Carnegie Mellon University

Recitation 4

15-440/640 - Distributed Systems S22

P2 Overview, Java Basics, Common Pitfalls of P2

Lily He

Chengkai Li

February 9th, 2022

Agenda

- Java Basics
- P2 Overview
- Common Pitfalls of P2

Java Basics: Interface

- Defines set of abstract methods to implement a class

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}
```

```
// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Timeline

Problem Set 1:

~~Out:~~	~~Thursday **February 3, 2022**, 11:59 PM EST~~
Due:	Thursday **February 10, 2022**, 11:59 PM EST

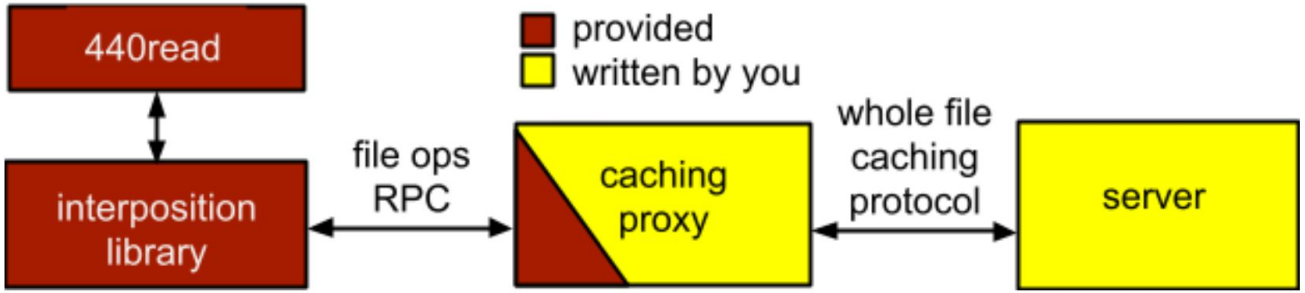
P2:

Checkpoint 1 Due:	Tuesday **February 15, 2022**, 11:59 PM EST
Checkpoint 2 Due:	Tuesday **February 22, 2022**, 11:59 PM EST
Final Due:	Tuesday **March 3, 2022**, 11:59 PM EST

Project 2 Overview

Not only a implementation but also a design problem

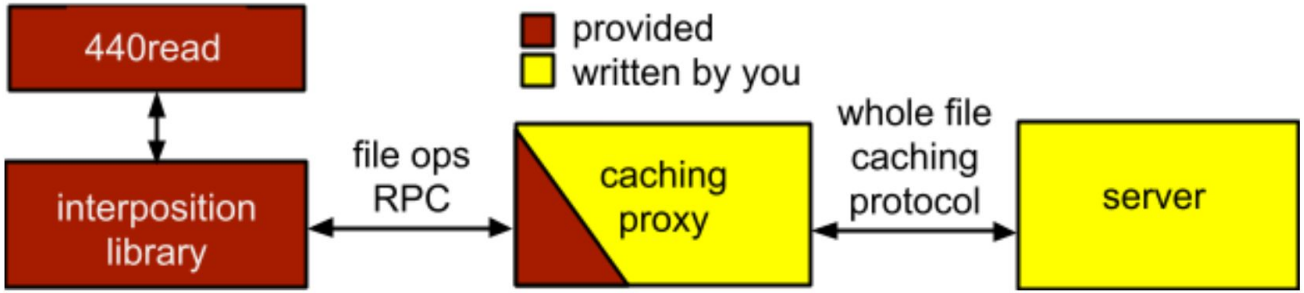
- Design the protocol between the proxy and the server
- Maintain the open-close session semantics
- Whole-file caching with LRU replacement policy
- Handle concurrent read/write requests



Project 2 Overview

Difference / Relationship with Project 1

- Still implementing RPCs for open, read, write, close, etc
- But with a middle layer cache proxy
- Using Java instead of C, need manually handle error cases
- Using Java RMI to handle parameter serialization



Checkpoint 1

- More like implementing Project 1 with Java
- Only implement the proxy
- No server needed for this checkpoint
  - Proxy act as a server
- No need to implement the low-level connection or parameter serialization
- Implement the client-facing aspects of the proxy
  - open, read, write, close, etc
  - Return expected outputs and error codes
  - Be careful about the Java and C semantics
- Proxy should be able to handle concurrent clients

