Embedded Linux workshop
C programming for Raspberry Pi

ผศ.ดร.ศุภชัย วรพจน์พิศุทธิ์
ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์
ม.ธรรมศาสตร์

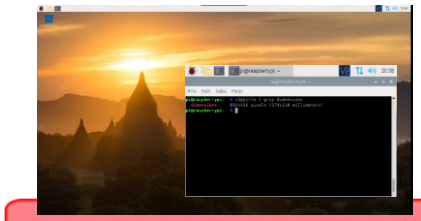
Learning outcomes

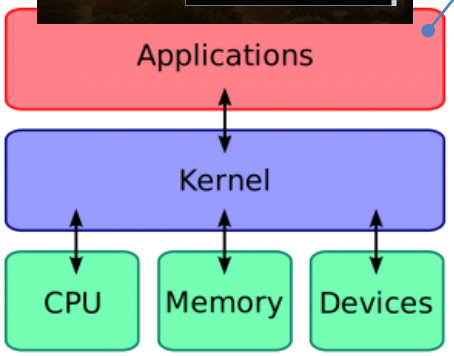
Objectives	Practice
1. Learn how to develop C/C++ application in embedded Linux environment	1. Prepare embedded Linux environment. 1. Develop C/C++ code remotely.
2. Learn how to write C code to access Linux services	2. Develop C code to access filesystem 2. Use Makefile to build database app
3. Understand multi-threading application	3. Develop multi-threading application using POSIX API.
4. Understand how to develop application for Internet of Things	4. Use libcurl to access REST API 4. Use libpaho-mqtt to access MQTT broker
5. Understand how to automate operations	5. Use cron to run application periodically

Evaluation criteria

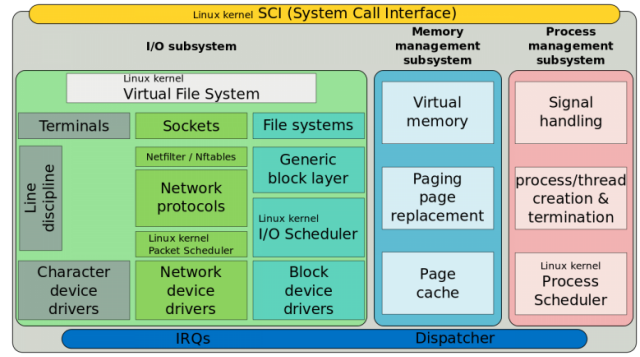
Objectives	Criteria (score)
1. Complete practice session	Screenshots (10 * 4)
2. Multi-thread two-way IoT application • JSON commands `{"check": "Mem????"}` • Record timestamp/cmd in **cmd.db** • Append data to Firebase **mem.json**	Makefile + Thread (10)
	Parse 2 JSON commands (15)
	Record in database (10)
	Firebase records (15)
3. Run at start	Run automatically (10)

Linux architecture





- Application
- Programming environment
- Library
- Shell
- Filesystem



Linux software architecture

Linux® is an open source operating system (OS). It was originally conceived of and created as a hobby by Linus Torvalds in 1991. Linus, while at university, sought to create an alternative, free, open source version of the MINIX operating system, which was itself based on the principles and design of Unix. That hobby has since become the OS with the largest user base, the most-used OS on publicly available internet servers, and the only OS used on the top 500 fastest supercomputers.

Users often choose to obtain the operating system and the application packages from one of the Linux distributors. Debian GNU/Linux is a particular distribution of the Linux operating system, and numerous packages that run on it. Debian includes more than 64961 software packages at present. Users can select which packages to install; Debian provides a tool for this purpose.

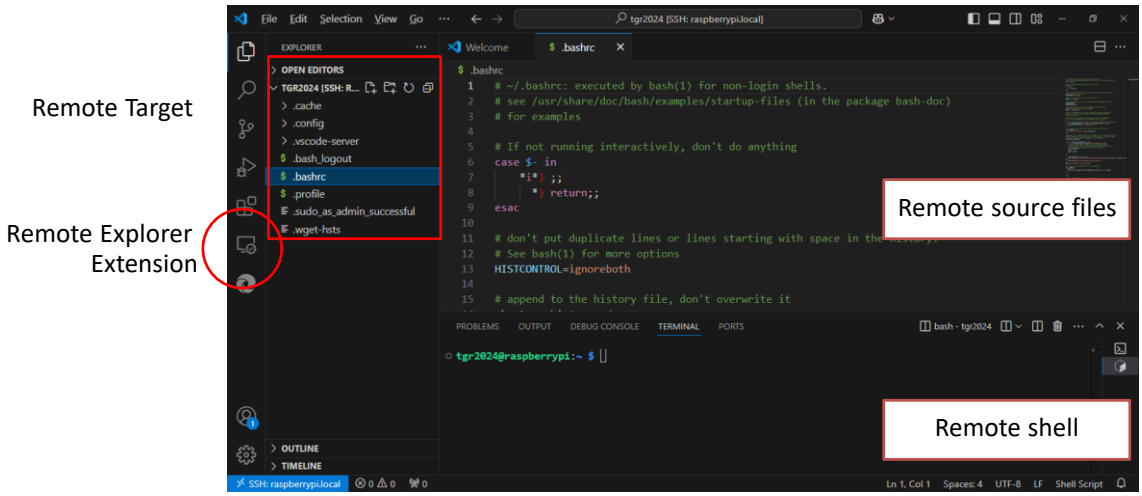
Raspberry Pi OS is a free, Debian-based operating system optimised for the Raspberry Pi hardware. Raspberry Pi OS supports over 35,000 Debian packages. We recommend Raspberry Pi OS for most Raspberry Pi use cases. The latest version of Raspberry Pi OS is based on Debian Bookworm.

