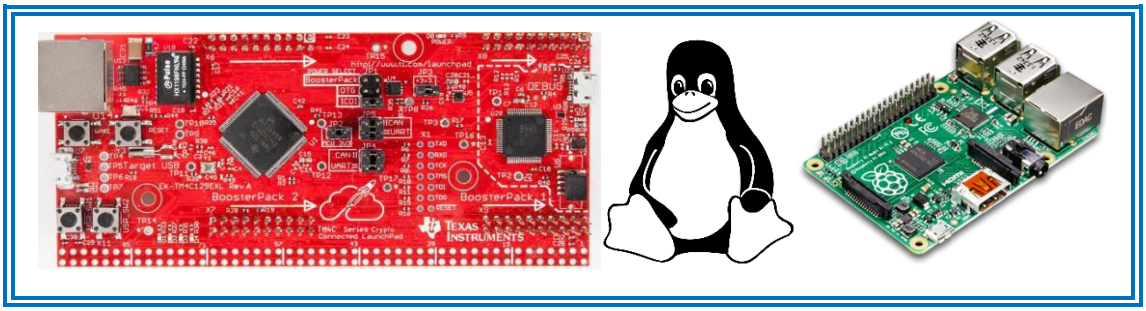
Ho Chi Minh University of Technology

Faculty of Electrical & Electronics Engineering

ELECTRONIC WORKSHOP

Project: Automatic Control via Internet

Instructor: MEng Nguyen Pham Minh Luan



Class: DD15KSVT *Group 6*
Le Van Hoang Phuong *1512579*
Nguyen Chinh Thuy *1513372*
Vo Dinh Tri *1513612*

Wed, November 1, 2017

I. Summary

I.1. Table of Content

I.2. Table of Figure

II. Preface

This document summaries the overall process that we designed and implemented our class project of Electronic Workshop. The topic is ***"Automatic Control via Internet".*** However, we do not just stop by *automatic control*, we developed our project to be made up of *Graphic User Interface* to widen our knowledge and accumulate our design skills.

This topic gives us various chance and challenges as this is the first time we have actually worked in collaboration with each other and approached a thoroughly new embedded system design methodology and PCB design. It also give us chances to accumulate our implementing printed circuit skills.

Our purpose is designing an embedded product containing our available hardware components and capable of connecting to the global network. In detail, we model the topic by designing a "smart door" automatically controlled by a server via Internet. We reckon that perhaps somewhere in the world, this model is put into practice. This model can be illustrated by diagram below:

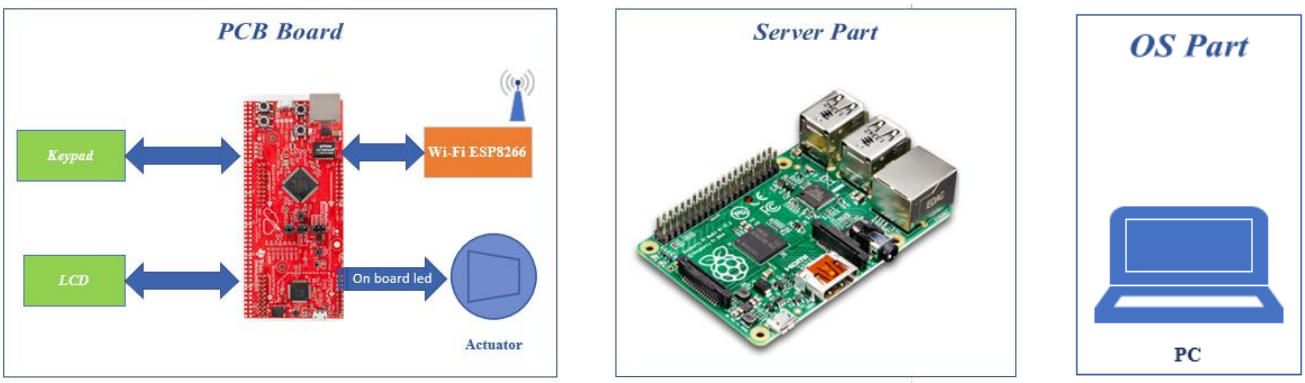


Figure I.2-1. Prototype of Desired Product

III. Market & Version

III.1. Customer's need

[1] Life is growing, people need more "smart" products, able to control automatically.
[2] Personal and household property needs to be kept strictly confidential
[3] Technology products should be simple, easy to use, suitable for those who are less exposed to technology.

III.2. Customer Classification

No.	*Classifications*	*Examples*
1	Customers need simplicity	• Family • Farm owner
2	Customers need security	• Warehouse manager • Laboratory • Computer room
3	Customers need to manage data	• Apartment • Company

III.3. Customer's requirement

- Simple to use
- High security
- Low power
- Low cost
- Managed on many different devices

III.4. Versions

Versions	*Detail*	*Advantage*	*Disadvantage*
1	Username + Password	- Simple, easy to use - Low cost	- Low security - Not aesthetic - Not suitable for children
2	Magnetic card	- High security - Simple, easy to use	- Easy to lose magnetic
3	Fingerprint	- Simple, easy to use - No lose data	- Speed authentication is slow - Easy to fake
4	Iris	- Simple, easy to use - No lose data	- High cost - Low security (the iris sensor on the Galaxy S8 may be fooled by the print image)
5	Face ID	- Simple, easy to use - No lose data - High security	- High cost - Only one face can be identified (Face ID of Iphone X identifies a single face)

IV. Introduction

IV.1. Function

Our system includes two main partitions, ***User interface*** and ***Database server***.

First, the ***User interface*** is responsible for collecting data from user. In our prototype, this data is called account containing username and password (each person has an individual account). After getting verification from user, the ***User interface*** will transfer this data to the ***Database server*** through the Internet for checking whether this login data is correct. Consequently, the result will be sent back to the ***User interface***. If the result reveals correctness, the ***User interface*** will allow the user move through the door, if not, the system will halt for an interval so that no one cannot enter the door.

Additionally, the whole of login history is saved in the ***Database server***, so when the admin want to review who accessed the door, it can use a PC and connect to the ***Database server*** through the Internet with a unique admin account in order to view the login history. Besides, ***Database server*** also provides account management ability for the admin so as to modify the information of door-permitted accessors.

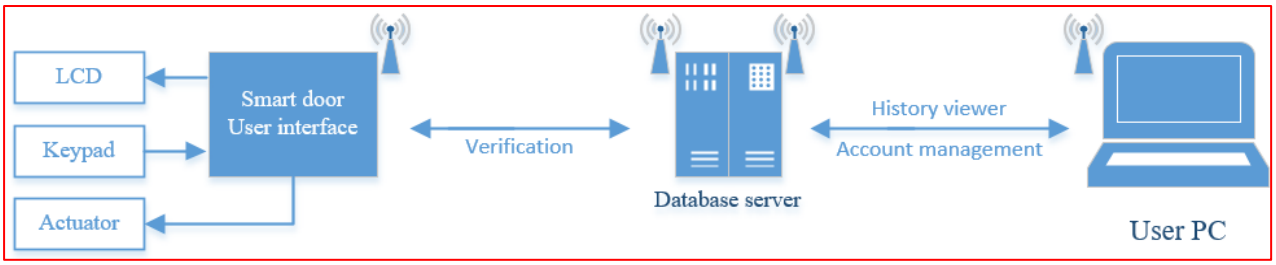


Figure IV.1-1. Overall of the system

IV.2. Prototype

Figure IV.2-1. User interface partition

Figure IV.2-2. Server partition

V. Project Management

V.1. System Specification

V.1.1. Product Specification

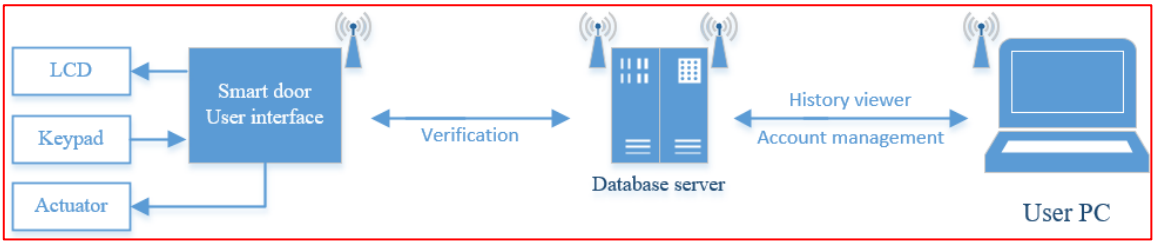


Figure V.1-1. Product block diagram

Product name	"***Smart door***"
Functions	- Verification in the door opening. - Connect to a database server to verify the right to open the door. - Account (right to open the door) management.
I/O	- Input: Account ID, Password (optional). - Output: Actuator (motor to open/close the door) port.
User interface	- LCD: Display input data, verification result. - Keypad: Input the verification data. - PC: Add/Remove accounts in the database server.
External interface	- Wi-Fi: Connect to the database server.
Constraints	- Real-time: Verification time is less than 2s.
Versions	- Being developed (will be ***used in the sequential documents***): TSD102: Username + Password. - Will be considered in the future: TSD302: Magnetic card. TSD312: Magnetic card + Password. TSD502: Fingerprint. TSD702: Iris. TSD902: Face.

V.1.2. Engineering Specification

Hardware components	- Microcontroller: TIVA-C family. - LCD 16x2, Keypad 4x4. - Wi-Fi module: ESP8266. - Power supply: Voltage regulator IC.
Hardware requirements	- Implement a single-layer PCB. - Size: Smaller than an A4 paper. - Overcurrent protection. - Have a button to isolate MCU power from the remaining parts on the PCB.
Firmware components	- Peripheral drivers. - User interface. - Server interface.

Firmware requirements	- Write in C language. - Read button and display the received character from button through LCD. - Send login data to the server and display the received result into LCD.
Software components	- Embedded shell (serve for embedded devices). - OS shell (serve for PC/laptop devices).
Software requirements	- Write in C language. - Establish connection with devices. - Verify login data received from devices. - Record the login data for history viewer. - Allow owner modify which accounts. - Allow owner backup the whole of data on the server. - Have some configuration modes for owner to select.

V.1.3. Hardware Specification

- Overview:

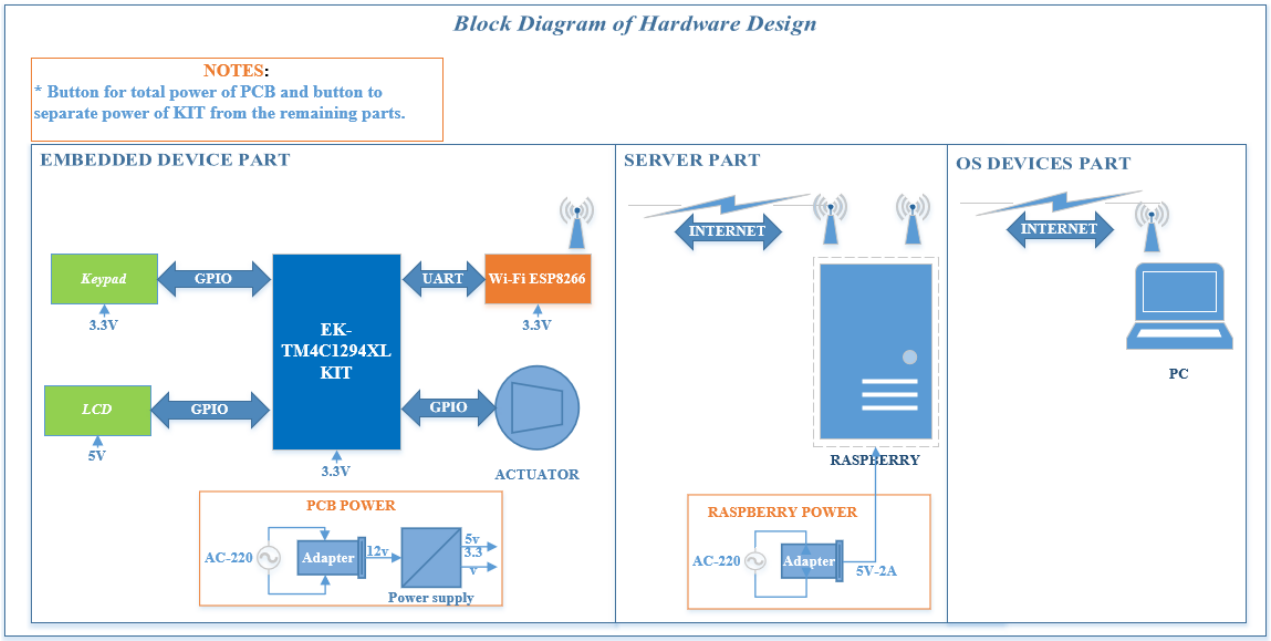


Figure V.1-2. Hardware block diagram

- Component list:

Module	*Specification*
Adapter	Input: 220VAC, Output: 12VDC – 1A
Power supply	IC LM7805: output 5VDC, IC LM1117: output 3.3VDC, Power button for the whole of PCB, Power button for isolating MCU from the rest of PCB
MCU	Kit EK-TM4C1294XL, Booster Packs for plugging Kit
Keypad	Keypad 4x4, The front of keypad is numbered

LCD	LCD 16x2 HD44780, Varistor for adjust the contrast of LCD, Using 4-bit data connection
Wi-Fi	ESP8266-v1

V.1.4. Sotfware Specification

- Overview diagram:

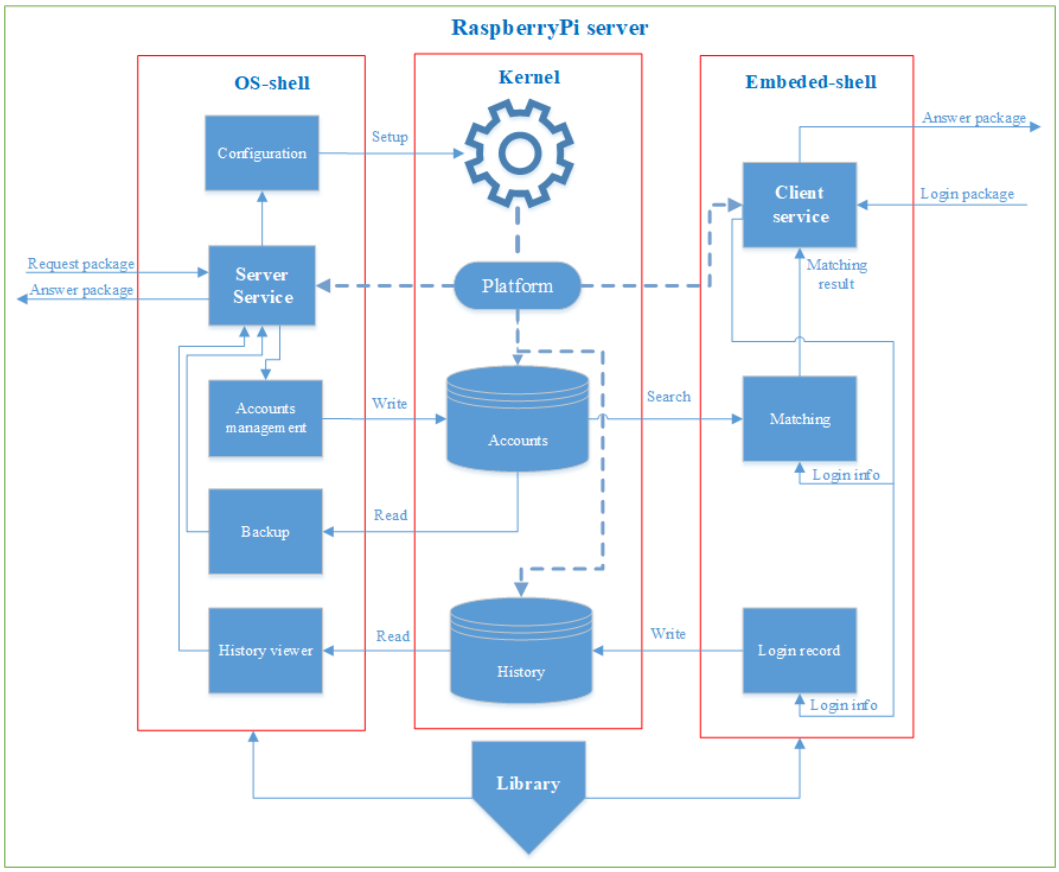


Figure V.1-3. Software block diagram

Detailed information:

Partition	*Module*	*Function*
Kernel	Configuration	Configuration modes of operation of server.
	Platform	Provide threads for Embedded shell and OS shell running.
	Accounts	Account database using for login verification. The file extension: ".acc"
	History	Login record using for history viewer. The file extension: ".log"
Lib		Some mutual and common software libraries for programmers.
Embedded shell	Client service	Receive data from embedded devices, transfer data to the corresponding module, answer the result for embedded devices.
	Matching	Verify the validity of login data by comparing the login information with database.
	Login record	Record time, username, password, and verification result when a login data is received from embedded shell.
OS shell	Configuration	Allow admin to configure operation mode of the server.

