Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7376: Operating Systems: Interface and Implementation

Course Project

Instructions

- For this project you need to submit the following to the project assignment page on Canvas:
  1. Your **project report** developed by a word processor and submitted as <u>one</u> PDF file. No handwritten contents are accepted. Include in the report a summary of your approach to implement the requirements and screen shots of the sample runs of your programs. This report should be no longer than 8 pages, using 11pt Times font.
  2. The well-commented **source code files** (i.e., the .c or .h files) that you add or modify. At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any). For the xv6 programs that you only make changes in them, only add comments to the parts you change/add and no need to add any names at the beginning of these programs. Do not submit any compiled object or executable files.
  3. A **recorded video** demonstrating sample runs of your project programs. The sample runs must cover all project requirements. The video should not exceed 5 minutes and must be narrated by all members of the project group.
  4. **Do NOT submit** any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.
  5. **Do NOT copy** any code from any sources or any person. Submitted code must be completely your own work.

- You can work in a group of at most 2 students. If you work in a group, only one member should submit the above requirements on Canvas. Make sure to add the group members info on project report.

<u>*Note:*</u> You can submit multiple attempts for this project, however, only what you submit in the last attempt will be graded. This means all required files must be included in this last submission attempt.

Part 1 - Shell

The goal of this part of the project is implementing your own shell program on a Linux environment (not xv6). The shell program repeatedly reads commands from the user, interprets them, and translates them into a sequence of actions that likely involve an interaction with the operating system through system calls. Your code can rely on the string manipulation routines implemented throughout our previous homework assignments. The shell must include the following set of required features:

1. The shell should print a `$` prompt symbol followed by a space when it is ready to receive a new command from the user.
2. When a command is launched in the foreground, the shell should wait until the last of its subcommands finishes before it prints the prompt again. A command is considered to be running in the foreground when operator `&` is not included at the end of the command line.
3. When a command is launched in the background using suffix `&`, the shell should print the `pid` of the process corresponding to the last sub-command. The `pid` should be printed in square brackets. It should then immediately display the prompt and accept new commands, even if any of the child processes are still running. For example:
```
$ ls -l | wc &
[3256]
$
```
4. When a sub-command is not found, a proper error message should be displayed, immediately followed by the next prompt. Example:
```
$ hello
hello: Command not found
$
```
5. The input redirection operator `<` should redirect the standard input of the first sub-command of a command. If the input file is not found, a proper message should be displayed. Example:
```
$ wc < valid_file.txt
223 551 5288
$ wc < invalid_file
invalid_file: File not found
```
6. The output redirection operator `>` should redirect the standard output of the last sub-command of a command. If the output file cannot be created, a proper message should be displayed. Example:
```
$ ls -l > invalid/path
invalid/path: Cannot create file
$ ls -l > hello
$ cat hello
< Contents of “hello” displayed here >
```

Part 2 – xv6 Extension

The goal of this part of the project is extending the xv6 operating system by modifying its scheduler. Modify the default round-robin (RR) scheduler implemented in xv6 to provide a more sophisticated scheduling policy that utilizes the Multi-Level Feedback Queue (MLFQ). Assume the new scheduler handles only 3 priority levels (0 to 2) of queues, with the highest priority queue at level 0. The scheduler uses RR to run the `RUNNABLE` processes from a priority queue only if all the `RUNNABLE` processes in the higher priority queues are done. Assume the default priority level assigned to any process is level 1 (the middle level). Implement system call `renice()` to allow a process to change its own priority, in a similar way to how this system call works on Linux (see `man renice`). You will need to add a priority attribute to `struct proc`

Show how your scheduler is affected by this modification by adding a user program to xv6 to test its new scheduler. The program needs to fork multiple processes. Each one of these processes needs to call `renice()` to change their default priority. Utilize some programming techniques, such as `sleep()` and long running loops, to simulate a situation to how your new scheduler works differently than the default RR scheduler. You might need to utilize the xv6 `wait()` system call to collect information about the order on which processes are completed.