Session #5: Network Programming

© Doug Abbott 2009 to 2013 | UCSD Extension Online Learning Course: Embedded Linux

Linux Device Drivers

- Devices treated like files
  - Everything in Linux is a file
  - Device files in /dev/...
- Device Classes
  - Character
  - Block
  - Pipe
  - Network

Low-level System Calls

```
int open (const char *path, int oflags);
size_t read (int file_des, void *data, size_t len);
size_t write (int file_des, const void *data, size_t len);
int close (int file_des);
int ioctl (int file_des, int cmd, …);
```

Miscellaneous devices

- /dev/adc
  - read() returns numeric text string for channel 0
- /dev/leds
  - ioctl() sets or clears individual LEDs
- /dev/buttons
  - read() returns a 6-character string, '1' or '0' for each button

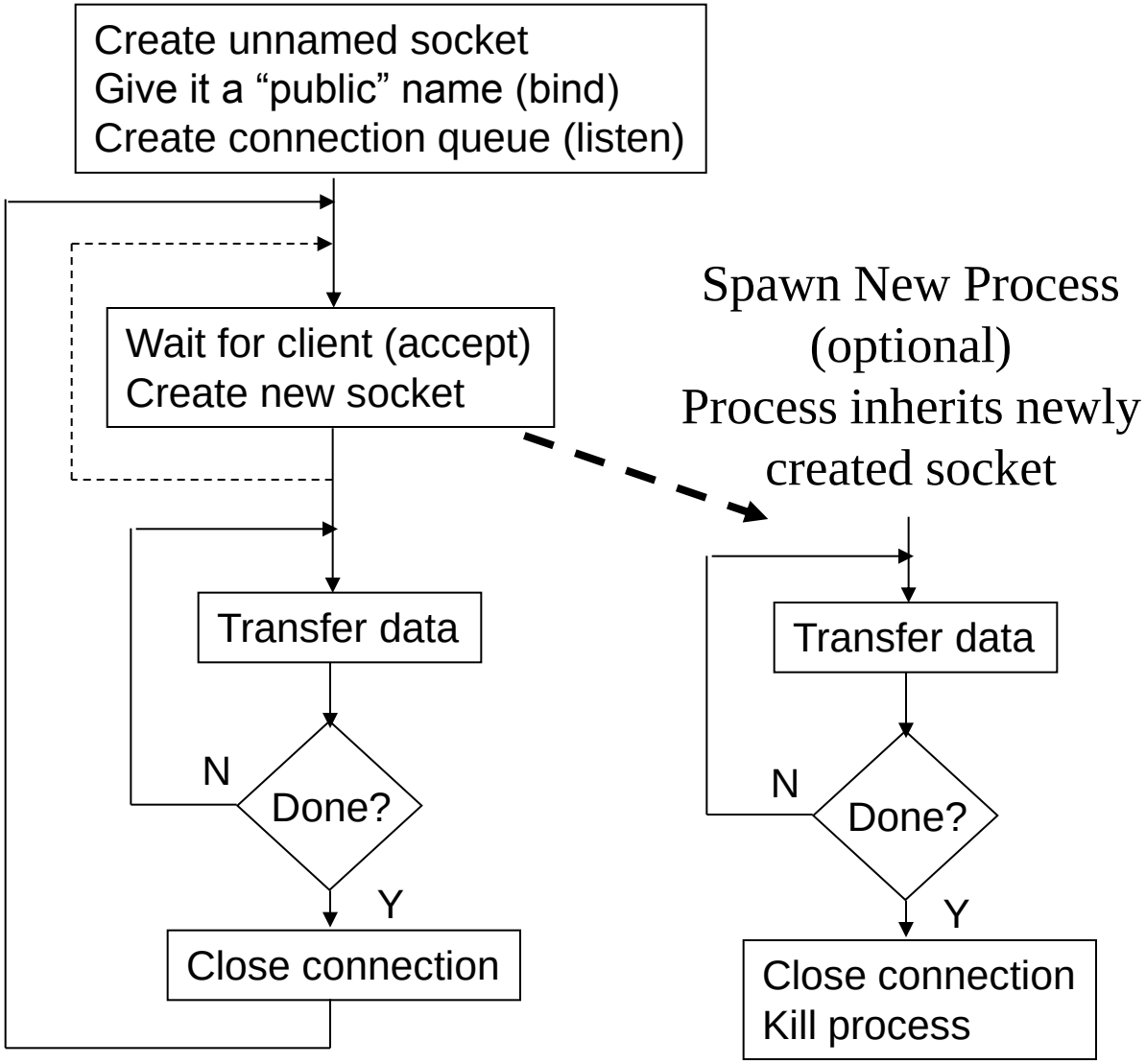
Network Programming: Sockets

- Client/Server paradigm
- Server and client on same machine or across a network

Server Process

- Create unnamed socket
- Give it a public name (bind)
- Create connection queue and wait for clients (listen)
- Accept connection (create new socket)
- Transfer data
- Close connection