	Server service	Receive data from OS devices, transfer data to the corresponding module, answer the result for OS devices.
	Accounts management	Allow admin to modify accounts on the server.
	Backup	Allow the admin backup the whole of database on the server on its OS device.
	History viewer	Allow the admin view the list of login record.

V.1.5. Test Specification

Section	*Content*
Testing equipment	Multi-meter, Oscillator
Testing environment	Strong Wi-Fi signal
Prototype	Self-implemented PCB (single layer)
Testing process	Check Keypad and LCD
	Check Wi-Fi connection of ESP8266
	Check Verification between MCU and Server
	Check History Viewer
	Check Account Management
	Check Data Backup

V.2. System Architecture development

Hardware	- Microcontroller: TIVA-C family. - LCD 16x2, Keypad 4x4. - Wi-Fi module: ESP8266. - Power supply: Voltage regulator IC.
Firmware	- Peripheral drivers. - User interface. - Server interface.
Software	- Embedded shell (serve for embedded devices). - OS shell (serve for PC/laptop devices).
Interface	- LCD driver - Keypad driver - Input/ Output control driver - User Interface

V.3. Design Issues

V.3.1. Constraint issues

Low price	**About 850,000đ**
Long life cycle	5 years
Reliability	Data is not encrypted
Response time for control	Less than 2s
Data transmission over the Internet	Less than 2s

V.3.2. Functional issues

- Connection can be corrupted, system must automatically reconnect.
- Devices can be overheated for long time working so it needs to be detected by a sensor.
- Door controlling devices can be broken or got stuck, it needs a component to detect and go off.
- Internet access points can be crashed, it needs a device to alarm.
- Such malfunctions as wrong data comparison, wrong control could cause damages to security and economy.

V.3.3. Real-time issues

Components	Real-time
Device for control the door	Soft real-time: delay less than 1s
Server Computer	Soft real-time: delay less than 1s
Micro-controller	Soft real-time: delay less than 1s
LCD display	Soft real-time: delay less than 0.5s
Keypad	Soft real-time: delay less than 0.5s

V.3.4. Concurrent issues

Server computer	**Multi-access**	**Serve clients and admins simultaneously**
Microcontroller	Multi-task	Operate keypad, display LCD, connection to server, compare data, control the door simultaneously

V.3.5. Reactive issues

- ❖ Continuous: server computer must run 24 hours a day to serve clients and admins.
- ❖ Non-periodic event response:
  - People use keypad.
  - Connection is crashed.
  - Administrators accesses.

V.4. House of Quality

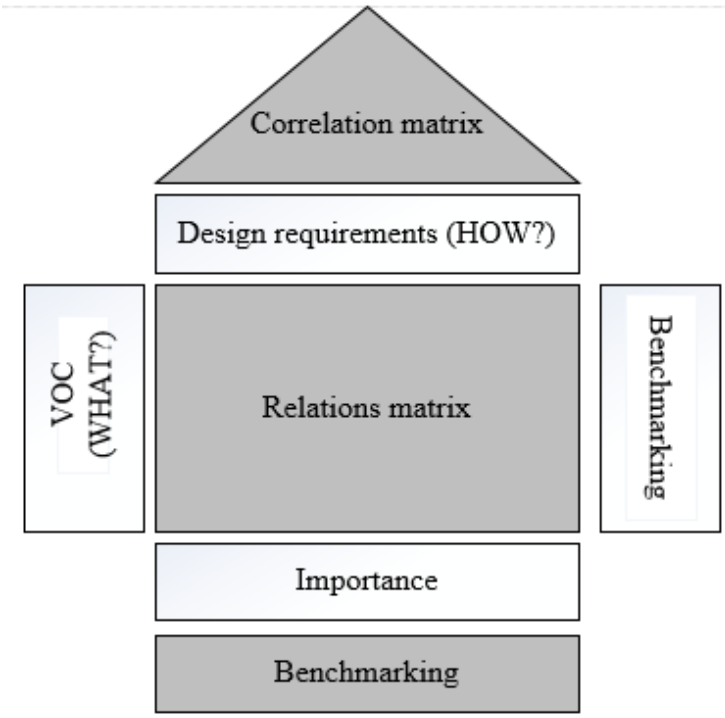


Figure V.4-1. House of quality

V.4.1. What?

No.	***Customer's requirement***
1	Simple to use
2	High security
3	Low power
4	Low cost
5	Managed on many different devices

Table V.4-1. Table of What

V.4.2. How list

No.	***Design requirement***
1	Low power microcontroller
2	Read keypad and display through LCD
3	Verify login data received from devices
4	Record the login data for history viewer
5	Backup the whole of data on the sever

Table V.4-2. Table of How list

V.4.3. Relation matrix

Design requirement / Customer's requirement	Low power microcontroller	Read keypad and display through LCD	Verify login data received from devices	Record the login data for history viewer	Backup the whole of data on the sever
Simple to use	**W**	**S**	**W**	**W**	**W**

High security	**W**	**M**	**S**	**S**	**M**
Low power	**S**	**S**	**W**	**W**	**W**
Low cost	**S**	**S**	**W**	**W**	**W**
Managed on many different devices	**W**	**W**	**M**	**M**	**S**

Table V.4-3. Table of relation matrix

W = weak *M = medium* *S = strong*

V.4.4. Benchmarking

Design requirement / Customer's requirement	Low power micro-controller	Read keypad and display through LCD	Verify login data received from devices	Record the login data for history viewer	Backup the whole of data on the sever	Bad	Average	Good
Simple to use	**W**	**S**	**W**	**W**	**W**	x		
High security	**W**	**M**	**S**	**S**	**M**			x
Low power	**S**	**S**	**W**	**W**	**W**			x
Low cost	**S**	**S**	**W**	**W**	**W**		x	
Managed on many different devices	**W**	**W**	**M**	**M**	**S**		x	

Table V.4-4. Table of Benchmarking

V.4.5. Importance level

Design requirement / Customer's Requirement		Low power micro-controller	Read keypad and display through LCD	Verify login data received from devices	Record the login data for history viewer	Backup the whole of data on the sever	Bad	Average	Good
What	Importance								
Simple to use	2	**W**	**S**	**W**	**W**	**W**	x		
High security	5	**W**	**M**	**S**	**S**	**M**			x

Low power	1	**S**	**S**	**W**	**W**	**W**			x
Low cost	4	**S**	**S**	**W**	**W**	**W**		x	
Managed on many different devices	3	**W**	**W**	**M**	**M**	**S**		x	
Importance		**55**	**81**	**61**	**61**	**49**			

Table V.4-5. Table of importance level

Strong = 9 *Medium = 3* *Weak = 1*

V.4.6. Correlation matrix

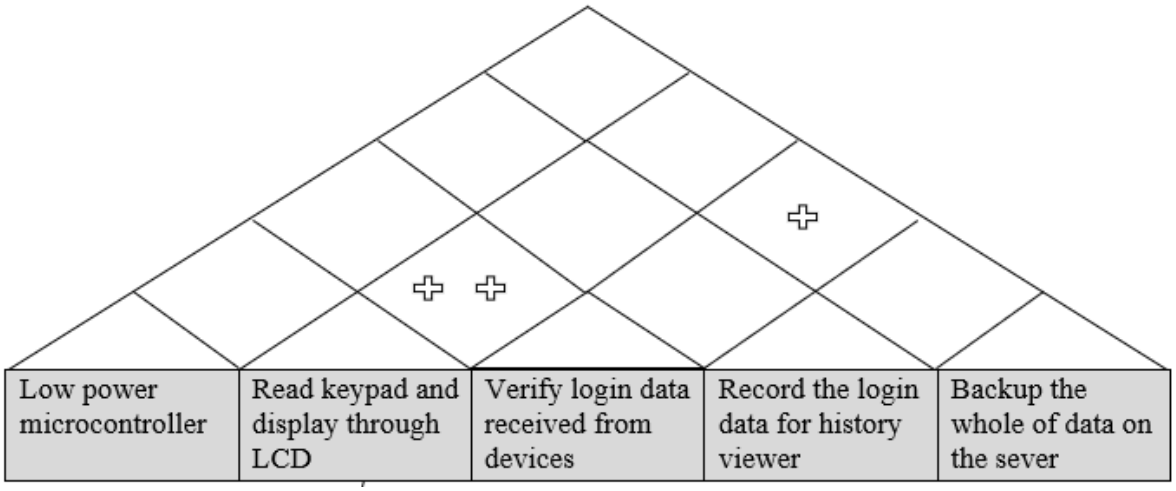


Figure V.4-2. Table of correlation matrix

- ++ strong positive
- \+ positive
- \- negative

V.5. Analyzing the impacts on the economy, environment, society and globe of your project's topic.

V.5.1. Identifying the social, economic, environment and global contexts related to embedded systems.

- ❖ Economy: saving time for ones to open the smart door, saving money for hiring guards.
- ❖ Society: consolidating security & stability of the society

V.5.2. Explaining the implications of technical solutions when designing your embedded system in those contexts.

- ❖ Economy:

- Integrating more useful features and optimizing the product to reduce one's costs.
- Power saving mode from 11 PM to 5 AM to reduce the electric bills.

❖ Society: technical solution is enhancing security for the system by using such ways as encrypting data.

V.5.3. Considering selection of technical solutions (including hardware and software) and select the appropriate solution basing on those effects.

No.	*Partition*	*Sol.1*	*Sol.2*	*Decision*
1	MCU	STM8: Low power/cost	TM4C1294NCPDT: High speed, Wide memory	#2: Available kit/experience, large resources to expand the application
2	Keypad	Plug-header 4x4 keypad	Solder-header 4x4 keypad	#1: Aesthetics
3	LCD	LCD 16x2 44780		#1: Widespread usage
4	Wi-Fi	Arduino Wi-Fi shield	ESP8266-v1	#2: Smaller cost, firmware programmability
5	Actuator	LED	Motor	#1: To just simulate the system, using Sol.1 is cheaper.
6	Server type	PHP server: Easier, Available for installing	C server: More difficult, cost a lot of time to build	#2: We want to try programming a server in C language so that we can improve C language programming skill.
7	Server hardware	Raspberry Pi	PC	#1: Available, Save energy
8	Server OS	Windows	Linux	#2: Free license, optimized for the hardware, secured, built-in platform.

VI. Designing

VI.1. Principle

VI.1.1. Block Diagram

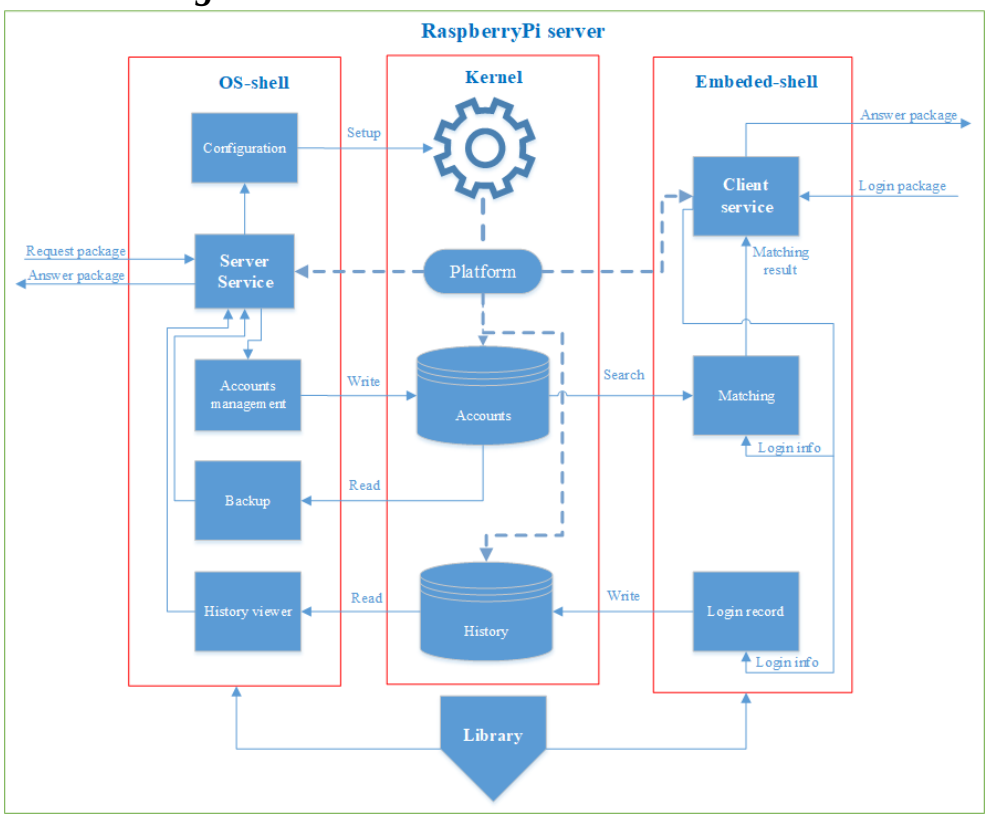


Figure VI.1-1: Block Diagram of Designing

VI.1.2. Frame of Data Transmission

0x01	*0x02*	*address*	*function*	*Number of data's byte*	*data*	*0x03*	*0x04*

Where:

0x01, 0x02	Start of frame
0x03, 0x04	End of frame
Address	Each admin/embedded device assigned to a specific ***number*** regarded as address
Function	We define services a ***number***
Number of data's byte	In unit byte

VI.1.3. Control Scheme

Each user (embedded device or admin) is delivered some specific services defined in a list, below is the list of delivered services.

List of services delivered for specified user is formed like below:

```
{address 0, service 0, a callback function to service 0},
{address 0, service 1, a callback function to service 1}
```

VI.1.4. Connection Prototype

Connection process of data link layer is illustrated by diagram below:

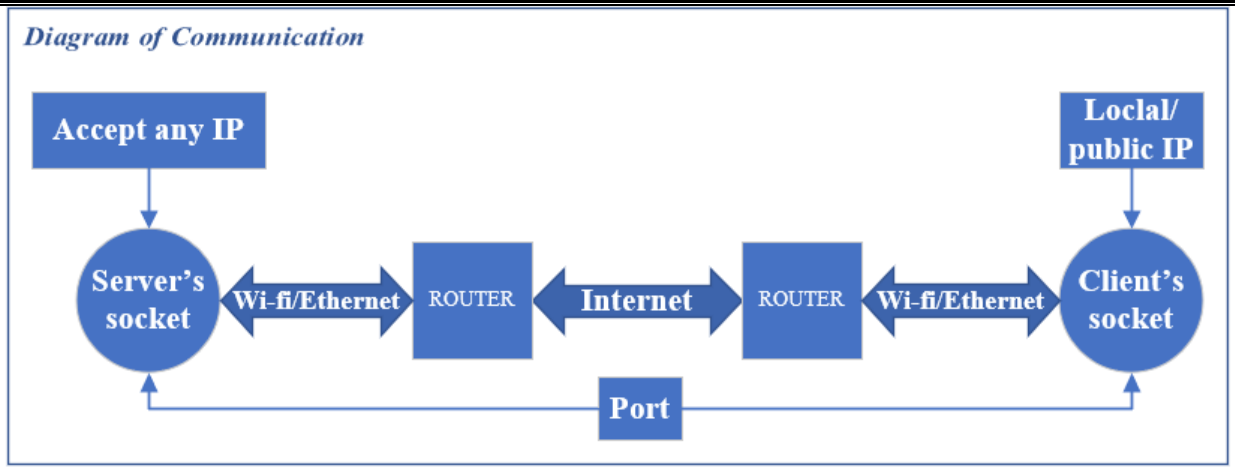


Figure VI.1-2. Socket connection

VI.1.5. Server & Client Model

VI.1.5.1. Server

Server can be a PC or an embedded computer. In this project, we concentrated on an embedded computer, say Raspberry Pi 3.

This server runs a program that stores accounts and responses to user. This program is considered as cloud database.

VI.1.5.2. Client

Clients are both embedded device and PCs. These clients require services according to its device's type.