Bash is the shell, or command language interpreter, for the GNU operating system. At its base, a shell is simply a macro processor that executes commands. A Unix shell is both a command interpreter and a programming language. As a command interpreter, the shell provides the user interface to the rich set of GNU utilities. The programming language features allow these utilities to be combined. Files containing commands can be created, and become commands themselves. These new commands have the same status as system commands in directories such as /bin, allowing users or groups to establish custom environments to automate their common tasks.

https://www.raspberrypi.com/documentation/computers/os.html

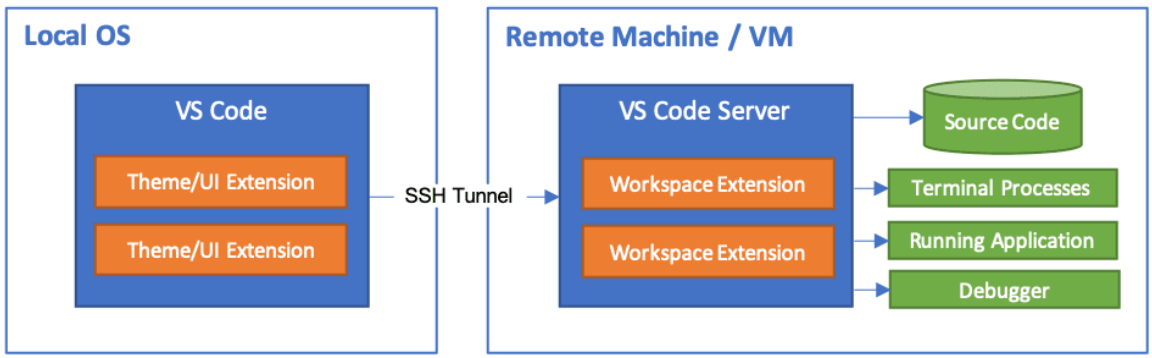
https://raspberrytips.com/raspberry-pi-commands/

VS Code + Remote Explorer extension



Remote software development

Visual Studio Code Remote Development allows you to use a container, remote machine, or the Windows Subsystem for Linux (WSL) as a full-featured development environment. No source code needs to be on your local machine to get these benefits. Each extension in the Remote Development extension pack can run commands and other extensions directly inside a container, in WSL, or on a remote machine so that everything feels like it does when you



Remote Development extension pack includes four extensions:

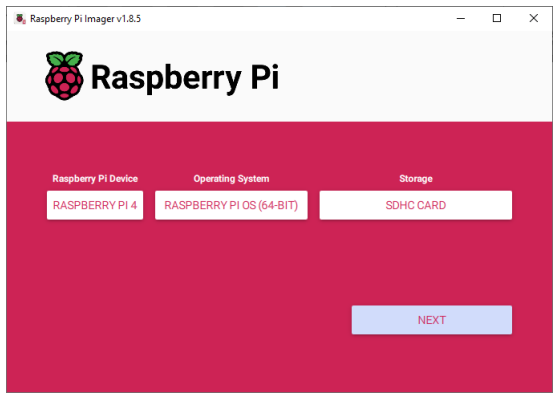
- Remote – SSH: Connect to any location by opening folders on a remote machine/VM using SSH.
- Dev Containers: Work with a separate toolchain or container-based application inside (or mounted into) a container.
- WSL: Get a Linux-powered development experience in the Windows Subsystem for Linux.
- Remote – Tunnels: Connect to a remote machine via a secure tunnel, without using SSH.

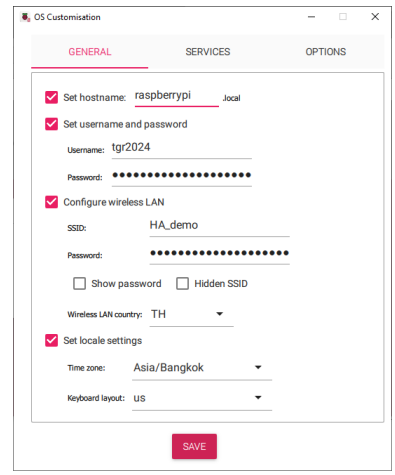
https://www.raspberrypi.com/news/coding-on-raspberry-pi-remotely-with-visual-studio-code/

