Contents

[**I.** **Programming skill** 3](#_Toc26121724)

[**1.** **Multithread** 3](#_Toc26121725)

[**2.** **Data alignment** 3](#_Toc26121726)

[**3.** **Memory Layout** 3](#_Toc26121727)

[**4.** **Mutex vs Semaphore** 3](#_Toc26121728)

[**5.** **Dynamic Programming** 3](#_Toc26121729)

[**6.** **Knapsack** 3](#_Toc26121730)

[**7.** **Ranking problem** 3](#_Toc26121731)

[**8.** **Recursion** 3](#_Toc26121732)

[**9.** **Linkage** 3](#_Toc26121733)

[**II.** **Embedded Knowledge** 4](#_Toc26121734)

[**1.** **Embedded system architecture** 4](#_Toc26121735)

[**2.** **Hardware component** 4](#_Toc26121736)

[**3.** **Register:** 4](#_Toc26121737)

[**4.** **Unix OS architecture:** 5](#_Toc26121738)

[**5.** **Linux kernel:** 5](#_Toc26121739)

[**6.** **Bootloader:** 6](#_Toc26121740)

[**7.** **Yocto project** 6](#_Toc26121741)

[**8.** **Middleware** 6](#_Toc26121742)

[**9.** **Driver** 6](#_Toc26121743)

[**10.** **Makefile and make** 6](#_Toc26121744)

[**11.** **MCU peripherals:** 7](#_Toc26121745)

[**12.** **Interrupt** 7](#_Toc26121746)

[**13.** **Firmware** 7](#_Toc26121747)

[**14.** **SoC vs MCU** 7](#_Toc26121748)

[**15.** **Virtualization** 7](#_Toc26121749)

[**16.** **Platform:** 7](#_Toc26121750)

[**17.** **Design Patern** 7](#_Toc26121751)

[**III.** **Programming Knowledge** 8](#_Toc26121752)

[**1.** **Object Oriented Programming (OOP trait)** 8](#_Toc26121753)

[**1.1.** **Inheritance**: 8](#_Toc26121754)

[**1.2.** **Polymorphism**: 8](#_Toc26121755)

[**1.3.** **Atraction**: 8](#_Toc26121756)

[**1.4.** **Encapsulation**: 8](#_Toc26121757)

[**2.** **Static:** 10](#_Toc26121758)

[**2.1.** Variable: 10](#_Toc26121759)

[**2.2.** Usage 10](#_Toc26121760)

[**3.** **String** 10](#_Toc26121761)

[**4.** **Pointer vs Reference:** 10](#_Toc26121762)

[**5.** **Array** 10](#_Toc26121763)

[**6.** **Exception** 11](#_Toc26121764)

[**7.** **Standard Template Library (STL)** 12](#_Toc26121765)

[***7.1.*** Algogithms 12](#_Toc26121766)

[***7.2.*** Containers 12](#_Toc26121767)

[7.2.1. Vector 12](#_Toc26121768)

[7.2.2. *Pair* 12](#_Toc26121769)

[7.2.3. *List* 12](#_Toc26121770)

[7.2.4. *Map* 12](#_Toc26121771)

[7.2.5. *Link List* 12](#_Toc26121772)

[**7.3.** Functions 12](#_Toc26121773)

[**7.4.** Iterator 12](#_Toc26121774)

[**8.** **Namespace** 12](#_Toc26121775)

[**9.** **Template** 12](#_Toc26121776)

Knowledge Systematic

# **Programming skill**

## **Multithread**

*- Refer:* <https://medium.com/@yutafujii_59175/multi-thread-and-multi-process-5559ea5b19ba>

* A process is an instance of a computer program containing binary code along with the resources above.
* A thread is a component of a process. It is an execution unit and it contains program counter, stack and set of registers.

