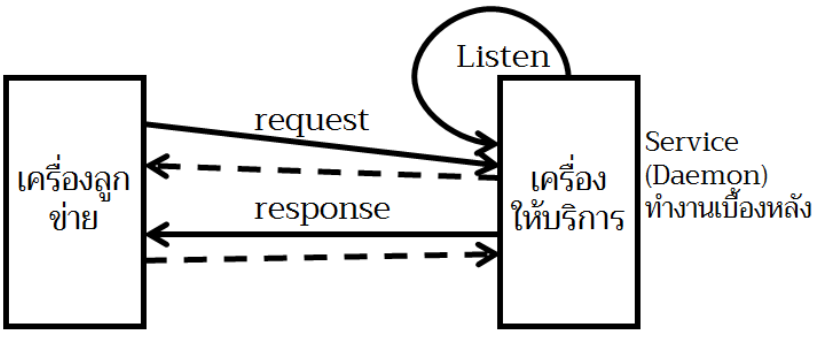
บทที่ 14
สื่อสารผ่านโพรโทคอลอินเทอร์เน็ต

ในบทนี้กล่าวถึงการเขียนโปรแกรมด้วยภาษาไพธอนเพื่อเชื่อมประสานกับระบบเครือข่ายที่ใช้งานโพรโทคอลอินเทอร์เน็ตทางซ็อกเก็ตเพื่อศึกษาแนวคิดเริ่มต้นของการเขียนโปรแกรมแบบเครื่องลูกข่าย-เครื่องบริการ หรือไคลเอนต์-เซิร์ฟเวอร์ (Client-Server) พร้อมทั้งทำความเข้าใจเกี่ยวกับการเขียนโปรแกรมเพื่อตั้งค่าการทำงาน การเขียนโปรแกรมเพื่ออ่านข้อมูลจากโพรโท-คอลแบบ HTTP พร้อมตัวอย่างการควบคุมอุปกรณ์ที่เชื่อมต่อกับบอร์ด Raspberry Pi จากเครื่องหรืออุปกณณ์อื่นๆ ที่รันสามารถประมวลผลชุดคำสั่งไพธอนฝั่งเครื่องลูกข่ายหรือไคลเอนต์ได้

เมื่อจบบทนี้ ผู้อ่านจะเข้าใจถึงขั้นตอนการเขียนโปรแกรม และวิธีการเขียนโปรแกรมรับข้อมูลนำเข้าข้อมูลในการสื่อสารกับเครือข่ายผ่านโพรโทคอลอินเทอร์เน็ต และประยุกต์ใช้การควบคุมอุปกรณ์ผ่านเครือข่ายด้วยการเขียนโปรแกรมแบบไคลเอนต์-เซิร์ฟเวอร์

1. แนะนำโพรโทคอลอินเทอร์เน็ต

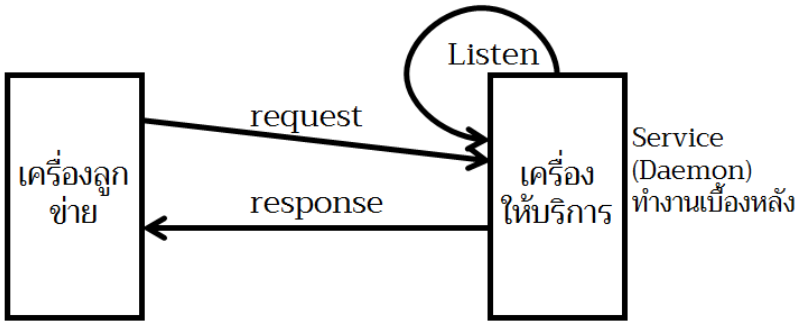
โพรโทคอลอินเทอร์เน็ต (Internet Protocol) เป็นรูปแบบของการสื่อสารที่ออกแบบเพื่อใช้กับเครือข่ายอินเทอร์เน็ต รองรับการทำงานทั้งแบบคอนเน็คชั่นออเรียนเต็ด (Connection-Oriented) ผ่านทางโพรโทคอลแบบทีซีพีหรือ TCP (Transmission Control Protocol) และแบบคอนเน็คชันเลส (Connectionless) ผ่านทางโพรโทคอลแบบยูดีพี หรือ UDP (User Datagram Protocol)



ภาพที่ 14-1-1 การสื่อสารแบบทีซีพี

ความแตกต่างระหว่างทีซีพีและยูดีพี คือ ทีซีพีรองรับการยืนยันการส่งข้อมูลจากผู้รับเพื่อป้องกันการสูญหายของข้อมูลที่อาจจะเกิดขึ้นได้ระหว่างการสื่อสาร แต่ยูดีพีเน้นการส่งโดยไม่ต้อง

มีการยืนยันการได้รับข้อมูลทำให้ข้อมูลอาจสูญหายได้ ดังภาพที่ 14-1-1 และ 14-1-2 ถึงอย่างไรข้อดีของยูดีพีอยู่ที่มีความเร็วของการสื่อสารที่มากกว่าแบบทีซีพี



ภาพที่ 14-1-2 การสื่อสารแบบยูดีพี

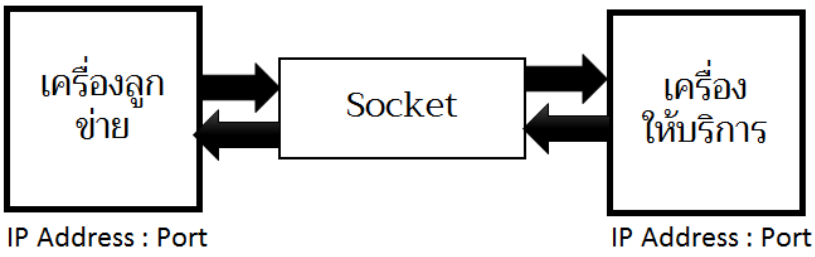
ภายใต้การสื่อสารโดยใช้โพรโทคอลอินเทอร์เน็ต อุปกรณ์หรือโฮสต์ (Host) แต่ละตัวในเครือข่ายจะต้องมีหมายเลขประจำตัวที่ไม่ซ้ำกัน เรียกว่า หมายเลขไอพี (IP Address) ซึ่งหมายเลขไอพีรุ่นที่ 4 ประกอบไปด้วยตัวเลขในช่วง 0 ถึง 255 จำนวน 4 ชุด โดยแต่ละชุดคั่นด้วย . และแต่ละแอพพลิเคชันจะสื่อสารผ่านทางหมายเลขพอร์ต (Port) เช่น โพรโทคอลประเภทเอชทีทีพี (HTTP) ใช้พอร์ตสื่อสารหมายเลข 80 และใช้การสื่อสารแบบทีซีพี โพรโทคอลประเภทเอสเอสเอช (SSH) ใช้พอร์ตสื่อสารหมายเลข 22 และใช้การสื่อสารแบบทีซีพี เป็นต้น ซึ่งหมายเลขพอร์ตสำหรับสื่อสารนั้นมีค่าได้ตั้งแต่ 0 ถึง 65535 ดังนั้น ทีซีพีมีหมายเลขพอร์ตตั้งแต่ 0 ถึง 65535 และยูดีพีมีหมายเลขพอร์ตตั้งแต่ 0 ถึง 65535

