Internet Programming

Programming Assignment 3: Remote Procedure Calls

Deadline: Friday 16 October 2015, midnight

☞ Make sure you **check the tips** at the end of this document (page 7) before starting working on your implementation.

1 A paper storage server

In the first part of this assignment, you will implement a *paper storage server* using Sun-RPC, as well as a command line client to interact with the server. The server and client could be used, e.g., to store and retrieve the articles to be presented at a conference.

1.1 Server implementation

The server does not take any command-line options. For simplicity, the server stores all papers *in main memory*. The program name of the server must be `paperserver`. The functionality of the server is described in Section 1.2, along with the client command-line options that trigger it.

1.2 Client implementation

The client program for the server must support a number of command-line options, described below. The command-line option parsing must be implemented using GNU `getopt`. The program name of the server must be `paperclient`. The functionality and command-line options that must be supported are as following:

- **Help:**

  ```
  $ paperclient
  ```

  or

  ```
  $ paperclient -h
  ```

  Displays a help message with the available command-line options.

- **Add an article:**

  ```
  $ paperclient dastardly.few.vu.nl -a 'Author Name' 'Paper Title' paper.pdf
  42
  ```

  Stores the article contained in file `paper.pdf` on the server running on `dastardly.few.vu.nl`. In addition to the contents of the file, the server must also store its author (“Author Name”) and title (“Paper Title”).

  When a new paper is stored, the server assigns a new paper number to it. Numbers must be serially assigned, starting from 0. If a paper with the same title and authors already exists, only the contents of the paper are updated. The paper number must remain the same.

The server must return the paper number to the client and the client must print that number (e.g., 42) to the standard output, followed by a single new line character. No additional information must be printed.

- **Fetch an article:**

```
$ paperclient dastardly.few.vu.nl -f 42
%PDF-1.5
%...
```

Fetches from the server the contents of the paper with the specified number (e.g., 42). The contents of the file are written to the standard output.

No assumptions must be made on the type of submitted papers. Your server and client must be able to store any type of file (e.g., .pdf, .doc, or whatever).

You can verify that your programs correctly store and retrieve files using the following commands. The checksums produced by the first and the last command must match. Otherwise, the file has changed at some point during transfer and storage.

```
$ md5sum paper.pdf
e12d74ccdcc54d7112862a90fdcc67f7  paper.pdf
$ paperclient dastardly.few.vu.nl -a 'Foo Bar' 'Baz' paper.pdf
42
$ paperclient dastardly.few.vu.nl -f 42 | md5sum
e12d74ccdcc54d7112862a90fdcc67f7  -
```

- **Get article information:**

```
$ paperclient dastardly.few.vu.nl -i 42
Foo Bar     Baz
```

Retrieves the authors/title information for the paper with the specified number (e.g., 42) from the server and outputs them to the standard output, followed by a single new line character. A tab character must be used to delimit the author and the title in the output.

- **Remove an article:**

```
$ paperclient dastardly.few.vu.nl -r 42
```

Removes the paper with the specified number (e.g., 42) from the server. If the number does not correspond to a paper, no action is taken by the server. The output of the client is undefined (i.e., not needed or checked).

- **List all articles:**

```
$ paperclient dastardly.few.vu.nl -l
1     S V    First Paper
2     M S    Second Paper
...
42    Foo Bar     Baz
```

Retrieves the information about all the papers stored on the server and outputs them to the standard output, one paper per line. For each paper its **number**, **author** and **title** must be displayed, delimited by tabs.

Your programs must support any number and any size of papers, up to the limits supported by the computer. Do not neglect doing **correct memory management**. Make sure you properly release both the memory used for storing the paper contents as well as any temporary buffers you use. You should use Valgrind to check your programs for memory leaks (see T.2) and avoid losing points for that.

1.3 Questions

Q-A When calling `paperclient <host> -l`, the server does not know in advance how many paper details will need to be delivered. How can one transfer *any number* of paper detail structures to the client?

Q-B When calling `paperclient -a` or `paperclient -f ...`, the transferred file can in principle be of any length. Explain *why* it is difficult to transfer strings of arbitrary size using Sun-RPC. How can you solve this problem?

2 A hotel reservation server

In the second part of this assignment, you will implement a hotel reservation server, as well as a command-line client to interact with the server. The server is meant to be used for making room reservations for the attendees of a conference. This means that **the reservation dates are fixed** and you don't need to deal with them. For the client-server communication, you must use Java-RMI.