Practice #1: headless dev. workflow

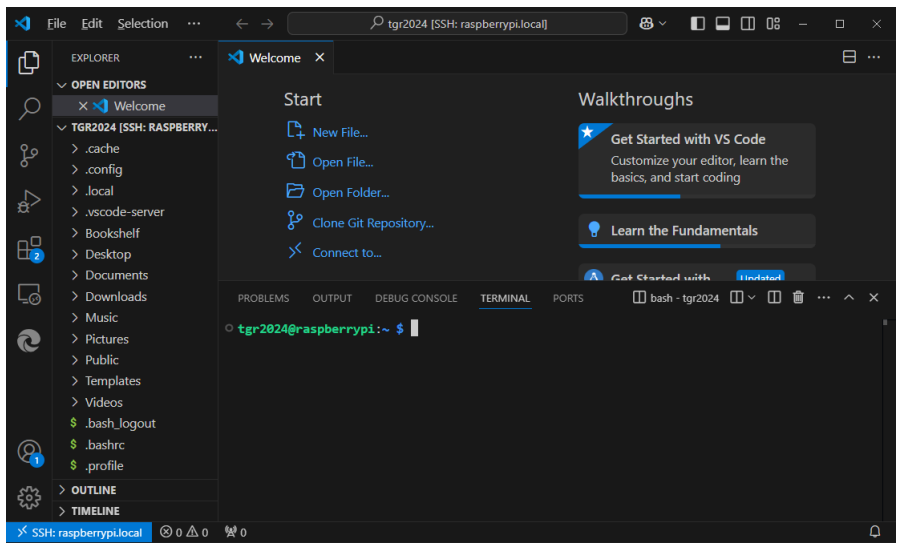
Procedure

1. Install Raspberry Pi Imager: https://www.raspberrypi.com/software/
2. Prepare SD-card with Raspberry Pi OS 64 bit (Full or Lite) with the following settings:
   - Hostname: `raspberrypi.local`
   - User/password: `tgr2024`/`tgr2024`
   - WiFi SSID/passphrase: your WiFi hotspot



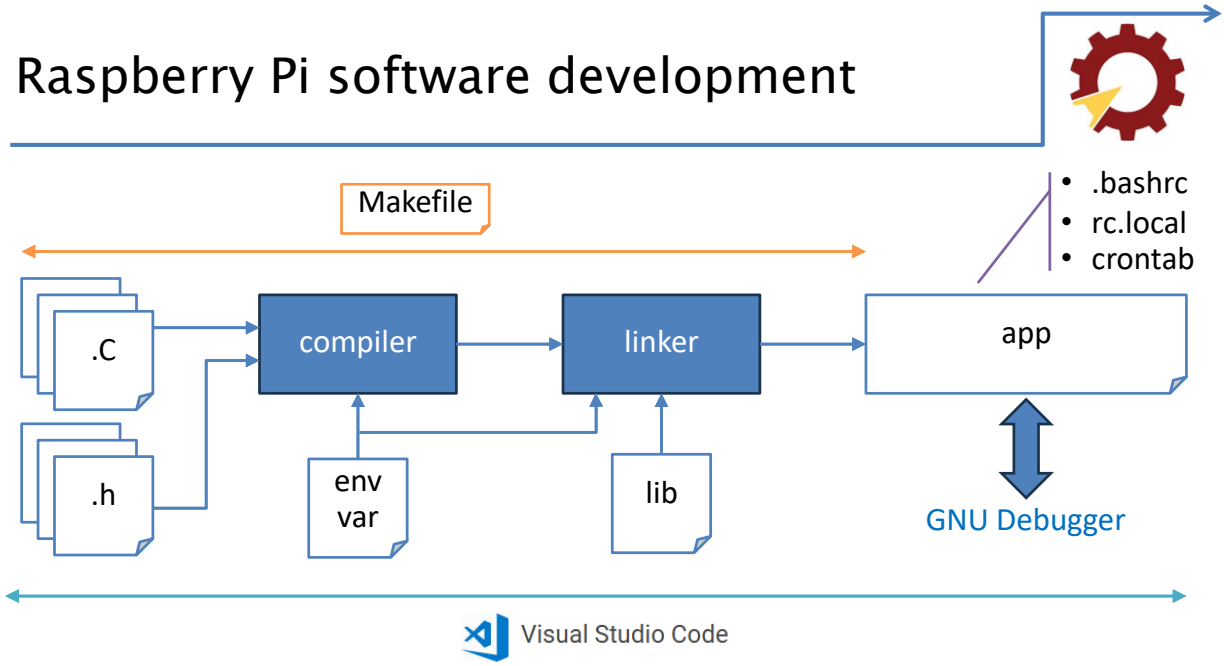


3. Put SD-card into Raspberry Pi.
4. Plug USB-C cable to power Raspberry Pi board.
   - Check board: `ping raspberrypi.local`
5. Install VS Code with Remote Explorer extension: https://code.visualstudio.com/
6. Choose Remote SSH to login: `ssh tgr2024@raspberrypi.local`
   - Check .ssh setting if connection error



7. Update software system with `apt` command:

   ```
   sudo apt update && sudo apt upgrade
   ```



C/C++ coding in Raspberry Pi

GCC is a key component of so-called "GNU Toolchain", for developing applications and writing operating systems. The GNU Toolchain includes:

- GNU Compiler Collection (GCC): a compiler suite that supports many languages, such as C/C++ and Objective-C/C++.
- GNU Make: an automation tool for compiling and building applications.
- GNU Binutils: a suite of binary utility tools, including linker and assembler.
- GNU Debugger (GDB).

