# CS 550 Operating Systems, Spring 2018
# Programming Project 1

Out: 2/13/2018, Tuesday
**Due date**: 3/4/2018, Sunday 23:59:59

In this project, you are going to add a new system call to enable a command named "shutdown', which shuts down the xv6 QEMU VM, in part I; and add a new shell program named "xvsh" to xv6 OS in part II.

## 1 Baseline xv6 source code

**ATTENTION**: before you proceed, read the whole spec carefully and make sure you understand all the requirements, especially those highlighted parts in this section, the submission requirements (Section 4), and the grading guidelines (Section 5).

- Log into your GitHub account whose username is the same your BU username.
- Go to the link at the end of this section to accept the assignment. This will create a private repository of your own on the instructor's teaching organization, and start importing the baseline code into your private repository.

  Note: sometimes the import process may take a long time. Do not click the "Cancel" button, which will lead to an empty repository. You can just close the page and come back later.

  If you have clicked the "Cancel" button or it already took really long (e.g., more than two hours), contact the instructor and the TAs so we can delete your repository, and you can click the assignment link to import again.
- Once the import process finishes, you can clone or download the repository into your computer and start working.
- Start early! The instructors and the TAs will be less responsive during the last several days before the due date, and will not respond to any email inquiries regarding the project in the last day.
- Refer back to the spec of PROJ0 for details of running xv6.

Assignment link: `https://classroom.github.com/a/LW6xu583`

## 2 PART I: system call that shuts down the machine (30 points)

### 2.1 What to do

As you may have noticed, the original xv6 system doesn't have a "`shutdown`" command to properly turn off the emulated machine. In the baseline code, a new file named `shutdown.c` implements the user space command `shutdown`. The missing parts for the `shutdown` command (and thus your jobs for this part) are the implementation of the system call that can shut down the machine and the corresponding system call user-level wrapper function, which is called by the `shutdown` command program (i.e., `shutdown.c`). The system call user-level wrapper function should have following prototype:

```
void shutdown(void);
```

and should be declared in "user.h". In `shutdown.c`, a stub implementation of the wrapper function is given to ensure the correct compilation of the baseline code. Remember to deactivate the stub function after having you own implementation.

After correctly implementing the missing parts, running the command `shutdown` should shut down the QEMU VM.

### 2.2 Hints

- To shut down the virtual machine in your system call, you only need two lines of code:

  ```
  outw(0xB004, 0x0|0x2000);
  outw(0x604, 0x0|0x2000);
  ```

- Reading and understanding how the existing system calls and the corresponding user-level wrapper functions are implemented will be helpful.
- Understanding the mechanism is important. You may be requested to explain how things work in xv6 in midterm/final exams.

## 3 PART II: the **xvsh** shell (70 points)

In the second part of the project you are going to practice basic multi-process programming by implementing a new shell program in xv6.

### 3.1 What to Do

The following are the details of your implementation.

(1) The name of the shell program should be "xvsh". When running the xvsh command, it should enter the new shell program. For example,

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 1973
cat            2 3 13264
echo           2 4 12484
forktest       2 5 8192
grep           2 6 14900
init           2 7 13032
kill           2 8 12560
ln             2 9 12448
ls             2 10 14676
mkdir          2 11 12604
rm             2 12 12580
sh             2 13 23392
stressfs       2 14 13180
usertests      2 15 58364
wc             2 16 13780
zombie         2 17 12256
shutdown       2 18 12352
xvsh           2 19 19048
console        3 21 0
$
$ xvsh
xvsh>
```

where "$" is the prompt of the original shell, and "xvsh>" is the prompt of the new shell program (i.e., xvsh).

(2) When running in the new shell, all the shell commands/user programs should be invokable normally. For example, running the ls command in xvsh will give us:

```
xvsh> ls
.              1 1 512
..             1 1 512
README         2 2 1973
cat            2 3 13264
echo           2 4 12484
forktest       2 5 8192
grep           2 6 14900
init           2 7 13032
kill           2 8 12560
ln             2 9 12448
```

```
ls             2 10 14676
mkdir          2 11 12604
rm             2 12 12580
sh             2 13 23392
stressfs       2 14 13180
usertests      2 15 58364
wc             2 16 13780
zombie         2 17 12256
shutdown       2 18 12352
xvsh           2 19 19048
console        3 21 0
xvsh>
```

As another example, assuming the `shutdown` command has been enabled, calling `shutdown` in `xvsh` should shut down the QEMU VM normally.

Yet another example, it should be possible to run "`xvsh`" again when already running in `xvsh`.

(3) The `xvsh` should support the "`&`" operator to put commands in the background.

By default, a command is executed in the foreground, with which the shell waits for the command to complete before showing the shell prompt again.

When a command/program is executed in the background (by appending an "`&`" operator to the end of the command line), the shell should display the shell prompt immediately after the command line is keyed into the system. The command/program will be running in the background, but should generate the terminal outputs (if there is any) as usual.

For example, suppose we write a program (named "`sleep-echo`") which simply sleeps for 5 seconds, then wakes up, echoes the message provided by the user and a newline character, and exits. Below is an example of the expected output if we run the program twice in the background.

