

# Homework 1

## Layered Half-duplex Communication Software

Netprog 2002 Homework 1

1

---

---

---

---

---

---

---

---

## Layer 1: byte transmission

- Layer 1 is the lowest level layer.
- Provides transmission of a single byte (8 bits).
- You don't write this layer – it is provided for you.

```
int ll_write(char b);  
int ll_read(char *b);
```

Netprog 2002 Homework 1

2

---

---

---

---

---

---

---

---

## Layer 2: Messages

- Layer 2 provides transmission of *messages*.
- A *message* is a sequence of bytes.
- The only way layer 2 can send or receive anything is by using layer 1!  
It must send the message 1 byte at a time,  
read the message 1 byte at a time.

Netprog 2002 Homework 1

3

---

---

---

---

---

---

---

---

## Layer 2 (continued)

- You must write these functions:

```
int l2_write(char *b, int n);  
int l2_read(char *b, int max);
```

- `l2_write` will use `l1_write`. `l2_read` will use `l1_read()`.

---

---

---

---

---

---

---

---

## More on layer 2

- Your two layer 2 functions depend on a peer-to-peer protocol.
- You must come up with the protocol.
  
- The protocol defines how the reader knows when it has the message.

---

---

---

---

---

---

---

---

## Layer 3: error detection

- Layer 3 provides transmission of messages with *error detection*.
- Suppose there are random errors in the transmission of individual bits – the receiver won't get the same message that the sender sent.
  - We need to detect this situation.

---

---

---

---

---

---

---

---

## Error Detection

- It's impossible to detect all possible transmission errors!
- The assumption is that there are *random* errors.
  
- We could send the entire message twice and compare the received messages.

---

---

---

---

---

---

---

---

## Better Error Detection

- A more efficient approach is to compute a *checksum* from the message.
- Send the checksum along with the message.
- The receiver computes a checksum from the message received, and compares to the checksum sent.
- If there is a match, there *probably* were not any errors.

---

---

---

---

---

---

---

---

## Checksum

- Treat each byte as a number.
- Add up all the bytes in the message.
- Send the sum modulo 256 (so it fits in one byte).
  
- Larger checksums are more accurate (can detect errors better).

---

---

---

---

---

---

---

---

## Layer 3 Functions

- Prototypes look just like layer 2:  
`int l3_write(char *b, int n);`  
`int l3_read(char *b, int max);`
- `l3_write` will use `l2_write`. `l3_read` will use `l2_read()`.

---

---

---

---

---

---

---

---

## Layer 4: named values

- Layer 4 transmits a *named value*.
  - Needs to transmit a name.
  - needs to transmit a value.
- The receiving end gets a name and a value.

---

---

---

---

---

---

---

---

## What are names and values?

- As far as layer 4 is concerned, names and values are just sequences of bytes.
- The application layer needs to worry about how to interpret the bytes, but layer 4 just needs to send/receive them.

---

---

---

---

---

---

---

---

## Layer 4 sending function

```
int l4_write(char *name, int namelen,  
            char *val, int vallen);
```

- Layer 4 should use layer 3 to send the name and the value.
  - could use `l3_write` twice
  - could use `l3_write` once (combine the name and value somehow).

Netprog 2002 Homework 1

13

---

---

---

---

---

---

---

---

## Layer 4 receiving function

```
int l4_read(char *name, int *namelenptr,  
            char *val, int *vallenptr);
```

- `namelenptr` is a pointer to an int:
  - initially the value is the size of the buffer `name`.
  - when `l4_read` returns, it must indicate the size of the name (in bytes).
- `vallenptr` works the same way.

Netprog 2002 Homework 1

14

---

---

---

---

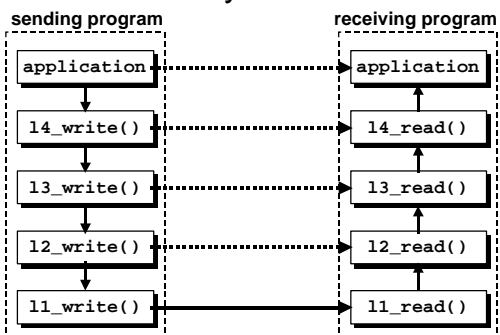
---

---

---

---

## HW1 Layers in action



Netprog 2002 Homework 1

15

---

---

---

---

---

---

---

---

## Sample Test Code

- A sample layer 1 that will work with a Unix Pipe.
- A main() that can act as a sender or receiver.
  - sender sends a *username* and *password*.
  - receiver reads to named values, hoping to get a *username* and *password*.

---

---

---

---

---

---

---

---

## Code you need to write

- 3 separate files – one for each layer you need to write:

`12.c 13.c 14.c`

- To compile with the sample testing code (named `hw1.c`):

```
gcc -o hw1 hw1.c 12.c 13.c 14.c
```

---

---

---

---

---

---

---

---

## Running sample code

- You need to run `hw1` twice:
  - once as a sender. It needs to be told a *username* and *password* to send.
  - once as a receiver.
- If you supply two command line arguments to `hw1`, it will assume it should be a sender. Otherwise it becomes a receiver.

---

---

---

---

---

---

---

---

## Command Line to test HW1

- To see what the sender will *send*:  
`./hw1 dave blah`
- To connect to a receiving hw1:  
`./hw1 dave blah | ./hw1`

---

---

---

---

---

---

---

---

## Homework 1 Submission

- Send email to [netprog-submit@cs.rpi.edu](mailto:netprog-submit@cs.rpi.edu).
- Subject line should be "1" (nothing else!)
- Attach a file named "README" that includes your name and some other information (details in the project description).
- Attach l2.c, l3.c and l4.c
- Don't send executables or object code!

---

---

---

---

---

---

---

---