintDoc) to maintain team's software and document repository. Use of `Git` is not permitted. Nor are the teams allowed to have simulator `Bochs` installed on their computers for the assessed exercises.

A work session is a period of time when member(s) of a team work on CS342 work. After each development/work session the team must commit the status of their code into their CVS repository. In addition to the code, the teams must include two additional documents into the repository as described below:

1. Run PintOS command "`make check`" and save its output as a separate document to record the status of work after the development session. It should have clear recording of the date-time when the test was run.
2. A text document must be included among the files committed to CVS repository that provides the summary description of the work done during the development session. This description must provide:
   A. Start date and time of the session/meeting in "year month date hour minute" format.
   B. Names of the team members present during the work session.
   C. A short text description of the aim/activity/achievements of the session.
   D. Duration of the session and its end time.
   E. Name the files and function affected by the work
   F. Estimated total number of code lines created/modified/removed during the session. Zero lines of affected code is ok if it was a planning or learning session.

Tutor will verify -- each week -- that the last committed records of the team's work into the CVS repository is correct and matches the state of the code development. It will be demonstrated to the tutors by the teams by actually checking out the pintos code from the repository and testing/matching recorded `make check` output.

Weekly recording of each student's Status by Tutors:

Each week, each team has one opportunity to get completion of one exercise recorded. The tutor will record assessments for each member of the team in a spreadsheet with the following columns. This is to be recorded even if the team has not reached a new grade milestone.

A. Roll number of the team member.
B. Name of the team member as in the enrolment records
C. Date of assessment
D. Time of the assessment
E. Exercise completed or working on.
F. Copy of the last output line of the "`make check`" command. (It shows the number of tests passed etc.)

A new grade is only awarded if all expected test cases pass the exercise.

Students must note that Exercise T 01 can be first assessed in week 03. The last assessment can be recorded in week 14. There is only one slack week for a group aiming for AA grade. A group working towards grade BB has four slack weeks. One or more slack weeks are lost if a team is asked to repeat an exercise due to its work not being approved as free from plagiarism during reassessment by a course instructor.

The teams may progress through these weekly exercises at different rates. Each team must take time in fully understand and complete the exercise even if their progress is behind the other groups. Use of other team's solutions lack the learning benefit. And, if noticed would force both teams to repeat the exercise(s). All major cases of plagiarism would be reported to IADC for award of grade F.

Instructor will make random checks of committed codes for completed (and tutor assessed) exercises to ensure that teams are being fair and free from code plagiarism. This is described previously as *reassessment by the course instructor*.

Tasks for the Week 02 (Not assessed)

This phase is designed for the teams to get some familiarity with PintOS projects and the tools needed for the work. A separate document is being prepared to provide an overview of the subject material that students will learn in CS341: Operating Systems and CS346: Compilers.

PintDoc Appendix E: *Debugging Tools* introduces a number of tools and methods to remove errors. A version control tool CVS is described in Appendix F: *Development Tools*. The basic familiarity with these tools is on your agenda for this week. In addition you must familiarize with the subroutine calling conventions as describe in PintDoc Section 3.5 *80x86 Calling Convention*.

You may read more about these calling conventions in document How main() is Executed on Linux. Since we are going to work on PintOS code, it would be best to use PintDoc description when it differs from the Linux practices.

The following function `test_stack()` is provided to you. It is a proxy for function `main()` of a PintOS based User Program. It prints the values of arguments `argc` and `argv` that have been prepared for delivery to a user program that has been loaded into memory for its run on (and by) PintOS kernel. Note this finction has a single pointer as argument that points to an entry in run-time stack of the program where parameters are exchanged between calling and called function. The calling arrangement for function `main()` is similar to other functions in the program.

```
void test_stack(int *t)
{
   int i;
        int argc = t[1];
        char ** argv;

        argv = (char **) t[2];//((char **)*(t+2));
        printf("ARGC:%d  ARGV:%x\n", argc, (unsigned int)argv);
        for (i = 0; i < argc; i++)
               printf("Argv[%d] = %x pointing at %s\n",
                          i, (unsigned int)argv[i], argv[i]);
}
```

You must verify that the function meets the specifications stated in PintDoc.