2.1 Server implementation

The class name for the server must be `HotelServer`. The server does not take any command-line options. The functionality of the server is described in Section 2.2, along with the client command-line options that trigger it. The offered rooms must be as following:

- 10 rooms of type 1 at 150 euros per night
- 20 rooms of type 2 at 120 euros per night
- 20 rooms of type 3 at 100 euros per night

Do not start the Java RMI registry (`rmiregistry`) from within your program. This will prevent automated tests from executing.

2.2 Client implementation

The class name for the client must be `HotelClient`. The reservation client program must support a number of command-line options, described below. The command-line option parsing must be implemented using the Java port of GNU `getopt`.

To ensure compatibility with the testing environment, use the following JAR archive for Java `getopt`: `http://www.urbanophile.com/arenn/hacking/getopt/java-getopt-1.0.14.jar`

The functionality and command-line options that must be supported are as following[1]:

- **Help:**

  ```
  $ java HotelClient
  ```

  or

  ```
  $ java HotelClient -h
  ```

  Displays a help message with the available command-line options.

- **List available rooms:**

  ```
  $ java HotelClient dastardly.few.vu.nl -l
  10    20    20
  ```

  Lists the number of rooms available in each price range, separated by a tab and followed by a single new line character.

- **Book a room:**

  ```
  $ java HotelClient dastardly.few.vu.nl -b <type> 'Guest Name'
  ok
  ```

  Books a room of the specified type (if available) on the given name. On successful booking, `ok` must be printed to the standard output. If the specified type is fully booked, `NA` (i.e., Not Available) must be printed instead.

- **List guests:**

  ```
  $ java HotelClient dastardly.few.vu.nl -g
  ```

  Lists the names of all registered guests, one per line, with no additional spaces or empty lines. The order does not matter, any order is ok.

3 A hotel reservation gateway

It is important that the hotel server can communicate with client programs written in any programming language. To this end, in the final part of this assignment you have to build a gateway program which allows making room reservations through regular sockets.

The gateway program must be implemented in Java and listen for socket client connections on port 3333. The class name for it must be `HotelGateway`. The program uses a single command-line argument, to indicate to which hotel server it should connect:

```
$ java HotelGateway dastardly.few.vu.nl
```

A plain-text protocol must be used to communicate with the clients, as following:

- A `hotelgw>` prompt must be displayed to connecting clients.
- Several commands may be issued in a single session.
- The commands supported must be the same as those of the `HotelClient`.
- The format of the commands must be the same as described in Section 2.2.
- An additional `q` command must be supported which terminates the session.

[1] Any Java options (e.g., `-classpath`) are omitted for brevity.

This designs allows easy testing of the gateway using the `telnet` program as to test the gateway:

```
$ telnet dastardly.few.vu.nl 3333
Trying 127.0.0.1...
Connected to dastardly.few.vu.nl.
Escape character is '^]'.
hotelgw>h
blah blah blah
hotelgw>l
10    20    20
hotelgw>b 1 Scott Ian
ok
hotelgw>b 2 Joey B
ok
hotelgw>g
Scott Ian
Joey B
hotelgw>q
$
```

3.1 Questions

Q-C Which server structure did you use for implementing `HotelGateway`? Did you need to use any synchronization primitives to guarantee correctness?

Q-D If you were to implement a gateway for the paper server (see Section 1.1), would you be able to use a text-based protocol for client-gateway communication? How would you support transfer of, e.g., `.doc` files?

A Submission and Grading

A.1 Grading

Your final grade for each assignment will range from 0 to 100. Most assignments will include *optional sub-components* that make the assignment sum-up to a figure larger than 100. Solving the optional parts will only count towards compensating for any other errors.

Grading for the different parts of this assignment is as following: TBA

Item	**Points**	**Note**
`paperserver`	20	
`paperclient`	14	
`HotelServer`	16	
`HotelClient`	16	
`HotelGateway`	16	
Questions	16	
Total	**100**	

A.2 What to submit

You must create and submit a zip file (name doesn't matter) with your assignment. Unless otherwise required, **all files must be stored on the top directory** of the archive. Make sure you don't include unneeded files in your submission (object files, executables, class files, etc.).

You can use the following commands to create the zip file:

```
cd assignmentX
zip -r ~/assignmentX.zip *
```

The contents of the zip file should be as following:

- `Makefile` → A top-level Makefile containing rules that compile all the submitted programs. See Section T.4 for help. A `build` recipe must be provided that compiles everything. I.e., issuing the following command must result in all the required binaries being compiled:

  ```
  make build
  ```

- `paper/Makefile`, `paper/paperserver.x`, `paper/paperserver.c`, `paper/paperclient.c` → The programs described in Section 1 and a Makefile that builds them.

- `hotel/Makefile`, `hotel/HotelInterface.java`, `hotel/HotelServer.java`, `hotel/HotelClient.java`, `paper/HotelGateway.java` → The programs described in Section 2 and Section 3 and a Makefile that builds them.

- `report.pdf` or `report.txt` → A pdf or plain ASCII text document containing: (*a*) a *short* report (less than one page) on the implementation of this assignment, (*b*) your answers to the questions in this assignment.

A.3 How to submit

Please read the respective section in the first assignment.

A.4 Disclaimers

Make sure you follow the archive structure described above in A.2. Otherwise, automatic evaluation will not work and you will end up with a null grade.

You may submit your assignment multiple times. **Only the last submitted version will be taken into account for your grade.**

Passing all the automated tests does not mean your submission is perfect! **Submissions may also be subject to manual inspection.**

T Useful Tips

T.1 RPC in Ubuntu

In Ubuntu the portmapper is included in the rpcbind package:

```
$ sudo apt-get install rpcbind
```

Once you install it, the portmapper will be started automatically. You can check its status using `rpcinfo -p`. If you get an RPC Authentication Error while trying to start your server, start your portmapper without authentication using the following commands. You need to repeat these steps **every time you reboot** your machine.

```
$ sudo service portmap stop
$ sudo rpcbind -wi
```

T.2 Memory checking with Valgrind

Memory management in C is an important issue, especially when it comes to long-running programs. Debugging memory management issues is not an easy tasks. For this libraries and tools have been developed to make it easier. A very popular tool for this is the Valgrind debugger. Valgrind check your program as it runs and identifies location when and where memory leaks occur.

To install Valgrind on Ubuntu, run:

```
$ sudo apt-get install valgrind
```

To run your program under Valgrind, use a command line like this:

```
$ valgrind -v --leak-check=full --show-reachable=yes --track-origins=yes
  <my_program> <arg_1> ... <arg_n>
```

Valgrind will sporadically print a summary of the memory leaks as the program runs. E.g.:

```
==2805== LEAK SUMMARY:
==2805==    definitely lost: 9 bytes in 4 blocks
==2805==    indirectly lost: 0 bytes in 0 blocks
==2805==      possibly lost: 0 bytes in 0 blocks
==2805==    still reachable: 81 bytes in 1 blocks
==2805==         suppressed: 0 bytes in 0 blocks
```

Having memory which is *definitely lost* by your program, is a strong sign of a poor implementation. Such errors must be fixed first, as they are also the cause of many *indirectly lost* memory allocations. Ideally, when your program exits Valgrind should report all zeroes in the leak summary.

Valgrind also points you where the memory leak has happened. E.g.:

```
==2805== 162 bytes in 2 blocks are definitely lost in loss record 5 of 5
==2805==    at 0x4028308: malloc (vg_replace_malloc.c:273)
==2805==    by 0x8048C6A: f_gets(int, _IO_FILE*) (mysh1.c:153)
==2805==    by 0x8048D7D: cmd() (mysh1.c:179)
==2805==    by 0x8048E2F: main (mysh1.c:207)
```

This means that the return value of malloc() overwrote a pointer which hadn't been freed. The sequence of function calls that lead to this is also listed. In order to make sure that Valgrind can provide you this type of information, you should use the -g flag when compiling your code:

```
$ gcc -g assignment.c -o assignment
```

For the full list of memory checks provided by Valgrind, please refer to the Valgrind Memcheck Manual.

T.3 Catching errors early

Modern compilers will do their best trying to produce an output from your source. However, this doesn't necessarily guarantee that the produced output will do what you intended. For this, it is considered a good practice to tell your compiler to be stricter when parsing your code and give you extra warnings.

This is done with the -Wall flag. To make the compiler warn you about even more potential problems, add the -Wextra flag. But since you will quickly start ignoring warnings instead of fixing them, you should also add the -Werror flag which stops the compilation if there are any warnings at all. Finally, don't forget the -g flag which adds debugging symbols to the compiler output for use by debuggers like Valgrind or gdb.

To sum up, compiling your code like this could save you from many hours worth of debugging:

```
$ gcc -g -Wall -Wextra -Werror assignment.c -o assignment
```

T.4 Multi-directory Makefiles

If you need to build program across directories and there are no cross-directory dependencies, you can create (*a*) one Makefile per directory and (*b*) one top-level Makefile that wraps them. Following is a sample wrapper Makefile.

```
.PHONY: build all paper hotel clean

build: all

all: paper hotel

paper:
    make -C paper build

hotel:
    make -C hotel build

clean:
    make -C paper clean
    make -C hotel clean
```

Targets must be marked as .PHONY to have Makefile always invoke their associated recipes. The -C flags allows invoking Makefiles residing in another directory, without manually changing the working directory. See the make documentation for more information.