เนื่องจากอุปกรณ์แต่ละตัวจะต้องมีหมายเลขไอพีเป็นตัวอ้างอิงตัวตนในการสื่อสาร ทำให้ไม่สามารถตั้งค่าซ้ำกันได้ และเพื่อความสะดวกผู้ดูแลระบบเครือข่ายมักติดตั้งบริการดีเอชซีพี (DHCP) สำหรับแจกจ่ายหมายเลขไอพีให้กับอุปกรณ์แบบอัตโนมัติ และเพื่อความง่ายต่อการจดจำเครื่องจึงมีระบบดีเอ็นเอส (DNS) สำหรับการฟอร์เวิร์ด (forward) เพื่อแปลงชื่อเป็นหมายเลขไอพี และแบ็คเวิร์ด (backward) สำหรับการแปลงหมายเลขไอพีเป็นชื่อ เช่น มอง www.google.com มีหมายเลขไอพีเป็น 216.58.221.36 เป็นต้น

2. ซ็อกเก็ต

ซ็อกเก็ต (Socket) เป็นช่องทางสื่อสารระหว่างอุปกรณ์ที่ทำงานด้วยโพรโทคอลอินเทอร์เน็ต ดังภาพที่ 14-2-1 ซึ่งในภาษาไพธอนจะต้องทำการเรียกไลบรารีซ็อกเก็ตดังนี้

```
import socket
```



ภาพที่ 14-2-1 ซ็อกเก็ต

คำสั่งสำหรับอ่านชื่อของโฮสมีรูปแบบดังนี้

```
ตัวแปรสตริง = socket.gethostname()
```

คำสั่งสำหรับอ่านหมายเลขไอพีของเครื่องที่กำลังรันโปรแกรมมีรูปแบบดังนี้

```
ตัวแปรสตริง = socket.gethostbyname( ชื่ออุปกรณ์ )
```

ตัวอย่างการเขียนโปรแกรมเพื่อแสดงข้อมูลชื่อโฮสและหมายเลขไอพีของ Raspberry Pi เป็นดังตัวอย่างโปรแกรมที่ 14-2-1 และตัวอย่างการเขียนโปรแกรมเพื่อสอบถามหมายเลขตำแหน่งไอพีจากชื่อโฮส เป็นดังตัวอย่างโปรแกรมที่ 14-2-2

โปรแกรมที่ 14-2-1 อ่านชื่อโฮสและตำแหน่งไอพีของ Raspberry Pi

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	**def show_info():**
4	host = socket.gethostname()
5	ip = socket.gethostbyname(host)
6	print("Host name : ", host)
7	print("IP address : ", ip)
8	print("Network Programming (14-2-1)")
9	show_info()

ผลลัพธ์จากการทำงานของโปรแกรมที่ 14-2-1 เป็นดังนี้

```
Network Programming (14-2-1)
Host name :  raspberrypi
IP address :  127.0.1.1
```

ผลลัพธ์จากการทำงานของโปรแกรมที่ 14-2-2 เป็นดังนี้

```
Network Programming (14-2-2)
Host name ? www.etteam.com        (พิมพ์ชื่อเวบไซต์)
www.etteam.com  is  166.62.96.79
```

โปรแกรมที่ 14-2-2 สอบถามตำแหน่งไอพีจากชื่อโฮส *(ต้องต่อ Internet ทางช่อง Lan ของ Rpi)*

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	**def show_remote_info():**
4	host = input("Host name ? ")
5	try:
6	ip = socket.gethostbyname(host)
7	except socket.error as err_msg:
8	print(host, err_msg)
9	print(host, " is ", ip)
10	print("Network Programming (14-2-2)")
11	show_remote_info()

ตัวอย่างการเขียนโปรแกรมเพื่ออ่านวันที่และเวลาปัจจุบันจากบริการ NTP เป็นดังตัวอย่างโปรแกรมที่ 14-2-3 ซึ่งจะต้องติดตั้งไลบรารี ntplib ดังคำสั่งต่อไปนี้ก่อนรันโปรแกรม

```
sudo pip3  install ntplib
```

โปรแกรมที่ 14-2-3 อ่านวันที่และเวลาปัจจุบันจาก pool.ntp.org *(ต่อ Internet ทางช่อง Lan ของ Rpi)*

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import ntplib
3	from time import ctime
4	print("Network Programming (14-2-3)")
5	ntp_client = ntplib.NTPClient()
6	response = ntp_client.request('pool.ntp.org')
7	print(ctime(response.tx_time))

ผลลัพธ์จากการทำงานของโปรแกรมที่ 14-2-3 เป็นดังนี้

```
Network Programming (14-2-3)
Fri Jan 20 21:24:55 2017
```

2.1 ชุดคำสั่งไลบรารี socket

ในหัวข้อนี้กล่าวถึงชุดคำสั่งไลบรารีซ็อกเก็ตสำหรับใช้ในการสื่อสารภายใต้โพรโทคอลอินเทอร์เน็ต

2.1.1 สร้างซ็อกเก็ต

วิธีการขอเปิดใช้ซ็อกเก็ตสำหรับสื่อสารในเครือข่ายอินเทอร์เน็ตจะต้องระบุค่า 2 สิ่ง คือ หมายเลขไอพีของอุปกรณ์ที่ต้องการเข้าถึง และหมายพอร์ตที่ต้องการสื่อสาร โดยรูปแบบสำหรับสร้างซ็อกเก็ตสื่อสารเป็นดังนี้

ตัวแปรซ็อกเก็ต = socket.**socket**(domain, type)

โดยที่ domain ต้องมีค่าเป็น socket.AF_INET เพื่อระบุว่าเป็นสื่อสารภายใต้โพรโทคอลอินเทอร์เน็ต และมีค่าสำหรับเครือข่ายชนิดอื่นๆ เช่น PF_INET, PF_UNIX และ PF_X25 เป็นต้น

type คือ วิธีการสื่อสารซึ่งมี 2 ประเภทคือ

socket.SOCK_STREAM สำหรับทีซีพี

socket.SOCK_DGRAM สำหรับยูดีพี

ตัวอย่างการเขียนโปรแกรมเพื่อสอบถามพอร์ตที่ระบุในตัวแปร Ports ของโฮสที่ระบุ ดังตัวอย่างโปรแกรมที่ 14-2.1.1-1 *(ต่อ Internet ทางช่อง Lan ของ Rpi)*

โปรแกรมที่ 14-2.1.1-1 สอบถามพอร์ตที่ระบุในตัวแปร Ports ของโฮสที่ระบุ

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	import sys
4	Ports = [80]
5	**def list_tcp_port():**
6	host = input('Host name ? ')
7	try:
8	ip = socket.gethostbyname(host)
9	except socket.error as err_msg:
10	print(host, err_msg)
11	sys.exit()
12	print("Scan IP Address "+ip)
13	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14	for port in Ports:
15	client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16	addr = (host, port)
17	result = client.connect_ex(addr)
18	if result == 0:
19	serv = socket.getservbyport(port, 'tcp')
20	print("Port ",port,"(",serv,") is open")
21	client.close()
22	print("Network Programming (14-2.1.1-1)")
23	list_tcp_port()

ผลลัพธ์จากการทำงานของโปรแกรมที่ 14-2.1.1-1 เป็นดังนี้

```
Network Programming (14-2.1.1-1)
Host name ? www.etteam.com (พิมพ์ชื่อ host)
Scan IP Address 166.62.96.79
Port  80 ( http ) is open
```

2.1.2 การปิดซ็อกเก็ต

การปิดซ็อกเก็ตสื่อสารที่สร้างไว้ใช้มีรูปแบบดังต่อไปนี้

ตัวแปรซ็อกเก็ต.close()

2.1.3 ค่าไทม์เอาต์

การอ่านค่าไทม์เอาต์ (Time-out) หรือระยะเวลามากที่สุดที่ถือว่าการสื่อสารมีปัญหา มีรูปแบบดังนี้

ชื่อตัวแปรซ็อกเก็ต.**gettimeout**()

