1. Write a program that fork once. Find out what happens when fork returns? Also find out what the parent gets as the return value of the fork and what the child gets as the return value of the same call.
2. Write a program that, forks to create a child process. The child exits with an exit status of 10 and the parent waits for the death of the child. Display the PID & PPID of both processes and let the parent display the child's exit status. (Use fork, exit, wait)
3. Write a program that executes the "cp filename1 filename2" command. Call your executable ***mycp.***

a) Create a file with the name of filename1.

b) The call to your program will be made with the following command:

***$ mycp filename1 filename2***

c) Your code will check for the correct number of arguments.

If it is not correct, then an error message will be produced and the code will exit.

d) Your code will call ***fork()***

e) The child process will use the ***execlp*** to call ***cp*** and use the filename1 and filename2 passed as arguments on the command line

f) The parent will wait for the child to finish

g) Your program will also print from the child process:

1) The process id

2) The parent id

And print from the parent process:

1. The process id
2. The parent id

Comment out the ***execlp*** call and add instead a call to ***execv***. Add any necessary variables to do that.

(Man Page for execlp and execvp - <http://linux.die.net/man/3/execlp>)

1. Write a program where alarm is set for 3 seconds and read system call is used to read data. If user takes more than 3 seconds then SIGALRM signal is generated and the signal handler prints a message and the program exits.
2. Implement a ***command interpreter (Shell)*** for Linux like operating system. It should support
3. Correctly parsing (breaking in tokens) the command line
4. Differentiate between an internal (builtins) and external command
5. List of internal commands to be implemented are
6. ***cd*** should change the working directory
7. ***history*** should print the list of previously executed commands. The list of commands should be numbered such that the numbers can be used with ! to indicate a command to repeat.
8. ***exit*** should terminate your shell process
9. ***kill %num*** should terminate the process numbered in the list of background processes returned by jobs (by sending it a SIGKILL signal).
10. ***jobs*** provide a numbered list of processes currently executing in the background.

( *waitpid system call* can be used with WNOHANG option to check without blocking.)

1. *help* lists the available built-in commands and their syntax
2. You should be able to place commands in the background with an & at the end of the command line
3. Piping and Redirection (for external commands only) should be supported.
4. If the user chooses to exit while there are background processes, notify the user that these background processes exist, do not exit and return to the command prompt. The user must kill the background processes before exiting

References for this question:

1. <http://web2.clarkson.edu/class/cs444/assignments/shell/>
2. <https://www.cs.drexel.edu/~jjohnson/2012-13/fall/cs543/assignments/shlab/index.html>
3. <http://pages.cs.wisc.edu/~dusseau/Classes/CS537-F04/Projects/P1/>