VI.1.6. Operation

VI.1.6.1. Server

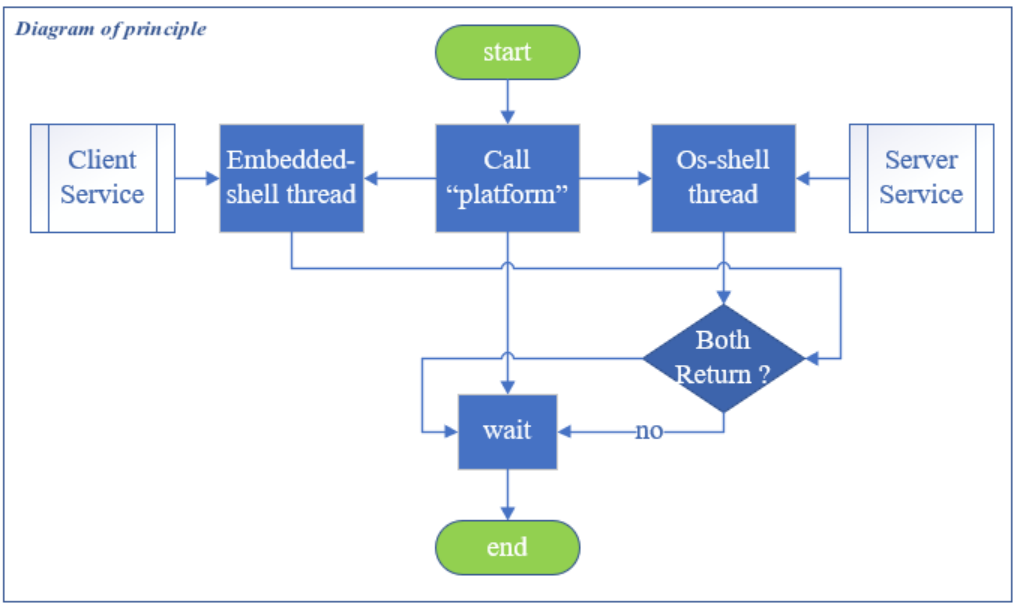


Figure VI.1-3. Diagram of server's operation

The most important server's function in our project is multi-thread. The platform block first creates 2 major threads, they are ***embedded-shell*** thread and ***os-shell*** thread. ***Embedded shell*** thread is used to serve embedded device, namely TivaC129 attached on PCB board. ***os-shell thread*** is in charge of serving devices using UNIX-like operating system (we call them **"os device").**

How can this server distinguish between embedded device's connection and os device's connection? Each shell is set to monitor only one "port number", so there are 2 different ports for 2 types of device and they cannot be misunderstood.

Each thread will control the service distribution center, say **"server service"** and **"client service**", and interacts with accordant devices with TCP/IP prototype. The **server service** manages 4 services and **client service** manages only ***"embedded login"*** service, see System Specification for more information. "***embedded login***" service will log clients' information to "***History***" each time._System_Specification

Once starting, server will always wait for server both device's types unless the admin stop it by using "configuration" service.

VI.1.6.2. Embedded Device

Embedded devices gain connection to server by using port number defined by ***embedded-shell***. Server will receive their requests and start it tasks, namely ***"embedded login".***

See Connection Prototype**.**_Connection_Prototype

VI.1.6.3. OS Device

OS devices set up their socket combined with the port number defined by os-shell and connect to server. Afterward, os-shell will receive their requests and call service distribution center to deal with it, namely ***"serverService".***

VI.2. Hardware Design

VI.2.1. Hardware Component

1. Microcontroller: 32- bit microcontroller: TM4C1294NCPDT.
2. Peripherals:
   - Input devices: Switch, kepad 4x4
   - Display devices: LED, text LCD.
   - Actutors: LED (for prototype).
   - Interfaces: UART, wifi.
3. Clock/ reset circuits: on board Tiva.
4. Power supply: use AC/ DC adapters 12V.
5. Connector:
   - Power connector
   - Header.

VI.2.2. Hardware Block Diagram

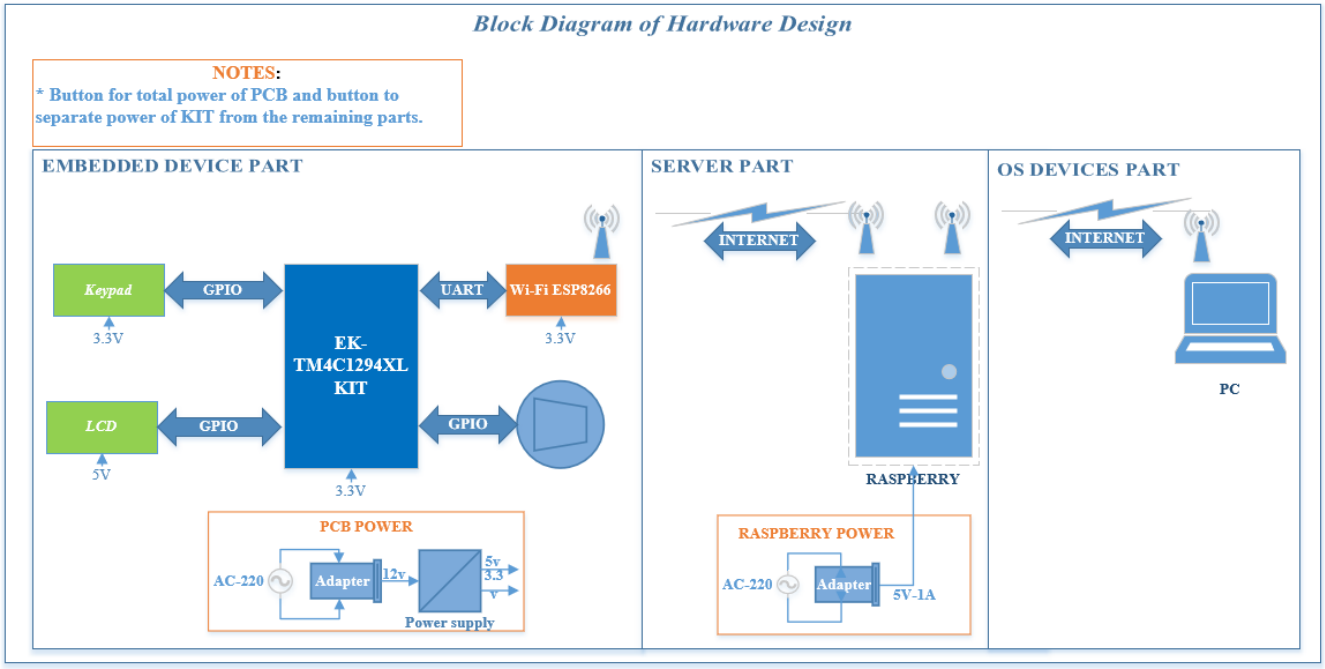


Figure VI.2-1. Hardware Block Diagram

VI.2.3. Schematic

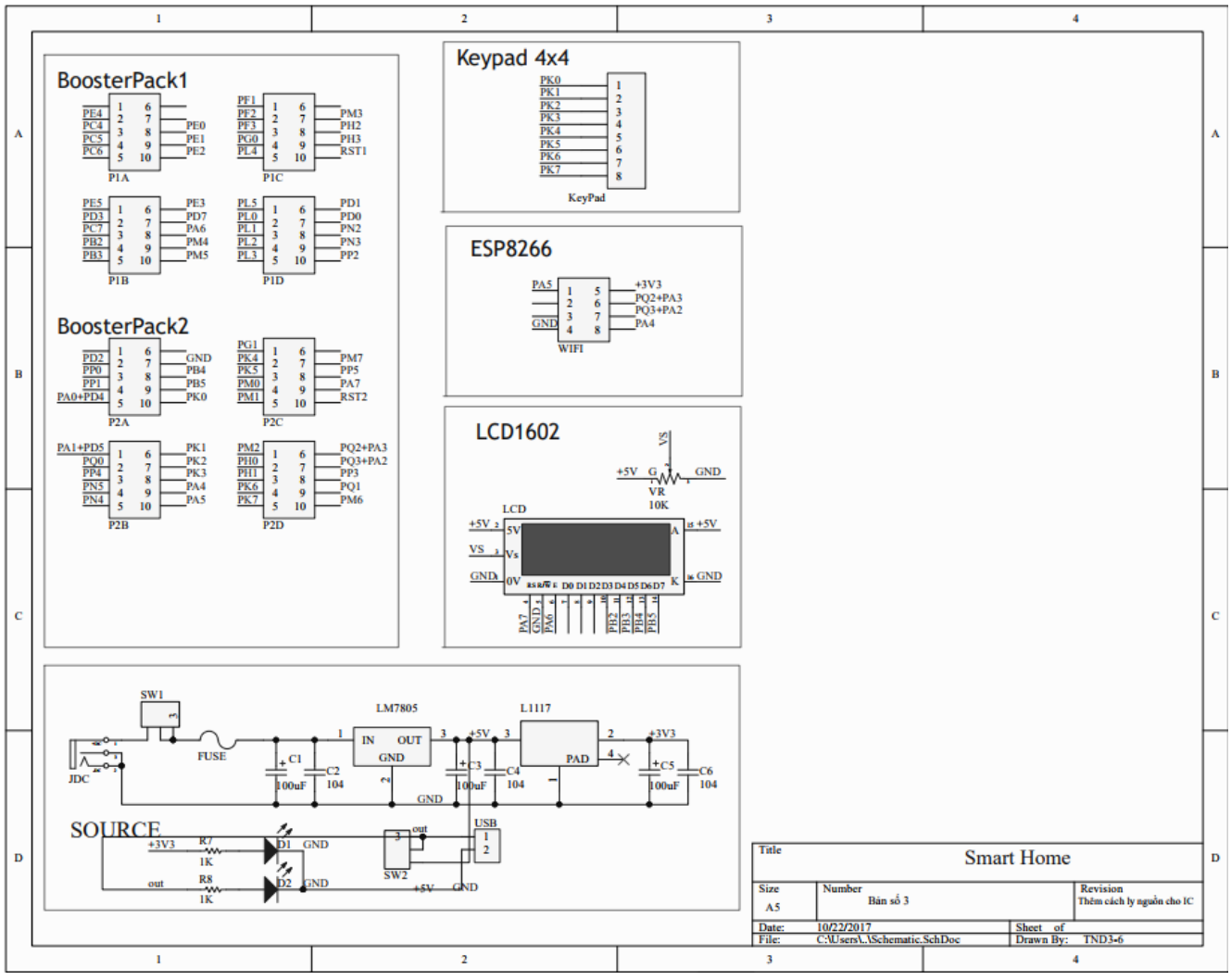


Figure VI.2-2. Schematic

VI.3. Firmware Design

This section interprets about firmware. To design firmware methodically, we follow this process.

Figure VI.3-1. Firmware design process

As the result, subsequent sub-sections also follow this process. Because, requirements and specifications are mentioned in the section **V**, so in this section, we start from listing components, then figure out our peripheral library as well as modules, last but not least, all of modules will be united.

VI.3.1. Components

Modules	Component	Connection to MCU
Microcontroller	TM4C1294NCPDT	EK-TM4C1294 Kit
Interface	LCD 16x2 Keypad 4x4	GPIO
Wi-Fi	ESP8266	UART
Actuator	LED (for prototype)	GPIO

VI.3.2. Peripheral control

VI.3.2.1. LCD 16x2

LCD 16x2 has two modes of control, 4- and 8-bit data. To save resource, we decide to communicate with the LCD in 4-bit mode. For convenience, we write a library for controlling LCD 16x2, this library has functions below:

Function	Description	Input	Output
`lcdSetup`	Setup GPIO for LCD, Setup LCD	None	None
`lcdClearScreen`	Clear screen, Move cursor to original	None	None
`lcdChangeLine`	Change the current line	(bool) line_th: false for the 1st line, true for the 2nd line.	None
`lcdDisplay`	Display a string to screen	(char*)str: String to display to screen	None
`lcdAddChar`	Add a character to the next position	(char)ch: The adding character	None

VI.3.2.2. Keypad 4x4

To control Keypad 4x4, we use the "***Sweep***" method. It can be explained as the following figure.

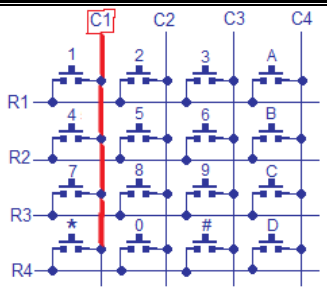


Figure VI.3-2. Keypad 4x4

Denote that C1-C4 is output signals and R1-R4 is input signals, which connect to the MCU. At a time, assume C1 is activated over the deactivated remaining ones. If user press button [1], subsequently, R1 is also activated and MCU can read this activation to know that button [1] is being pressed. Similarly, we need to activate each column in a specific interval so that MCU can read which button is pressed.

Based on this method, we built a library for controlling a keypad.

Function	Description	Input	Output
`kpSetup`	Setup GPIO for Keypad	None	None
`kpSweep`	Sweep output signals	None	None
`kpCheck`	Read input signals to indicate pressed button	None	None

VI.3.2.3. ESP8266

ESP8266 is a Wi-Fi chip with full TCP/IP stack. Besides, its communication is UART with supporting AT command set.

Function	*AT Command*	*Response*
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1) +CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1) OK
Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>, <port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>: (CIPMUX=1): + IPD, <id>, <len>: <data>

Figure VI.3-3. ESP8266 AT command set

Take this platform into account, we built our library for ESP8266 as the following table.

Function	Description	Input	Output
`Wi-FiSetup`	Setup GPIO for ESP8266, Setup ESP8266	None	None
`Wi-FiCheckServerConn`	Check whether the connection with server is established	None	(bool): true if connection has been established, false if not
`Wi-FiConnServer`	Connect to server	None	None
`Wi-FiDisconnServer`	Disconnect from server	None	None
`Wi-FiSendData`	Send data to the server	(char*)usr: Username	None

		(char*)pass: Password	
`Wi-FiRecData`	Wait for receiving data from server	None	(bool): true if the result is correct, false if not

VI.3.2.4. Led

Because the EK-TM4C1294 Kit has four LEDs so that we use these available LEDs for the prototype. There are some functions for controlling LED.

Function	Description	Input	Output
`actSetup`	Setup GPIO for LEDs	None	None
`actServe`	Control LED following the result received from the server	(bool)result: If true, turn on LED for 1s, else do not turn on LED	None
`actIdle`	Toggle LED in idle time for notifying the system is not trapped	None	None

VI.3.3. Interface module

Interface module includes LCD 16x2 and Keypad 4x4 that the user inputs character from the Keypad, and LCD will display it into the screen. At the beginning, we represent meaning of buttons on the Keypad.

Figure VI.3-4. Keypad 4x4

[0]-[9], [A]-[D] : Character buttons
[*] : Backspace
[#] : Confirm

Because an account contains of Username and Password, so user must input data two times. To be appropriate with this, we make two relevant pages, one for inputting username, the rest for inputting password. In the username page, character is raw, however, in the password page, character is encrypted as ***. Nonetheless, to bring the Interface to be familiar, we added two more pages, clearly, idle page and result page. In the idle page, LCD just displays the string “Welcome”, whilst, result page notifies the verification result. Furthermore, the result page only exists in a short interval (e.g., 1 second for a right result and 5 second for a false result), then the screen will return to idle page.

To summarize all of the Interface, we expose the algorithm to implement it by the following diagram.

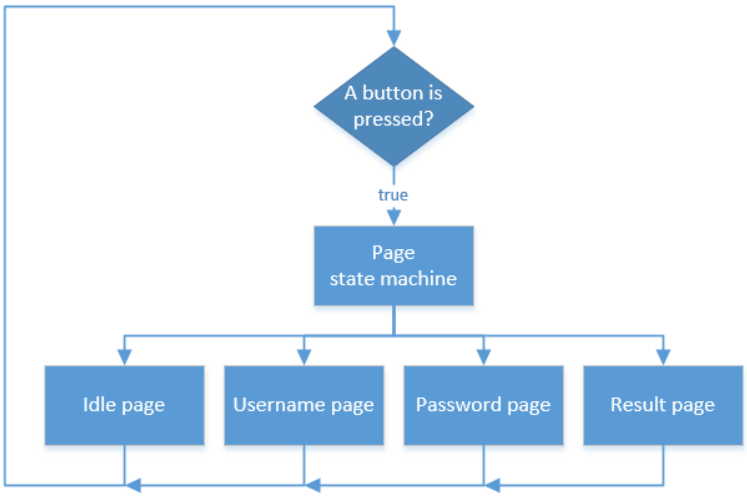


Figure VI.3-5. Interface implementation

VI.3.4. Wi-Fi module

To control ESP8266, programmer must obey the AT command set. Thank to this command set, we can ignore Wi-Fi handshake protocols and use UART protocol as the alternative. However, a barrier rising up is that the MCU must automatically analyze the received string from ESP8266 to be aware of the current state. For instance, when MCU sends a command to ESP8266, ESP8266 will implement its work and the return a result string, to know this result, MCU must find a key character in the received string. To overcome this obstacle, we introduce an algorithm (see Figure VI.3-6. Automatic analysis algorit) to analyze the returned string from the ESP8266.

