Computer Communications

CSE 132

1

2

Today's Outline

- Communicating between PC and Arduino
  - Java on PC (either Windows or Mac)
- Streams in Java
- Protocol Design
- Observability
- Information Representation – beyond numbers

3

Computer Communications

- Link that provides byte-level data delivery
  - Network
  - Serial port
- Ability to send and receive on each endpoint
- Must use a protocol to understand anything other than individual bytes
  - Individual data elements (ints, chars, strings, etc.)
  - Higher-level, application-specific messages
    - The user just pressed button "X"
    - The pressure in vessel X is Y psi at time Z
- Needs to work across platforms
  - E.g., Java on PC and C on Arduino

4

Java Communications uses Streams

- Upstream writer, downstream reader

- Source writes to stream
- Destination reads from stream
- Either endpoint might be a file or some other input/output device, e.g.,
  - Dest. could be Arduino connected via serial port
  - Source could be the keyboard

5

Stream Conventions

- FIFO ordering (First-In-First-Out)
- Protocol must be same at both ends of stream for effective communication to take place
  - Stream of bytes? chars? integers? what is a char?
- Properties supported by streams that "wrap" other streams, e.g.,

```
InputStream in = new InputStream(…);
BufferedStream dataIn = new BufferedStream(in);
```

- Watch video for examples of user input streams

6

Wrapping Streams

- A stream can take another stream as a parameter to its constructor
- The outer stream adds functionality to the wrapped stream
- E.g.,

  SerialComm sc = new SerialComm(new SerialPort(…));

- This is called "decorator" pattern
- We will use SerialComm and SerialPort in upcoming studios and assignments – starting with 4

7

Communications in Java

- Open COM port with SerialPort object
  - Use SerialPort class, which we provide
  - Works in Windows and Mac
- Wrap SerialPort with a SerialComm object
  - You will write SerialComm class (some of it, anyway)
  - What "properties" does SerialComm provide?
    - Fixes up a data type inconvenience on input bytes
    - Most importantly, provides a debugging capability

8

Back to Communications

- Streams are sequences of bytes
- We need data at a higher level of abstraction
  - Integers
  - Floats, Doubles
  - Characters
  - Strings
  - More
- Protocols must be designed to enable this
  - Build bigger things out of streams of bytes

9

Individual Data Elements

- Byte – basic network element
  - writeByte(), readByte() in SerialComm class
  - Serial.read(), Serial.write() in Arduino C
- Character
  - Two bytes in Java
  - One byte in C
- Integer
  - Four bytes in Java
  - Two bytes in C

10

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
    - On PC in Java, use the debugger!
    - Or use System.out.print() to display on console
    - On Arduino in C, use Serial.print()

11

Observability in Communications

- Need to know what is really going across the communication link
- On sender and receiver:
  - Display what is going out the output stream
  - Display what is coming in the input stream
  - Show the raw data (sequence of bytes)
- You will build these tools
  - This is the primary purpose of SerialComm class

12

Information Representation

- We've covered integers
  - Including 2's complement
- We've covered reals (OK, their approximation)
  - Including fixed point and Q notation
  - Including floating point
- But there are many other types of information

14

Text – Characters and Strings

- ASCII – American Standard Code for Information Interchange
  - 7-bit codes representing basic Latin characters and numbers [A-Z, a-z, 0-9], some common punctuation, and control characters
  - There are a number of extensions to 8 bits, but only the 7-bit codes really standard.
- Unicode – 8- or 16-bit codes extending to a much wider set of languages
  - The first 128 codes are equivalent to the 7-bit ASCII standard

15

C Strings

- Strings are sequences of ASCII characters, stored one byte per character (8 bits), terminated by a NULL (zero) character
- E.g., "Hello!"

