COL334 A2 Evaluation

@October 28, 2022

Introduction

Today, you would be simulating a server-client file exchange, similar to A2, but much simpler.

The goal of the simulation is for the client to request and receive 10 chunks from the server, and finally print them all to a file (in order). However, this time, there are no text files containing the chunks and no threading. We would be providing your server the required chunks. We essentially provide you with 2 functions, which can be called to get the chunks. They can be downloaded from this link.

You can choose the language of your choice (it is suggested that you work with the same language as your assignment submission so that the code can be reused). You are not allowed to use verbatim the code from other students' assignment submissions.

Functions given

From the link given, find the folder of your language and download Chunks.ext from there (tester files are given for your reference, but are not needed). The chunks.ext file has two functions:

- `get_hash(….)`: returns the hash of the string passed to it
- `get_chunk(….)`: takes a chunk_id as argument. It computes the hash for the corresponding chunk. With some probability on each invocation, it corrupts the actual chunk. It finally returns the (maybe corrupted) chunk along with the hash (of the actual chunk).

(The exact language specific declarations and instructions to call these functions are given in the next section).

Note that get_chunk should only be called by the server.

Simulation Flow

The high level simulation flow is as follows:

- There is one server (single threaded) and one client (single threaded)
- The client loops from i = 1 to 10
- Client requests chunk i (note that indexes are 1-10)
- Server calls get_hash() with i, and sends the chunk along with the hash to the client
- Client checks if the hash of the chunk received matched with the hash received (for computing the hash, only use the get_hash function provided)
  - if they match, the chunk is not corrupted. Client stores it and moves to the next chunk
  - if they don't match, client requests it again it receives the correct chunk
- When all chunks are received, client writes them in order to a file (naming convention given in Submission Instructions)

Calling the functions

python

```
def get_hash(s) # s is the string to be hashed
def get_chunk(id) # id ranges from 1-10. This is the id of chunk required
```

import these functions into your file using `from chunks import *`

java

put your .java outside the `util` folder (provided). Import Chunks from util. An object of the chunks class would have to be created. Please refer to the code below

```
// your .java file
import util.Chunks;

// other code
Chunks c = new Chunks();
c.get_chunk(id);
// after this function call, the corresponding hash and chunk are stored in
// c.hash and c.chunk variables respectively
String hash = c.get_hash(c.chunk); // compute hash of some string
```

cpp

import the chunks.h header file using `#include “chunks.h”`

```
// chunks.h
size_t get_hash(string s);
pair<string, size_t> get_chunk(int id);
```

c

import the chunks.h header file using `#include “chunks.h”`

```
// chunks.h
unsigned long get_hash(unsigned char *str);
void get_chunk(int id, unsigned long *hash, char* chunk);
```

Note that get_chunk requires both hash and chunk to be passed by pointers. There is no return. The function will update the values on these pointes.

Submission Instructions

Name your server and client files as `server.ext` and `client.ext` respectively. Also write a shell script that can be run to run the simulation (like in A2). Name it `run.sh`. Your shell file should print the output in a file named `output.txt`. This file should contain all the 10 chunks received, appended together. It should not

contain any other text. There should not be any other files or dependencies. We will run the server file first, then the client file, and check the output in the output.txt created.

All these files, along with the files provided by us, should be put in a folder named `<entry_number>_<firstname>_<lastname>`. There should be no subdirectories here for languages except java. For java, this folder should contain `util` (provided) and your files (outside util). Finally, zip the folder and name it is `<entry_number>_<firstname>_<lastname>.zip` and submit this zip file.

> ⚠️ No changes should be done to the files provided by us. They would be replaced before evaluating your submission.