## **Data alignment**

## **Memory Layout** (memory of a process)

* Code segment: Store only code in ROM
* Heap segment: Store dynamic allocation: malloc, calloc, realloce, free, new and delete.
* Data segment: Global & local variables were initialised.
* BSS (uinitialised segment): Global & local variables are not initialised.
* Stack segment: Local, tempory return address.

## **Mutex vs Semaphore**

## **Dynamic Programming**

## **Knapsack**

## **Ranking problem**

## **Recursion**

* Concept: The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
* Type:

+ A direct recursive function: it calls the ITSELF inside.

+ A indirect recursive function: it calls another function say fun\_new and fun\_new calls IT directly or indirectly.

* Usage: provides a clean and simple way to write code.
* Disavantage:

+ recursive and iterative programs have the same problem-solving powers.

+ Recursive program has greater space, time requirements than iterative.

## **Linkage**

# **Embedded Knowledge**

What is Embedded system?

* <https://vi.wikipedia.org/wiki/H%E1%BB%87_th%E1%BB%91ng_nh%C3%BAng>

## **Káº¿t quáº£ hÃ¬nh áº£nh cho what is middleware eetimeEmbedded system architecture**

* Application layer.
* Software system layer:

+ OS(+middleware)

+ driver

* Hardware layer.

Figure 1: Embedded system architecture

## **Hardware component**

* CPU: (Central processing unit) process data and command due to control system.
* RAM: Store data and command.
* I/O modules: connect device, send data signal to CPU and recive control signal.
* Bus system: Connect RAM, CPU, I/O modules together.

+ Bus data: express data among RAM-CPU-I/O and command RAM->CPU-I/O.

+ Bus address:

+ Bus control: express control signal from CPU-> RAM, I/O; notification signal RAM, I/O -> CPU.

## **Register:**

* Thanh ghi là đang nói đến cấu trúc phần cứng.
* Trên con vi xử lý, vi điều khiển…có một vùng nhớ đặc biệt. Trong vùng nhớ này mỗi bit, byte có giá trị ảnh hưởng trực tiếp đến hoạt động của chip hoặc chip sẽ dùng các byte đó trong tính toán. Tập hợp của một hay nhiều byte có cùng chức năng gọi là một thanh ghi.
* Nhà sản xuất sẽ phân biệt các thanh ghi với nhau bằng cách đặt tên cho nó. Do nó là cấu tạo của phần cứng nên số lượng là cố định và sẽ không có khái niệm hệ điều hành cấp thanh ghi cho ứng dụng.

## **Virtual memory**

* Avoid crash.
* Manage effectivwly physical memory: Mapping flexible.
* Protected stored data, avoid overiding (RAM is random access memory).

## D:\E\01_Programming\Linux\Anatory of the Linux Kernel.jpg**Unix OS architecture:**

* Refer: https://www.guru99.com/operating-system-tutorial.html
* Usage: Manage HW resource and provide HW usage services.
* Component:

+ Kernel

+ System call ( Các API được cung cấp bởi Kernel)

+ Libraries

+ Application

Figure 2: Architecture of the UNIX operating system

## **Linux kernel:**

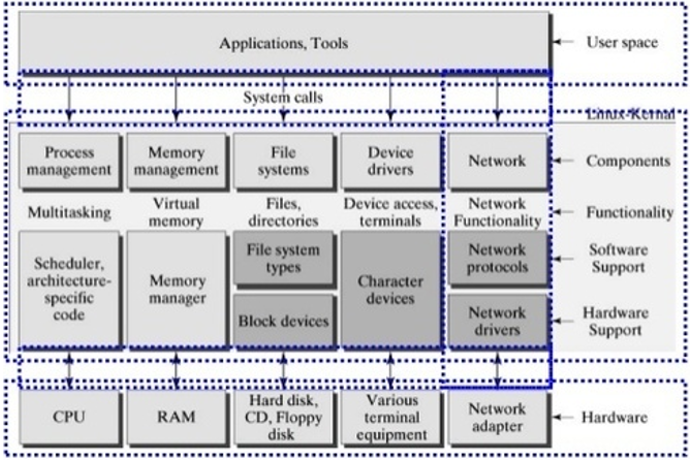
* Refer: <https://vimentor.com/vi/lesson/gioi-thieu-ve-linux-kernel-1>
* Main part of OS.
* Configured to run on a particular board (may be part of a BSP supplied by a hardware vendor).

