한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Internet of Things (IoT) Systems

Week 12

Raspberry Pi Programming

Ikram Syed, Ph.D.
Associate Professor
Department of Information and Communication Engineering
Hankuk University of Foreign Studies (HUFS)

Spring – 2025

OUTLINE

- Last Lecture Overview
- Controlling Raspberry Pi Sensors via the Internet
- Using Web Server and PHP

What We Want to Achieve

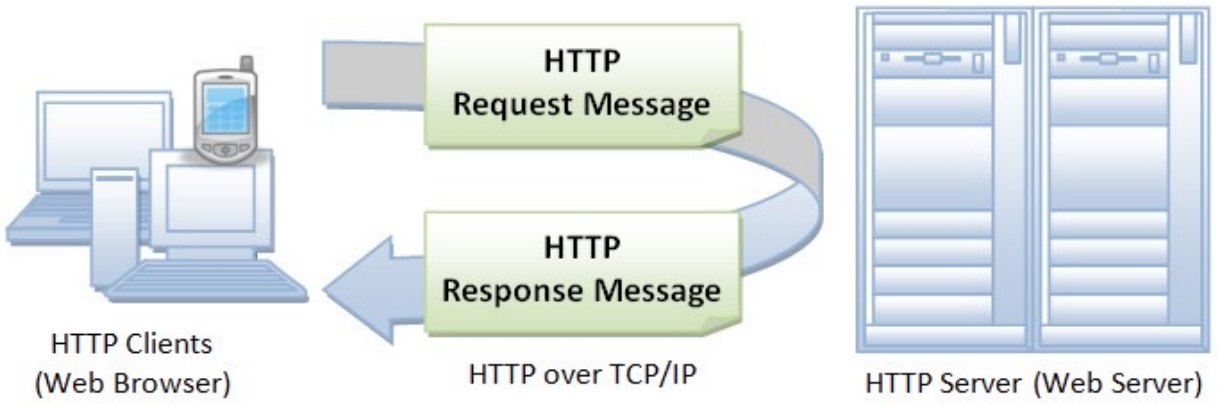
- Control different sensors from a phone or PC
- Build a web interface to send commands
  - Client: Web browser (PC or phone)
  - Server: Raspberry Pi
  - Protocol: HTTP
- **Goal:** Browser sends request → RPi executes command → sends back a response to the Client

What Tools Do We Need?

- HTML: HyperText Markup Language (for web pages)
  - HTML is used to **display contents on the client side**,
- Lighttpd: **light** + **tpd** (**t**hreaded **p**rocess-based **d**aemon) (web server)
  - lighttpd is used to **receive HTTP requests and serve web content**,
- PHP : **H**ypertext **P**reprocessor (server-side logic)
  - PHP is used to process the request and control sensors or generate responses,
- HTTP: **H**yper**T**ext **T**ransfer **P**rotocol (Communication Protocol)
  - HTTP is used to communicate between client (browser) and server (Raspberry Pi).
- GPIO: (Sensors and outputs)

HTTP

- HTTP is the communication protocol used to send and receive hypertext (HTML) documents on the Internet.
- HTTP functions as a request–response protocol in the client–server computing model
  - The client submits an HTTP request message to the server
  - The server returns a response message to the client



https://medium.com/@RayRay504/hypertext-transfer-protocol-http-6358be84964b

Overview of HTTP

- HTTP Requests
  - An HTTP request consists of a request method, a request URL, header fields, and a body
  - HTTP 1.1 defines the following request methods:
    - GET retrieves the resource identified by the request URL(page or file)
    - HEAD returns the headers identified by the request URL
    - POST sends data of unlimited length to the web server (login form)