```
1   xvsh> sleep-echo Hello World! &
2   [pid 29] runs as a background process
3   xvsh>
4   xvsh> sleep-echo Goodbye World! &
5   [pid 30] runs as a background process
6   xvsh> Hello World!
7   Goodbye world!
8
```

A few more details about this task:

- After a command is keyed into the system to run in the background, `xvsh` should output a message as:

  ```
  [pid-of-the-new-process] runs as a background process
  ```

  before displaying the prompt again, such as line 2 and line 5 in the above example.
- Line 2 and the shell prompt in line 3 should be displayed immediately after command line in line 1 is keyed in. Similary, line 5 and the shell prompt in line 6 should be output immediately after command line in line 4 is keyed in.

- In this particular example, the message "Hello world!" plus the newline character are displayed 5 seconds after the command line in line 1 was keyed into the system. Again similarly, the message "Goodbye world!" plus the newline character are output 5 seconds after the command line in line 4 was keyed into the system.

(4) `xvsh` should be able to handle empty command line (i.e., displaying a new line with the shell prompt if just the "Enter/Return" key was pressed).

(5) `xvsh` should support a built-in command named "exit", which causes return to the previous level of shell. In other words, your implementation of `xvsh` will need to capture the command "exit", then leaves the current shell. For example,

```
xvsh>
xvsh> exit
$
```

where "`$`" is the prompt of the original shell, and "`xvsh>`" is the prompt `xvsh`.

If there are background processes are running when the "exit" command is keyed in, `xvsh` should wait for all the background processes to complete before returning to the previous level of shell.

(6) `xvsh` should output message "Cannot run this command xxx" for wrong command line input xxx. For example,

```
xvsh> a-wrong-cmd arg1 arg2
Cannot run this command a-wrong-cmd
```

## 3.2 Hints

- Build and test one functionality at a time.
- You need to use the following functions for the implementation: `read()`, `fork()`, `exec()`, `wait()`, `exit()`. The usage of these functions may be slightly different from those in standard `libc`. You can refer to [1] or the source code of xv6 to see the detailed usage of these functions in xv6.
- The original shell of xv6 does not handle "`&`" operator properly: every background process will become a zombie process. Your `xvsh` implementation should avoid this.
- Three header files may be needed for the xvsh implementation: `types.h`, `fcntl.h`, and `user.h`.
- In xv6, the support of string manipulation functions is limited, you may need to write your own function for this purpose.

## 4 Submit your work

If you use the GitHub website upload function to commit code to the GitHub server, remember to "`make clean`" before uploading. Uploading intermediate files, such as `.o`, `.d` files will lead to points taken (see grading section for details).

A submission link be available on the MyCourses website. **Once your code in your GitHub private repository is ready for grading, submit a file named "DONE", which should include the following info**, to that link:

- Your name and B-number.
- **The SHA1 value (a 40-char string) of the latest commit.** You can obtain it from the GitHub website or from the "`git log`" command. (Note: this is new compared to PROJ0.)

Additionally, you can include the following info in this file:

- The status of your implementation (especially, if not fully complete).
- A description of how your code works, if that is not completely clear by reading the code (note that this should not be necessary, ideally your code should be self-documenting).
- Possibly a log of test cases which work and which don't work.
- Any other material you believe is relevant to the grading of your project.

**Suggestion**: Test your code thoroughly on a CS machine before submitting.

## 5 Grading

The following are the general grading guidelines for this and all future projects.

(1) **The code in your repository will not be graded until a "DONE" file is submitted to MyCourses**.

(2) The submission time of the "DONE" file shown on the MC system will be used to determine if your submission is on time or to calculate the number of late days. Late penalty is 10% of the points scored for each of the first two days late, and 20% for each of the days thereafter.

(3) Your GitHub username should be the same as your BU username. If your BU username has been used by someone else, contact the instructor for the solution.

If you have multiple GitHub accounts, please make sure you log into the one which has your BU username as the GitHub username before you accept the assignment.

Once you accept the assignment, you can check what account you used to accept the assignment. For example, if your the GitHub account you use to accept the assignment is "abc", the name of your private repository will be "cs550-18s-projX-abc" (with X=0, 1, 2, 3, or 4). If you use a GitHub account whose username is different from you BU username, our script will not be able to locate your repository.

If you made the mistake, you can correct it by logging into the correct GitHub account, and go to the assignment link and accept again.

Any grading issues caused by using incorrect GitHub account will lead to 10 points off.

(4) Uploading intermediate files (e.g., `.o`, `.d` files) will lead to 5 points off (“`make clean`” before uploading can avoid this).

(5) No SHA1 value of the latest commit in the “DONE” file, 5 points off.

(6) If you are to compile and run the xv6 system on the department’s remote cluster, remember to use baseline xv6 source code provided by our GitHub classroom. Compiling and running xv6 source code downloaded elsewhere can cause 100% CPU utilization on QEMU.

Removing the patch code from the baseline code will also cause the same problem. So make sure you understand the code before deleting them.

If you are reported by the system administrator to be running QEMU with 100% CPU utilization on QEMU, 10 points off.

(7) If the submitted patch cannot successfully patched to the baseline source code, or the patched code does not compile:

```
1   TA will try to fix the problem (for no more than 3 minutes);
2   if (problem solved)
3     1%-10% points off (based on how complex the fix is, TA’s discretion);
4   else
5     TA may contact the student by email or schedule a demo to fix the problem;
6     if (problem solved)
7       11%-20% points off (based on how complex the fix is, TA’s discretion);
8     else
9       All points off;
```

So in the case that TA contacts you to fix a problem, please respond to TA’s email promptly or show up at the demo appointment on time; otherwise the line 9 above will be effective.

(8) If the code is not working as required in the project spec, the TA should take points based on the assigned full points of the task and the actual problem.

(9) Lastly but not the least, stick to the collaboration policy stated in the syllabus: you may discuss with you fellow students, but code should absolutely be kept private. Any kind of cheating will result in zero point on the project, and further reporting.

## References

[1] XV6 a simple, Unix-like teaching operating system. https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf.