EECS 489 Lab 3: ImageDB with Bloom Filter

This assignment is due on Wed, 4 Feb 2015, 10 pm.

WARNING: Please note the change in SHA1 submission instructions.

Introduction

In terms of number of lines you need to write to complete this lab, it is very short. You only need to write about 9 lines of code. One line for Task 1, 8 lines for Task 2. The amount of time needed depends on how comfortable you are with modular arithmetic and bitwise operations.

We assume a client-server setup in this lab. The server (`imgdb`) will eventually be our distributed hash table (DHT) node, but in this lab, we assume there is only one such node.

```
% imgdb [ -b <beginID> -e <endID> ]
```

Upon start up, the server loads up its database with images from an "`images`" folder under the current working directory (where you run the server from). For each image, the server computes a SHA1 value from which an ID is derived. Only those images whose IDs fall within the range of the server's IDs will be loaded onto its database. When an image is loaded onto a database, it is also entered into a Bloom filter by computing three indices from the above SHA1 value. The function to load the database and populate the Bloom filter is provided to you in the support code under function `imgdb::loaddb()`. You should study this function carefully to see how to generate a SHA1 value from an image name and also how to generate an ID and populate the Bloom filter from the SHA1 value. By default, the range of the server is (0, 0], i.e., the full identifier ring. You can use the `-b` and `-e` command line options to set the begin and end values of the server's ID range.

The client (`netimg`) works similarly to the client in Lab1. It simply connects to the server specified with the `-s` command line option, and sends a query for an image, specified with the `-q` command line option. If the server has the image, it returns it to the client. Otherwise, it returns an `imsg_t` packet with the `im_found` set to `NETIMG_NFOUND` and the client simply prints out an appropriate error message. The full client code is provided as part of the support code. You don't have to write any client code.

Task 1

Your first task is the write the function `ID_inrange(ID, begin, end)` in `hash.cpp`. Given an ID, return true (1) if ID is between `begin` and `end` modulo `HASH_IDMAX+1`, defined in `hash.h`. For example, 147 is between (138, 150] but not between (150, 200], whereas 210 is between (200, 10]. This function is used by `imgdb::loaddb()`, so you can observe its working by modifying the server's ID range (using `-b` and `-e` command line options) and watching which images in the database are loaded. This task takes all but one line of code.

Task 2

Your second task is to complete the `imgdb::searchdb(imgname)` function in `imgdb.cpp`. You're asked to compute the SHA1 value of the given `imgname`. From the computed SHA1 value, you're asked to compute

the object ID and to determine if the image name is present in the bloom filter. You want to throughly understand how `imgdb::loaddb()` works before attempting this function. This task takes about 8 lines of code.

Assumptions

We make some assumptions for this and subsequent labs and for programming assignment 2.

- We assume an object ID size of 8 bits. To compute an object ID, we "fold up" a 160-bit SHA1 value into 8 bits. So the probability of IDs colliding become much higher. For the images, once we have a hit on the Bloom filter, we simply do a linear search of the database. A match requires matching both the image's ID and name, which also resolves any hashing collision (false positive) for us.
- We assume a fixed maximum size of the image database, `IMGDB_MAXDBSIZE`. Once this capacity is reached, we simply print out a message to inform the user that we're not adding more images, but the server continues to run otherwise.
- We assume that once loaded, images are never removed. So we don't have to worry about holes in the database or resetting the Bloom filter.
- We assume that only one image is read into memory at any one time. Each time there is a search hit, the image will be read from file. Please be sure to use the `ltga.cpp` provided with this support code and not the one from earlier labs as this one does memory management to allow for reuse of the `LTGA` object.

