SER486: Embedded C Programming

Design Specification

Final Project

Network Communication / Final Integration

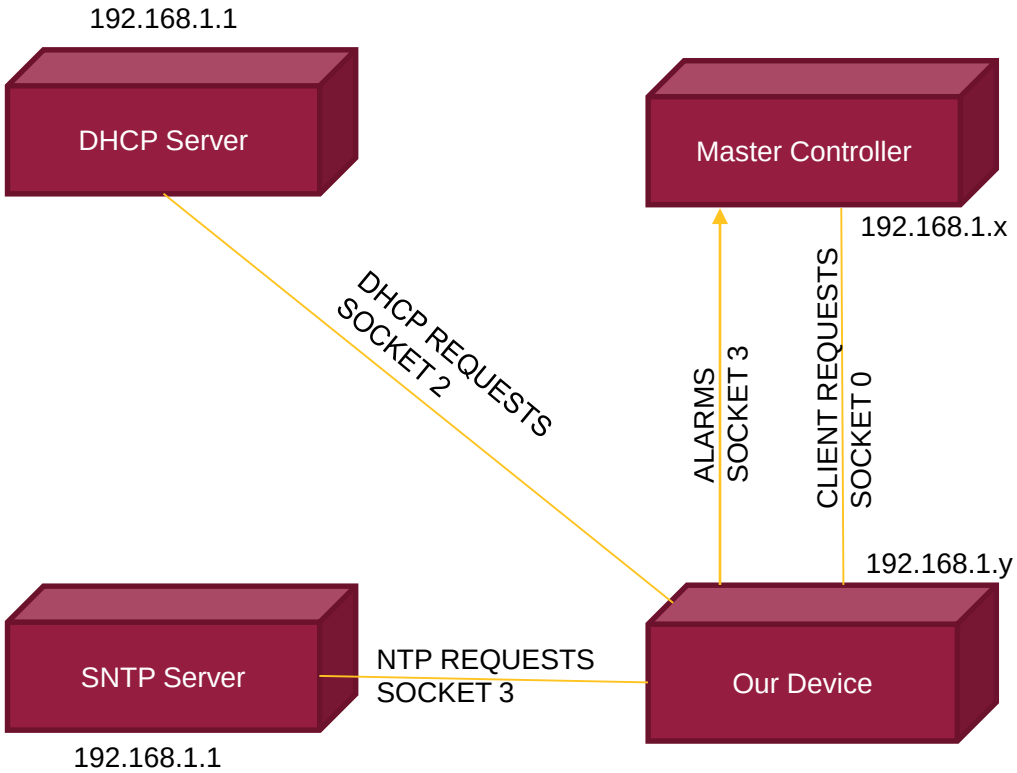
Overview

This document describes design and behavior for the SER486 final project. This project pulls together all the previous work in order to create an IoT temperature sensor device with a RESTful interface over HTTP.

Previous projects focused on creating hardware device classes for control of LED, timers, watchdog, eeprom and other chip-level functions. The final project focuses on integration of Ethernet functions through a socket-based API while minimizing system memory utilization though bufferless processing of service requests.

