

LẬP TRÌNH HỆ NHÚNG

GV: Phạm Ngọc Hưng

Bộ môn Kỹ thuật Máy tính

Viện CNTT&TT- ĐH BKHN

email: hungpn@soict.hut.edu.vn



Nội dung

Chương 1. Giới thiệu lập trình hệ nhúng

Chương 2. Lập trình vào ra cơ bản

Chương 3. Lập trình vào ra nâng cao

Chương 4. Các kỹ thuật lập trình nâng cao

Chương 5. Lập trình device driver trên Linux

Chương 6. Lập trình nền tảng QT

Chương 7. Lập trình mạng trên Linux nhúng

Chương 8. Lập trình xử lý ảnh trên nền nhúng



Tài liệu tham khảo

- **Tài liệu tham khảo chính:**

- Micro2440 User Manual
- S3C2440 MicroController User's Manual
- Beginning Linux Programming
- Advanced Linux Programming
- Linux Device Driver
- C++ GUI programming with QT
- Website:

<https://sites.google.com/site/embedded247/>



Chương 1

Giới thiệu Lập trình hệ nhúng



Nội dung chương 1

- 1.1. Giới thiệu về lập trình hệ nhúng
- 1.2. Giới thiệu KIT FriendlyArm micro2440
- 1.3. Hệ điều hành nhúng Linux
- 1.4. Môi trường lập trình KIT FriendlyArm 2440



1.1. Giới thiệu lập trình hệ nhúng

- *Lập trình ứng dụng trên hệ nhúng phụ thuộc vào nền tảng (platform) phần cứng, phần mềm của hệ nhúng đó.*
- **Hệ nhúng không có hệ điều hành:**
 - Thường sử dụng các vi điều khiển hiệu năng tương đối thấp (8051, ATMega, PIC, ARM7, ...)
 - Lập trình bằng C, ASM
 - Môi trường, công cụ lập trình tùy theo từng dòng vi điều khiển (CodeVision, AVR Studio, Keil...)
 - Phù hợp các ứng dụng điều khiển vào/ra cơ bản, các giao tiếp ngoại vi cơ bản.





1.1. Giới thiệu lập trình hệ nhúng

▪ **Hệ nhúng có hệ điều hành:**

- Dựa trên các vi điều khiển, vi xử lý (CPU) có hiệu năng cao (Ví dụ: AVR 32, ARM 9, ARM 11, ...)
- Nhiều nền tảng hệ điều hành nhúng : uCLinux, **Embedded Linux**, Windows CE, ...
- Môi trường, công cụ lập trình tùy thuộc nền tảng hệ điều hành: **C/C++, QT SDK** (Nokia), .Net Compact FrameWork (Microsoft), ...
- Ứng dụng nhiều bài toán phức tạp: GPS Tracking/Navigator, Xử lý ảnh, ứng dụng Client/Server, ...



1.1. Giới thiệu lập trình hệ nhúng

- **Các thiết bị di động thông minh:**

- Xu hướng công nghệ hiện nay
- Nhiều nền tảng: iOS, Android, Windows Phone, Symbian OS/Memo,
- Môi trường, công cụ:
 - ✓ iOS: Xcode + iOS SDK (ngôn ngữ Object-C)
 - ✓ Android: C, Java + Android SDK, Eclipse/Netbean
 - ✓ Windows Phone: SDK + Visual Studio (C#)
- Các ứng dụng phong phú: Google Play Store, Apple Store, Windows Market Place, ...



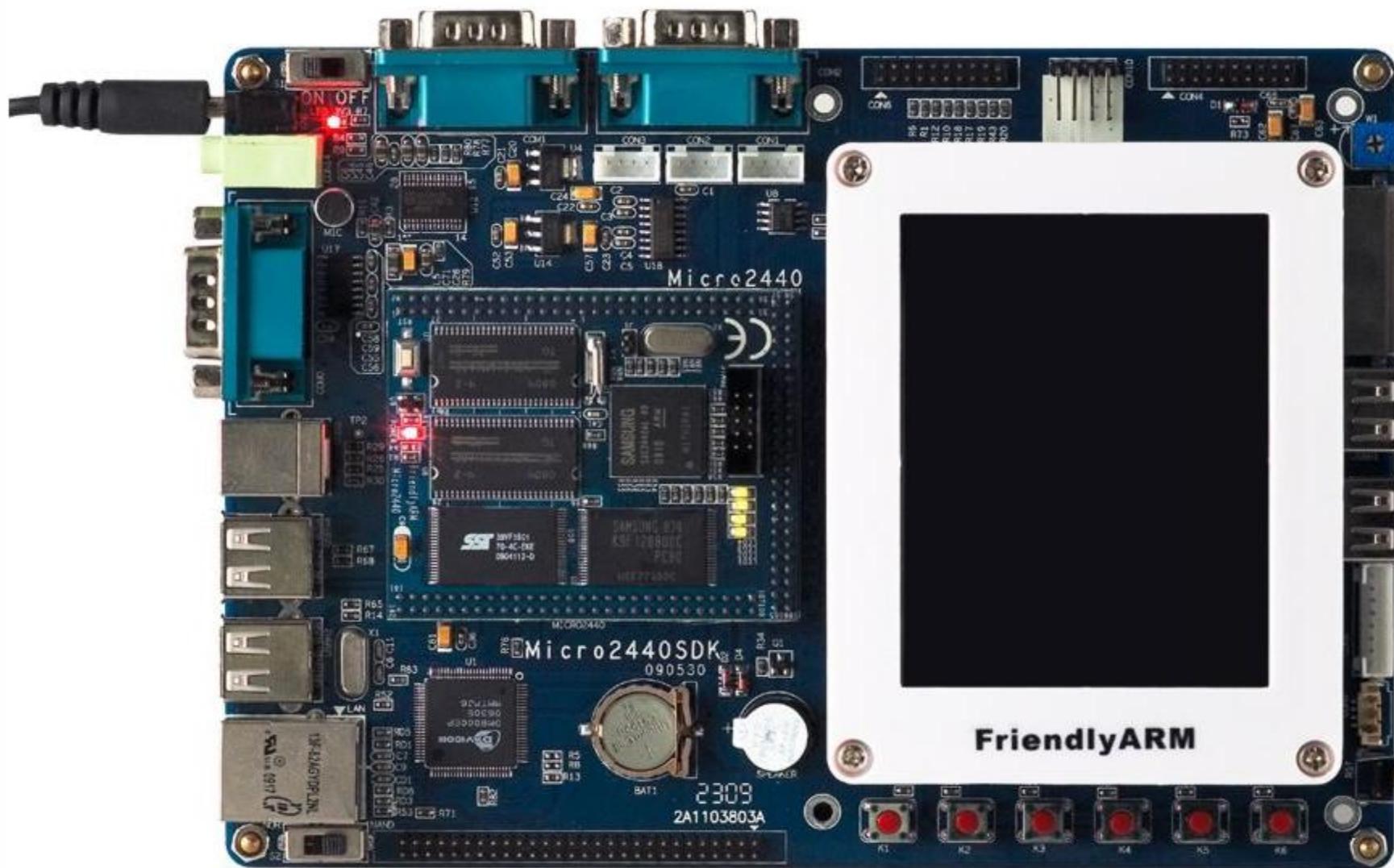
1.1. Giới thiệu lập trình hệ nhúng

- Môn học này hướng tới:
 - Lập trình hệ nhúng nền tảng **ARM + Linux**
 - Minh họa trên KIT **FriendlyArm micro 2440**
 - Lập trình **C/C++**, lập trình giao diện đồ họa **QT**
- Lý do:
 - **ARM ?** > 90% thị phần thiết bị nhúng,
là dòng vi điều khiển hiệu năng cao.
 - **Embedded Linux ?** Mã nguồn mở, khả năng can thiệp, hiểu
sâu hệ thống. Nhiều OS khác (iOS, Android) dựa trên Linux
kernel



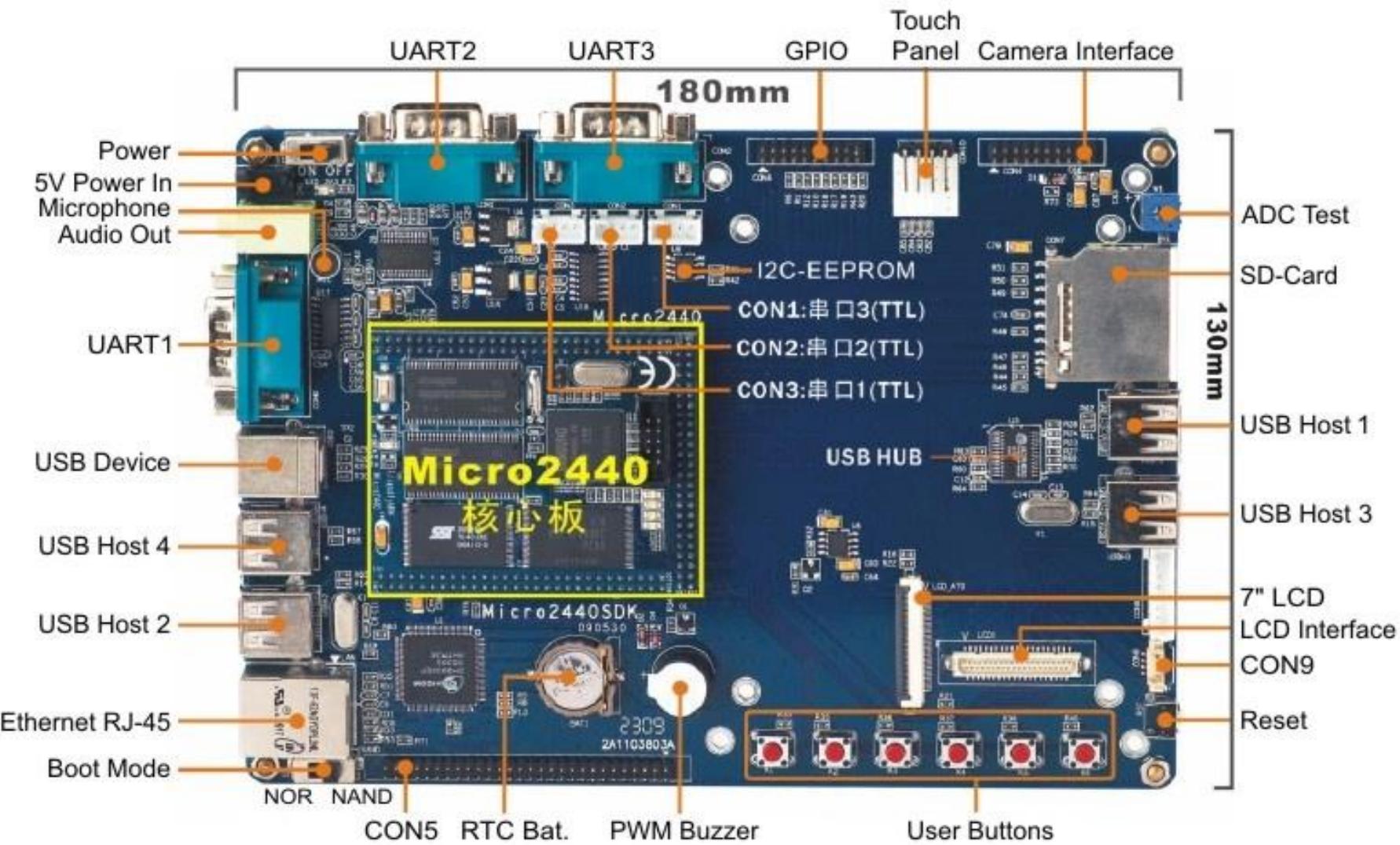


1.2. Giới thiệu KIT nhúng micro2440





Giới thiệu KIT nhúng micro2440





Giới thiệu KIT nhúng micro2440





Giới thiệu KIT nhúng micro2440

■ Thông số kỹ thuật

Khối chức năng	Thông số kỹ thuật
CPU	Samsung S3C2440A, 400MHz, Max. 533Mhz
SDRAM	<ul style="list-style-type: none">- 64M SDRAM- 32bit DataBus- SDRAM Clock 100MHz
Flash	<ul style="list-style-type: none">- 64M or 128M Nand Flash,- 2M Nor Flash (đã được cài đặt sẵn BIOS)
Màn hình LCD	<ul style="list-style-type: none">- Màn hình cảm ứng- Tối đa 4096 màu STN, kích thước từ 3.5 inches tới



Giới thiệu KIT nhúng micro2440

Các thiết bị ngoại vi	<ul style="list-style-type: none">- 1 khối 10/100M Ethernet RJ-45(DM9000)- 3 Serial Port- 1 USB Host- 1 USB Slave Type B- 1 giao tiếp SD Card- 1 Stereo Audio out, 1 Micro In- 1 20-Pin JTAG (kết nối với mạch nạp, debug)- 4 đèn LED đơn- 6 nút bấm- 1 còi điều khiển sử dụng PWM- 1 biến trở sử dụng kiểm tra bộ chuyển đổi số/tương tự (A/D converter)- 1 EEPROM giao tiếp theo chuẩn I²C- 1 giao tiếp với cảm biến ánh (20-chân)- 1 pin cho đồng hồ thời gian thực- Nguồn 5V
Các hệ điều hành được hỗ trợ	<ul style="list-style-type: none">- Linux 2.6- Android- WinCE 5 and 6



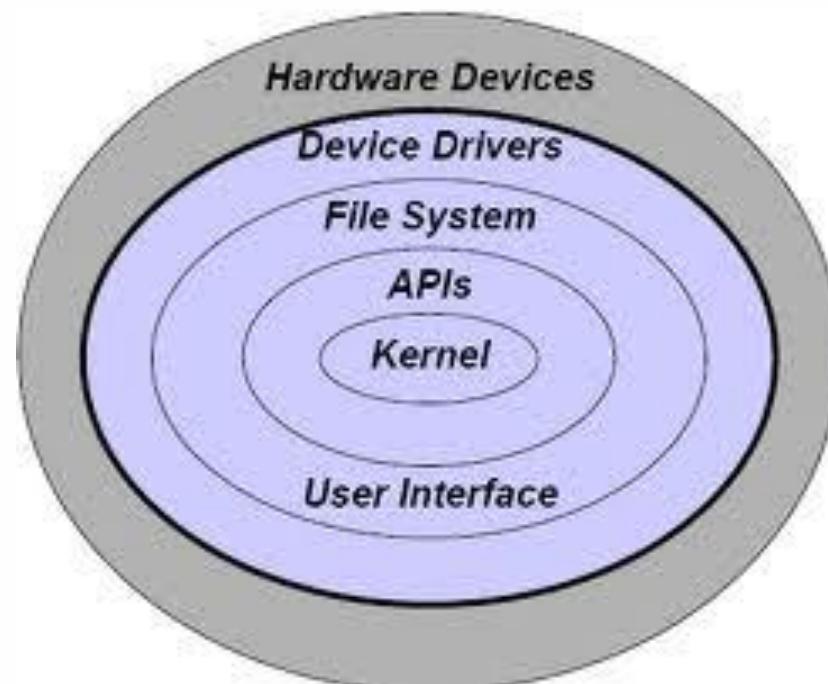
1.3. Hệ điều hành nhúng Linux

- 1.3.1. Tổng quan hệ điều hành nhúng Linux
- 1.3.2. Cài đặt Embedded Linux trên Micro2440
- 1.3.3. Biên dịch, tùy biến nhân Linux



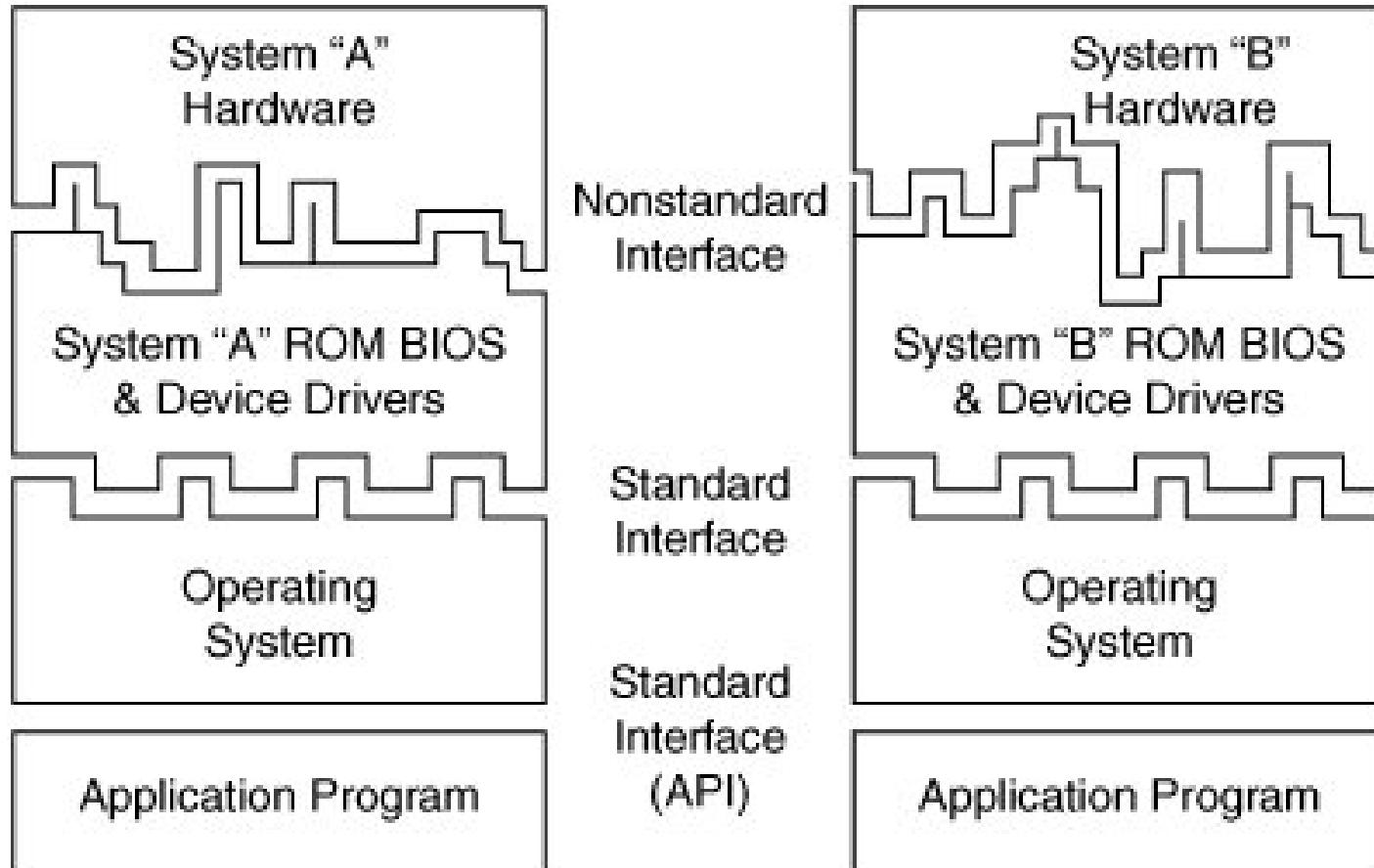
1.3.1. Tổng quan Embedded Linux

- Hệ điều hành nhúng (embedded os) ?
 - Là hệ điều hành cài đặt cho các hệ thống nhúng (embedded system)
 - Được thiết kế: compact, efficient, reliable.



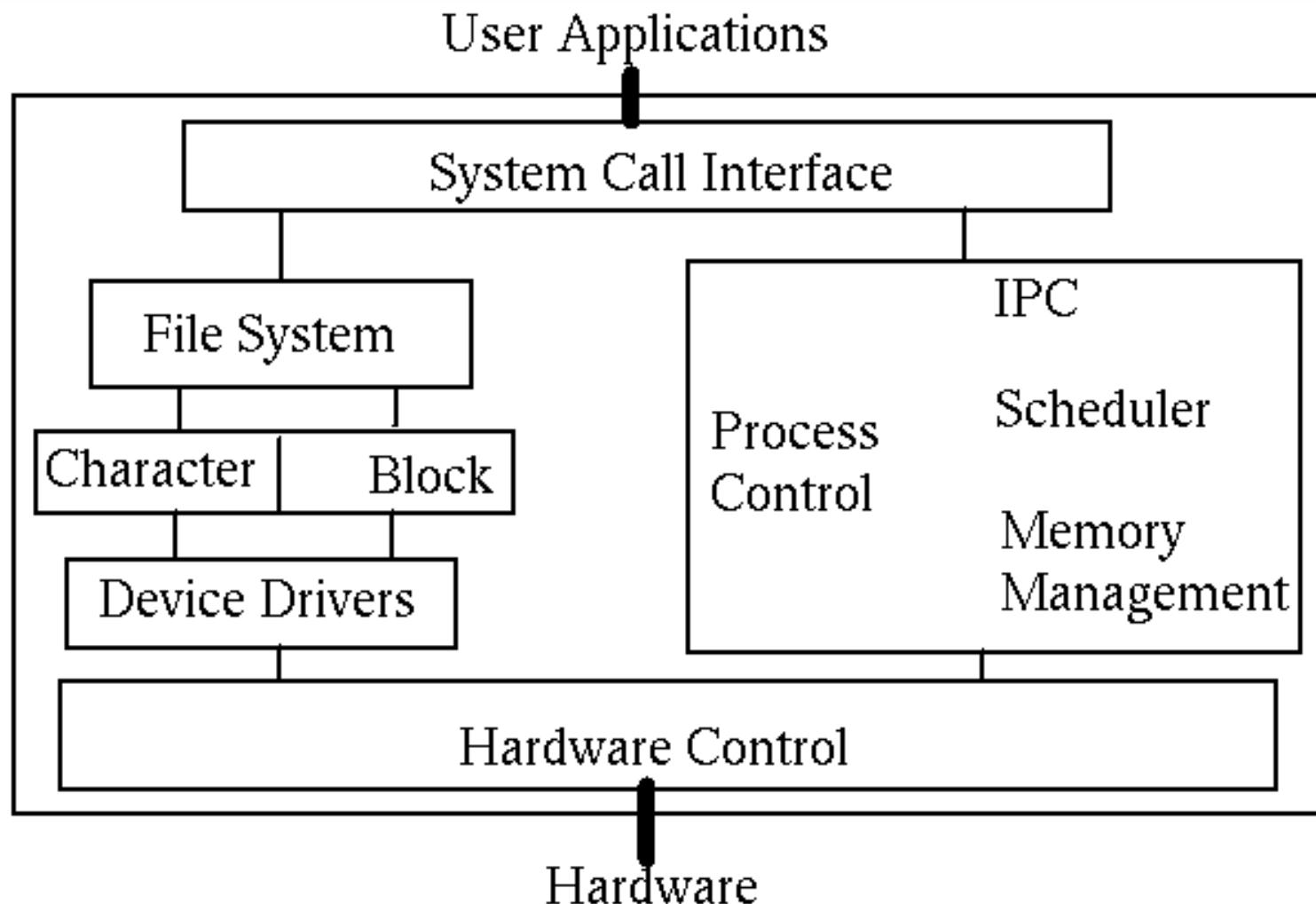


Sơ đồ phân cấp hệ thống





Kiến trúc hệ điều hành Linux





Đặc trưng hệ điều hành nhúng

- Tăng tính tin cậy (reliability)
- Tăng tính khả chuyển (portability)
- Khả năng tương thích mềm: dễ dàng nâng cấp hay thu gọn để tương thích với nền tảng hệ thống
- Thu gọn, đòi hỏi ít bộ nhớ hơn. Có thể hỗ trợ khởi động từ bộ nhớ ROM, Flash (hệ thống không có ổ cứng)
- Cung cấp các cơ chế lập lịch (scheduler) hỗ trợ thời gian thực (Realtime OS – RTOS)

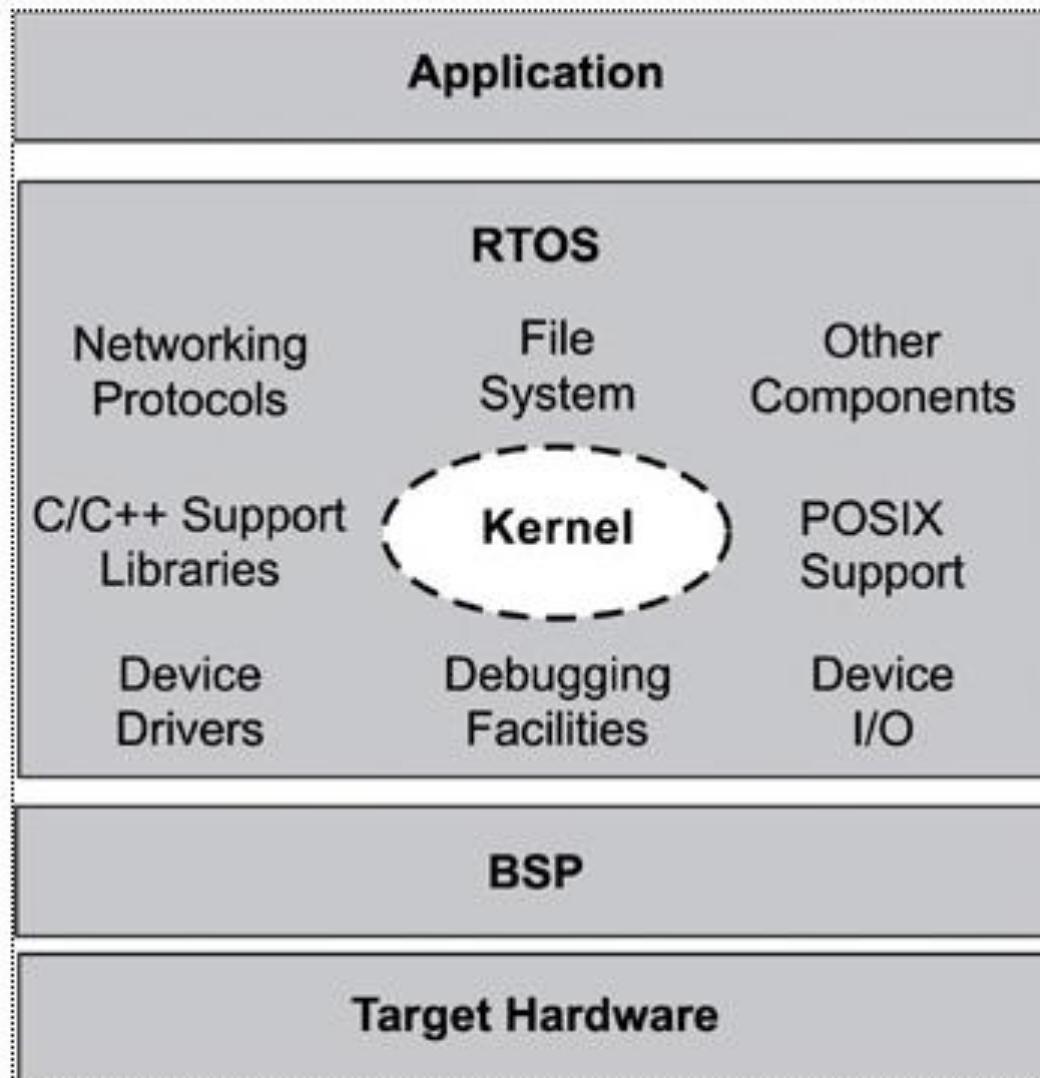


Hệ điều hành thời gian thực

- Hệ thống thời gian thực (Realtime): các phần mềm, phần cứng hoạt động thỏa mãn các ràng buộc về thời gian
- Phân loại:
 - Hard Realtime: không đáp ứng deadline -> lỗi hệ thống
 - Soft Realtime: không đáp ứng deadline -> giảm chất lượng dịch vụ (QoS)

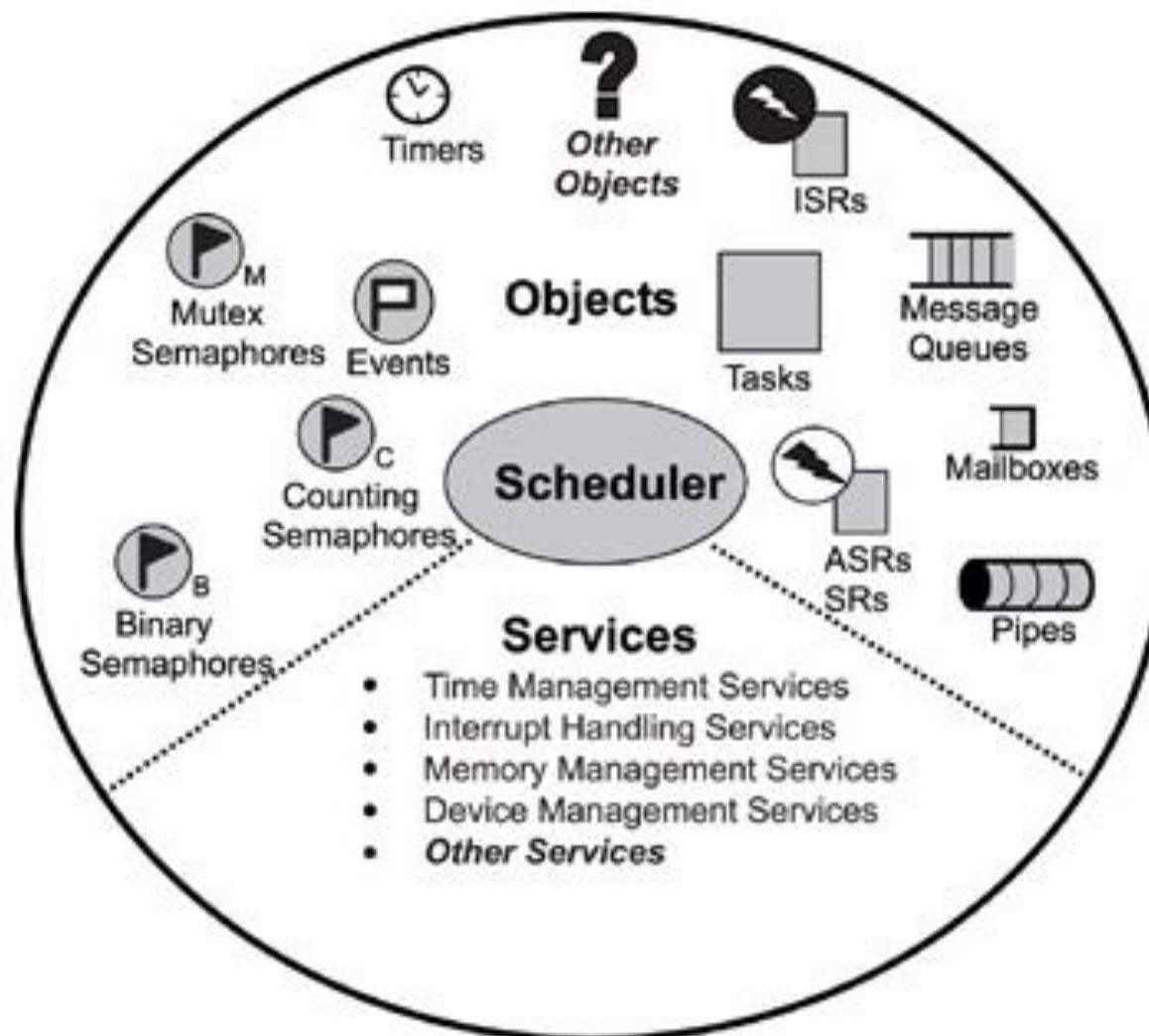


Hệ điều hành thời gian thực





Cấu trúc nhân hệ điều hành





Hệ thống file trong Linux

▪ Một số thư mục quan trọng

- /home: thư mục người dùng
- /dev: chứa các file thiết bị
- /bin: chứa các file thực thi của hệ thống
- /etc: chứa các file cấu hình
- /var: chứa các file log
- /opt: chứa các gói chương trình cài đặt thêm
- /proc: chứa thông tin về các tiến trình, các thành phần phần cứng, phần mềm đang chạy trong hệ thống
- /usr: chứa các file thực thi, tài liệu liên quan tới người dùng



- Hỗ trợ rất nhiều kiến trúc vi xử lý (cả 32 bit và 64 bit)
 - Intel X86, ARM, PowerPC, MIPS, AVR32, ...
- Không hỗ trợ các vi điều khiển hiệu năng thấp
- Hỗ trợ cả kiến trúc có và không có khối quản lý bộ nhớ (MMU)
- Các hệ thống có thể dùng chung toolchains, bootloader và kernel, các thành phần khác phải riêng biệt và tương thích với từng hệ thống



Quá trình boot hệ thống Linux trên PC

Power-up / Reset

System startup

BIOS / BootMonitor

Stage 1 bootloader

Master Boot Record

Stage 2 bootloader

LILO, GRUB, etc.