รูปแบบคำสั่งสำหรับตั้งค่าระยะเวลาไทม์เอาต์เป็นดังนี้

ชื่อตัวแปรซ็อกเก็ต.**settimeout**(ระยะเวลา)

ตัวอย่างการเขียนโปรแกรมเพื่ออ่านและตั้งค่าไทม์เอาต์ ในการสื่อสารผ่านเครือข่ายอินเทอร์เน็ต เป็นดังตัวอย่างโปรแกรมที่ 14-2.1.3-1

โปรแกรมที่ 14-2.1.3-1 อ่านและตั้งค่าไทม์เอาต์

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	print("Network Programming (14-2.1.3-1)")
4	s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5	print("Default socket timeout: ", s.gettimeout())
6	s.settimeout(100)
7	print("Current socket timeout: ", s.gettimeout())

ผลลัพธ์จากการทำงานของโปรแกรมที่ 14-2.1.3-1 เป็นดังนี้

```
Network Programming (14-2.1.3-1)
Default socket timeout:  None
Current socket timeout:  100.0
```

2.1.4 การเชื่อมต่อไปยังเครื่องให้บริการ

การเชื่อมต่อไปยังเครื่องให้บริการเป็นคำสั่งสำหรับโปรแกรมฝั่งเครื่องลูกข่ายที่ใช้ระบุการเชื่อมต่อไปยังเครื่องให้บริการเพื่อร้องขอการทำงานหรือสื่อสารกับเครื่องให้บริการ โดยรูปแบบของการเชื่อมต่อเป็นดังนี้

ตัวแปรซ็อกเก็ต.**connect**((โฮส, พอร์ต))

หรือ

ผลการเชื่อมต่อ = ตัวแปรซ็อกเก็ต.connect_to((โฮส, พอร์ต))

2.1.5 การส่งข้อมูล

การส่งข้อมูลจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งมีรูปแบบการใช้งานดังนี้

ผลลัพธ์ = ตัวแปรซ็อกเก็ต.send(ข้อมูลแบบไบต์)

หรือ

ผลลัพธ์ = ตัวแปรซ็อกเก็ต.sendall(ข้อมูลแบบไบต์)

2.1.6 คำสั่งรับข้อมูล

คำสั่งรับข้อมูลเป็นคำสั่งสำหรับนำข้อมูลจากการสื่อสารมาเก็บในตัวแปรแบบไบต์ โดยรูปแบบของคำสั่งเป็นดังนี้

ตัวแปรไบต์ = ตัวแปรซ็อกเก็ต.recv(ขนาดข้อมูล)

ตัวอย่างโปรแกรม 14-2.1.6-1 เป็นโปรแกรมเพื่อร้องขอไฟล์หน้าหลักของเว็บที่ระบุ โดยอ่านมาสูงสุด 4096 ไบต์ และต้องทำการแปลงคำสั่งจากสตริงให้เป็นไบต์เพื่อส่งไปให้ฝั่งเครื่องให้บริการ หลังจากรับข้อมูลจึงต้องแปลงข้อมูลไบต์ที่รับเข้ามาให้เป็นสตริงเพื่อแสดงผลที่จอภาพ

โปรแกรมที่ 14-2.1.6-1 ร้องขอไฟล์หน้าหลักของเว็บ *(ต่อ Internet ทางช่อง Lan ของ Rpi)*

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	import sys
4	Ports = [80]
5	**def http_get():**
6	host = input('Host name ? ')
7	try:
8	ip = socket.gethostbyname(host)
9	except socket.error as err_msg:
10	print(host, err_msg)
11	sys.exit()
12	print("Scan IP Address "+ip)
13	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14	for port in Ports:
15	client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16	addr = (host, port)
17	result = client.connect_ex(addr)
18	if result == 0:
19	cmd_str = "GET / HTTP/1.1\r\nHost: google.com\r\n\r\n"

```
20                              cmd_byte = cmd_str.encode(encoding='UTF-8')
21                              client.send( cmd_byte )
22                              response = client.recv(4096)
23                              resp_str = response.decode(encoding='UTF-8')
24                              print(resp_str)
25                      client.close()
26  print("Network Programming (14-2.1.6-1)")
27  http_get()
```

ผลลัพธ์จากการทำงานของโปรแกรมที่ 14-2.1.6-1 เป็นดังนี้

```
Network Programming (14-2.1.6-1)
Host name ? www.google.com   (พิมพ์ชื่อเวปไซต์)
Scan IP Address 172.217.24.228
HTTP/1.1 302 Found
Location: http://www.google.co.th/?gws_rd=cr&ei=zB6CWL6jA4uMvQT0ya6wBA
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See
https://www.google.com/support/accounts/answer/151657?hl=en for more info."
Date: Fri, 20 Jan 2017 14:29:32 GMT
Server: gws
Content-Length: 261
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie:
NID=95=Yueiu9pUiszTyTa2FdKlSAsfzmgJTP_1KNBTLjtQqbM5d_uhdUuGandY2G9l2V6iywKo4WJdf9kwXI
oH6K2SqEj6zJbspN2-diEZlrIFd2hLK-BFnlV2jA_JccPuNRq-; expires=Sat, 22-Jul-2017 14:29:32 GMT;
path=/; domain=.google.com; HttpOnly

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A
HREF="http://www.google.co.th/?gws_rd=cr&amp;ei=zB6CWL6jA4uMvQT0ya6wBA">here</A>.
</BODY></HTML>
```

2.1.7 การไบนด์

การไบนด์ (bind) คือ การเชื่อมความสัมพันธ์ของโฮสต์เข้ากับพอร์ตสำหรับการเขียนโปรแกรมของฝั่งเครื่องให้บริการเพื่อให้เครื่องบริการจองพอร์ตสื่อสารสำหรับรอรับ ซึ่งรูปแบบของคำสั่งเป็นดังนี้

ตัวแปรซ็อกเก็ต.**bind**((ตัวแปรซ็อกเก็ต.gethostname(), พอร์ต))

2.1.8 การรอการสื่อสาร

การรอการสื่อสาร (listen) คือ คำสั่งที่เครื่องให้บริการใช้สำหรับรอการสื่อสารจากเครื่องลูกข่าย เมื่อเครื่องลูกข่ายทำการเชื่อมต่อด้วยคำสั่ง connect หรือ connect_to เครื่องให้บริการจะตรวจสอบจำนวนลูกข่ายที่ได้เชื่อมต่อมาก่อนหน้านี้ เพื่อไม่ให้มีการเชื่อมต่อมากกว่า

จำนวนที่กำหนด ถ้ายังมีจำนวนเหลือ เครื่องให้บริการจะยอมรับการขอเชื่อมต่อจากเครื่องลูกข่าย โดยรูปแบบของการรอการสื่อสารเป็นดังนี้

ตัวแปรซ็อกเก็ต.**listen**(จำนวนลูกข่ายที่เข้ามาได้พร้อมกัน)

2.1.9 การยอมรับการสื่อสาร

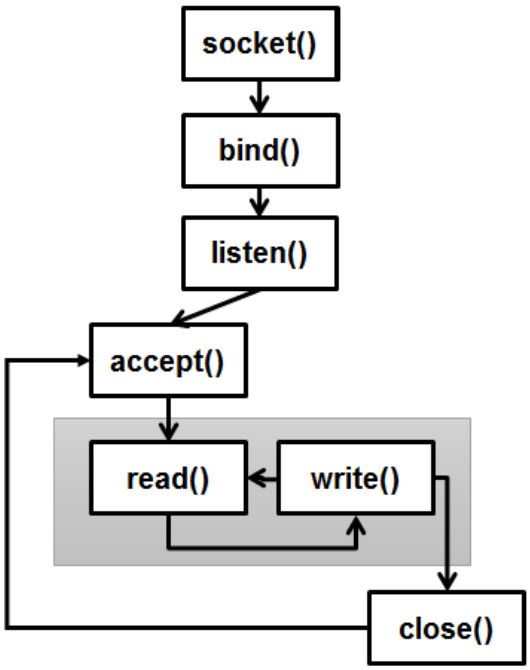
คำสั่งสำหรับตรวจสอบเหตุการณ์ยอมรับการสื่อสารจากเครื่องลูกข่าย มีรูปแบบดังนี้

