Observability and Protocols

CSE 132

Today's Outline

- Observability – smart debugging
- Protocols – network communication

Observability

- What is really going on?
- Option 1: stare at the code until inspired
  - When that doesn't work, make random change
- Option 2: don't assume the code you actually wrote does what you think it does!
  - Alter code so that you discover what it really does
  - PCS system is great for this
  - What about network code?
    - Is issue on client or server?

Observability for Network Code

- Need to know what is really going across the network
- On server, client, or maybe both:
  - Display what is going out the output stream
  - Display what is coming in the input stream
  - Support multiple interpretations of the raw data
- You can build these tools
  - Do a good job and it will help you the rest of the semester!

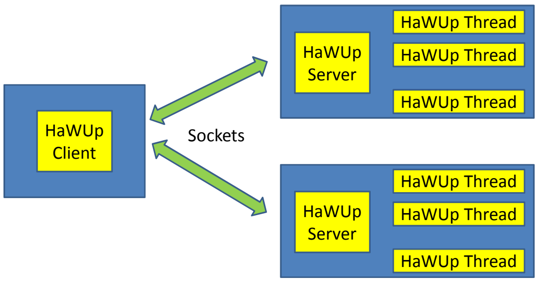
Observability Tools

- One for InputStream and one for OutputStream
- Extend FilterInputStream (and its counterpart) as ViewInputStream
- ViewInputStream's read() method should:
  - read() from the provided InputStream
  - Display the character(s) (in a specifiable form)
    - As an ASCII character
    - As a decimal value (0 to 255)
    - As a hexadecimal value (0x00 to 0xff)
    - As an integer (collecting 4 bytes before displaying)
  - Control format via constructor (or something else)

Protocols

- How do we build a networked HaWUp protocol?
  - What things get said (in both directions)?
  - How do we say those things?
- At least 3 things to think about:
  - Efficient use of network bandwidth (verbose), vs. Efficient use of programmer's time
  - Uninformative response(s) in the case of an error
  - Poor recovery from client mistake

Networked HaWUp



Commands (Client to Server)

- Which job – sleep, sum, crack (as a UTF8 string)
- Sleep
  - Duration (max) – 4-byte integer (in ms)
  - # of sleepers – 4-byte integer
- Sum
  - Range of numbers – 2 4-byte integers – bottom and top
  - # of workers – 4-byte integer
- Crack
  - # of workers – 4-byte integer
  - Range to try – 2 4-byte integers
  - Public key – string

Replies (Server to Client)

- Sleep
  - Return value (time) - integer
- Sum
  - Return value (sum) - integer
- Crack
  - Return value (private key) - string
- Error returns?
  - First send state ("success", "error")
  - Then send return value or error code

Individual Data Elements

- Byte – basic network element
  - writeByte(), readByte() in Data(Input/Output)Stream
- Character – two bytes in Java
  - writeChar(), readChar(), high byte first
- Short Integer – two bytes – bits can be anything from 0x0000 to 0xffff
  - writeShort(), readShort()
- Integer – four bytes in Java – value -2^{31} to $2^{31}-1$
  - writeInt(), readInt(), most significant byte (MSB) first

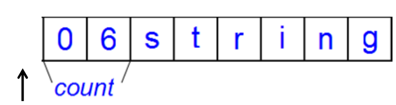
Strings

- Not just a sequence of two-byte characters!
- Network communication is language agnostic, so must acknowledge that others do things in different ways
- UTF-8 is common character encoding
- String is
  - 2-byte length (of bytes in string), followed by
  - Characters in UTF-8 encoding
  - Supported by writeUTF(), readUTF()

Good Protocols

- What might I do about the issues from before?

1. Use clear command names
2. Provide return error codes or strings
3. Start all msgs with magic number



Insert '!' (ASCII 33) as magic number up front

Command protocol

- Sleep
  - Magic number (?)
  - Command “sleep”
  - # of sleepers (4 bytes)
  - # max duration (4 bytes, ms)
- Sum
  - Magic number
  - Command “sum”
  - # of workers (4 bytes)
  - Low end of range (4 bytes)
  - High end of range (4 bytes)

Crack

- Crack
  - Magic number
  - Command “crack”
  - # of workers
  - Low end of range
  - High end of range
  - Public key (string)