Java Basics: FileHandling Class

- Defines constants and abstract functions to perform file operations in a C compatible manner
- Documentation found in handout's doc folder

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method
void	**clientdone**()
int	**close**(int fd)
long	**lseek**(int fd, long pos, **FileHandling.LseekOption** o)
int	**open**(java.lang.String path, **FileHandling.OpenOption** o)
long	**read**(int fd, byte[] buf)
int	**unlink**(java.lang.String path)
long	**write**(int fd, byte[] buf)

Checkpoint 1

- Access file:
  - C: file descriptor(int), Java: RandomAccessFile (Object)
- Errors:
  - C: return -1, set errno, Java: maybe exception
  - In this project, return -errno.

```
/**
 * Handle open() request from client
 * @param path: path of target file
 * @param o: open option
 * @return file descriptor when succeed, error code when failed
 */
public int open(String path, OpenOption o) {
}
```

Java Basics: Exceptions & Errors

- Exceptions disrupt the normal flow of the program
- Exceptions should be caught whenever thrown

```
try {
    // Some code
} catch (IOException e1) {
    // Some error handling
    e1.printStackTrace();
} catch (Exception e2) {
    // Some error handling
    e2.printStackTrace();
}
```

- Your code should provide return values / errors according to ***C file semantics***
- Errors.*xxx* can be found here

Java Basics: File & RandomAccessFile Class

File Class:

- Abstract representation of files and directory pathname
- File objects are immutable

```
File myFile = new File(String pathname);
```

RandomAccessFile Class:

- Random access file: a large array of bytes stored in the file system with a file pointer “indexing” into the array.
- Can apply read/write operations on the file

```
RandomAccessFile randomFile = new RandomAccessFile(File myFile, String modes);
```

Java Basics: File & RandomAccessFile Class

- Assign appropriate file descriptors to new files
  - Take a look at the “static” keyword

Checkpoint 2

- Design the protocol between the proxy and the server
- Proxy should read cache entries and push modifications to the server
- When cache miss, fetch file from the server
- Assuming cache entries are always valid
- No cache size limitation
- Server should be able to handle concurrent proxies

Checkpoint 3

Full implementation of all the required features

- Everything in checkpoint 2
- Ensure cache freshness by checking cache validity
- Maintain open-close session semantics
- Implement a cache with LRU replacement policy
- Correctness should be the first priority but performance will also be tested
- Documentation (10 points)
  - A 1-2 page report describing your major design choices
- Code style (10 points)
  - Same as project 1

Good Practice

- **Start early!! Ck2 and ck3 are far more complicated than ck1!**
- **Read the whole writeup and starter code before writing your code**
- Read previous Piazza posts
  - To give you insight on what direction to take
  - Someone might have encountered the same issue you are facing now
- Read documentation
  - Online documentation (Java RMI, File, RandomAccessFile Classes…)
  - Project documentation in doc/allclasses-frame.html
- If the code has a bug…
  - **Don’t use autolab as a debugging tool**
  - Test locally before submission (Piazza post @364 and recitation 3)
  - Post on Piazza or go to office hours

Common Pitfalls

- Malicious paths: ../foo
  - Should not open files outside server's root directory
- Redundant path: ./A/foo & ./A/B/../foo
  - Refer to Path.normalize()
- Huge files: chunking
- Race conditions: Java synchronized or lock
  - Trade off between safety and performance
  - But be careful about deadlocks
  - By default, assuming Java classes are not thread safe
- File versions:
  - Do not simply use timestamps, conflicts within concurrent clients

Common Pitfalls

- Differences between Java and C semantics
  - Java throws exception during runtime error
  - C returns invalid values and set error codes
  - Return values/ error codes according to C semantics
- Do not forget to set the same pin15440 environment variable both on the server and the client side!