(หมายเลขซ็อกเก็ต, ตำแหน่ง) = ตัวแปรซ็อกเก็ต.**accept**()

ค่าที่คืนกลับมาจากคำสั่ง accept() เป็นลิสต์ของข้อมูลหมายเลขซ็อกเก็ตของการเชื่อมต่อกับลูกข่ายและค่าตำแหน่งไอพีของเครื่องลูกข่าย

2.2 หลักการทำงานของฝั่งเครื่องให้บริการ

เครื่องให้บริการมีหน้าที่รอการร้องขอจากเครื่องลูกข่าย และเมื่อเครื่องลูกข่ายทำการเชื่อมต่อสำเร็จจะเป็นการทำงานตามรูปแบบโพรโทคอลที่ออกแบบไว้ โดยปกติแล้ว ผังการทำงานของเครื่องให้บริการโดยภาพรวมเป็นดังภาพที่ 14-2.2-1



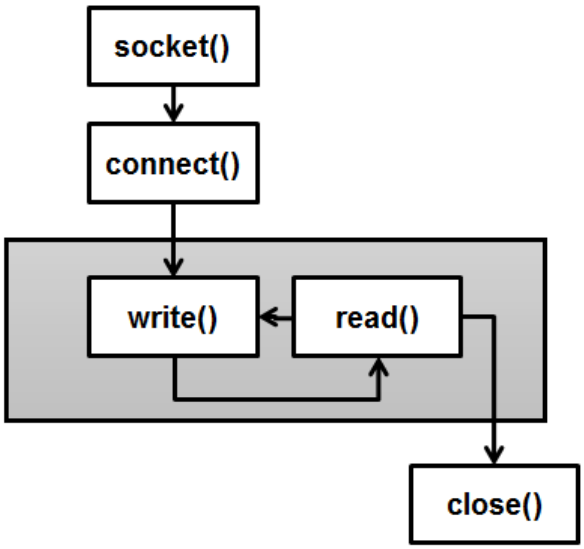
ภาพที่ 14-2.2-1 ต้นแบบผังการทำงานของโปรแกรมเอคโคฝั่งเครื่องให้บริการ

จากภาพที่ 14-2.2-1 จะพบว่า ขั้นตอนการเขียนโปรแกรมฝั่งเครื่องให้บริการมีรูปแบบพื้นฐานดังนี้

```
import socket
ssock = socket.socket( socket.AF_INET, รูปแบบการเชื่อมต่อ )
ตัวแปรตำแหน่ง=('localhost', พอร์ต)
ssock.bind( ตัวแปรตำแหน่ง )
ssock.listen( จำนวนลูกข่ายที่เข้าได้พร้อมกัน )
while True:
        client, address = ssock.accept()
        data = client.recv( ขนาดข้อมูล )
        client.send( ข้อมูลไบต์ )
        client.close()
```

2.3 หลักการทำงานของฝั่งเครื่องลูกข่าย

เครื่องลูกข่ายมีหน้าที่ร้องขอการเชื่อมต่อไปยังเครื่องให้บริการโดยระบุหมายเลขไอพีและพอร์ตเพื่อใช้ในการสื่อสารระหว่างกัน ซึ่งผังการทำงานของเครื่องให้บริการโดยภาพรวมเป็นดังภาพที่ 14-2.3-1



ภาพที่ 14-2.3-1 ต้นแบบผังการทำงานของโปรแกรมเอคโคฝั่งลูกข่าย

จากภาพที่ 14-2.3-1 จะพบว่า ขั้นตอนการเขียนโปรแกรมฝั่งเครื่องลูกข่ายมีรูปแบบพื้นฐานดังนี้

```
import socket
ssock = socket.socket( socket.AF_INET, รูปแบบการเชื่อมต่อ )
ตัวแปรตำแหน่ง=(ตำแหน่งเครื่องให้บริการ, พอร์ต)
ssock.connect( ตัวแปรตำแหน่ง )
ssock.send( ข้อมูลไบต์ )
ข้อมูลที่รับ = ssock.recv( ขนาดข้อมูล )
ssock.close()
```

2.4 การดักข้อผิดพลาด

การดักข้อผิดพลาดในการเขียนโปรแกรมสื่อสารผ่านโพรโทคอลอินเทอร์เน็ตสามารถทำได้โดยใช้ try/except แยกตามประเภทของความผิดพลาดดังนี้

2.4.1 ความผิดพลาดเมื่อสร้างซ็อกเก็ต

การเขียนโค้ดเพื่อดักข้อผิดพลาดจากการสร้างซ็อกเก็ตเขียนตามรูปแบบต่อไปนี้

```
try:
    ตัวแปรซ็อกเก็ต = socket.socket( socket.AF_INET, ... )
except socket.error, msg:
    print("ไม่สามารถสร้างซ็อกเก็ตได้!")
```

2.4.2 ความผิดพลาดเมื่อทำการไบนด์

การเขียนโค้ดเพื่อดักข้อผิดพลาดจากการไบนด์เขียนตามรูปแบบต่อไปนี้

```
try:
    ตัวแปรซ็อกเก็ต.bind( (โฮสต์, พอร์ต) )
except socket.error, msg:
    print("ไม่สามารถ bind ได้!")
```

2.4.3 ความผิดพลาดที่เกิดจากการส่งข้อมูล

การเขียนโค้ดเพื่อดักข้อผิดพลาดจากการส่งข้อมูลเขียนตามรูปแบบต่อไปนี้

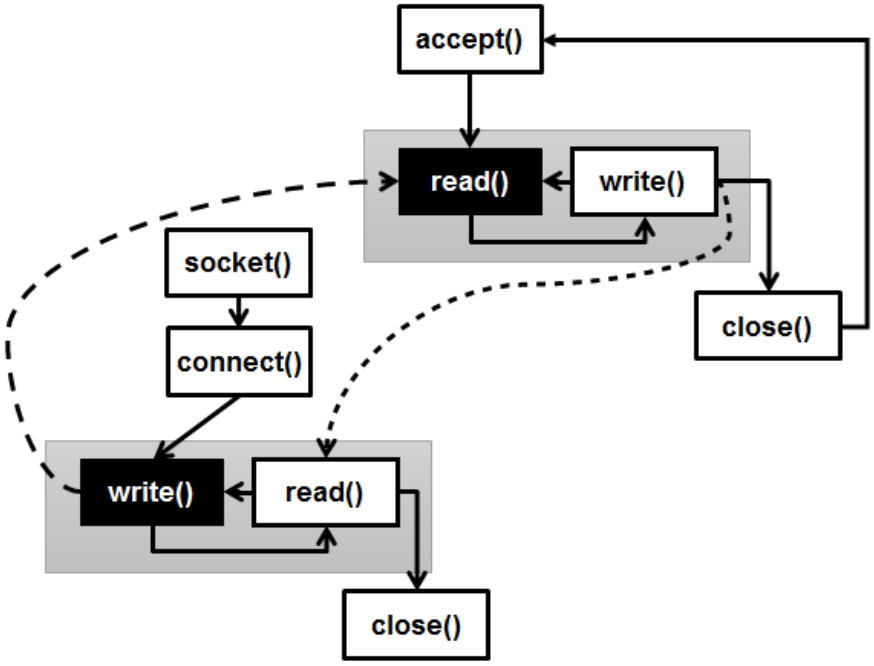
```
try:
    ตัวแปรซ็อกเก็ต.sendall( ข้อมูล )
except socket.error:
    print("ไม่สามารถส่งข้อมูลได้!")
```