Kernel

Linux

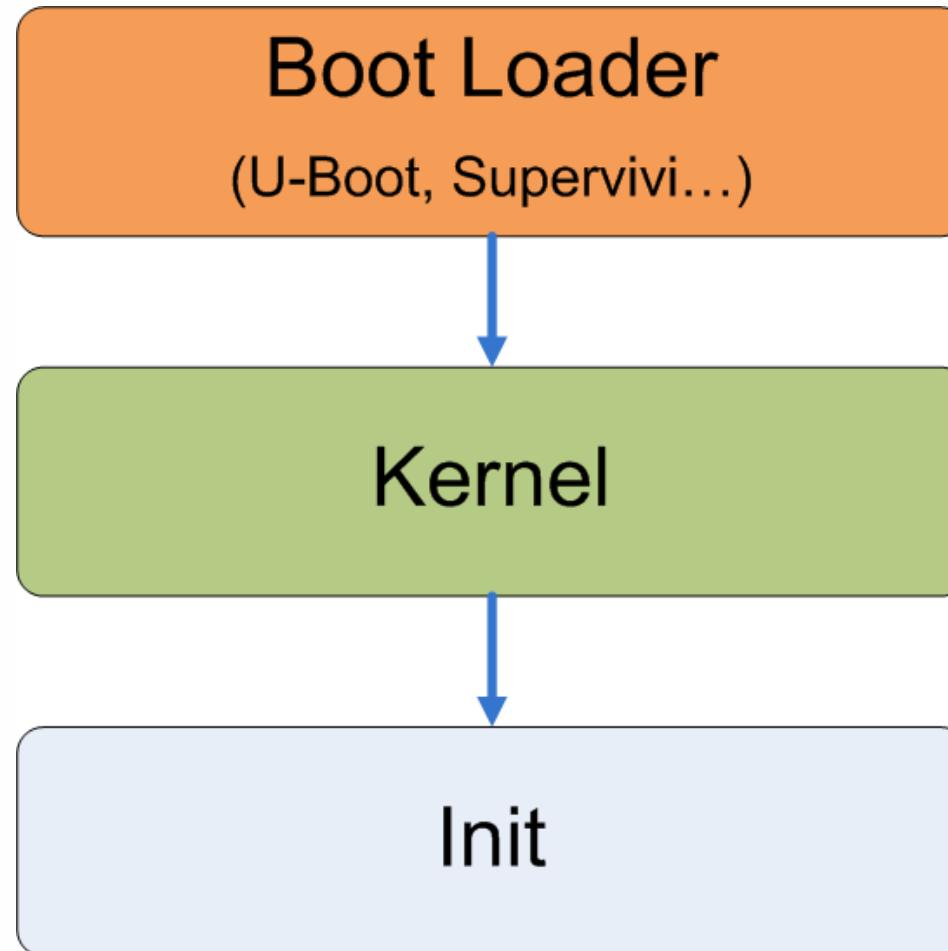
Init

User-space

Operation



Quá trình boot hệ thống Linux nhúng





Quá trình boot hệ thống Linux nhúng

- **Boot loader:** chương trình mồi, thực hiện kiểm tra phần cứng hệ thống và nạp nhân (kernel) của hệ điều hành
- **Kernel:** nhân hệ điều hành, chứa các thành phần cơ bản nhất
- **Root file system:** hệ thống file, chứa các modules bổ sung và các phần mềm ứng dụng



1.3.2. Cài đặt Embedded Linux

- **Bước 1:** Cài đặt bootloader (VD: U-Boot, Supervivi)
- **Bước 2:** Cài đặt kernel
- **Bước 3:** Cài đặt hệ thống file (root file system)



Cài đặt từ môi trường Windows

- Công cụ
 - Phần mềm **HyperTerminal**: kết nối với KIT micro2440 qua cổng COM
 - Phần mềm **DNW**: kết nối với KIT micro2440 qua cổng USB
- Cách thức
 - Phần mềm HyperTerminal (giao tiếp với BIOS trên Nor Flash qua cổng rs232) truyền các lệnh điều khiển
 - Phần mềm DNW trao đổi file



Cài đặt từ môi trường Linux

- Công cụ:
 - Phần mềm **minicom**: kết nối với KIT micro2440 qua cổng COM
 - Phần mềm **usbpush**: kết nối với KIT micro2440 qua cổng USB
- Cách thức
 - Phần mềm minicom cho phép giao tiếp serial, truyền các lệnh điều khiển
 - Phần mềm usbpush nạp file xuống KIT



Cài đặt hệ điều hành nhúng Linux



Demo

**<Xem hướng dẫn chi tiết trong tài liệu
hướng dẫn cài đặt và sử dụng KIT
micro2440>**



1.3.3. Biên dịch, tùy biến nhân Linux

- Khi nào cần biên dịch lại nhân?
 - Khi nâng cấp hệ thống lên các phiên bản mới hơn
 - Khi cần sửa lỗi, thay đổi, tùy chỉnh các module
- Quá trình biên dịch nhân
 - Download nhân tại địa chỉ: kernel.org (Hoặc trong CD đi kèm KIT)
 - File nén linux-2.6.32.2.tar (tùy phiên bản);
 - Giải nén được thư mục toàn bộ mã nguồn của nhân (ví dụ thư mục linux-2.6.32.2)



Biên dịch nhân Linux

- Quá trình biên dịch nhân (tiếp):
 - Vào thư mục chứa mã nguồn nhân (linux-2.6.32.2)
 - Cấu hình trước khi biên dịch bằng lệnh:
make menuconfig
 - Xuất hiện giao diện cấu hình, tùy chỉnh phù hợp với hệ thống.
 - Thực hiện biên dịch bằng lệnh: **make zImage**
 - Biên dịch thành công kết quả sẽ là file zImage (trong thư mục linux-2.6.32.2/arch/arm/mach-s3c2440) , sẽ được nạp (porting) xuống KIT



Biên dịch nhân Linux

Demo

A large blue starburst shape, resembling a five-pointed star with jagged edges, is centered on the slide. Inside the starburst, the word "Demo" is written in a bold, orange-red font.

**<Xem hướng dẫn chi tiết trong tài liệu
hướng dẫn cài đặt và sử dụng KIT
micro2440>**



1.4. Môi trường lập trình KIT FriendlyARM Micro2440

- 1.4.1. Môi trường phát triển ứng dụng nhúng
- 1.4.2. Cài đặt môi trường phát triển KIT 2440
- 1.4.3. Chương trình HelloWorld

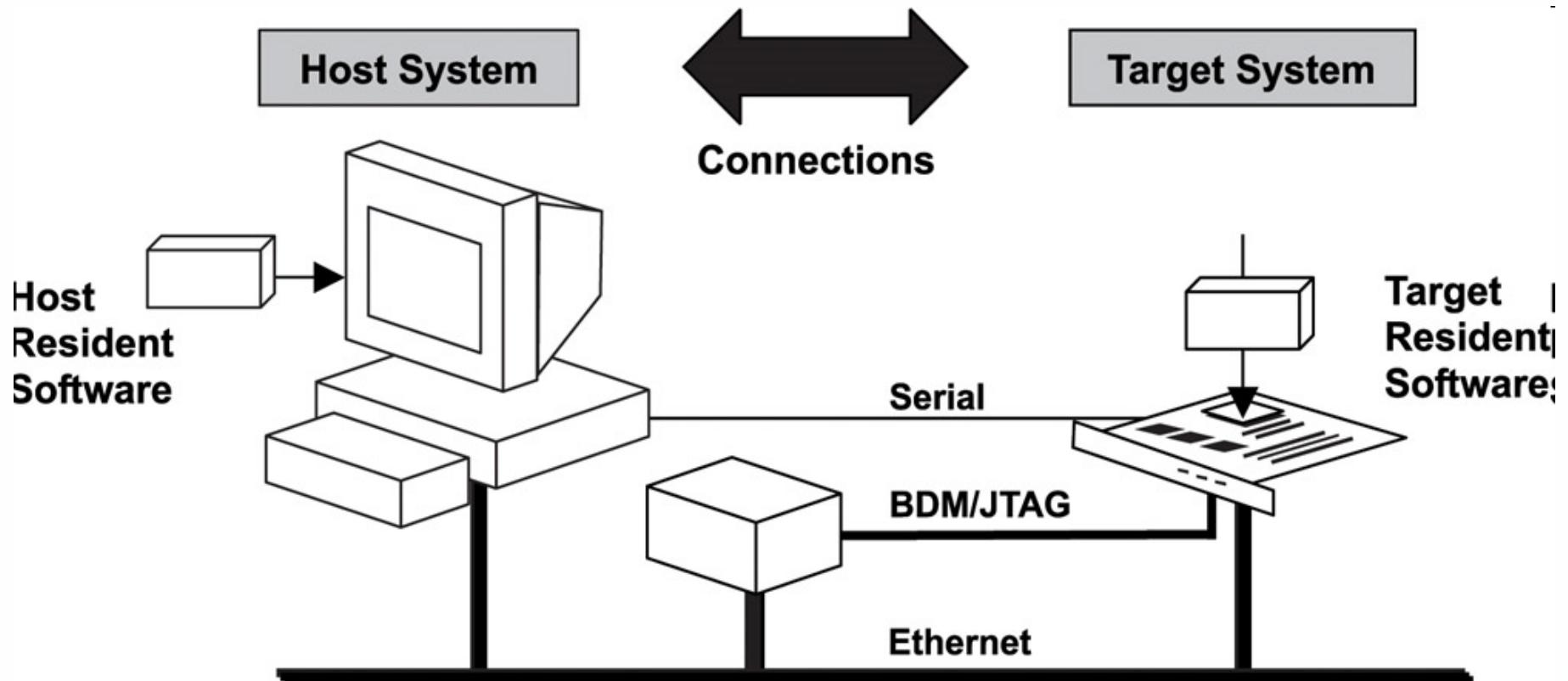


1.4.1. Môi trường phát triển ứng dụng nhúng

- Mô hình lập trình hệ thống nhúng
- Môi trường lập trình KIT micro 2440



Mô hình lập trình hệ thống nhúng

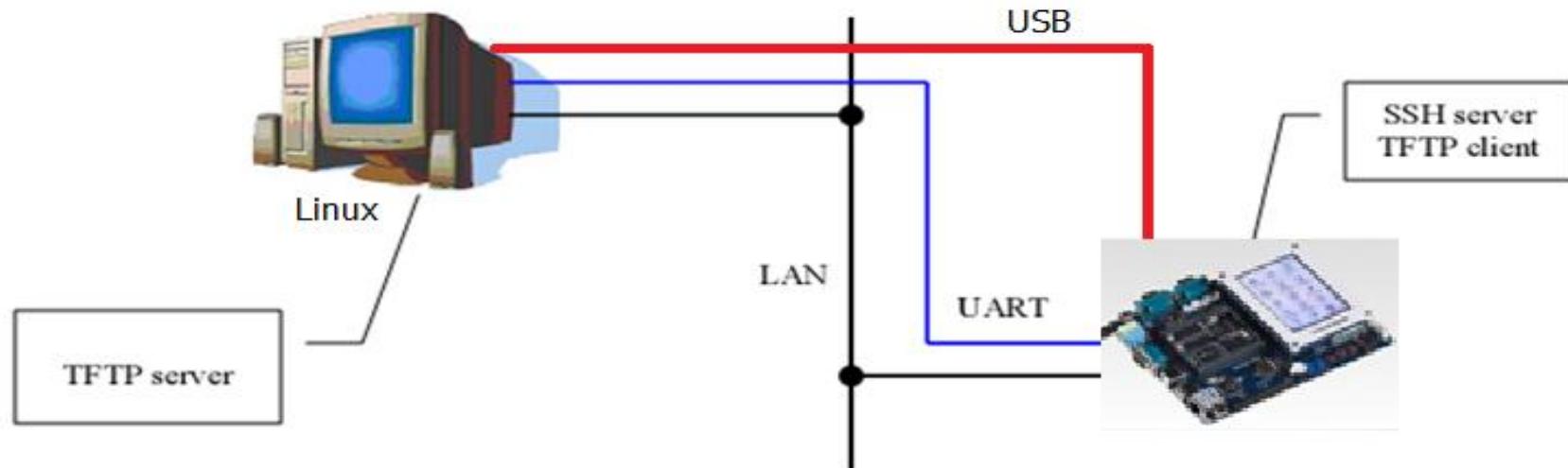


- **Host: hệ thống chứa môi trường phát triển**
- **Target: hệ nhúng cần phát triển ứng dụng**



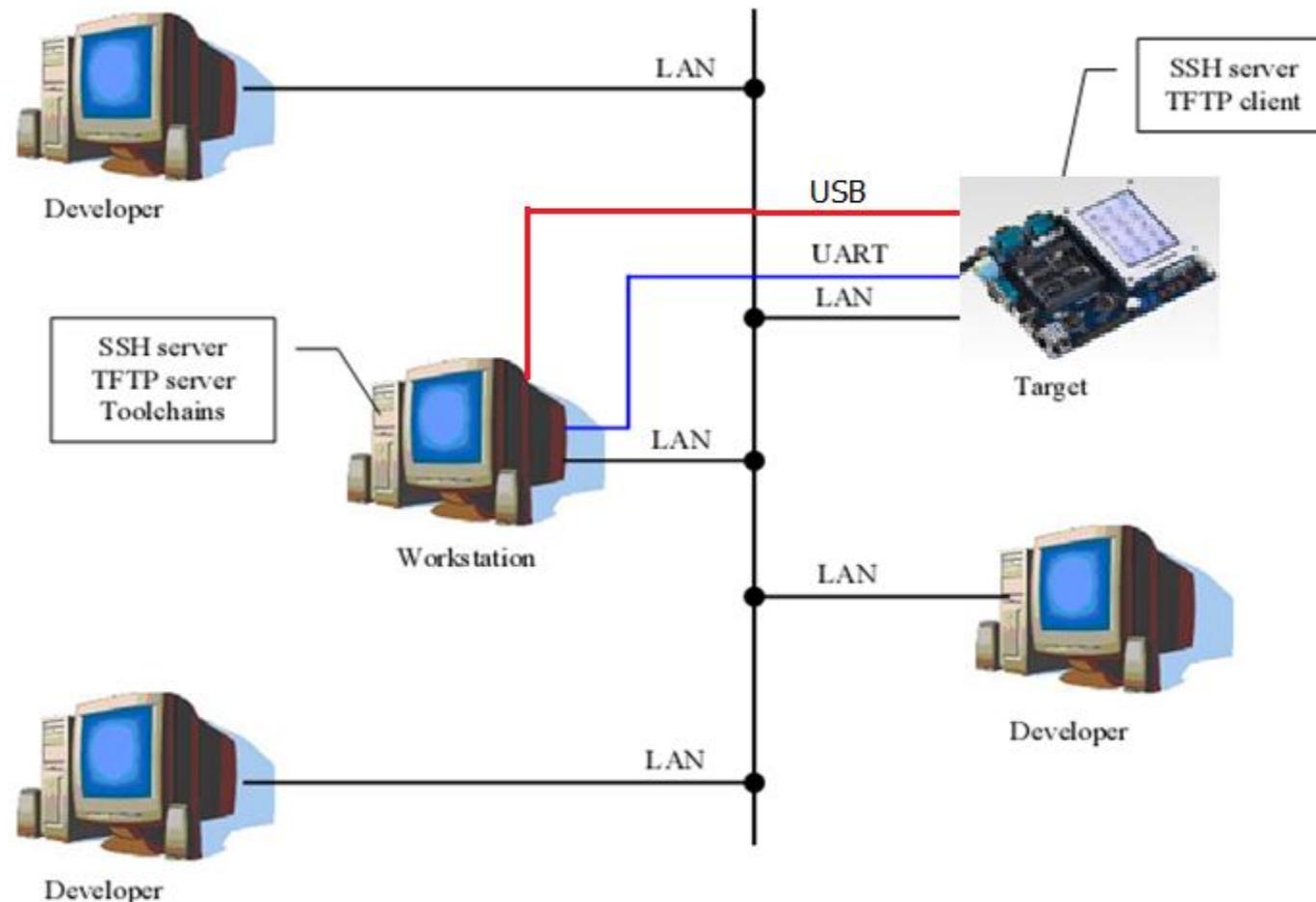
Môi trường lập trình KIT micro 2440

- Máy host cài hệ điều hành Linux (Ubuntu 10.04)
- Trình biên dịch chéo Cross toolchains (arm-linux-gcc 4.4.3): biên dịch ứng dụng (viết bằng C/C++)
- Công cụ viết mã nguồn chương trình C (dùng gedit, eclipse)
- gFTP: truyền nhận file Host<->KIT qua giao thức FTP
- Telnet: kết nối KIT qua Ethernet (sử dụng cross cable)





Môi trường lập trình theo nhóm



Môi trường phát triển ứng dụng theo nhóm



1.4.2. Cài đặt môi trường phát triển

- Cấu hình mạng LAN (host + KIT) qua cáp chéo và sử dụng IP cùng dải:
 - Linux host: 192.168.1.30
 - Linux target: 192.168.1.230 (default)



Cài đặt trình biên dịch chéo

- **Bước 1:** Giải nén arm-linux-gcc-4.4.3.tar.gz
tar -zxvf arm-linux-gcc-4.4.3.tar.gz
- **Bước 2:** Cập nhật biến môi trường PATH
 - Thêm đường dẫn tới thư mục **bin** của arm-linux-gcc-4.4.3 (Cập nhật biến môi trường PATH trong file .bashrc trong đường dẫn chỉ ra bởi biến \$HOME)
- **Bước 3:** Kiểm tra trình biên dịch
 - Mở cử sổ console, gõ lệnh: ***arm-linux-gcc --version***
 - Thông báo về phiên bản của arm-linux-gcc hiện ra => quá trình cài đặt thành công



Kiểm tra trình biên dịch chéo

The screenshot shows a terminal window titled "root@thuan-laptop: /home/thuan". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The terminal content is as follows:

```
root@thuan-laptop:~$ su
Password:
root@thuan-laptop:/home/thuan# arm-linux-gcc --version
.arm-none-linux-gnueabi-gcc (ctng-1.6.1) 4.4.3
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

root@thuan-laptop:/home/thuan#
```



Cài đặt phần mềm gFTP

- **Bước 1:** Cài đặt phần mềm gFTP (nếu chưa có)
 - Gõ lệnh: **sudo apt-get install gftp**
- **Bước 2:** Kiểm tra kết nối giữa Host và Target
 - Mở phần mềm gFTP: **Applications->Internet->gFTP**
 - Thiết lập các tham số
 - ✓ Địa chỉ IP của KIT: 192.168.1.230
 - ✓ Username: root
 - ✓ Password: ktmt (**có thể đổi bằng lệnh passwd**)
 - Mở kết nối



Kết nối sử dụng gFTP

The screenshot shows the gFTP 2.0.19 interface. The title bar reads "gFTP 2.0.19". The menu bar includes ETP, Local, Remote, Bookmarks, Transfer, Log, Tools, and Help. The toolbar includes icons for Host, Port, User, Pass, and FTP. The left panel shows a local directory structure under "/home/thuan" with files like ".adobe", ".cache", ".compiz", ".config", ".dbus", ".ddd", and ".designer". The right panel shows a remote directory structure on the host "192.168.1.230" with directories like "bin", "dev", "etc", "home", "lib", "lost+found", and "mnt". A central transfer area has a blue arrow pointing from left to right. The bottom status bar shows the command "PASV" and the response "227 Entering Passive Mode (192,168,1,230,193,39)", followed by "LIST -aL", "150 Opening BINARY mode data connection for '/bin/ls'.", and "226 Transfer complete."

Filename	Size	User
..	4,096	root
.adobe	4,096	thuan
.cache	4,096	thuan
.compiz	4,096	thuan
.config	4,096	thuan
.dbus	4,096	thuan
.ddd	4,096	thuan
.designer	4,096	thuan

Filename	Size	User
..	2,048	root
bin	2,048	root
dev	0	root
etc	2,048	root
home	2,048	root
lib	2,048	root
lost+found	2,048	root
mnt	2,048	root

PASV
227 Entering Passive Mode (192,168,1,230,193,39)
LIST -aL
150 Opening BINARY mode data connection for '/bin/ls'.
226 Transfer complete.



Cài đặt phần mềm debug GDB

- **Bước 1:** download mã nguồn gdb (version 7.0)
 - Cách 1: [apt-get source gdb](#)
 - Cách 2: <http://ftp.gnu.org/gnu/gdb/>
- **Bước 2:** Biên dịch và cài đặt gdb client trên máy HOST
- **Bước 3:** Biên dịch và cài đặt gdb server trên máy TARGET

(Chi tiết xem trong tài liệu hướng dẫn cài đặt môi trường phát triển ứng dụng)



1.4.3. Chương trình HelloWorld

- Cấu trúc chương trình đơn giản
- Cách thức biên dịch chương trình
- Nạp file thực thi xuống KIT và chạy ứng dụng



Cấu trúc chương trình

■ Tuân thủ cấu trúc chương trình ANSII C

Khai báo tệp tiêu đề

```
#include
```

Định nghĩa kiểu dữ liệu

```
typedef ...
```

Khai báo các hàm nguyên mẫu

Khai báo các biến toàn cục

Định nghĩa hàm **main()**

```
main()
```

```
{
```

```
    ...
```

```
}
```

Định nghĩa các hàm đã khai báo nguyên mẫu



Chương trình Hello World

- Soạn thảo mã nguồn chương trình C bằng gedit (file Hello.c)

```
#include <stdio.h>
int main (int argc, char* argv[])
{
    printf("Hello World !\n");
    printf ("Ten chuong trinh la '%s'.\n", argv[0]);
    printf ("Chuong trinh co %d tham so \n", argc - 1);
    /* Neu co bat cu tham so dong lenh nao*/
    if (argc > 1) {
        /* Thi in ra*/
        int i;
        printf ("Cac tham so truyen vao la:\n");
        for (i = 1; i < argc; ++i)
            printf (" Tham so %d: %s\n", i, argv[i]);
    }
    return 0;
}
```



Cách thức biên dịch chương trình

- **Cách 1:** Sử dụng lệnh của cross compiler
 - VD: `arm-linux-gcc -g -o Hello Hello.c`
 - Kết quả: biên dịch ra một file thực thi có tên là Hello từ một file mã nguồn là Hello.c, file này có hỗ trợ khả năng debug (-g)
- **Cách 2:** Tạo và sử dụng Makefile
 - make là một tool cho phép quản lý quá trình biên dịch, liên kết ... của một dự án với nhiều file mã nguồn.
 - Tạo Makefile lưu các lệnh biên dịch theo định dạng của Makefile
 - Sử dụng lệnh `make` để chạy Makefile và biên dịch chương trình
- **Cách 3:** Sử dụng **automake** và **autoconf**
 - Tạo makefile tự động



Cấu trúc Makefile

- Makefile cấu thành từ các target, variables và comments
- Target có cấu trúc như sau:

target: dependencies

[tab] system command

- target: make target
- Dependencies: các thành phần phụ thuộc (file mã nguồn, các file object...)
- System command: các câu lệnh (lệnh biên dịch, lệnh linux)



VD 1: Makefile đơn giản

```
CC=arm-linux-gcc
```

```
all: Hello.c
```

```
    $(CC) -g -o Hello Hello.c
```

```
clear:
```

```
    rm Hello
```

- **Biên dịch chương trình: make all**
- **Xóa file sinh ra trước đó: make clear**



VD 2: Makefile liên kết

Hello.c

include

Display.c

void display(int index, char* str)

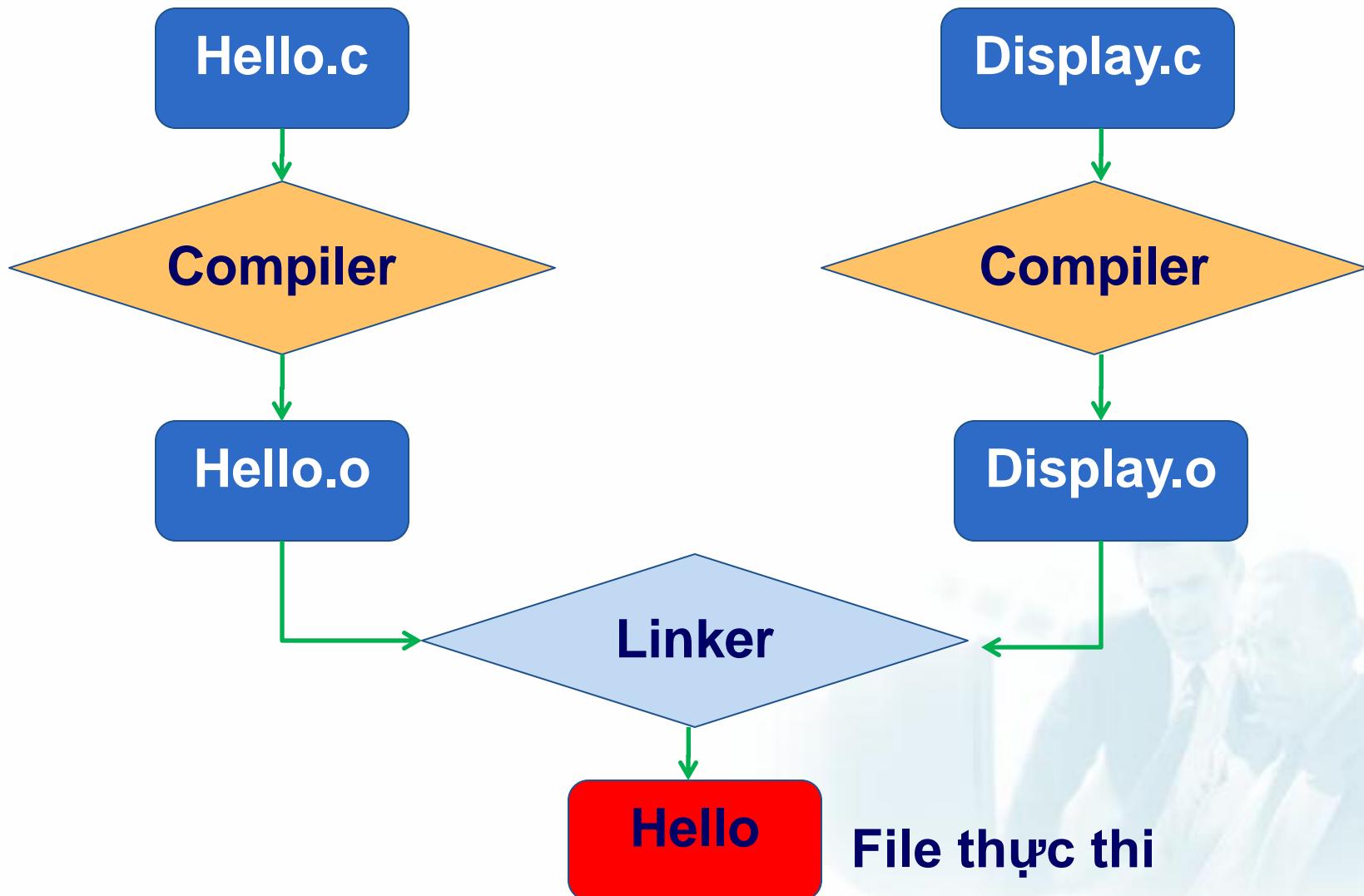
include

Display.h

void display(int index, char* str)



