VOL. 12, NO. 17, SEPTEMBER 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

INTEGRATED INFORMATION AND COMMUNICATION LEARNING MODEL FOR RASPBERRY Pi ENVIRONMENT

Y. J. Lee
Department of Technology Education, Korea National University of Education, South Korea
E-Mail: lyj@knue.ac.kr

ABSTRACT

This paper aims to present the information and communication learning model applicable in teaching students at school site. The proposed model deals with the information creation and control based on the physical computing and information transfer based on TCP/IP socket in one framework. It thus provides students with the total understanding and practice opportunity about information and communication technology simultaneously. Our proposed learning models are classified into the client-server based model and the web based model. The model has been implemented on Raspberry Pi well known as a physical computing education platform. In the real implementation, the information acquisition and control are performed by C and Python, and the information communications are carried by socket interface. Our proposed learning model is also used for students to understand the basic concept of Internet of Things (IoT), which provides us with worldwide control and communication of information.

Keywords: raspberry Pi, physical computing, information, communication learning model, IoT.

1. INTRODUCTION

The important key for setting up a basic learning model to educate ICT (information and communication technology) on the school site is to teach the information acquisition and the communication to students harmoniously.

To realize these learning models, both hardware training and software training are required. However, in traditional ICT education, students have shown a limited understanding of the overall concept of information literacy, with only software training that does not involve hardware education or software education that does not involve hardware.

Physical computing [1] taught in technical education or engineering education in recent days provides the effective interaction between humans and physical devices by simultaneously using hardware and software.

If we utilize the physical computing in the education, we can control electronic parts and sensors through simple software programming and obtain the information. Therefore, we can learn the IoT (Internet of Things) concept through the practice [2, 3].

There are representative platforms for educating physical computing such as Arduino, Galileo, and Raspberry Pi already on sale in the marketplace.

Arduino [4, 5] is a deceptively simple device that does not require interpreter and operating systems. Sketch codes based on C and C ++ are compiled into machine language codes and executed inside the chip. Arduino is a control oriented device because it controls the external device (hardware) directly. It is suitable equipment for obtaining data from the components, such as sensors, and generating information directly from the sensors. However, in terms of communication, Arduino mainly focuses on serial communication. Thus, there is a problem with the cost and the complexity of using the additional shield for Ethernet, Wi-Fi, and Bluetooth when we want to perform the universal communications, such as the IoT.

The Galileo of Intel [6] is a full-scale computing system equipped with Yocto Linux, commonly used in the embedded system. Therefore, users can use computer language such as Python and C. However, it needs additional parts for communication such as Wi-Fi or Bluetooth.

The Raspberry Pi [7, 8] is a processor with the Linux based operating systems (raspbian). Therefore, users can use C language. Recent Raspberry Pi 3 has the embedded Wi-Fi and Bluetooth modules. Specially, the embedded Python is very concise and easy to learn than the C language. Also, since it has a conventional physical input/output (GPIO) to control sensors, physical computing education is easily performed. Additionally, it is possible to use the graphics and games, etc. However, the controller's role in Arduino is inadequate due to the general purpose orientation. Also, it can be a disadvantage for students who are familiar with conventional sketch language must use C or Python on Raspberry Pi.

As noted above, Arduino has the advantage on acquiring information, and Raspberry Pi has the advantage of communicating, it is desirable to use these two devices with the balance. In this study, we propose a learning model in one framework for acquiring information and transferring information on the Raspberry Pi. The proposed model is also available to educate students about the IoT [9, 10].

The rest of the paper is organized as follows. Section 2 presents a learning model that utilizes a client and a server, and section 3 presents a web based learning model, followed by concluding remarks in section 4.

2. CLIENT-SERVER LEARNING MODEL ON RASPBERRY Pi

Wiring Pi is one of the GPIO interface library for the Raspberry Pi to help the physical computing [11]. Now let us consider the LED blink program as a basic example. We connect the number 6 pin of wiring Pi (BCM No. 25)

to the LED via 150 ohm resistor and connect to the LED (-) of the bread board, and then obtain the circuit diagram as like Figure-1.

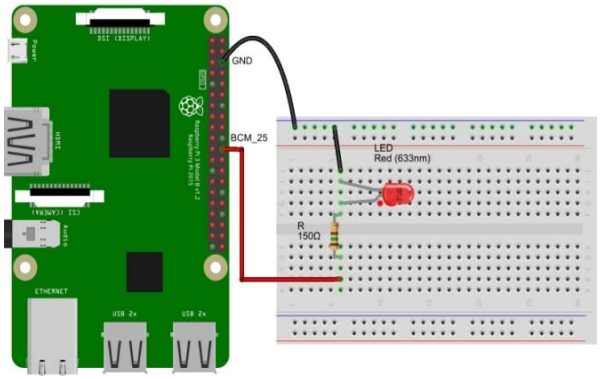

Figure-1. LED blink on Raspberry Pi.