GNU Compiler Collection (GCC) is a collection of compilers from the GNU Project that support various programming languages, hardware architectures and operating systems. GCC has been ported to more platforms and instruction set architectures than any other compiler, and is widely deployed as a tool in the development of both free and proprietary software. GCC is also available for many embedded systems, including ARM-based and Power ISA-based chips. Arm GNU Toolchain is a community supported pre-built GNU compiler toolchain for Arm based CPUs.

GNU Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files. Make enables the end user to build and install your package without knowing the details of how that is done -- because these details are recorded in the makefile that you supply. A rule in the makefile tells Make how to execute a series of commands in order to build a target file from source files. It also specifies a list of dependencies of the target file. Make uses the makefile to figure out which target files ought to be brought up to date, and then determines which of them actually need to be updated.

Practice #2: C/C++ programming

Procedure

1. Prepare hello.c for trying argument passing and standard input:

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

int main(int argc, char *argv[]) {
    if (argc == 2) {
        printf("Sawasdee %s\n", argv[1]);
        char surname[20];
        printf("Enter your surname: ");
        if (fgets(surname, sizeof(surname), stdin) != NULL) {
            printf("Hello %s %s\n", argv[1], surname);
        }
    } else {
        printf("Use hello YOUR_NAME\n");
    }
}
```

2. Build code with gcc toolchain: `gcc hello.c -o hello`
3. Execute application: `./hello YOUR_NAME`
4. Use pipe mechanism to pass data via stdin: `echo "SURNAME" | ./hello NAME`
5. Prepare csv_gen.c for trying file I/O:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
    const char fname[] = "data.csv";
    if (argc == 2) {
        struct stat buffer;
        FILE *file;
        if (stat(fname, &buffer) != 0) {
            file = fopen(fname, "w");
            fprintf(file, "index, value\n");
        } else {
            file = fopen(fname, "a");
        }
        int rows = atoi(argv[1]);
        for (int i=0; i < rows; i++) {
            fprintf(file, "%d, %f\n", i, (float)rand()/RAND_MAX);
        }
    } else {
        printf("Use ./csv_gen NUM_ROWS\n");
    }
    return 0;
}
```

6. Build code with gcc toolchain: `gcc csv_gen.c -o csv_gen`
7. Execute application: `./csv_gen NUM_ROWS`, then check content in data.csv.

Practice #2: C/C++ programming

Procedure

8. Prepare hello.py script that print a text.

```
#!/bin/python
if __name__ == '__main__':
    print('Hello, world')
```

9. Change permission of hello.py to be executable: `chmod a+x hello.py`
10. Prepare call_py.c for invoking hello.py with pipe:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    FILE *fp;
    char buffer[100];
    fp = popen("./hello.py", "r");
    if (fp != NULL) {
        if (fgets(buffer, 100, fp) != NULL) {
            printf("Got %s\n", buffer);
        }
    }
}
```

11. Build code with gcc toolchain: `gcc call_py.c -o call_py`
12. Execute application: `./call_py`
13. Install sqlite3 header and library files: `sudo apt install sqlite3 libsqlite3-dev`
14. Modify Makefile to build with db_app.c, db_helper.c, db_helper.h. and libsqlite3.

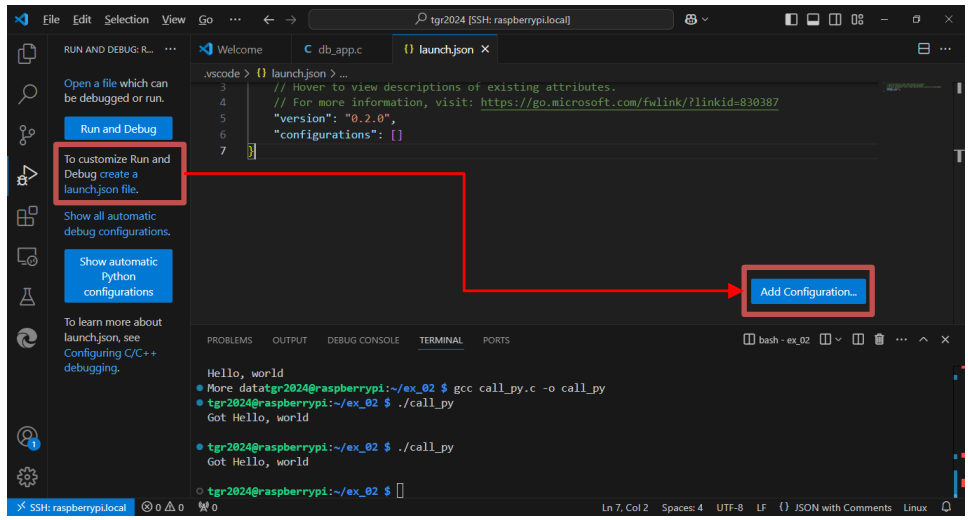
```
CC = gcc                                # Compiler and flags
CFLAGS = -Wall -Wextra -g
LIBS = -lm -lsqlite3                    # Libraries to link