Printed below is a part of output that was received from a user program test case. Function `test_stack()` was called with the argument prepared (by PintOS kernel) for a call to function `main()` of user program `args-multiple`:

```
Executing 'args-multiple some arguments for you!':
ARGC:5  ARGV:bffffb4
Argv[0] = bffffe7 pointing at args-multiple
Argv[1] = bffffe2 pointing at some
Argv[2] = bffffd8 pointing at arguments
Argv[3] = bffffd4 pointing at for
Argv[4] = bffffcf pointing at you!
```

We will use this function in a later week to verify that we are able to setup stack properly. You may refer to these sites to learn to use some new tools:

- https://pintosiiith.wordpress.com/2012/09/18/using-cscope-and-ctags-to-navigate-pintos-code/
- https://pintosiiith.wordpress.com/2012/09/24/pintos-using-backtrace-utility-for-debugging/
- (IITG students must use the instructions given later to install PintOS) https://pintosiiith.wordpress.com/2012/09/13/install-pintos-with-qemu/

To practice the lessons proposed for this week, work on the following exercise.

1. Write/download a `quicksort` program. Locate each function of the program in a different directory under a suitably named top directory. Also, create a directory for function `test_stack()`.
2. Initiate a version control repository to store your program components.
3. Use a suitably modified version of function `test_stack` to print the parameters for various calls to the functions in your sort program.
4. As you make changes to the files, commit changes into a version repository.
5. Use GDB to probe and control sort program. Verify and debug your program and its execution stacks. Ensure that `test_stack` output is correct.
6. Once you have familiarized yourselves with the tools and you understand working of the program's runtime stack well you are ready to move onto PintOS projects.
7. To do this, you may remove the program directories and repositories. There is no requirement to demonstrate any part of these practice sessions to your tutor.

8. Follow the instructions given in readme link at http://172.16.112.136/cs342/index.php to setup and install your initial PintOS kernel code. Also, follow the instructions in PintDoc Appendix F.3 to setup access arrangements for your group's version control repository.
9. It is now time to browse the document http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html
10. Start familiarizing yourselves with the overall structure and contents of PintDoc. A quick read now (even if it does not make sense) will be useful.
11. Enjoy. You have done enough for Week 01. Report your readiness to begin work to your tutor in Lab session in week 02.

Steps to install Pintos in Bochs emulator.

1. Connect to the server using ssh [example: "`ssh gb@progsrv.cse.iitg.ernet.in`"].

2. Download Pintos from the course website. Extract it in some directory, say for example `$HOME="/home/cse/gb"` . The files will be extracted to the folder "`$HOME/pintos`". Copy the `perl` scripts "`backtrace`", "`pintos`", "`pintos-gdb`", "`pintos-mkdisk`" from "`$HOME/pintos/src/utils`" into a directory in your PATH. For example, I have added `$HOME/bin` to my PATH. So, I copied these scripts into the `$HOME/bin` folder.

3. If `~/bin` is not included in your PATH, you may add it by modifying the "`.bash_profile`" file located in your home directory. To add the path `~/bin` in your PATH, edit your "`.bash_profile`" file and add the line "`$PATH = $PATH:$HOME/bin`" to it.

4. Next, open the script "`pintos-gdb`" (the one in your "`~/bin`" folder) using any text editor. Find the variable "`GDBMACROS`" and set it to point to "`~/pintos/src/misc/gdb-macros`". Now, you have installed `pintos-gdb`.

5. Now, go back to your "`~/pintos/src/utils`" folder and compile the rest of the Pintos utilities by typing "`make`" as shown in the example below.

```
$ cd $HOME/pintos/src/utils
$ make
```

Copy the script "`squish-pty`" into your "`~/bin`" folder, this is where you had copied the scripts mentioned above as well.