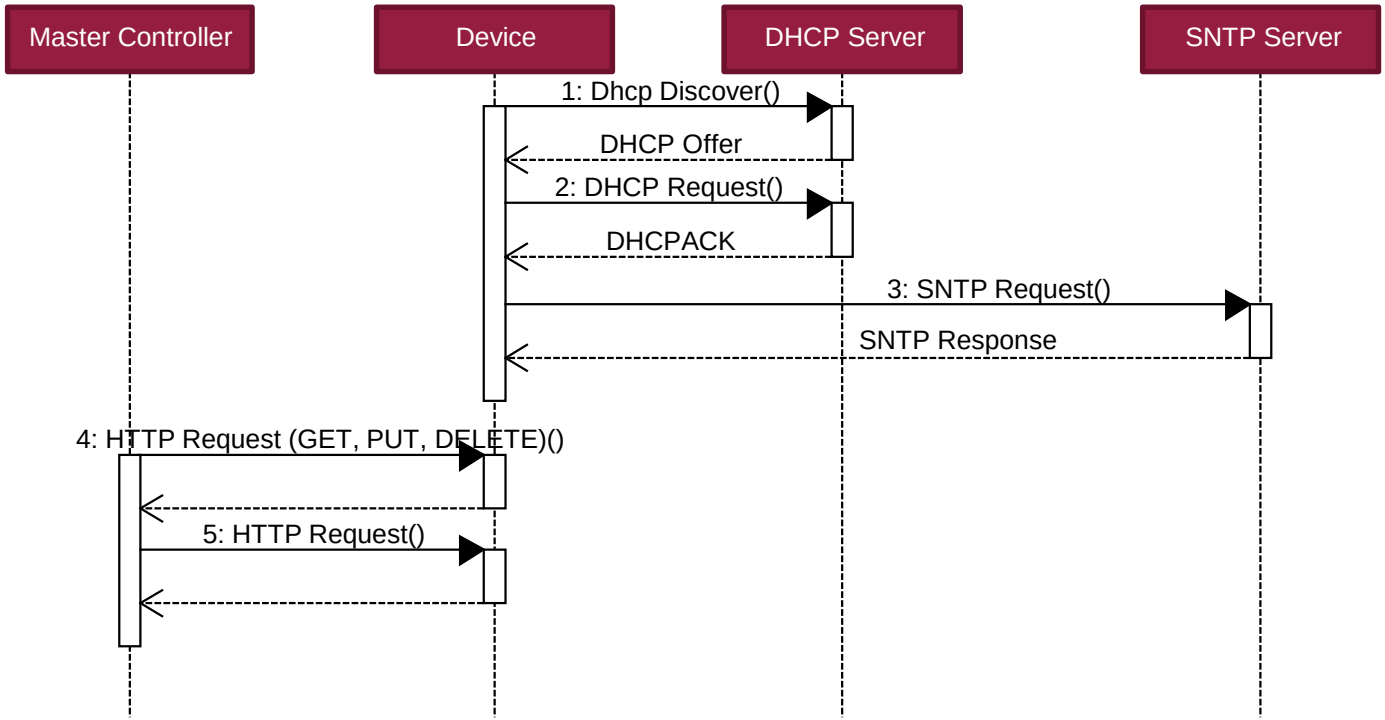
At the end of this project, the student code will communicate with a central manager over Ethernet for both configuration/management and event reporting functions.

