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: 20 February 2017

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 past 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 from the learning, and not from the grade received.

To support your learning and training, the subject and the exercises are now fully self-paced. All learning is through group activities and team-based work. 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 uses.
2. Students will have ability to understand, interact, modify and organize a large piece of software product. The students will have understanding of the need for continuous testing and debugging during the software development phases. They 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 mistakes. And,
4. Not the least, students will have experienced the frustrations and surprises of the software development exercises. Fortunately, not 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 erudite internal details of the Unix-like operating systems. The primary document that we use in this subject was originally written at the 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 must adapt the model to fit into our resource constraints and timetable limitations. Overly hard training regime diverts the students from learning to mark-scoring mode.

The original training model proposed by Ben Pfaff has four projects: Threads, User Programs, Virtual Memory, and File System. We adapt the model to our situation – we propose weekly sessions by splitting the 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 suit their learning objectives.

The proposed breakup of the exercises (for year 2017 and may be later years) is as listed in the table below:

Project	**Exercise**	**Advice**
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	FS 01 ††
File Systems	Validating `base` test set	FS 02 ††
File Systems	Validating all test sets	FS 03 ††
Cases marked by symbol (†) have no additional guide provided for the students. The cases marked by †† may have their relevant guides available by mid-September 2017. No promise is being made!		

To ensure that the students work and progress on the exercises listed above at an even pace, 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. This is a firm non-negotiable arrangement for the subject. [*At IITG, each day of an academic week is counted through an arrangement approved by the senate. For assessment recording, we shall use days of the week as listed by the academic section:* http://www.iitg.ernet.in/acad/acadCal/academic_calander.htm]

Grading 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 command `make check`. The subject instructor will randomly reassess some of the recorded exercise completions to ensure compliance at the desired quality standards. [This may happen up to two weeks after the assessment by a tutor]. It is anticipated that all assessments and reassessments will 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 group’s progress. A repeat outcome from a reassessment would preclude any other exercise assessment being recorded for the week in which the repeated exercise is assessed by the tutor. **Academic cheating should not bring a better grade!**

Grade	**Set of exercises to be completed for the grade**	**An alternate set of exercises for the grade**
DD	Start, T 01	Start, UP 01, UP 02
CD	Start, T 01, T 03	Start, T 01, T 02
CC	Start, T 01, T 02, T 03	Start, UP 01, UP 02, UP 03, UP 04
Groups may progress to exercises for the higher grades only after achieving grade CC		
BC	UP 01, UP 02, UP 03	T 01, T 02
BB	T 01, T 02, T 03, UP 01, UP 02, UP 03, UP 04 (All together)	
Groups may progress to exercises for the higher grades only after achieving grade BB		
AB	VM 01, VM 02	FS 01, FS 02, FS 03
AA	VM 01, VM 02, VM 03	FS 01, FS 02, FS 03, VM01, VM 02
AS	T01, T02, T03, UP01, UP02, UP03, UP04, VM01, VM02, VM03, FS01, FS02, FS03 (To be marked in the last week of the semester by the course-in-charge and only for those who reached AA level before the start of the last week)	

Summary: Our experience suggests that students find PintOS exercises challenging and may spend many hours to complete a PintOS project. To reduce the workload without affecting the pedagogical outcomes, we have divided these projects into smaller weekly exercises. An exercise may have smaller steps called Tasks to support their progress. Majority of these exercises include helpful guides.

Project teams/groups:

In the first week of the semester, the students must organize into teams of three students. It would be sensible if all team members have similar final grade ambitions for the subject. Each member 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 achievement. 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 designing solutions for the exercises, developing new PintOS code to implement their solutions, writing logbook entries to create record of work by the team, getting the exercise completion records validated and accepted by the team’s tutor.

It is important that all three 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 for the week) will be considered as submitted in the following week. Please take note of this constraint if you make a private arrangement with the tutor to have assessment completed at time and place other than the scheduled lab session.

The first exercise (labelled start) is slated for assessment in week 02 of the semester. From that time on you must report your progress in each lab session.

Note: A progress report is not the same as the recording of an exercise completion (assessment). Progress report or logbook is a weekly spreadsheet document with one row for each session of work the team has done since the previous submission. The columns in the logbook spreadsheet are:

A. Group name: smallest roll number among the members of the group.
B. Start date and time of the session/meeting in "year:month:date:hour:minute" format. All these values are to be filled as numerals as in this example: 2017:08:31:19:30
C. Names of the team members present during the work session.
D. Ten to fifteen words description of the aim/activity/achievements of the session.
E. Duration of the session (minimum 30 minutes)
F. Name the primary file:function affected by the work (one only – note both the file name and the function name is required)
G. Name any other file:function affected (one only)
H. 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.
I. How many relevant test cases for the exercise still report FAIL?
J. How many *relevant* test cases for the exercise report pass?
K. Final status of "`make check`" run (if applicable) at the end of the session. (Copy the last line of `make check` output). Examples: *All 27 tests passed*. And, *109 of 109 tests failed*.

The tutor when accepting a spreadsheet from a group must ensure that the columns are in correct order as listed above. In addition, they will verify the entry under the last column in the most recent logbook row.

The code must also be committed in the version control system each week. The tutor will verify that this action has been performed. (This is a necessary step to enable validation of assessments by the subject instructor.)

Recording Exercise Completion Status:

Each week, each team has one opportunity to get completion of one exercise recorded. The tutor will record all assessments made by him/her during the week in a spreadsheet with following columns (there is one row for each group assessed by the tutor in the week):

A. Team name: lowest roll number of the team member.
B. Roll number the second member (middle of the three roll numbers)
C. Roll number of the third member
D. Assessed exercise identifier
E. Date the assessment was witnessed
F. Time of the assessment.

An assessment is only recorded if all expected test cases pass the exercise.

Students must note that the first Exercise can be first assessed in week 02. 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 BB has four slack weeks. The number of slack weeks are also affected if the group is asked to repeat an exercise.

The teams may progress through these weekly exercises at different rates. Each team must take time in fully understanding and completing the exercise even if their progress falls behind the other groups.

Use of other team's solution lacks the learning benefit. And, if noticed would force both teams to repeat the exercise.

Tasks for the First week

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 is being 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.

```
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 for a call to function `main()` of user program `args-multiple`:

```
Executing 'args-multiple some arguments for you!':
ARGC:5  ARGV:bfffffb4
Argv[0] = bfffffe7 pointing at args-multiple
Argv[1] = bfffffe2 pointing at some
Argv[2] = bfffffd8 pointing at arguments
Argv[3] = bfffffd4 pointing at for
Argv[4] = bfffffcf 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 overcome your reluctance 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 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.

Steps to install Pintos in the 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`".

Contributing Authors:
Vishv Malhotra, Gautam Barua, Rashmi Dutta Baruah