3. ตัวอย่างโปรแกรม

ตัวอย่างการเขียนโปรแกรมเพื่อสื่อสารบนโพรโทคอลอินเทอร์เน็ตในบทนี้ประกอบด้วย 4 ตัวอย่าง คือ โปรแกรมเอคโค, โปรแกรมสั่งแสดงผลที่แอลซีดีผ่านเครือข่าย, โปรแกรมเปิดปิดรีเลย์ผ่านเครือข่าย และโปรแกรมร้องขอค่าความชื้นและอุณหภูมิ

3.1 โปรแกรมเอคโค

โปรแกรมเอคโคเป็นโปรแกรมที่เครื่องลูกข่ายทำการส่งข้อความไปยังเครื่องให้บริการ และเครื่องให้บริการทำการส่งข้อความนั้นกลับมาที่เครื่องลูกข่าย ตัวอย่างโปรแกรมที่ 14-3.1-1 และ 13-3.1-2 มีหลักการทำงานดังผังการทำงานในภาพที่ 14-3.1-1



ภาพที่ 14-3.1-1 ผังการทำงานโปรแกรมเอคโค

จากภาพที่ 14-3.1-1 จะพบว่า ขั้นตอนการทำงานของโปรแกรมฝั่งเครื่องลูกข่ายและเครื่องให้บริการต้องมีความสัมพันธ์กันคือ เมื่อต้องการส่งข้อมูลจากเครื่องลูกข่ายไปให้เครื่องให้บริการ โค้ดโปรแกรมของฝั่งเครื่องลูกข่ายใช้คำสั่ง send( ) และฝั่งของเครื่องให้บริการต้องเป็นคำสั่ง recv( ) และในทางตรงกันข้ามกัน ถ้าเครื่องลูกข่ายต้องการรับข้อมูลจากเครื่องให้บริการ ฝั่งลูกข่ายจะต้องใช้คำสั่ง recv( ) โดยจะต้องตรงกันกับการทำงานของฝั่งเครื่องให้บริการที่เรียกคำสั่ง send( )

โปรแกรมที่ 14-3.1-1 โปรแกรมเอคโคฝั่งเครื่องให้บริการ

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	host = '192.168.1.8' *#แก้ IP (IP ของ Host เครื่องให้บริการ)*
4	data_payload = 2048
5	backlog = 5
6	ssock=0
7	**def server(port):**
8	global ssock
9	ssock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10	ssock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
11	server_address = (host, port)
12	print("Starting up echo server on ",server_address)
13	ssock.bind(server_address)
14	ssock.listen(backlog)
15	while True:
16	print("Waiting to receive message from client")
17	client, address = ssock.accept()
18	data = client.recv(data_payload)

```
19                      if data:
20                              data_str = data.decode(encoding='UTF-8')
21                              print("Data: ", data_str)
22                              client.send(data)
23                              print("sent ",data_str," bytes back to ", address)
24                      client.close()
25      print("Network Programming :: Echo (14-3.1-1)")
26      print("กดแป้น  Ctrl-C เพื่อออกจากโปรแกรม")
27      try:
28              server(9999)
29      except KeyboardInterrupt:
30              ssock.close()
```

จากโปรแกรม 14-3.1-1 ได้ตัวอย่างของผลลัพธ์การทำงานเมื่อฝั่งลูกข่ายส่งข้อความ “Hello, Raspberry Pi3. I’m ETT” ดังนี้

```
Network Programming :: Echo (14-3.1-1)
กดแป้น  Ctrl-C เพื่อออกจากโปรแกรม
Starting up echo server on  ('192.168.1.8', 9999)
Waiting to receive message from client
Data:  Hello, Raspberry Pi3. I'm ETT.
sent  Hello, Raspberry Pi3. I'm ETT.  bytes back to  ('192.168.1.11', 9821)
```

โปรแกรมที่ 14-3.1-2 โปรแกรมเอคโคฝั่งเครื่องลูกข่าย

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	host = '192.168.1.8' *#แก้ IP (IP เหมือนตัวอย่าง14-3.1-1)*
4	**def client(port):**
5	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6	server_address = (host, port)
7	print("Connecting to ", server_address)
8	sock.connect(server_address)
9	message = input("Message ? ")
10	msg_byte = message.encode(encoding='UTF-8')
11	print("Sending ",message)
12	sock.sendall(msg_byte)
13	amount_received = 0
14	amount_expected = len(message)
15	while amount_received < amount_expected:
16	data = sock.recv(16)
17	amount_received += len(data)
18	data_str = data.decode(encoding='UTF-8')
19	print("Received: ",data_str)
20	sock.close()
21	print("Network Programming :: Echo/Client (14-3.1-2)")
22	client(9999)

จากโปรแกรม 14-3.1-2 ได้ตัวอย่างของผลลัพธ์การทำงานเมื่อรับข้อความ “Hello, Raspberry Pi3. I’m ETT” และกดแป้น Enter ดังนี้

```
Connecting to  ('192.168.1.8', 9999)
Message ? Hello, Raspberry Pi3. I'm ETT.      <- -พิมพ์ข้อความ
Sending  Hello, Raspberry Pi3. I'm ETT.
Received:  Hello, Raspberry
Received:   Pi3. I'm ETT.
Network Programming :: Echo/Client (14-3.1-1)
```

หมายเหตุ

1. การทดลองโปรแกรมจะต้องใช้ Raspberry Pi จำนวน 2 บอร์ด หรือใช้บอร์ด Raspberry Pi กับเครื่องคอมพิวเตอร์ โดยโปรแกรมฝั่งเครื่องลูกข่ายต้องรันที่เครื่องตัวใดตัวหนึ่ง และอีกเครื่องหนึ่งต้องรันโปรแกรมฝั่งเครื่องให้บริการ โดยผู้รันโปรแกรมจะต้องทราบหมายเลขไอพีของอุปกรณ์ทั้ง 2 ฝั่ง พร้อมทั้งต้องเปิดให้ไฟล์วอลล์ยอมรับหมายเลขพอร์ตที่ระบุ (ตัวอย่างใช้พอร์ต 9999)

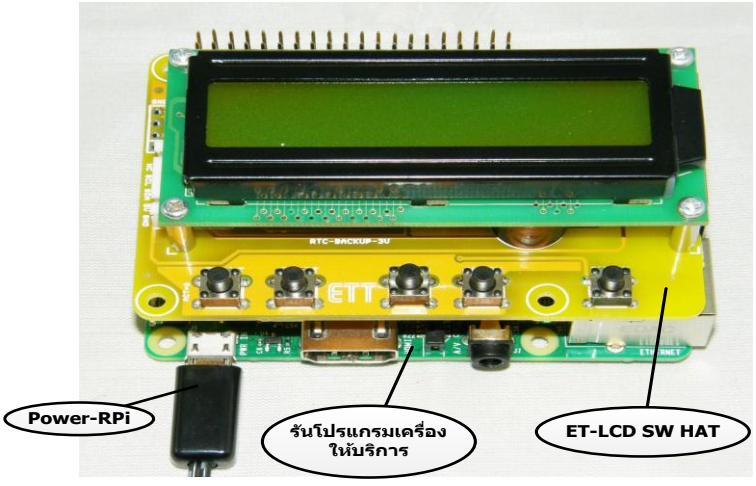
2. ต้องกำหนดหมายเลขไอพีของเครื่องให้บริการในบรรทัดที่ 3 ของโปรแกรมตัวอย่าง 14-3.1-1 และ 14-3.1-2 ให้ถูกต้อง ห้ามใส่เป็น ‘localhost’