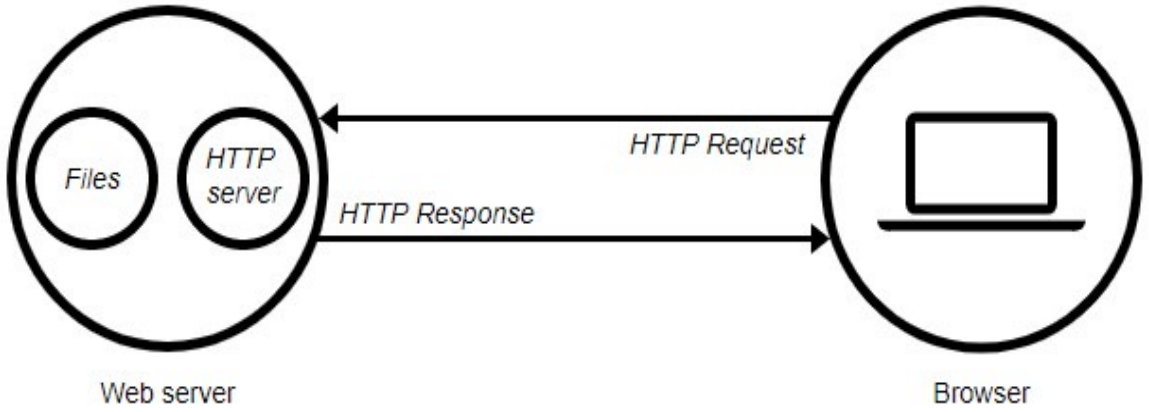
Method	Type of request (e.g., GET, POST)
URL	What resource is being requested (like /index.html)
Headers	Extra information (like browser type, language, cookies)
Body	Optional; usually used when sending form data (like login info)

Overview of HTTP

- HTTP Responses
  - An HTTP response contains a status code, header fields, and a body containing fetched resource
  - The HTTP protocol expects the result code and all header fields to be returned before any body content
  - Some commonly used status codes include:
    - 200 OK – Everything worked fine
    - 404 indicates that the requested resource is not available
    - 401 indicates that the request requires HTTP authentication
    - 500 indicates an error inside HTTP server which prevented it from fulfilling the request
    - 503 indicates that HTTP server is temporarily overloaded and unable to handle the request

Web Server

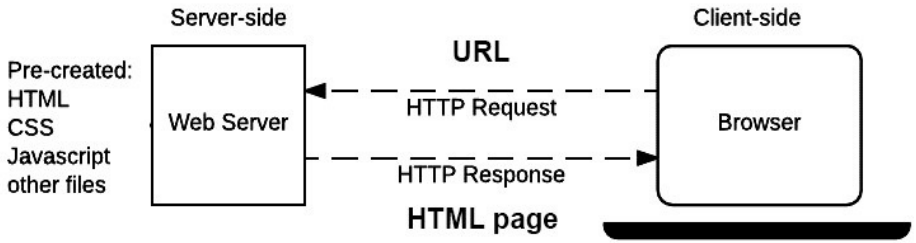
- The term web server can refer to hardware or software or both.
  - Hardware side (Raspberry Pi, cloud server)
    - A web server is a computer that stores web server software and website files
  - Software side (Apache, Nginx, lighttpd,)
    - At a minimum, web server is an HTTP server, a software that understands URLs and HTTP.
    - It's a program that handles HTTP requests and send back HTTP responses.

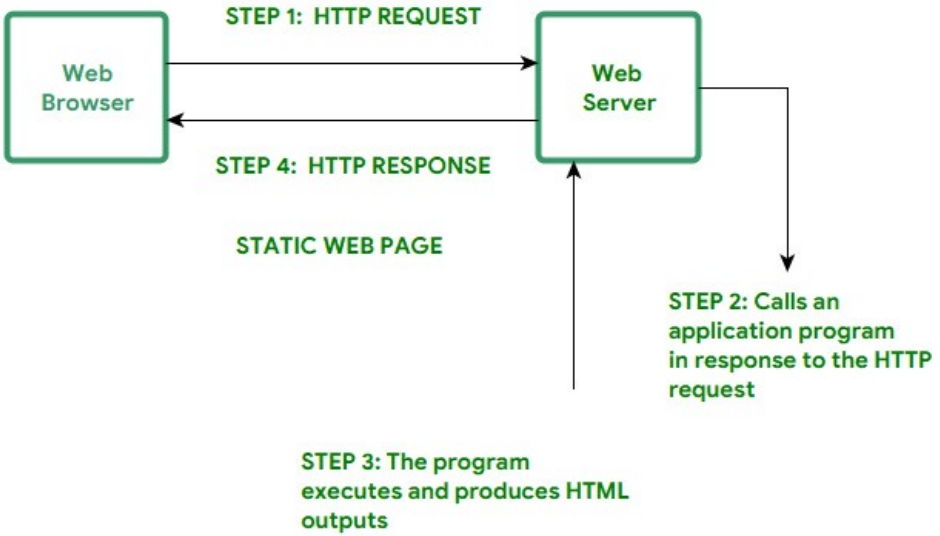


https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

Web Server

- A static web page is a web page that is delivered to the user's web browser exactly as stored
- A dynamic web page is one where contents are generated dynamically
- Static: Fixed content (HTML only)
- Dynamic: Changes based on user/action (HTML + PHP)