Figure 3: Linux kernel Structure

## **Bootloader:**

* Refer: <https://www.sciencedirect.com/topics/engineering/bootloader>
* Usage: Main task is loading kernel to RAM.

(BIOS itself is copied to RAM, CPU reads introductions from memory and executes them.)

* Booting processing:
* Step 1: set up and initialize the RAM;
* Step 2: initialize one serial port (optional);
* Step 3: detect the machine type;
* Step 4: set up the kernel tagged list;
* Step 5: call the kernel image.

## **Yocto project**

* Refer: <https://lazytrick.wordpress.com/2017/01/20/so-sanh-buildroot-va-yocto-project/>
* An open source collaboration project.
* Provides templates, tools and methods to help you create custom Linux-based systems for embedded and IOT products, regardless of the hardware architecture.
* BitBake translates "recipe" files -> make a Sequential task queue.
* Poky is a reference distribution of the Yocto Project

Involves: OpenEmbedded Build System (BitBake + OpenEmbedded-Core) and a set of metadata (metadata includes: recipes, config files…)

## **Middleware**

* Refer: [https://www.eetimes.com/document.asp?doc\_id=1276764#](https://www.eetimes.com/document.asp?doc_id=1276764)
* It is any system software that is not the OS kernel, device drivers, or application software that connects software components or applications

## **Driver**

* Device Drivers are only written in C.
* Device driver presents a generic interface to the control and status registers for applications at higher level to access the devices.

## **Makefile and make**

* Refer: http://clinuxcode.blogspot.com/2017/03/make-and-make-file-interview-questions.html

## **MCU peripherals:**

* 1. ***Watchdog***
* A hardware timer used to detect and recover microcomputer malfuntions (due to noise, bugs, etc...)
* During normal operation, the microcomputer regularly restarts the watchdog timerto prevent it from elapsing or “timing out”. If there is hardware fault or program fault and the microcomputer cannot resart the watchdog timer. This timer will elapse and generate an Interrupt signal.
* 2 Types:
  + Build inside microcontroller.
  + Made out of microcontroller.
  1. ***Serial tranfer***
* It is a technique of sending data ono-by-one bit through a data line at each predetermined time.
* 2 Methods:
  + Clock Synchronous: have 2 line connect to another:
    - Control line: inform tranfer timing for each bit.
    - Data line: used to send data.
      * Data is transmitted in synchronization with the *Control signal* (Clock signal).
  + Clock Asynchronous: Only 1 line(Don’t need *Signal line)*
    - Data line: Sending and receiving sides are matched in data format and transfer rate.
      * When no data sent out, *Data line* set to “H”, so Receiving side can detect the start bit “L”.
* **Note:** 1 line can include serveral wire-signal.
  + 1. ***UART***
* Universial Asynchronous Receiver Transmitter.
* Used to communicate to modules: Zigbee, Bluetooth, Wifi…
* Some Concept:
  + Baud rate: Số bit truyền được trong 1s (tốc độ truyền dữ liệu đo bằng đơn vị Baud).
  + …
* Refer: <https://tapit.vn/luoc-ly-thuyet-ve-chuc-nang-uart-va-mot-thanh-ghi-trong-chip-stm32f103c8t6/>
  + 1. ***SPI***
* Serial Peripheral Interface is synchronous serial comunication interface. Used for short distance, primarily in embedded systems.
* 4-Wires serial:
  + SCLK (serial clock): determine the speed of transfer.
  + MOSI (Master Output – Slave Input).
  + MISO (Master Input – Slave Output).
  + SS (Slave Select): active low.
    - * SPI is full-duplex (both directions). Used in typical application: sensors, memory card, display control.
    1. ***I2C (IIC)***
* Inter-Intergrated Circuit is a synchonous serial interface.
* 2-Wires of BUS:
  + SDA (Serial data line).
  + SCL (Serial clock line).
    - * Each slave device connect to the BUS has a unique address (Master device will use it to indentify the slave devices)
  1. ***TCP/IP***
* Transmission Control Protocol/ Internet Protocol.
* là một bộ giao thức trao đổi thông tin được sử dụng để truyền tải và kết nối các thiết bị trong mạng Internet.
* **Ethernet** là một dạng công nghệ truyền thống dùng để kết nối các mạng LAN cục bộ, cho phép các thiết bị có thể giao tiếp với nhau thông qua một giao thức - một bộ quy tắc hoặc ngôn ngữ mạng chung. *Là một lớp giao thức data-link trong tầng TCP/IP.*
* Refer: <https://vnpro.vn/tin-tuc/gioi-thieu-tong-quan-ve-bo-giao-thuc-tcpip-1103.html>
  1. *USB*
  2. *Socket*