3. กรณีที่เกิดการเตือนดังข้างล่าง ให้ทำการรีบูตเครื่องที่เป็นเครื่องให้บริการใหม่

```
PermissionError: [Errno 13] Permission denied
```

3.2 โปรแกรมสั่งแสดงข้อความที่แอลซีดีผ่านเครือข่าย

โปรแกรมสั่งแสดงข้อความที่แอลซีดีผ่านเครือข่ายเป็นการรันโปรแกรมฝั่งเครื่องให้บริการที่บอร์ด Raspberry Pi ที่ติดตั้งบอร์ด ET-LCD SW HAT ดังภาพที่ 14-3.2-1 และเครื่องลูกข่ายเป็นอุปกรณ์อื่นๆ ที่รันโปรแกรมฝั่งลูกข่ายโดยรับข้อความสตริงเพื่อนำมาแปลงเป็นไบต์เพื่อส่งไปให้ Raspberry Pi หลังจากนั้นทำการแปลงข้อมูลไบต์ให้กลายเป็นสตริงเพื่อส่งให้บอร์ด ET-LCD SW HAT ดังตัวอย่างโปรแกรมที่ 14-3.2-1 และ 14-3.2-2



ภาพที่ 14-3.2-1 การต่อบอร์ดเพื่อโปรแกรมสั่งแสดงข้อความที่ LCD ผ่านเครือข่าย

โปรแกรมที่ 14-3.2-1 โปรแกรมสั่งแสดงข้อความที่แอลซีดีผ่านเครือข่ายฝั่งเครื่องให้บริการ

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	import RPi.GPIO as GPIO
4	import time
5	host = '192.168.1.8' *#แก้ IP (IP ของ Host เครื่องให้บริการ)*
6	data_payload = 2048
7	backlog = 5
8	ssock=0
9	LCD_RS = 7
10	LCD_EN = 8
11	LCD_D4 = 18
12	LCD_D5 = 23
13	LCD_D6 = 24
14	LCD_D7 = 25
15	LCD_BL = 4
16	LCD_CMD = 0
17	LCD_DATA = 1
18	LCD = [LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7, LCD_BL]
19	**def setup():**
20	GPIO.setmode(GPIO.BCM)
21	GPIO.setwarnings(False)
22	for i in LCD:
23	GPIO.setup(i, GPIO.OUT)
24	GPIO.output(i, False)
25	**def lcd_init():**
26	time.sleep(0.5)
27	lcd_write(0x33,LCD_CMD)
28	lcd_write(0x32,LCD_CMD)
29	lcd_write(0x06,LCD_CMD)
30	lcd_write(0x0C,LCD_CMD)
31	lcd_write(0x28,LCD_CMD)
32	lcd_write(0x01,LCD_CMD)
33	time.sleep(0.005)
34	lcd_back_light(True)
35	**def lcd_send_bit(pin, byte, pos):**
36	if (byte & pos):
37	GPIO.output(pin, True)
38	else:
39	GPIO.output(pin, False)
40	**def lcd_write(byte,mode):**
41	GPIO.output(LCD_RS, mode)
42	lcd_send_bit(LCD_D4, byte, 0x10)
43	lcd_send_bit(LCD_D5, byte, 0x20)
44	lcd_send_bit(LCD_D6, byte, 0x40)
45	lcd_send_bit(LCD_D7, byte, 0x80)
46	lcd_enable()
47	lcd_send_bit(LCD_D4, byte, 0x1)
48	lcd_send_bit(LCD_D5, byte, 0x2)
49	lcd_send_bit(LCD_D6, byte, 0x4)
50	lcd_send_bit(LCD_D7, byte, 0x8)
51	lcd_enable()

```
52  def lcd_enable():
53          GPIO.output(LCD_EN,0)
54          time.sleep(0.003)
55          GPIO.output(LCD_EN,1)
56          time.sleep(0.003)
57          GPIO.output(LCD_EN,0)
58  def lcd_back_light(isOn):
59          GPIO.output(LCD_BL,isOn)
60  def lcd_cursor(colum,line):
61          if line==0x01:
62                  line = 0x80 | colum
63                  lcd_write(line,LCD_CMD)
64          else:
65                  line = 0xC0 | colum
66                  lcd_write(line,LCD_CMD)
67  def lcd_string(message):
68          for i in range(len(message)):
69                  lcd_write(ord(message[i]),LCD_DATA)
70  def server(port):
71          global ssock
72          ssock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
73          ssock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
74          server_address = (host, port)
75          print("Starting up echo server on ",server_address)
76          ssock.bind(server_address)
77          ssock.listen(backlog)
78          while True:
79                  print("Waiting to receive message from client")
80                  client, address = ssock.accept()
81                  data = client.recv(data_payload)
82                  if data:
83                          data_str = data.decode(encoding='UTF-8')
84                          print("Data: ", data_str)
85                          lcd_cursor(0,1)
86                          lcd_string( "               " )
87                          lcd_cursor(0,1)
88                          lcd_string( data_str )
89                  client.close()
90  print("Network Programming :: LCD Server (14-3.2-1)")
91  print("กดแป้น  Ctrl-C เพื่อออกจากโปรแกรม")
92  try:
93          setup()
94          lcd_init()
95          server(9999)
96  except KeyboardInterrupt:
97          ssock.close()
98          GPIO.cleanup()
```

ตัวอย่างผลการทำงานตามโปรแกรมที่ 14-3.2-1 เมื่อเครื่องลูกข่ายรันโปรแกรม 14-3.2-2 และส่งข้อมูล “Hello. I’m ETT.” เป็นดังภาพที่ 14-3.2-2

ภาพที่ 14-3.2-2 ผลลัพธ์การทำงานของโปรแกรม 14-3.2-1

โปรแกรมที่ 14-3.2-2 โปรแกรมสั่งแสดงข้อความที่แอลซีดีผ่านเครือข่ายฝั่งเครื่องลูกข่าย

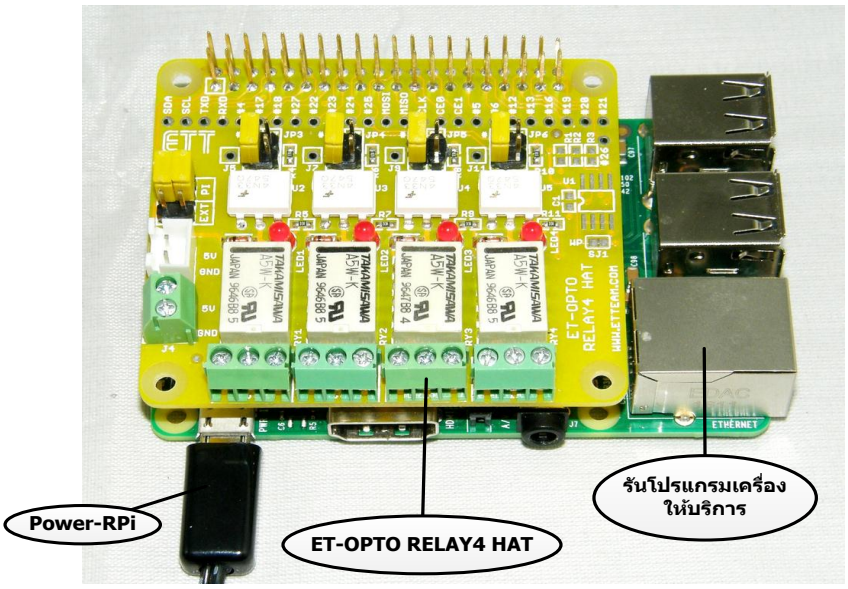
บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	host = '192.168.1.8' *#แก้ IP (IP เหมือนตัวอย่าง14-3.2-1)*
4	**def client(port):**
5	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6	server_address = (host, port)
7	print("Connecting to ", server_address)
8	sock.connect(server_address)
9	message = input("Message ? ")
10	msg_byte = message.encode(encoding='UTF-8')
11	print("Sending ",message)
12	sock.sendall(msg_byte)
13	sock.close()
14	print("Network Programming :: LCD Client (14-3.2-2)")
15	client(9999)