VD2: Makefile liên kết





VD 2: Makefile liên kết

```
CC=arm-linux-gcc
```

```
OUTPUT=Hello
```

```
all:Hello.o display.o
```

```
    $(CC) -o $(OUTPUT) Hello.o display.o
```

```
Hello.o:Hello.c
```

```
    $(CC) -c Hello.c
```

```
display.o:display.c
```

```
    $(CC) -c display.c
```



Nạp file thực thi xuống KIT

- **Bước 1:** sử dụng phần mềm gFTP chuyển file Hello (đã được biên dịch trước đó) xuống KIT, ví dụ xuống thư mục: /ktmt
- **Bước 2:** telnet xuống KIT, chuyển tới thư mục /ktmt, thực thi chương trình
 - Gõ lệnh: ./Hello
 - Nếu chương trình chưa có quyền thực thi, thực hiện cấp quyền: chmod +x Hello
- **Bước 3:** quan sát kết quả



Thảo luận





Chương 2

Lập trình vào ra cơ bản



Mục tiêu chương 2

- Sau khi kết thúc chương này, sinh viên có thể
 - Nắm được nguyên tắc lập trình giao tiếp vào ra cơ bản trên hệ điều hành Linux nhúng
 - Lập trình giao tiếp thiết bị (ghép nối GPIO) với driver đã có (led, button)
 - Biết cách lập trình giao tiếp GPIO mở rộng dựa trên giao diện sysfs (gpiolib)



Nội dung bài học

- 2.1. Cơ chế lập trình giao tiếp thiết bị
- 2.2. Lập trình điều khiển led đơn
- 2.3. Lập trình giao tiếp nút bấm
- 2.4. Lập trình giao tiếp GPIO mở rộng

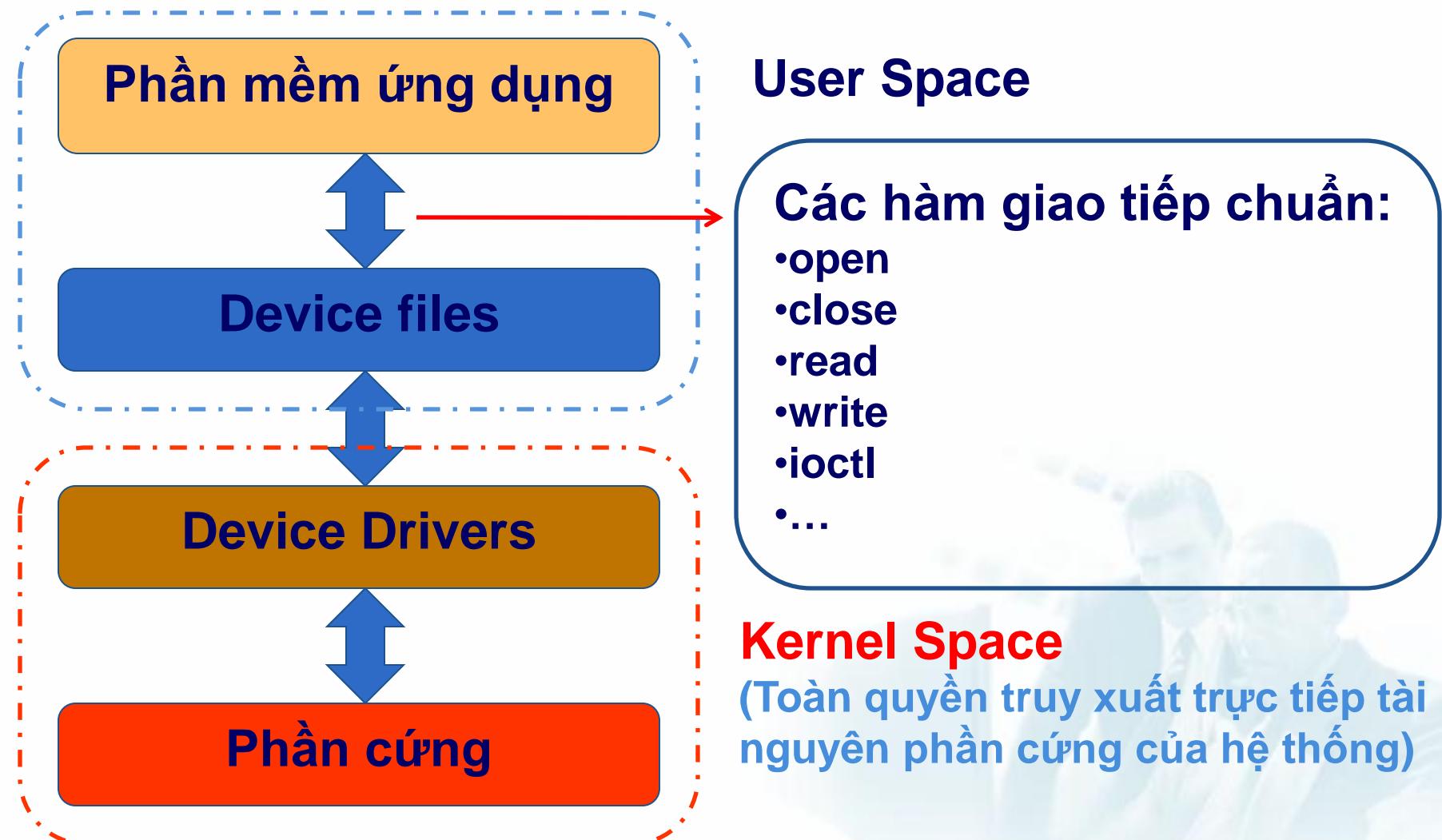


2.1. Cơ chế lập trình giao tiếp thiết bị

- Device files, Device number
- Kiểm tra danh sách device driver, thiết bị
- Cơ chế giao tiếp



Mô hình giao tiếp ứng dụng - thiết bị





Device files, Device number

- Device files: `ls -l /dev`

- Device file không phải là file thông thường, không phải là một vùng dữ liệu trên hệ thống file
 - Quá trình đọc ghi device file
 - ✓ Giao tiếp với device driver
 - ✓ Đọc, ghi phần cứng của thiết bị

- Phân loại device files

- Character device: thiết bị phần cứng đọc, ghi một chuỗi các byte dữ liệu
 - Block device: thiết bị phần cứng đọc, ghi một khối dữ liệu



Device files, Device number

- Device number: mỗi thiết bị được xác định bởi hai giá trị
 - Major device number: xác định thiết bị này sử dụng driver nào
 - Minor device number: phân biệt giữa các thiết bị khác nhau cùng sử dụng chung một device driver



Kiểm tra danh sách thiết bị

- Kiểm tra danh sách các thiết bị

- Gõ lệnh **ls -al /dev**

```
brw-rw---- 1 root disk 3, 0 May 5 1998 /dev/hda  
brw-rw---- 1 root disk 3, 1 May 5 1998 /dev/hda1
```

- Giải thích thông tin

- ❖ Loại thiết bị: char device hay block device
 - ❖ Tài khoản người dùng
 - ❖ Tên thiết bị
 - ❖ Major và minor number
 - ❖ Mount point



Kiểm tra danh sách thiết bị

- Kiểm tra danh sách các nhóm thiết bị
 - Gõ lệnh **cat /proc/devices**

```
Character devices:  
 1 mem  
 4 /dev/vc/0  
 4 tty  
 4 ttys  
 5 /dev/tty  
 5 /dev/console  
 5 /dev/ptmx  
 7 vcs  
 10 misc  
 13 input  
 29 fb  
 36 netlink  
 128 ptm  
 136 pts  
 180 usb  
  
Block devices:  
 1 ramdisk  
 3 ide0  
 9 md  
 22 ide1  
 253 device-mapper  
 254 mdp
```



Cơ chế lập trình giao tiếp thiết bị

- Cơ chế lập trình

- Sử dụng các hàm vào ra file

- ✓ open

- ✓ close

- ✓ read

- ✓ write

- Sử dụng hàm điều khiển vào ra: **ioctl**



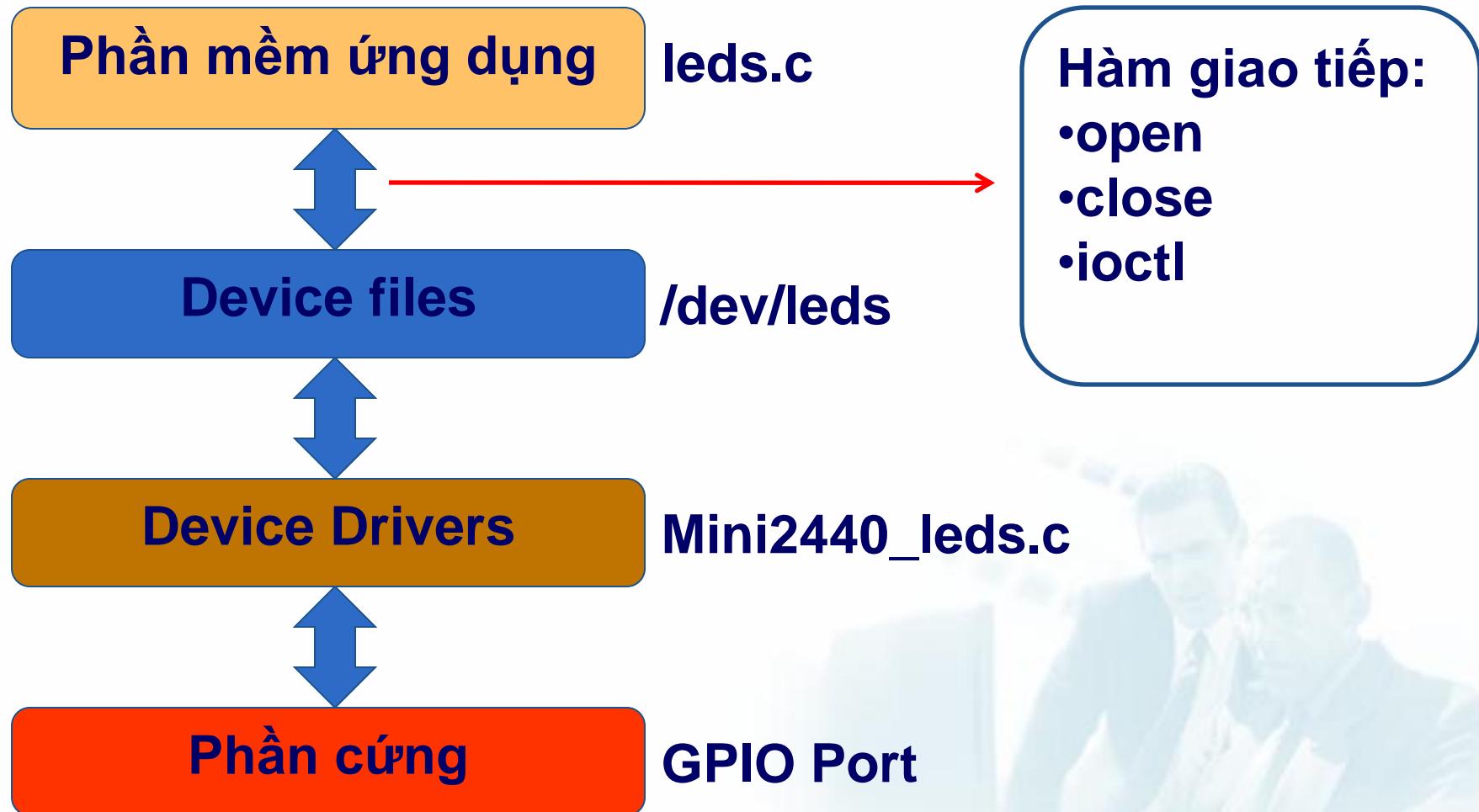
2.2. Lập trình điều khiển led đơn

- Sử dụng led driver đã có
- 4 led đơn, ghép nối qua GPB5,6,7,8
- Điều khiển led on/off, tạo hiệu ứng: nhấp nháy, chạy đuối, ...
- Cần sử dụng hàm trễ (delay): sleep, usleep (thư viện sys/time.h)





Mô hình giao tiếp điều khiển led





Lập trình điều khiển led đơn

- **fd=open("/dev/leds",0)**

- fd: file id
 - /dev/leds: device file
 - 0: WRITE_ONLY

- **ioctl(fd, on, led_no)**

- ioctl: IO control
 - Điều khiển bật/tắt led đơn có số hiệu led_no

- Driver cho led đơn:

linux-2.6.32.2/drivers/char/mini2440_leds.c



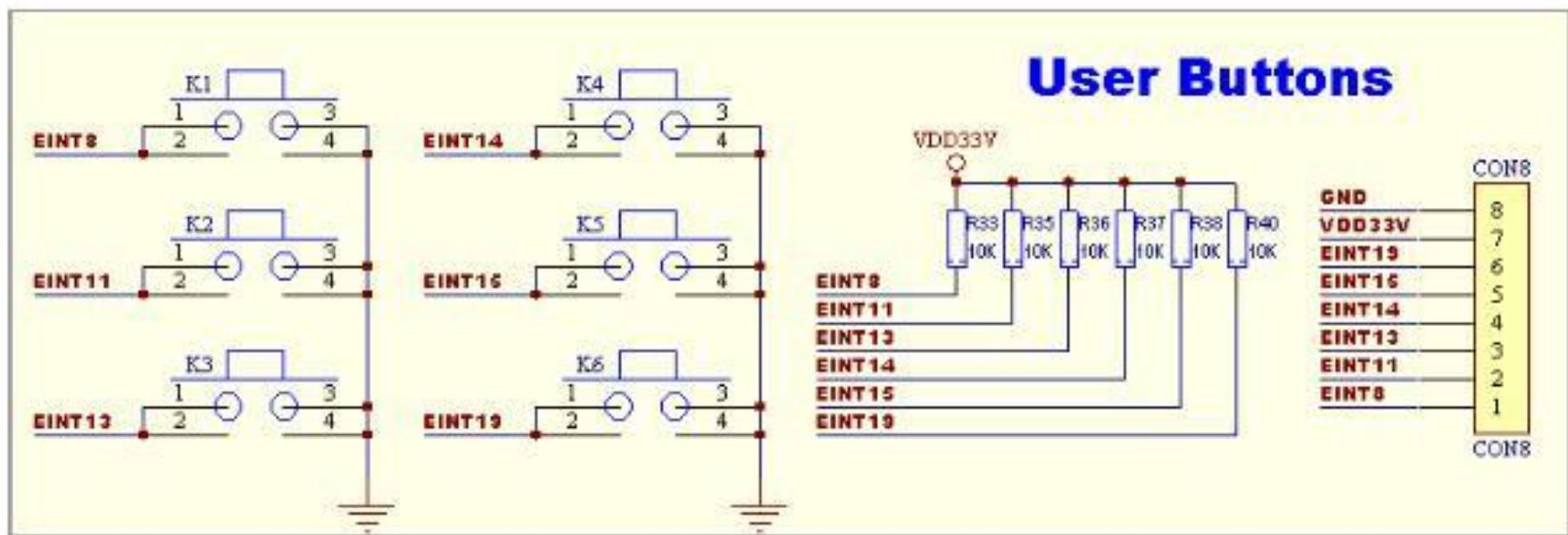
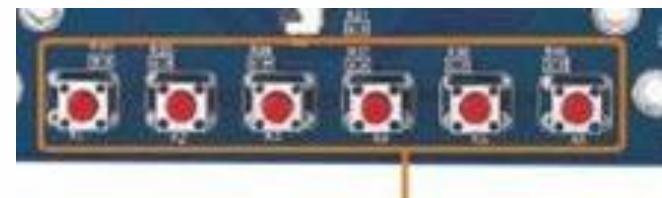
Mã nguồn minh họa điều khiển led đơn

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
int main(int argc, char **argv)
{
    int on;
    int led_no;
    int fd;
    sscanf(argv[1], "%d", &led_no);
    sscanf(argv[2], "%d", &on);
    fd = open("/dev/leds", 0);
    if (fd < 0) {
        perror("open device leds");
        exit(1);
    }
    ioctl(fd, on, led_no);
    close(fd);
    return 0;
}
```



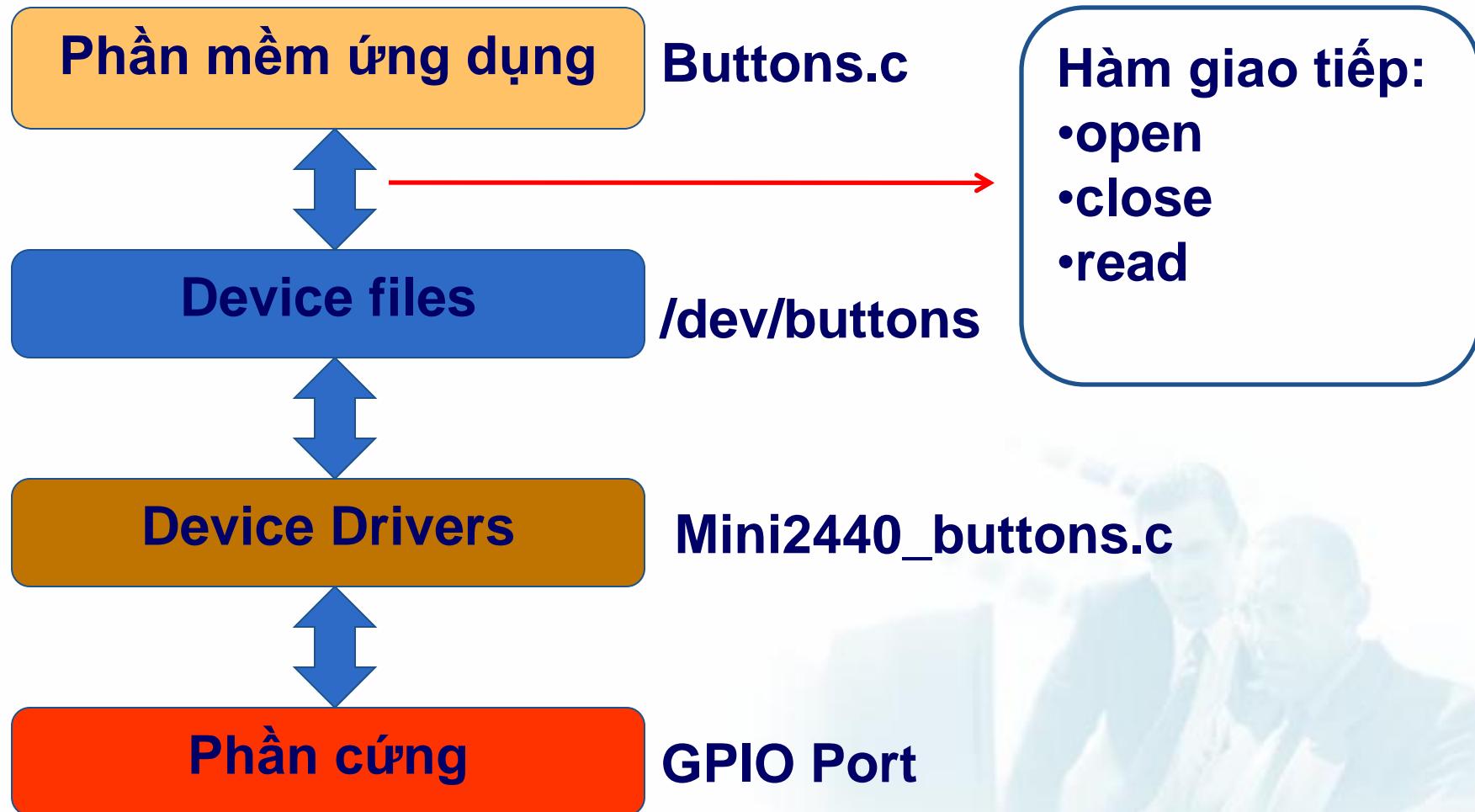
2.3. Lập trình giao tiếp nút bấm

- Giao tiếp qua driver đã có





Mô hình giao tiếp điều khiển nút bấm





Lập trình ghép nối nút bấm

- **buttons_fd=open("/dev/buttons",0)**
 - buttons_fd: file id
 - /dev/buttons: device file
- **read(buttons_fd,current_buttons,sizeof(current_buttons))**
 - Đọc trạng thái các nút bấm
- **close(buttons_fd):** đóng file
- Mã nguồn driver cho nút bấm
**linux-
2.6.32.2/drivers/char/mini2440_buttons.c**



Mã nguồn chương trình đọc nút bấm

```
int main(void)
{
    int buttons_fd;
    char buttons[6] = {'0', '0', '0', '0', '0', '0'};

    buttons_fd = open("/dev/buttons", 0);
    if (buttons_fd < 0) {
        perror("open device buttons");
        exit(1);
    }
    for (;;) {
        char current_buttons[6];
        int count_of_changed_key;
        int i;
        read(buttons_fd, current_buttons, sizeof current_buttons);
        for (i = 0, count_of_changed_key = 0;
             i < sizeof buttons / sizeof buttons[0]; i++) {
            if (buttons[i] != current_buttons[i]) {
                buttons[i] = current_buttons[i];
                count_of_changed_key++;
            }
        }
        close(buttons_fd);
        return 0;
    }
}
```



2.4. Lập trình giao tiếp GPIO mở rộng

- 2 cách sử dụng giao tiếp gpio (từ Linux user space)
 - Cách 1: Viết gpio driver (trên không gian nhân hệ điều hành, kernel space), giao tiếp qua driver này.
(Ví dụ với led, button đã làm)
 - Cách 2: giao tiếp các chân gpio trực tiếp từ không gian người dùng (user space) dựa trên API thư viện gpiolib cung cấp. Linux cung cấp giao diện GPIO sysfs cho phép thao tác với bất kỳ chân GPIO từ userspace.



Lập trình giao tiếp GPIO mở rộng

- Tất cả các giao diện điều khiển GPIO thông qua sysfs nằm trong thư mục /sys/class/gpio
- Kiểm tra bằng lệnh: ls /sys/class/gpio

```
File Edit View Terminal Help

Kernel 2.6.32.2-FriendlyARM on (/dev/pts/0)
FriendlyARM login: root
Password:
[root@FriendlyARM /]# ls
bin      home   linuxrc    opt       sbin      usr
dev      kmt     lost+found  proc      sys       var
etc      lib      mnt       root      tmp       www
[root@FriendlyARM /]# cd /sys/class/gpio
[root@FriendlyARM gpio]# ls
export  gpiochip128  gpiochip192  gpiochip32  gpiochip96
gpiochip0  gpiochip160  gpiochip224  gpiochip64  unexport
[root@FriendlyARM gpio]# echo 161 > /sys/class/gpio/export
[root@FriendlyARM gpio]# ls
export  gpiochip0  gpiochip160  gpiochip224  gpiochip64  unexport
gpio161  gpiochip128  gpiochip192  gpiochip32  gpiochip96
[root@FriendlyARM gpio]# echo 161 > /sys/class/gpio/unexport
[root@FriendlyARM gpio]# ls
export  gpiochip128  gpiochip192  gpiochip32  gpiochip96
gpiochip0  gpiochip160  gpiochip224  gpiochip64  unexport
[root@FriendlyARM gpio]# 

PIO sysfs interface cần cấu hình nhân hệ điều hành cho phép sử dụng giao
cấu hình trước khi biên dịch nhân (nếu hệ điều hành cài đặt chưa có)
```



Lập trình giao tiếp GPIO mở rộng

- Giao diện này cung cấp các files điều khiển sau đây:

`export` Make a specific GPIO pin available for userspace control. Write the pin number N (e.g. "55", ASCII); the gpioN directory should appear.

`gpioN/direction` Write "in" or "out" to set pin direction. Write "high" or "low" to set direction to output, with initial value, atomically.

`gpioN/value` Read the current pin status in input. For output, write "0" or "1" to set the pin status. To get change notification (`interrupt`) `Iseek()` to end of file, and either `poll()` for POLLPRI and POLLERR, or `select()` with the file descriptor in `exceptfds`.

`gpioN/edge` Write "none", "rising", "falling", or "both" to select the signal that makes `poll()` return.



Lập trình giao tiếp GPIO mở rộng

- Ví dụ minh họa

Cấu hình chân GPF5 (micro2440) output, và xuất giá trị 0 ra chân này