## **Interrupt**

## **Firmware**

* Refer: <https://voh.com.vn/cong-nghe/firmware-la-gi-co-nen-nang-cap-firmware-hay-khong-319479.html>
* Firmware is a software program or set of introductions programmed on hardware device that makes the hardware function according to the manufacturer’s intended purposes.
* It provides the mecessary introductions for how the device communicates with the other computer hardware.
* Stored on ROM or PROM of a device. Can be erased and rewritten
* Device has one or more processor, and firmware is executed on those.

\* Note: Firmware vs Driver: <https://www.cybrary.it/2018/08/what-is-firmware/>

- Firmware is installed in the hardware of a device,

- Driver is stored inside of the operating system.

## **SoC vs MCU**

* A MCU *(microcontroller unit)* consists of a CPU, ROM, RAM, and basic peripherals etc. To integrate a micrcontroller to a system one might have to add a Wi-Fi module or some GPU chip.
* A SoC *(System-on-Chip)* will contain a microcontroller and some advanced peripherals like bluetooth, WiFi, so that the whole system is on that chip.

## **Virtualization**

## **Platform:**

* Platform chính là nền tảng. Đây chính là nên tảng công nghệ sử dụng.
* Là nền tảng, cốt lõi quan trọng để những lập trình viên dựa vào đó thực hiện công việc
  + Hardware platform
  + Software platform

## **Design Patern**

# **Programming Knowledge**

## **Object Oriented Programming (OOP trait)**

## **Inheritance**:

* Inherit member and method from base class.
* 2 types:
  + ***Multiple level Inheritance:* B inherit A Then C inherit B → C can access ALL in A and B.**
  + ***Multiple Inheritance* (Only in C++): Class C: public B, public A;**

## **Polymorphism**:

* 1 method = A lot of usages.
* 2 types:
* *Overloading*: Same: *Name and implement*, but different: *Return type* or *number or type of parameters*.
  + **Operator overloading**
  + **Method overloading**
* *Overriding***:** Same: *Name and parameter*, but different *implement*.
  + **virtual**: keyword to declare a virtual method (can be overided).
  + ***pure virtual function:*** *virtual method\_name () =0;*

ERROR if not define in Child class)

* + Referred to as dynamic linkage, or late binding.

## **Atraction**:

* Refer: <https://yellowcodebooks.com/2017/09/26/java-bai-32-tinh-truu-tuong-abstraction/>
* Concept use in design phase:
  + - * Bóc tách **các** đối tượng thực tế
      * Tìm được những data&method **chung**
      * **Gom** chúng lại, tạo thành Attract-class.
* Attract class trait:
  + KO ý nghĩa hoặc KO thể dùng tạo object.
  + Có ít nhất 1 Attract-method inside.
  + Đặc trưng cho nhóm đối tượng đó.
* Purpose
  + Only **khung sườn** cho các class con.
  + Optimize code, tạo tính liên kết.