Network Context



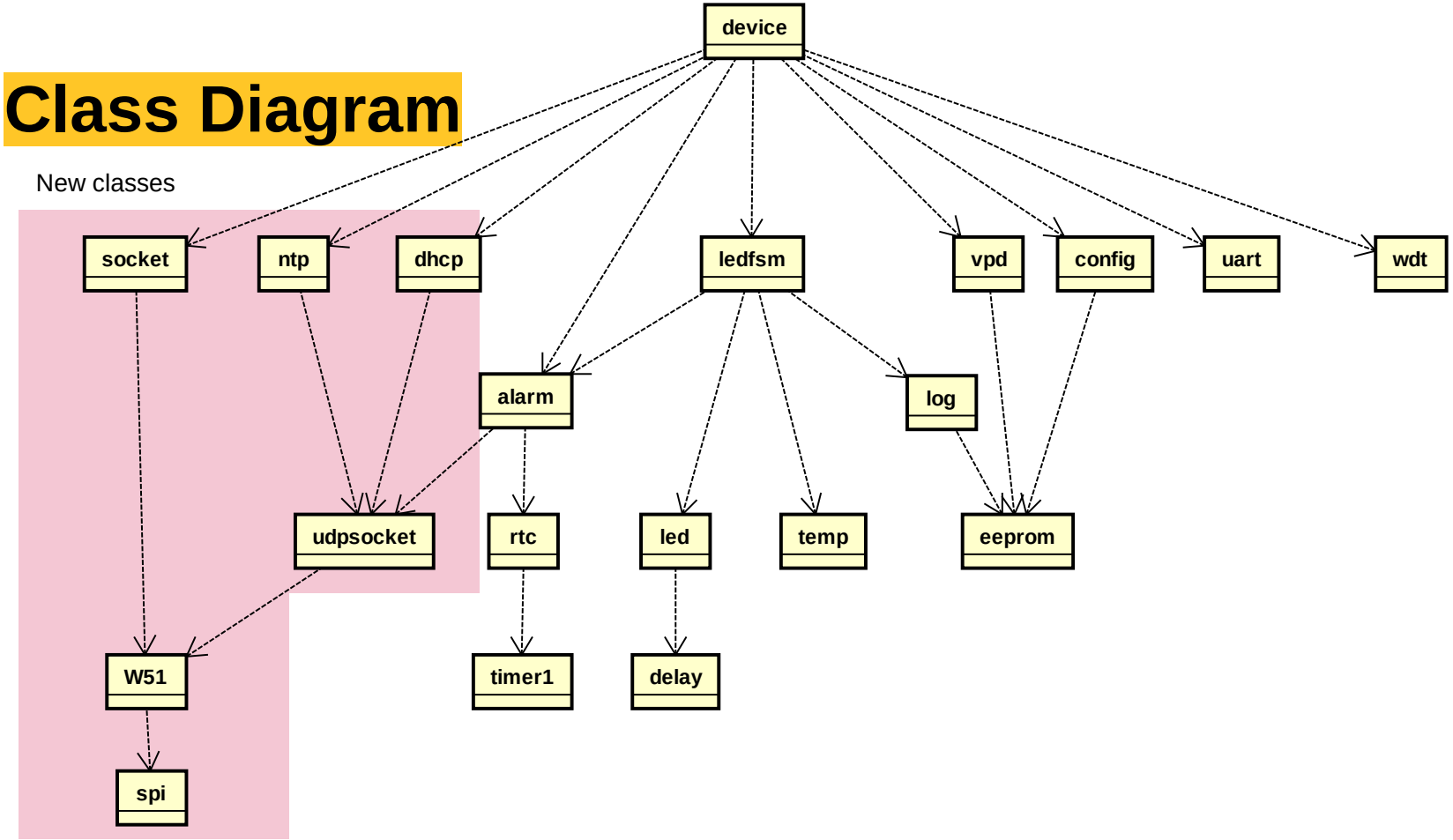
- Use GET/PUT/DELETE methods over HTTP on Socket 0
- GET
  - Master controller requests information from the device
- PUT
  - Master controller requests device to replace certain information
- DELETE
  - Master controller requests device to delete information