```
echo 165 > /sys/class/gpio/export
```

```
echo "out" > /sys/class/gpio/gpio165/direction
```

```
echo 0 > /sys/class/gpio/gpio165/value
```

Chi tiết xem bài viết:

<https://sites.google.com/site/embedded247/ddcourse/giao-tiep-gpio-tu-userspace-1>



Thảo luận





Chương 3

Lập trình vào ra nâng cao



Mục tiêu chương 3

- Sau khi kết thúc chương này, sinh viên có thể
 - Nắm được chuẩn RS232
 - Lập trình giao tiếp chuẩn RS232 trên kit nhúng Micro2440
 - Nắm được chuẩn giao tiếp USB
 - Lập trình ghép nối USB Joystick qua cổng USB
 - Lập trình giao tiếp ADC



Nội dung bài học

- 3.1. Giới thiệu chuẩn RS232
- 3.2. Lập trình giao tiếp chuẩn RS232
- 3.3. Giới thiệu chuẩn USB
- 3.4. Lập trình giao tiếp USB Joystick
- 3.5. Lập trình giao tiếp ADC



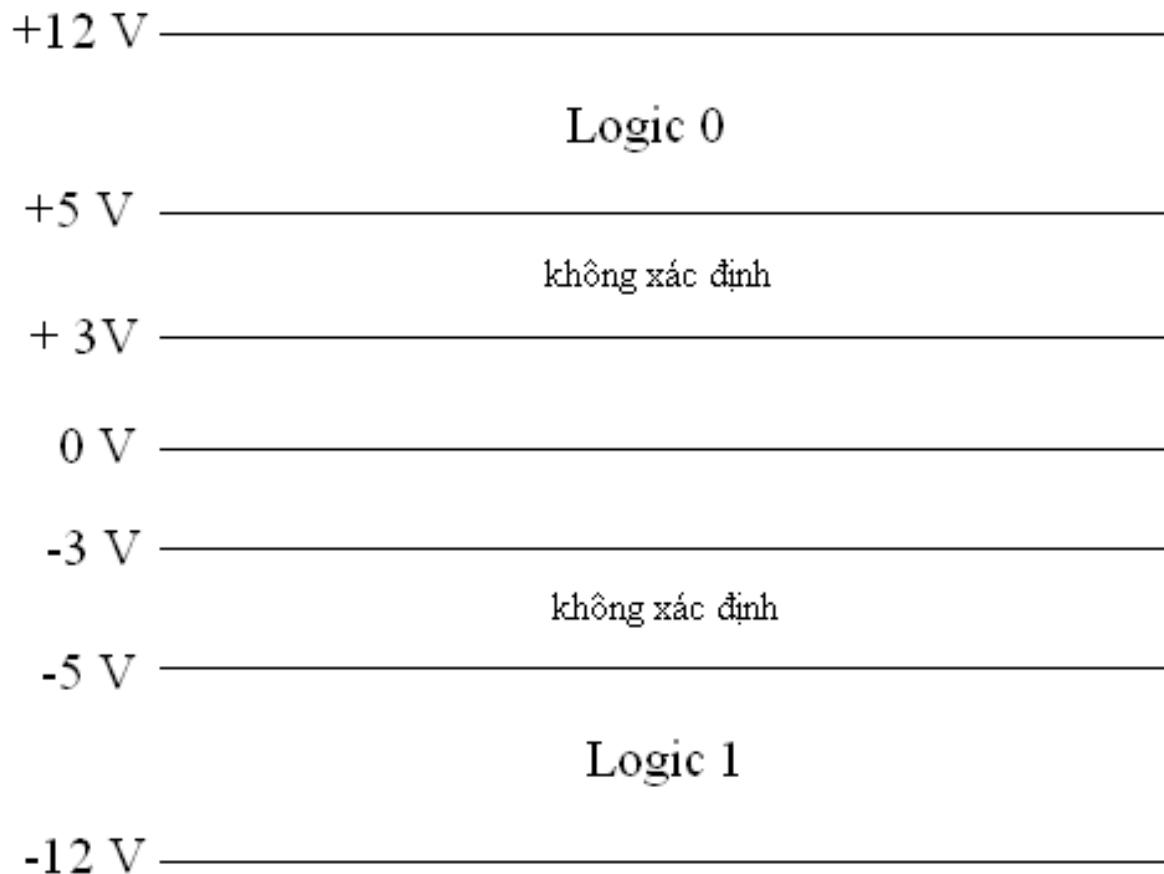
3.1. Giới thiệu chuẩn RS232

- Mức điện áp đường truyền
- Chuẩn đầu nối trên máy tính PC
- Khuôn dạng khung truyền
- Tốc độ truyền
- Kích bản truyền



Chuẩn RS232

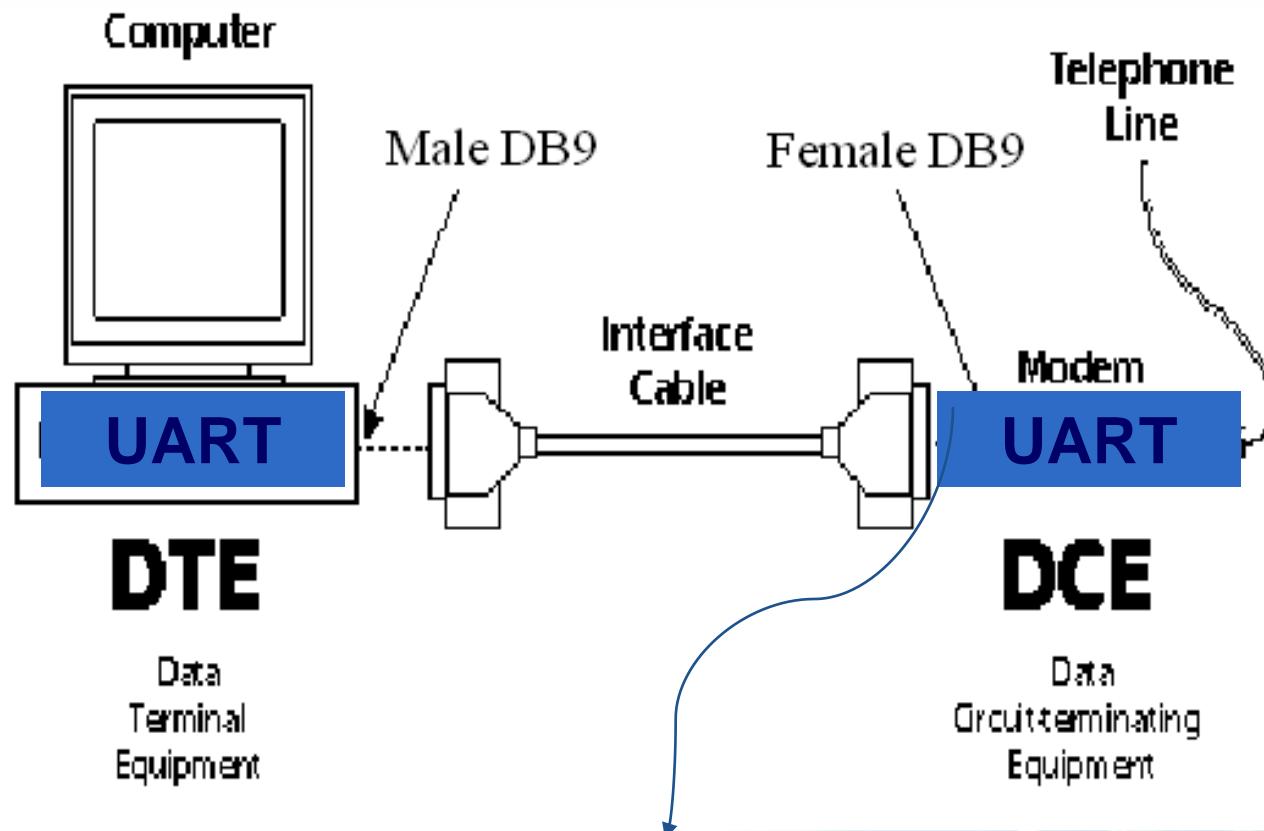
- Mức điện áp đường truyền (Chuẩn RS-232C)





Chuẩn RS232

- Chuẩn đấu nối trên PC



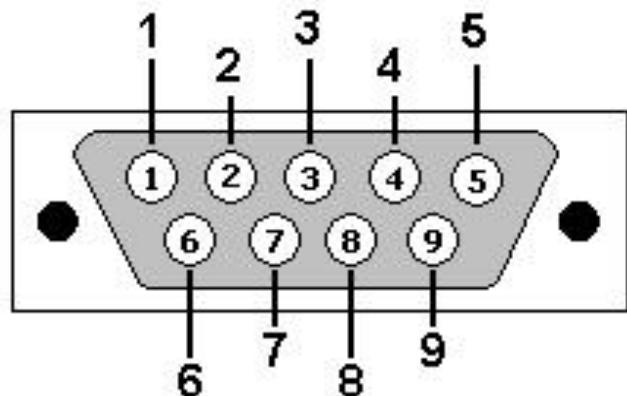
UART (Universal Asynchronous receiver/transmitter)



Chuẩn RS232

▪ Chuẩn đầu nối trên PC

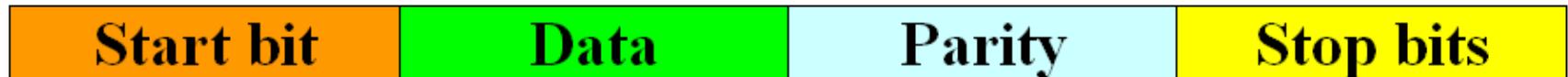
- Chân 1 (DCD-Data Carrier Detect): phát hiện tín hiệu mang dữ liệu
- Chân 2 (RxD-Receive Data): nhận dữ liệu
- Chân 3 (TxD-Transmit Data): truyền dữ liệu
- Chân 4 (DTR-Data Terminal Ready): đầu cuối dữ liệu sẵn sàng
- Chân 5 (Signal Ground): đất của tín hiệu
- Chân 6 (DSR-Data Set Ready): dữ liệu sẵn sàng
- Chân 7 (RTS-Request To Send): yêu cầu gửi
- Chân 8 (CTS-Clear To Send): Xóa để gửi
- Chân 9 (RI-Ring Indicate): báo chuông





▪ Khuôn dạng khung truyền

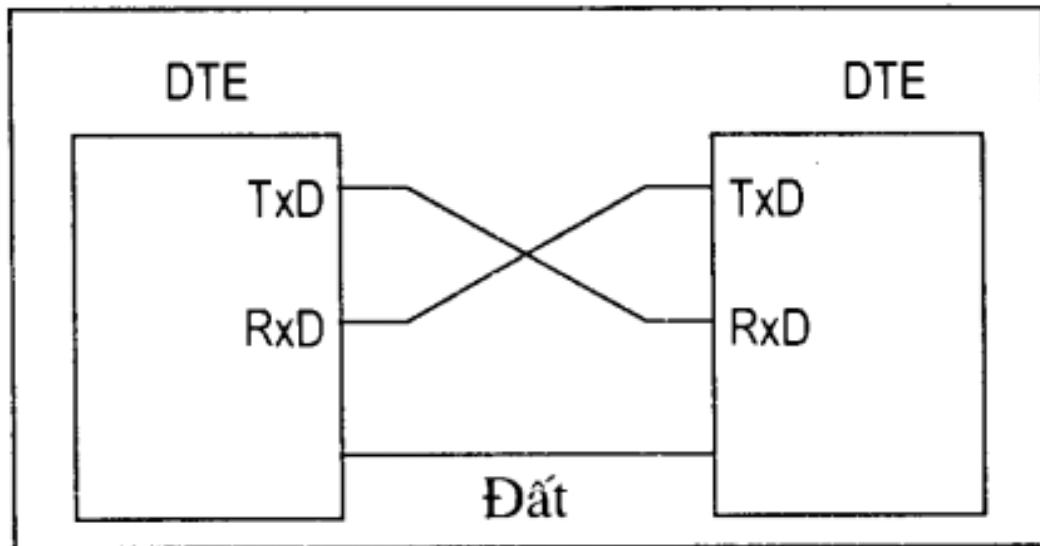
- PC truyền nhận dữ liệu qua cổng nối tiếp RS-232 thực hiện theo kiểu không đồng bộ (Asynchronous)
- Khung truyền gồm 4 thành phần
 - ✓ 1 Start bit (Mức logic 0): bắt đầu một gói tin, đồng bộ xung nhịp clock giữa DTE và DCE
 - ✓ Data (5,6,7,8 bit): dữ liệu cần truyền
 - ✓ 1 parity bit (chẵn (even), lẻ (odd), mark, space): bit cho phép kiểm tra lỗi
 - ✓ Stop bit (1 hoặc 2 bit): kết thúc một gói tin





▪ Kịch bản truyệ̀n

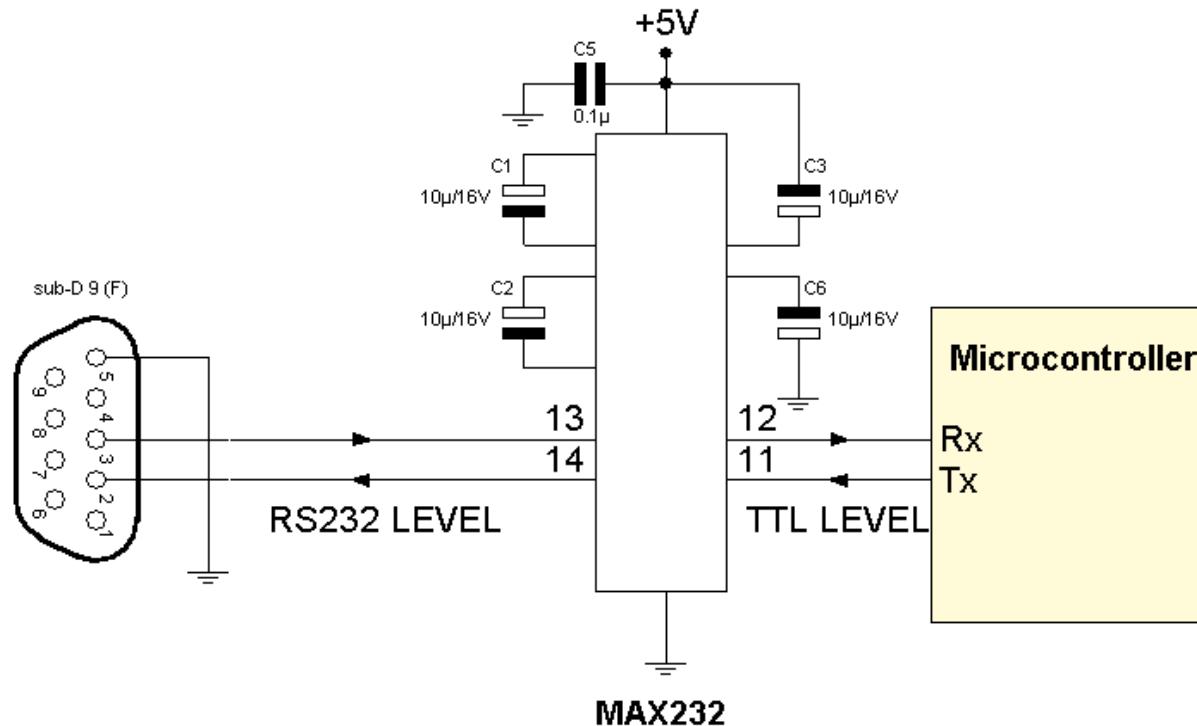
- Khộng có bắt tay (none-handshaking): máy thu có khả năng đọc các ký tự thu trước khi máy phát truyệ̀n ký tự tiếp thẹ



**Kết nối không cần bắt tay giữa hai thiết bị
(cùng mức điện áp)**



▪ Kịch bản truyề



**Ghép nối không bắt tay giữa hai thiết bị
(Khác nhau về mức điện áp)**



3.2. Lập trình giao tiếp chuẩn RS232

- **Khởi tạo:** Khai báo thư viện
- **Bước 1:** Mở cổng
- **Bước 2:** Thiết lập tham số
- **Bước 3:** Đọc, ghi cổng
- **Bước 4:** Đóng cổng



Khai báo thư viện

- #include <stdio.h>
- #include <stdlib.h>
- #include <string.h>
- #include <unistd.h> // UNIX standard function
- #include <fcntl.h> // File control definitions
- #include <errno.h> // Error number definitions
- #include <termios.h> // POSIX terminal control
- #include <time.h> // time calls



Bước 1: Mở cổng

- Sử dụng lệnh mở file

int fd = open ("/dev/ttySAC0", O_RDWR);

- Fd >0 nếu mở file thành công
- Fd<0 nếu mở file thất bại



Bước 2: Thiết lập tham số

- Sử dụng cấu trúc termios

struct termios port_settings;

- Thiết lập tham số (9600, 8, n, 1)

cfsetispeed(&port_settings, B9600);

cfsetospeed(&port_settings, B9600);

port_settings.c_cflag &= ~PARENB;

port_settings.c_cflag &= ~CSTOPB;

port_settings.c_cflag &= ~CSIZE;

port_settings.c_cflag |= CS8;

tcsetattr(fd, TCSANOW, &port_settings);



Bước 3: Đọc, ghi cổng

- Đọc cổng: sử dụng lệnh đọc file

`n=read(fd,&result,sizeof(result));`

➤ N: số ký tự đọc được

➤ Result: chứa kết quả

- Ghi cổng: sử dụng lệnh ghi file

`n=write(fd,"Hello World\r",12);`

➤ N:số ký tự đã ghi

➤ Fd: file id (có được từ thao tác mở file thành công)



Bước 4: Đóng cổng

- Đóng cổng: sử dụng lệnh đóng file
`close (fd);`

Fd: file ID (có được từ thao tác mở file thành công)



Demo



3.3. Giới thiệu chuẩn USB

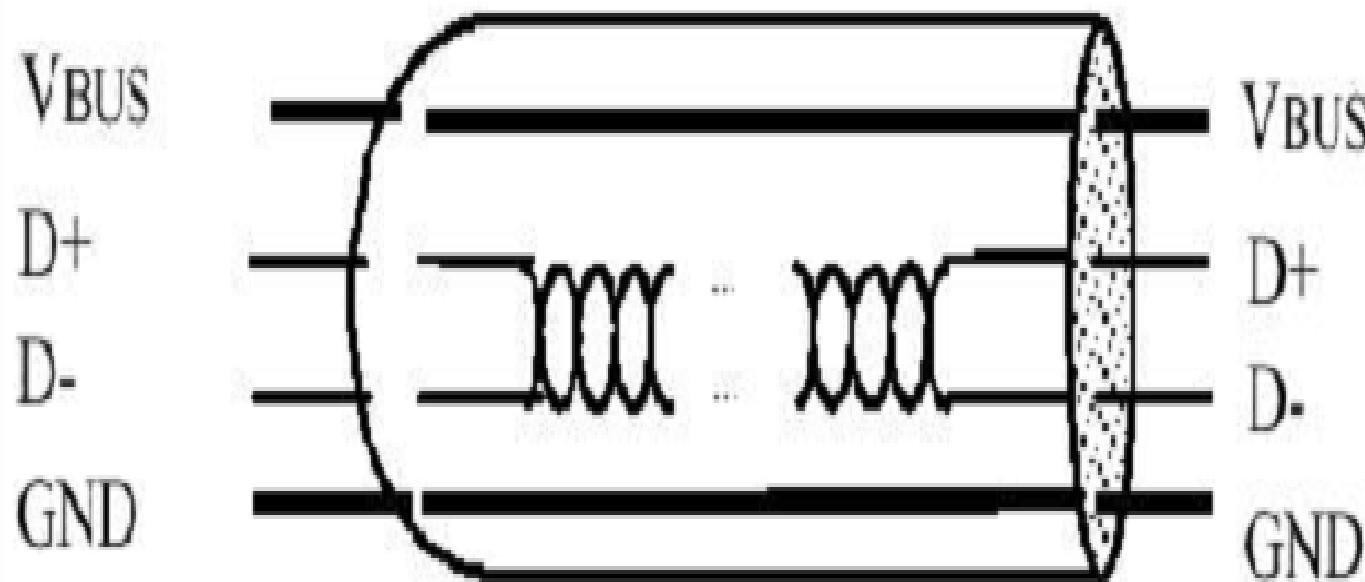
- Năm 1995: USB 1.0
 - Tốc độ Low-Speed: 1.5 Mbps
 - Tốc độ tối đa (Full-Speed): 12 Mbps
- Năm 1998: USB 1.1 (Sửa lỗi của USB 1.0)
 - Tốc độ tối đa (Full-Speed): 12 Mbps
- Năm 2001: USB 2.0
 - Tốc độ tối đa (High-Speed): 480 Mbps
- Năm 2008: USB 3.0
 - Tốc độ tối đa (Super-Speed): 4.8 Gbps



Tín hiệu chuẩn USB

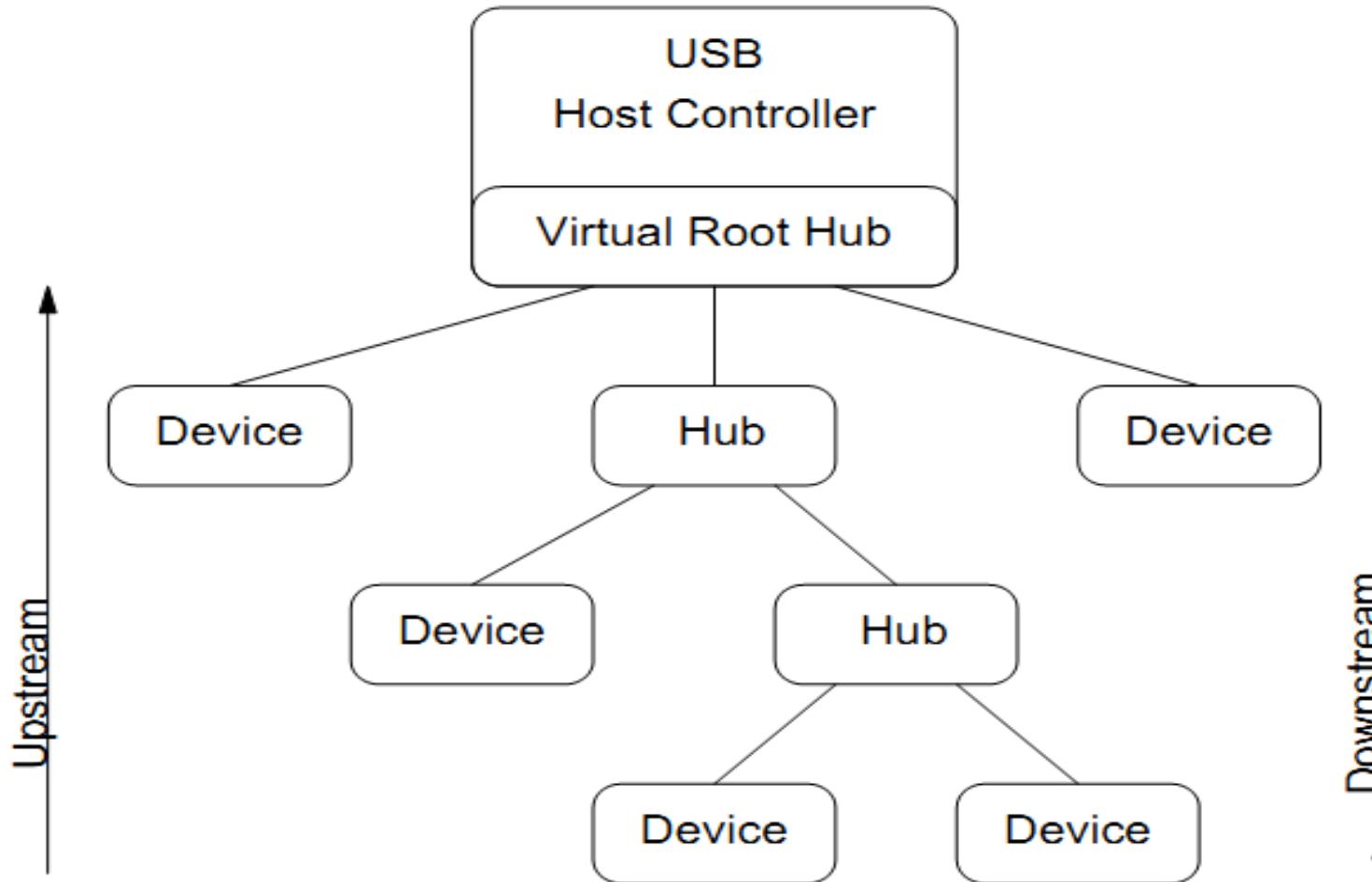
- Tín hiệu

- Truyền kiểu nối tiếp
- Tín hiệu trên hai đường D+ và D- là tín hiệu vi sai





Mô hình bus USB





Vai trò của các thành phần

- Vai trò của USB host:

- Trao đổi dữ liệu với các thiết bị ngoại vi
- Điều khiển USB bus:
 - ✓ Quản lý được các thiết bị kết nối vào đường bus và khả năng của mỗi thiết bị đó: sử dụng cơ chế điểm danh (Enumeration)
 - ✓ Phân xử, quản lý luồng dữ liệu trên bus, đảm bảo các thiết bị đều có cơ hội trao đổi dữ liệu
- Kiểm tra lỗi: thêm các mã kiểm tra lỗi vào gói tin cho phép phát hiện lỗi và yêu cầu truyền lại gói tin
- Cung cấp nguồn điện cho tất cả các thiết bị



Vai trò của các thành phần

- Vai trò của thiết bị ngoại vi
 - Trao đổi dữ liệu với host
 - Phát hiện các gói tin hay yêu cầu (request) được gửi tới thiết bị để xử lý phù hợp
 - Kiểm tra lỗi: tương tự như Host, các thiết bị ngoại vi cũng phải chèn thêm các bit kiểm tra lỗi vào gói tin gửi đi
 - Quản lý nguồn điện: các thiết bị có thể sử dụng nguồn điện ngoài hay nguồn từ bus. Nếu sử dụng nguồn từ bus, phải chuyển sang chế độ tiết kiệm điện năng.



Endpoint & pipes

- Mỗi quá trình truyền nhận dữ liệu bao gồm một hay nhiều giao dịch (transactions), mỗi giao dịch gồm một hay nhiều packets
- > Để hiểu được các giao dịch, các packet và nội dung của chúng -> cần tìm hiểu hai khái niệm Endpoint và Pipes



- Endpoint của thiết bị:

- Chỉ có thiết bị mới có Endpoint, Host không có Endpoint
- Endpoint là bộ đệm (gửi, nhận)
- Các Endpoint được đánh địa chỉ và xác định hướng
 - ✓ In Endpoint: bộ đệm gửi
 - ✓ Out Endpoint: bộ đệm nhận
- Tất cả các thiết bị đều phải có Endpoint 0, đây là endpoint mặc định để gửi các thông tin điều khiển



- Pipes: kết nối Endpoint của thiết bị tới Host
 - Phải thiết lập pipe trước khi muốn trao đổi dữ liệu
 - Host thiết lập pipe trong quá trình điểm danh (Enumeration)
 - Các Pipe sẽ được hủy khi thiết bị ngắt kết nối khỏi bus
 - Tất cả các thiết bị đều có một đường ống điều khiển (control pipe) mặc định sử dụng Endpoint 0



Device Classes

- Các thiết bị ngoại vi cùng chức năng (chuột, máy in, ổ nhớ flash...) có đặc tính truyền nhận dữ liệu chung -> Hệ điều hành có thể cung cấp driver chung cho các nhóm, các nhà sản xuất thiết bị không cần viết driver riêng.
- Các nhóm thiết bị đã được định nghĩa
 - Audio
 - Communication devices
 - **Human interface (HID)**
 - IrDA Bridge
 - **Mass Storage**
 - Cameras and scanners
 - Video



Quá trình trao đổi dữ liệu

- Các thiết bị USB có thể trao đổi dữ liệu với Host theo 4 kiểu hoàn toàn khác nhau, cụ thể:
 - Truyền điều khiển (control transfer)
 - Truyền ngắt (interrupt transfer)
 - Truyền theo khối (bulk transfer)
 - Truyền đẳng thời (isochronous transfer)



Các kiểu truyền

- **Truyền điều khiển:** để điều khiển phần cứng, các yêu cầu điều khiển được truyền. Chúng làm việc với mức ưu tiên cao và với khả năng kiểm soát lỗi tự động. Tốc độ truyền lớn vì có đến 64 byte trong một yêu cầu (request) có thể được truyền.
- **Truyền ngắn:** các thiết bị, cung cấp một lượng dữ liệu nhỏ, tuần hoàn chẵng hạn như chuột, bàn phím đều sử dụng kiểu truyền này. Hệ thống sẽ hỏi theo chu kỳ, chẵng hạn 10ms mỗi lần xem có các dữ liệu mới gửi đến.



Các kiểu truyền

- **Truyền theo khối:** khi có lượng dữ liệu lớn cần truyền và cần kiểm soát lỗi truyền nhưng lại không có yêu cầu thúc ép về thời gian truyền thì dữ liệu thường được truyền theo khối. VD: máy in, máy quét
- **Truyền đẳng thời:** khi có khối lượng dữ liệu lớn với tốc độ dữ liệu đã được quy định, ví dụ như card âm thanh. Theo cách truyền này một giá trị tốc độ xác định được duy trì. Việc hiệu chỉnh lỗi không được thực hiện vì những lỗi truyền lẻ tẻ cũng không gây ảnh hưởng đáng kể.



3.4. Lập trình giao tiếp USB Joystick





Cấu trúc JOYINFO trên Windows

- Windows định nghĩa cấu trúc JOYINFO để lưu các thông tin về tình trạng các nút bấm trên Joystick

```
typedef struct {  
    UINT wXpos;  
    UINT wYpos;  
    UINT wZpos;  
    UINT wButtons;  
} JOYINFO;
```

Nút trái, phải

Nút lên, xuống

Các nút chức
năng: 1, 2, 3, 4,
L1, L2, R1, R2,
Select, Start



- wXpos
 - wXpos=0 -> nút sang trái được bấm
 - wXpos=65535 -> nút sang phải được bấm
- wYpos
 - wYpos=0 -> nút lên được bấm
 - wYpos=65535 -> nút xuống được bấm
- wButtons: mỗi bit biểu diễn trạng thái của một nút chức năng
 - VD: Button 1 -> bit 0, Button 2 -> bit 1...



Cấu trúc js_event trên Linux

- Linux định nghĩa cấu trúc js_event để lưu các thông tin khi có phát sinh sự kiện (khởi tạo thiết bị, người dùng bấm nút chức năng, nút chỉnh hướng)
- Định nghĩa trong **include/linux/joystick.h**

```
struct js_event {  
    unsigned int time; /* event timestamp in milliseconds */  
    short value; /* value */  
    unsigned char type; /* event type */  
    unsigned char number; /* axis/button number */  
};
```



Cấu trúc js_event

- Nội dung các trường dữ liệu
 - **Time:** nhãn thời gian phát sinh sự kiện
 - **Value:** giá trị, phụ thuộc vào nút chức năng hay nút chỉnh hướng
 - ✓ Nút chức năng: 0/1
 - ✓ Nút chỉnh hướng: -32768 -> 32767
 - **Type:** loại sự kiện
 - ✓ Khởi tạo thiết bị: 0x80
 - ✓ Nút chỉnh hướng: 0x02
 - ✓ Nút chức năng: 0x01
 - **Number:** xác định nút được nhấn



Lập trình kết nối joystick

- Mở file thiết bị:

**joystick_fd = open(JOYSTICK_DEVNAME,
O_RDONLY | O_NONBLOCK);**

- JOYSTICK_DEVNAME: tên của file thiết bị, thường là /dev/input/js0
- O_RDONLY | O_NONBLOCK: mở file chỉ đọc ở chế độ NONBLOCK



Lập trình kết nối joystick

- **Đọc dữ liệu từ thiết bị (khi có phát sinh sự kiện)**

bytes = read(joystick_fd, jse, sizeof(*jse));

- joystick_fd: con trỏ file có được khi mở file
- jse: biến cấu trúc js_event
- bytes: Tổng số file đọc được, nếu số này bằng kích thước của cấu trúc js_event thì quá trình đọc thành công



Demo



Kết quả demo

■ Các sự kiện khi khởi tạo thiết bị

```
thuan@thuan-Inspiron-N4050: ~/Micro2440/code/joystick
File Edit View Search Terminal Help
thuan@thuan-Inspiron-N4050:~/Micro2440/code$ cd joystick/
thuan@thuan-Inspiron-N4050:~/Micro2440/code/joystick$ ls
huongdan.txt joystick.c joystick.h joytest joytestARM
thuan@thuan-Inspiron-N4050:~/Micro2440/code/joystick$ ./joytest
Event: time 4294736924, value      0, type: 129, axis/button: 0
Event: time 4294736928, value      0, type: 129, axis/button: 1
Event: time 4294736928, value      0, type: 129, axis/button: 2
Event: time 4294736928, value      0, type: 129, axis/button: 3
Event: time 4294736932, value      0, type: 129, axis/button: 4
Event: time 4294736932, value      0, type: 129, axis/button: 5
Event: time 4294736932, value      0, type: 129, axis/button: 6
Event: time 4294736936, value      0, type: 129, axis/button: 7
Event: time 4294736936, value      0, type: 129, axis/button: 8
Event: time 4294736936, value      0, type: 129, axis/button: 9
Event: time 4294736936, value      0, type: 129, axis/button: 10
Event: time 4294736940, value      0, type: 129, axis/button: 11
Event: time 4294736940, value      0, type: 130, axis/button: 0
Event: time 4294736940, value      0, type: 130, axis/button: 1
Event: time 4294736944, value      0, type: 130, axis/button: 2
Event: time 4294736944, value      0, type: 130, axis/button: 3
Event: time 4294736944, value      0, type: 130, axis/button: 4
Event: time 4294736944, value      0, type: 130, axis/button: 5
Event: time 4294736948, value      0, type: 130, axis/button: 6
```



Kết quả demo

■ Các sự kiện khi người dùng nhấn các nút

```
thuan@thuan-Inspiron-N4050: ~/Micro2440/code/joystick
File Edit View Search Terminal Help
Event: time 4294777004, value      2702, type: 2, axis/button: 1
Event: time 4294777012, value      2026, type: 2, axis/button: 1
Event: time 4294777020, value     -14527, type: 2, axis/button: 0
Event: time 4294777020, value      2364, type: 2, axis/button: 1
Event: time 4294777028, value     -9121, type: 2, axis/button: 0
Event: time 4294777028, value      2026, type: 2, axis/button: 1
Event: time 4294777028, value     -12162, type: 2, axis/button: 2
Event: time 4294777036, value      -338, type: 2, axis/button: 0
Event: time 4294777036, value       675, type: 2, axis/button: 1
Event: time 4294777036, value     -6081, type: 2, axis/button: 2
Event: time 4294777044, value        0, type: 2, axis/button: 0
Event: time 4294777044, value        0, type: 2, axis/button: 1
Event: time 4294777044, value        0, type: 2, axis/button: 2
Event: time 4294780132, value        1, type: 1, axis/button: 0
Event: time 4294780348, value        0, type: 1, axis/button: 0
Event: time 4294781156, value        1, type: 1, axis/button: 3
Event: time 4294781396, value        0, type: 1, axis/button: 3
Event: time 4294781892, value        1, type: 1, axis/button: 1
Event: time 4294782092, value        0, type: 1, axis/button: 1
Event: time 4294784396, value        1, type: 1, axis/button: 2
Event: time 4294784636, value        0, type: 1, axis/button: 2
Event: time 4294785488, value        1, type: 1, axis/button: 0
Event: time 4294785720, value        0, type: 1, axis/button: 0
```



QT Joystick Demo





3.5. Lập trình giao tiếp ADC

- Giới thiệu ADC
- Minh họa lập trình ADC



Giới thiệu ADC

- ADC: Analog to Digital Converter

- Thông số quan trọng của ADC
- Dải điện áp chuyển đổi
- ADC 8 bit, 10 bit, 12 bit...
- Bao nhiêu kênh?
- Độ phân ly



Minh họa lập trình ADC

- Khai báo thư viện

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/fs.h>
#include <errno.h>
#include <string.h>
```