The C language program for LED blink by using wiringPi is presented in Figure-2.

```
// ------------ blink.c --------------------
// $ gcc blink.c -o blink -lwiringPi
// $ sudo ./blink
// -----------------------------------------
#include <stdio.h>
#include <wiringPi.h>
#define LED 6
int main(void)
{
   if (wiringPiSetup() == -1)
   {
      return 1;
   }
   pinMode(LED, OUTPUT);
   digitalWrite(LED, 0);

   for( ; ; )
   {
      digitalWrite(LED, 1);
      delay(1000);
      digitalWrite(LED, 0);  // turn off LED
      delay(1000);
   }
   return 0;
}
```

Figure-2. LED blink program on Raspberry Pi.

We include wiringPi header file supporting the wiringPi library function to perform physical computing. We indicate the pin number -6 as the LED. We initialize HW using wiringPiSetup(). We then set the pin mode as output. After turning off LED pin, we turn on the LED pin. We delay in executing the run for 1000 milliseconds (1 second). Finally, we turn on the LED for 1 second and the blink endlessly repeats the loop.

Since wired and/or wireless LAN card are embedded in the Raspberry Pi, we can transfer the information obtained from the physical computing to other computers through socket. In order to teach information and communications in one learning model, we write network programs to combine sockets and physical computing.

Figure-3 depicts the client-server learning model and AP represents wireless access pointer.

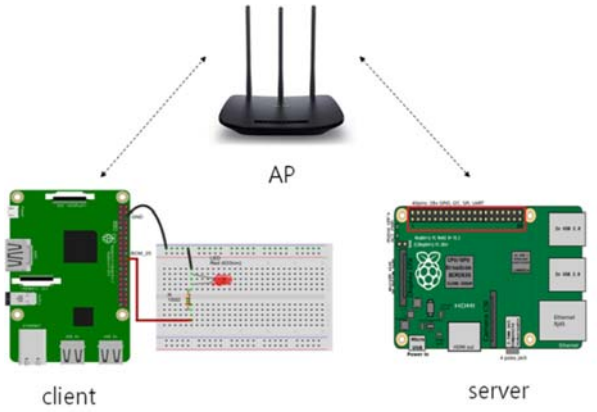

Figure-3. Client-server learning model on Raspberry Pi.

2.1 C Program

The client program for LED blink by using C language is represented in Figure-4. The program handles both information handling and communication through socket simultaneously.

```
// ------------------- blink_client.c ----------------------
// $ gcc blink_client.c -o client -lwiringPi
// $ sudo ./client IP_addr Port
//-------------------------------------------------------------
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <wiringPi.h>
#define LED 6

int main(int argc, char *argv[])
{
   int x;
   struct sockaddr_in p;
   struct hostent *host;
   char message[100];
   char buf[100];
   char buffer[BUFSIZ];

   if ((x = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
      perror("client: socket"); return 1;
   }
   //--- DNS processing ---
```

```
   host = gethostbyname(argv[1]);
   if( host == NULL)
   {
      printf("Host not found !!: %s\n\r", argv[1]);
      return;
   }
   bzero((char *)&p, sizeof(p));
   p.sin_family = AF_INET;
   memcpy((char *)&p.sin_addr, host->h_addr,
               host->h_length);
   p.sin_port = htons(atoi(argv[2]));
   inet_ntop(AF_INET, &p.sin_addr, buf, sizeof(buf));
   printf("Trying %s ...\n", buf);
   if (connect(x, (struct sockaddr *)&p, sizeof(p)) == -1)
   {
      perror("client: connect"); return 1;
   }
   printf("Connected %s...\n", argv[1]);
   //-------- GPIO processing ---------
   if (wiringPiSetup() == -1)
   {
      return 1;
   }
   pinMode(LED, OUTPUT);
   digitalWrite(LED, 0);

   for( ; ; )
   {
      //---- turn on LED ----
      digitalWrite(LED, 1);
      strcpy(message, "On");
      printf("%s\n", message);
      if (write(x, message, strlen(message) + 1) == -1)
      {
         perror("client: write"); return 1;
      }
      delay(3000);
      //---- turn off LED ----
      digitalWrite(LED, 0);
      strcpy(message, "Off");
      printf("%s\n", message);
      if (write(x, message, strlen(message) + 1) == -1)
      {
         perror("client: write"); return 1;
      }
      delay(3000);
   }
   close(x);
   return 0;
}
```

Figure-4. LED blink client program.