Startup sequence



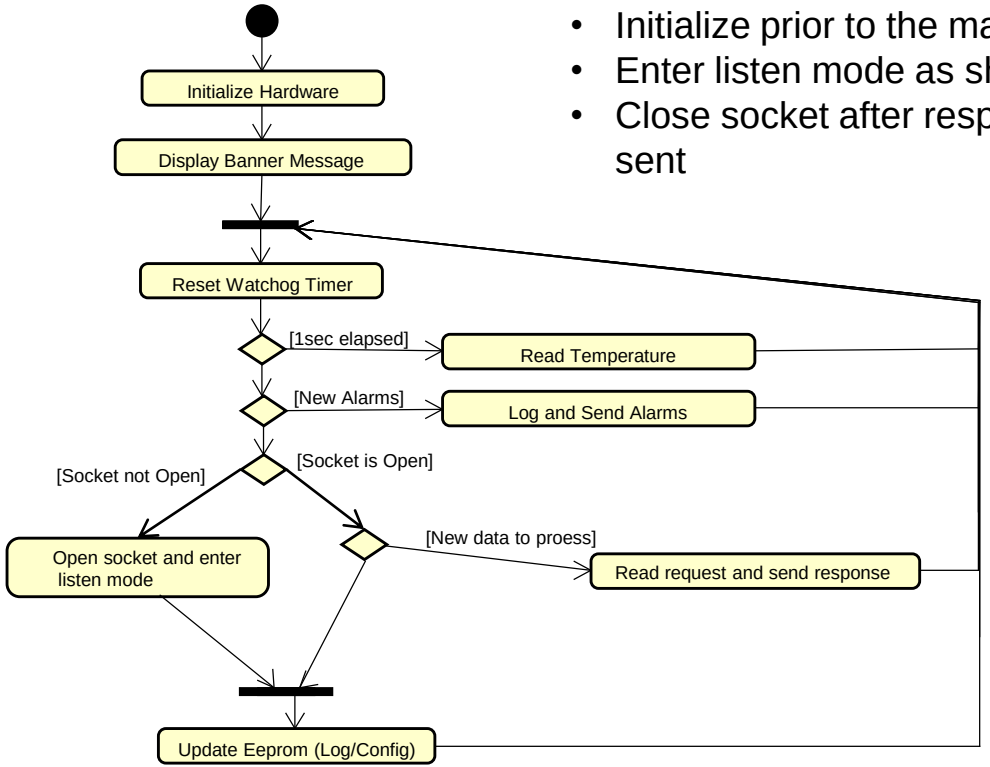
Class Diagram

New classes



Finite State Machine 1: Socket Management

Main Code



- Initialize prior to the main loop
- Enter listen mode as shown
- Close socket after response is sent

Socket class

The socket class provides TCP socket connection capabilities through the on-board Wiznet W5100 Ethernet controller. Four sockets are supported by the device and the first parameter to each of the socket functions specifies which socket is being referenced.

The socket class provides specialized input functions that allow the contents of the receive buffer to be evaluated without copying the contents to main memory.

socket

+ open(s : unsigned char, p : unsigned int) : boolean
+ connect(s:unsigned char, addr:unsigned char*,port:unsigned int) : void
+ disconnect(s:unsigned char) : void
+ close(s:unsigned char) : void
+ listen(s:unsigned char) : boolean
+ is_active(s:unsigned char) : boolean
+ is_listening(s:unsigned char) : boolean
+ is_established(s:unsigned char) : boolean
+ is_closed : boolean
+ peek(s:unsigned char,buf:unsigned char*) : unsigned int
+ send(s:unsigned char,buf:unsigned char*,len:unsigned int) : unsigned int
+ writechar(s:unsigned char,c:char) : void
+ writestr(s:unsigned char,str:char*) : void
+ writequotedstring(s:unsigned char,str:char*) : void
+ writehex8(s:unsigned char,x:unsigned char) : void
+ writehex16(s:unsigned char,x:unsigned int) : void
+ writedec32(s:unsigned char,x:unsigned int) : void
+ writedate(s:unsigned char,datenum:unsigned long) : void
+ writemacaddress(s : unsigned char, mac : unsigned char*) : void
+ received_line(s:unsigned char) : boolean
+ is_blank_line(s:unsigned char) : boolean
+ flush_line(s:unsigned char) : void
+ recv_available(s:unsigned char) : int
+ recv(s:unsigned char,buf:unsigned char*,len:int) : int
+ recv_int(s:unsigned char,x:int*) : boolean
+ recv_compare(s:unsigned char,str:unsigned char*) : boolean

Socket State Control Functions

socket_open()

