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

Exercise 5: Tic-Tac-Toe using UNIX domain sockets

The goal of this exercise is to develop a multi-process application that implements the well-known "Tic-Tac-Toe" game (http://en.wikipedia.org/wiki/Tic-tac-toe) using UNIX domain sockets for IPC.

UNIX domain sockets are one of the most versatile IPC tools in UNIX systems. They allow processes running on the same machine to exchange data in a bidirectional fashion. Unlike unnamed pipes, UNIX domain sockets do not require that processes using them share a common ancestor. A special file is in fact created in the filesystem when a UNIX domain socket is set up (more precisely, during the *bind* phase), as a means of advertising the socket name to other processes.

Your solution to this exercise should be structured as a client-server application. One process (the server) should open a UNIX-domain socket and listen on that socket for incoming connections. Other processes (the clients) should try to connect to the server. When two clients have connected, the server should stop accepting connections and should start the game.
The server process should maintain the state of the game and keep track of the turns. It should notify the clients of the game start, receive from them string-based descriptions of their moves, update the state of the match and send updates to the clients (*i.e.*, send the opponent's move, notify of the outcome of the match).
Each client should read the synchronisation messages of the server, provide a simple yet clear visual output describing the status of the match and acquire the moves from the user.
Note that the processes described above **must not** share a common ancestor. You must create two separate programs for the server and the clients, resulting in two distinct executables.

Some additional comments about the task:

- the first client to connect to the server should take the first move;
- both the server and the client program should take as their first command line argument the pathname to be used as address for the server socket;
- in case the file to be used for the server socket already exists, the server process should take care of removing it (using `unlink()`);
- you should create sockets that use the stream interface (*i.e.*, specify `SOCK_STREAM` in the `socket()` call).

Further clarification on this exercise will be provided during the tutorial session of November, 9th.

Notes:

- you are provided with some sample source 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: 22nd November, 2015 at 23.59h.

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 fully understood the relevant theory chapters from the Operating Systems reference books.