CSCI 3614, Systems Programming

Assignment 3 (22.04.2019)

This assignment contains 5 tasks, giving you 100 points in total. Each task is a separate application, related to the material covered (or to be covered) during the lectures. Topics for these tasks are Signals and Threads.

For each task create a corresponding folder (see the names of folders in the header of corresponding tasks inside quotes). Upload only source files (not the binary files). Pack all your folders into single ZIP archive.
Always make sure that your code is neat, clean and thoroughly commented. Try to use meaningful variable names.
Deadline: 30.04.2019 23:59

1 Double Quit - double_quit (20 points)

Write a simple program that continuously reads some text from the standard input and prints it to the standard output. To quit the program user has to send a **quit signal** (Ctrl+\) two times in a row. If the second **quit** does not come within 5 seconds, then the state of the quit signal should be reset, i.e. user should press it twice again.

2 Signal Blocking - blocker (30 points)

Write a program that copies a source file to a destination file. Both file names can contain absolute or relative path, will given through command line arguments (e.g. **./blocker source.txt destination.txt**). Make sure that while in a process, your program handles signals, such as SIGINT, SIGQUIT or SIGTERM, only after it finishes copying process (signals should not be lost).

3 Shell Reloaded - shell (40 points)

As you have already implemented the shell, you won't have to implement it from scratch. The only thing that is required from you to extend it a bit (see **Assignment 2, Task 2**). This time, instead of running child processes in the foreground, you will have to run them all in the background and the parent shouldn't wait for them, meaning you should be able to execute new commands, while the old ones still run in the background. Furthermore, whenever the background process you have created terminates, the shell must be notified about it and should print information about the terminated background process - print out the process id and exit status of this process.

4 Multithreaded Merge Sort - merge_sort (10 points)

There are many sorting algorithms, and among those there are such algorithms, that can perform even better if they are run on a multiprocessor platform. Quicksort and merge sort fall into this category of algorithms as they both use the divide and conquer strategy. Your task is to implement a multithreaded version of the merge sort algorithm, where parts of the array, with the help of threads, can be sorted independently after partitioning.