Web Server

- Client side and server side are web development terms that describe where application code runs
  - Client-side scripting simply means running scripts on the client device, usually within a browser, HTML, CSS
    - A script is a set of programming instructions that is interpreted at runtime
  - Server-side scripts run on the server instead of the client, often in order to deliver dynamic content to webpages in response to user actions
    - PHP
    - ASP
    - JSP
    - Perl

Lighttpd: A Lightweight Web Server

- Lighttpd stands for Light + threaded process-based daemon
- It is a fast, secure, and lightweight web server designed for low-resource systems like Raspberry Pi
- Ideal for serving static files or dynamic content using PHP (via FastCGI)
  - CGI (Common Gateway Interface) launches a new process for every request
  - FastCGI handle multiple requests
- Low memory usage and high performance
- Suitable for IoT and embedded applications

PHP

- PHP stands for Hypertext Pre-processor
- It is a server scripting language
- PHP scripts can only be interpreted on a server that has PHP installed.
- Can interact with files, databases, GPIO (on Raspberry Pi)
- Commonly used with web servers like lighttpd or Apache.

```
<html>
<body>

<?php
echo "My first PHP script!";
?>

</body>
</html>
```

Web Server Installation

- lighttpd is an open-source web server optimized for speed-critical environments
- Install lighttpd

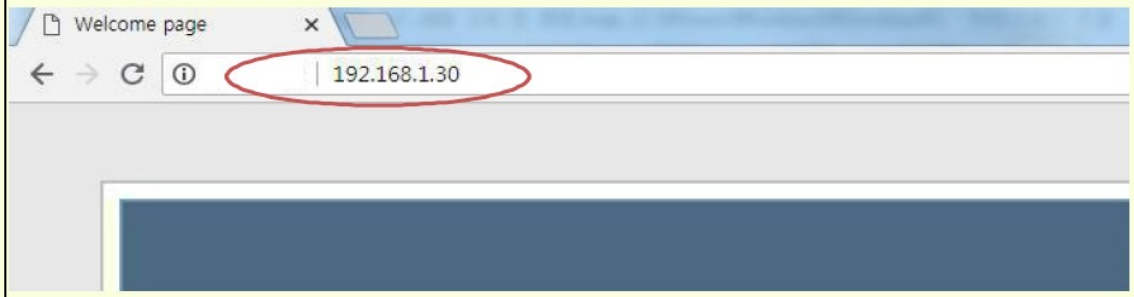
Enter 'sudo apt install lighttpd'

- Confirm installation

```
pi@raspberrypi:~ $ cd /var/www/html/
pi@raspberrypi:/var/www/html $ ls
index.lighttpd.html
```

[Figure 5-3] Confirming Lighttpd File

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome</title>
</head>
<body>
  <h1>Welcome to RPi</h1>
</body>
</html>
```



[Figure 5-4] Confirming Access to Lighttpd Web

PHP Installation

- Install PHP

  Enter 'sudo apt install php'

- Install PHP CGI Package

```
sudo apt install php-cgi
sudo lighttpd-enable-mod fastcgi
sudo lighttpd-enable-mod fastcgi-php
sudo service lighttpd restart
```

```
pi@raspberrypi:~ $ sudo apt install php-cgi
```

[Figure 5-6] Installign PHP CGI

```
pi@raspberrypi:~ $ sudo lighttpd-enable-mod fastcgi fastcgi-php
Enabling fastcgi: ok
Met dependency: fastcgi
Enabling fastcgi-php: ok
already enabled
Run /etc/init.d/lighttpd force-reload to enable changes
```

[Figure 5-7] Activating PHP CGI