6. Next, compile pintos as shown in the example below.

```
$ cd $HOME/pintos/src/threads/
$ make
```

7. Now, open the file "`$HOME/bin/pintos`" (the one you copied into the PATH) in any text editor and edit the following lines as shown in the example below.

Line no.24: replace "`os.dsk`" with "`$HOME/pintos/src/threads/build/os.dsk`"

Line no. 90: `$vga = "none" if !defined vga;`

This is all. Now, you are ready to run pintos. Check your installation by typing in the terminal as shown in the next step.

8. Run pintos by typing the following command.

```
$ pintos run alarm-multiple
```

This will create 5 threads and sleep for some predefined times. You can see all these messages in the terminal. Now you are ready to implement changes and enhance pintos. To exit, press the keys "`Ctrl + z`" or "`Ctrl + c`".

How PintOS Kernel runs under Various Projects

This is a short description of how the code we write interacts with the provided code. Overall, the provided PintOS code is spread over a number of directories under `pintos/src` directory. A full list and brief description of the directories is given in PintDoc on page 2. We give a bit more text here to further familiarize you with the code.

Perl script `pintos` in `~/bin` directory is in-charge of all activities. Files `Makefile` in various directories compile, link and setup binary executables to support script `pintos`. Files `Make.vars` contain the variables that you may sometimes wish to set differently to modify actions of `Makefiles`.

The set of files used to create PintOS kernel differ between projects. The operating system (pintos) is compiled by command `make` in the primary directory of the appropriate project. Thus, command `make` in directory `pintos/src/threads` compiles pintos as needed in project `thread`. This "threads-only" OS is very primitive and lacks most functionalities that the OS in projects/directories `pintos/src/userprog/ pintos/src/filesys/` etc have.

As user programs (usually called process in OS textbooks) are not supported by `PintOS` until the second project (`userprog`), the testing procedures for project `threads` are setup differently from the later projects. Good news is that in this project you can add `printf()` statement anywhere in your program. This will not be possible to do in the early stages of project `user program`.

The compiled and linked executable for `PintOS` kernel is placed in simulated disk `os.dsk`. This is a file on your server (real) computer but is a (simulated) disk on simulator `bochs`. This is the only simulated disk used in project `threads`. Later projects also introduce a second (simulated) disk `fs.dsk`

The test programs during project `threads` are kernel threads whose source codes are in directory `pintos/src/tests/threads/`. These programs generate object codes that are linked into `pintos` kernel and loaded on `os.dsk` as executable `pintos`.

A more complete description is being prepared as a separate document named `overview-threads-userprog` to provide more information about *threads* and *user programs* to students who are yet to cover these topics in the CS341 classes.

The later projects use a separate disk `fs.dsk` for the user programs. Compiled user programs are inserted into this disk and these executables run on operating system kernel `pintos` located in disk `os.dsk`.

Let us now look at a different issue. The text below was prepared for project `userprog` and the information may not be needed for project `thread`. So you ust skip it for your first few exercises.