TARGET = db_app                         # Target application name
SRCS = db_app.c db_helper.c             # Source files and object files
OBJS = $(SRCS:.c=.o)

all: $(TARGET)                          # Default target
$(TARGET): $(OBJS)                      # Rule to link object/lib files
    $(CC) $(CFLAGS) -o $(TARGET) $(OBJS) $(LIBS)
%.o: %.c                                # Rule to compile .c to .o files
    $(CC) $(CFLAGS) -c $< -o $@

clean:                                  # Clean up build artifacts
    rm -f $(OBJS) $(TARGET)

run: $(TARGET)                          # Run the application
    ./$(TARGET)

.PHONY: all clean run                   # Phony targets
```

15. Build application with `make` command.
16. Study example code, then use commands to initialize database, append new data, and query the value of last entry.

Practice #2: C/C++ programming

Procedure

17. Press Ctrl + Shift + D to activate Run and Debug environment.



18. Modify **Launch.json** file by add C/C++: (gdb) Launch configuration, then modify

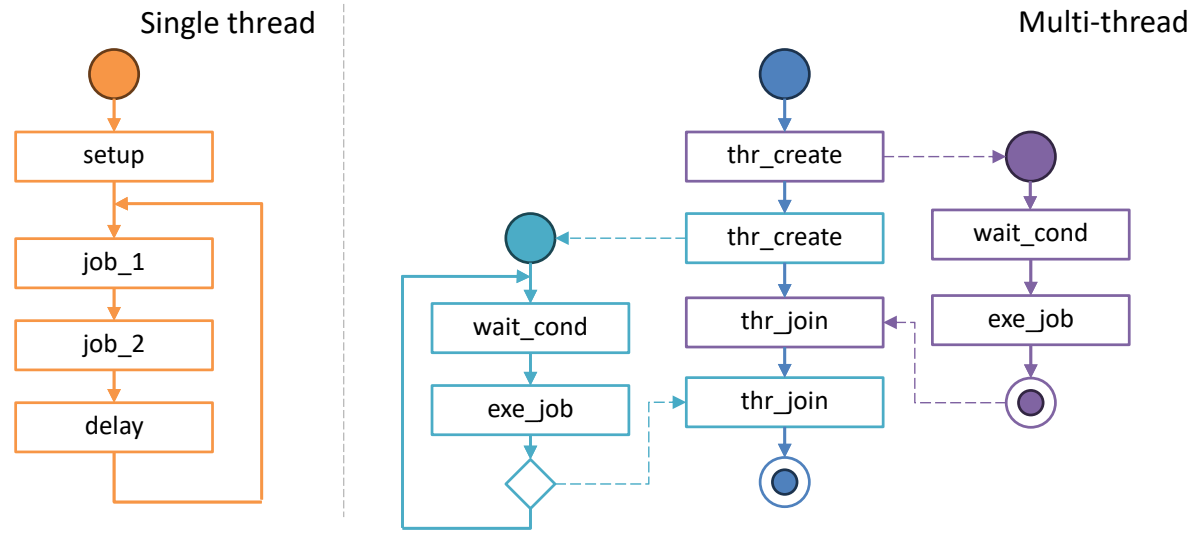
```
"configurations": [
    {
        "name": "(gdb) Launch",
        ...
        "program": "${workspaceFolder}/db_app",
        "args": ["INIT"],
        "preLaunchTask": "build",
```

19. Modify tasks.json file by adding `"build"` setting:

```
{
    "label": "build",
    "type": "shell",
    "command": "make",
    "args": [],
    "group": { "kind": "build", "isDefault": true }
}
```

20. Use menu Run > Start Debugging (F5) to activate gdb debugging environment.

Real-time multi-thread programming



POSIX programming

POSIX is shorthand for Portable Operating System Interface. It is an IEEE 1003.1 standard that defines the language interface between application programs (along with command line shells and utility interfaces) and the UNIX operating system. POSIX defines its standards in terms of the C language. The POSIX C API adds more functions on top of the ANSI C Standard for a number of aspects, such as file operations, processes/threads/shared memory/scheduling parameters, networking, memory management, and regular expressions.

The POSIX thread libraries are a C/C++ thread API based on standards. It enables the creation of a new concurrent process flow. It works well on multi-processor or multi-core systems, where the process flow may be scheduled to execute on another processor, increasing speed through parallel or distributed processing.