Minh họa lập trình ADC

```
int main(void){  
    fprintf(stderr, "press Ctrl-C to stop\n");  
    int fd = open("/dev/adc", 0);  
    if (fd < 0) {  
        perror("open ADC device:");  
        return 1;  
    }  
    for(;;) {  
        char buffer[30];  
        int len = read(fd, buffer, sizeof buffer -1);  
        if (len > 0) {  
            buffer[len] = '\0';  
            int value = -1;  
            sscanf(buffer, "%d", &value);  
            printf("ADC Value: %d\n", value);  
        } else {  
            perror("read ADC device:");  
        }  
        return 1;  
    }  
    usleep(500* 1000); }  
    close(fd); }
```



Chương 4

Kỹ thuật lập trình nâng cao



Mục tiêu chương 4

- Sau khi kết thúc chương này, sinh viên có thể
 - Nắm được khái niệm tiến trình (process), quan hệ giữa các tiến trình
 - Trình bày được cơ chế sử dụng signal để giao tiếp giữa các tiến trình
 - Lập trình sử dụng kỹ thuật đa tiến trình
 - Trình bày khái niệm luồng
 - Lập trình ứng dụng đa luồng



- 4.1. Tiến trình (process)
- 4.2. Cơ chế sử dụng signal
- 4.3. Lập trình giao tiếp đa tiến trình
- 4.4. Luồng (thread)
- 4.5. Lập trình ứng dụng đa luồng



4.1. Tiến trình (Process)

- Khái niệm tiến trình
- Lập trình đa tiến trình





Khái niệm tiến trình

- Tiến trình được tạo ra khi ta thực thi một chương trình
- Đa tiến trình cho phép nhiều chương trình cùng thực thi và chia sẻ dữ liệu với nhau
- Các tham số của một tiến trình
 - PID (Process ID): số hiệu tiến trình
 - PPID (Parent Process ID): số hiệu tiến trình cha
 - Command: câu lệnh được gọi để thực thi tiến trình

ls -e -o pid,ppid,command



- **Lấy về PID: sử dụng hàm getpid()**
- **Lấy về PPID: sử dụng hàm getppid()**
- **Hàm getpid() và getppid() trả giá trị kiểu pid_t (bản chất là kiểu int)**

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

int main ()
{
    printf ("The process ID is %d\n", (int) getpid ());
    printf ("The parent process ID is %d\n", (int) getppid ());
    return 0;
}
```



Dừng tiến trình

- Cách 1: Sử dụng tổ hợp phím Ctrl + C
- Cách 2: Sử dụng shell command

kill PID



Tạo tiến trình mới

- **Cách 1:** sử dụng hàm system

```
#include <stdlib.h>

int main ( )
{
    int return_value;
    return_value = system ( "ls -l /" );
    return return_value;
}
```



Tạo tiến trình mới

■ Cách 2: sử dụng hàm fork và exec

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main ()
{
    pid_t child_pid;

    printf ("the main program process ID is %d\n", (int) getpid ());

    child_pid = fork ();
    if (child_pid != 0) {
        printf ("this is the parent process, with id %d\n", (int) getpid ());
        printf ("the child's process ID is %d\n", (int) child_pid);
    }
    return 0;
}
```



4.2. Cơ chế sử dụng signal

- Signal là cơ chế cho phép giao tiếp giữa các tiến trình
- Signal là cơ chế không đồng bộ
- Khi tiến trình nhận được signal, tiến trình phải xử lý signal ngay lập tức
- Linux hỗ trợ 32 SIGNAL



Danh sách signal thường dùng

Kiểu SIGNAL	Lý do gửi SIGNAL
SIGHUP	Báo cho chương trình khi thoát khỏi terminal
SIGINT	Khi người dùng nhấn Ctrl + C để tắt chương trình
SIGILL	Khi chương trình chạy lệnh không hợp lệ
SIGABRT	Khi chương trình nhận được lệnh abort
SIGKILL	Khi chương trình nhận được lệnh kill (đóng chương trình)
SIGUSR1	Tùy biến theo ứng dụng
SIGUSR2	Tùy biến theo chương trình



Gửi SIGNAL tới process

- **Cách 1:** sử dụng shell command

kill [-SIGNAL_TYPE] PID

- **Cách 2:** sử dụng hàm kill trong chương trình, cho phép process này gửi signal tới process khác

kill(PID, SIGNAL_TYPE)



4.3. Lập trình giao tiếp đa tiến trình

▪ Cơ chế:

- Tiến trình chính tạo ra các tiến trình con sử dụng lệnh fork và exec
- Sử dụng cơ chế signal để trao đổi tín hiệu giữa các tiến trình



- Bắt và xử lý signal được gửi tới một tiến trình (SIGTERM và SIGINT)



killsignal.c

```
int main ()
{
    struct sigaction sa, sa_term;
    printf ("The process ID is %d\n", (int) getpid ());
    printf ("The parent process ID is %d\n", (int) getppid ());
    //Thiet lap signal handler
    memset (&sa, 0, sizeof (sa));
    sa.sa_handler = &handler;
    sigaction (SIGINT, &sa, NULL);
    memset (&sa_term, 0, sizeof (sa_term));
    sa_term.sa_handler = &handler_term;
    sigaction (SIGTERM, &sa_term, NULL);
    //Start running
    while(1)
    {
        printf("\nIn main");
        sleep(1);
    }
    return 0;
}
```



Ví dụ: killsignal.c (tiếp)

```
sig_atomic_t sigusr1_count = 0;
void handler (int signal_number)
{
    FILE *fid=fopen ("/home/thuan/output.txt", "a");
    fprintf(fid, "\nThuan beo vo dich");
    fclose(fid);
    fprintf(stdout, "\nSIGUSR1 (number:%d) was raised %d times
\n", signal_number, sigusr1_count++);
    if(sigusr1_count>=3)
    {
        exit(0);
    }
}
void handler_term (int signal_number)
{
    fprintf(stdout, "\nReceive kill signal, signal number is %
d", signal_number);
}
```



Demo



4.4. Luồng (thread)

- Một chương trình mặc định chạy một luồng -> luồng chính
- Luồng chính có thể tạo ra các luồng khác, các luồng sẽ chạy đồng thời -> tăng tốc chương trình
- Các luồng chia sẻ không gian nhớ, truy xuất file và các tài nguyên khác
- Tham số của một luồng:
 - **thread ID:** số hiệu luồng (kiểu dữ liệu pthread_t)



4.5. Lập trình xử lý đa luồng

- Tạo luồng
- Truyền tham số cho luồng
- Nhận giá trị trả về từ luồng
- Tắt luồng



- Khai báo thư viện: pthread.h
- Hàm tạo luồng: pthread_create

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,  
                  void *(*start_routine) (void *), void *arg);
```

- ❖ **thread:** thread id
- ❖ **attr:** các thuộc tính của luồng, mặc định để NULL
- ❖ **start_routine:** hàm thực thi trong luồng
- ❖ **arg:** các tham số truyền cho luồng
- **Biên dịch chương trình:**
gcc –o multithread multithread.c -pthread



Mã nguồn tạo luồng

```
#include <pthread.h>
#include <stdio.h>
//thread function
void* print_xs (void* unused)
{
    while (1)
        fputc ('x', stdout);
    return NULL;
}
int main ()
{
    pthread_t thread_id;
    //Tao thread moi
    pthread_create (&thread_id, NULL, &print_xs, NULL);
    //Thuc hien main thread
    while (1)
        fputc ('o', stdout);
    return 0;
}
```



Truyền tham số cho luồng

- Khai báo cấu trúc dữ liệu chứa dữ liệu cần truyền cho luồng. Ví dụ:

```
struct arg {  
    //Ky tu can in  
    char character;  
    //So lan can in  
    int count; };
```

- Truyền dữ liệu cho luồng khi tạo luồng qua tham số **arg**
- Chương trình con thực thi luồng nhận tham số về và xử lý



Mã nguồn truyền tham số cho luồng

```
#include <pthread.h>
#include <stdio.h>
//thread function
void* print_xs (void* param)
{
    int i;
    struct arg* p=(struct arg*)param;
    for(i=0;i<p->count;i++)
        fputc (p->character, stdout);
    return NULL;
}
int main ()
{
    pthread_t thread_id1, thread_id2;
    struct arg arg1,arg2;
    //Tao thread moi
    arg1.character='a';
    arg1.count=10000;
    arg2.character='b';
    arg2.count=10000;
    pthread_create (&thread_id1, NULL, &print_xs, &arg1);
    pthread_create (&thread_id2, NULL, &print_xs, &arg2);
    //Thuc hien main thread
    while (1);
    return 0;
}
```



- Sử dụng hàm pthread_cancel:

```
int pthread_cancel(pthread_t thread);
```

- thread: nhận tham số thread id của luồng muốn tắt**



Mã nguồn tắt luồng

```
int count=0;
//thread function
void* print_xs (void* param)
{
    int i;
    struct arg* p=(struct arg*)param;
    while(1)
        fputc (p->character, stdout);
    return NULL;
}
int main ()
{
    pthread_t thread_id1, thread_id2;
    struct arg arg1,arg2;
    int thread_number,i;
    //Tao thread moi
    arg1.character='a';
    arg1.count=1000;
    arg2.character='b';
    arg2.count=200000;
    pthread_create (&thread_id1, NULL, &print_xs, &arg1);
    pthread_create (&thread_id2, NULL, &print_xs, &arg2);
    for(i=0;i<1000000;i++);
    pthread_cancel(thread_id1);
    printf("\nThread thu 1 da bi huy");
    pthread_cancel(thread_id2);
    printf("\nThread thu 2 da bi huy");
    return 0;
}
```



Thảo luận





Lập trình device driver



Nội dung

5.1. Giới thiệu về Kernel Module

5.2. Cơ chế xây dựng Device Driver

5.3. Tìm hiểu, tùy chỉnh một số driver đã có

5.4. Xây dựng usb device driver



5.1. Kernel Module

- Hoạt động trên Kernel Space, có thể truy xuất tới các tài nguyên của hệ thống
- Kernel Module cho phép thêm mới các module một cách linh hoạt, tránh việc phải biên dịch lại nhân hệ điều hành
- Kernel Module là cơ chế hữu hiệu để phát triển các device driver
- Xem danh sách các module đang chạy: **lsmod**



Kernel Module

- Các bước để thêm một kernel module vào hệ thống
 - Viết mã nguồn: chỉ sử dụng các thư viện được cung cấp bởi kernel, không sử dụng được các thư viện bên ngoài
 - Biên dịch mã nguồn module
 - Cài đặt module: dùng lệnh **insmod Tên_Module.ko**
 - Gỡ module: dùng lệnh **rmmod Tên_Module**
 - Xem các thông tin log: sử dụng System Log Viewer



Mã nguồn kernel Module

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
static int hello_init(void)
{
    printk(KERN_ALERT "Khoi tao thanh cong\n");
    return 0;
}
static void hello_exit(void)
{
    printk(KERN_ALERT "Ket thuc thanh cong\n");
}
module_init(hello_init);
module_exit(hello_exit);
```



Kernel Module Makefile

```
obj-m += hello.o
```

all:

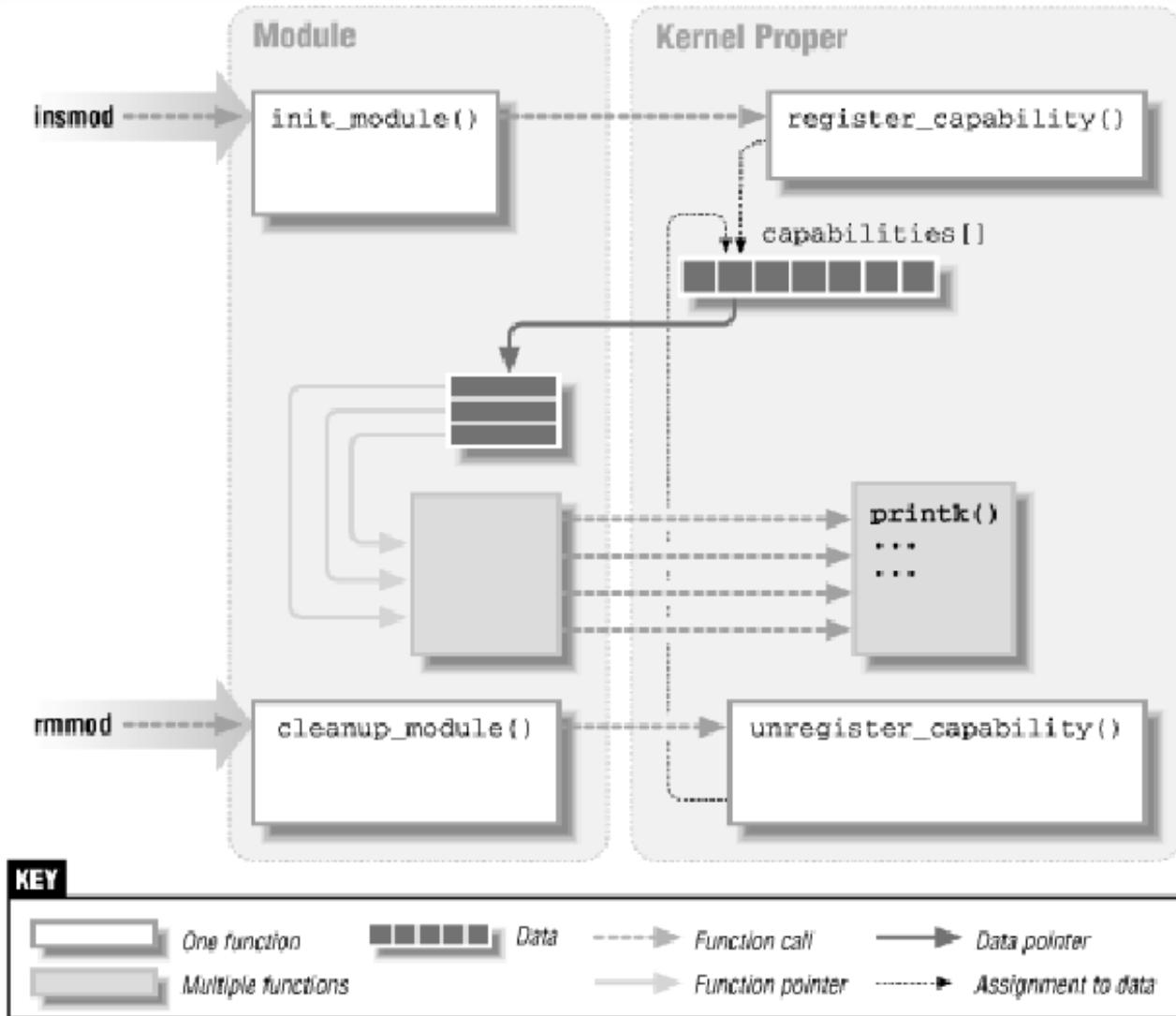
```
make -C /lib/modules/$(shell uname -r)/build  
M=$(PWD) modules
```

clean:

```
make -C /lib/modules/$(shell uname -r)/build  
M=$(PWD) clean
```



Liên kết Kernel Module





5.2. Device Driver

- Thêm các device driver theo cơ chế sử dụng Kernel Module
- Các thao tác thêm driver vào hệ thống
 - Viết mã nguồn (cấu trúc tương tự kernel Module).
 - Đăng ký Major ID
 - Biên dịch mã nguồn
 - Cài đặt sử dụng lệnh **insmod**
 - Sử dụng lệnh **mknod** để tạo device file trong /dev
mknod [options] NAME Type [Major Minor]

```

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
static ssize_t hello_read(struct file *filp, char *buffer, size_t length, loff_t * offset);
static int Major;
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = hello_read,
};
static int hello_init(void)
{
    Major = register_chrdev(0, "KTMT_Device", &fops);
    if (Major < 0) {
        printk(KERN_ALERT "Dang ky that bai, Major id=%d\n", Major);
        return Major;
    }
    else{
        printk(KERN_ALERT "Dang ky thiet bi thanh cong, Major id=%d\n", Major);
    }
    return 0;
}
static void hello_exit(void)
{
    unregister_chrdev(Major, "KTMT_Device" );
}
static ssize_t hello_read(struct file *filp, char *buffer, size_t length, loff_t * offset)
{
    return 100;
}
module_init(hello_init);
module_exit(hello_exit);

```



Demo

- Ví dụ 1: Chỉnh sửa driver sẵn có
 - Chỉnh sửa driver điều khiển led, bổ sung thêm hàm write để điều khiển trực tiếp tất cả các led đơn trên KIT
- Ví dụ 2: Tạo driver mới theo cơ chế kernel module



Thảo luận





Lập trình nền tảng QT



Mục tiêu bài học số 6

- Sau khi kết thúc bài học này, sinh viên có thể
 - Nắm được các vấn đề cơ bản, đặc trưng của nền tảng Qt
 - Cài đặt Qt Creator (Qt SDK) trên máy phát triển (Ubuntu)
 - Làm quen với lập trình ứng dụng giao diện đồ họa sử dụng nền tảng Qt
 - Cài đặt Qt Everywhere để phát triển ứng dụng cho nền tảng Arm Embedded Linux



Nội dung bài học

6.1. Giới thiệu QT

6.2. Cài đặt môi trường phát triển Qt

6.3. Làm quen với lập trình QT

6.4. Cài đặt Qt Everywhere (Qt Embedded)



6.1. Giới thiệu Qt

- Qt Development Frameworks được sáng lập năm 1994 bởi TrollTech
- 2008: TrollTech sáp nhập vào Nokia
- Qt là một Framework phát triển ứng dụng đa nền tảng (desktop, mobile, embedded).
- Hỗ trợ các nền tảng: Windows, Linux, Embedded Linux, Win CE, Symbian, Maemo...



Giới thiệu QT

- Qt cho phép viết ứng dụng một lần và biên dịch chéo trên nhiều nền tảng hệ điều hành khác nhau mà không phải viết lại mã. Tuy nhiên, mã nguồn cần được biên dịch trên nền tảng mà muốn ứng dụng được thực thi.
- Lập trình Qt theo chuẩn C++.



Giới thiệu QT

- Qt Framework bao gồm:
 - a cross-platform class library (Thư viện các lớp hướng đối tượng)
 - integrated development tools (Các công cụ phát triển tích hợp)
 - a cross-platform IDE. (Môi trường phát triển ứng dụng)
- Tham khảo: qt.nokia.com; qtcentre.org



QT được sử dụng rộng rãi

From embedded devices to desktop applications



By companies from many industries





Kiến trúc Qt

Qt SDK

Class library

Core
GUI
WebKit
Graphic View
Scripting
OpenGL

XML
Multimedia
Database
Network
Unit Tests
Benchmarking

Development tools

Cross-platform IDE - Qt Creator
GUI Designer Help System
I18n Tools Build Tool

Cross-platform support

Windows Mac OS X Linux/X11 Embedded Linux Win CE/Mobile Maemo Symbian

Flexible Licensing

GPL, LGPL, Commercial

Qt Services

Support, Training, Consulting



6.2. Cài đặt Qt SDK

- Cài đặt Qt SDK trên máy phát triển (Linux, Windows, MacOS)
- File cài đặt

qt-sdk-linux-x86-opensource-2010.05.1.bin

(<http://qt.nokia.com/downloads>)

- Thực thi file cài đặt:

\$./qt-sdk-linux-x86-opensource-2010.05.1.bin

- Đợi quá trình cài đặt diễn ra thành công, mặc định thư mục cài đặt chứa tại

\$HOME/qt-sdk-2010.01/qt/bin



Cài đặt Qt SDK

- Sau khi cài đặt xong Qt SDK, công cụ Qt Creator cho phép phát triển ứng dụng với lựa chọn mặc định biên dịch trên máy tính Linux. Để biên dịch chéo ứng dụng thực thi trên KIT FriendlyArm (Embedded Linux) cần cài đặt Qt Everywhere



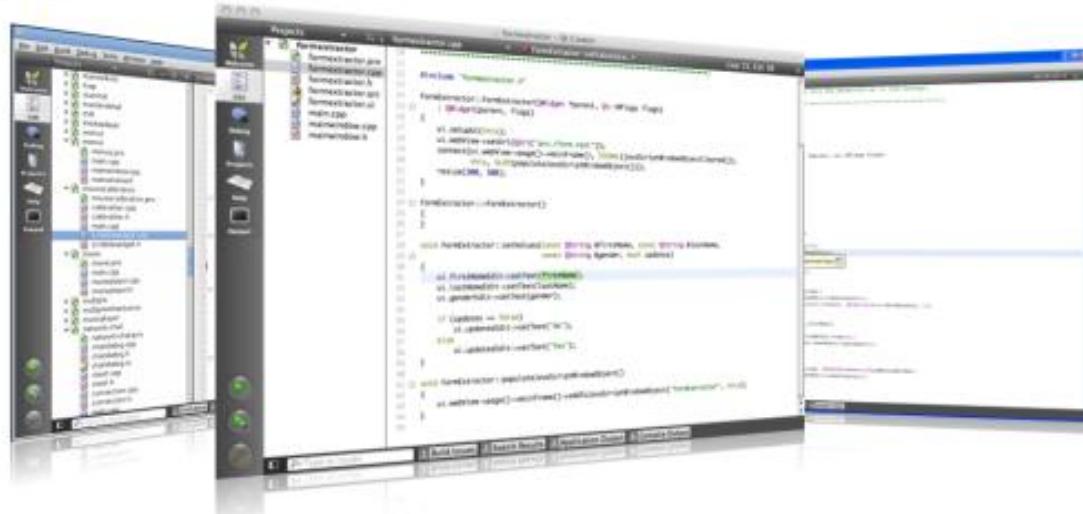
6.3. Làm quen với lập trình QT

- Sử dụng môi trường phát triển Qt Creator (IDE)
- Chương trình HelloQt
- Cơ chế Signals/Slot
- Quản lý layout



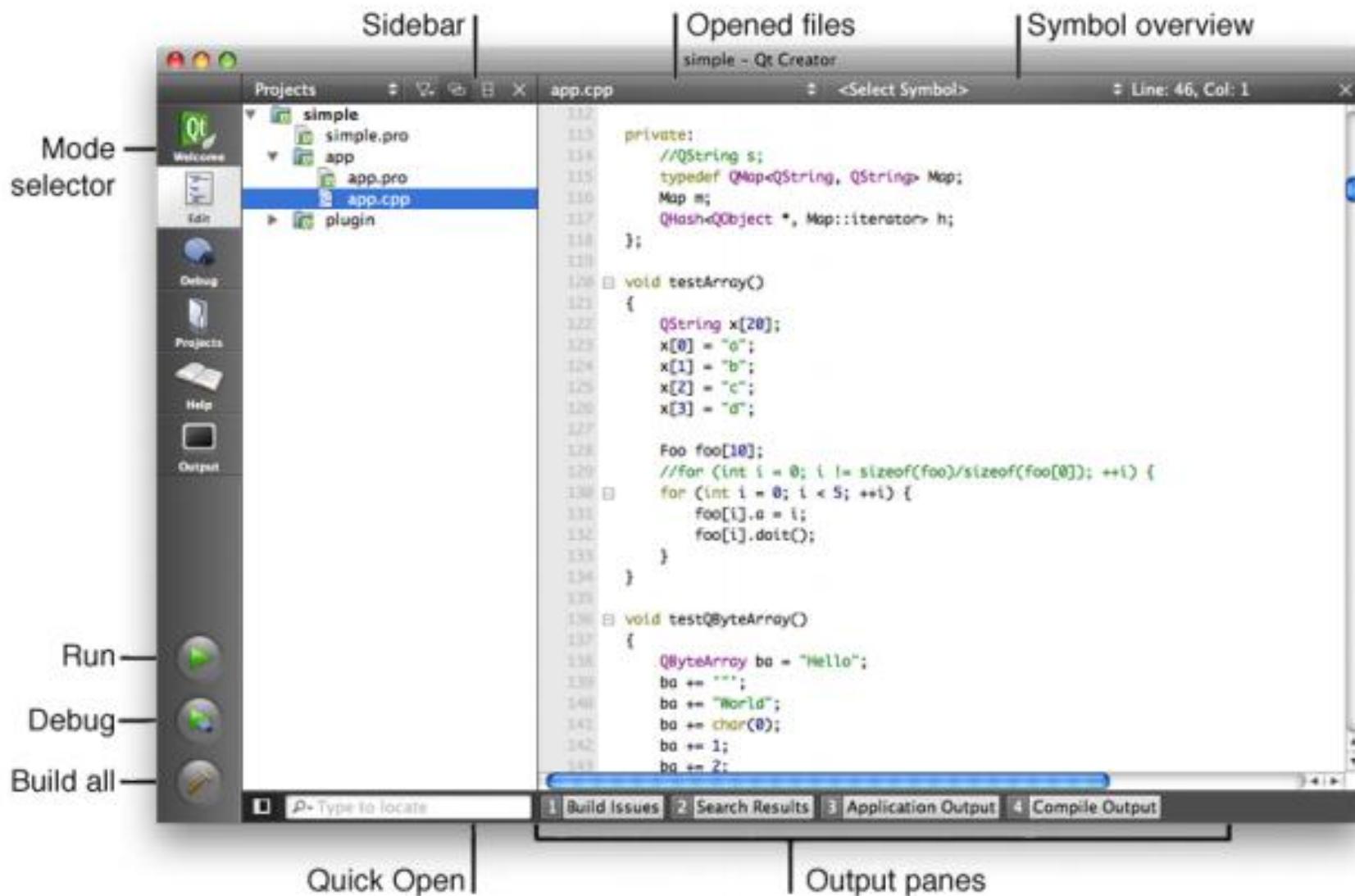
Môi trường phát triển Qt Creator IDE

- Advanced C++ code editor
- Integrated GUI layout and forms designer
- Project and build management tools
- Integrated, context-sensitive help system
- Visual debugger
- Rapid code navigation tools
- Supports multiple platforms





Các thành phần Qt Creator





Các điều khiển (widgets) cơ bản

- QLabel
- QPushButton
- QLineEdit
- QTextEdit
- QSpinBox
- QComboBox
- QSlider
- V.v...