We first include wiringPi header file representing the prototype of the wiringPi library function. The pin number used for LED is 6. We use TCP socket. By using wiringPiSetup(), we initialize the H/W. We send a message (On) to the server when the LED is turned on. We send a message (Off) to the server when the LED is turned off. Figure-5 represents a corresponding server program.

```
// -------- blink_server.c ----------------
// $ gcc blink_server.c -o server
// $./server 8900
//-------------------------------------------
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   int x, y;
   int clilen;
   struct sockaddr_in addr, cli_addr;
   char buf[100];
   char buffer[BUFSIZ];
   if ((x = socket (AF_INET, SOCK_STREAM, 0)) == -1)
   {
      perror("server: socket"); return 1;
   }
   addr.sin_family = AF_INET;
   addr.sin_addr.s_addr = INADDR_ANY;
   addr.sin_port = htons(atoi(argv[1]));
   if (bind (x, (struct sockaddr *)&addr, sizeof(addr))
      == -1)
   {
      perror("server: bind"); return 1;
   }
   if (listen (x, 5) == -1)
   {
      perror("server: listen"); return 1;
   }
   printf("server start !!! \n");
   clilen = sizeof(cli_addr);
   if ((y = accept (x, (struct sockaddr *)&cli_addr,
      &clilen)) == -1)
   {
      perror("server: accept"); return 1;
   }
   inet_ntop(AF_INET, &cli_addr.sin_addr, buf,
      sizeof(buf));
   printf("client address => %s\n", buf);
   while(1)
   {
      if (read (y, buffer, BUFSIZ) == -1)
      {
         perror("server: read"); return 1;
      }
      printf("receive message: %s\n", buffer);
   }
   close(x);
   close(y);
   return 0;
}
```

Figure-5. LED blink server program.

We first create the TCP socket for Internet. We then bind the server's IP address and port number. After waiting for a connection request from the client, we accept the connection requests. We finally receive a message from the client.

2.2 Pyton program

Client program and server program written in Python language for LED blink is represented in Figure-6 and Figure-7 respectively. We see that client program handles both information acquisition and communication through socket one framework.

```
#------- blink_client.py -------------
# $ sudo python ./blink_client.py
#---------------------------------------
import RPi.GPIO as GPIO
import socket
import time
HOST = '192.168.0.101'
PORT = 8900

s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.connect((HOST, PORT))

GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
try:
   while True:
      GPIO.output(25, GPIO.HIGH)
      s.sendall("On")
      time.sleep(1)
      GPIO.output(25, GPIO.LOW)
      s.sendall("Off")
      time.sleep(1)
except KeyboardInterrupt:
   GPIO.cleanup()
   s.close()
```

Figure-6. LED blink client program.

We import GPIO module to perform physical computing and socket module for communication. We create TCP socket and try to connect to the server. After setting GPIO as BCM (broadcomm) mode, we setup the pin number 25 (pin number 6 for wiringPi) for LED. We send a message (On) to the server when the LED is turned on. We send a message (Off) to the server when the LED is turned off.

```
#-------- blink_server.py -------------
# $ sudo python ./blink_server.py
#---------------------------------------
import socket
HOST = ''
PORT = 8900

s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
try:
   while True:
      data = conn.recv(1024)
      if not data: break
      print(data)
except KeyboardInterrupt:
   conn.close()
```

Figure-7. LED blink server program.

We first create the TCP socket for Internet. We then bind the server's IP address and port number. After waiting for a connection request from the client, we accept the connection requests from client and obtain new socket descriptor instance (conn). We receive a message from the client through conn and display on the screen.

3. WEB BASED CLIENT-SERVER LEARNING MODEL

The client and server based learning model presented in section 2 is efficient in terms of the information delivery and security. However, it has inefficient when we try to control sensors through Raspberry Pi in any place. The reason is why we need to have specific client programs to connect to a server computer. Therefore, a web based model supporting any web browser is required. In other words, if we install web servers in the raspberry pie and access the web server through the web browser, global communications and control can be achieved.

Python provide us with flask as a micro framework for the web service. We install flask under /home/pi/ web_flask_control directory [12]. Figure-8 shows web based learning model on Raspberry Pi.

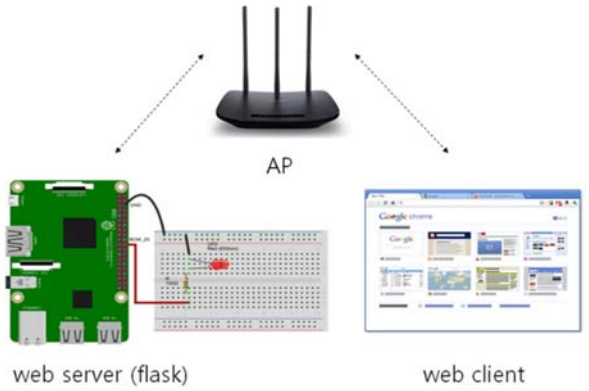

Figure-8. Web based learning model on Raspberry Pi.

