ESP8266 Wifi Module Interfacing with Raspberry Pi

Submitted for completion of the course of

Device Drivers (EEE G547)

Submitted to,

Mr. Devesh Samaiya

Submitted by,

Arpita Gupta (2016H140102P)

Shivi Mishra(2016H140100P)

I. Purpose of the code

The goal of our project was to port the existing esp8266 wifi driver on Raspberry Pi and be able to access internet using it as one would be able to if the wifi module was already present on the board.

Since the driver written for ESP8266 is incorrect online which we came to realize later after trying hard to debug and change hardware modules, we were unable to make it run though we tried debugging the code.

Now our code will demonstrate IoT application using ESP8266 wifi module on Raspberry Pi where we send data to an online IoT logger and analytics website (www.thingspeak.com) and retrieve data from the same website.

II. Usage of the code

Two lua scripts have been written and saved on esp8266 for execution. One script (thingspeaksend.lua) sends data online to our thingspeak channel and another script (script1.lua) retrieves the same data from the channel.

A C code has been written which reads and writes into the serial port file (/dev/ttyAMA0) thus sending the lua commands to ESP8266 via UART.

Lua script-1 : thingspeaksend.lua

```lua
--Connect to wifi where ssid and password are hardcoded
wifi.sta.config("shivi" ,"shivi1893")
wifi.sta.connect()

--Create a function sendThingSpeak which establishes connection and sends data online
function sendThingSpeak(level)
    local connout = nil

    --Create TCP connection
    connout = net.createConnection(net.TCP, 0)

    --Receive details from online server/website about status etc
    connout:on("receive", function(connout, payloadout)
        if (string.find(payloadout, "Status: 200 OK") ~= nil) then
            print("Posted OK");
        end
    end)

    --Connection is On
    connout:on("connection", function(connout, payloadout)

        print ("Posting...")

        --Data is being posted on thingspeak channel
        connout:send("GET http://52.7.7.190/update?api_key=Q00XI7ZOGTMJBN55&field1=6\r\n\r\n")
        print("Data Sent")
    end)

    connout:on("disconnection", function(connout, payloadout)
        connout:close()
        collectgarbage()
    end)

    --A socket is created which binds port number to IP address of the website. This creates a medium/channel for transfer of data
    connout:connect(80,'52.7.7.190')
end

--A timer which sends data at a frequency of 1 minute
```

Lua script 2 : script1.lua

```
wifi.sta.config("shivi" ,"shivi1893")
wifi.sta.connect()

function GetDataFromThingspeak(_server, _port, _channel, _field)
--print("Querying data from thingspeak")
conn=net.createConnection(net.TCP, 0)
conn:on("receive", function(conn, payload)
   tmr.wdclr()
   --print(payload)
   tmr.wdclr()
   conn:close()
   ThingspeakReply = payload
   print("Query "..server..":".._port.." channel".._channel.." field".._field.." REPLY: "..ThingspeakReply.."\n")

end)
conn:connect(_port,_server)

--Send request for receiving data in payload
conn:send("GET /channels/263048/fields/1/last?key=ND74EYBDOR6TQBTD HTTP/1.1\r\n")
conn:send('Host: 52.7.7.190\r\nAccept: */*\r\nUser-Agent: Mozilla/4.0 (compatible; esp8266 Lua; Windows NT 5.1)\r\n\r\n')

conn:on("disconnection", function(conn)
          --print("Got disconnection...")
         conn=nil
  end)
end
-- END GET DATA FROM THINGSPEAK



function data_init()
   if ThingspeakReply == nil then
      GetDataFromThingspeak("52.7.7.190", 80, 263048, 1)
   else
   print("Last entry in Thingspeak database: "..ThingspeakReply.."\n")
   -- now a valid reply is available, do whatever you want with the read value
   end


end

tmr.alarm(0,5000, 1, function() data_init() end )
```

C code for calling these scripts on ESP12

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <time.h>
#include <string.h>
#define blocksize 255
#define DEBUG 1

int sfd;                                          //File descriptor
char buf[blocksize];

void delay(void)
{
 int c,d;
 for(c=1;c<=32767;c++)
 for(d=1;d<=32767;d++)
 {}
}
void delay_less(void)
{

   int c,d;
   for(c=1;c<=20000;c++)
   for(d=1;d<=20000;d++)
   {}
}

int initWiFi()
{
 //Relinquish access of serial port on Raspberry Pi for our use
 system("sudo systemctl stop serial-getty@ttyAMA0.service");

 //Open the file /dev/ttyAMA0 in read/write mode
 sfd = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY);

 //Check whether the file has opened or not
 if (sfd == -1)
 {
         printf("Error no is : %d\n", errno);
         printf("Error description is : %s\n", strerror(errno));
         return -1;
     }