ตัวอย่างการทำงานของโปรแกรม 14-3.2-2 ของฝั่งเครื่องลูกข่ายที่ส่งข้อความ “Hello. I’m ETT.” ไปที่เครื่องให้บริการเพื่อส่งข้อความไปที่แอลซีดี

```
Network Programming :: LCD Client (14-3.2-1)
Connecting to  ('192.168.1.8', 9999)
Message ? Hello. I'm ETT.    <- - พิมพ์ข้อความ
Sending  Hello. I'm ETT.
```

3.3 โปรแกรมเปิด/ปิดรีเลย์ผ่านเครือข่าย

โปรแกรมเปิด/ปิดรีเลย์ผ่านเครือข่ายเป็นการรันโปรแกรมฝั่งเครื่องให้บริการที่บอร์ด Raspberry Pi ที่ติดตั้งบอร์ด ET-OPTO RELAY 4 HAT ดังภาพที่ 14-3.3-1 และเครื่องลูกข่ายรันโปรแกรมฝั่งลูกข่ายที่มีเมนูให้เลือกการทำงาน 8 รายการ คือ เปิดรีเลย์1, ปิดรีเลย์1, เปิดรีเลย์2, ปิดรีเลย์2, เปิดรีเลย์3, ปิดรีเลย์3, เปิดรีเลย์4 และปิดรีเลย์4 โดยส่งรายการตัวเลือกไปที่ฝั่งเครื่องให้บริการเพื่อให้ Raspberry Pi สั่งเปิดหรือปิดรีเลย์ตามตัวเลือกที่ได้รับ ดังตัวอย่างโปรแกรมที่ 14-3.3-1 และ 14-3.3-2



ภาพที่ 14-3.3-1 การต่อบอร์ดเพื่อโปรแกรมเปิด/ปิดรีเลย์ผ่านเครือข่าย

โปรแกรมที่ 14-3.3-1 โปรแกรมเปิด/ปิดรีเลย์ผ่านเครือข่ายฝั่งเครื่องให้บริการ

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	import RPi.GPIO as GPIO
4	import time
5	host = '192.168.1.8' *#แก้ IP (IP ของ Host เครื่องให้บริการ)*
6	data_payload = 2048
7	backlog = 5
8	ssock=0
9	Relays = [4, 18, 22, 24]
10	**def setup():**
11	GPIO.setmode(GPIO.BCM)
12	GPIO.setwarnings(False)
13	for i in Relays:
14	GPIO.setup(i, GPIO.OUT)
15	GPIO.output(i, False)
16	**def server(port):**
17	global ssock
18	ssock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19	ssock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
20	server_address = (host, port)
21	print("Starting up echo server on ",server_address)
22	ssock.bind(server_address)
23	ssock.listen(backlog)
24	while True:
25	print("Waiting to receive message from client")
26	client, address = ssock.accept()
27	data = client.recv(data_payload)
28	if data:
29	data_str = data.decode(encoding='UTF-8')
30	print("Data: ", data_str)
31	if (data_str == '1'):
32	GPIO.output(Relays[0], True)
33	if (data_str == '2'):

```
34                                      GPIO.output( Relays[0], False )
35                              if (data_str == '3'):
36                                      GPIO.output( Relays[1], True )
37                              if (data_str == '4'):
38                                      GPIO.output( Relays[1], False )
39                              if (data_str == '5'):
40                                      GPIO.output( Relays[2], True )
41                              if (data_str == '6'):
42                                      GPIO.output( Relays[2], False )
43                              if (data_str == '7'):
44                                      GPIO.output( Relays[3], True )
45                              if (data_str == '8'):
46                                      GPIO.output( Relays[3], False )
47                      client.close()
48      print("Network Programming :: Relay Server (14-3.3-1)")
49      print("กดแป้น  Ctrl-C เพื่อออกจากโปรแกรม")
50      try:
51              setup()
52              server(9999)
53      except KeyboardInterrupt:
54              ssock.close()
55              GPIO.cleanup()
```

โปรแกรมที่ 14-3.3-2 โปรแกรมเปิด/ปิดรีเลย์ผ่านเครือข่ายฝั่งเครื่องลูกข่าย

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	host = '192.168.1.8' *#แก้ IP (IP เหมือนตัวอย่าง14-3.3-1)*
4	**def menu():**
5	print("1. On Relay1\n2. Off Relay1")
6	print("3. On Relay2\n4. Off Relay2")
7	print("5. On Relay3\n6. Off Relay3")
8	print("7. On Relay4\n8. Off Relay4")
9	**def client(port):**
10	menu()
11	choice = int(input("Select ? "))
12	if (choice >= 1) and (choice <= 8):
13	ch_str = str(choice)
14	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15	server_address = (host, port)
16	print("Connecting to ", server_address)
17	sock.connect(server_address)
18	msg_byte = ch_str.encode(encoding='UTF-8')
19	print("Sending ",ch_str)
20	sock.sendall(msg_byte)
21	sock.close()
22	else:
23	print("Wrong choice ....")
24	print("Network Programming :: Relay Client (14-3.3-2)")
25	client(9999)

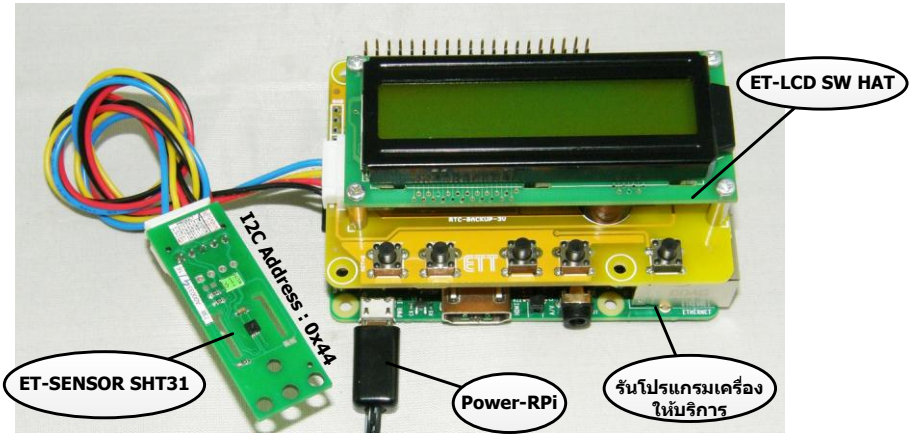
ตัวอย่างหน้าจอของฝั่งลูกข่ายเมื่อรันโปรแกรมที่ 14-3.3-2 เป็นดังนี้

```
Network Programming :: Relay Client (14-3.3-1)
1. On Relay1
2. Off Relay1
```