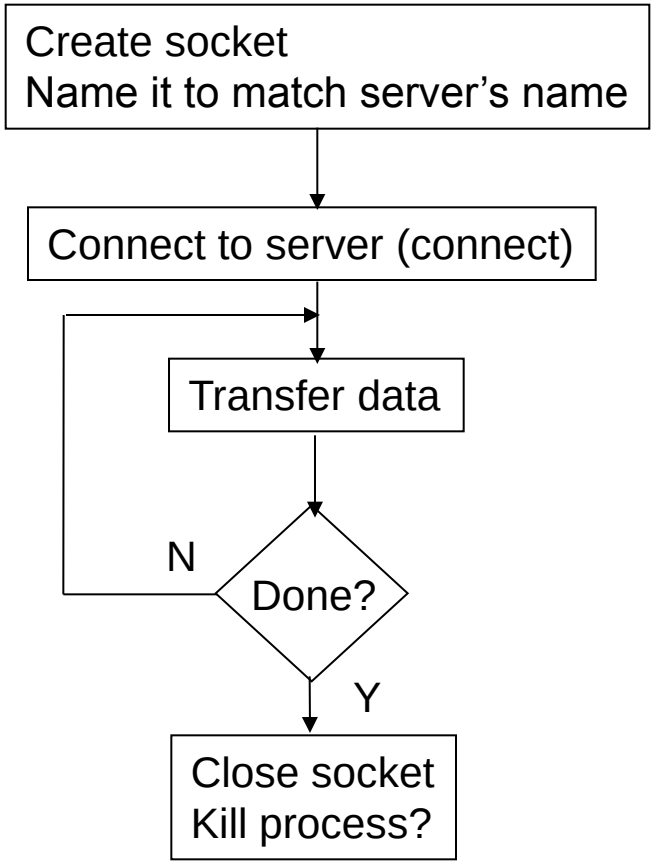
Server Process II



Client Process

- Create socket
- Name it to match server's name
- Connect to server
- Transfer data
- Close connection (close socket)

Client Process II



Socket Attributes

- int socket (int domain, int type, int protocol)
  - Domain -- specifies communication medium
    - AF_INET - Internet protocols
    - AF_UNIX - Local file system
  - Type -- specifies how to communicate
    - SOCK_STREAM - Reliable streams
    - SOCK_DGRAM - Connectionless datagrams
  - Protocol -- often defaults depending on domain and type

Simple Client and Server

- cd ../network
- Create new Eclipse project, network
- Create server and client build targets
- Examine netserve.c and netclient.c.
  - Look at calls to socket(), bind(), listen() and accept().
- ./netserve (in one terminal window)
- ./netclient (in another window)

Export include paths

- Select measure project in Project Explorer
  - Right click, select Export…
- Expand C/C++
  - Select C/C++ Project Settings, click Next
- Uncheck # Symbols
- Browse to home/src, enter file name
  - Data saved as XML
- Finish

