CSCE 5640: Operating System Design

Project 1, Fall 2024

There will be two projects on this course for which you will be expected to work in small groups. This document contains the list of projects for Project 1. Your team needs to choose ***one*** from the list. The maximum group size is four (4) but can be less than four. You should implement it in C/C++/Java and testable in CSE machines.

First, please submit your project proposal (see Section: Project Proposal Format) which should include the selected project, team members, and tasks divisions by the due date (see Canvas). Proposal submission by one of the team members is sufficient. Then, for the project 1 submission, submit project report document (see Section: Project Report Format), and all the source code files, and a readme or any other file(s) that help to compile and test your project on canvas by the due date (see Canvas).

Projects:

Here is a list of project topics.

1. Unix/Linux shell
    a. Implement a command line interpreter or shell.
    b. A shell interface that accepts user commands, executes each command a user enters in a separate process, and then prompts for more user inputs.
    c. Should be similar to the shell that you use in everyday Linux such as bash or sh.
    d. The shell program should run in two modes: interactive and batch.
        i. In interactive mode, you will display a prompt (any string of your choosing) and the user of the shell will type in a command at the prompt.
            - **$./myshell**
            - **osh> ls -l**
        ii. In batch mode, your shell is started by specifying a batch file on its command line; the batch file contains the list of commands (in each line) that should be executed. In batch mode, you should not display a prompt.
            - **osh>./myshell cmds.txt**
    e. Your implementation will also support input and output redirection, as well as pipes as a form of IPC between a pair of commands.
    f. Minimum features:
        i. Two running modes: interactive and batch
        ii. Creating the child process and executing the command in the child
        iii. Providing a history feature
        iv. Adding support of input and output redirection
        v. Allowing the parent and child processes to communicate via a pipe.
    g. See more in Chapter 3 -> Programming Projects -> Project 1 – Unix Shell
    h. Useful link: https://pages.cs.wisc.edu/~dusseau/Classes/CS537-F07/Projects/P1/p1.html

2. Task Manager or System Resource Monitor
    a. Similar to task manager in Windows/Linux OS – shows list of processes/tasks, CPU utilizations, memory usage, disk usage, network usage, kill process/task, CPU time etc. Also, see "**top**" command in Linux.
    b. The top program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel.
    c. May use Linux kernel modules for task information/listing tasks.
    d. May use Kernel data structures.
    e. Minimum features:
        i. Show at-least these columns (and values) in tabular format: PID, USER, %CPU, %MEM, TIME+, COMMAND
        ii. Update the result of the top program frequently or when the value changes.
    f. Useful information can be found in Chapter 3 -> Programming Projects -> Project 2 and Project 3

3. Scheduling Algorithms
    a. This project involves implementing several different process scheduling algorithms. The scheduler will be assigned a predefined set of tasks and will schedule the tasks based on the selected scheduling algorithm. Each task is assigned a priority and CPU burst.
    b. Implement the following scheduling algorithms:
        i. First-come, first-served (FCFS), which schedules tasks in the order in which they request the CPU.
        ii. Shortest-job-first (SJF), which schedules tasks in order of the length of the tasks' next CPU burst.
        iii. Priority scheduling, which schedules tasks based on priority.
        iv. Round-robin (RR) scheduling, where each task is run for a time quantum (or for the remainder of its CPU burst).
        v. Priority with round-robin, which schedules tasks in order of priority and uses round-robin scheduling for tasks with equal priority.
    c. Priorities range from 1 to 15, where a higher numeric value indicates a higher relative priority. For round-robin scheduling, the length of a time quantum is 10 milliseconds.
    d. Implementation:
        i. Each algorithm should take a schedule of tasks from external files.
        ii. The schedule of tasks has the form ***[task name] [priority] [CPU burst]***, with the following example format:

```
T1, 4, 20
T2, 2, 25
T3, 3, 25
T4, 3, 15
T5, 10, 10
```

iii. Thus, task T1 has priority 4 and a CPU burst of 20 milliseconds, and so forth. It is assumed that all tasks arrive at the same time, so your scheduler algorithms do not have to support higher-priority processes preempting processes with lower priorities.
iv. Example of running executable in C/C++ for FCFS: **./fcfs schedule.txt**
- Executing in Java (if you use Java): **java FCFC schedule.txt**

e. Create test cases and compare the scheduling algorithms to find the best performing algorithms based on metrics such as average waiting time, turnaround, response time etc.
i. You should create multiple test cases with the same and different number of processes, their priorities and burst times in separate files for testing.
ii. For example, create at least 5 schedule files with 5 processes with their priorities and burst times, create at least 5 schedule files with 10 processes with their priorities and burst times, create at least 5 schedule files with 15 processes with their priorities and burst times, etc.
iii. You may randomly generate priorities and burst times for the test schedule files.
iv. Analyze your experimental results and draw conclusion on which scheduling algorithm performs better in your overall experiments.

f. See more in Chapter 5 (CPU Scheduling) -> Programming Projects -> Scheduling Algorithms

Useful resources:

1. https://pages.cs.wisc.edu/~dusseau/Classes/CS537-F07/projects.html
   a. The Unix Shell
   b. Multi-threaded Web Server
   c. A "Better" Malloc
   d. A "Slower" File System

2. https://www.cs.princeton.edu/courses/archive/fall16/cos318/projects.html
   a. Bootloader
   b. Non-preemptive kernel
   c. Preemptive scheduler
   d. Inter-process communication and process management
   e. Virtual memory

Other resources:

http://www.cs.columbia.edu/~nieh/teaching/e6118_s00/projects/index.shtml

http://www.cs.columbia.edu/~nieh/teaching/e6118_s00/projects/miniproject.shtml

https://ocw.mit.edu/courses/6-828-operating-system-engineering-fall-2012/pages/projects/

https://cseweb.ucsd.edu/classes/sp00/cse221/projects.html#topics

https://www.cs.colostate.edu/~cs551/Project/ListOfTopics.html

https://www.freebsd.org/

https://www.minix3.org/

https://www.raspberrypi.com/

https://www.raspberrypi.com/software/

http://www.kernel.org

References:

1. Operating System Concepts, Tenth Edition. Avi Silberschatz, Peter Baer Galvin, Greg Gagne, John Wiley & Sons, Inc., ISBN 978-1-118-06333-0

Proposal Format:

In the project proposal, the following sections should be presented:

1. Overview and objective(s) of the project.
2. Team size and team members.
3. Project plan
    a. Task divisions for the team members.
    b. Due date for subtasks.
4. Experimental environment
    a. Programming language for implementation.
    b. Operating system to test the project.
    c. Test cases.

Project Report Format:

The project must be accompanied by a detailed project report describing the problem, the implementation, experiments, and results as well as their interpretation. The final report should contain at least the following sections:

1. Title
    a. Title of your project
    b. Member name(s)
2. Introduction
    a. Overview of the project
    b. What problem being solved?
    c. Why is it important?
3. Background

    a. What does one need to know to understand the problem?
4. Implementation
    a. What are the solutions to your problem?
    b. How do you implement?
        i. Programming language for implementation.
        ii. Operating system to test the project.
    c. Test cases.
5. Experimental Results
    a. Any results or output in graphs or tables or figures that shows results of improvements/implementations.
    b. Interpretation of the results.
6. Conclusion
    a. What has been accomplished and what is still left to be done?
7. References