Fundamentos de Sistemas de Operação

MIEI 2014/2015

Programming Assignment 1

Overview

A calendar service with threads: programming assignment 1.

Schedule and Rules

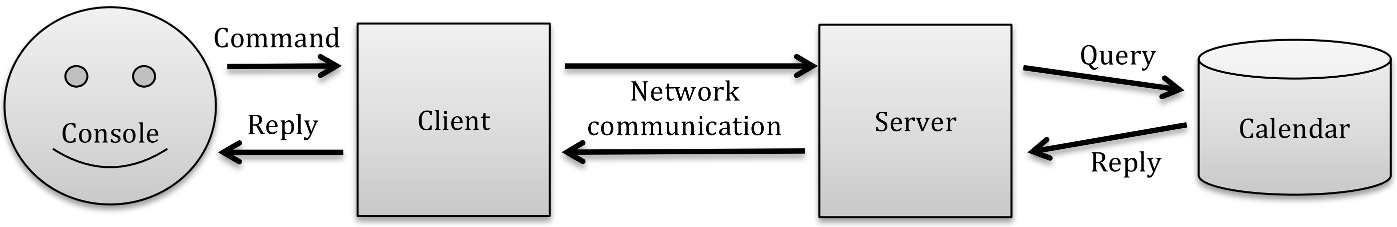
- Programming assignment 1 (PA1) will be developed in the classes of the week starting October 20th.
- The **deadline** for delivering PA1 (report and code) is November 10th..
- The instructions for the delivering process will be given via CLIP.
- The assignment must be done by groups of two students enrolled in the same laboratorial class. If this is not possible, both students must, at least, be enrolled in classes thought by the same professor.

A calendar service with threads

The main objective of this assignment is to build an application for the scheduling of events on a shared calendar. The application must also allow the user to inspect the properties of previously registered events, remove an event, and receive notifications of the events currently in progress. There is no notion of private event, each user may operate freely on events posted by himself or any other user.

Starting Point

The starting point is a two-program client-server application, where the *server* offers the calendar service and the client provides simple command line interpreter that serves as interface with the user.



The *server* program comprises two main functionalities: the management of a simple database that stores the current state of the calendar, and the handling of the requests received from the *client*. In turn, the *client* also embeds two main functionalities: the command line interpreter (console) and the communication with the server.

The initial code for both is available from CLIP, and comprises the following files:

- `server.c` – the server main file
- `client.c` – the client main file
- `calendar.c` and `calendar.h` – the data layer that manages the calendar database.
- `event.h` and `event.c` – the `Event` data-type that represents a calendar event and some event manipulation functions.
- `mysocks.h` and `mysocks.c` – a simple API for network communication.

To compile both the *client* and *server* programs just type

```
make
```

on your terminal. To run the application, first launch the *server* by simply typing

```
server
```

being that every time that you want to create a fresh (empty) calendar database, you have to use the `–c` option as:

```
server -c
```

Next, run the *client*, supplying the mandatorily user nickname:

```
client some_nickname
```

Other options are available for both the *server* and the *client*. You may try them if you want, but they are not essential for the completion of this assignment.

Step 1:

The command line interpreter provides a command `help` to obtain the list of supported commands, and their syntax:

```
> help
    add description day month year hour duration
        hour format: HHMM
        duration is given in minutes
        example: add test 2 10 2014 1805 30
    remove event_id
    list event_id
    listall
```

Of the listed commands, only `add` and `listall` are fully implemented. The objective of the assignment's step is to conclude the implementation of the other two: `remove` and `list`. For that purpose, you will have to complement the given partial implementation of the:

- command line interpreter – to build the request to send to the server
- client-side communication layer – to send the request to the server
- server-side communication layer – to receive the request, forward it to the calendar module, and send the reply back to the client

Step 2:

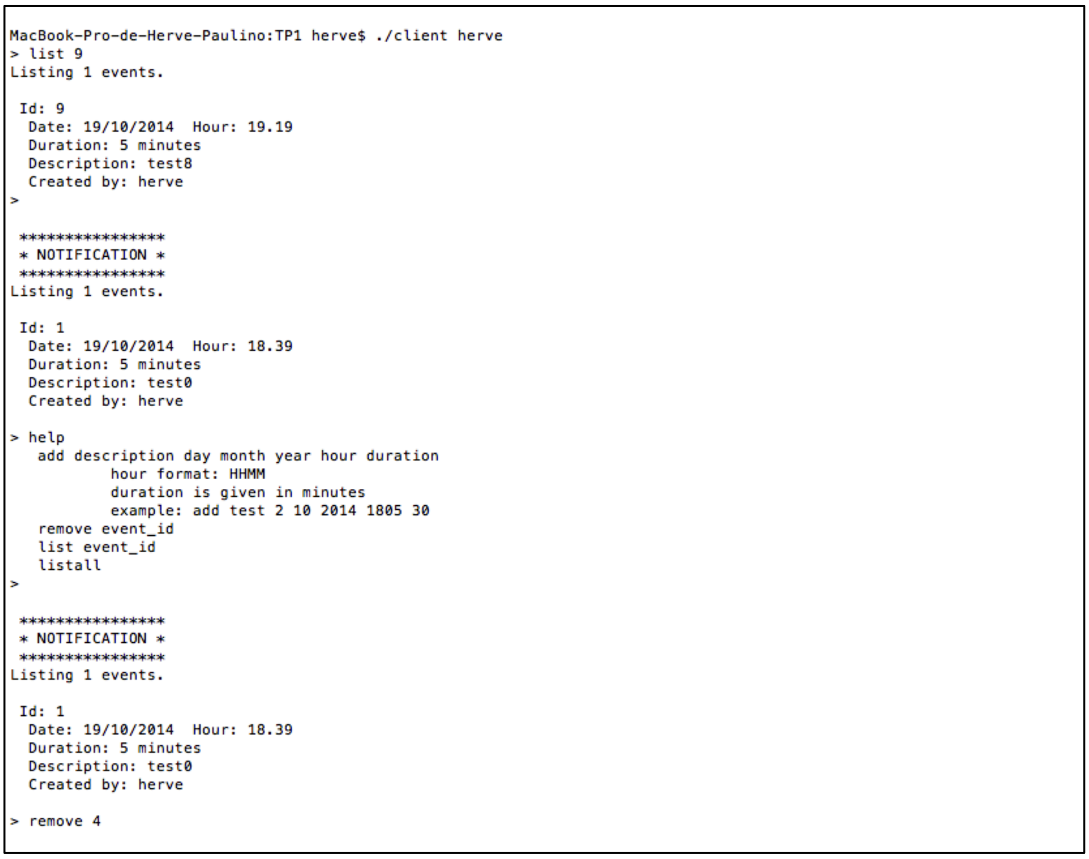
Launch the server and two (or more) clients, each on its own terminal. Then, issue a `listall` command in all clients. You will certainly observe that only one will receive a reply. The remainder will block until the one currently communicating with the calendar service terminated its execution (assess this fact experimentally). The reason for this behaviour is the fact that the *server* only attends one client connection at the time.

The objective of this second step is overcome this limitation by rendering the calendar server multithreaded, so that it can process multiple clients concurrently. **Suggestion:** there are several strategies to accomplish this behaviour, we suggest you to launch one thread per client connection. Also remember to guarantee exclusive access to the calendar database.

To test your solution, launch once again the server and two (or more) clients. All clients must now execute concurrently and observe the effects of the operations performed by the remainder.

Step 3:

The objective of the third, and final, step is to allow the client application to asynchronously receive notifications of events currently in progress. In other words, the command line interface must post information about incoming notifications at the same it allows the user to insert new commands. An example follows:


```
MacBook-Pro-de-Herve-Paulino:TP1 herve$ ./client herve
> list 9
Listing 1 events.

 Id: 9
  Date: 19/10/2014  Hour: 19.19
  Duration: 5 minutes
  Description: test8
  Created by: herve
>

 ****************
 * NOTIFICATION *
 ****************
Listing 1 events.

 Id: 1
  Date: 19/10/2014  Hour: 18.39
  Duration: 5 minutes
  Description: test0
  Created by: herve

> help
   add description day month year hour duration
           hour format: HHMM
           duration is given in minutes
           example: add test 2 10 2014 1805 30
   remove event_id
   list event_id
   listall
>

 ****************
 * NOTIFICATION *
 ****************
Listing 1 events.

 Id: 1
  Date: 19/10/2014  Hour: 18.39
  Duration: 5 minutes
  Description: test0
  Created by: herve

> remove 4
```

The client-server communication layer is equipped with an operation – LIST_ONGOING_EVENTS – for the client to receive the list of on-going events.

Suggestion: decouple the client's functionalities into two modules: the processing of the user commands and the communication with the server, each ran by a different thread. Use the `request` data structure (supplied to the console as argument) to communicate information between both threads.

Disclosure

The code supplied does not intend to be a fully developed calendar application. For instance, there are features provided by the calendar database that have not been made available to the user, such as event subscription, and not all user-given data is being validated.

The objectives of this assignment are clearly stated in this document. The addition of features not explicitly demanded in either of the three steps will not contribute to increase the assignment's grading.