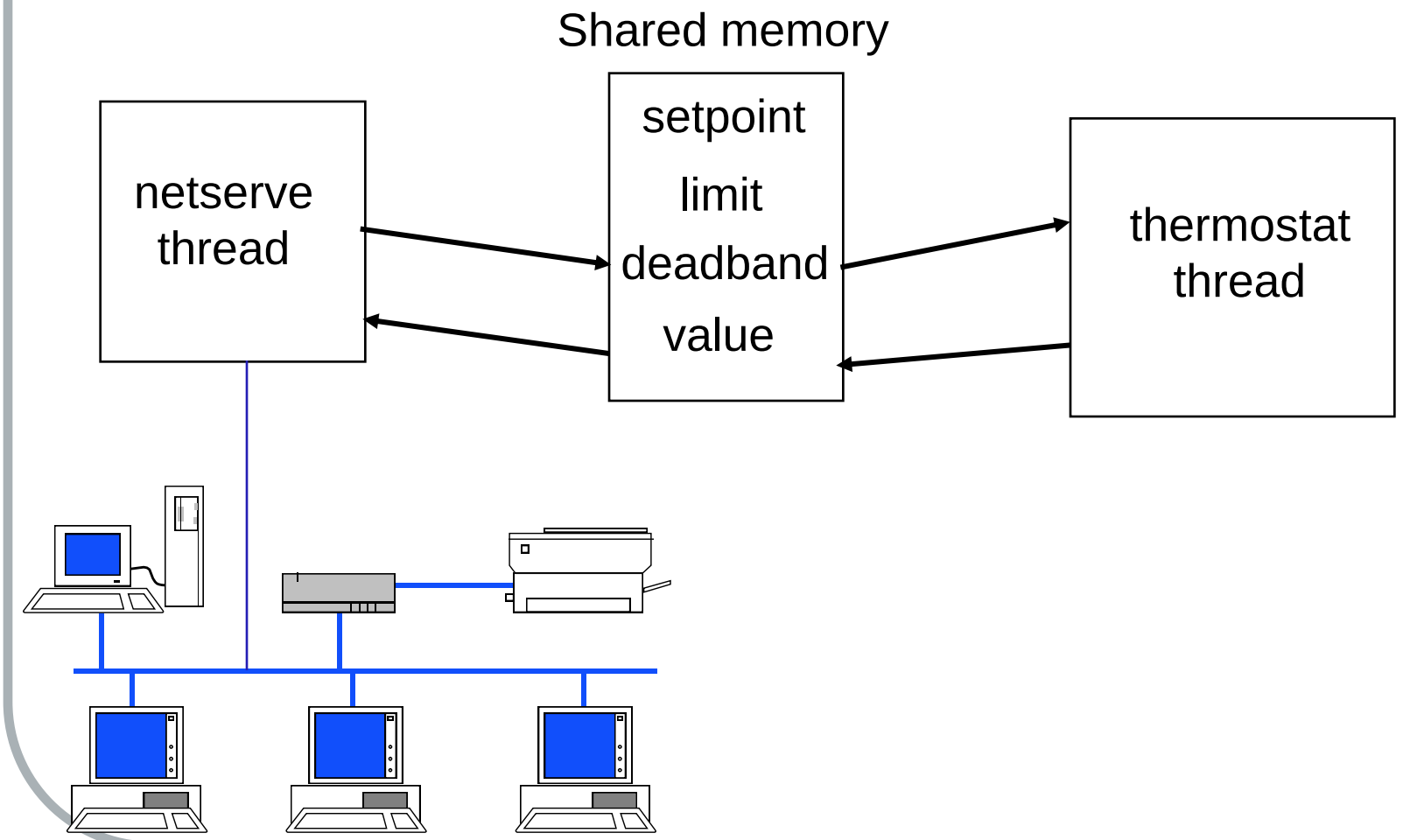
Import include paths

- Right-click network project in Project Explorer
  - Select Import…
- Expand C/C++
  - Select C/C++ Project Settings, click Next
- Browse to home/src, select file
  - OK
- Finish

Across the network

- Create make target “remote”
  - Target server
  - make SERVER=REMOTE
- From minicom window
  - cd /home/src/network
  - ./netserve
- In another shell window
  - ./netclient remote

Thermostat over the network



© Doug Abbott 2009 to 2013 | UCSD Extension Online Learning Course: Embedded Linux

Changes to monitor.c

- Add int createServer (void) function
  - Use lines 18 to 55 of netserve.c
  - Return value is client socket
- Call createServer() just before while (1) loop
- Replace fgets() with read() on client socket
- Add case '?':
- Reply OK to parameter changes

Net Thermostat

- Simulation
  - Create make target “netthermo”
    - Target netthermo
  - Need three terminal windows
- Target
  - Create make target “remote_thermo”
    - Target netthermo
    - make SERVER=REMOTE
  - [doug@localhost measure]$ netclient remote

Multiple monitor threads

- Recast createServer() as a thread
- In createThread(), create the server thread
- Make an infinite loop in createServer()
- Replace return client_socket; with pthread_create() to create monitor thread
- Add case 'q' to monitor thread—client is terminating

Problems with multiple monitors

- How to allocate pthread_t objects?
  - Array of pthread_ts
  - malloc()
- How to pass socket to thread?
  - Thread takes a parameter
- How to recover pthread_t object when thread terminates?