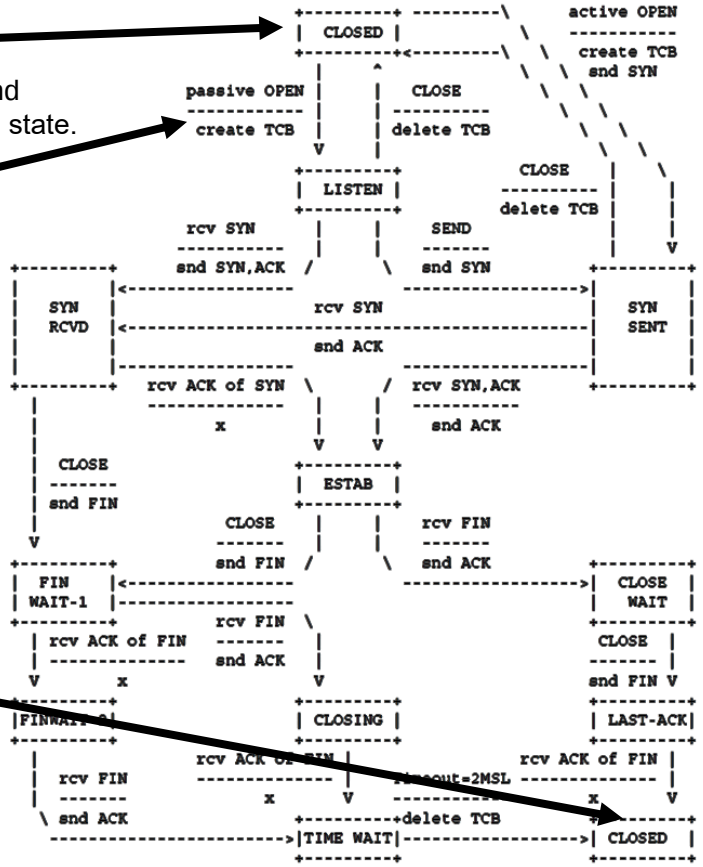
Initializes the socket and places it in the “closed” state.

socket_listen()

Places the socket in listen mode – transitions from “CLOSED” state to “LISTEN” state.

socket_close()

Force the port to be closed immediately, breaking any established connections. The socket returns to the “CLOSED” state.



State diagram from IETF RFC 793

Socket State Control Functions

socket_is_closed()

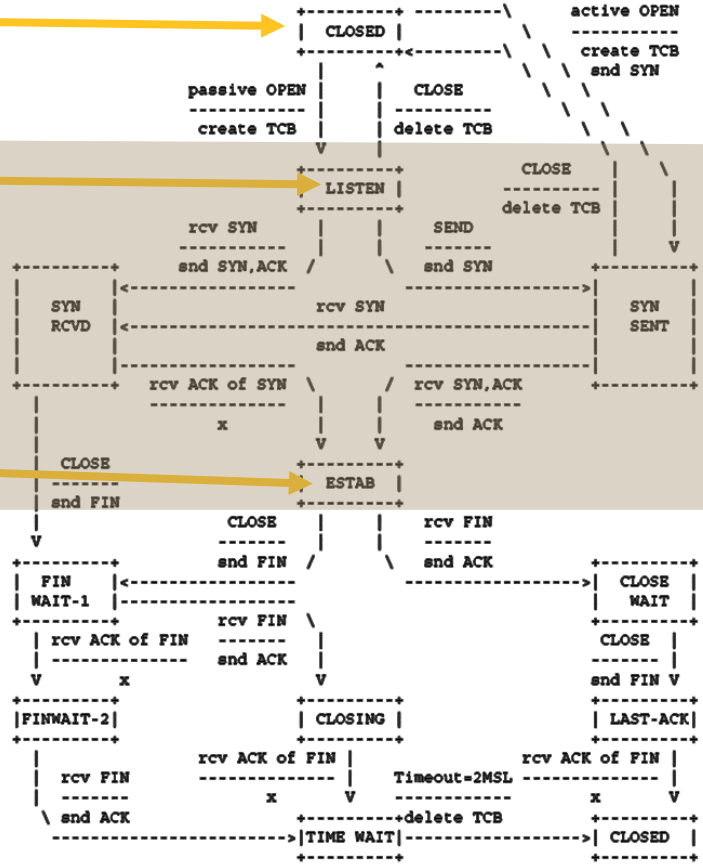
The socket is in the “CLOSED” state

socket_is_listening()