Chương trình HelloQt

- Tạo project HelloQt
- Trong file main.c bổ sung đoạn mã:
- Biên dịch, chạy chương trình:

```
#include <QApplication>
#include <QLabel>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```





Chương trình HelloQt

- Giải thích ?
- Sửa đoạn mã với HTML style



```
#include <QApplication>
#include <QLabel>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    //QLabel *label = new QLabel("Hello Qt!");
    QLabel *label = new QLabel("<h2><i>Hello</i> "
        "<font color=red>Qt!</font></h2>");
    label->show();
    return app.exec();
}
```



Cơ chế signals - slot

- Cơ chế event - handler
- Xử lý các sự kiện (sự kiện tương tác người dùng, sự kiện của hệ thống)
- Cho phép tạo các kết nối (connections) giữa sự kiện (signals) với hàm xử lý (slot)
- Có 2 cách tạo:
 - Tạo tự động (wizard)
 - Tạo bằng tay (manual, hand-code)



Minh họa cơ chế signals/slot

- Tạo bằng code (dùng phương thức Qobject::connect)

```
1 #include <QApplication>
2 #include <QPushButton>

3 int main(int argc, char *argv[])
4 {
5     QApplication app(argc, argv);
6     QPushButton *button = new QPushButton("Quit");
7     QObject::connect(button, SIGNAL(clicked()),
8                      &app, SLOT(quit()));
9     button->show();
10    return app.exec();
11 }
```

Figure 1.3. The Quit application





Minh họa cơ chế Signals/Slot

- Tạo bằng code

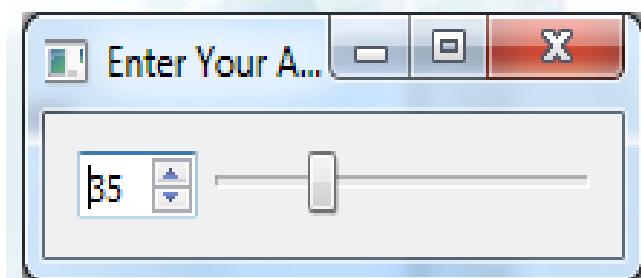
`connect(sender, SIGNAL(signal), receiver, SLOT(slot));`

- Trong đó: sender, receiver là con trỏ Qobjects, signal và slot là các tên hàm không có tham số.

- Các macro SIGNAL() và SLOT() biến đổi tham số thành string.

VD: đồng bộ giữa 2 điều khiển slider và spinBox

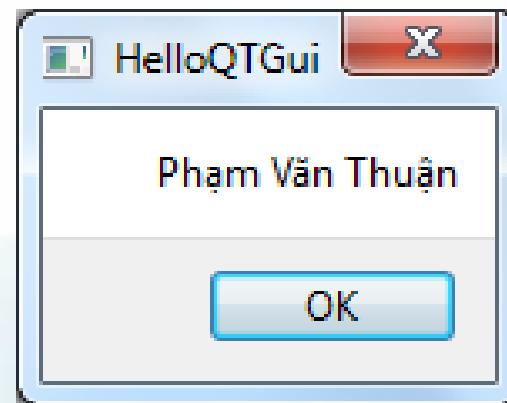
```
QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                  slider, SLOT(setValue(int)));
QObject::connect(slider, SIGNAL(valueChanged(int)),
                  spinBox, SLOT(setValue(int)));
```





Minh họa cơ chế Signals/Slot

- Tạo tự động (wizard): chuột phải vào đối tượng muốn xử lý sự kiện, chọn Go to slot, tìm slot là hàm xử lý sự kiện tương ứng muốn dùng.
- Ví dụ xử lý sự kiện nút bấm (QPushButton)





Quản lý layout trong ứng dụng Qt

- Kỹ thuật lay out: Cho phép sắp xếp các điều khiển (widgets) trên một form. Kích thước và vị trí sẽ thay đổi linh hoạt khi form thay đổi kích thước.
- Có các kiểu lay out:
 - Horizontal lay out
 - Vertical lay out
 - Grid lay out
 - Form lay out



Chương trình TextFinder

■ Xây dựng ứng dụng TextFinder

The screenshot shows a window titled "TextFinder". In the top-left corner, there is a "Keyword" input field containing the text "lay out". To its right is a red "Find" button with a white cursor arrow pointing to it. Below the input field, the text "Su dung lay out trong Qt" is highlighted in a yellow box. The main content area contains several paragraphs of Vietnamese text about layout management in Qt:

Viec sap xep cac dieu khien (widget) tren form su dung layout cho phep giao dien (UI) cua ung dung co the co gian linh hoạt theo kich thuoc man hinh.
Kich thuoc cua cac dieu khien se duoc thay doi theo kich thuoc form chua.
Co the su dung nhieu kieu lay out (vertical, horizontal, grid layout, form layout)
Ket hop su dung cac thuoc tinh sizePolicy, minimumSize, maximumSize (trong cua so Properties) de thiet lap cac rang buoc ve kich thuoc cho cac dieu khien khi duoc layout.



Qt Documentations

- Documentation in Qt Assistant (or QtCreator)
- Qt's examples
- Qt developer network:
 - <http://developer.qt.nokia.com/>
- Qt Center Forum:
 - <http://www.qtcn.org>



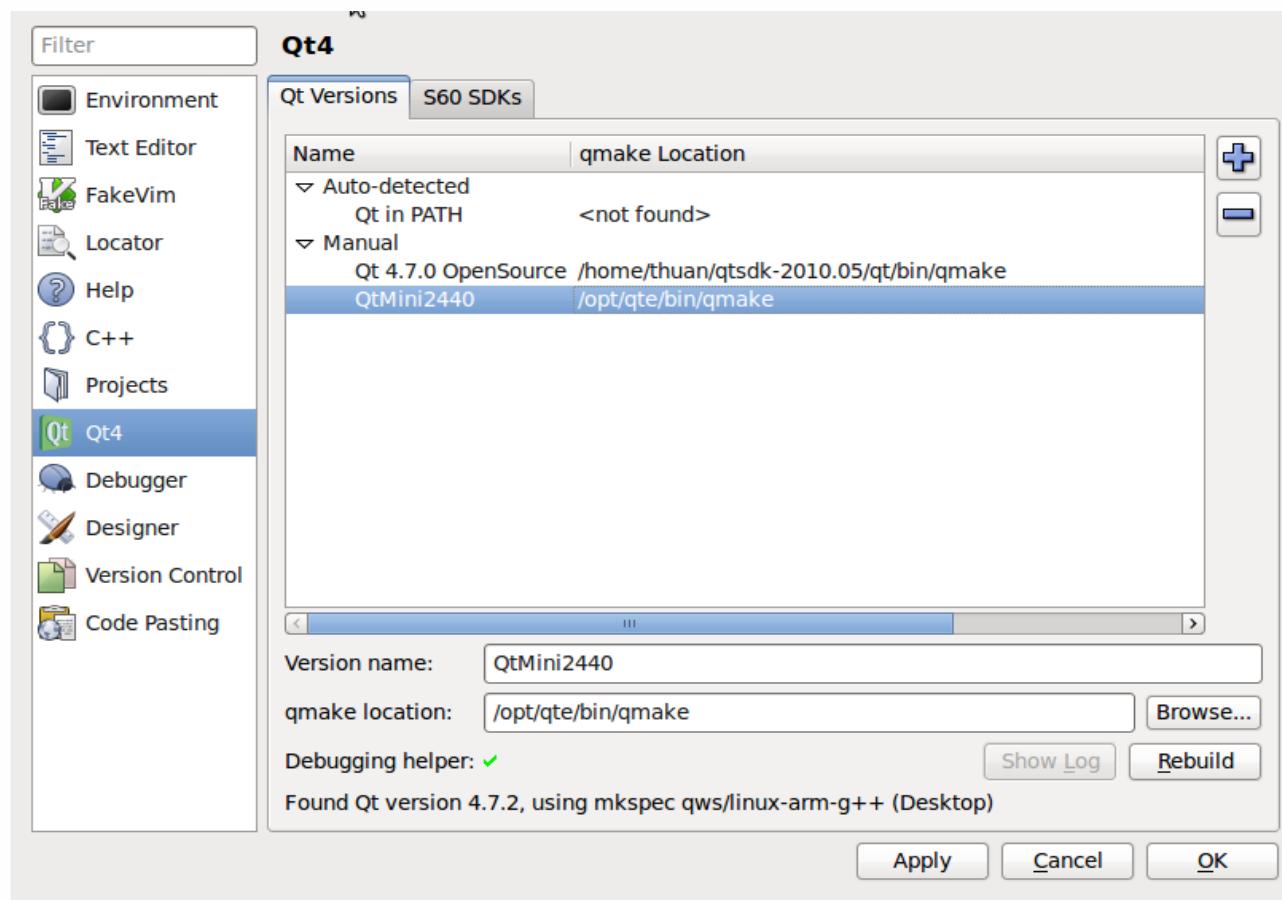
6.4. Cài đặt Qt Everywhere

- **Bước 1:** Cài đặt QT Embedded (QT Everywhere)
(Xem hướng dẫn chi tiết kèm theo)
- **Bước 2:** Copy các file thư viện cần thiết xuống KIT
 - 3 thư viện quan trọng (VD: copy xuống thư mục /opt/qte/lib)
 - ✓ libQtCore.so.4
 - ✓ libQtGui.so.4
 - ✓ libQtNetwork.so.4
 - Copy các fonts vào thư mục /opt/qte/lib/fonts
- **Bước 3:** Chỉnh file cấu hình /etc/init.d/rcS, tắt Qtopia để tránh tranh chấp



Cấu hình trình dịch Qmake cho Kit

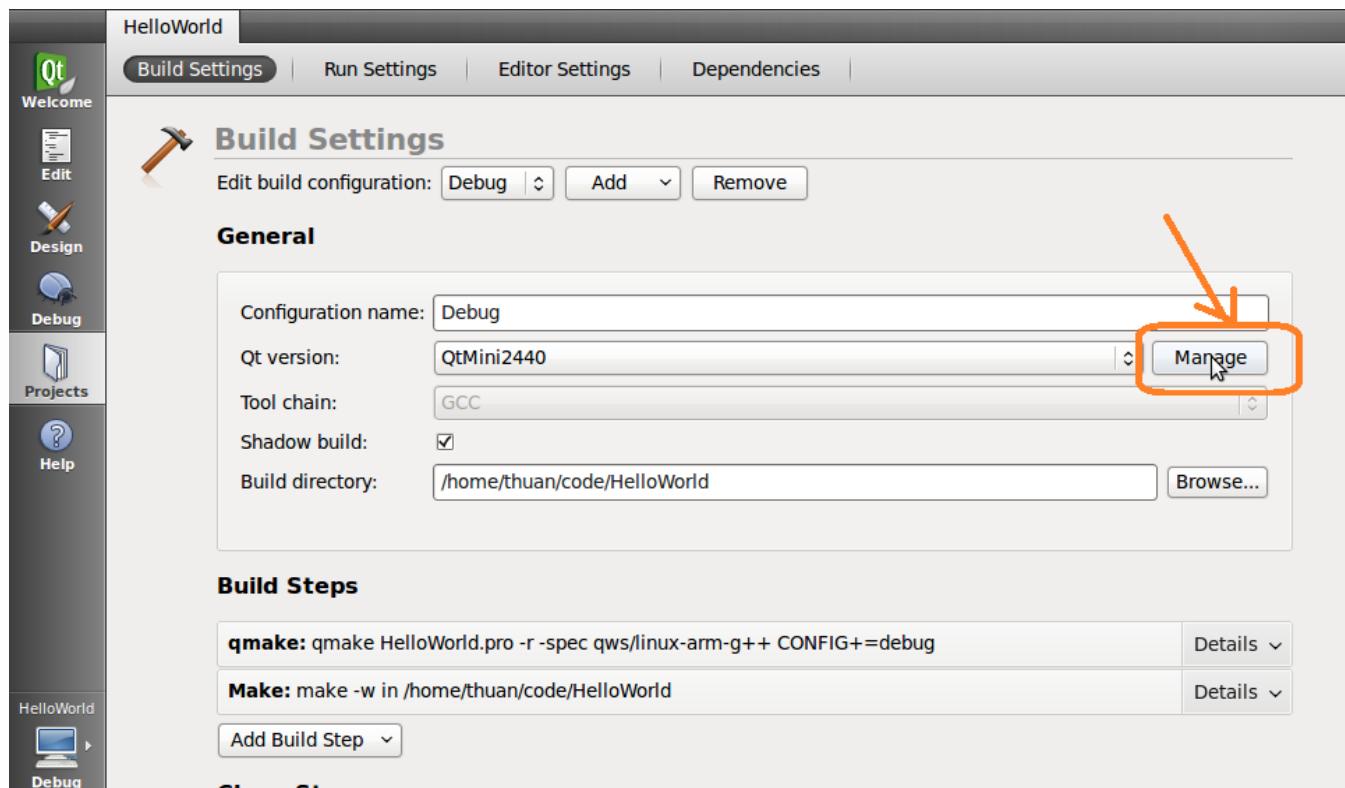
- **Bước 4:** Tạo cấu hình biên dịch cho Mini2440, trỏ tới Qmake đã biên dịch được ở trên





Cấu hình trình dịch Qmake cho Kit

- **Bước 5: Dịch chương trình QT cho KIT**
 - Chọn đúng bộ biên dịch Qmake cho QT Embedded





Thảo luận





Lập trình mạng trên Linux nhúng



Mục tiêu bài học số 7

- Sau khi kết thúc bài học này, học viên có thể
 - Xây dựng ứng dụng giao diện, sử dụng các điều khiển (widgets)
 - Vận dụng kỹ thuật quản lý layout để sắp xếp các điều khiển trên form
 - Vận dụng cơ chế xử lý sự kiện (signal/slot)
 - Lập trình socket trên nền Linux nhúng
 - Lập trình mạng với Qt



Nội dung bài học

- 7.1. Lập trình socket trên Linux nhúng
- 7.2. Thư viện lập trình mạng trên Qt
- 7.3. Lập trình ứng dụng ChatRoom
- 7.4. Lập trình ứng dụng gửi/nhận ảnh qua socket



7.1. Lập trình Socket trên Linux

- Giới thiệu lập trình socket
- Mô hình lập trình
- Minh họa



Giới thiệu lập trình socket

- Socket: Kết nối đầu cuối giữa 2 tiến trình/2 máy qua mạng (mô hình client/server)
- Tiến trình client kết nối đến tiến trình server yêu cầu trao đổi dữ liệu
- Client cần biết về địa chỉ và sự tồn tại của server, trong khi server không cần biết về client cho đến khi nó được kết nối đến.
- Mỗi khi thiết lập kết nối, cả 2 bên có thể gửi và nhận dữ liệu
- *Liên hệ như kết nối trong một cuộc gọi điện thoại*



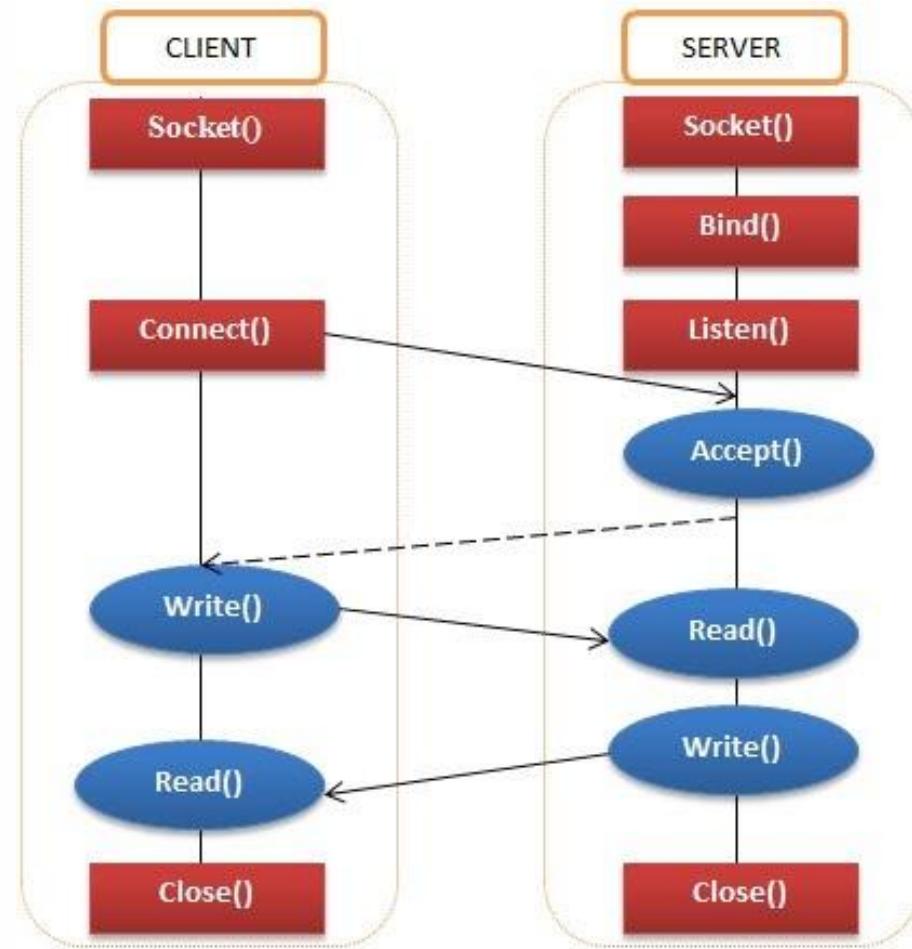
Giới thiệu lập trình socket

- Các hệ thống (Windows, Linux, ...) đều cung cấp các hàm hệ thống lập trình socket
- Có 2 loại socket sử dụng rộng rãi:
 - Stream socket
 - Datagram socket
- Stream sockets: Dựa trên giao thức TCP (Transmission Control Protocol), là giao thức hướng luồng (stream oriented).
- Datagram sockets: Dựa trên giao thức UDP (User Datagram Protocol), là giao thức hướng thông điệp (message oriented)



Mô hình lập trình socket

- Mô hình lập trình socket TCP giữa 2 tiến trình client/server





Chương trình minh họa

- 2 tiến trình (Mã nguồn tham khảo):
 - server.c
 - client.c
- Biên dịch và chạy 2 chương trình này (trên cùng một máy local host, hoặc 2 máy riêng biệt kết nối mạng)



Demo

- Lập trình giao tiếp socket giữa KIT micro 2440 và PC



Demo



7.2. Thư viện lập trình mạng trên QT

- QtNetwork

- QTcpSocket
- QUdpSocket
- QTcpServer
- QFtp: làm việc với giao thức truyền file FTP
- QHttp: làm việc với giao thức Http

(Xem Qt documentation)



7.3. Chương trình ChatRoom

Server 6789

Nick

<hungpn> Hello server, i'm hungpn
<anonymous> hi hungpn
<anonymous> my message is sent broadcast by server
<hungpn> hi anonymous
<hungpn> i've got ur mesg

Message

Server 6789

Nick

<anonymous> hi hungpn
<anonymous> my message is sent broadcast by server
<hungpn> hi anonymous
<hungpn> i've got ur mesg

Message



7.4. Chương trình gửi/nhận ảnh

- Lập trình socket client/server
- Sử dụng lớp QImage



Thảo luận





Chương 8

Lập trình xử lý ảnh trên nền nhúng



Nội dung

- 8.1. Tổng quan về xử lý ảnh
- 8.2. Giới thiệu OpenCV
- 8.3. Các phép biến đổi ảnh cơ bản



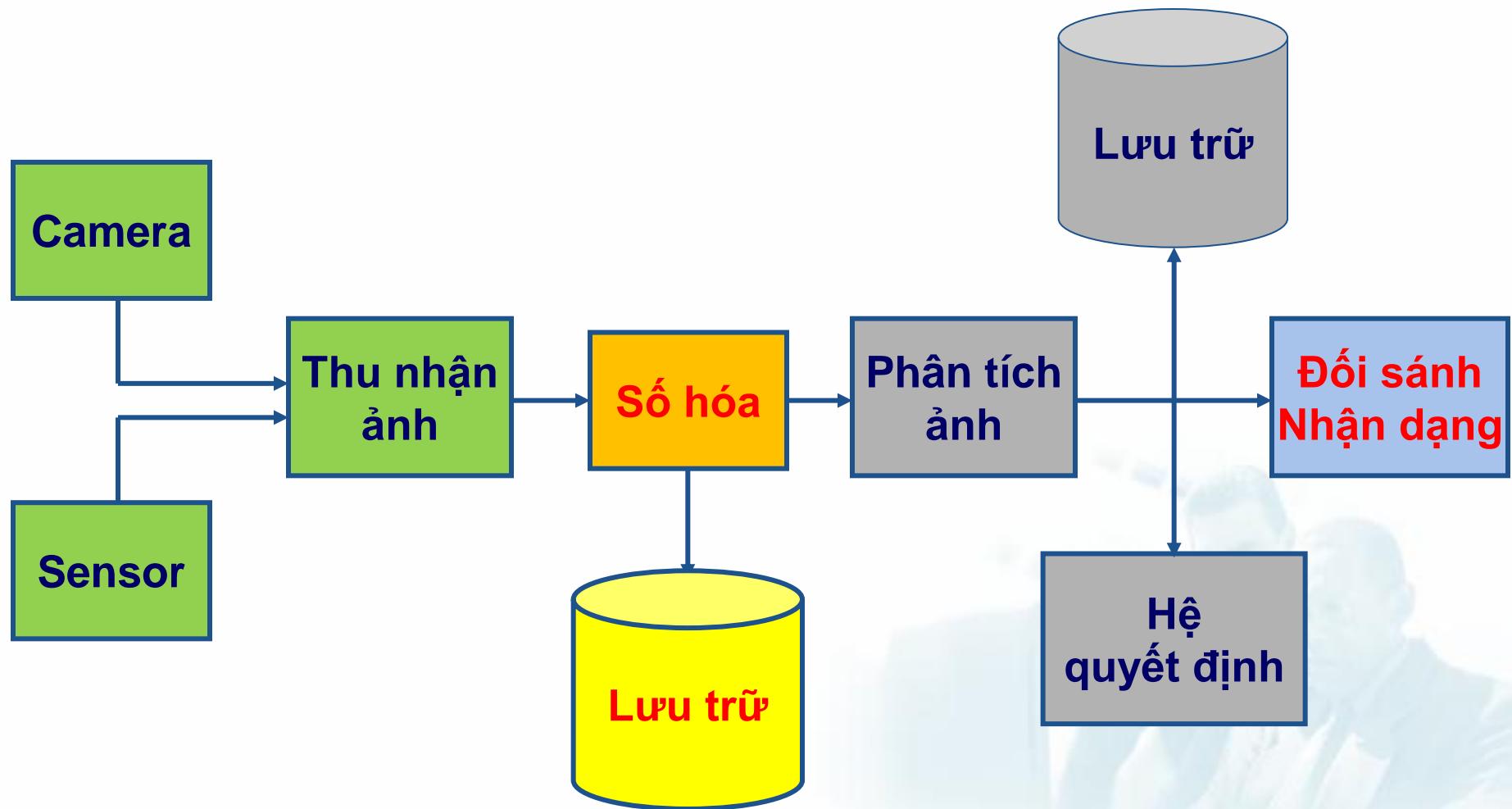
8.1. Tổng quan xử lý ảnh

Khái niệm xử lý ảnh

- Nâng cao chất lượng hình ảnh theo một tiêu chí nào đó (Cảm nhận của con người)
- Phân tích ảnh để thu được các thông tin đặc trưng giúp cho việc phân loại ảnh (**image classification**), nhận dạng ảnh (**image recognition**).
- Hiểu ảnh đầu vào để có những mô tả về ảnh ở mức cao hơn, sâu hơn.

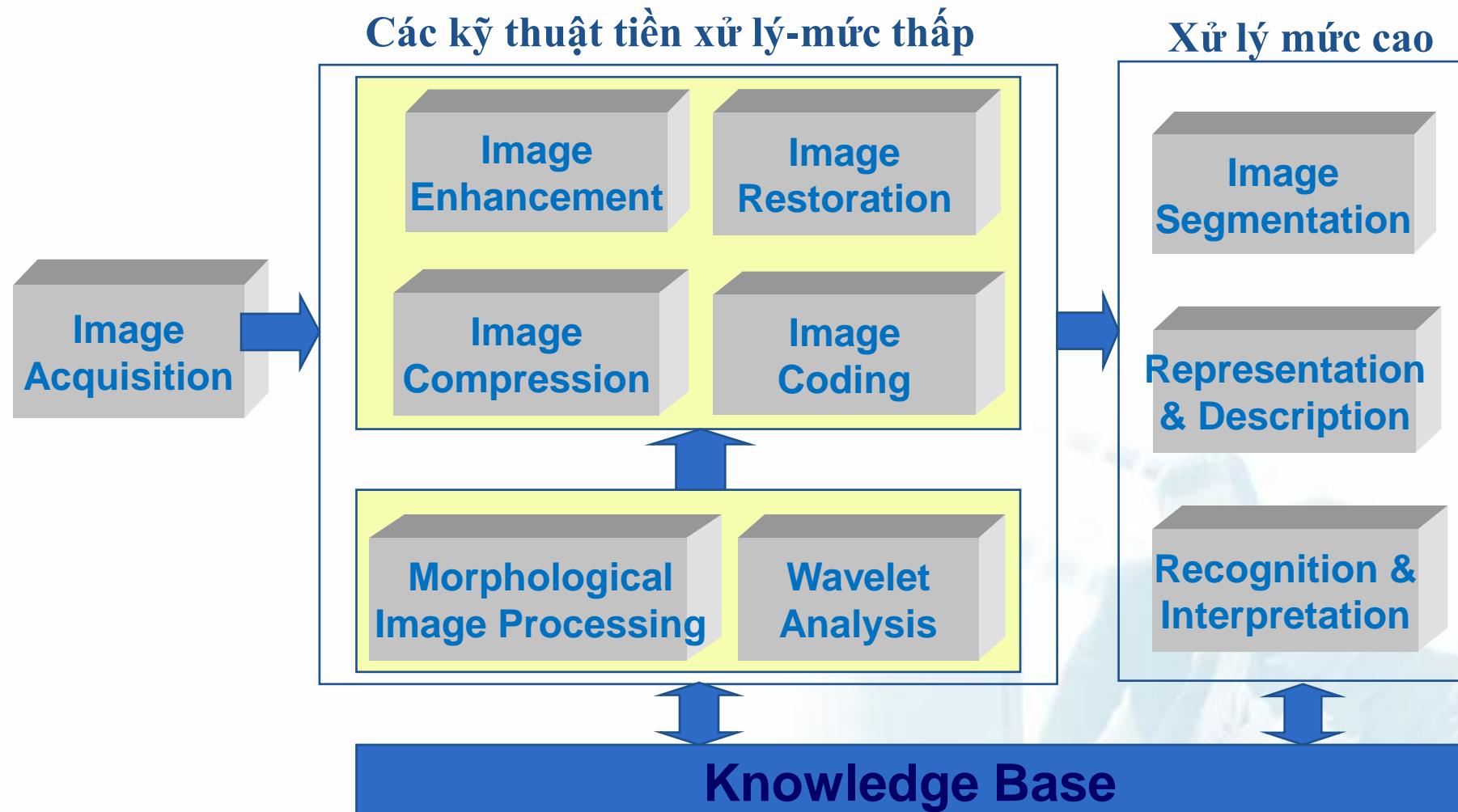


Mô hình hệ thống xử lý ảnh





Các bài toán xử lý ảnh





Các bài toán xử lý ảnh

- **Thu nhận ảnh, số hóa ảnh (image aquisition)**
 - Hệ thống chụp ảnh, tín hiệu ảnh
 - Hệ thống số hóa ảnh: Các phương pháp lấy mẫu, lượng tử hóa
- **Cải thiện ảnh, khôi phục ảnh, lọc nhiễu (tiền xử lý – image pre-processing)**
 - Các phép xử lý điểm ảnh
 - Các phép xử lý trên miền không gian
 - Các phép xử lý trên miền tần số



Các bài toán xử lý ảnh

▪ Phân tích ảnh

- Trích chọn đặc trưng (feature extraction)
- Biểu diễn, mô tả ảnh (image representation, image description)
- Phân lớp ảnh (image classification)
- Nhận dạng ảnh (image recognition)
- ...

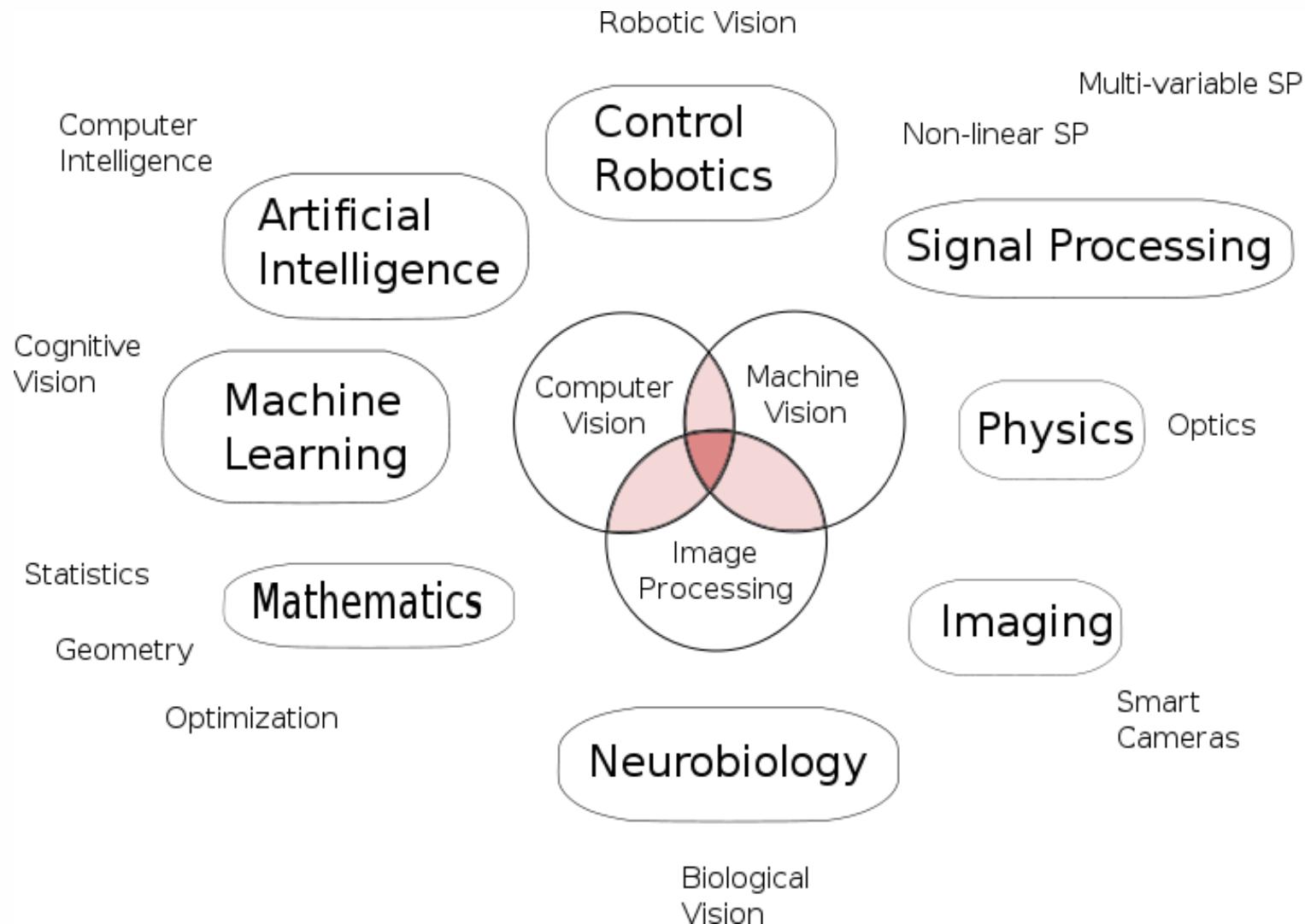
▪ Mã hóa, nén ảnh

- Các phương pháp nén ảnh, các chuẩn nén ảnh

▪ Truyền thông ảnh: các kỹ thuật streaming



Ứng dụng xử lý ảnh





Ứng dụng tăng cường chất lượng



Original



Automatic Enhancement



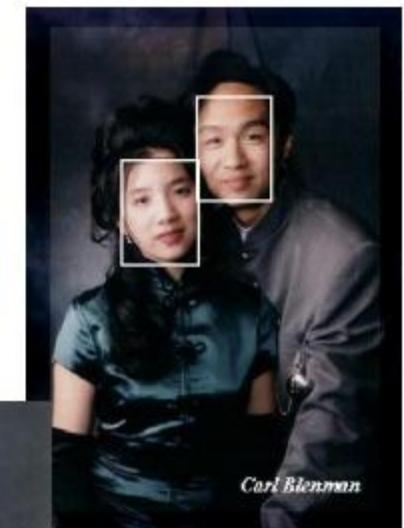
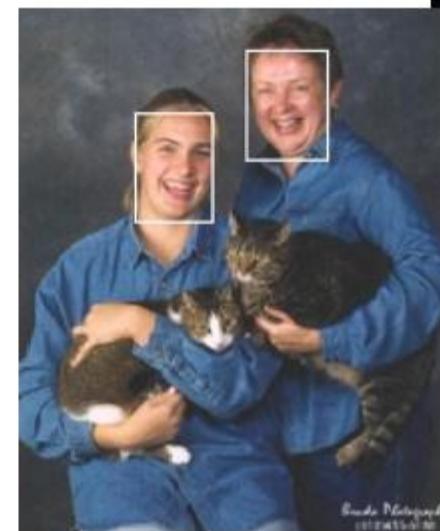
Hiệu ứng panorama





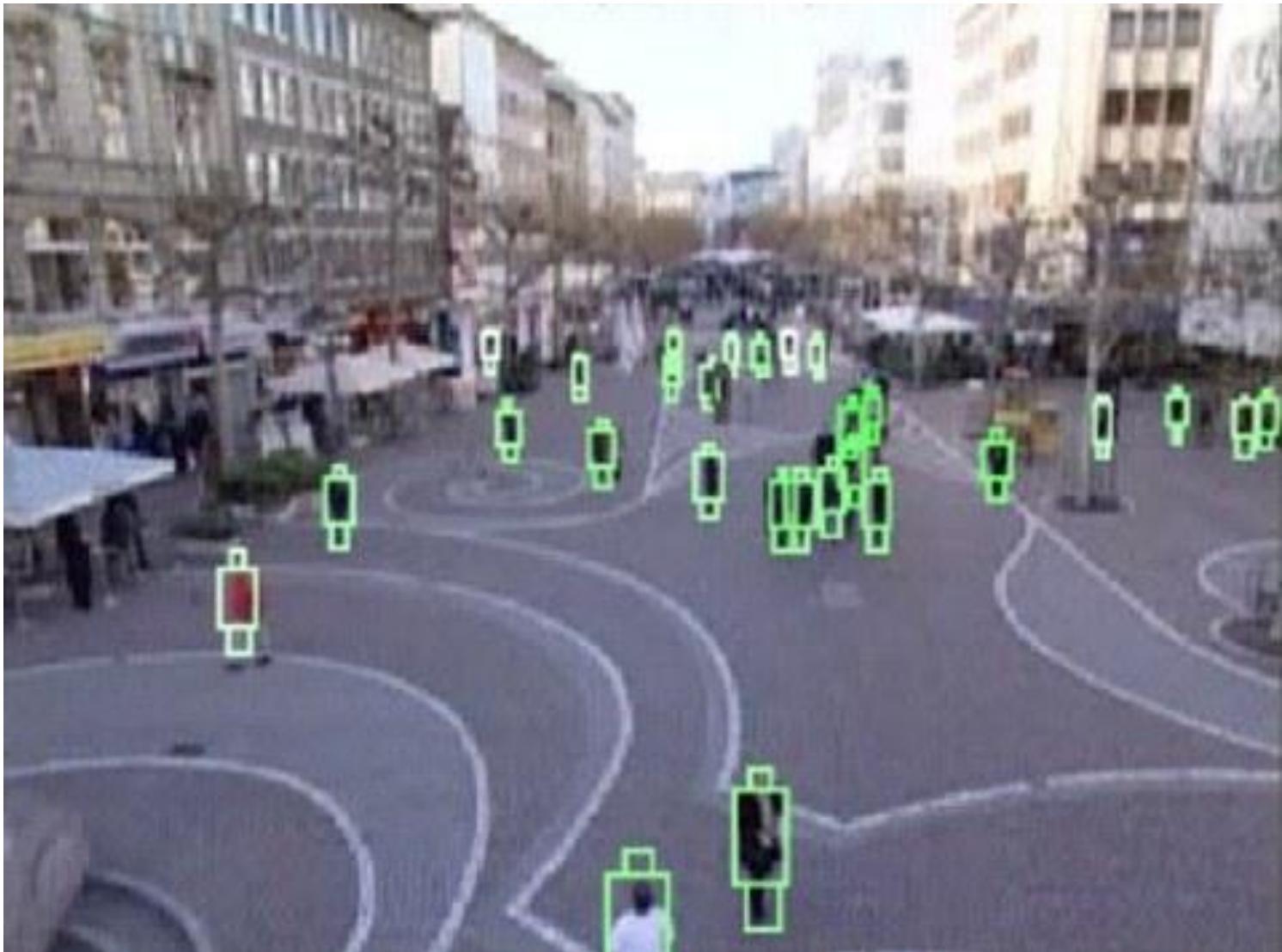
Ứng dụng nhận dạng khuôn mặt

Face Detection





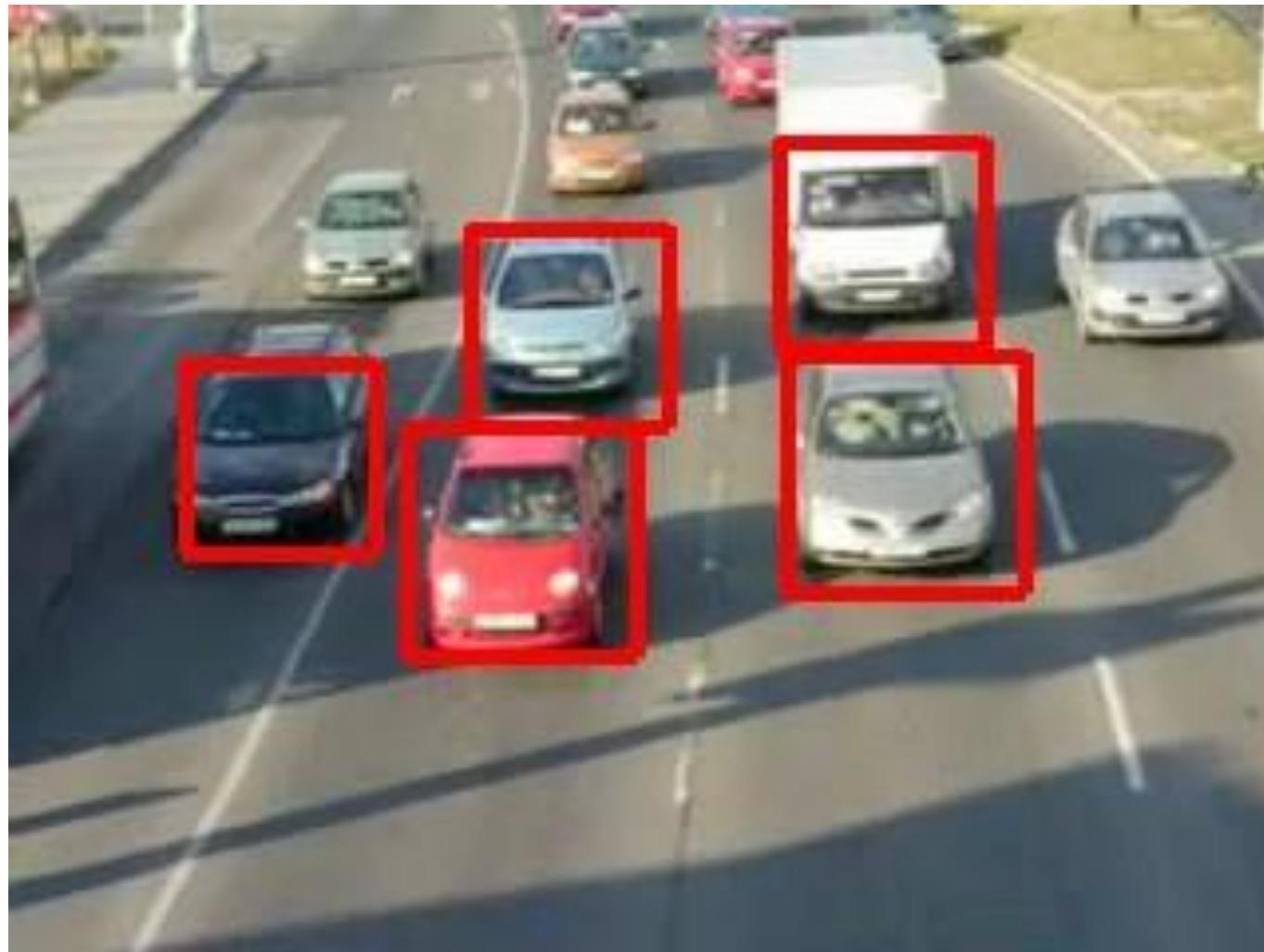
Nhận dạng người chuyển động



Lập trình nhúng ARM-Linux

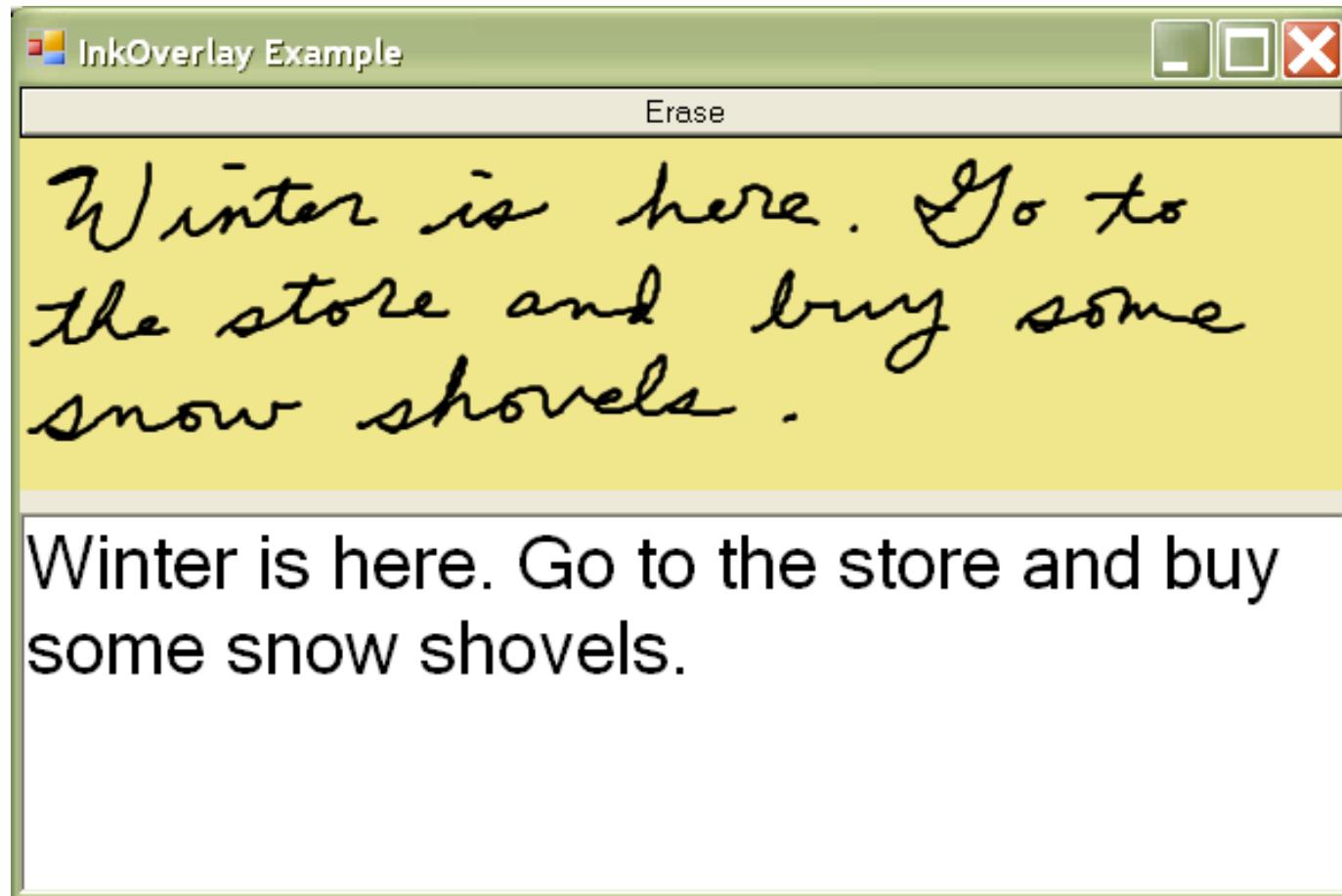


Nhận dạng đối tượng chuyển động





Nhận dạng chữ viết tay

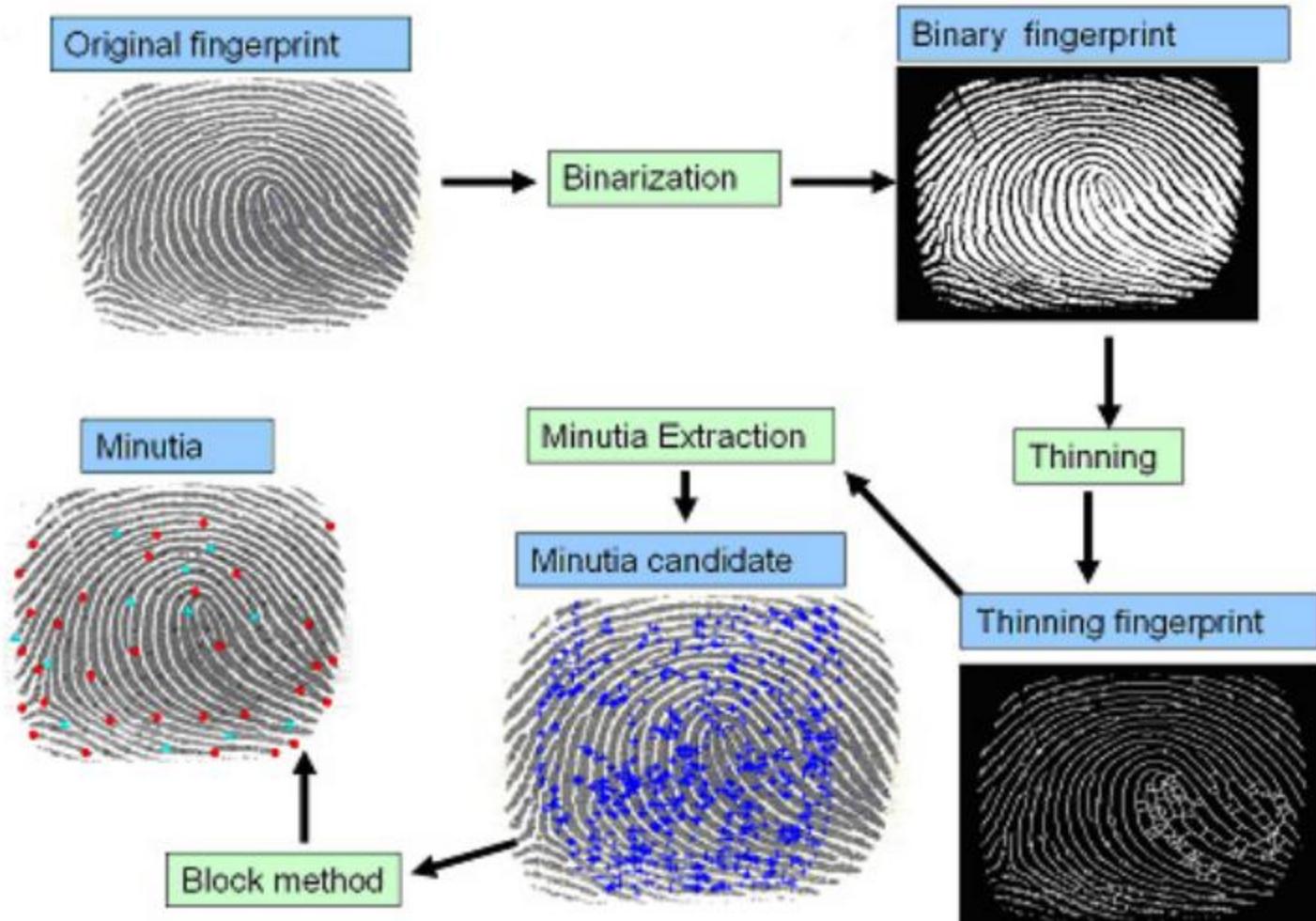




Nhận dạng vân tay

Biometrics: Fingerprint recognition

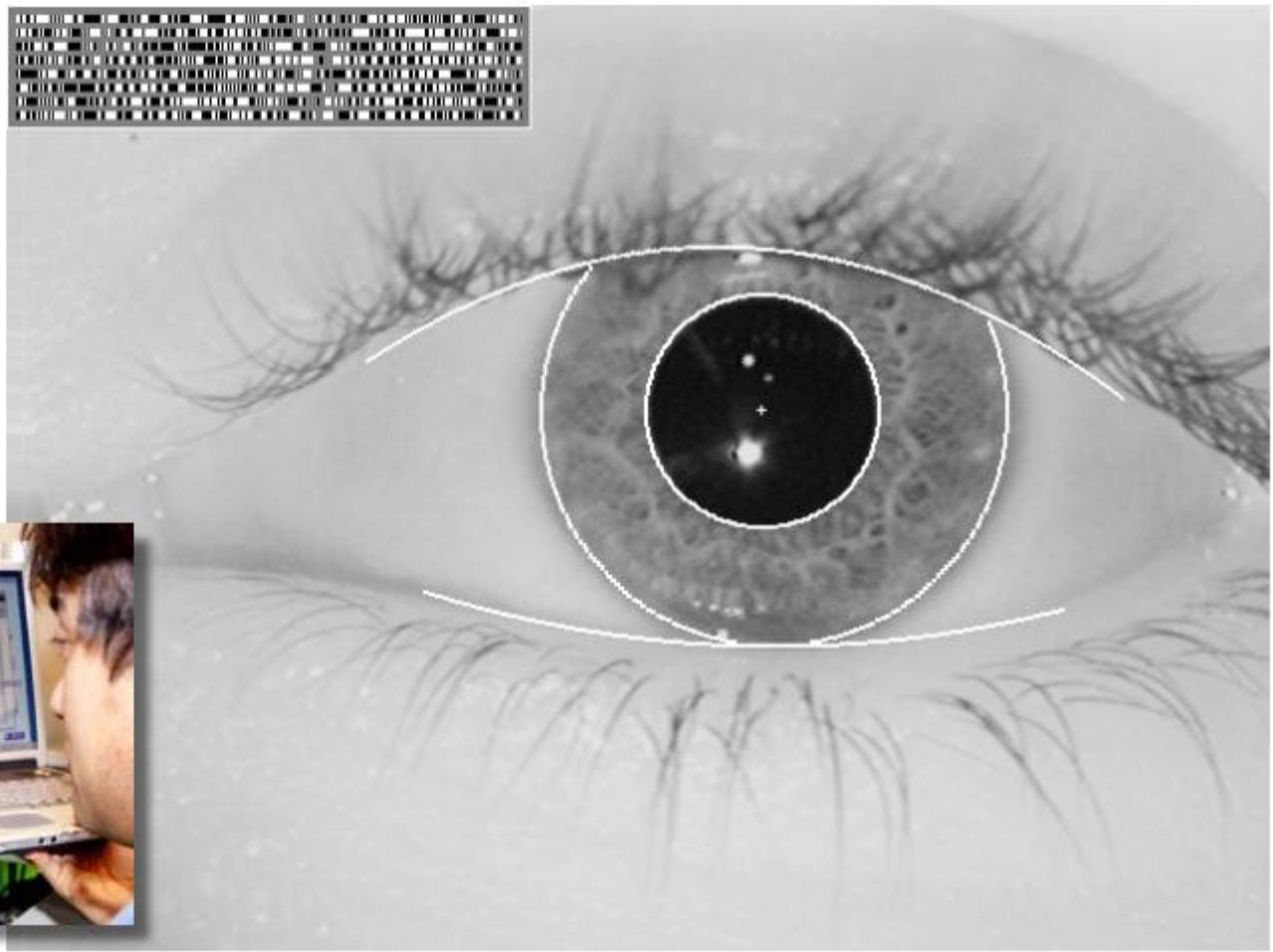
FBI's
Integrated
Automated
Fingerprint
Identification
System
IAFIS





Nhận dạng mống mắt (iris)

Biometrics: Iris recognition





Mô hình hóa 3D & AR (Augmented Reality)



KINECT





8.2. Giới thiệu OpenCV

- OpenCV: Open Computer Vision Library
 - Tập hợp các hàm C và một số lớp C++ giải quyết các bài toán, thuật toán cơ bản trong xử lý ảnh
 - Đa nền tảng, đã porting được trên rất nhiều nền tảng khác nhau: Windows, Linux, Embedded Linux, iOS, Android...



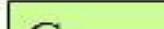
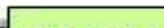
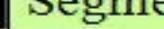
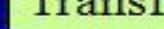
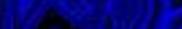
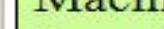
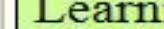
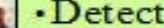
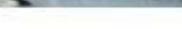
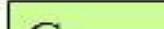
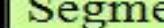
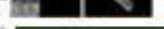
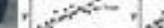
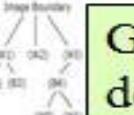
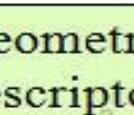
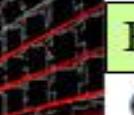
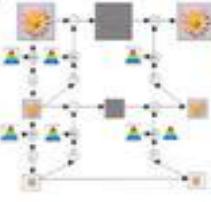
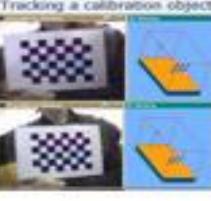
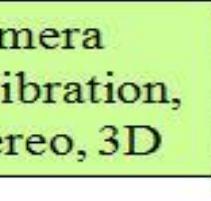
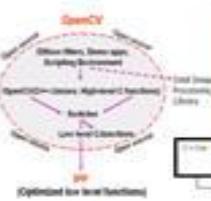
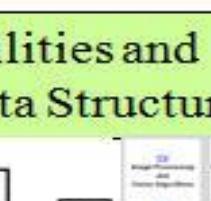
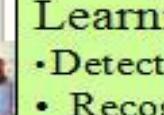
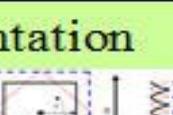
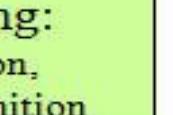
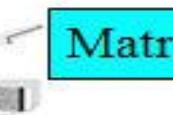
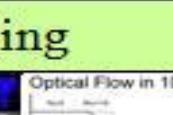
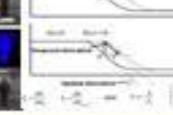
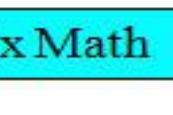
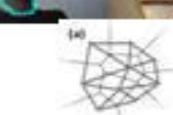
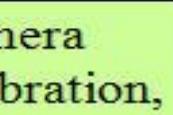
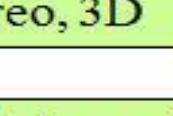
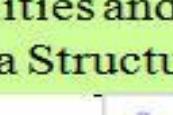
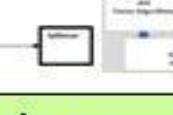
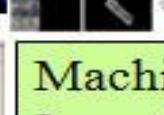
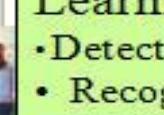
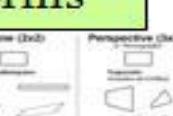
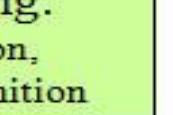
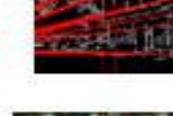
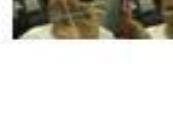
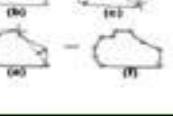
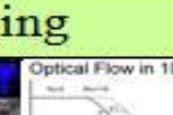
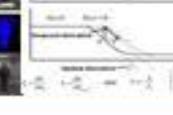
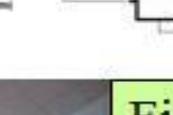
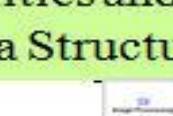
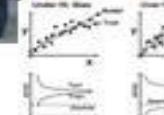
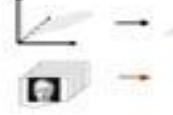
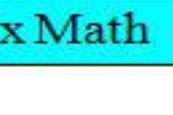
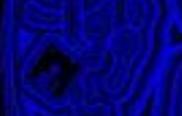
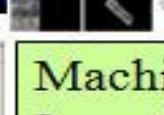
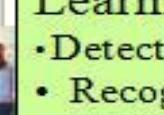
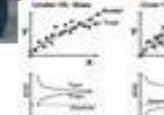
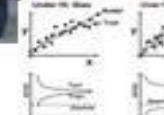
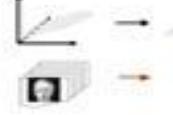
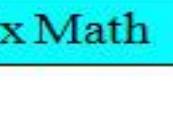
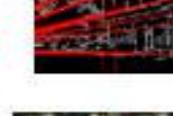
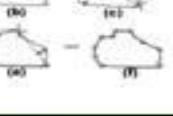
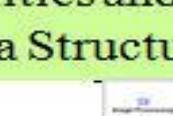
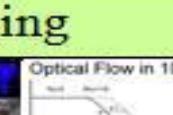
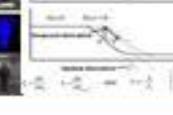
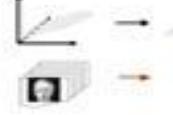
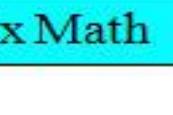
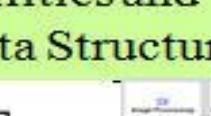
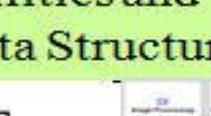


Thư viện OpenCV

OpenCV Overview: > 500 functions

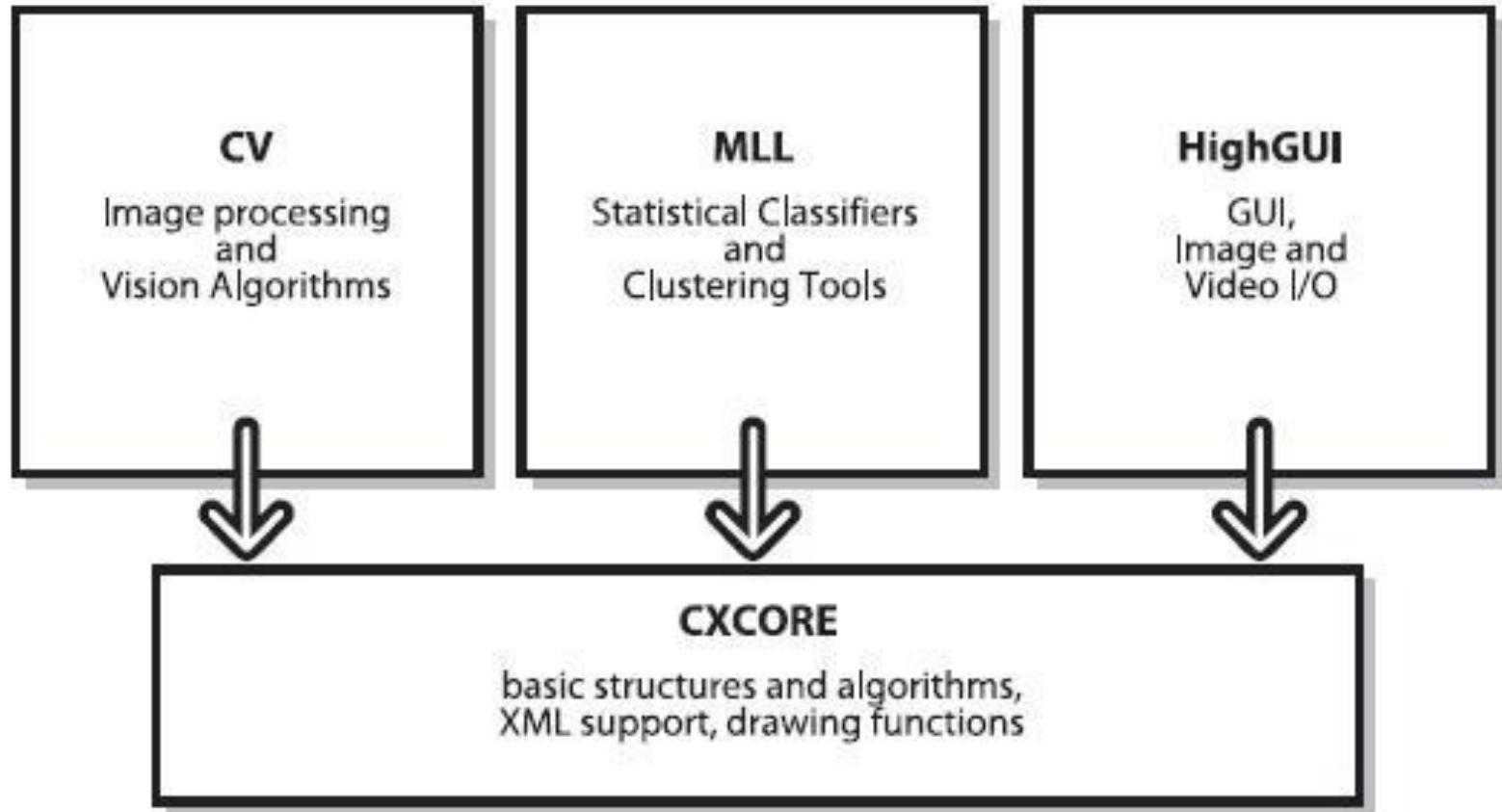
opencv.willowgarage.com



                                         	<h3>General Image Processing Functions</h3>                        	<h3>Geometric descriptors</h3>             	<h3>Image Pyramids</h3>          	<h3>Segmentation</h3>                                                	<h3>Transforms</h3>                                    	<h3>Machine Learning: Detection, Recognition</h3> <ul style="list-style-type: none">DetectionRecognition        	<h3>Matrix Math</h3>    	<h3>Tracking</h3>                            	<h3>Utilities and Data Structures</h3>         	<h3>Fitting</h3>         
---	---	---	--	--	---	--	--	---	--	--



Kiến trúc thư viện OpenCV





Kiến trúc thư viện OpenCV

- **CV:** chứa các hàm cơ bản và nâng cao thực thi các bài toán thị giác máy (computer vision)
- **ML (Machine Learning):** thư viện học máy với các công cụ phân lớp (classifier) và phân cụm (clustering).
- **HighGUI:** các hàm vào ra và các hàm lưu trữ, nạp và hiển thị ảnh và video
- **CXCore:** chứa các kiểu dữ liệu cơ bản, một số thuật toán cơ bản và các hàm vẽ, có hỗ trợ XML



Cài đặt thư viện OpenCV

- **Bước 1:** Cài đặt thư viện OpenCV trên máy host (Linux Desktop)
- **Bước 2:** Biên dịch chéo, cài đặt thư viện OpenCV để biên dịch cho các ứng dụng trên KIT
- **Chi tiết:** Xem tài liệu hướng dẫn cài đặt



Tích hợp OpenCV và QT

- Khai báo trong file .pro của dự án QT: thêm các dòng lệnh sau vào cuối file .pro