```c
#include <pthread.h>
void *thr_fnc( void *ptr );
int main() {
    pthread_t thr_handle;
    pthread_create(&thr_handle, NULL, thr_fnc, (void*)message);
    pthread_join(thr_handle, NULL);
    exit(0);
}
```

The threads library provides three thread synchronization mechanisms: mutex, join, and condition variable. Several IPC mechanisms are available for POSIX systems, including shared memory and message passing. POSIX message queues allow processes to exchange data in the form of messages. POSIX shared memory is organized using memory-mapped files, which associate the region of shared memory with a file.

https://pubs.opengroup.org/onlinepubs/9699919799/

https://www.baeldung.com/linux/

Practice #3: multi-thread programming

Procedure

1. Install required libraries: `sudo apt install libreadline-dev`
2. Modify thr_front.c to trigger when parsed integer from user input is ready.

```
pthread_mutex_lock(&data_cond_mutex);
pthread_cond_signal(&data_cond);
...
pthread_mutex_unlock(&data_cond_mutex);
```

3. Modify thr_mid.c to handle and trigger when average value is ready.

```
pthread_mutex_lock(&data_cond_mutex);
pthread_cond_wait(&data_cond, &data_cond_mutex);
...
pthread_mutex_lock(&avg_data_cond_mutex);
pthread_cond_signal(&avg_data_cond);
pthread_mutex_unlock(&avg_data_cond_mutex);
...
pthread_mutex_unlock(&data_cond_mutex);
```

4. Modify thr_end.c to display average value.

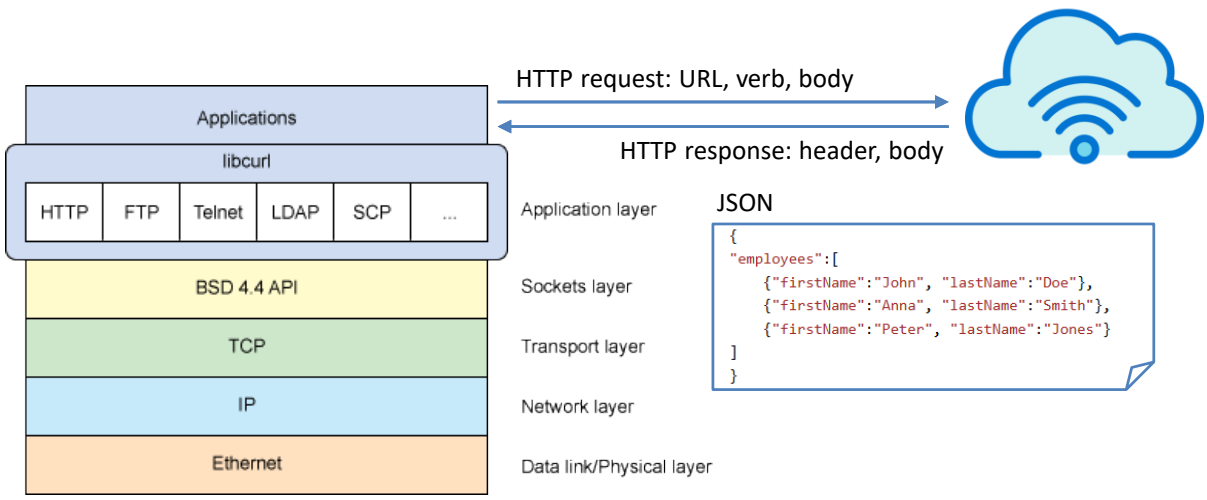
```
pthread_mutex_lock(&avg_data_cond_mutex);
pthread_cond_wait(&avg_data_cond, &avg_data_cond_mutex);
...
pthread_mutex_unlock(&avg_data_cond_mutex);
```

5. Modify the template Makefile to build from thr_app.c, front_thr.c, mid_thr.c, end_thr.c and libraries readline and pthread.