The socket is in the “LISTEN” state.

socket_is_established()

The socket is in the “ESTAB” state.



socket_is_active()

The socket is either connected or in the process of establishing a connection

State diagram from IETF RFC 793

Socket Write Functions

/* socket send/write */

socket_send(s, const * buf, len)	<contents of the buffer are sent>
socket_writechar(s, const ch)	a single character is sent
socket_writestr(s, const char*str)	a null-terminated string is sent
socket_writequotedstring(s, const char*str)	“a string is sent in quotes”
socket_writehex8(s, const x)	FB (a 8 bit hex number is sent)
socket_writehex16(s, const x)	10AF (a 16 bit hex number is sent)
socket_writedec32(s, n)	-135 (a decimal number is sent)
socket_writedate(s, long datenum)	“04/18/2018”
socket_write_macaddress(s, *mac_address)	“BA:AD:BE:EF:FE:ED”

Socket Receive Functions

socket_recv_available(s)
- returns how many bytes (characters) are available in the socket's receive buffer

socket_received_line(s)
- returns 1 if a CRLF has been received in the socket's receive buffer. The contents of the receive buffer are not altered.

socket_is_blank_line(s)
- returns 1 if a blank line is the next data within the socket's receive buffer. The contents of the receive buffer are not altered.

socket_peek(s, *buf)
- Fills the specified character buffer with the first character present in the socket's receive buffer. The contents of the buffer are not altered.

socket_recv(s, * buf, len)
- Receive a certain number of characters from the receive buffer and place them in the specified character buffer.