```
CONFIG += build_x86
#Khi can bien dich cho X86 LINUX DESKTOP
build_x86{
    INCLUDEPATH += /usr/local/include/opencv
    LIBS += /usr/local/lib/*.so
}
#Khi can bien dich de chay tren KIT
build_arm{
    INCLUDEPATH += /opt/opencv.arm/include/opencv
    LIBS += /opt/opencv.arm/lib/*.so
    LIBS += /opt/opencv.arm/lib/*.so.4
}
```



Tích hợp OpenCV và QT

- Khai báo các thư viện sẽ được sử dụng: để ứng dụng linh hoạt, tạo file global.h chứa include tới các thư viện của OpenCV

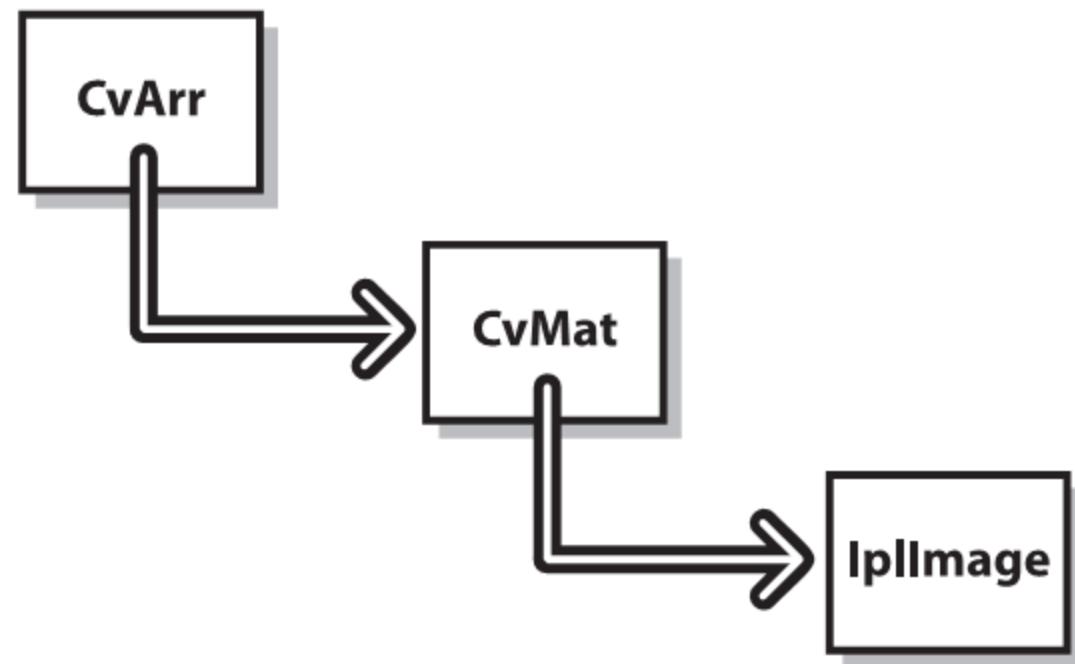
```
#define X86_DESKTOP
#ifndef GLOBAL_H
#define GLOBAL_H
#endif X86_DESKTOP
#include <opencv/cv.h>
#include <opencv/highgui.h>
#endif
#endif ARM_LINUX
#include <cv.h>
#include <highgui.h>
#endif
#endif //GLOBAL_H
```



Đọc ảnh và hiển thị

- Các kiểu dữ liệu cơ bản trong OpenCV

- cvArr
- cvMat
- IplImage
- CvCapture





Đọc ảnh và hiển thị

▪ Hàm đọc ảnh: *cvLoadImage*

