Project 2: File Caching Proxy

15-440/640 Spring 2022

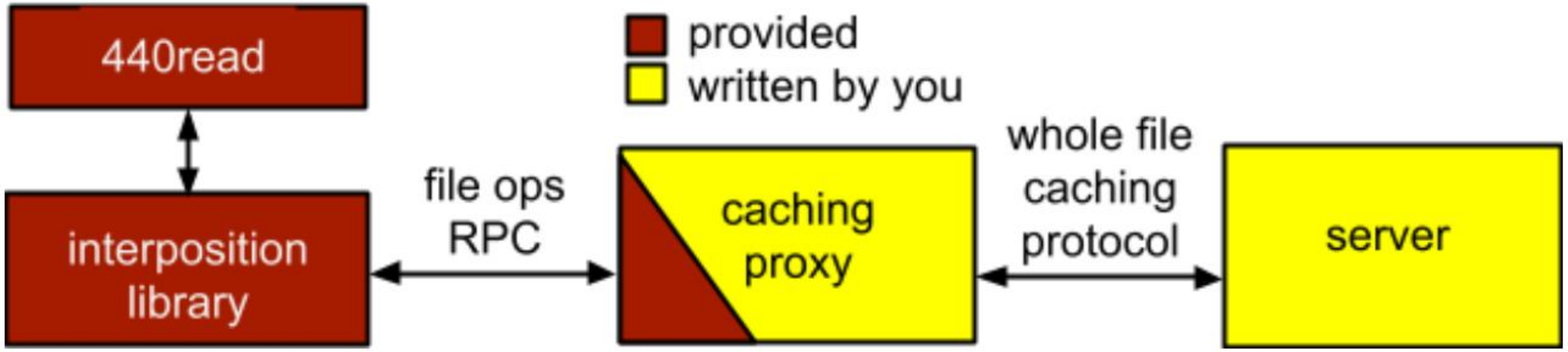
February 8

Main Task:

- Extend system similar to Project 1, adding a caching layer
  - A new proxy layer will sit between the client library and the server
  - It will provide whole-file caching with LRU replacement
  - Multiple clients may connect to each proxy, and multiple proxies to the server
  - Need to support: `open, close, read, write, lseek, unlink`

What we provide

- We will provide 440-tools, and a working interposition library
- We will provide the RPC server stubs for the proxy (RPCreceiver class)
- You need to write the proxy and server in Java (not C)
- You will design your caching protocol and RPCs between proxy and server



You don’t need to write low-level RPC code!!

- The low-level implementation of RPC and serialization between client and proxy is already provided
- You just need to add the functions to handle each operation
- Remember to set “proxyport15440=xxx” and “pin15440=xxx” environment variables for both the client and the proxy. [The pin is a 9-digit number of your choice and needs to match on client and proxy]
- Between Proxy and Server, build RPC using Java RMI
  - This makes it easy to call methods on remote objects, with syntax like local method calls
- Remember that the Java RMI registry address and port are command line arguments to your proxy. The port is also an argument to your server which is expected to start a Java RMI registry on that port.

RPCreceiver class

- Java class file we provide that implements an RPC server and stubs to handle the client requests
- You need to provide the actual service functions
- You do this by creating a class that implements FileHandling interface, and a “factory” class that can create new instances of this class
- RPCreceiver will create a new instance of your class for each client that connects, and uses the methods you define to perform file operations
- Note that the file operations are the C/POSIX ones, just like in Project 1; these can be different from the way files are handled in Java

Caching requirements

- Your proxy will keep cache contents in a cache folder
- It must ensure the total data here is kept below the specified cache size – so eviction, using an LRU policy is needed
- The proxy and server must implement whole file caching
- Check-on-open style for invalidation is fine
- The system should provide some form of session semantics (i.e., open-close semantics, AFS v1)
  - Key idea: between open and close of a file by a particular client, the client will see the file as it exists at open (and any changes it makes); changes by other clients won't be seen; no one else sees its changes until close
  - Should allow many to read from file, even as others make changes; you can restrict to single writer if you wish, but concurrent reads should not block

Checkpoint 1: Due Tues. February 15, 2022

- No server, just proxy and clients
- Proxy's cache directory will be pre-populated with files
- Just serve these to the clients (e.g., just like Project 1 server)
- Goal here is to ensure your file operation functions are implemented correctly
- No actual cache stuff (no eviction, no invalidation, etc.)
- Should handle concurrent clients

Checkpoint 2: Due Tues. February 22, 2022

- Adds Server, and makes Proxy implement a whole-file cache
- Read caching of files should work, and push of updates back to server
- Interactions between writes and reads will not be tested
- Actual cache limits will not be enforced; eviction, LRU, or even invalidation are not needed yet
- However, server should be able to handle more than one concurrent proxy (each with multiple clients)

Checkpoint 3: Due Thurs. March 3, 2022

- Full working implementation
- All operations must work
- Evict to keep cache size within limits, using an LRU replacement policy
- Must ensure fresh data is seen by clients (subject to session semantics)
- Must be able to handle concurrent readers and writers
- Don’t forget your design document!

Logistics

- Project should be written entirely in Java (ver 8 to ver 11)
- Must use a 64-bit Linux environment (e.g., unix.andrew.cmu.edu); the provided interposition library will not work on Mac/Windows!!!!
- Do not hardcode IP addresses / ports in your code; read these from environment variables instead
- Use different ports to avoid colliding with your fellow students
- Be careful – your server and proxy can potentially allow anyone on the Internet to read and write files in your account!!!! DON'T LEAVE THEM RUNNING!!!
- Same submission policies on Autolab as in Project 1