Besides, the main function of ESP8266 is transferring data through Wi-Fi, so we continue introducing an algorithm (see Figure VI.3-7. Send data to server) to transfer data from ESP8266. To send data, MCU must check the connection with server, if the connection does not exist, it will tell ESP9266 connect to the server. After the handshake is established, MCU encodes data and sends it to the server.

Finally, we want to announce the Data package algorithm, which is used to encode/decode the data after sending/receiving. As mentioned in the section **VI.1.2**, a data frame includes four fields (e.g., address, function, length of data, data), so our Data package block has one terminal is fields, and the remaining terminal is the frame. In this context, we use a state machine to build this block (see Figure VI.3-8).

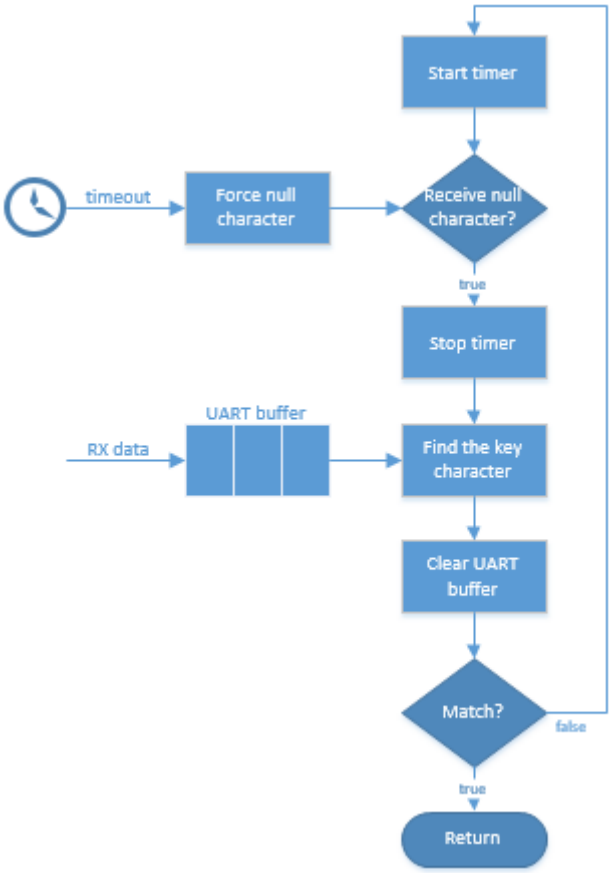


Figure VI.3-6. Automatic analysis algorithm

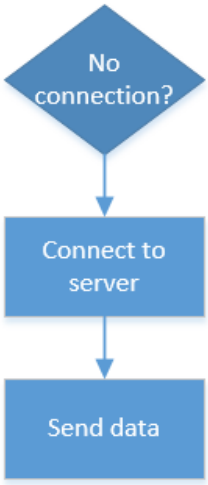


Figure VI.3-7. Send data to server

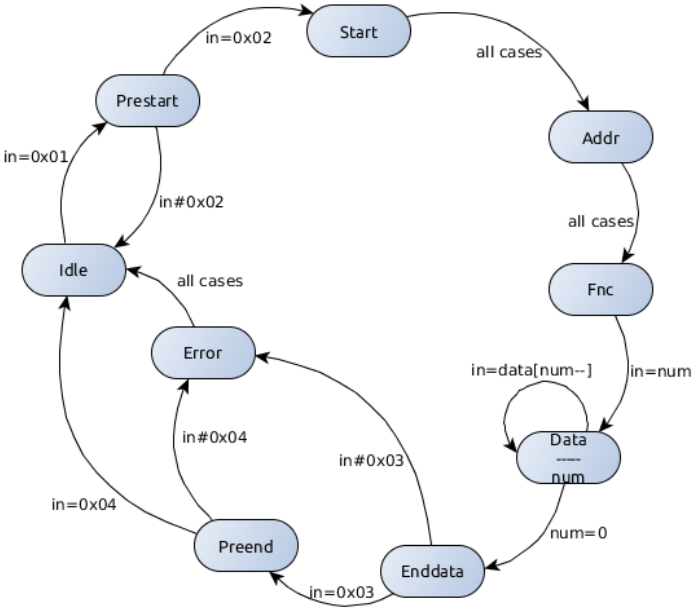


Figure VI.3-8. State machine of data frame decoding

VI.3.5. Actuator module

Actuator is the final module of firmware, so it uses result from previous modules to deicide its behavior. If the result is true, the notification LED is turned on for 1 second, else, this LED is not turned on. Moreover, we also provide an alive LED that continuously toggle so that people can know the system is still alive by observing this LED. Furthermore, a Wi-Fi LED is included that it will light when the data transaction is being taken place. If this LED is turned on too long,

the Wi-Fi connection maybe corrupted, so user can be aware of this situation and try to fix the Wi-Fi connection.

VI.3.6. Unite the whole of firmware modules

After completing all of modules, we start combining them. In order to obtain a successful combination, we specify each independent input and output.

Module	Input	Output
Interface	String to be displayed	Pressed button
Wi-Fi	Encoded data frame to be sent	Received encoded data frame
Actuator	Verification result	None

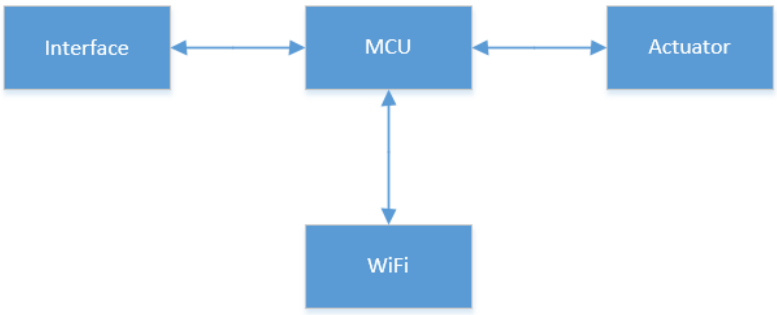


Figure VI.3-9. United firmware

VI.4. Software Design

We used C programming language and a variety of supporting libraries to create Graphic User Interface which administrator can see the operating progress on a Linux distro' terminal.

To avoid failures and speed up programming process, we design software structurally by programming one side in parallel with the other and design algorithms in form of a couple.