## **Encapsulation**:

* Data hiding
* **Prevent** access directly member (private), must through method.
* MIX (Inheritance + Polymorphism(virtual) + static\_member)

|  |  |
| --- | --- |
| Asume | *Class Parent{};*  *Class Child : public Parent {};* |
| Type 1 | Access Child directly:  *Child new derive();* // Parent(constructor) -> Child(constructor).  ….  *delete(derive);*  // Child(de-constructor) -> Parent(de-constructor).  *Or [Out of scope]*  Content:  - *"derive”* Obj can access to *ANYTHING* within both (*Parent* & *Child* class). |
| Type 2 | Access Child by Parent pointer:  *Parent\* Base;*  *Base = new derive(); | Parent(constructor) -> Child(constructor).*  *….*  *delete Base; (freed memory)*  *Base = NULL; (pointed dangling ptr to NULL) Only Parent(de-constructor).*  *Or [Out of scope]*  Content:  - "Base pointer" Only access to "Parent\_method".  - Add "virtual" in front of "that Parent-method" --> Access to Overided\_in\_Child. |
| Type 3 | Child access permission:   |  |  |  |  | | --- | --- | --- | --- | | Access | public | protected | private | | Same class | O | O | O | | Derived classes | O | O | X | | Outside classes | O | X | X |     A derived class inherits all base class methods with the following exceptions:  - Constructors, destructors and copy constructors of the base class.  - Overloaded operators of the base class.  - The friend functions of the base class. |
| Type 4 |  |

# **Static:**

## Variable:

* Static Global: như nhau, Nhưng ko extern đc từ File khác.
* Static in function: vùng nhớ đều ko hủy khi thoát khỏi hàm, Nhưng static var chỉ hoạt động trong function của nó.

## Usage

* Nó là hàm local trong file nó đc khai báo, KHÔNG dùng được cho file khác.

# **String**

* String thực chất là một vector<char> có bổ sung thêm một số hàm và thuộc tính,

do đó, nó có toàn bộ các tính chất của 1 vector, như hàm size(), push\_back(), toán tử [], ...

# **Pointer vs Reference:**

* 1. General Features*:*
* Pointer: Can be NULL and can be re-assigned many times.
* Reference: Always refers to an object, CANNOT be re-assigned after binding.
* There's no "reference arithmetic" (but you can take the address of an object pointed by a reference and do pointer arithmetic on it as in &obj + 5).
  1. Usages:
* Use references in function parameters and return types to provide useful and self-documenting interfaces.
* Use pointers for implementing algorithms and data structures.

# **Array**

# **Exception**

* Keywords: try - throw – catch
* throw:

+ built-in C++ standard (std::exception)

+ defined-in yourself

* catch(ex\_type &e): 3 type
  + Expexcted thow type ⬄ (standard or new-define).
  + *Std::exception* ⬄ (All standard)
  + *(...)* ⬄ (anything)

|  |
| --- |
| New-define Example   * Inherit:   *$ struct MyException : public exception {*  *$ const char \* what () const throw () {*  *$ return "C++ Exception";*  *$ }*  *$ };*   * Alsolute new:   *$ struct MyException {*  *$ const char \* what (){*  *$ return "C++ Exception";*  *$ }*  *$ };* |

# **Standard Template Library (STL)**

## Algogithms

* Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.

## Containers

* Used to manage collections of objects of a certain kind.

### Vector

### *Pair*

### *List*

### *Map*

### *Link List*

## Functions

## Iterator

* Used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

# **Namespace**

* Phân vùng source code, giải quyết những trường hợp cùng tên hàm nhưng khác nội dung, ý nghĩa.

# **Template**

* Template” là từ khóa báo cho trình biên dịch rằng đoạn mã sau đây định nghĩa cho nhiều kiểu dữ liệu và mã nguồn của nó sẽ được compile sinh ra tương ứng cho từng kiểu dữ liệu trong quá trình biên dịch.