SAINT MARY'S UNIVERSITY SINCE 1802 | FACULTY OF SCIENCE

CSCI 3431.1 Fall, 2017 – Assignment 1

Operating Systems

Due Date: Friday, September 29, 2017
Closing Date: Sunday, October 1, 2017

Deadline Policy

There will be a due date and a closing date for each assignment

- If you submit your work after the due date (but before the closing date), a *late submission* is applied to your overall assignments submission.
- You are allowed to use a *late submission* for up to 2 assignments throughout the semester without being penalized. After that, your assignment grade will be penalized by 5%.
- You will not be able to submit or receive credit for your work after the closing date.

Submission

Once you are done with your assignment, make sure you do the following before the assignment due date:

- Prepare a report containing the answers to the assignment questions.
- Fill out your Name and A# as indicated in the last page and attach this page to your report.
- Create one PDF file for your report.
- Combine your source code and PDF report into one zip file.
- Name your file following this convention: CSCI3431-Assignment1- *<StudentID>*, where *StudentID* is your Banner ID number starting with A.
- Submit your PDF file via Brightspace at https://smu.brightspace.com before the deadline.
- Do not submit a program that either won't compile or won't run. Instead of debugging your code, the marker/I will assign **Zero** credit for your submission.

Assignment Description

The assignment consists of two parts: practice exercises and a programming exercise.

0.1 Practice Exercises

1. **[2 points]** What is the difference between timesharing and multiprogramming systems?

2. **[4 points]** You would like to install and run both Windows XP and a ReHat distribution of Linux on your computer. Discuss how your system can allow a choice of OS from which to boot?

3. **[2 points]** What is the main advantage for an OS designer of using a virtual-machine architecture? What is the main advantage for a user?

4. **[3 points]** What type of multiplexing (time, space, or both) can be used for sharing the following resources: CPU, RAM, Network card, Printer, Keyboard and display

5. **[4 points]** There are several design goals in building an operating system, for example, resource utilization, timelines, robustness, and so on.

    a. Give an example of two design goals that may contradict one another. Explain.

    b. What are the trade-offs inherent in designing OS for handheld computers?

6. **[11 points]** The Android operating system is not only the most popular mobile OS [Morra J., 2016] but also a widely used OS for the Internet of Things. Describe (in at most 1 page) the Android Software Architecture and highlight key features that contribute to the success of this system. Please make sure you cite your work and use at least 3 references.

7. **[4 points]** A multiprocessor with eight processors has 20 attached tape drives. There is a large number of jobs submitted to the system that each require a maximum of four tape drives to complete execution. Assume each job starts running with only three tape drives for a long period before requiring the fourth tape drive for a short period toward the end of its operation. Also assume an endless supply of such jobs.

    a. Assume the scheduler in the OS will not start job unless there are four tape drives available. When a job is started, four drives are assigned immediately and are not released until the job finishes. What is the maximum number of jobs that can be in

progress at once? What are the maximum and minimum number of tape drives that may be left idle as a result of this policy?

b. Suggest an alternative policy to improve tape drive utilization and at the same time avoid system deadlock. What is the maximum number of jobs that can be in progress at once? What are the bounds on the number of idling tape drives?

0.2 Programming Exercise

The objective of this exercise is to refresh your programming skills as well as introduce you to processes.

Creating a Shell Interface

For this part, you will complete the project described in your textbook on p.149-152. You can use the `SimpleShell.java` skeleton code as a starting point. Here's what you need to do:

- ✓ Read the project description carefully.
- ✓ Compile and run `SimpleShell.java`. Make sure you understand how it works.
- ✓ Add code to parse the user input and split it into an array of strings. The input is split based on the space character.
- ✓ Process the user input as described in your textbook. Add the following functionality: – if the command entered is "`exit`" or "`quit`", your program should output a "`Terminating Shell...`" message and exit.
- ✓ In order for your program to execute the command entered by the user, you will need to create a `ProcessBuilder` object as described in `Figure 3.13` in your textbook. As well as handle the input and output of your process.
- ✓ Your program should now be ready! Test it with some commands like "`ls`", "`ps`" and "`cat`".
- ✓ Now add the changing directories ("`cd`" command) as described in Part 2 of your project description on p.151.
- ✓ Your program is once again ready! Test it with a few "`cd`" commands.
- ✓ Now add the history feature as described in Part 3 of your project description on p.151.
- ✓ Your program should perform basic error checking and exception handling.

Self-evaluation Please answer the following questions:

1. **[1 points]** Were you able to complete this assignment? What grade are you expecting? Please justify.

2. **[2 points]** Describe 2-3 challenges you faced while completing this assignment. How did you tackle those challenges?

3. **[2 points]** Provide a break down for the activities/milestones for this assignment. Give an estimate of hours spent on each activity. Try to be honest!

Date	Activities	Hours	Outcome

Hints and Suggestions

- ✓ Start early, the solution for the programming exercise totals less than 100 lines, but if you are not comfortable with JAVA/shell, it may take you a bit longer to implement.
- ✓ Document your work properly.
- ✓ Backup your work frequently. It's possible (and most likely) you go try a new feature and your program crashes!

Academic Integrity

You are required to demonstrate academic integrity in all of the work that you do. The University provides policies and procedures that every member of the university community is required to follow to ensure academic integrity. Unless stated otherwise, it is expected that all the work you submit for this course, is your OWN work.

Lack of knowledge of the academic integrity policy is not a reasonable explanation for a violation. You are encouraged to consult the Academic Integrity and Student Code of Conduct sections of the Academic Regulations in the Academic Calendar, in order to be well informed on the consequences of dishonest behavior. Please visit the links below for more information.

Evaluation Scheme

<table>
<tr><td colspan="2">Student Name:</td></tr>
<tr><td colspan="2">Student ID:</td></tr>
<tr><td>Total Points</td><td>/130</td></tr>
<tr><td>Practice Exercises
<table>
<tr><td>Question</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>Points</td><td>2</td><td>4</td><td>2</td><td>3</td><td>4</td><td>11</td><td>4</td></tr>
<tr><td>Score</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td><td>/30</td></tr>
<tr><td>Programming Exercise</td><td>/95</td></tr>
<tr><td>Starting the Shell
The program compiles and runs successfully.</td><td>/10</td></tr>
<tr><td>Terminating the Shell
Exit and quit commands work properly.
The program is not case sensitive to these commands.</td><td>/5</td></tr>
<tr><td>Creating Processes
The process builder is properly implemented.
Basic commands (ps, ls, cat, …) run successfully.</td><td>/20</td></tr>
<tr><td>Changing Directories
Absolute path is handled cd /home works properly
cd followed by pwd changes directory to /home/user
error message for invalid path</td><td>/20</td></tr>
<tr><td>Adding a History Feature
history command works properly.
!! command executes the last command in from the history.
!# executes the command # from the history or returns an error message if # is not a valid history command index.</td><td>/20</td></tr>
<tr><td>Handling Errors and Exceptions
Code deals with exceptions in an appropriate manner. For example, exceptions such as attempting to change directory to an invalid directory should result in a message to the user and the continuation of the program.</td><td>/5</td></tr>
<tr><td>Documentation
Code shows good indentation, meaningful variable names, modularity, and helpful comments.</td><td>/5</td></tr>
<tr><td>Testing
Readme file contains a script showing all the test cases performed along with screenshots.</td><td>/10</td></tr>
<tr><td>Self-evaluation questions</td><td>/5</td></tr>
</table>