There are many tests in `make check` suite when used in project `userprog`. Sometimes, we may seek to run a single test. Full test suite takes a long time to run and also produces a large output making it difficult to search information easily. You may run a single test as described in the following example (Caution: some lines shown below have wrapped around to the next lines):

```
[vmm@progsrv userprog]$ cd build/
[vmm@progsrv build]$ pwd
……/merging/src/userprog/build
[vmm@progsrv build]$ pintos -v -k -T 60 --bochs  --fs-disk=2 -p
tests/userprog/sc-bad-sp -a sc-bad-sp -- -q   -f run sc-bad-sp <
/dev/null 2> tests/userprog/sc-bad-sp.errors > tests/userprog/sc-
bad-sp.output
[vmm@progsrv build]$ pintos -v -k -T 60 --bochs  --fs-disk=2 -p
tests/userprog/sc-bad-sp -a sc-bad-sp -- -q   -f run sc-bad-sp <
/dev/null 2> tests/userprog/sc-bad-sp.errors
Copying tests/userprog/sc-bad-sp to scratch partition...
Writing command line to /tmp/oyVKMsYbtY.dsk...
squish-pty bochs -q
========================================================================
                     Bochs x86 Emulator 2.5.1
             Built from SVN snapshot on January 6, 2012
                 Compiled on Oct 10 2012 at 11:12:02
========================================================================
Kernel command line: -q -f extract run sc-bad-sp
Pintos booting with 4,096 kB RAM...
369 pages available in kernel pool.
368 pages available in user pool.
Calibrating timer...  204,600 loops/s.
hd0:0: detected 1,008 sector (504 kB) disk, model "Generic 1234",
serial "BXHD00011"
hd0:1: detected 4,032 sector (1 MB) disk, model "Generic 1234",
serial "BXHD00012"
hd1:0: detected 1,008 sector (504 kB) disk, model "Generic 1234",
serial "BXHD00021"
Formatting file system...done.
Boot complete.
Extracting ustar archive from scratch disk into file system...
Putting 'sc-bad-sp' into the file system...
Erasing ustar archive...
Executing 'sc-bad-sp':
(sc-bad-sp) begin
sc-bad-sp: exit(-1)
Execution of 'sc-bad-sp' complete.
Timer: 300 ticks
Thread: 30 idle ticks, 241 kernel ticks, 32 user ticks
```

```
hd0:0: 0 reads, 0 writes
hd0:1: 61 reads, 198 writes
hd1:0: 96 reads, 2 writes
Console: 872 characters output
Keyboard: 0 keys pressed
Exception: 0 page faults
Powering off...
[vmm@progsrv build]$
```

You may ask the question: where from I got the following command line?

```
pintos -v -k -T 60 --bochs  --fs-disk=2 -p tests/userprog/sc-bad-sp
-a sc-bad-sp -- -q   -f run sc-bad-sp < /dev/null 2>
tests/userprog/sc-bad-sp.errors > tests/userprog/sc-bad-sp.output
```

The answer is that I copied it from the output of command `make check`. The command runs a series of tests. Each test compiles a test program written in C. places its executable code in disk `fs.dsk` so that kernel `pintos` can transfer it into memory and run the program.

If you run this command you will notice, pintos run does not produce any output. This is so because the output is redirected in this command to a file on your physical computer. We can remove the redirection to display the output on our screen.

Further, notice that my current directory is `~/merging/src/userprog/build`

You are also warned that this (current) directory is deleted each time one runs command `make` or `make clean`. Therefore, one needs to change directory after each run of command `make`.

Typical arrangement for a test is as follows:

1. Test program written in C is located in subdirectories under directory `pintos/src/tests/` matching the project name.
2. The expected output of these test programs is in the script files with extension `.ck`. For example, the desired output for test program `userprog/halt.c` is in `userprog/halt.ck`.

Enjoy test-driven development of PintOS projects.

You may also like to read (or at least quickly view) Section 3.1.2 *Using the File System* to learn more about `fs.dsk`. It explains how you can interact with the contents of this file/disk from your server computer.

Where can you find your program output from a test, the expected output of the test, and the C program defining the test?

Following lines are from output placed on screen by command `make check` under project `filesys` (when doing this test by running command `make check` my current directory was `~/pintos/src/filesys/`)

```
pass tests/filesys/base/syn-write
FAIL tests/filesys/extended/dir-empty-name
FAIL tests/filesys/extended/dir-mk-tree
```

My first thought is to determine what the output of the first failed test is? I can locate it using command (the part of the command that is not underscored is from the output of `make check` listed above informing me of a failed test):

```
$more build/tests/filesys/extended/dir-empty-name.output
```

My next question is where do I find the expected output for the test? The best place for this information is:

```
$more ../tests/filesys/extended/dir-empty-name.ck
```

(This is actually a script but gives good information to determine what output is needed for a pass.)

Finally, to see what the test program was, use the following command:

```
$ more ../tests/filesys/extended/dir-empty-name.c
```

Contributing Authors:

Vishv Malhotra, Gautam Barua, Rashmi Dutta Barua, Arnab Sarkar