```
LIBS = -lm -lreadline -lpthread

TARGET = thr_app

SRCS = thr_app.c front_thr.c mid_thr.c end_thr.c
```

6. Build application with `make` command.
7. Study example code, then run `./thr_app` to process average values from user input.

Networking programming: cURL



Networking with cURL

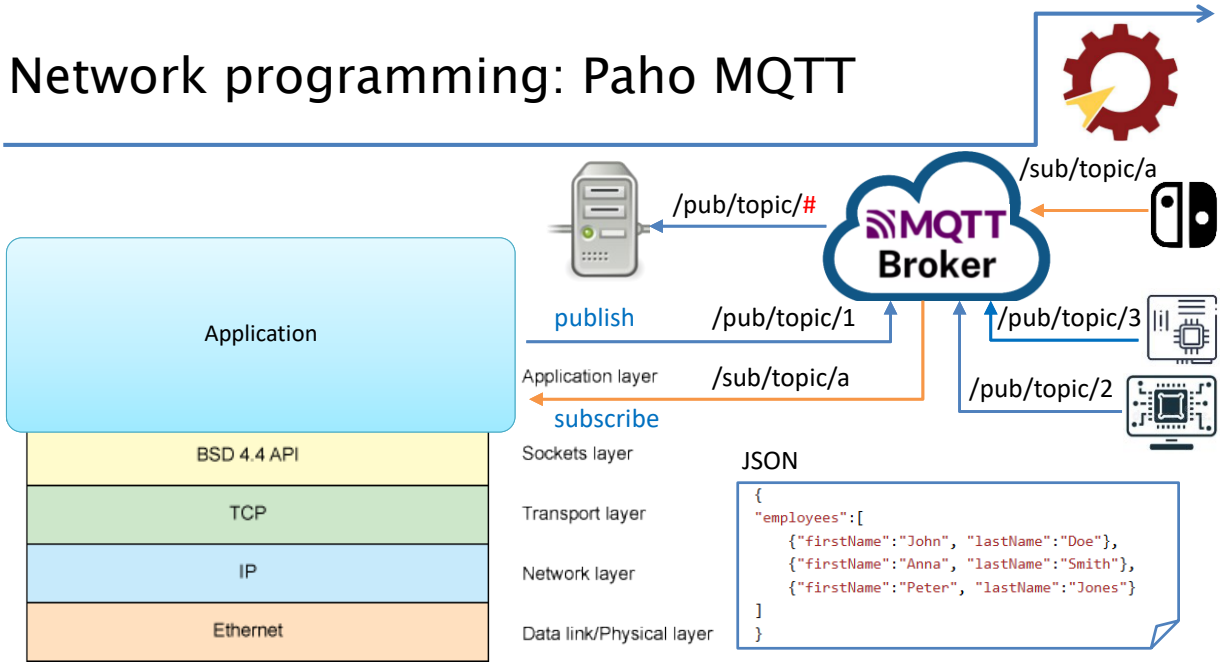
The cURL project is a number of loosely coupled individual volunteers working on writing software together with a common mission: to do reliable data transfers with Internet protocols, as Open Source.

HTTP is a protocol that is easy to learn the basics of. A client connects to a server—and it is always the client that takes the initiative—sends a request and receives a response. Both the request and the response consist of headers and a body. All transfers involving HTTP start with an HTTP request. An HTTP request sent by a client starts with a request line, followed by headers and then optionally a body. An HTTP response is a set of metadata and a response body, where the body can occasionally be zero bytes and thus nonexistent.

libcurl is a library of functions that are provided with a C API, for applications written in C. **libcurl** generally does the simple and basic transfer by default, and if you want to add more advanced features, you add that by setting the correct options. An HTTP request is what curl sends to the server when it tells the server what to do. When it wants to get data or send data.

https://everything.curl.dev/index.html

Network programming: Paho MQTT



Paho MQTT

MQTT is a lightweight publish/subscribe messaging transport for TCP/IP and connectionless protocols (such as UDP) respectively. MQTT clients are very small, require minimal resources so can be used on small microcontrollers. MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things.

An MQTT broker is an intermediary entity that enables MQTT clients to communicate. Functioning as a central hub, an MQTT broker receives messages published by clients, filters the messages by topic, and distributes them to subscribers. This broker-mediated communication enables a lightweight, scalable, and reliable mechanism for devices to share information in a networked environment, playing a crucial role in the establishment of efficient and responsive IoT ecosystems and other distributed applications.

The Eclipse Paho project provides open source, mainly client side, implementations of MQTT in a variety of programming languages. The Paho C client comprises four variant libraries, shared or static:

- libpaho-mqtt3a.so - asynchronous (MQTTAsync)
- libpaho-mqtt3as.so - asynchronous with SSL/TLS (MQTTAsync)
- libpaho-mqtt3c.so - "classic" / synchronous (MQTTClient)
- libpaho-mqtt3cs.so - "classic" / synchronous with SSL/TLS (MQTTClient)

https://eclipse.dev/paho/files/mqttdoc/MQTTClient/html/index.html

Practice #4: REST – MQTT endpoint

Procedure

1. Install required libraries:
   `sudo apt install libcurl4-openssl-dev libcjson-dev libpaho-mqtt-dev`
2. Use `curl` to test REST endpoint on Google Firebase:
   `curl https://tgr2024-demo-app-default-rtdb.firebaseio.com/data.json`
3. Test MQTT broker with MQTTX web client: https://mqttx.app/web-client#/
4. Modify **rest_thr.c** to trigger MQTT thread if new value from Firebase.

