Project 3

Networking I: Unreliable Datagrams

Zhiyuan Teo

Slide heritage: Previous TAs → Robert Escriva

Cornell CS 4411, October 07, 2011
Updated October 08, 2011

Announcements

- Project 3 will be out this evening, due 11.59pm Sunday, 16 Oct.
- Project 2 grading and concerns.
- Project 1 regrading is still in progress.
- Some partners may be reshuffled again; affected groups will be informed by e-mail.
- See course staff if you encountered serious difficulty with projects 1 or 2.

Other announcements

- Next week: Supplementary lecture 3.
- No office hours or lecture on the week of 24-28 Oct. E-mail course staff if you have any questions on project 4.
- You are strongly encouraged to attend lectures and office hours.
- Don't rely solely on project documentation; refer to project slides too.

A quick primer on networking

- We will defer the OSI layer discussion to the 4410 lecture.
- What are datagrams*?
- Reliable vs unreliable network services.
- Architectural considerations in networking.

*Sometimes loosely interchanged with the term ‘packet’.

Project Scope

- Build a UDP/IP networking stack.
- Use `network.h` for “raw IP interface”.
- Build UDP abstractions: use ports to identify endpoints.
- A minimessage layer for thread I/O.

The Interface

```
void minimsg_initialize();
miniport_t miniport_create_unbound(int port);
miniport_t miniport_create_bound(
           network_address_t addr, int port);
void miniport_destroy(miniport_t miniport);
int minimsg_send(miniport_t local_unbound,
                 miniport_t local_bound,
                 minimsg_t msg, int len);
int minimsg_receive(miniport_t local_unbound,
                    miniport_t* new_local_bounc
                    minimsg_t msg, int *len);
```

Overview

The networking device should be treated as a raw IP interface; it sends byte packets.

Your minimsg layer enables communication between other systems running minithreads.

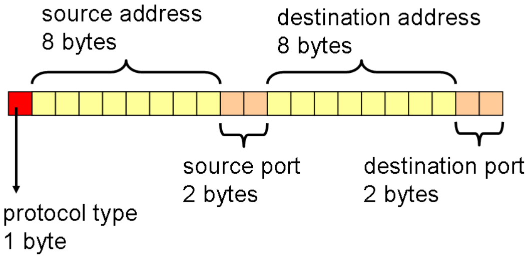
- `network5.c`
- `network6.c`

Sending datagrams

```
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.
- Header has a fixed format and length; this makes communications with your friends' Minithreads possible.

Header format



port numbers are in network byte order.

The header contains 5 fields packed back-to-back and is exactly 21 bytes long.

Big vs little endian

Different hardware architectures store integers differently.

Big-endian (SPARC, DLX, etc):

`32 bitvalue: 0x12345678`

12	34	56	78

`16 bitvalue: 0xdead`

de	ad

Little-endian (Intel, VAX, etc):

`32 bitvalue: 0x12345678`

78	56	34	12

`16 bitvalue: 0xdead`

ad	de

Header generation

- Use the provided `miniheader` functions.
- Pack source and destination addresses using `pack_address(char* buf, network_address_t address)`.
- Pack source and destination ports using `pack_unsigned_short(char* buf, unsigned short val)`.
- Set the protocol field to `PROTOCOL_MINIDATAGRAM` for this project.
- A pointer to the miniheader can be directly supplied to the `network_send_pkt()`, since the struct is correctly formatted in memory.†

† Padding is not an issue here since all fields in the struct are chars.

Header generation example

```
mini_header_t hdr =
    (mini_header_t)malloc(
    sizeof(struct mini_header));

/* pack fields here... */
pack_unsigned_short(hdr->source_port,
    local_unbound_port);

/* more packing here... */

network_send_pkt(dest_address,
    sizeof(struct mini_header),
    (char*) hdr, data_len, data);
```

Receiving datagrams

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

network_handler

```c
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.

