— Assignment #4 —

If you are reading this either the day before or the same day this assignment is due:

- the only way to earn marks now is if you write test code for the others members in your group
- no marks will be earned for a copy-paste or refactor of someone else's code
- test code submitted will have a 10% penalty applied for not fully participating in your group

The basic work to do for test code:

- check each of the functions of the others group members code
- if the others have all code in the same function, then split into reasonable smaller functions yourself
- test the organized functions with enough variety of input to see that they work correctly
- use a text file to log the results of executing your tests with time and date for each test session

If there is not code submitted from others in your group, then that is your choice to gamble working on assignment last-minute.

Remember, you must have a git log of group collaboration progress as ONE main branch to earn full marks. Commit to your git at least every other day, and break down your work into small pieces. Check Blackboard group for messages.

Learning Goals

1. write a Bash script to manage compile and execution of programs
2. write a C program that sends a signal to another process with kill system call
3. write a C program that handles signals

You will work in groups of two or three on this assignment, and submission is by group:

- you do not need to wait for anyone else to get started programming;
- using UFV GitLab as much as you can in your group;
- reference any website you use to help you write *small* parts of your code
  (e.g.: the code used from an outside source should not solve the entire assignment),
- if so, give brief comments describing how you modified something you used and why;
- do not copy an entire program.
- Backup your code separately—code only in the group shared repo can easily be lost by others in the group experimenting with git commands; one git clone folder and a different folder of your own.
- make your own separate copy, IN A DIFFERENT FOLDER, and not a subfolder of the clone.

Together, choose a group leader and have them:

- create a project named `ScriptSignalProject` on GitLab
- add the others to this project (their UFV GitLab username should be on Blackboard discussion)
- set "Developer" role for each person in the project in UFV GitLab (otherwise they cannot git push)
- post your UFV GitLab username in the Blackboard discussion if you have not done so already (make sure to repost if you change it again)

Memory Mapping

The instructions this time will not be giving a step-by-step description.

Write two programs and a Bash script program

- name the first program `one.txt`
- name the second program `two.txt`
- name the Bash script `run_both`
- make the first program send a signal
  - control which process ID to send to by passing in a command-line argument
  - send the signal `SIGUSR1`
  - output a final statement that the end of the program is reached just before terminating
- make the second program handle general signals
  (you cannot override some signals, so do not worry about `SIGKILL` or `SIGSTOP`)
  - have the handler print out the signal number
  - output a final statement that the end of the program is reached just before terminating
- make the `run_both` script:
  - compile the two programs
  - run the second program as a background job, which will then have a process ID
  - have the script `sleep 0.01` seconds
    (otherwise, the next step may execute too quickly and not work)
  - store the process ID number in a script variable (get the process ID with `pgrep two`)
  - execute the first program with the stored process ID given as the first argument
- prefix output statements in your programs with `(PROCESS ONE)` or `(PROCESS TWO)` accordingly
- observe which output statements

No video demos this time. Make sure to cite (give URL and date of access) for any reference source material you use as comment above in file. Your code should not be a complete copy/refactor of reference code.

Debugging

With a VirtualBox, we can install GDB, which is the GNU Project Debugger. We do not have this installed on Jupyter Notebooks. Install and then run your program with GDB by compiling your program with the `gcc` option `-g`. GDB can tell you what line segmentation faults (bad memory accesses) happen inside your program.

Various commands for GDB:

- `gdb` begin an interactive session of GDB
- `quit` exit the interactive GDB terminal prompt
- `file <your_program>` load your program
- `run` let GDB execute your program

Submission

Submit your git repository as a bundle that contains ONE main branch with collaboration in your group.

Create a bundle file on the command line using git:

```
git bundle create ScriptSignalProject.bundle main
```

inside the `ScriptSignalProject` folder. Do not worry about adding this file to your project itself. It is meant to compress the project into one file with the records of your commits. You can check what is in your bundle file with the following command:

```
git bundle verify ScriptSignalProject.bundle
```

Submit the bundle file only to our Blackboard Assignment and Tests section in Assignment 3. You can submit multiple times, but I only mark the last submission before deadline. Submit every couple of days, and then you are always worry free to know your submission has most of your hard work.

Please do not email me to check that your files are submitted to Blackboard. You are capable of checking that yourself.

This assignment is due on Monday, Dec 5, 11:59 pm.

Marking Rubric

20 marks total

	excellent	**good**	**adequate**	**poor**
code style (4 marks)	no mixed whitespace (tabs and spaces), indentation aligns per block of code, meaningful variable names, and no unnecessary lines of code	one or two issues with mixed whitespace, alignment, variable names, or extra unnecessary code	more than two issues with style	many issues with style
code functionality (16 marks)	project has regular (every other day) git log where group members contribute to each other's progress (such as helping to debug), no runtime errors, all functionality described is completed without compiler errors, warnings, or runtime misfunction	project has git log where group members contribute to each other's progress (such as helping to debug) with work split up in smaller pieces at least across three days, script and programs have no errors and have most of the functionality expected	git log with few commits, or a few runtime errors, or variable names are trivial `"a", "b", "c",`..., or output is difficult to read, or memory is not managed properly	no git log, there are runtime errors that halt the program, or output is not presented in a readable format, or the program does not compile