```
pi@raspberrypi:~ $ sudo service lighttpd force-reload
```

[Figure 5-8] Restarting Lighttpd

PHP Installation

- Write PHP installation verification program

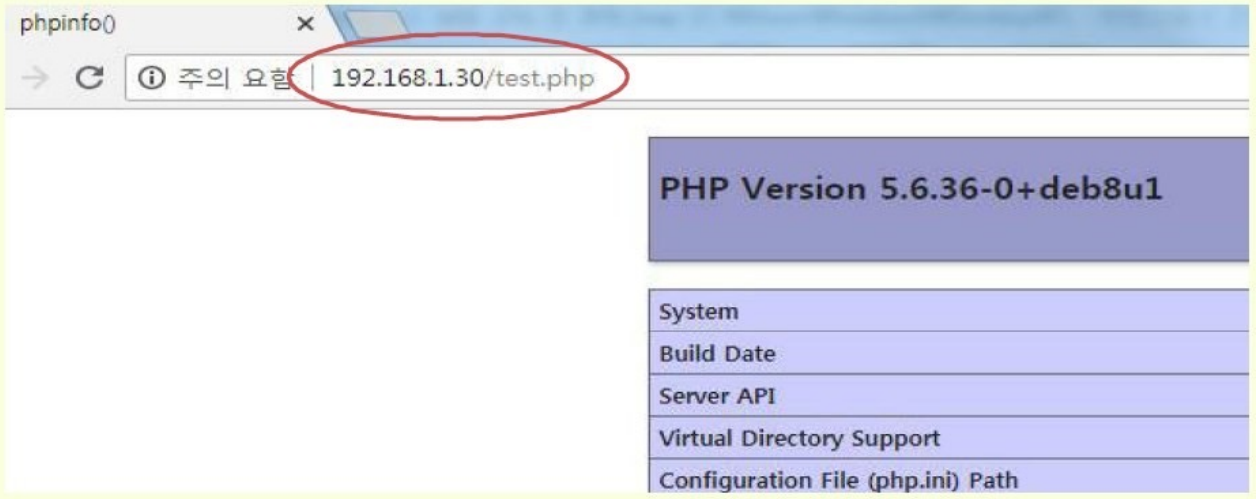
```
pi@raspberrypi:~ $ cd /var/www/html/
pi@raspberrypi:/var/www/html $
```

[Figure 5-9] Web Server Default Folder

```
pi@raspberrypi:/var/www/html $ sudo nano test.php
```

[Figure 5-10] Creating test.php File

```
<?php
phpinfo();
?>
```



[Figure 5-11] PHP Installation Information

Remote Control Using PHP

Program to control LED

```
1   #include <wiringPi.h>
2   #include <stdio.h>
3
4    #define PIN 7
5
6   int main(void) {
7       if(wiringPiSetup() == -1) return 1;
8       pinMode(PIN,OUTPUT);
9       digitalWrite(PIN,HIGH);
10      printf("1");
11  }
```

Part	Meaning
-rwsr-sr-x	File permissions and type
1	**Number of hard links** to the file
root	Owner (user) of the file
pi	Group that owns the file

```
pi@raspberrypi:~ $ gcc -o PHP_LEDON PHP_LEDON.c -lwiringPi
pi@raspberrypi:~ $ sudo chown root PHP_LEDON
pi@raspberrypi:~ $ sudo chmod +s PHP_LEDON
```

[Figure 6-6] Compiling File & Providing PHP Execution Permission

```
-rwsr-sr-x 1 root pi
```

[Figure 6-7] Changed Permissions & Owner

Remote Control Using PHP

Program to control LED

```c
#include <wiringPi.h>
#include <stdio.h>

 #define PIN 7

int main(void)  {

    if(wiringPiSetup() == -1) return 1;
    pinMode(PIN,OUTPUT);
    digitalWrite(PIN,LOW);
    printf("0");
}
```

```
pi@raspberrypi:~ $ gcc -o PHP_LEDOFF PHP_LEDOFF.c -lwiringPi
pi@raspberrypi:~ $ sudo chown root PHP_LEDOFF
pi@raspberrypi:~ $ sudo chmod +s PHP_LEDOFF
```

[Figure 6-9] Compiling File & Providing PHP Execution Permission

```
-rwsr-sr-x 1 root pi
```

[Figure 6-10] Changed Permissions & Owner

Create PHP file

```
pi@raspberrypi:~ $ cd /var/www/html
pi@raspberrypi:/var/www/html $ sudo nano remote_con.php
```

[Figure 6-11] Creating PHP File

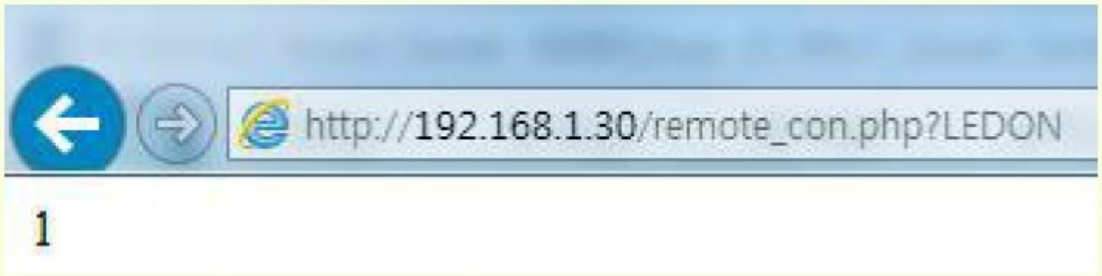
remote_con.php