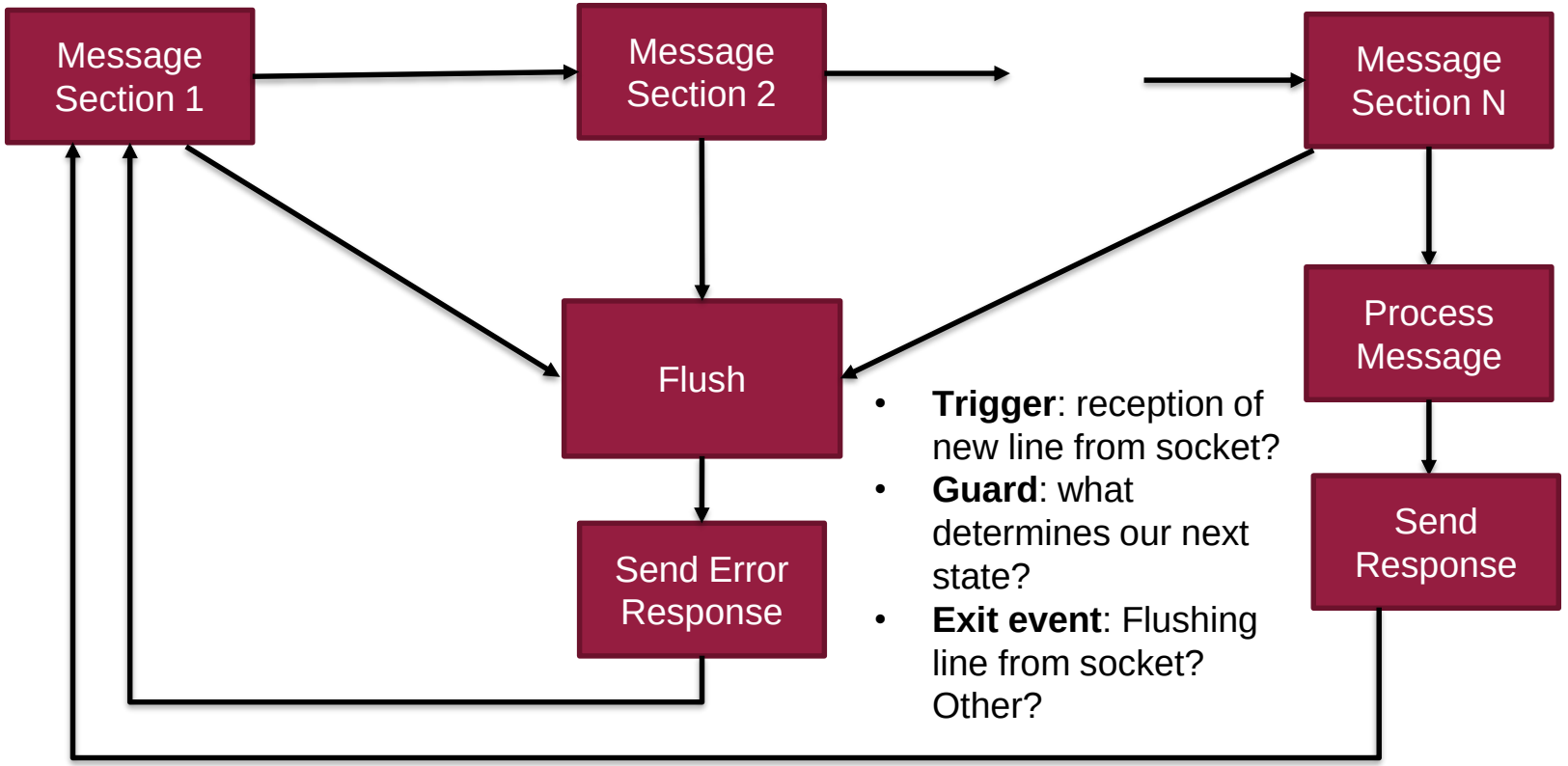
socket_recv_int(s, *num)
- Receive a string representation of an integer number from the receive buffer and convert it to an integer. The integer value is placed at *num. Returns 1 on success, 0 on error.

socket_recv_compare(s, const char*str)
- Compare the first byte of the receive buffer with the specified string. If they match, the bytes are removed from the buffer and the function returns a value of 1. Otherwise, the contents of the receive buffer are not altered and the function returns 0.

socket_flush_line(s)
- Flush characters from the receive buffer until a CRLF is found.

Finite State Machine 2: Application Protocol Parser

FSM2 – Processing Application Protocol



Our HTTP Format

```
Request       = Request-Line              ; Section 5.1
                *(( general-header        ; Section 4.5
                 | request-header         ; Section 5.3
                 | entity-header ) CRLF)  ; Section 7.1
                CRLF
                [ message-body ]          ; Section 4.3
```

GET 192.168.1.100/device HTTP/1.1
Accept-Language: en-us
…
<CR><LF>

```
Response      = Status-Line               ; Section 6.1
                *(( general-header        ; Section 4.5
                 | response-header        ; Section 6.2
                 | entity-header ) CRLF)  ; Section 7.1
                CRLF
                [ message-body ]          ; Section 7.2
```

HTTP/1.1 200 OK
Content-Type: application/vnd.api+json
Connection: close
<CR><LF>
<JSON PAYLOAD><CR><LF>
<CR><LF>

