Systems Software

VISUALIZATIONANDMULTIMEDIALAB

BINF31aa (BINF3101)

Prof. Dr. Renato Pajarola

Exercise Completion Requirements

- Exercises are mandatory, at least 3 of them must be completed successfully to finish the class and take part in the final exam. The first exercise serves as an introduction to C/C++ and does not count towards admission to the final exam.
- Exercises are graded coarsely into only two categories: ***fail*** or ***pass***
  - if all the exercises (without considering the first introductory exercise) are completed successfully, then a bonus is carried over to the final exam[1]
- Turned in solutions will be scored based on functionality and readability
  - a solution which does not compile, run or produce a result will be a ***fail***
  - the amount of time spent on a solution cannot be taken into account for the score
- Exercises can be made and submitted in pairs
  - both partners must fully understand and be able to explain their solution
- Students may randomly be selected to explain their solution
  - details to be determined during exercises by the assistant

Exercise Submission Rules

- Submitted code must compile without errors.
- Code must compile and link either on Mac OS X or on a Unix/Linux/Cygwin environment using a Makefile
- The whole project source code (including Makefile) must be submitted via OLAT by the given deadline. Include exactly the files as indicated in the exercise.
- We will use MOSS to detect code copying or other cheating attempts, so write your own code!
- The whole project (including Makefile) must be zipped, and the .zip archive has to be named: ss_ex_EXERCISENUMBER_MATRIKELNUMBER.zip (ss_ex_3_01234567.zip - one student, Systems Software exercise number 3; ss_ex_1_01234567_02345678.zip - two students, Systems Software exercise number 1).
- Submit your code through OLAT course page.
- Teaching Assistant: Claudio Mura (claudio@ifi.uzh.ch)

Help with C++, Makefiles and UNIX programming

C++ Tutorial:	http://www.cplusplus.com/doc/tutorial/
C++ Library Reference:	http://www.cplusplus.com/reference/
C++ STL:	http://www.sgi.com/tech/stl/table_of_contents.html
GNU Make Tutorial:	http://www.gnu.org/software/make/manual/make.html
Reference book:	"Advanced Programming in the UNIX® Environment, 2nd edition" *W. R. Stevens, S.A. Rago,* Addison Wesley

[1] the bonus will be in the order of 10% of the total number of points achievable in the final exam

NOTE: for this exercise it is assumed that you have acquired sufficient familiarity with the C++ language and with Makefiles. Before you start writing the solution make sure you have understood the relevant theory chapters from the Operating Systems book. You might find it useful to check the man pages of `fork`, `exec`, `getpid`, `wait`.

2. Process creation and management

The goal of this exercise is to write a multi-process program that, given a string (the *pattern*) and a list of filenames of text files, counts the occurrences of the pattern in the files. The search should be done in parallel, creating one process for every text file to be analyzed. To accomplish this goal you will make use of the process management routines of the UNIX environment.

Your C++ application will receive the pattern string and the filenames as command-line arguments. After checking that the correct number of parameters has been provided, your program will extract the number *n* of filenames provided in input and will create *n* child processes by repeatedly calling the system call `fork()`. Every child process will consider one file, will determine the number of occurrences of the pattern in that file and will write this number to a text file named `result-PID.txt`, with PID being the ID of the child process. Note that the search should be performed by replacing the image of the process with that of an instance of the shell executing the following command: `grep -o PATTERN FILENAME | wc -l > result-PID.txt` . You can do this by means of the system call `exec()`, providing it with the path to the executable of the shell (typically, `/bin/sh`) and, as arguments of `/bin/sh`, the string `-c` and the string corresponding to the search command (`grep -o ...`).

After spawning the children, the main process will wait for their completion (using the `wait()` system call) and check their termination status. If all children terminated correctly, the process will read the results files generated, compute the overall number of occurrences and print it to standard output.

Some additional comments on this task:

- do not search for the pattern manually! you *must* use the `exec()` function;
- you can choose which version of the `exec()` function to use, as long as it is used correctly and it achieves the functionality requested;
- make sure you detect and cope with the following anomalous situations: insufficient number of command-line arguments; errors when opening a file; error in the calls to `fork` and `exec`; abnormal termination of the child processes.

Further clarification on this exercise will be provided during the tutorial session of September, 28th.

Notes:

- you are provided with some sample sources files and with a simple Makefile; you may use them as a basis for your solution.

Deliverables:

Electronically submit your project files (i.e. source files and Makefile) and a README file that briefly explains your program, all in a single ZIP file.

Deadline: 11th October, 2015 at 23.59h.