RPC-based Proxy Server Web Cache

The end goal of this project is to write the components of a proxy-server web cache and to experiment with various cache policies. Along the way, you will learn curl's C interface and program remote procedure calls (RPC) with the rpcgen tool.

Contents

1 Setup
2 Collaboration
3 RPC Tasks
3.1 RPC Client
3.2 RPC Server and libcurl
4 Cache Policy Implementations
5 Cache Policy Experiments
6 Grading

Setup

1. Gain access to a well-maintained linux machine. If you do not have this already, then it is recommended that you follow the instructions for downloading a installing VirtualBox and the AOS Virtual Image.
2. Download the proxy-server project starter code proxy-server.tar.gz
3. Check that that rpcbind daemon is running on your system with

   ps -e | grep rpcbind
4. Check that libcurl is installed with

   which curl-config

Collaboration

For this and all projects, students must work on their own.

RPC Tasks

RPC Client

Your first task is to complete a toy program using rpcgen. Start by defining the desired interface in the file proxy_rpc.x. For this assignment, we need a single procedure called httpget_1. The file is written in a special RPC language and is compiled into C code that handles the client-server connection. Consult the rpcge Programming Guide for details. Running

```
rpcgen proxy_rpc.x
```

generates three files

- proxy_rpc_clnt.c — the code for the client side
- proxy_rpc_svr.c — the code for the server side
- proxy_rpc.h — a header file useful to both None of files should be modified. They are intended to be linked with other files, as we will do in the remainder o exercise.

Next, write a client program that makes a remote procedure call to httpget_1 and prints the string that is returned. Detailed instructions can be found in client.c you will find the rpcgen Programming Guide useful. Compile your code with

```
rpcgen proxy_rpc.x
make client_test client
```

Then run

```
sudo ./client_test
```

in one terminal window (depending on how rpcbind is setup you may or may not need sudo), and

```
./client localhost example.com
```

in another. The window where you ran the client program should print "SUCCESS!"

Files for submission

- proxy_rpc.x
- client.c

Support files:

- Makefile
- client_test.c

RPC Server and libcurl

Now we consider the proxy server. Inside the file proxy.c you need to define the httpget_1 procedure with the correct signature. Again, the rpcgen Programming will be useful.

To actually get the contents of the requested url, you will use libcurl. The libcurl project site contains more information and more examples. A particularly usefu example will be getinmemory.c. The Makefile already links proxy.c to the libcurl library. Replacing client_test with proxy in the instructions for the rpc client abo print of the data returned from the provided url in the client's window.

Files for submission:

- proxy.c

Support files:

- Makefile

Cache Policy Implementations

Next, you will implement four different replacement policies for the webcache.

- Least Recently Used (LRU) — remove the item that has been requested least recently.
- Random — remove an item chosen uniformly at random from the cache.
- Least Frequently Used (LFU) — remove the item that has been used the fewest times from the cache.
- LRUMIN — Group the cache entries by size on a base-2 logarithmic scale and remove entries from the group closest to the size needed. Precise specificatio can be found in the comments of lrumin.c

Write your code for these policies in the files lru.c, rnd.c, lfu.c, and lrumin.c, respectively, all of which implement the API given in gtcache.h. All implementations be efficient. Inefficiency will result in points lost.

To help you implement these policies, the following data structures are provided.

- hstbl.[ch] — contains a string-indexed hashtable to help you quickly retrieve data from the cache.
- indexminpq.[ch] — contains a indexed minimum priority queue[link] generic data structure to help you quickly figure out which item to replace in the cach
- indexrndq.[ch] — contains an indexed randomized queue supporting constant-time arbitrary deletion.
- steque.[ch] — contains a steque data structure which might be useful for keeping track of available ids.

If you need to record the time a request is made, use the gettimeofday method in sys/time.h. While testing you code, you not want to actually use curl and make web request. The file webdata.txt contains a list of 95 urls and the content lengths of the responses and the files fake_www.[ch] contain some utilities for simula web requests. You may find these helpful as you write your testcode.

Files for submission:

- lru.c
- rnd.c
- lfu.c
- lrumin.c

Support files:

- Makefile
- gtcache.h
- hshtable.[ch]
- indexminpq.[ch]
- indexrndq.[ch]
- steque.[ch]

Cache Policy Experiments

No one cache policy is best for all request patterns. To illustrate this point, you are to provide lists of requests that affect the hit rates of the cache algorithms in particular ways.

The program cache_perform.c will be used to evaluate your data over cache sizes between 1 and 16 MB. Compile your code with

```
make perform
```

which will produce lru_perform, rnd_perform, lfu_perform, and lrumin_perform executables. If you list of requests is in a file requests.txt, you can then you can t performance of your algorithms by passing the file through stdin. For example, to see the performance of lru on requests.txt with 2000 requests, one would run

```
./lru_perform webdata.txt 2000 < requests.txt
```

The requests.txt file should contain one url per line. When the end of the list is reached cache_perform starts over until 2000 requests have been made.

We will say that one policy “beats” another if it outperforms it on a majority of the cache sizes between 1 and 16MB that we test. For simplicity of grading, your d will be run against our implementations of these policies. You are to submit four request pattern files. The requirements are indicated by the filename.

Files for submission:

- rnd_beats_lru.txt
- lfu_beats_lru.txt
- lru_beats_lfu.txt
- lrumin_beats_lru.txt

Support files:

- Makefile
- cache_perform.c
- gtcache.h

Grading

Deliverable	Percentage
client.c, proxy_rpc.x	15%
proxy.c	15%
lru.c	10%
rnd.c	8%
lfu.c	10%
lrumin.c	10%
rnd_beats_lru.txt	8%
lfu_beats_lru.txt	8%
lru_beats_lfu.txt	8%
lrumin_beats_lru.txt	8%