The Raspberry Pi should run a web server program using the Bluetooth module and the Wi-Fi (or Ethernet) module. Therefore, we install web server program under the /home/pi/web_flask_control directory.

The approximate code of the web server program is represented in Figure-9.

```
#--- webserver.py ---
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
import time
import datetime
app = Flask(__name__)

PIN = 25
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN, GPIO.OUT)
GPIO.output(PIN, GPIO.LOW)

@app.route("/")
def hello():
   if GPIO.input(PIN) == True:
      message = " On"
      curr_time = datetime.datetime.now()
      time_string = curr_time.strftime("%Y-%m-
                    %d %H:%M:%S")
      time.sleep(1)
      GPIO.output(PIN, GPIO.LOW)
   else:
      message = " Off"
      curr_time = datetime.datetime.now()
      time_string = curr_time.strftime("%Y-%m-
                    %d %H:%M:%S")
      time.sleep(1)
      GPIO.output(PIN, GPIO.HIGH)

   template = {
      'title' : 'LED Status',
      'value' : message,
      'time'  : time_string
   }
   return render_template('index.html', **template)

if __name__ == "__main__":
app.run(host='0.0.0.0', port=80, debug=True)
```

Figure-9. Web server program using flask.

We first import flask module. Whenever we access web server by using web browser, web server sends the current time (time) and the LED status (value) to us through index.html (render_template('index.html', **template)).

In order to provide the web browser with the home page, we set index.html file under the /home/pi/ web_flask_control/templates directory. Figure-10 and Figure-11 shows index.html and the execution result, respectively.

```
<!DOCTYPE html>
<html>
<head>
<title> {{title}} </title>
</head>
<body>
Current time is: {{time}}
<br>
Current LED status is {{value}}
</body>
</html>
```

Figure-10. Index.html.

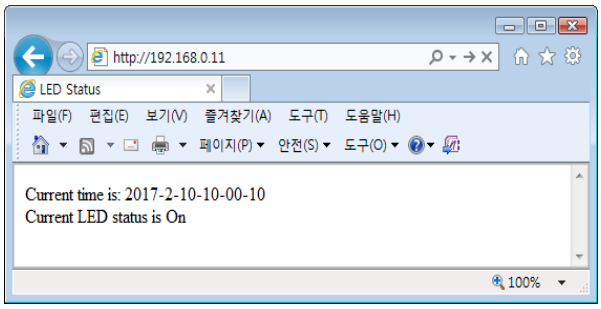

Figure-11. Execution result of index.html.

4. CONCLUSIONS

This paper proposes a practical learning model that is applicable to the school site when teaching the information and communications technologies. The model is developed using the Raspberry Pi platform for physical computing. The proposed learning model is composed of client-server model and web based model using the web flask. The model needs minimal hardware configuration and uses a built-in module. The communication function between the Raspberry Pi is implemented using C and Python socket.

Through this learning model, students can understand information and communications within one integrated framework and acquire the basic concepts of the Internet of Things. In future studies, it is necessary to evaluate the effectiveness of the proposed model through real survey at school sites and to improve the performance of the models quantitatively.

REFERENCES

[1] Surllivan D. and Igoe T. 2004. Physical Computing: Sensing and Controlling the Physical World with Computers, Course Technology Press.

[2] Marquez J., Villanueva J., Solarte Z. and Garcia A. 2016. IoT in Education: Integration of Objects with Virtual Academic Communities. New Advances in Information Systems and Technologies. pp. 201-212.

[3] Bulla C., Hunshai B. and Mehta S. 2016. Adoption of Cloud Computing in Education System: A Survey. International Journal of Engineering Science and Computing. 6(2): 6375-6380.

[4] Arduino. 2016. Arduino open-source prototyping platform. http://www.arduino.cc.

[5] Cheng H. Cheng, L. Hao, Z. Luo and Wang F. 2016. Establishing the Connection between Control Theory Education and Application: An Arduino Based Rapid Control Prototyping Approach. International Journal of Learning and Teaching. 2(1): 67-72.

[6] Kurniawan. 2014. Getting Started with Intel IoT and Intel Galileo. Berlin. October.

[7] The Raspberry Pi Foundation. Raspberry Pi. 2016. http://www.raspberrypi.org.

[8] Sobota J., Pisl R, Balda P. and Schlegel M. 2013. Raspberry Pi and Arduino boards in control education. The Proc. of 10th IFAC Symposium Advances in Control Education. August 28-30. pp. 7-12.

[9] Lee Y. 2016. Physical computing learning model for information and communication education. Journal of the Korea Internet of Things Society. 2(3): 1-6.

[10] Lee Y. 2016. Information and Communication Applications. Green Media.

[11] Wring Pi. 2017. GPIO interface library for the Raspberry Pi. http://wiringPi.com/.

[12] Flask. 2016. Web development. One drop at a time. http://flask.pocoo.org/.