01001000	'H'	0x48
01100101	'e'	0x65
01101100	'l'	0x6c
01101100	'l'	0x6c
01101111	'o'	0x6f
00100001	'!'	0x21
00000000	NULL	0x00

16

ASCII Facts

- Numerical digits are assigned in order of increasing value

  i.e., '0' = 0x30
  '1' = 0x31
  '2' = 0x32

  '9' = 0x39

- For single character, value conversion is simply a difference of 0x30

17

More ASCII Facts

- Letters are also assigned in lexicographical order:

  'A' = 0x41
  'B' = 0x42

  'Z' = 0x5a

  'a' = 0x61
  'b' = 0x62

  'z' = 0x7a

- Upper/lower case conversion is simply a difference of 0x20

18

Still More ASCII Facts

- First 32 characters (0-0x1f) are control codes:

  0x00 ^@ null (C string terminator)
  0x07 ^G bell
  0x0a ^J line feed (or newline)
  0x0c ^L form feed
  0x0d ^M carriage return

19

Line breaks are not standardized

- End of line conventions differ by operating system:
  - In MS Windows: 0x0a, 0x0d is end of line
  - In Unix/Linux: 0x0a is end of line
  - 0x0a, linefeed, is sometimes called 'newline'
- In C, '\n' is mapped to OS end of line termination convention

20

Java Strings

- Strings are represented via the class "String"
- String objects are immutable
- The character encoding is system specific, e.g., either UTF-8 or UTF-16 (typical).
- The length is an instance variable in the object (in most implementations)
- The characters are stored in a char[] array (again, in most implementations)

21

Unicode

- Standard for character representation
  - Supports wide variety of languages, symbols
- UTF-8
  - Variable length code with 8-bit code units
  - U+0000 to U+007F are the same as ASCII
- UTF-16
  - Uses 16-bit code units, also variable length
  - Latin + Greek + Cyrillic + Coptic + Armenian + Hebrew + Arabic + Syrian + Tāna + N'Ko fit in 16 bits
- UTF-32
  - Uses 32-bit code units, fixed length

22

Communicating Strings

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

23

Images

- Consider the following bits:

0x002400081881423c

0000 0000 0010 0100 0000 0000 0000 1000

0001 1000 1000 0001 0100 0010 0011 1100

- Make 1 dark and 0 light:

24

Images

- Arrange in rows, one byte per row:

- Each bit is a "pixel" in the image

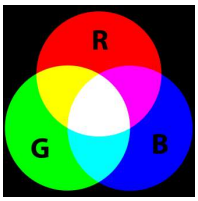
25

Add color and more pixels

26

Color

- Additive color – primaries Red, Green, Blue



- Position close together and put diffuser above
  - This builds one pixel

27

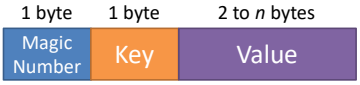
Protocol Design

- What do we want to communicate?
- How do we want to say it?

28

A Protocol for Us

Message format:

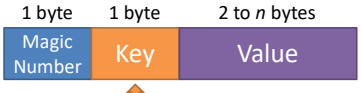


- Magic number is anchor of message
- Always first byte
- Unlikely in rest of message
- Reader can ignore bytes until it sees magic number and then receive

29

A Protocol for Us

Message format:

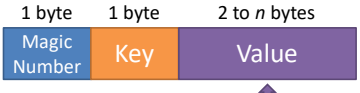


- Key tells what type of message
- Indicates both size and interpretation
- E.g., 2-byte temperature value
- E.g., 4-byte timestamp
- E.g., UTF-8 encoded error string
- Table of legal keys must be maintained

30

A Protocol for Us

Message format:



- Actual content of message
- Key tells how to interpret

31

Logistics

- Exam 1 is coming – Feb 19 (2 weeks from today)
  - Next week's lecture will include review for exam
  - Review will include boundaries on scope
  - Communication is *not* included (saved for exam 2)
  - Information representation *is* included
    - E.g., strings in Java and in C

32