VI.4.1. Main function Algorithm

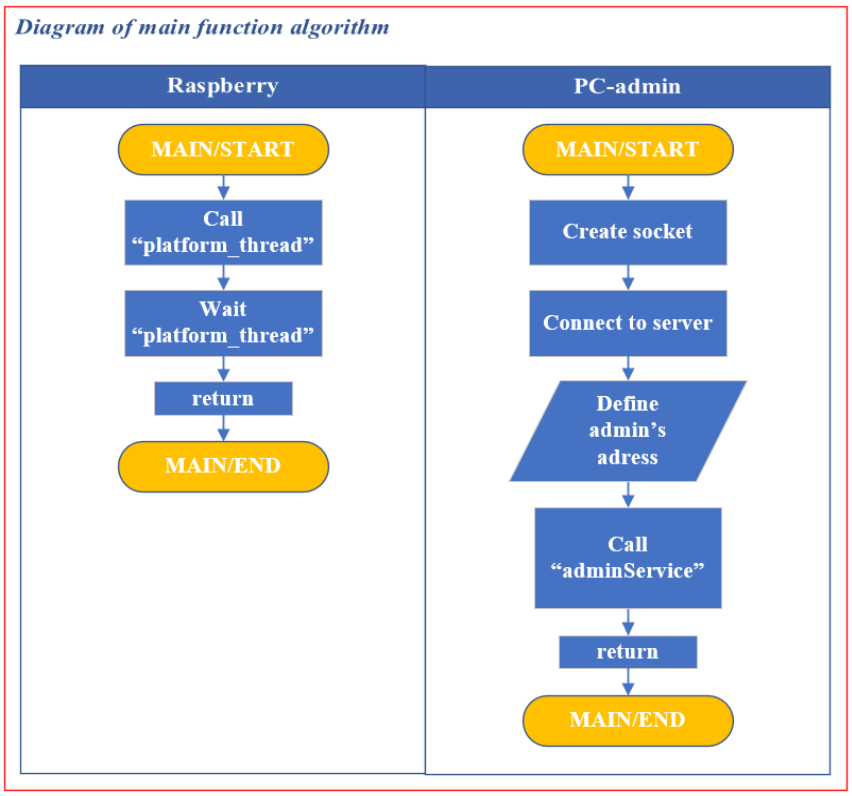


Figure VI.4-1. Main Function Algorithm

VI.4.2. Platform Algorithm

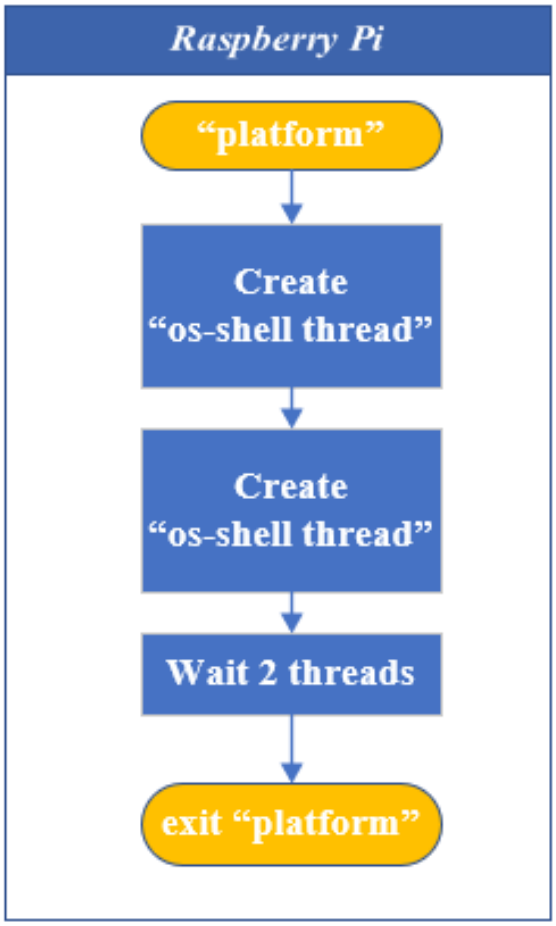


Figure VI.4-2. Platform algorithm

VI.4.3. Os-shell Algorithm

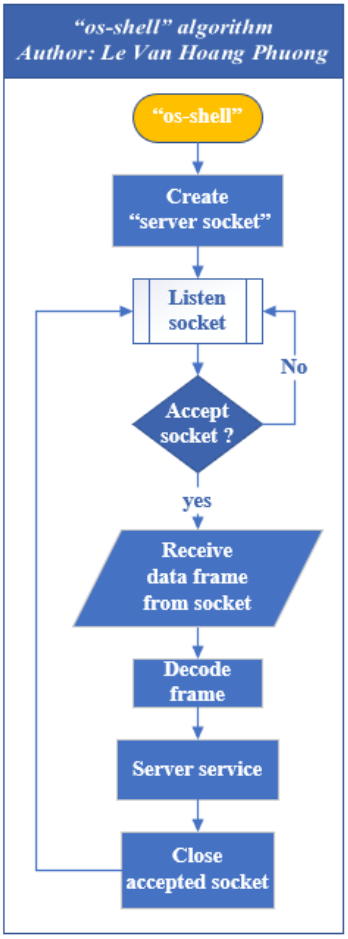


Figure VI.4-3. os-shell algorithm

VI.4.4. Server Service Algorithm

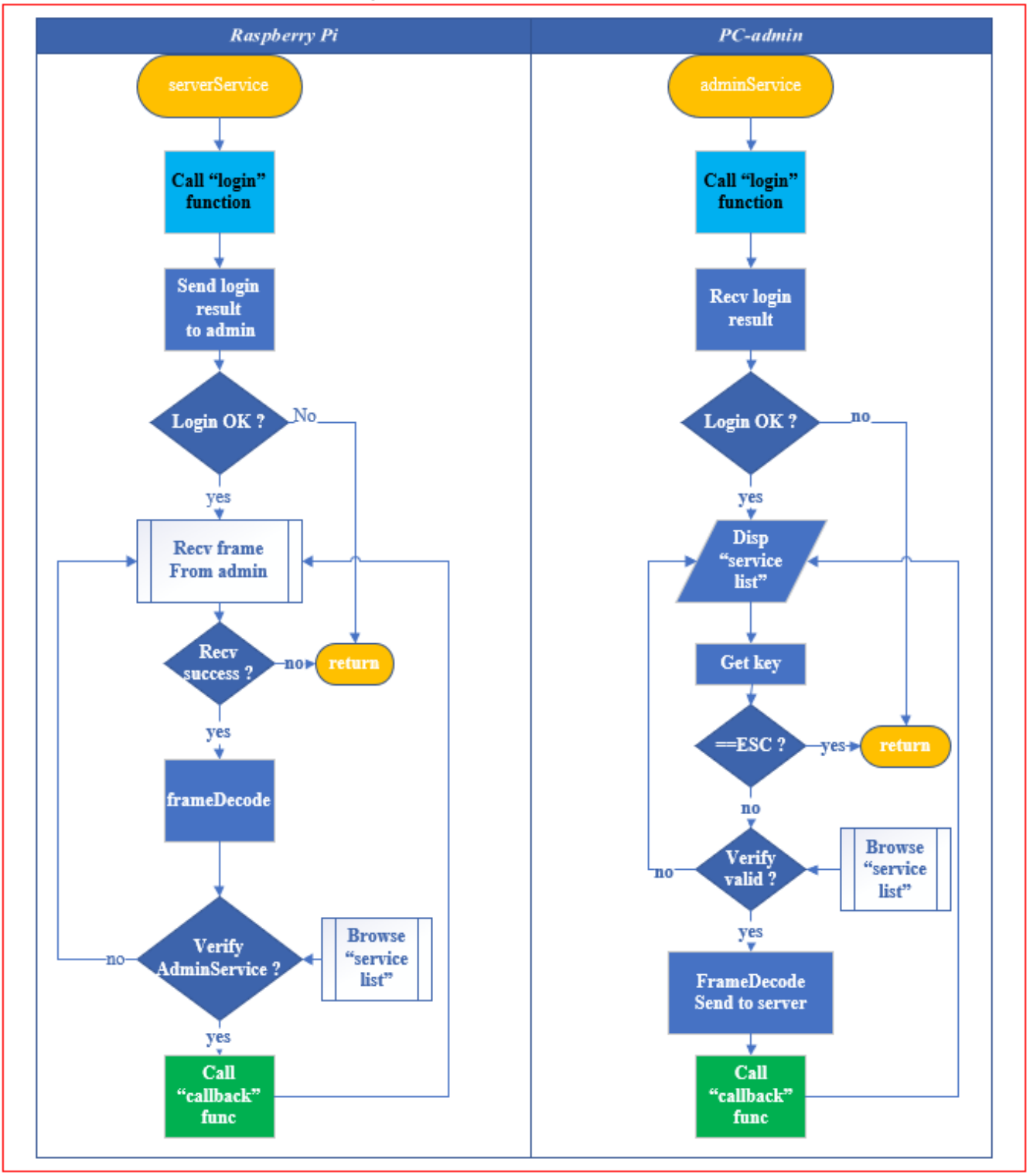


Figure VI.4-4. serverService & adminService algorithm

VI.4.5. Backup Algorithm

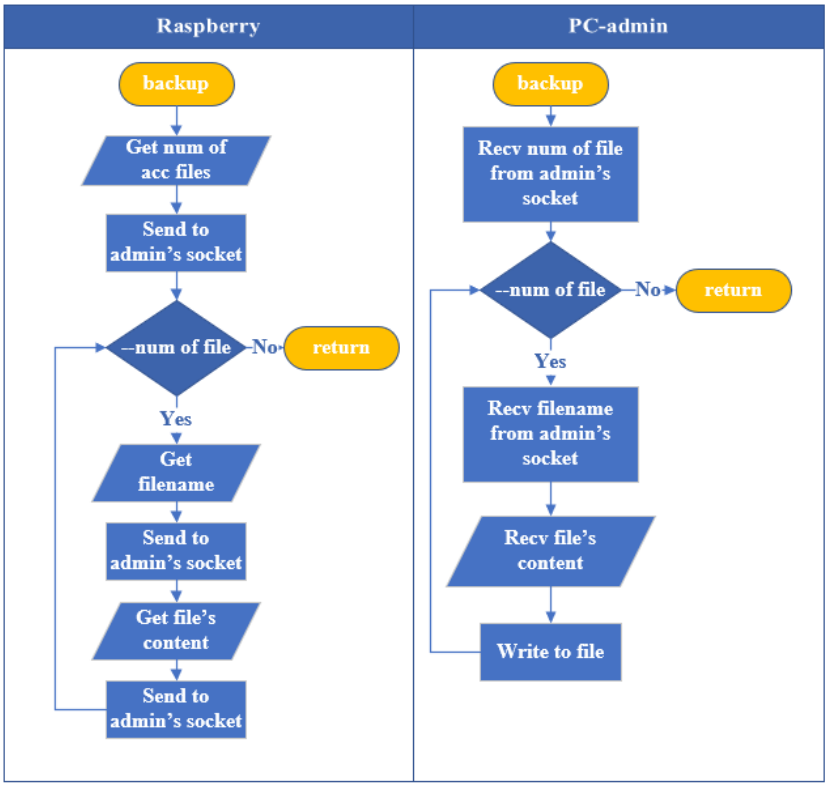


Figure VI.4-5. Backup algorithm

VI.4.6. Account Management Algorithm

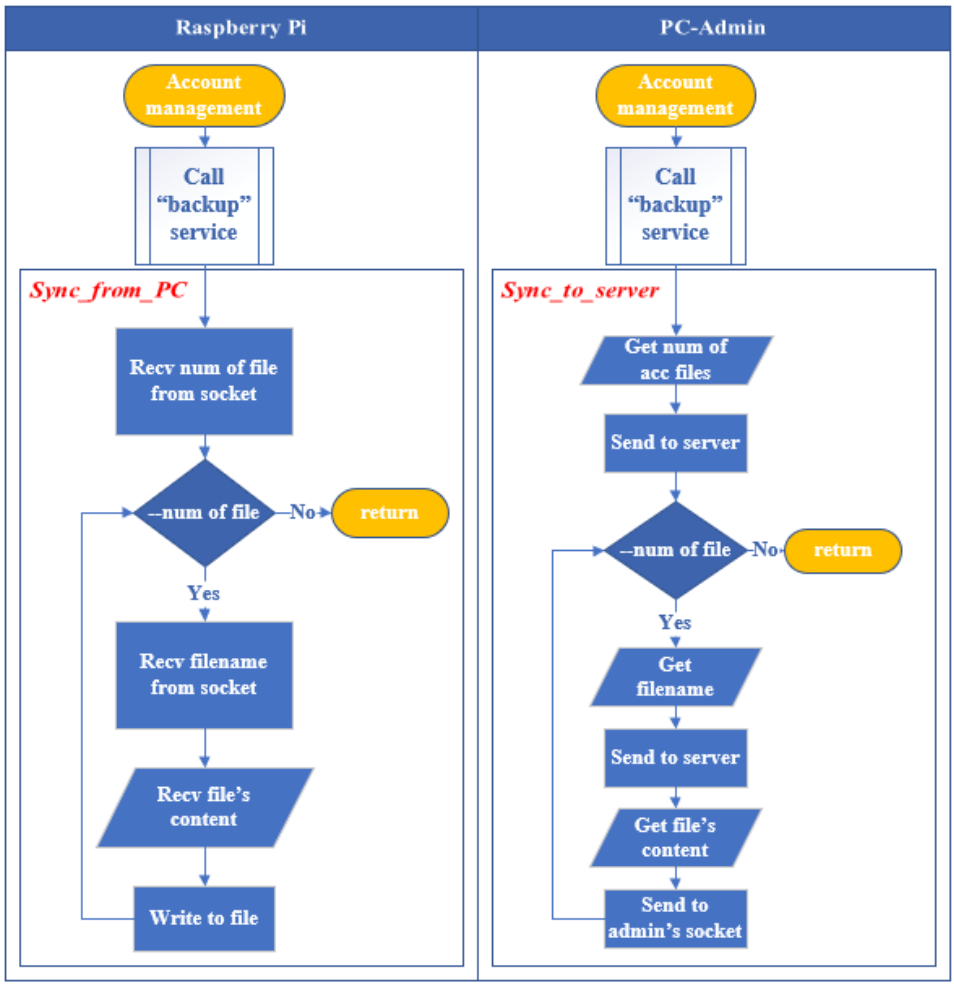


Figure VI.4-6. Account management algorithm

VI.4.7. Configuration Algorithm

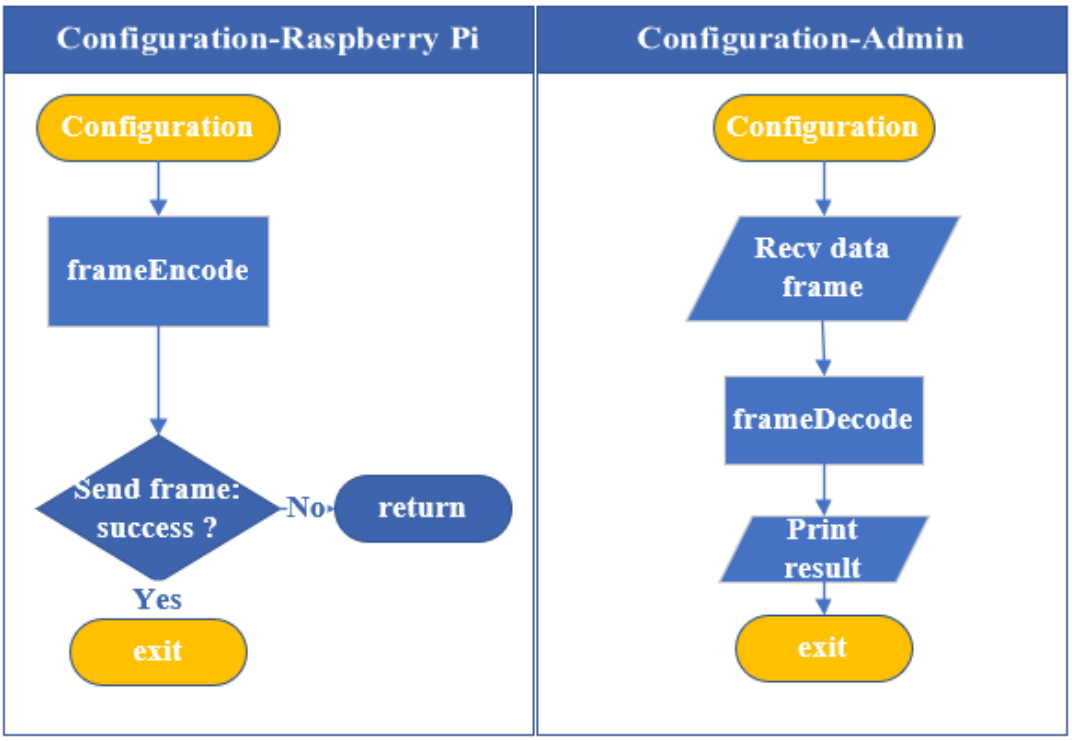


Figure VI.4-7. Configuration algorithm

VI.4.8. History Viewer Algorithm

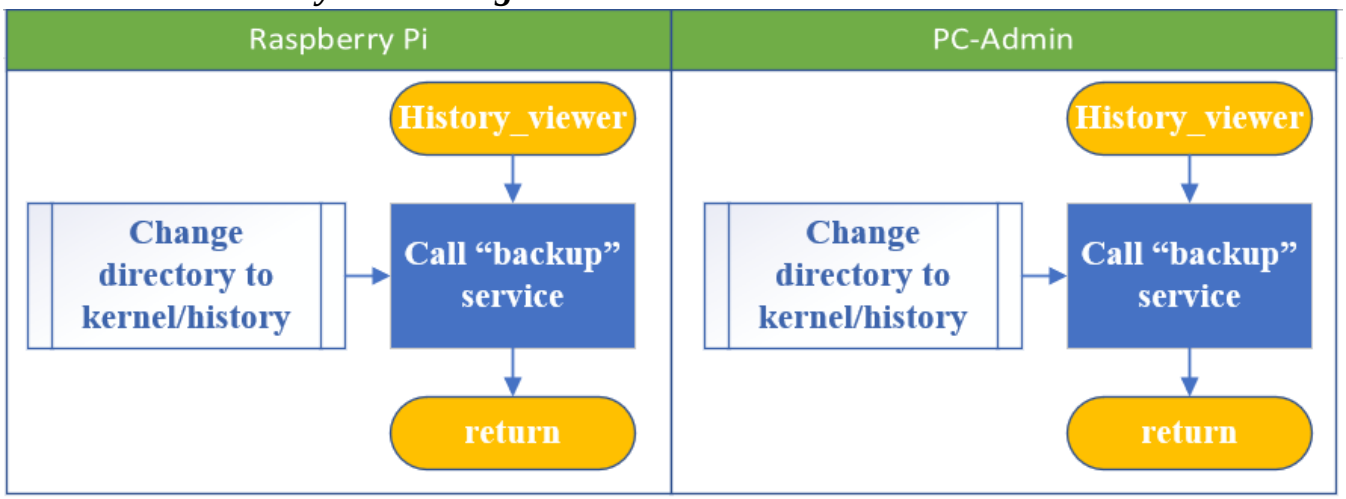


Figure VI.4-8. History_viewer algorithm

VII. Operating Condition

VII.1. Ideal Condition

[1] Strong Wi-Fi signal.
[2] People use keypad to do the procedure actively.
[3] No power interrupts at user's site.
[4] No power interrupts at server's site.

VII.2. Real Condition

[1] Enough strong Wi-Fi signal level, Wi-Fi can be crashed.
[2] Power can be interrupted at user's site.
[3] No power interrupts at server's site.

VIII. Result of Testing

VIII.1. Networking

Tool	Arduino UART terminal, Linux terminal
Condition	Strong Wi-Fi, Local network
Target	ESP8266 connects to the server (handle) and sends an arbitrary data in one direction
Process	Build a simple server by Linux terminal. Use Arduino terminal, type AT command to connect ESP8266 to the server. Use Arduino terminal, type AT command to send an arbitrary data to the server.
Result	ESP8266 can connect to the server. The data is received correctly by the server (see Figure VIII.1-1).

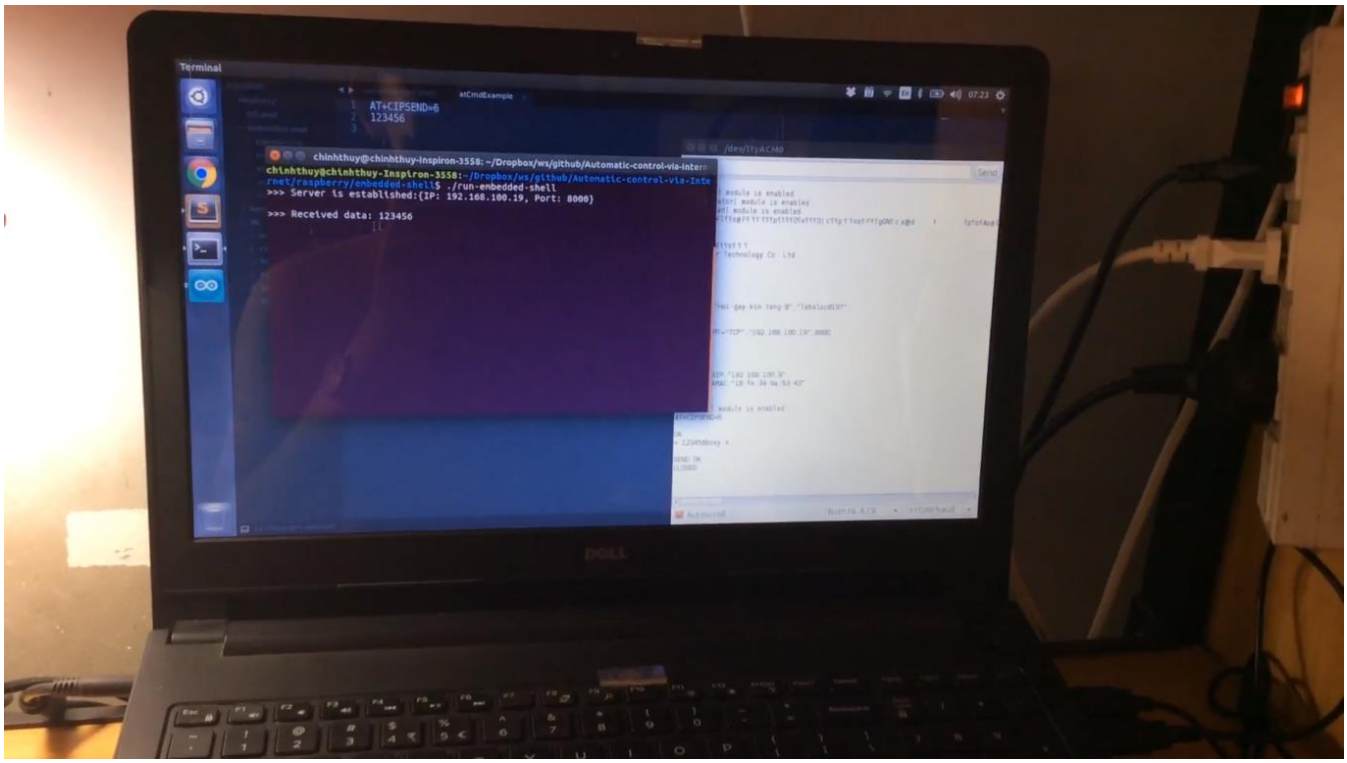


Figure VIII.1-1. [TCP connection exam] The sent string "123456"

is received correctly by the server

VIII.2. OS Shell

VIII.2.1. Os-shell & Platform

Tool	2 Linux terminals, one for PC-admin, the other for Raspberry-server
Network	LAN
Target	PC in role of admin send its username and password to server, the server verifies that information and responses login's result to admin
Process	[1] Run program [2] See server's terminal and check if os-shell & embedded-shell are set up (platform) and whose 2 sockets are also established.
Result	Os-shell and embedded-shell are set up. 2 sockets are set up on the same IP and 2 ports: 8000 & 9000.

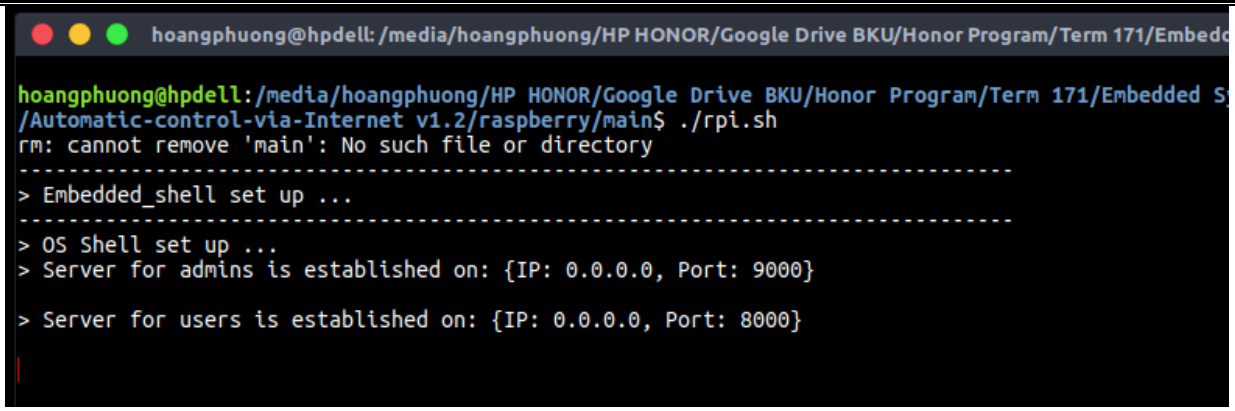


Figure VIII.2-1. Platform and 2 sockets

VIII.2.2. Login

Tool	2 Linux terminals, one for PC-admin, the other for Raspberry-server
Network	LAN
Target	PC in role of admin send its username and password to server, the server verifies that information and responses login's result to admin
Process	[3] Make full use of built platform (os-shell). [4] Create some samples of account files on server's kernel. [5] Use admin's terminal, connect to server, input login information, repeat this task for other cases of wrong information. [6] Display information of processing on server's terminal. [7] See response on admin's terminal.
Result	Admin gain connection successfully Admin receive right login's results.