 //Setting the UART baud rate,stop bits, flags etc
     struct termios options;
 //Fills the struct with current values specified by the file descriptor
     tcgetattr(sfd, &options);
 cfsetspeed(&options, B9600);
     cfmakeraw(&options);          //Enable raw mode on input/output
     options.c_cflag &= ~CSTOPB;   //cflag = control options
     options.c_cflag |= CLOCAL     //Ignore modem control lines
     options.c_cflag |= CREAD;     //Enables read operation
     options.c_cc[VTIME] = 1;      //timeout value
     options.c_cc[VMIN] = blocksize;  //Min no. of characters before the
call is considered satisfied
 //Stores values back in the device, TCSANOW-Immediate change
     tcsetattr(sfd, TCSANOW, &options);
};

//Receive data from the opened file
int getBlock()
{
 int bytes;
 struct timespec pause;
 pause.tv_sec = 0;
 pause.tv_nsec = 100000000;
 nanosleep(&pause, NULL);
 memset(buf, '\0', sizeof (buf));
 //Return the number of characters waiting
 ioctl(sfd, FIONREAD, &bytes);
 if (bytes == 0)return 0;
 int count = read(sfd, buf, blocksize - 1);
 buf[count] = 0;
 if (DEBUG) {
  printf("%s", buf);
  fflush(stdout);
 }
 return count;
}

int main(int argc, char** argv)
{

 initWiFi();
 delay_less();

 //write lua command to file which in turn sends to esp via UART for
execution
 //Sending data to Thingspeak channel
 char file_name1[]= "thingspeaksend.lua";
 dprintf(sfd,"dofile(\"%s\")\r\n",file_name1);
 delay();
 delay();
 delay();
```

```c
delay();
delay();
delay();
delay();
delay();
delay();
delay();
delay();
delay();
delay();
delay();
delay();

//Receiving last send data from Thingspeak Channel
char file_name[]= "script1.lua";
dprintf(sfd,"dofile(\"%s\")\r\n",file_name);
return (EXIT_SUCCESS);
}
```

III. Issues faced and solutions during the entire process before and after switching

- Building Kernel for Raspberry Pi
    - Raspberry Pi Jessie Lite doesn't work for complete header installation. Hence complete Jessie OS had to be installed.
    - Original kernel headers were different from upgraded headers. Both of them were incomplete.
    - Various hacks were tried to install headers or link them but didn't work. Major methods are described in the latter points.
    - It was tried to be linked (using symlink)but that did not work and no build directory could be linked to working kernel
    - At one point of time, linking worked but module symbols were not being imported for the new kernel source. New Module.Symvers was tried but it did not work
    - Finally after trying this we had to recompile the entire kernel several times.
    - What finally worked was-
        - Installing latest headers from RPi-Source which installs the kernel headers in /root/linux

          $sudo wget https://raw.githubusercontent.com/notro/rpi-source/master/rpi-source -O /usr/bin/rpi-source

          $sudo chmod +x /usr/bin/rpi-source

          $/usr/bin/rpi-source -q --tag-update

          $rpi-source --skip-gcc

        - After installation of kernel headers, the entire kernel was rebuilt in the root directory
        - SDIO polling had to be enabled hence dtoverlay parameter in /boot/config/ had to be changed according to our use.

          ```
          $sudo sed -i -e "/^dtoverlay.*sdio/d" /boot/config.txt
          ```

    - Esp8089 wifi kernel library was cloned from github and compiled to generate .ko file
    - Since insmod was not working for our recompiled kernel, we had to first depmod it and then modprobe which inserted the driver in the kernel which configured it to restart at every soft reset or rmmod
- Making the driver work-
    - We had established the connections as mentioned using resistors for SDIO connections, but the module did not boot up at all. It kept throwing up the following errors-
        - esp_init_retry __no power up__
        - sip_rx first read error -84 12
        - Unable to enable sdio function : -62
        - Probe of mmc1 :0001:1 failed with error -62
    - We were using NodeMCU initially but that didn't work. Reason was that NodeMCU internally pulls down GPIO15 to boot in Flash mode.
    - There were various solutions suggested for this issue-
        - Different power supply
        - Pulling up GPIO 15 to boot it in SDIO mode
        - Pulling up GPIO 0 and GPIO2 to boot in SDIO mode
        - Loose connections
    - To tackle these issues- we tried running the driver using ESP12F modules. None of the above solutions worked
    - We then removed the resistors and directly connected SDIO pins to the relevant GPIO of RPi. It got the power and booted up the module which was being shown in kernel logs.
    - Finally the module powers up, mmc card gets flashed and initialized but error thrown is
        - esp_init_all failed : -110
        - first error exit
    - Many people have already faced this issue and one of the suggestions was loose SDIO connections.
    - For this purpose we ordered another Soldered ESP12F smd module with headers. That did not work either.

    - The above issue has been faced by many people and no solution to this problem has been suggested or tested by anybody till date.
    - We have been debugging the driver since then, printing the logs at various places to see where the driver stops working but haven't debugged it till date.

- After switching to IoT application few challenges faced were
    - We first tried establishing communication using AT commands but because of wrong firmware were not able to do so. We had to switch to Lua
    - Install correct esp8266/nodemcu firmware for lua
    - Lua commands worked fine when sent directly from the terminal whereas when the same commands were being sent from C code they were not executing. Solution was to save lua script on ESP and call a single dofile command from C to remove any possible errors which might otherwise arise and be extremely tough to detect.