```
<?php
        if(isset($_GET['LEDON'])){
                $value = shell_exec("/home/pi/PHP_LEDON");
                echo $value;
        }else if(isset($_GET['LEDOFF'])){
                $value = shell_exec("/home/pi/PHP_LEDOFF");
                echo $value;
        }
?>
```

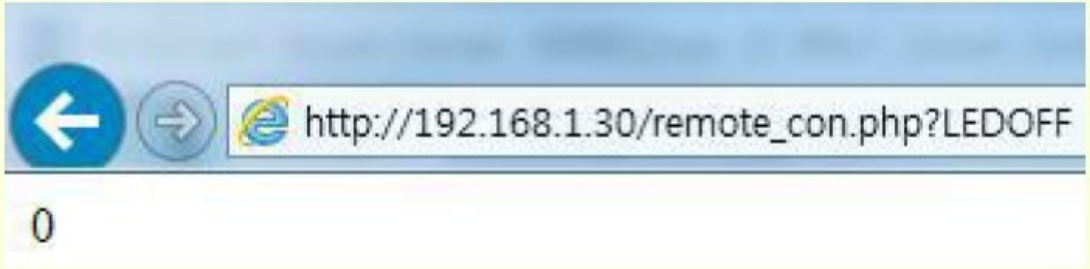
Make Sure:

- The file is executable (chmod +x)
- The path is correct and accessible to PHP
- Ownership and permissions are properly set

Check result



[Figure 6-12] Requesting LED ON



[Figure 6-13] Requesting LED OFF

Remote Control Using PHP

remote_con.php

```
<?php
        if(isset($_GET['LEDON'])){
                $value = shell_exec("/home/pi/PHP_LEDON");
                echo $value;
        }else if(isset($_GET['LEDOFF'])){
                $value = shell_exec("/home/pi/PHP_LEDOFF");
                echo $value;
        }
?>
```

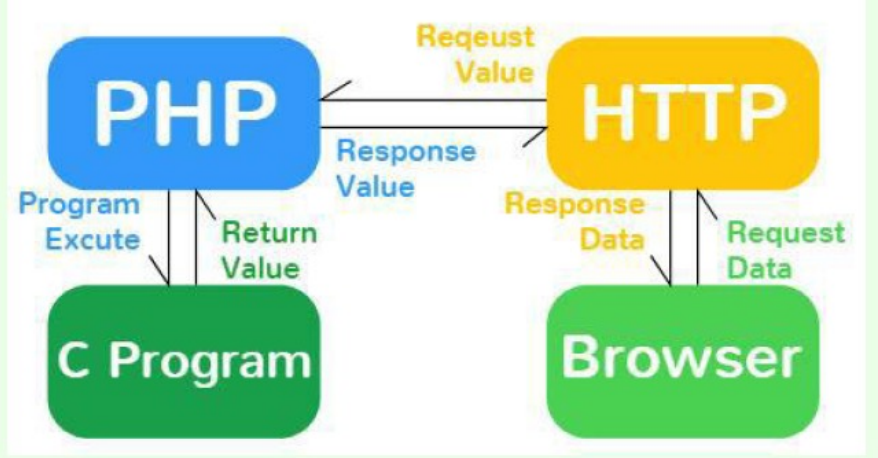
isset function returns true if the variable exists and is not NULL, otherwise it returns false

$_GET can be used to collect data sent in the URL

shell_exec function is used to execute the commands via shell and return the complete output as a string

How this process work?

- Client (browser) sends an HTTP request (e.g., clicking a button).
- lighttpd web server receives the request.
- If the request is for a .php file, lighttpd sends it to the PHP engine.
- PHP script runs on the server (Raspberry Pi):
  - Reads the request
  - Decides what to do (e.g., turn on LED)
  - Interacts with the system (calls shell command or Python code)
- The PHP script sends a response back to the client (HTML, text, sensor values).
- Browser displays the result.



Any Questions!