Possible Solution

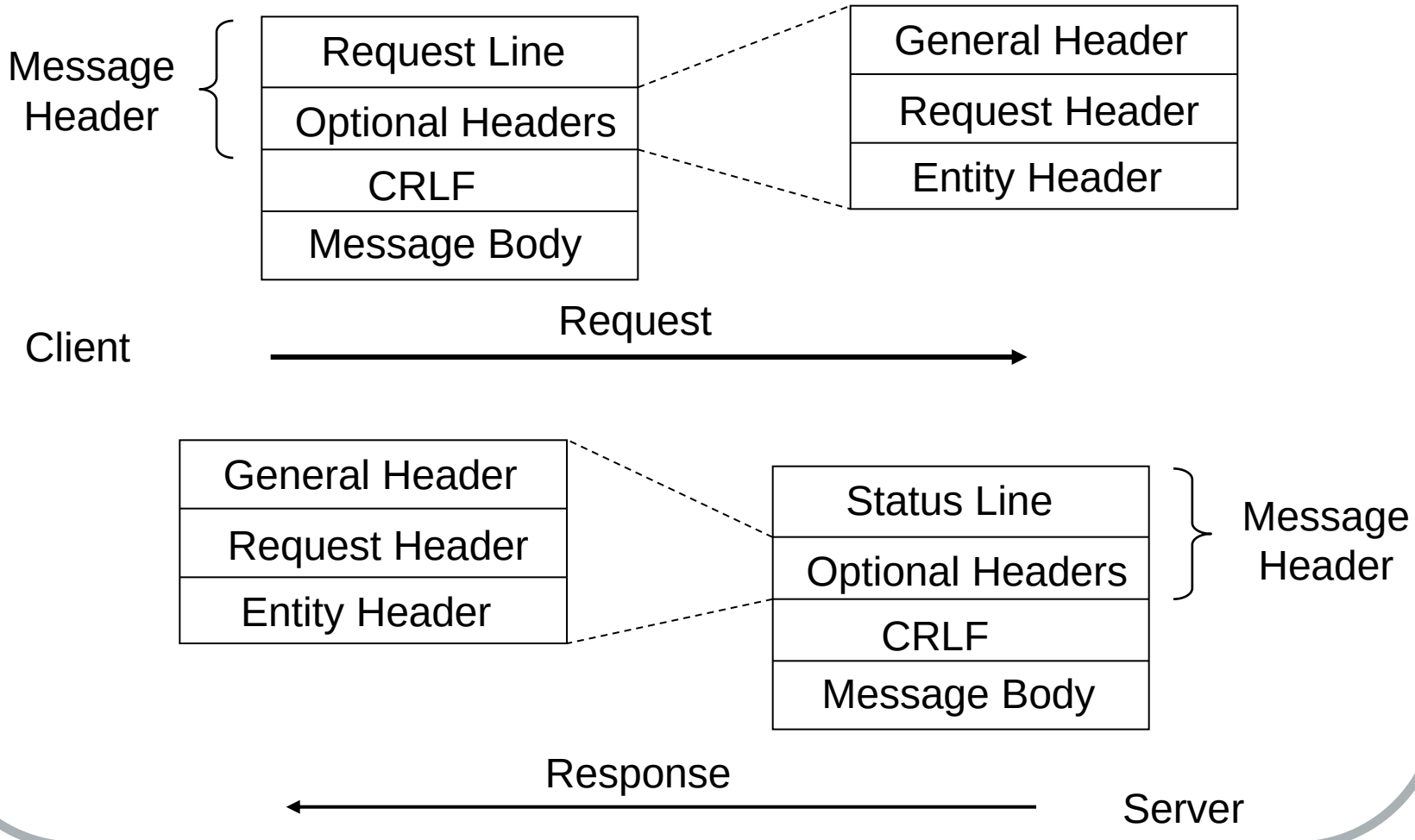
```
typedef enum {FREE, IN_USE, PENDING} used;
typedef struct {
    used flag;
    int socket;
    pthread_t thread;
} meta_pthread_t
```

- Allocate array of meta_pthread_ts
- meta_pthread_t is parameter to monitor thread
- Create thread that looks for terminating monitors
  - Frees meta_pthread_t
- int pthread_join (pthread_t *thread*, void ***ret_value*)

Additional mods to multimon.c

- Add the resource thread
- createThread()
  - Create resource thread
- terminateThread()
  - Cancel and join any currently running monitor threads
  - Cancel and join resource thread
- New make targets
  - make -f Makefile.multi (SERVER=REMOTE)

Embedding the Web



GET / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; U; Linux i586; en-US; rv:1.2.1) Gecko/20030225
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
<blank line>

© Doug Abbott 2009 to 2013 | UCSD Extension Online Learning Course: Embedded Linux

Embedded Web Servers

- Apache is overkill in an embedded app
- Comparison of light weight web servers
  - http://en.wikipedia.org/wiki/Comparison_of_lightweight_web_servers
- boa – happens to be in our file system
  - www.boa.org
- In a web browser, go to http://192.168.1.50
- Look at /etc/boa/boa.conf

Simple Web Server

- GET Send file to client
- Dynamic content
  - Invent <DATA data_function> tag[1]
  - Inserts data into HTML file
- POST receive data from client
  - Forms

1. Jones, M. Tim, *TCP/IP Application Layer Protocols for Embedded Systems*

More Eclipse make targets

- Simulation
  - Target: web
- Target board
  - Target: web
  - Build command: make SERVER=REMOTE
- Must be root user to run webthermo_s
- In a web browser: http://127.0.0.1

Running web server on target

- Build target version
- On the target:
  - ps | grep boa
  - kill the process ID returned by ps
- Start webthermo_t
- In a web browser: http://192.168.1.50

Review

- Device drivers
  - “Miscellaneous” devices
- Exporting/importing project properties
- Sockets
  - Client/server paradigm
- Multiple clients
- Web server