Fundamentos de Sistemas de Operação

MIEI 2013/2014

Laboratory session 6

Programming Assignment 1 (2nd part)

Deadlines

Programming assignment 1 (PA1) will be developed in the classes of the weeks starting 14th October, 21st October, and 28th October – The deadline for submitting PA1 (report and code) is ***Monday, November, 4th – 20:00****. Please see the instructions for submission at the end of this document.*

Printing with threads (2nd Part)

In this part we will build on the code developed in laboratory session 5, namely on the way to simulate a slow printer. The main difference is that we will implement a client-server system composed by:

- A single-threaded client that accepts from the user a set of commands similar to the ones specified in the guide of laboratory session 5.
- A multi-threaded server that controls the simulated printer and will receive commands from one or more clients.

Client and server communicate through TCP sockets. A library to simplify the use of sockets is available.

Socket library

Please see the slides of October 21th´s lecture for an introduction to sockets and to the C library to be used in this assignment. The API implemented is sketched in the following table:

Operation	Signature	Returns
myServerSocket **(only for Server side)**	**int myServerSocket(int port)** creates a TCP socket in the port indicated	> O I/O channel associated to server socket; -1 in case of error
myAcceptServerSocket **(only for Server side)**	**int myAcceptServerSocket(int sock)** waits for a connection to the server socket; returns a new socket	> 0 I/O channel allowing access to the socket that is connected to the client; -1 in case of error
myConnectSocket **(only for Client side)**	**int myConnectSocket(char * host, int port number)** makes a connection to a server socket located in host *host* and port number *port*	> 0 I/O channel connection client to server; -1 in case of error
myWriteSocket **(client and server)**	**int myWriteSocket(int channel, char *buf, int nbytesToWrite)** writes to channel *channel nbytesToWrite* bytes starting at *buf*	> 0, number of bytes sent; -1 in case of error
myReadSocket **(client and server)**	**int myReadSocket(int channel, char *buf, int maxNbytesToRead)** reads from channel *channel* at most *maxNbytesToRead* starting at *buf*	> 0, number of bytes received; < 0, error; 0, peer has closed the channel
myCloseSocket **(client and server)**	**int myClosetSocket(int ch)** close channel *ch*	0 OK; -1 error

The socket library is implemented in files `mysocks.c` and `mysocks.h`. These files can be downloaded from CLIP in *Documentação de Apoio -> Outros*. In order to use the library you should

- Compile it `gcc –g –c mysocks.c`
- Add the following line to your code: `#include "mysocks.h"`
- Compile and link like this: `gcc –g –o server server.c mysocks.o`

Client

The command line for client invocation contains the host name and the port number where the print server is waiting for requests. An example is:

```
./myPrint localhost 5555
```

After initialization, *myPrint* will enter the following loop:

- Writes prompt
- Gets a line from the user
- Recognizes the following three commands:
  - *print filename*
    - connects to the print server
    - sends through the socket the text message "P filename"
    - waits for a reply from the server
    - displays the reply
    - closes the socket used
  - *status*
    - connects to the print server
    - sends a text message with "S"
    - waits for a reply from the server
    - displays the reply
    - closes the socket used
  - *quit* – terminates this program

Server

The print server will be started with two command line parameters: the pseudo terminal used for simulating the printer and the port number used for receiving client requests, like in

./printServer /dev/pts/2 5555

The print server will have two threads:

- One that waits for connections and parses messages received from clients
- One that will print files in the pseudo terminal (the printer)

The two threads communicate through a bounded buffer.

The server will start by creating and binding the server socket, initialize the bounded buffer and creating the printer thread. After initialization, the main thread will enter a loop waiting for client connections. After reading a message from the socket it will proceed according to the following:

- if the message read is "S" replies with a text message indicating how many files are waiting to be printed;
- if the message read contains "P filename":
  - tries to open *filename* for read
  - if this fails replies to client with an error message
  - if the open succeeds, deposits the result of the open operation in the bounded buffer; replies to the client indicating that the print request has been queued;
- for other messages, replies "Unknown request;"

Submission instructions

The work should be made by groups of two students. Each group definition is responsibility of its members. Individual work can only be accepted if is **first** authorized by the student's teacher. The work deliverables are:

- The source code for client and server programs. *Makefile*s and scripts needed to compile and test your work should also be included.
- A small report (in PDF format with no more than 4 pages) describing implementation options, assumptions made, and possible limitations of your system.

These should be archived in a ZIP-file named as "AAAAA-BBBBB.zip" were "AAAAA" and "BBBBB" are the student's numbers (AAAAA < BBBBB). Send this file by email to your lab instructor with the subject:

FSO - Trabalho 1: alunos XXXXX nº AAAAA e YYYYY nº BBBBB

(put your group colleague in Cc). The instructor will acknowledge the reception; if you don't receive a confirmation message after 48h contact your instructor.