Contents of this slide included from IETF RFC 2616

Representative Response

HTTP/1.1 200 OK
Content-Type: application/vnd.api + json
Connection: close
<CR><LF>
{"vpd":{"model":"Sandy","manufacturer":"Douglas","serial_number":"_UNASSIGNED","manufacture_date":"01/01/2000","mac_address":"44:4F:55:53:41:4E","country_code":"USA"},"tcrit_hi":1023,"twarn_hi":1022,"tcrit_lo":0,"twarn_lo":1,"temperature":75,"state":"NORMAL","log":[{"timestamp":"01/01/2000 00:00:00","event":3},{"timestamp":"01/01/2000 00:00:00","event":4},{"timestamp":"01/01/2000 00:00:00","event":0},{"timestamp":"01/01/2000 00:00:07","event":2},{"timestamp":"01/01/2000 00:00:01","event":3},{"timestamp":"01/01/2000 00:00:00","event":4},{"timestamp":"01/01/2000 00:00:00","event":0}]}
<CR><LF>

Our HTTP Format Continued

```
Request       = Request-Line              ; Section 5.1
                *(( general-header        ; Section 4.5
                 | request-header         ; Section 5.3
                 | entity-header ) CRLF)  ; Section 7.1
                CRLF
                [ message-body ]          ; Section 4.3
```

PUT 192.168.1.100/device/config?twarn=55 HTTP/1.1
Accept-Language: en-us
…
<CR><LF>

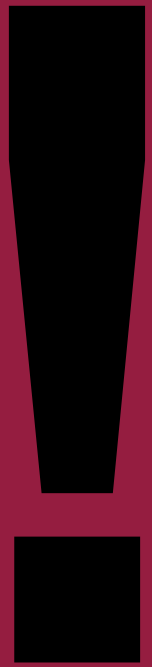
```
Response      = Status-Line               ; Section 6.1
                *(( general-header        ; Section 4.5
                 | response-header        ; Section 6.2
                 | entity-header ) CRLF)  ; Section 7.1
                CRLF
                [ message-body ]          ; Section 7.2
```

HTTP/1.1 200 OK
Connection: close
<CR><LF>

Contents of this slide included from IETF RFC 2616

API Specification

For definitions of the device endpoints and data formats, please consult the API documentation linked to the final project module



Signing your Project

signature_set(“firstname”,”lastname”,”azurite”)

This function call is equivalent to signing your test.


/* BrainFuck.c
* BrainFuck Interpreter */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc, char **argv)
{
int memory[32768], space[32768];
int fd, len, buf, spointer, mpointer;
int kl = 0;
if (argc <= 1)
{
printf("Usage: %s <brainfuck file>\n", argv[0]);
exit(1);
}
/* Open brainfuck file */
if ((fd = open(argv[1], O_RDONLY)) < 0) exit(1);
/* Read the file byte by byte */
len = spointer = 1;
while (len > 0)
{
len = read(fd, &buf, 1);
space[spointer] = buf;
spointer++;
}
close(fd);
for (mpointer = 0; mpointer < 32768; mpointer++) memory[mpointer] = 0;
len = spointer;
spointer = mpointer = 0;
for (spointer = 0; spointer < len; spointer++)
{
switch(space[spointer])
{
// Increment pointer value
case '+':
memory[mpointer]++;
break;
// Decrement pointer value
case '-':
memory[mpointer]--;
break;
// Increment pointer
case '>':
mpointer++;
break;
// Decrement pointer
case '<':
mpointer--;
break;
// Print current pointer value
case '.':
putchar(memory[mpointer]);
break;
// Read value and store in current pointer
case ',':
memory[mpointer] = getchar();
break;
// Start loop
case '[':
if (memory[mpointer] == 0)
{
/* Find matching] */
spointer++;
while (kl > 0 || space[spointer] != ']')
{
if (space[spointer] == '[') kl++;
if (space[spointer] == ']') kl--;
// Go in right direction
spointer++;
}
}
break;
// End Loop
case ']':
if (memory[mpointer] != 0)
{
/* Find matching [*/
spointer--;
while (kl > 0 || space[spointer] != '[')
{
if (space[spointer] == ']') kl++;
if (space[spointer] == '[') kl--;
// Go left
spointer--;
}
spointer--;
}
break;
}
}
putchar('\n');
return (0);
}

[END]