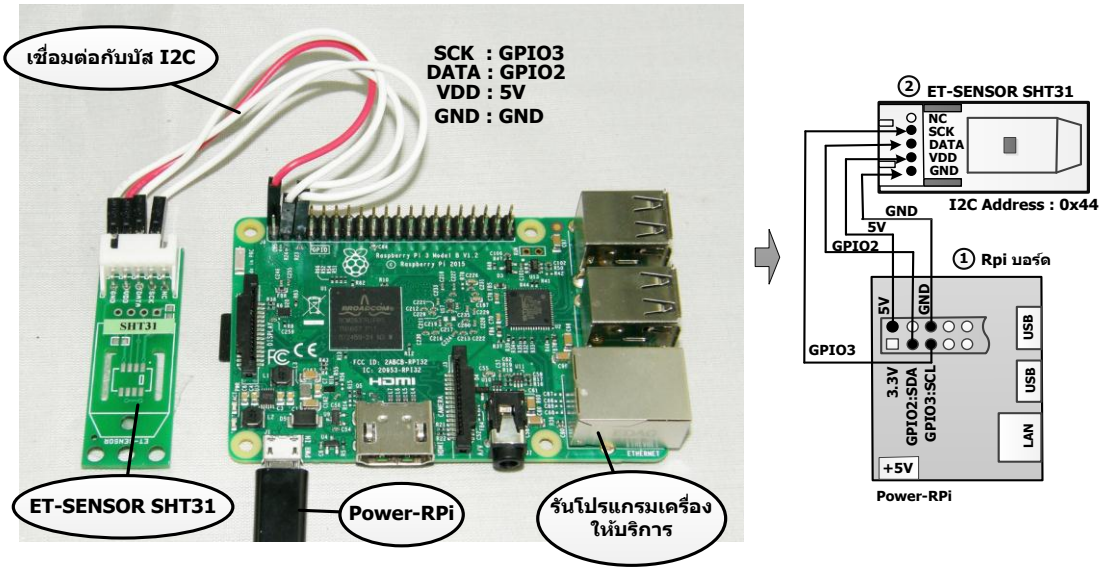
3. On Relay2
4. Off Relay2
5. On Relay3
6. Off Relay3
7. On Relay4
8. Off Relay4

3.4 โปรแกรมร้องขอค่าความชื้นและอุณหภูมิ

โปรแกรมร้องขอค่าความชื้นและอุณหภูมิผ่านเครือข่ายเป็นการรันโปรแกรมฝั่งเครื่องให้บริการที่บอร์ด Raspberry Pi ที่ติดตั้งบอร์ด ET-LCD SW HAT และ ET-SENSOR SHT31 หรือต่อ ET-SENSOR SHT31 โดยตรงกับบอร์ด Raspberry Pi ดังภาพที่ 14-3.4-1 และ 14-3.4-2 เพื่ออ่านค่าความชื้นและอุณหภูมิของจุดที่บอร์ด Raspberry Pi โดยการทำงานของโปรแกรมลูกข่าย คือ เชื่อมต่อ เมื่อเชื่อมต่อสำเร็จจะร้องขออ่านข้อมูลความชื้น หลังจากนั้นร้องขออ่านข้อมูลอุณหภูมิเพื่อแสดงที่หน้าจอของฝั่งลูกข่ายหลังจากนั้นจบการทำงานของโปรแกรมฝั่งลูกข่าย โดยโค้ดภาษาไพธอนเป็นดังโปรแกรมที่ 14-3.4-1 และ 14-3.4-2



ภาพที่ 14-3.4-1 การต่อบอร์ดเพื่อโปรแกรมร้องขอค่าความชื้นและอุณหภูมิแบบที่ 1



ภาพที่ 14-3.4-2 การต่อบอร์ดเพื่อโปรแกรมร้องขอค่าความชื้นและอุณหภูมิแบบที่ 2

โปรแกรมที่ 14-3.4-1 โปรแกรมร้องขอค่าความชื้นและอุณหภูมิฝั่งเครื่องให้บริการ

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	import time
4	import smbus
5	SHT31 = 0x44
6	host = '192.168.1.8' *#แก้ IP (IP ของ Host เครื่องให้บริการ)*
7	data_payload = 2048
8	backlog = 5
9	ssock=0
10	SHT31_Enabled = 0x2C
11	SHT31_Enabled_High = [0x06]
12	SHT31_Data = [0,0,0,0,0,0]
13	bus = 0
14	**def setup():**
15	global bus
16	bus = smbus.SMBus(1)
17	**def sht31_start():**
18	global bus
19	bus.write_i2c_block_data(SHT31, SHT31_Enabled, SHT31_Enabled_High)
20	**def sht31_read():**
21	global bus
22	SHT31_Data = bus.read_i2c_block_data(SHT31, 0x00, 6)
23	Humidity = (int(SHT31_Data[3])<<8) + SHT31_Data[4]
24	RH = round(100 * (Humidity / 65535.0))
25	Temp = (int(SHT31_Data[0])<<8)+SHT31_Data[1]
26	TemperatureC = round(-45 + (175 * (Temp /65535.0)))
27	return [str(RH), str(TemperatureC)]
28	**def server(port):**
29	global ssock
30	ssock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
31	ssock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
32	server_address = (host, port)
33	print("Starting up echo server on ",server_address)
34	ssock.bind(server_address)
35	ssock.listen(backlog)
36	while True:
37	print("Waiting to receive message from client")
38	client, address = ssock.accept()
39	sht31_start()
40	data = sht31_read()
41	print(data)
42	data1 = data[0].encode(encoding='UTF-8')
43	time.sleep(0.1)
44	data2 = data[1].encode(encoding='UTF-8')
45	time.sleep(0.1)
46	client.send(data1)
47	client.send(data2)
48	client.close()
49	print("Network Programming :: RH/Temp Server (14-3.4-1\กดแป้น Ctrl-C เพื่อออกจากโปรแกรม")
50	try:
51	setup()
52	server(9999)
53	except KeyboardInterrupt:
54	ssock.close()

โปรแกรมที่ 14-3.4-2 โปรแกรมร้องขอค่าความชื้นและอุณหภูมิฝั่งเครื่องลูกข่าย

บรรทัด	โค้ด
1	# -*- coding: utf-8 -*-
2	import socket
3	host = '192.168.1.8' *#แก้ IP (IP เหมือนตัวอย่าง14-3.4-1)*
4	
5	**def client(port):**
6	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7	server_address = (host, port)
8	print("Connecting to ", server_address)
9	sock.connect(server_address)
10	data1 = sock.recv(32)
11	data2 = sock.recv(32)
12	sock.close()
13	print("%RH = "+str(data1))
14	print("Temp = "+str(data2))
15	
16	print("Network Programming :: RH/Temp Client (14-3.4-2)")
17	client(9999)

4. สรุป

จากบทนี้จะพบว่าการเขียนโปรแกรมเพื่อสื่อสารกับบอร์ด Raspberry Pi ผ่านทางโพรโทคอลอินเทอร์เน็ตนั้นไม่มีความซับซ้อนถ้ามีการออกแบบขั้นตอนของการรับส่งข้อมูลระหว่างโปรแกรมทั้ง 2 ฝั่งอย่างรอบคอบ และมีการแปลงข้อมูลจากสตริงเป็นไบต์และจากไบต์เป็นสตริงทุกครั้งในการส่งออกและรับข้อมูลเข้า

ในการทดสอบโปรแกรมกับระบบปฏิบัติการจะต้องตรวจสอบในเรื่องของการเปิดพอร์ตสื่อสารหมายเลขที่กำหนดเอาไว้ เนื่องจากระบบไฟร์วอลล์มักปิดกั้นการเชื่อมต่อในพอร์ตที่ไม่ได้ถูกใช้เป็นมาตรฐาน และในบางหน่วยงานมีการวางไฟร์วอลล์เพื่อปิดกั้นการเชื่อมต่อที่ไม่พึงประสงค์ทำให้โปรแกรมไม่สามารถทำงานได้