Support Code

Since Lab3 support code contains solution to parts of PA1, it will be made available only to those who have turned in PA1. To those who have turned in PA1, the support code will be available as lab3.tgz in the Course Folder by the end of day Friday, 1/30, after PA1's due date. At that time, if your `pa1` folder is not empty, we will assume that you have turned in your PA1, your write permission to the folder will be revoked, and you will be given read permission to the Lab3 support code. If you want to use some late days on your PA1, please leave your `pa1` folder empty. You will not have access to the Lab3 support code until you have turned in your PA1. If you've decided **not** to turn in PA1, please email sugih and you will be given access.

You can also find `refimgdb` (the one from Lab1 has been renamed `refimgdb-lab1`), `refnetimg`, and an `images` folder in the usual course `FILES` folder. If you'd like to download the images to your own computer, you can grab images.tgz (22.5 MB). As usual both `refnetimg` and `refimgdb` were compiled on CAEN's GNU/Linux, so don't try to run them on Debian, Ubuntu, Mac OS X, or Windows machines. Recall that the complete source code for `netimg` in included in the support code, so you should be able to build the client on your local platform. The support code has been built and tested on Linux, Mac OS X, and Windows. If you're not using the provided `Makefile`, note that `imgdb.cpp` must be compiled with the compiler option `-DLAB3` for the `main()` function to be included.

Windows specific note:

- On Windows, you'd need to install the OpenSSL library to build `imgdb`. I've updated the Windows section of the *Building Socket Program* course note with links and instructions to install and use the OpenSSL library.

- Since we're reading both from files and from sockets, we can no longer simply redefine `close()` to `closesocket()`. Instead, you'd have to use each one specifically, or you can use the provided `socks.cpp:socks_close()`.

Testing Your Code

Run `imgdb` without any command line option. Run `netimg` to connect to the running `imgdb` and request for `ShipatSea.tga`. The image should be served and displayed. Now run:

```
% imgdb -b 220 -e 20
```

You should see "`*in range*`" printed next to the name of each image whose ID is within your `imgdb`'s ID range.

Next run `netimg` to connect to the running `imgdb` and request for `ShipatSeat.tga`. Assuming the ID you compute for `ShipatSea.tga` is outside the `(220, 20]` range, you should get an

```
imgdb: ShipatSea.tga: Bloom filter miss.
```

message on server side and

```
netimg: ShipatSea.tga image not found.
```

on the client side. Test for other boundary conditions.

Submission Instructions

Test your compilation! Your submission must compile **without** errors.

Your "*Lab3 files*" comprises your `hash.cpp` and `imgdb.cpp` files.

To turn in your Lab3:

1. Submit the SHA1's of your *Lab3 files* on the CTools Assignments page. (If the URL doesn't work for you, just click the "Assignments" item on the left menu of the CTools page for EECS 489.) Once you've submitted your SHA1's, don't make any more changes to the files, or your SHA1's will become invalid.
2. Upload your *Lab3 files* by pointing your web browser to Course folder and navigate to your `lab3` folder under your uniqname. Or you can `scp` the files to your `lab3` folder on IFS: `/afs/umich.edu/class/eecs487/w15/FOLDERS/<uniqname>/lab3/`.
   This path is accessible from any machine you've logged into using your ITCS (`umich.edu`) password. Please report any problems to ITCS.
3. Keep your own backup copy! Don't make any more changes to the files once you've submitted your final SHA1's.

The timestamp of your SHA1 submission on CTools' Assignments page will be your time of submission. If this is past the deadline, your submission will be considered late. You are allowed multiple "submissions" without late-policy implications as long as you respect the deadline. CTools keeps only your last submission.

Do NOT turn in an archival (`.zip` or `.tgz`) file, instead please turn in your solution files individually.

Turn in ONLY the files you have modified. Do not turn in support code we provided that you haven't modified. Do not turn in any binary files (object files, executables, or images) with your assignment.

Do remove all `printf()`'s or `cout`'s and `cerr`'s and any other logging statements you've added for debugging purposes. You should debug using a debugger, not with `printf()`'s. If we can't understand the output of your code, you will get zero point.