Grading Rubric for Assignment 1

Basic shell with **system()**	5%
Implementation with **fork()**	10%
Implementation with **vfork()**	5%
Implementation with **clone()** ***Getting cd to work [5%]***	20%
Piping with FIFO	25%
Timing	10%
Report Needs to answer all questions asked in the assignment Properly formatted	15%
Code quality and readability **Error handling [2%]** **Signal handling [2%]** **Formatting [2%]** **Comments [2%]** **Misc [2%]**	10%

Different implementation versions will be tested for:

- One shell command produced interactively **(eg: ls)**
- Multiple shell commands produced interactively **(eg: ls, pwd, echo, date)**
- Commands with flags **(eg: ls -al, date +%s)**
- Multiple commands redirected from a file
- Commands that complete with one output **(eg: ls)**
- Commands that produce continuous output **(eg: ping 8.8.8.8)**
- Commands that expect input from STDIN upon execution

  (eg: tr 'ABC' '123' <press_enter> then it expects some input)
- A command with incorrect flags
- A command that does not exist **(eg: pswedk)**
- A simple C program
- A simple C program which SegFaults (crashes)
- Typing "exit" should exit your tiny shell
  - This is important. If you are stuck in the child process then you will have to type "exit" twice, which is wrong.
  - Must handle this properly by checking for errors with the exec() function execution and calling "exit()" or “_exit()”.
    [Which of the exit to be called is also important]

- "cd" will be checked only with the clone() based my_system() implementation
  - It is important to note/check how it works in a usual shell
- For a running command if a signal (SIGTERM) is sent then only that command should be terminated and the tiny_shell should continue to accept new commands

- No commands with shell piping (|) will be tested for

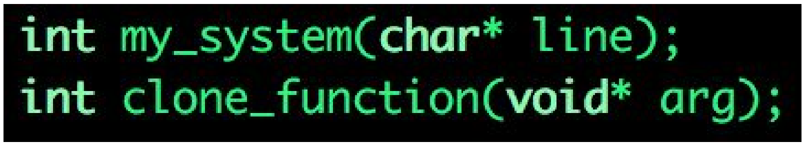
Notes:

- Any plagiarized content (code or report) will be heavily penalized.
- Debugging with **printf()** is not a good idea. It does buffered writes. Thus, you may not see the prints on the console even after the program had gone past a certain point unless the buffer is flushed. As a workaround follow every printf() statement with **fflush(stdout);** to force a buffer flush.
- Use **gdb** for debugging. Will try to have a gdb tutorial before the deadline. If not possible will schedule one before the next one.
- Look up ***follow-fork-mode*** for using gdb with fork() and multiple processes.
- Follow the following code structure to make grading easier and efficient

Code structure

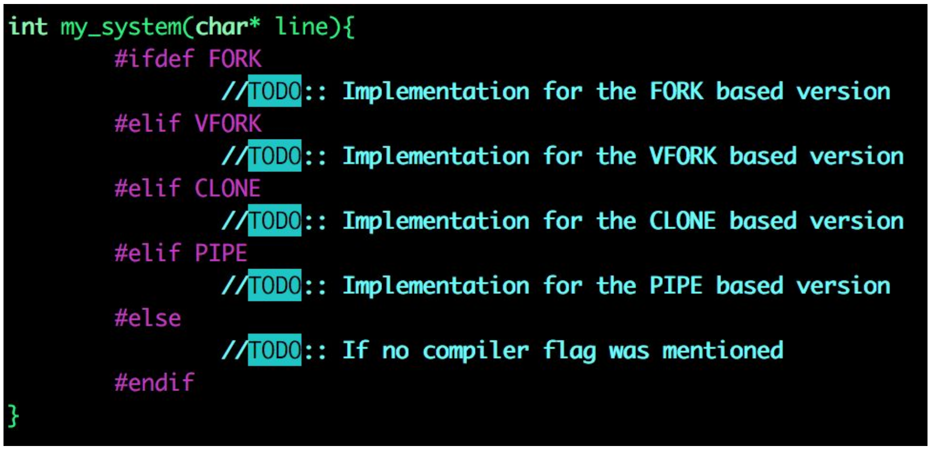
[If already submitted then it is fine. If already written a lot of code and cannot change it then that is also fine. But following the structure below will ease the task of the TAs]

- Should have the methods called **my_system()** and **clone_function()** with the following signatures in addition to main() and any other supplementary methods.



```
int my_system(char* line);
int clone_function(void* arg);
```

- Have all the different implementation of the my_system() method in the same file guarded by compiler macros as follows:



```
int my_system(char* line){
        #ifdef FORK
                //TODO:: Implementation for the FORK based version
        #elif VFORK
                //TODO:: Implementation for the VFORK based version
        #elif CLONE
                //TODO:: Implementation for the CLONE based version
        #elif PIPE
                //TODO:: Implementation for the PIPE based version
        #else
                //TODO:: If no compiler flag was mentioned
        #endif
}
```

So the TA's can compile your code as follows:

```
gcc -D FORK tiny_shell.c -o tshell
gcc -D CLONE tiny_shell.c -o tshel
```

If you had structured your code like this, then you can use the given makefile to quickly compile different versions of the tiny_shell as follows:

make fork	-	compiles the FORK version
make clone	-	compiles the CLONE version

We have also given a sample hello world program to test your shell with a simple C program.

make hello	-	compiles the sample C program
make hello_seg	-	compiles the crashing version of the above

You can test your shell by running the two versions of the C program and further test it by expanding the C program a little bit more

- The main source file [the one with **main()** and **my_system()**] must be named **tiny_shell.c**

Hand in

- Report
- Source code (**tiny_shell.c** and other files if more)
- **makefile** if any
- README file with any information pertaining to how to run the source