Stripping the header

- You can't return the packet to the user as-is because of the header.
- Copy the header from the byte buffer into a `struct mini_header`.
- Read the protocol field.
- Use `unpack_address(char* buf, network_address_t address)` to extract the source and destination addresses.
- Use `unpack_unsigned_short(char* buf)` to extract port numbers back to host order.

Header stripping example

```
mini_header_t hdr =
    (mini_header_t)malloc(
    sizeof(struct mini_header));

memcpy(hdr, buf,
    sizeof(struct mini_header));

/* unpack fields here... */
source_port =
    unpack_unsigned_short(
    hdr->source_port);

/* more unpacking here... */
```

Miniports

- Why ports?
  - Multiplexing: different threads may want to use the network simultaneously.
  - Abstraction: communication with pipe-like semantics.
  - Isolation: a communication channel should not be aware of data in other channels.
- A miniport is a data structure that represents a one-way communication endpoint.

Two way communications with one-way endpoints

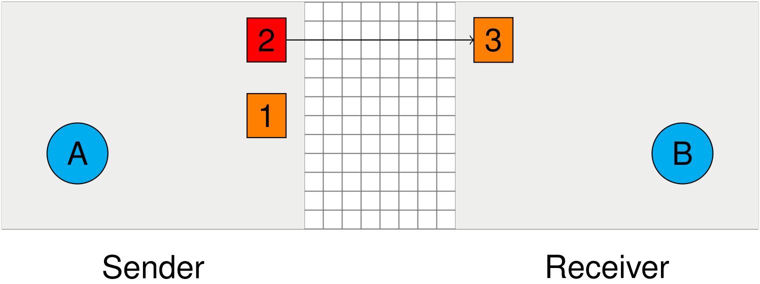
- How would you communicate with someone using disposable one-way cellphones each? (Assume that person doesn't know you.)
  - Call that person at his cellphone number (a 'magic' number you know).
  - Tell him the number to call you back at, then proceed to talk to him about other things.
  - The person calls you back through the cellphone number you provided and he gives you a reply.

Port binding

- A port is said to be bound if the remote end has assumed a fixed identity.
  - identity = (network address, port)
- Ports for receiving data are unbound.
  - We do not fix the identity of the remote end, so any (network address, port) can send to it.
  - Typically, the receiving port is some well-known number.
- Ports for sending data are bound.
  - Sending to this port will result in some (network address, port) receiving the data.

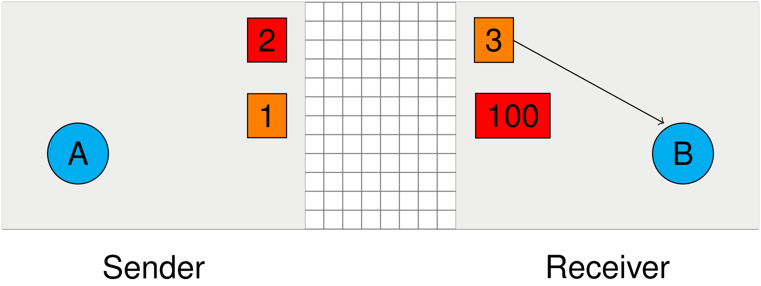
A sends from its port 2 to B's port 3

- Unbound (listening) Ports: 1, 3
- Bound (used for sending) Ports: 2
- Threads: A, B



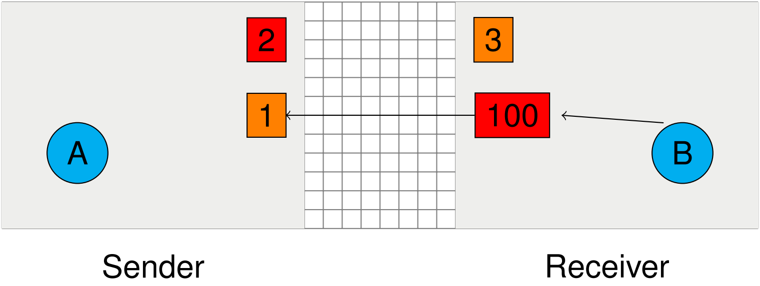
Minimsg layer creates bound port 100 and delivers message

- The bound (used for sending) port 100 is created in order to allow B to respond.
- A's message is delivered to B's unbound (listening) port 3.
- B is unblocked.



