Project 3

Unreliable Datagrams

Robert Escriva

Slide heritage: Previous TAs → Krzysztof Ostrowski

Cornell CS 4411, September 27, 2010

Announcements

- Project 2 due Wednesday at 11:59PM.
- Project 1 will be returned (with feedback) before then.
- Web page is being updated frequently; check for updates.
- Email `cs4410staff@systems.cs.cornell.edu` for help.

The real hero of programming is the one who writes negative code.

Douglas McIlroy, Cornell ’54

1. Project Scope

2. Implementation details
   - Using the networking pseudo-device
   - Interrupts
   - Miniports
     - An Example

3. Concluding Thoughts (Grading)

What are do unreliable datagrams involve?

- Simulate (parts of) UDP/IP
- Build a datagram networking stack.
- Use the pseudo-network interface `network.h` for “IP”.
- Using ports to identify endpoints.
- A minimessage layer for thread I/O.

The Interface

```
void minimsg_initialize();
miniport_t miniport_local_create();
miniport_t miniport_remote_create(
           network_address_t addr, int id);
void miniport_destroy(miniport_t miniport);
int minimsg_send(miniport_t local,
                 miniport_t remote,
                 minimsg_t msg, int len);
int minimsg_receive(miniport_t local,
                    miniport_t* remote,
                    minimsg_t msg, int *len);
```

Overview

The networking device should be treated as the IP layer of your system.

It transparently enables communication between other systems running minithreads.

- `network5.c`
- `network6.c`

Networking is interrupt-driven

- `network_initialize()` installs the handler.
- Should be initialized after `clock_initialize` and before interrupts.
- The prototype/behavior is similar to the clock interrupt.
- Each packet triggers an interrupt.
- Interrupts are delivered on the current thread's stack.
- This should finish as soon as possible!

network_handler

```
typedef struct {
    // sender
    network_address_t addr;
    // hdr+data
    char buffer[MAX_NETWORK_PKT_SIZE];
    // size
    int size;
} network_interrupt_arg_t;
```

The header and the data are joined in the buffer; you must strip it off.

Networking Functions

```
int network_send_pkt(
        network_address_t dest_address,
        int hdr_len, char* hdr,
        int data_len, char* data);
```

- Header contains information about the sender and receiver.
- As small as possible

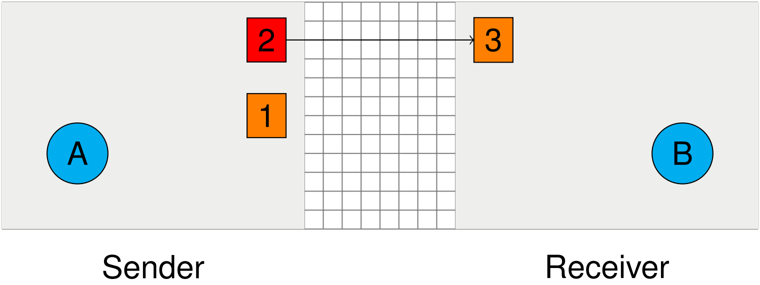
Overview

A miniport is a datastructure that represents an endpoint.

- Local ports are unbound ports; they may be used for listening and can receive from any remote port.
- Remote ports are bound ports; they make replies possible.

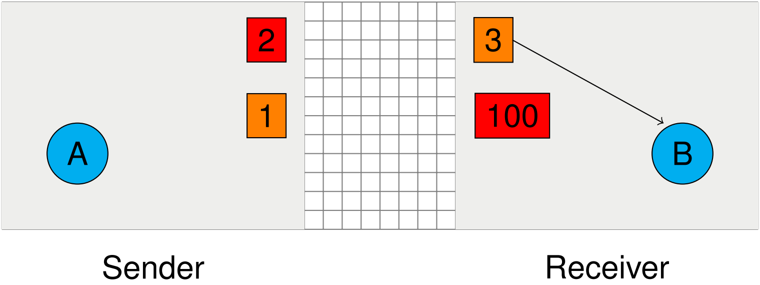
A sends from port 1 to port 3

- Local Ports: 1, 3
- Remote Ports: 2
- Threads: A, B



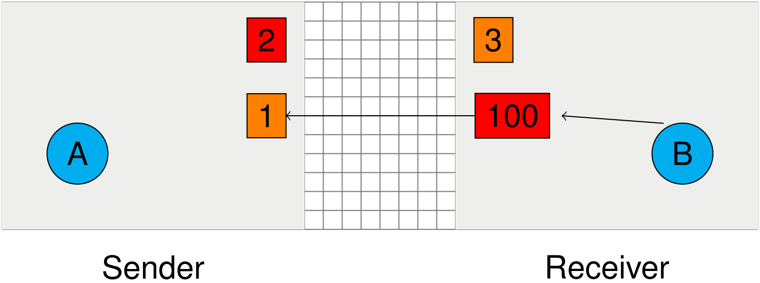
Minithreads creates port 100 and delivers message

- The port 100 is created in order to allow B to respond.
- The message is delivered to local 3.
- B is unblocked.



B responds to A over the new remote port.

- B received a reference to port 100.
- B can send to 100.
- The message will be sent to 1 (A).



What does the datastructure look like?

Conceptually it looks like this*:

```
struct miniport {
    char port type;
    int port number;

    queue_t data;
    semaphore_t lock;
    semaphore_t ready;

    network_address_t remote_addr;
    int remote_port;
    int remote_is_local;
}
```

*the next slide should be referenced when implementing

You should use unions

Unions store two overlapping datastructures[†].

```
union {
    struct {
        queue_t data;
        semaphore_t lock;
        semaphore_t ready;
    } loc;
    struct {
        network_address_t addr;
        int portno;
    } rem;
} u;
```

[†] You should use this to replace the last 6 variables from the struct on the previous page

Implementation hints - Local communication

- `miniport_destroy` will be called by the receiver.
- `miniport_send` sends data to the “remote port”.
- Remote ports can refer to a local port.

Implementation hints - Miniports

- Identified by a 16-bit unsigned number (the actual datatype is bigger).
- Assign successive numbers (even if the port closes).
- Local miniports are 0-32767.
- Remote miniports are 32768-65535.

Minimsg Layer

- The sender assembles a header that identifies the end points of communication.
- The receiver parses the header to identify the destination, enqueue the packet, and wake up any sleeping threads.

Minimsg Functions

```
int minimsg_send(miniport_t local,
                 miniport_t remote,
                 minimsg_t msg, int len);
```

- Non-blocking (i.e. doesn't wait for the send to succeed).
- Sends data using `network_send_pkt()`.

```
int minimsg_receive(miniport_t local,
                    miniport_t* remote,
                    minimsg_t msg, int *len);
```

- Blocks until a message is received.
- Provides remote port so a reply may be sent.

Grading

- Include the address of the sender in the header (used in Project 5).
- Port operations must be $O(1)$.
- Do not waste resources.
- Make sure to not reassign ports that are in-use.
- The application destroys remote miniports.
- We will be grading you on your implementation and test cases.