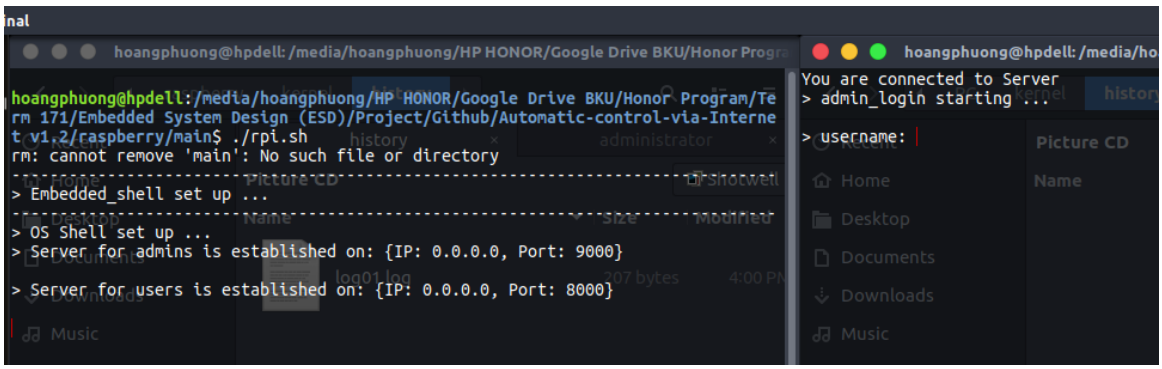


Figure VIII.2-2. Connection succeeded

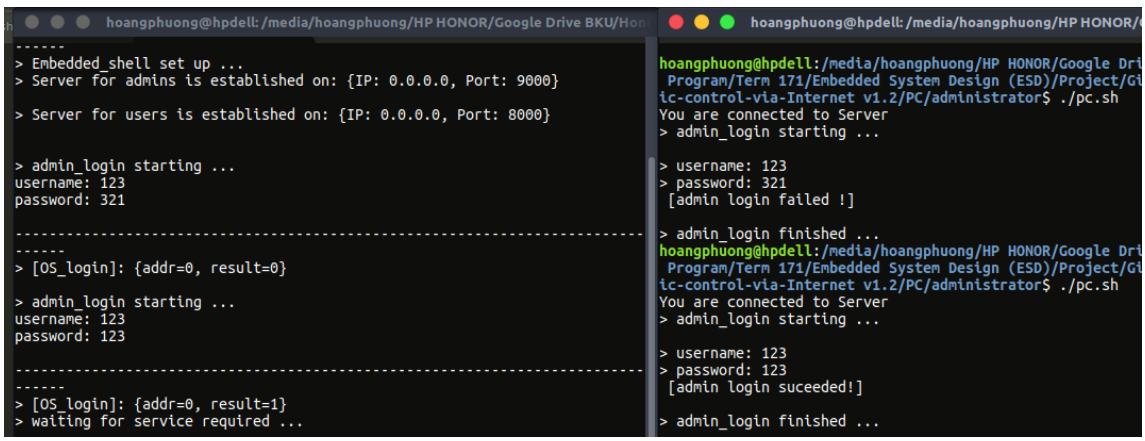


Figure VIII.2-3. Right information {123-123}. The first login failed for wrong information.

VIII.2.3. Backup

Tool	2 Linux terminals on the same PC, one for PC-admin, the other for Raspberry-server
Network	LAN
Target	Embedded device's accounts will be copied to PC-admin
Process	[1] Make full use of built platform (os-shell). [2] Create some samples of account files on server's kernel. [3] Run backup service on PC-admin and on Raspberry-server. [4] Display information of processing on both terminals. [5] Check if the copied database on PC exists.
Result	Admin gain connection successfully Admin receive the database completely.

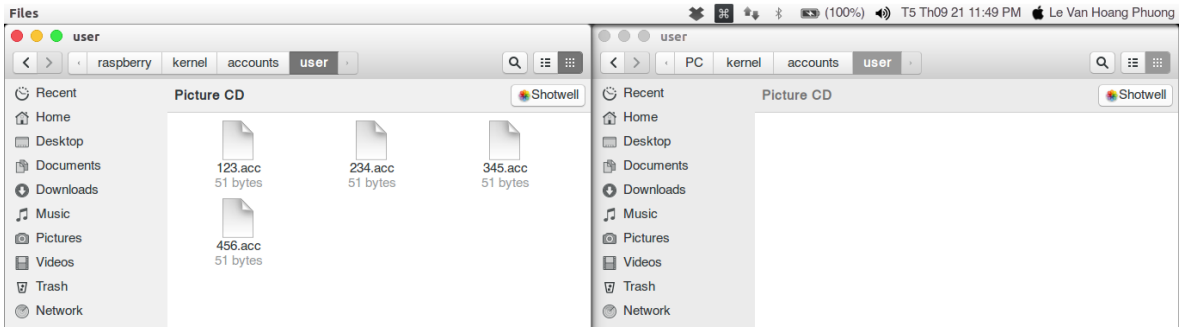


Figure VIII.2-4. Initial state

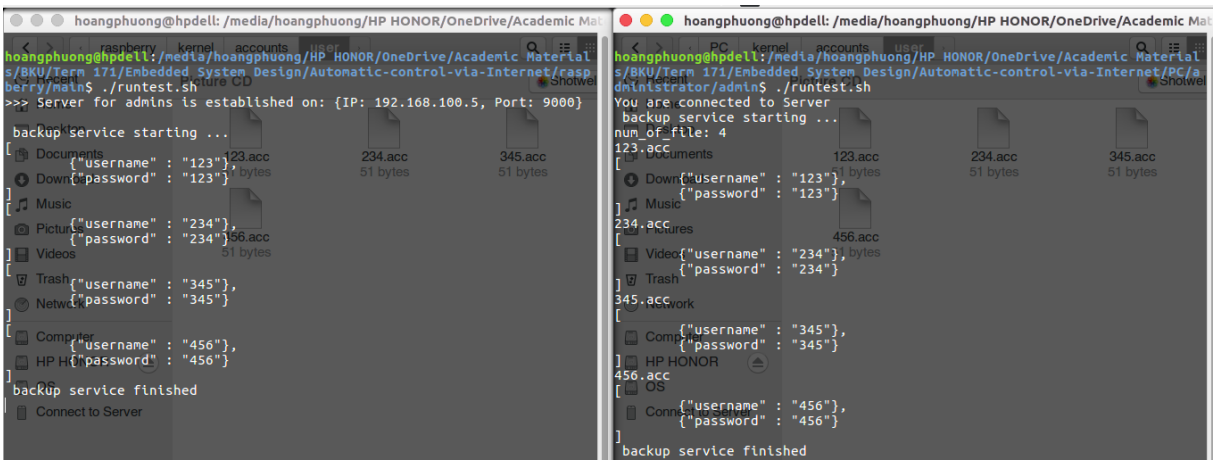


Figure VIII.2-5. Backup process

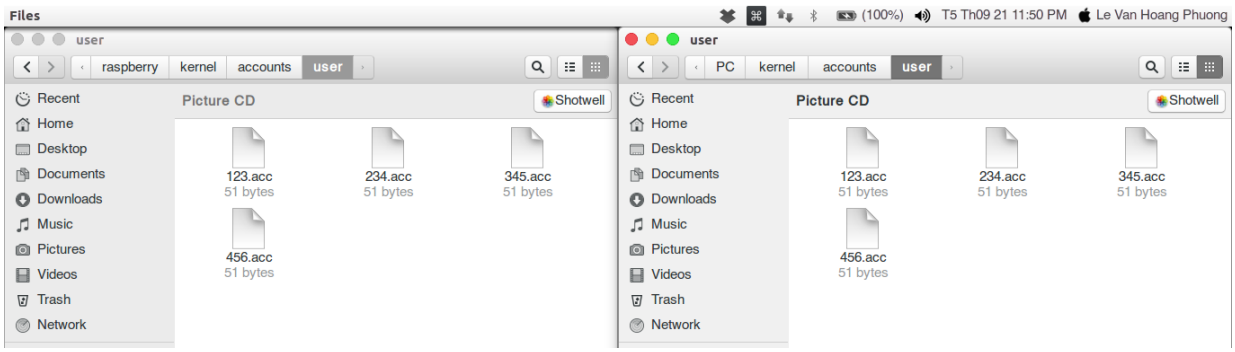


Figure VIII.2-6. After backup succeeded

VIII.2.4. Account Management

Tool	2 Linux terminals on the same PC, one for PC-admin, the other for Raspberry-server.
Network	LAN.
Target	Admin can modify the database on server.
Process	[1] Make full use of built platform (os-shell). [2] Create some samples of account files on server's kernel. [3] Run account management service on PC-admin and on Raspberry-server. [4] Display information of processing on both terminals. [5] Admin modifies database on PC's kernel. [6] Check if the database on server is up to date.
Result	Admin gain connection successfully. Admin receive the database completely. Server's database is updated.

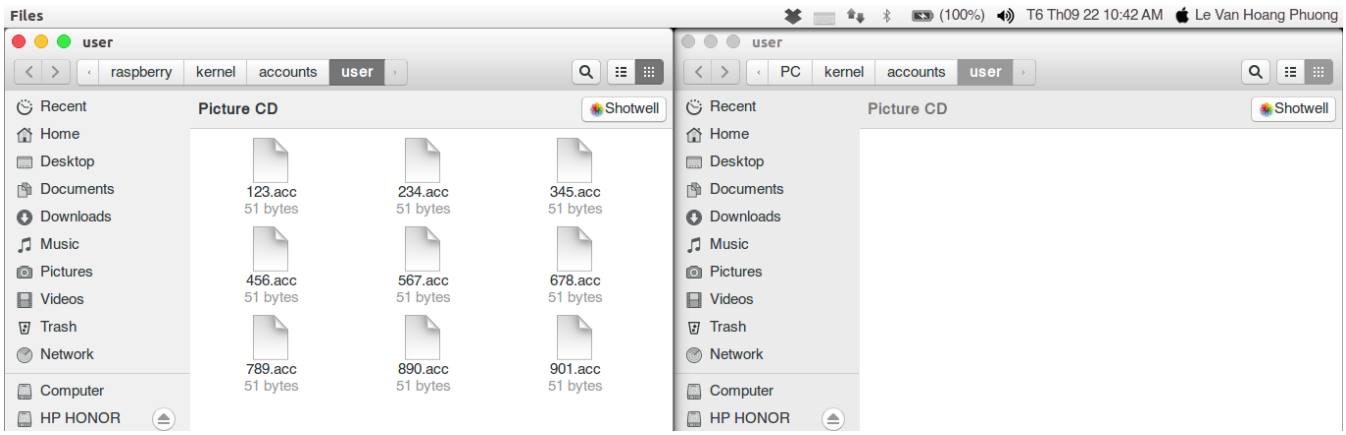


Figure VIII.2-7. Initial state of database of raspberry and PC

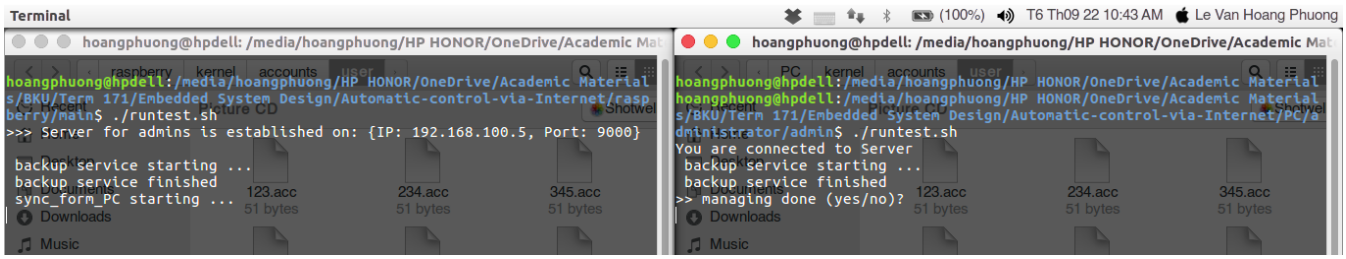


Figure VIII.2-8. Run service- phase 1: copied database of server to PC (backup)