```
pthread_mutex_lock(&data_cond_mutex);
shared_value = value;
pthread_cond_signal(&data_cond);
pthread_mutex_unlock(&data_cond_mutex);
```

5. Modify **mqtt_thr.c** to publish MQTT message when new value is triggered.

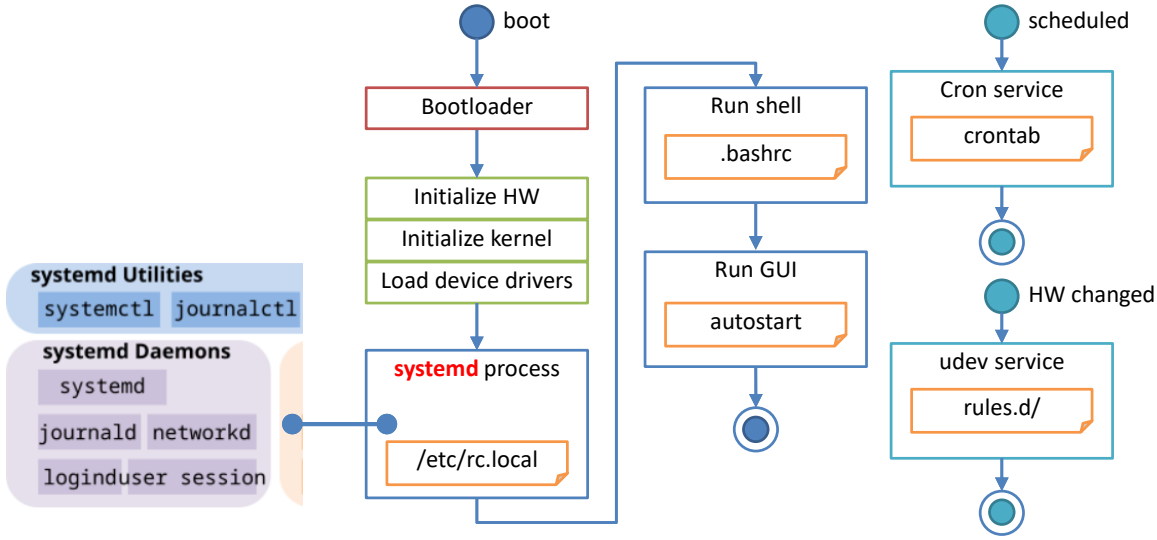
```
pthread_mutex_lock(&data_cond_mutex);
pthread_cond_wait(&data_cond, &data_cond_mutex);
value = shared_value;
pthread_mutex_unlock(&data_cond_mutex);
```

6. Modify the template **Makefile** to build from **iot_app.c**, **rest_thr.c**, **mqtt_thr.c** and libraries **curl** , **paho-mqtt3c**, **cjson** and **pthread**.

```
LIBS = -lm -lpthread -lcurl -lpaho-mqtt3c -lcjson

TARGET = iot_app

SRCS = iot_app.c rest_thr.c mqtt_thr.c
```

7. Build application with `make` command.
8. Study example code, then run `./iot_app` to forward data from Firebase to EMQX.

Automating Linux software



Linux automation

The Linux kernel boot process is a sequence of events that begins when a computer is powered on or restarted. This process involves several key stages, each responsible for initializing different parts of the system and ensuring that the operating system is loaded correctly.

Boot Loader Stage: The boot loader is a small program responsible for loading the operating system into memory.

Kernel Initialization Stage: Linux Kernel decompresses itself, performs hardware checks, gains access to vital peripherals, and starts the systemd process.

Systemd Stage: Systemd is the init system that manages system processes after the kernel is loaded. It mounts filesystems, initiates services, and manages user logins. Systemd uses the `/etc/systemd/system/default.target` file to determine the default run level or target state of the system.

`systemctl` is a command-line utility that interacts with the `systemd` system and service manager. It is used to control and manage `systemd` services, allowing users to start, stop, enable, disable, and check the status of services.

- To start a service: `sudo systemctl start app.service`
- To stop a running service: `sudo systemctl stop app.service`
- To restart a running service: `sudo systemctl restart app.service`
- To start a service at boot: `sudo systemctl enable app.service`
- To disable a service to start automatically: `sudo systemctl disable app.service`

Practice #5: automated RPi device

Procedure

1. Run `cat /proc/meminfo` to check memory status.
2. Modify dump_mem.c to append memory data to /home/tgr2024/mem.db.

```
const char db_name[] = "/home/tgr2024/mem.db";
int mem_free_size = get_mem_free();
dbase_init(db_name);
dbase_append(db_name, mem_free_size);
```

3. Modify the template Makefile to build from dump_mem.c, db_helper.c and libraries sqlite3.

```
LIBS = -lm -lsqlite3

TARGET = dump_mem

SRCS = dump_mem.c db_helper.c
```

4. Build application with `make` command.
5. Run `./dump_mem` to print and store memory free data in database.
6. Run `sqlite3 mem.db` to check data with SQL command: `SELECT * FROM data_table;` , then use `.quit` command to exit.
7. Run `crontab -e` to edit crontab file to run `dump_mem` every minute:

```
.---------------- minute (0 - 59)
| .------------- hour (0 - 23)
| |  .---------- day of month (1 - 31)
| |  | .-------- month (1 - 12)
| |  | | .------ day of week (0 - 6) (Sunday=0 or 7)
| |  | | |           command to be executed
* *  * * *           /home/tgr2024/ex_05/dump_mem
```

8. Run `sqlite3 mem.db` to check data with SQL command: `SELECT * FROM data_table;`