Makefile and Test

Makefile is relatively simple for project 2

```
all: Proxy.class # And other .class files you want to compile

%.class: %.java
	javac $<

clean:
	rm -f *.class
```

Makefile and Test

Run and test the program locally

```
export CLASSPATH=$PWD:$PWD/../lib
export proxyport15440=15440      #Set your own value
export pin15440=123456789        #Set your own value

# command line arguments: <serverip> <port> <cachedir> <cachesize>
java Proxy 127.0.0.1 11122 /tmp/cache 100000

# command line arguments: <port> <rootdir>
java Server 11122 fileroot

# test client command
LD_PRELOAD=../lib/lib440lib.so ../tools/440read foo
```

Java RMI: Remote Interface

- Remote interface extends `java.rmi.Remote` and declares set of remote methods
  - Remote methods report failures through `java.rmi.RemoteException`

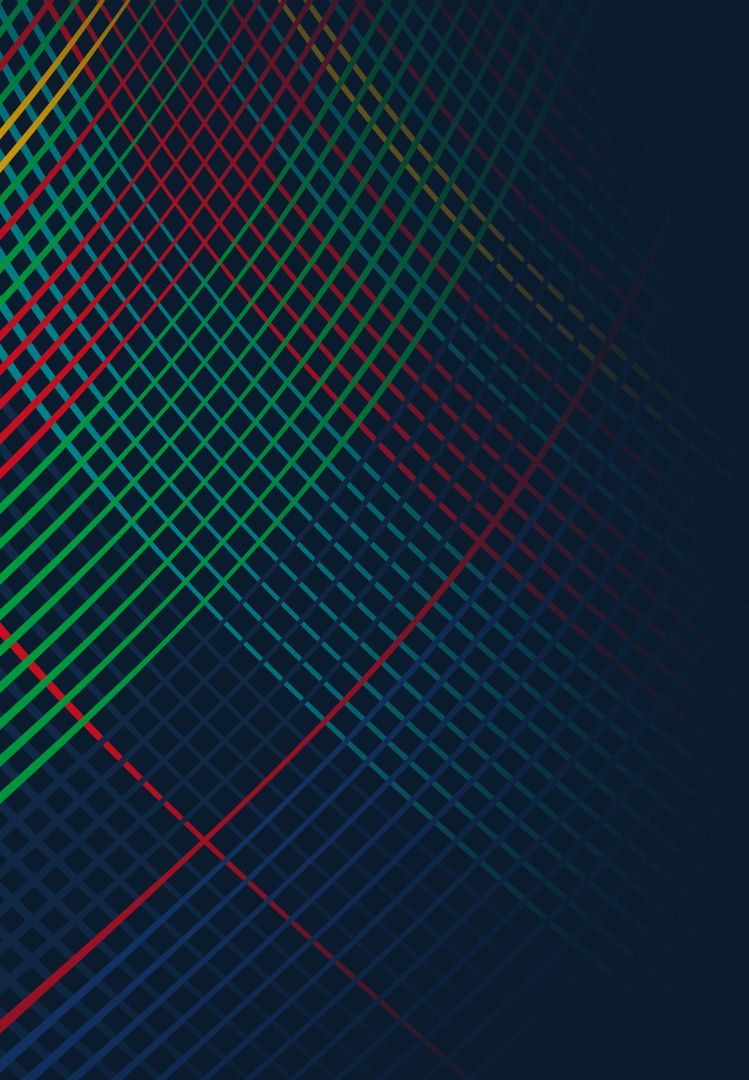
```
package example.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

Java RMI: Server Registry & Client

```java
public class Server implements Hello {
    public Server() {}

    public String sayHello() {
        return "Oi Oi Oi";
    }

    public static void main(String args[]) {
        // Create a registry that listens on port myPort
        LocalRegistry.createRegistry(myPort);

        // Create new RMI Server object
        Server server = new Server();

        // Specify URL of registry, including port
        Naming.rebind(String.format("//127.0.0.1:%d/Server", myPort), server);
    }
}
```

```java
public class Client {
    private Client() {}

    public static void main(String[] args) {
        String url = "//" + args[0] + ":" + args[1] + "/Server";
        Server s = (Server) Naming.lookup(url);
        System.out.println(s.sayHello());
    }
}
```

Q & A



Thanks!