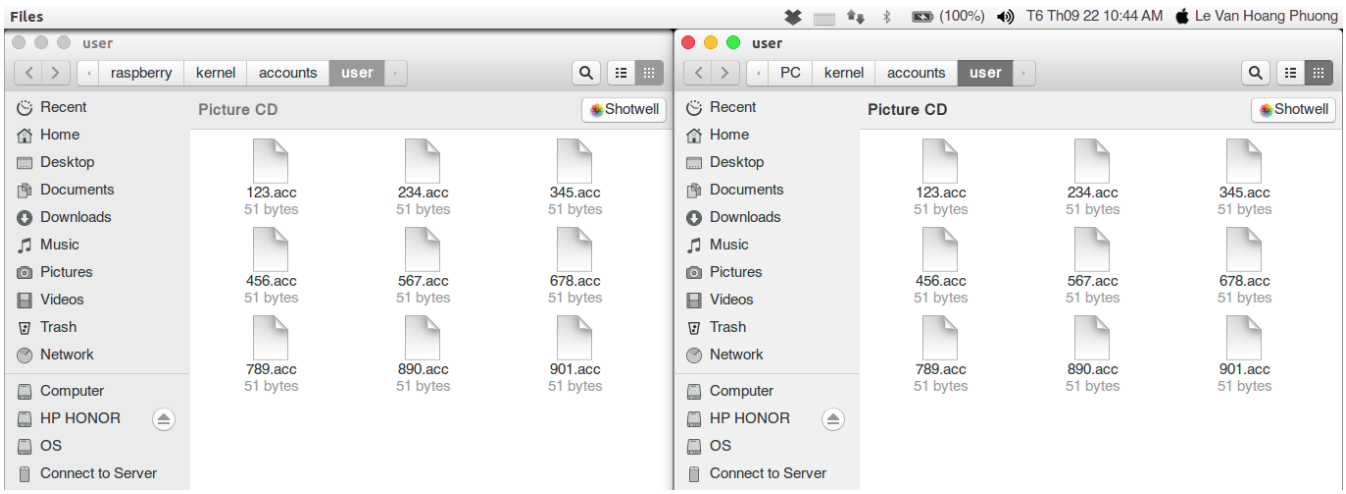


Figure VIII.2-9. Phase 1: copying database to PC done

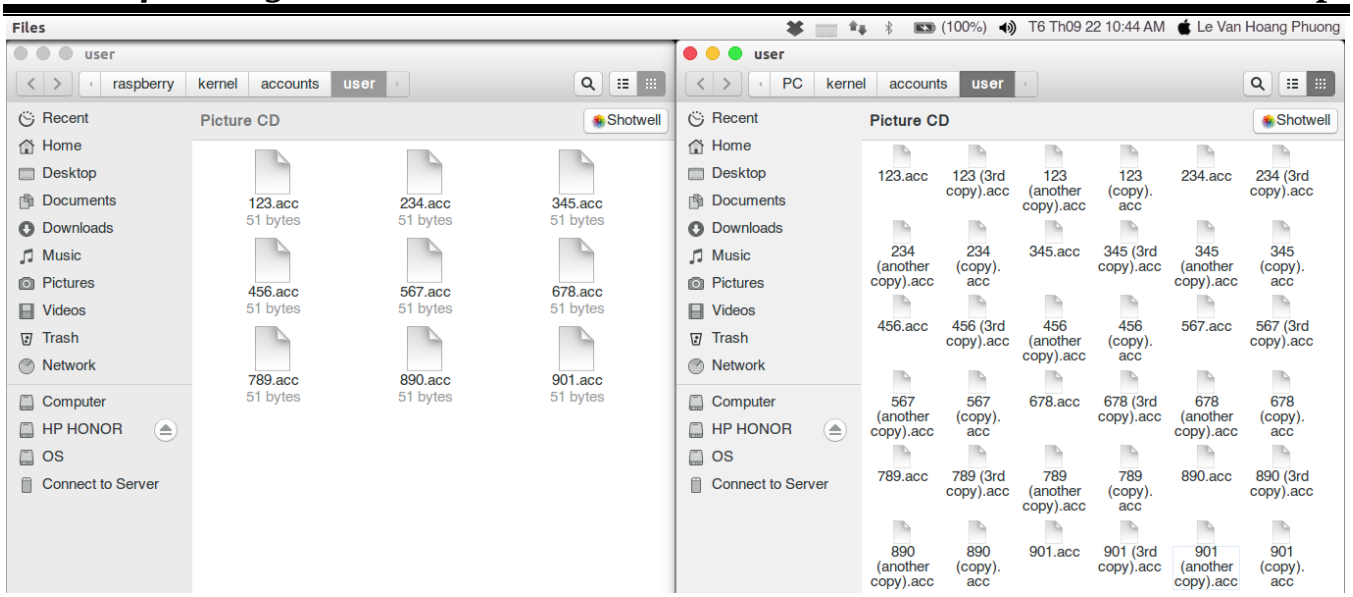


Figure VIII.2-10. Phase 2: modifying database on PC

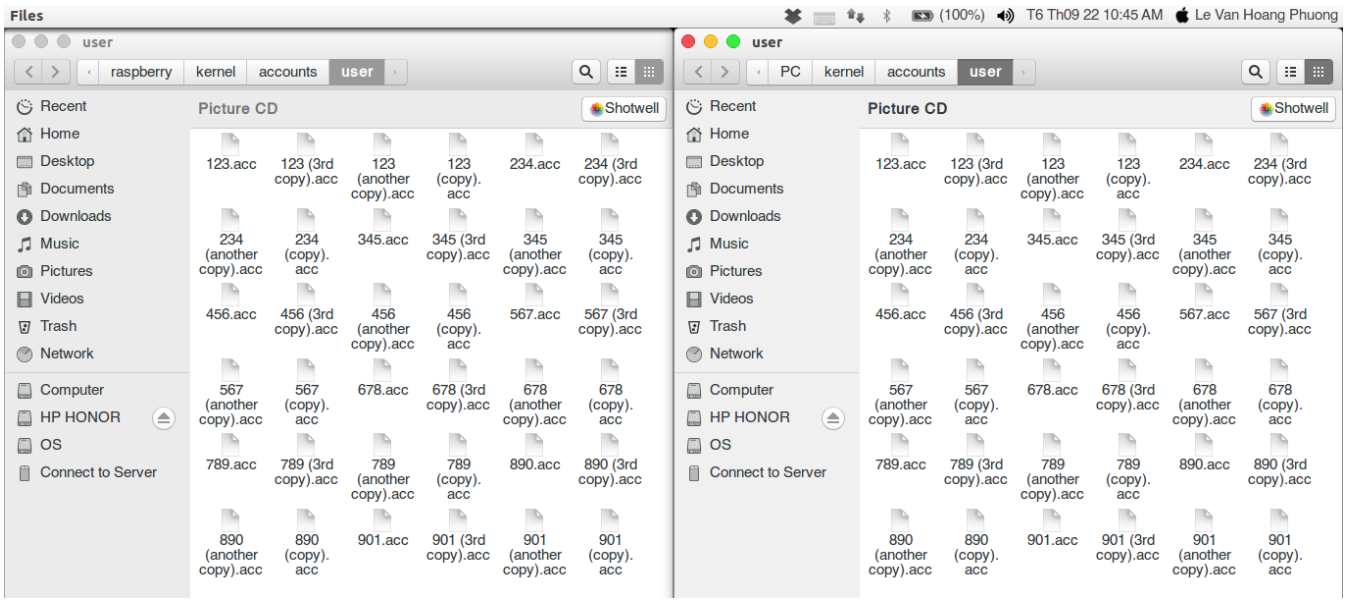


Figure VIII.2-11. Phase 2: syncing to server

VIII.2.5. Configuration

Tool	2 Linux terminals on the same PC, one for PC-admin, the other for Raspberry-server.
Network	LAN.
Target	Admin can turn off server remotely.
Process	[1] Make full use of built platform (os-shell). [2] Run configuration service on PC-admin. Simulating the server is running any tasks. [3] Display information of processing on both terminals. [4] Check server's terminal if the program is exited.
Result	Admin gain connection successfully. Server is stopped by admin.

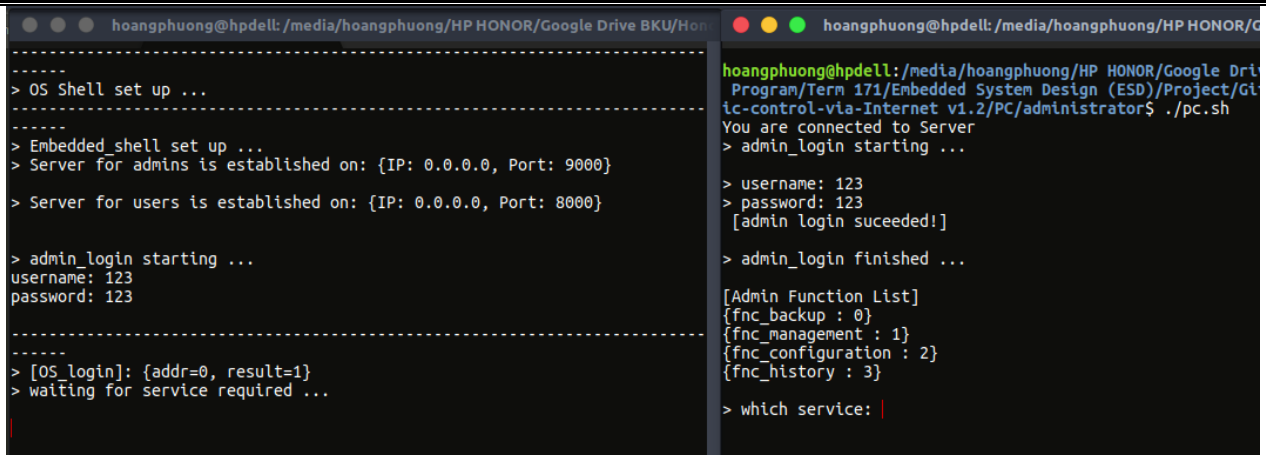


Figure VIII.2-12. Server is executed tasks, admin logged in

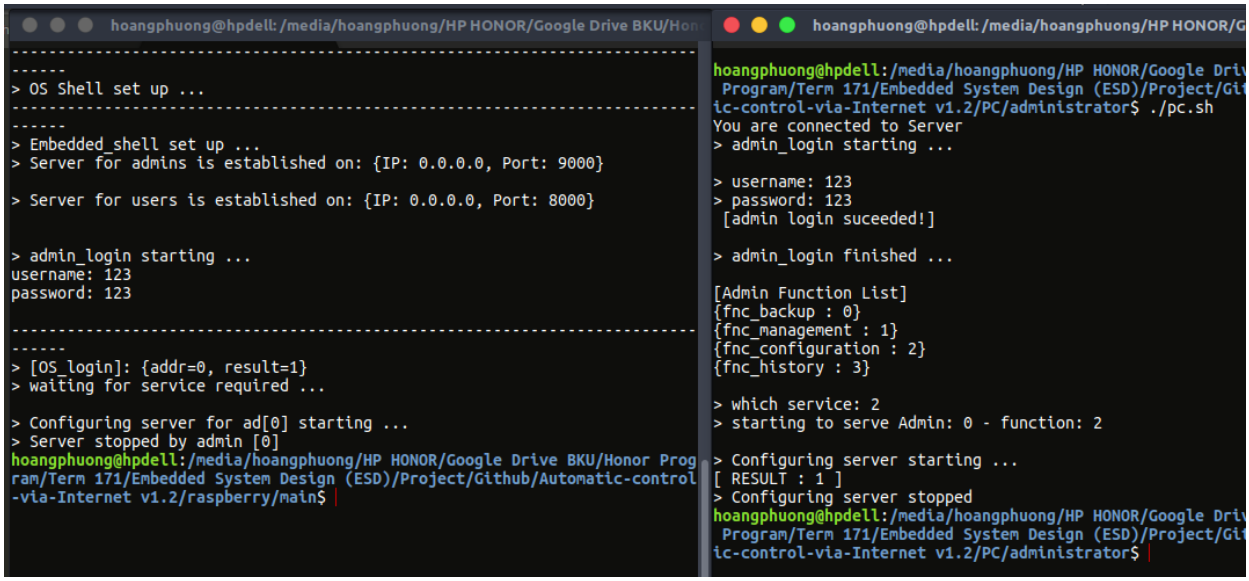


Figure VIII.2-13. Admin called configuration service & result

VIII.2.6. History Viewer

Tool	2 Linux terminals on the same PC, one for PC-admin, the other for Raspberry-server.
Network	LAN.
Target	Admin can see the history of login tasks of embedded devices.
Process	[1] Make full use of built platform (os-shell). [2] Create 2 samples of log file on server's kernel. [3] Admin run history viewer service. [4] Check if those log files are existed in kernel and match server's ones in term of content.
Result	Admin gain connection successfully. Receive log files successfully and their contents are reserved.

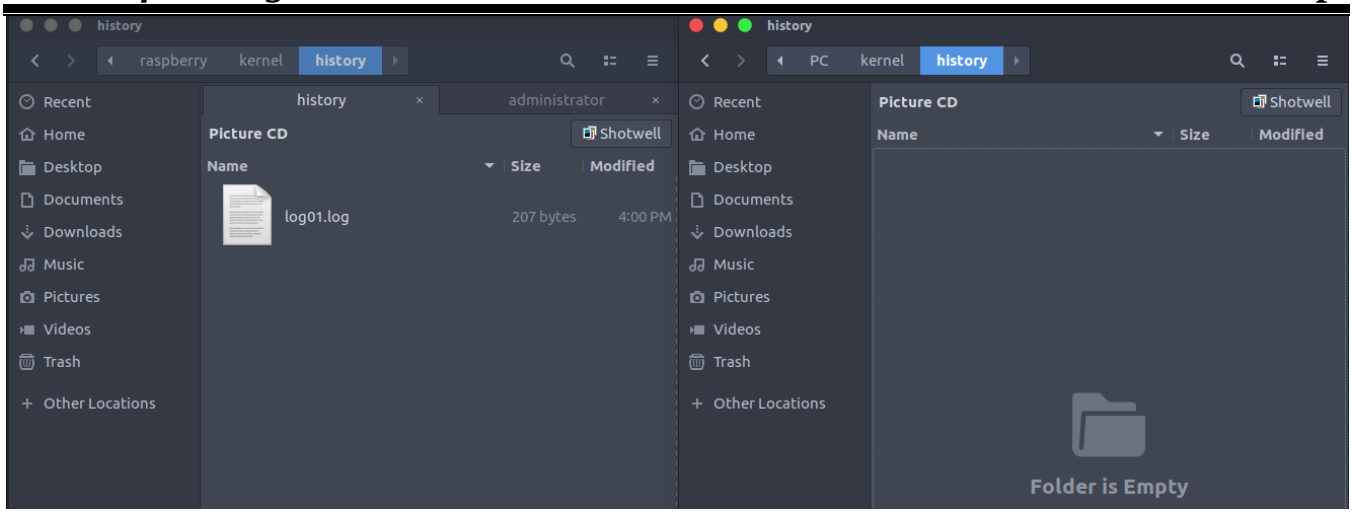


Figure VIII.2-14. Initial state of the server and admin's kernel

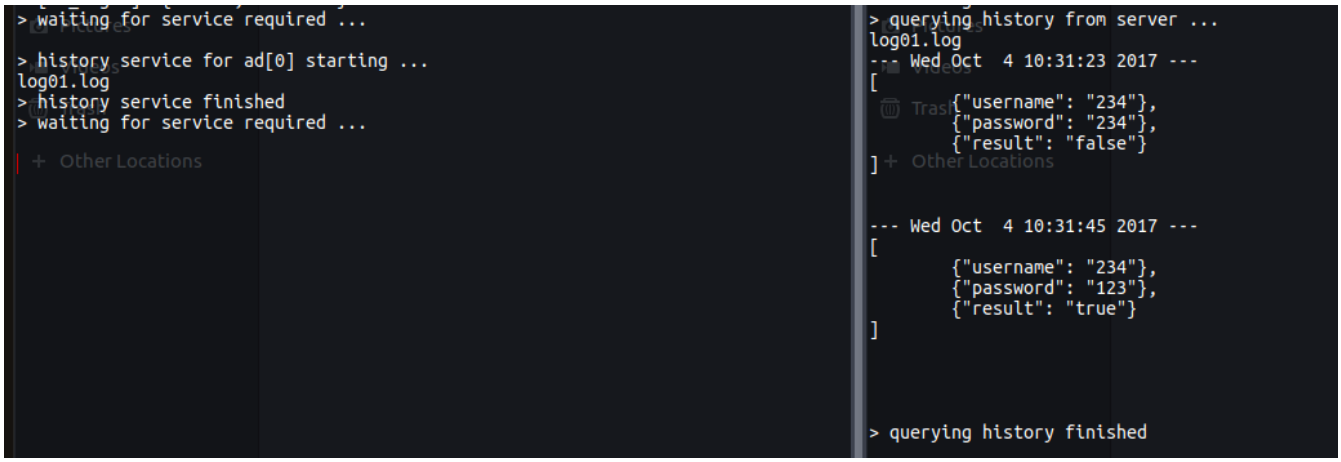


Figure VIII.2-15. Querying history from server

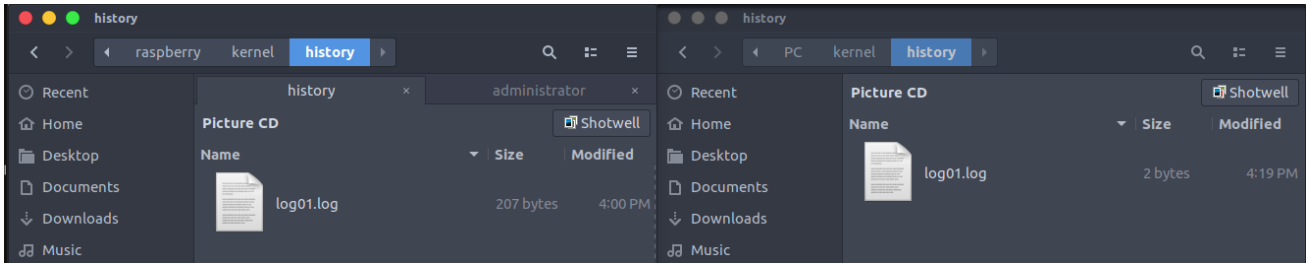


Figure VIII.2-16. Querying history done

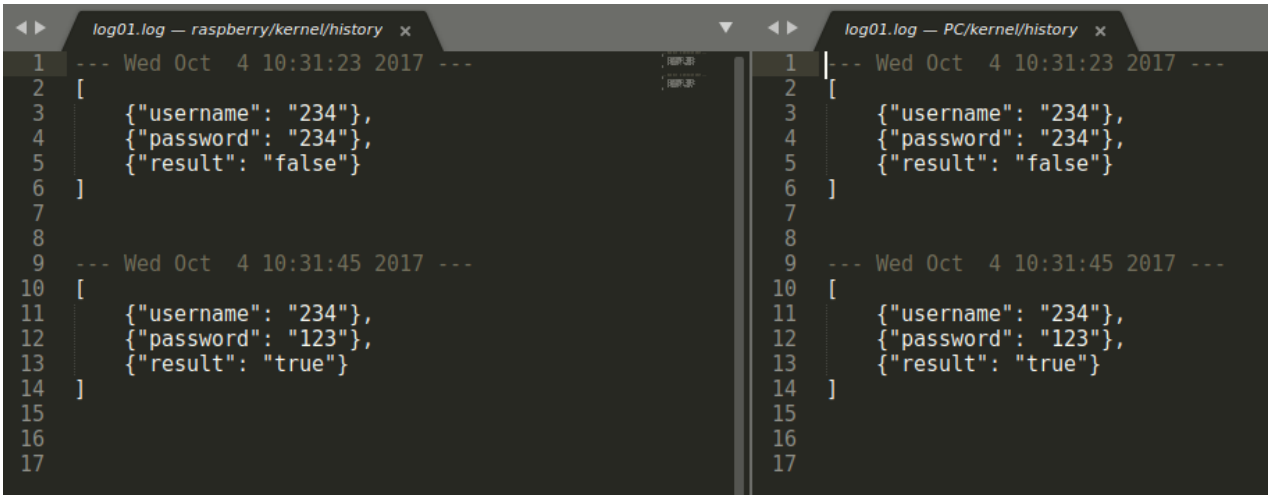


Figure VIII.2-17. Both files' content match

VIII.3. Embedded Shell

Tool	Arduino UART terminal, Linux terminal
Condition	Strong Wi-Fi, Local network
Target	ESP8266 connects to the server (auto), send login data, receive verification result
Process	Run the Embedded shell by Linux terminal. Run firmware on the prototype. Use Arduino terminal to replace LCD – Keypad. Type login to the Arduino terminal.
Result	Connection is established. Data transaction is done correctly (see Figure VIII.3-1).