B responds to A over the new bound port.

- B receives a reference to its bound (used for sending) port 100.
- B can send to 100.
- The message will be sent to A's unbound (used for listening) port 1.



What does the data structure look like?

Conceptually it looks like this[‡]:

```
struct miniport {
    enum port_type type_of_port;
    int port_number;

    queue_t incoming_data;
    semaphore_t mutex_lock;
    semaphore_t datagrams_ready;

    network_address_t remote_addr;
    int remote_port;
}
```

[‡]the next slide should be referenced when implementing.

You should use unions

Unions store multiple overlapping datastructures[§].

```
union {
    struct {
        queue_t data;
        semaphore_t mutex_lock;
        semaphore_t datagrams_ready;
    } unbound;
    struct {
        network_address_t addr;
        int remote_port;
    } bound;
} u;
```

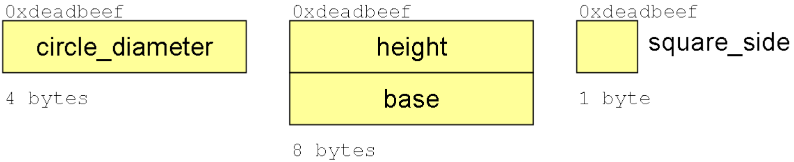
[§] You should use this to replace the last 5 variables from the struct on the previous page.

How unions work

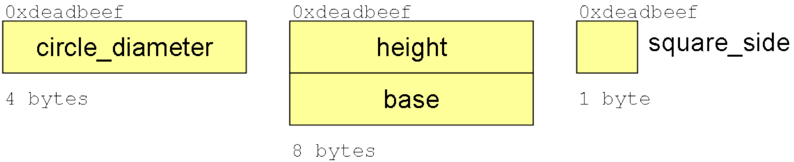
```
union {
   int circle_diameter;

   struct {
      int height;
      int base;
   } triangle;

   char square_side;

} shape_u;

shape_u shape;   // &shape == 0xdeadbeef
```

Each distinct data field defined in a union maps to the same starting address.

Distinct data fields are laid 'on top' of each other and thus share memory.



How unions work



All fields in the union share the same memory, so the size of the union is the size of the largest field.

```
sizeof(shape_u) == MAX(sizeof(int), sizeof(struct triangle),
                       sizeof(char))

                == MAX(4, 8, 1)

                == 8 bytes
```

How unions work

Modifying any of the individual union fields will change the value of other fields. (Assume big endian for this discussion).

1

```
shape.circle_diameter = 5;
```

0	0	0	5	?	?	?	?

2

```
shape.square_side = 1;
```

1	0	0	5	?	?	?	?

3

```
shape.triangle.base = 3;
```

1	0	0	5	0	0	0	3

How unions work

Distinct data fields are laid ‘on top’ of each other and thus share memory.

1	0	0	5	0	0	0	3

```
shape.circle_diameter == 0x01000005
```

Moral of the story: you better remember which subset of the union you used...

Back to Miniports

- You can embed a union in a struct.
- Only members within the union will share memory; other struct members are distinct.
- Use the `enum port_type` to decide which subset of the union to use.

Implementation specs - Minimsg

- `miniport_destroy` will be called by the receiver.
- `miniport_send` sends data through a bound port.
- You can also talk to yourself on the same machine!

Implementation specs - Miniports

- Identified by a 16-bit unsigned integer (the actual datatype is bigger).
- Unbound miniports are 0-32767 and can be chosen by the user.
- Bound miniports are 32768-65535 and are assigned in incremental order (even if the port closes).

Minimsg Layer

- The sender assembles a header that identifies the end points of communication.
- The receiver strips the header to identify the destination, enqueues the packet, and wakes 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 onto the IP interface 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

- 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.

Advice

- Come for office hours to test your implementation against the TAs' implementation.
- Ask questions over e-mail or in person.
- Enjoy Fall break... but start early!