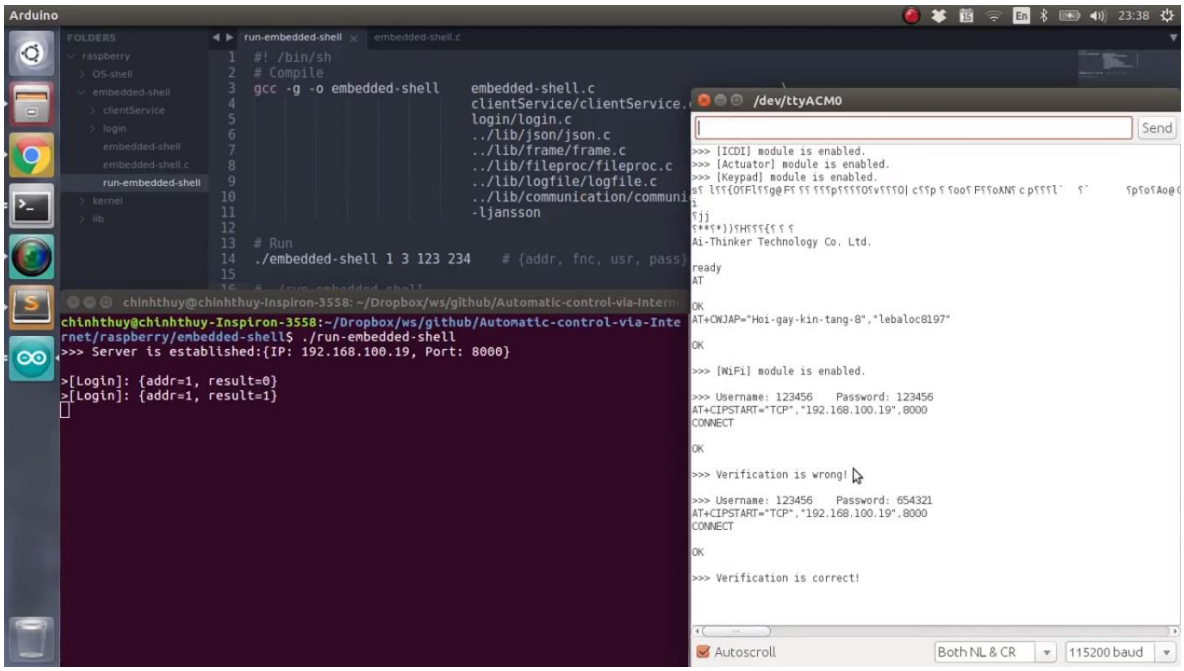
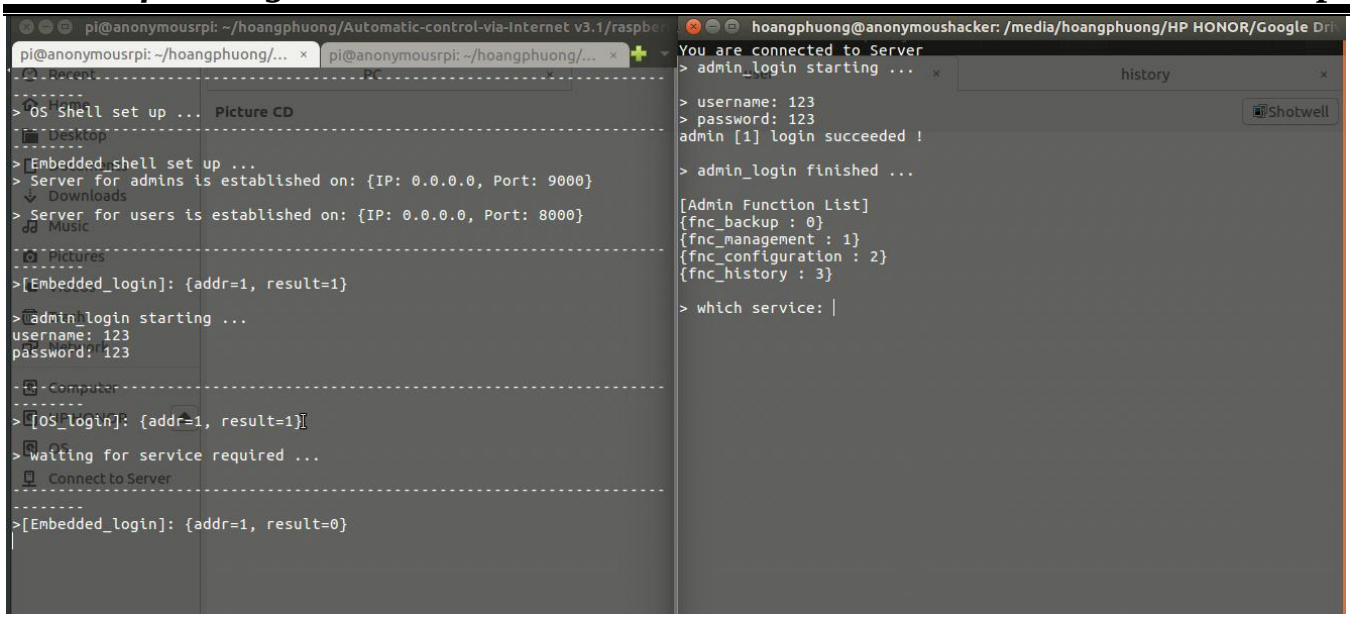


Figure VIII.3-1. [Data transaction exam] The right account is "123456" – "654321". At the first time, the login "123456" – "123456" is wrong. Afterward, the true login is recognized

VIII.4. The Whole of System

Tool	2 Linux terminals, one for PC-admin, the other for Raspberry-server, "ssh" tool for remote control server and display on a terminal.
Network	Internet
Target	PC and Embedded device (Tiva) connect to server simultaneously, server should serve both at the same time
Process	[1] Run server's program [2] See server's terminal and check if os-shell & embedded-shell are set up (platform) and whose 2 sockets are also established. [3] See operating processes on server's terminal
Result	Os-shell and embedded-shell are set up. 2 sockets are set up on the same IP and 2 ports: 8000 & 9000. 2 kinds of device connected at the same time and operating properly.


pi@anonymousrpi: ~/hoangphuong/Automatic-control-via-Internet v3.1/raspber
pi@anonymousrpi: ~/hoangphuong/...
pi@anonymousrpi: ~/hoangphuong/...
> OS Shell set up ...
> Embedded shell set up ...
> Server for admins is established on: {IP: 0.0.0.0, Port: 9000}
> Server for users is established on: {IP: 0.0.0.0, Port: 8000}
>[Embedded_login]: {addr=1, result=1}
>admin_login starting ...
username: 123
password: 123
> [OS_login]: {addr=1, result=1}
> waiting for service required ...
>[Embedded_login]: {addr=1, result=0}
hoangphuong@anonymoushacker: /media/hoangphuong/HP HONOR/Google Dri
You are connected to Server
> admin_login starting ...
> username: 123
> password: 123
admin [1] login succeeded !
> admin_login finished ...
[Admin Function List]
{fnc_backup : 0}
{fnc_management : 1}
{fnc_configuration : 2}
{fnc_history : 3}
> which service:
history
Shotwell

IX. Conclusion

IX.1. Achievement

In this project, we have built successfully a system that supports people in building-access management. With a prototype of self-implemented single-layer PCB and a Raspberry board, we respond almost planning requirements, e.g., input login data, online verification, notification, history viewer, account management. Besides, we also survey a market analysis to study how a security system that people need. Furthermore, at the beginning, we design a clear plan, and, during conducting the project, we monitor ourselves tightly by using Trello software. As the result, our working has completed on time.

Nevertheless, there are a few shortcomings that we are stuck. First, we carry out the system in a LAN network, whilst, our target is Internet scale. Second, the actuator in our prototype is not as considered. In our desire, it would be a mechanical motor structure, which can control the door open/close. However, because of time lack, and cost, we cannot bring this hope to this current prototype. Besides, when the admin connects to the server, it requires that the admin must know how to use Linux terminal, which is fairly inconvenient. Finally, the whole of the server data is not encrypted so that hacker can access and steal data.

IX.2. Future Work

Basing on drawbacks of this current prototype, we plan a coming work that fixes those weaknesses. In the future, we will use a VPN client to expand the scope from LAN to WAN. Subsequently, a new sub-team will be created to develop the mechanicals field to realize the actuator. Additionally, a GUI will be also formed to help the admin easily in interacting with the server. Especially, we are in the progress of designing an encryption method with interval-changing key to protect our data. Last but not least, thank to general definitions in specifications, we can develop a number of versions in order to adapt to a large range of using purpose types.

X. Reference

- Programming language
  [1] Kernighan, Brian W.; Ritchie, Dennis M. (February 1978). The C Programming Language (1st ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110163-3.
  [2] Beginning Linux® Programming, 4th Edition. Published by. Wiley Publishing, Inc. 10475 Crosspoint Boulevard. Indianapolis, IN 46256.
- IDE and editor
  [1] Code Composer Studio: http://processors.wiki.ti.com/index.php/Download_CCS
  [2] Tivaware: http://www.ti.com/tool/SW-TM4C
  [3] Sublime Text: https://www.sublimetext.com
  [4] GCC integrated on Linux's distros.

XI. Appendix A

XI.1. Hardware Design

The Software used to design PCB circuit is Alitum 16.0. This software has a user-friendly interface for designing, managing and editing, easy compilation, file management, version management for design documents. With a widely shared library source, it is easy to choose the components for the printed circuit.

XI.2. Firmware & Software Design

To develop Firmware, we use software Code Composer Studio of Texas Instrument to program the microcontroller. Code Composer Studio is a software built based on Eclipse, a familiar IDE, so it inherits a powerful debugger from Eclipse, which helps us extremely to overcome uncomfortable bugs. This software can be download in the official website of Texas Instrument. Besides, to program the TM4C-129 family microcontroller easily, we use the supporting library TivaWare._Reference_Reference

Whilst, we just use a trial version of Sublime Text editor for developing Software. This editor is combined with available Linux terminal to run our code._Reference

XII. Appendix B

XII.1. Code Organization

We structured our source code in form of hierarchy of system's partitions. It is a bit of complicated from ***root*** to ***leaf folder***.

Root folder (home folder)	**Automatic-control-via-Internet v1.2**
3 main function's folders	**PC**
	raspberry
	tiva

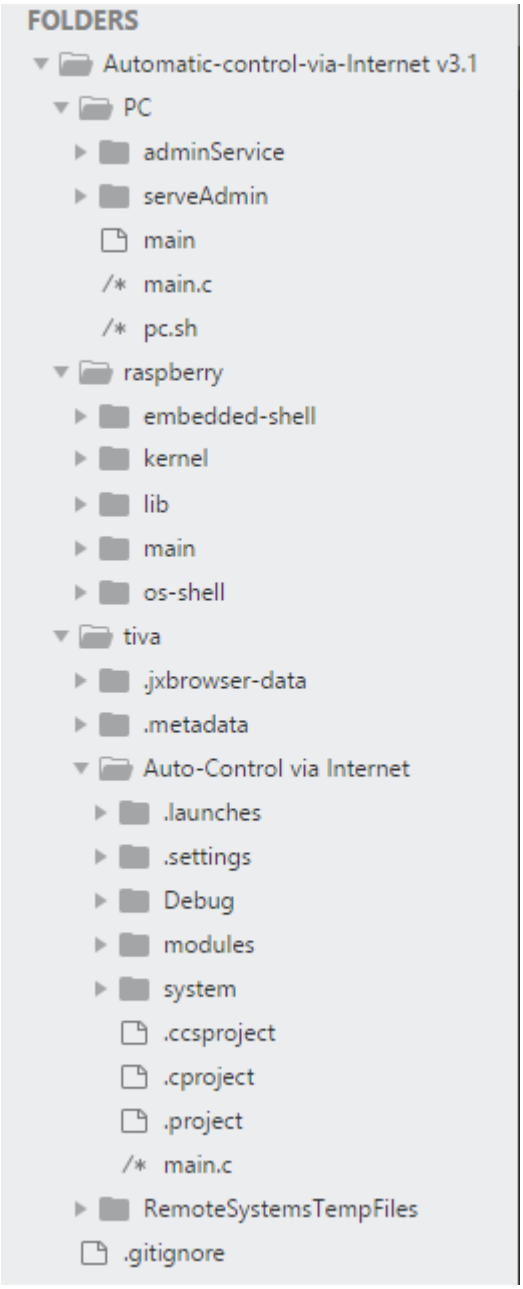


Figure XII.1-1. Code organization

XII.2. Repository

For further details, see:

Below are 3 main functions in our project:

[1] Raspberry-main.c

```
/************************************
* Date created:     6-8-2017
* Date finished:
* Editor: Sublime Text 3
* Compiler: gcc
* Author: Le Van Hoang Phuong
* Description: raspberry main program
************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>     //boolean
#include <dirent.h>      //folder proc
#include <pthread.h>     //multi-threads

/*common private lib*/
#include "../lib/fileproc/fileproc.h"
#include "../lib/communication/communication.h"
#include "../lib/frame/frame.h"
#include "../lib/json/json.h"
#include "../lib/platform/thread.h"
#include "../lib/bufferproc/bufferproc.h"

/*include platform for shells*/
#include "platform.h"

/*include shell*/
#include "../os-shell/os_shell.h"
#include "../embedded-shell/embedded_shell.h"

int main(int argc, char const *argv[])
{
     /*platform thread: control os_shell_thread &
embedded_shell_thread*/
     void *    thread_exit;   //message when thread cancelled
     pthread_t platform_thread;    //declare thread name
     /*run platform in parallel with main*/
     /*this platform will run os_shell & embedded_shell*/
```

```
     if(!createThread(&platform_thread, platform, thread_exit))
exit(1);
     /*force main to hang to wait for platform exited first*/
     pthread_join(platform_thread, &thread_exit);

     /*print the result as platform exited*/
     fprintf(stderr, "%s\n", (char *)thread_exit);

     /*remove main.o*/
     unlink("main");
     /*return*/
     return 1;
}
```

[2] PC- main.c

```
/*************************************
* Date created:     6-8-2017
* Date finished:
* Editor: Sublime Text 3
* Compiler: gcc
* Author: Le Van Hoang Phuong
* Description: PC main program
*************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <dirent.h>
#include <pthread.h>

/*user library*/
#include "../../lib/fileproc/fileproc.h"
#include "../../lib/communication/communication.h"
#include "./adminService/backup/backup.h"
#include "./adminService/adminService.h"
#include "adminService/management/management.h"
#include "./serveAdmin/serveAdmin.h"

/*main function*/
int main(int argc, char const *argv[])
{
     int connfd;
     /*create socket for admin*/
     connfd = createClientSocket(argv[1], atoi(argv[2]));

     /*run admin Service control*/
     /*define admin addr here: ad_0*/
     adminService(&connfd, ad_0);

     /*remove main.o*/
     unlink("main");
     /*return*/
     return 0;
}
```

[3] Tiva- main.c

```
/*
 *   Author      : Thuy Nguyen-Chinh.
 *   Date        : Sep 04, 2017
 *   Description : This is the main file of the project.
 *   Version     : 1.0.1.
 */
/*****************************************************************
 *  Include
 *****************************************************************/
/* Project */
#include "modules/pin_def.h"
#include "system/system.h"
#include "modules/ui/ui.h"


/*****************************************************************
 *  Main
 *****************************************************************/
void main()
{
    /* Setup */
    systemSetup();

    /* Serve */
    while(1)
    {
        /* Idle: Toggle LED */
        actIdle();
        clkDelayMs(10);

        /* Sweep keypad */
        kpSweep();
        flgBtnInt = kpCheck();

        /* User interface */
        uiServing();
    }
}
```