```
IplImage* img = cvLoadImage( argv[1] );
```

- Tham số đầu vào: đường dẫn tới file ảnh
- Tham số đầu ra: dữ liệu ảnh lưu theo kiểu dữ liệu con trả của IplImage

▪ Ví dụ:

```
IplImage* img = cvLoadImage("/home/oto.jpeg");
```



Kết nối Camera

- Hàm mở file video:

```
CvCapture* capture = cvCreateFileCapture( argv[1] );
```

- Hàm mở webcam:

```
CvCapture* capture=cvCreateCameraCapture(0)
```



Kết nối Camera

- **Bước 1:** mở kết nối với Camera mặc định

```
CvCapture* camera = cvCreateCameraCapture(0);
```

- **Bước 2:** lấy về từng Frame ảnh của camera

```
IplImage* preImage=cvQueryFrame(camera);
```

- **Bước 3:** giải phóng đối tượng camera

```
cvReleaseCapture(&camera);
```



8.3. Các phép biến đổi cơ bản

8.3.1. Tìm hiểu cách thức biểu diễn ảnh

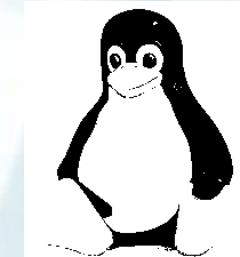
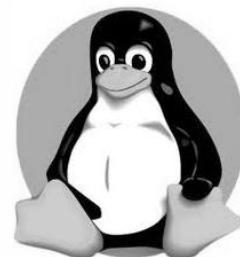
8.3.2. Biến đổi ảnh màu sang ảnh đa mức xám

8.3.3. Xây dựng phân bố Histogram của ảnh

8.3.4. Lập trình dãn độ tương phản

8.3.5. Lập trình cân bằng độ tương phản

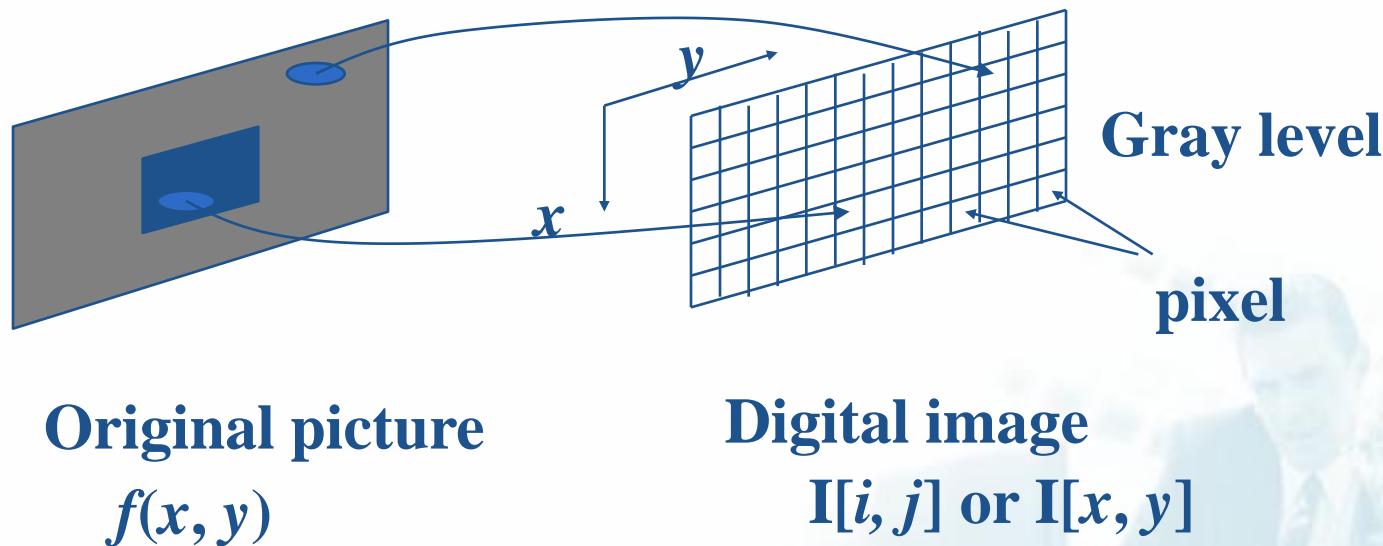
8.3.6. Biến đổi ảnh đa mức xám sang ảnh nhị phân





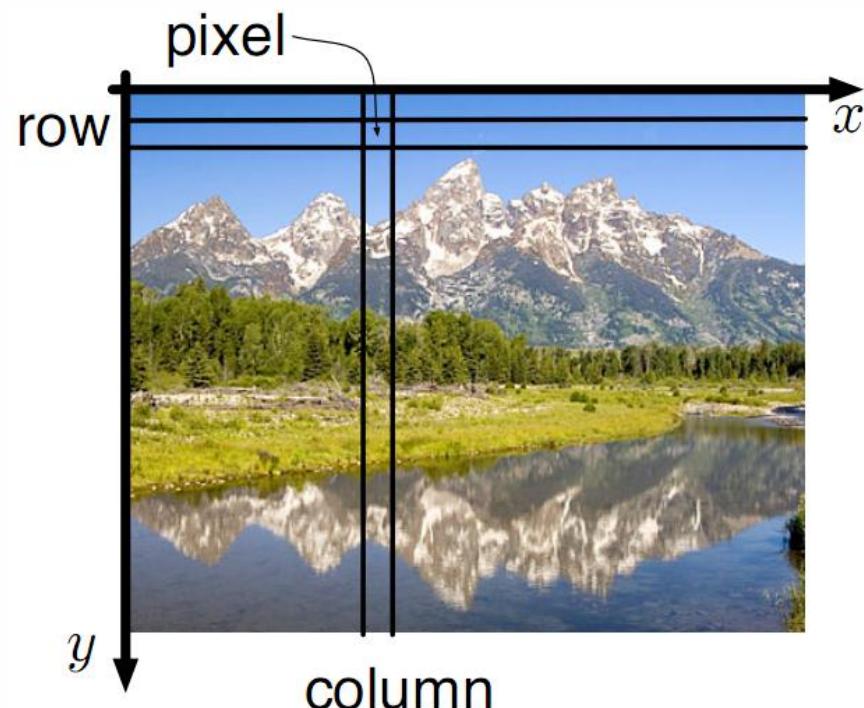
8.3.1. Cách thức biểu diễn ảnh

- **Khái niệm ảnh số:** là ảnh thu được từ ảnh liên tục bằng phép lấy mẫu và lượng tử hóa





- Một ảnh số thường được biểu diễn như một ma trận các điểm ảnh
- Trong đó mỗi điểm ảnh có thể được biểu diễn bằng
 - 1 bit (ảnh nhị phân)
 - 8 bit (ảnh đa mức xám)
 - 16, 24 bit (ảnh màu)





Cách thức biểu diễn ảnh

▪ **Ảnh màu (Color image)**

- Chứa thông tin màu của ảnh
- Không gian màu thường sử dụng: RGB, CMYK
- Ảnh RGB 3 kênh màu, mỗi kênh sử dụng 8 bit

▪ **Ảnh đa mức xám (Grayscale image)**

- Ảnh đa mức xám là ảnh có sự chuyển dần mức xám từ trắng sang đen.
- Sử dụng 8 bit để biểu diễn mức xám

▪ **Ảnh nhị phân (Binary image)**

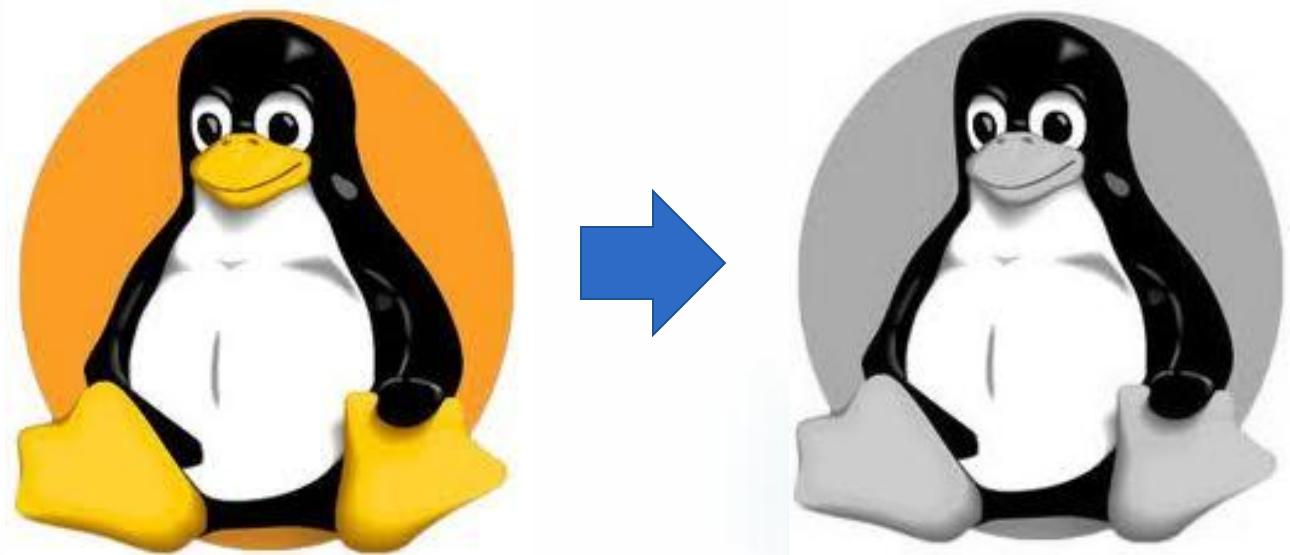
- Chỉ có hai màu đen, trắng (tương đương giá trị 1,0)



8.3.2. Chuyển ảnh màu -> đa mức xám

- Sử dụng công thức:

Gray scale= 0.2989*R + 0.5870*G + 0.1140*B;





Chuyển ảnh màu -> đa mức xám

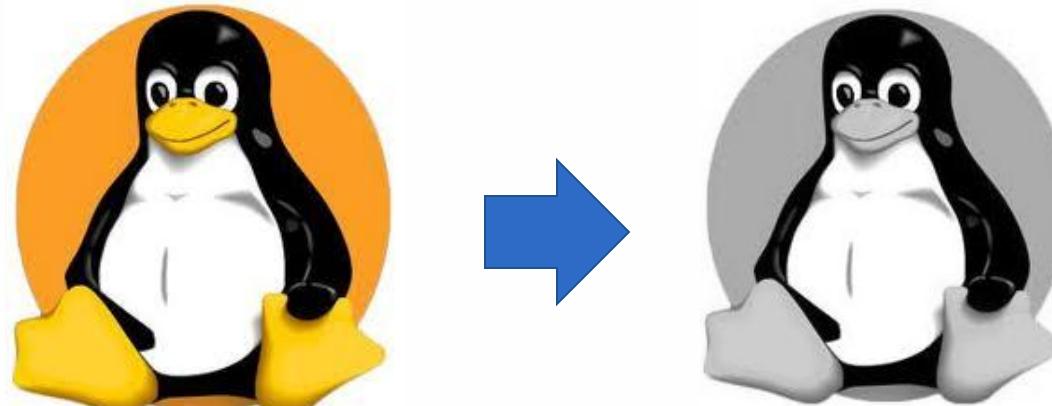
▪ Hàm chuyển ảnh màu sang ảnh đa mức xám:

- **Bước 1:** Tạo 1 ảnh trắng (chưa có dữ liệu) định dạng ảnh đa mức xám

```
IplImage *grayimage = cvCreateImage( cvSize( colorimg->width, colorimg->height ), IPL_DEPTH_8U, 1 );
```

- **Bước 2:** Chuyển đổi ảnh màu sang đa mức xám

```
cvCvtColor( img, grayimage, CV_RGB2GRAY );
```





Demo chuyển sang ảnh đa mức xám

Lab2

Convert to Gray scale image

Display Histogram

Histogram Equalization

Convert to Binary Image

AdaptiveThreshold

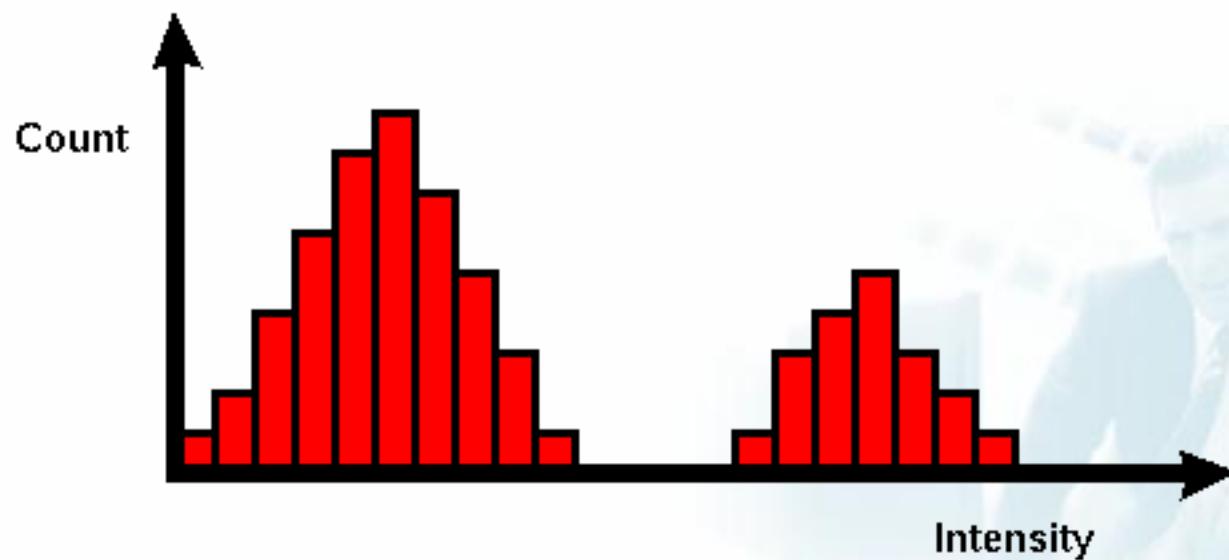
Lab 2-Convert Color to Gray Image

Color Image

Gray Image

8.3.3. Histogram

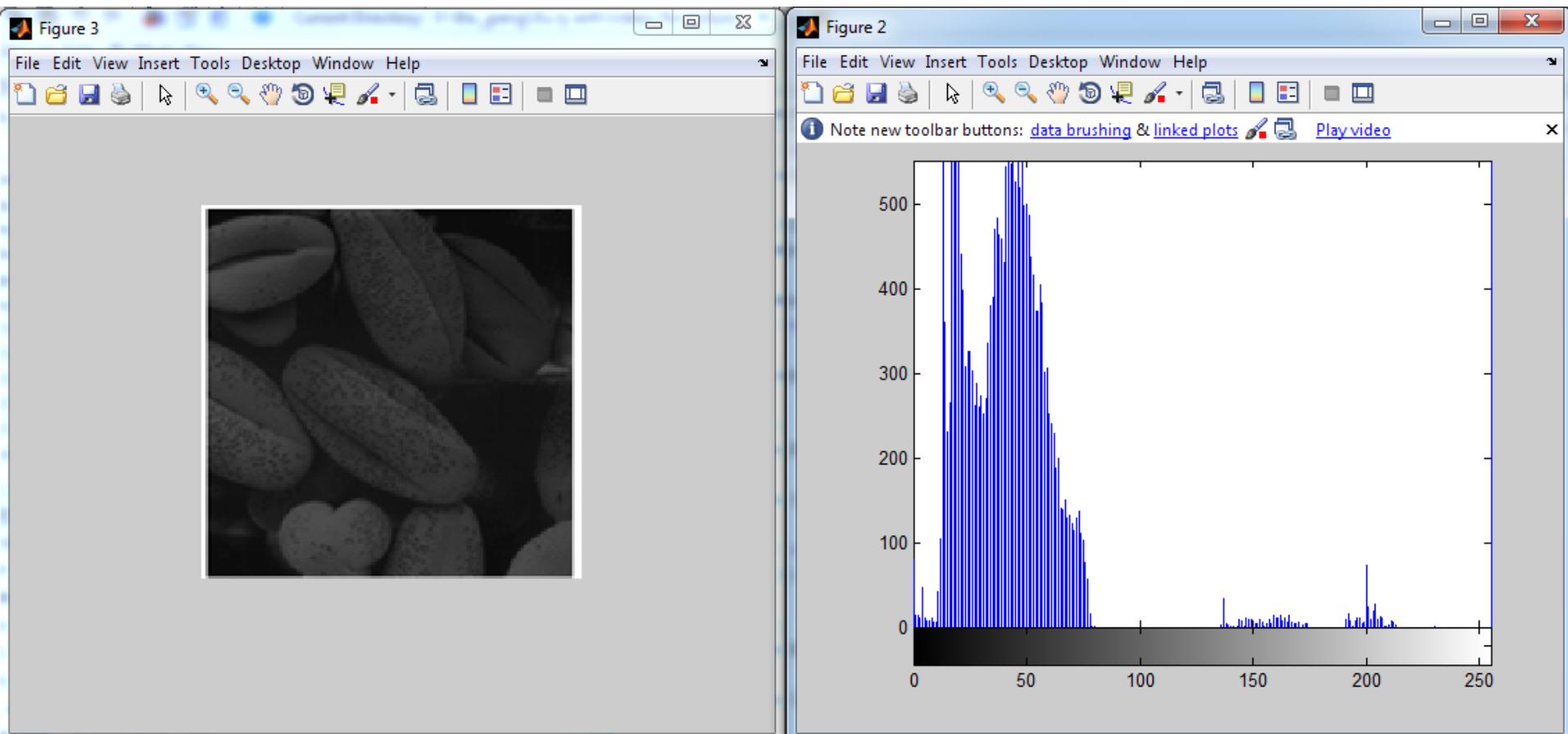
- Biểu đồ này đơn giản cho chúng ta biết số điểm ảnh (pixel) trong một ảnh đa mức xám có một giá trị mức xám tương ứng.
- Thông số: số bins (=256 với ảnh đa mức xám)





Histogram

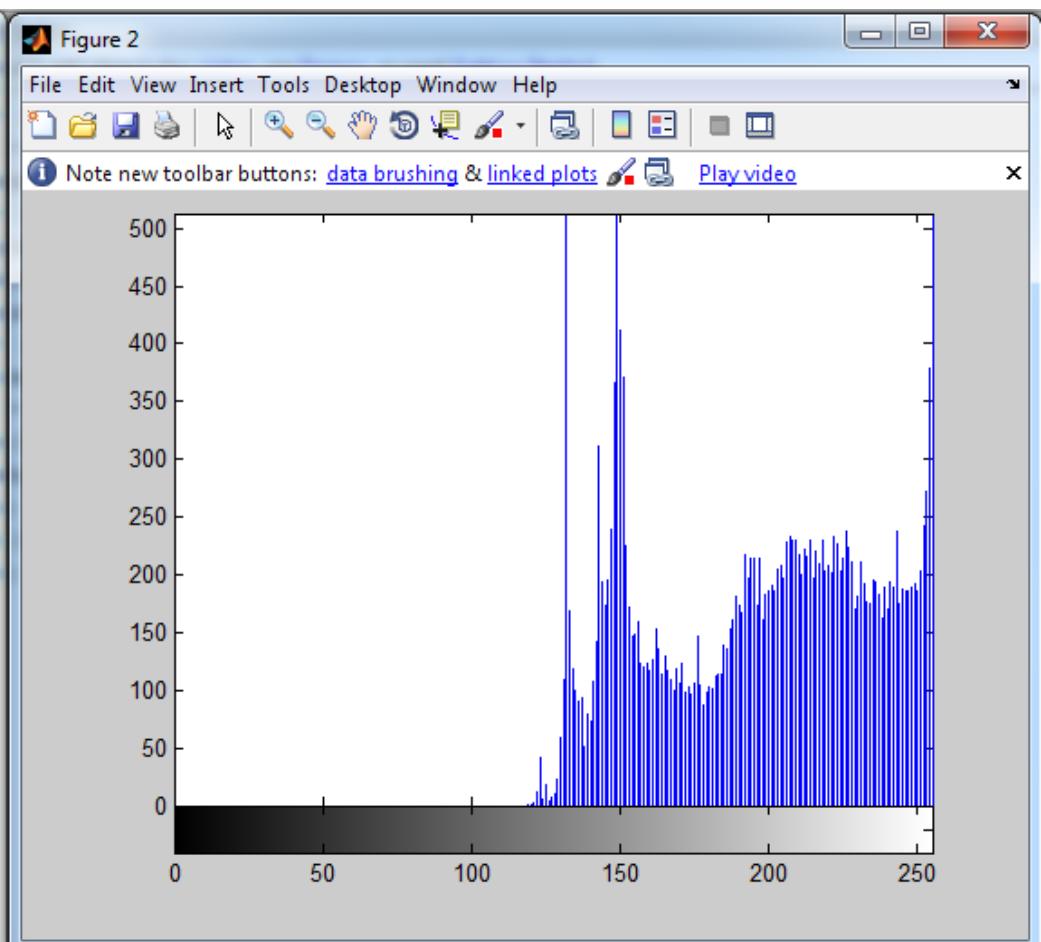
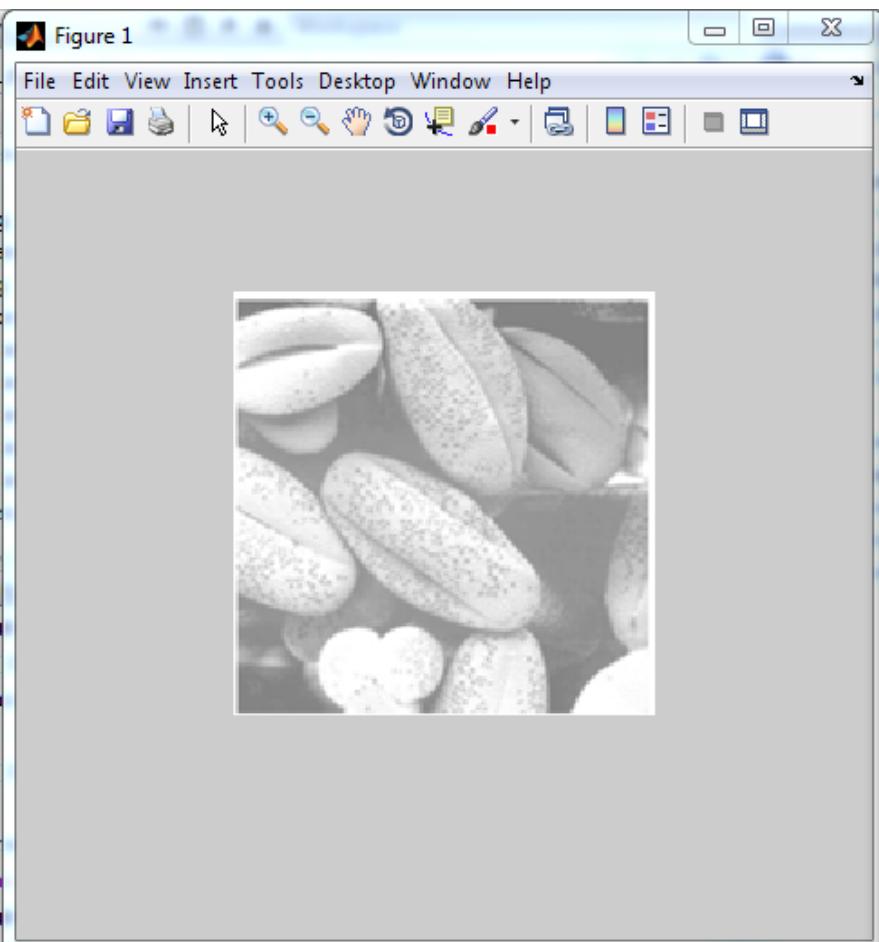
■ Ảnh tối





Histogram

- Ảnh sáng





Hàm tính Histogram

- Cấu trúc CvHistogram: lưu các thông tin về phân bố Histogram của ảnh

```
typedef struct CvHistogram
{
    int      type;
    CvArr*  bins;
    float   thresh[CV_MAX_DIM][2]; /* for uniform histograms */
    float** thresh2; /* for non-uniform histograms */
    CvMatND mat; /* embedded matrix header for array histograms */
}
CvHistogram;
```



Hàm tính Histogram

- **Bước 1:** Sử dụng hàm cvCreateHist khởi tạo cấu trúc CvHistogram để chuẩn bị chứa kết quả
 - Dims: số chiều của Histogram
 - Sizes: số lượng bins
 - Type: định dạng dữ liệu (thường sử dụng CV_HIST_ARRAY)
 - Ranges: Dải các khoảng giá trị để tính Histogram

```
CvHistogram* cvCreateHist(int dims, int* sizes, int type, float** ranges=NULL, int uniform=1)
```



Hàm tính Histogram

▪ **Bước 2:** Tính Histogram sử dụng hàm cvCalcHist

- Image: Ảnh cần tính Histogram
- Hist: lưu kết quả tính Histogram
- Accumulate: tùy chọn tích lũy, cho phép tính Histogram từ nhiều ảnh
- Mask: xác định phạm vi các pixel sẽ được sử dụng để tính Histogram, mặc định tính toàn ảnh

```
void cvCalcHist(IplImage** image, CvHistogram* hist, int accumulate=0, const CvArr*  
mask=NULL)
```



Hàm tính Histogram

- **Bước 3:** Vẽ phân bố Histogram
- **Bước 4:** Giải phóng bộ nhớ sử dụng hàm cvClearHist



Demo tính Histogram của ảnh

Lab2

Convert to Gray scale image

Display Histogram

Histogram Equalization

Convert to Binary Image

AdaptiveThreshold

Lab 2-Calculate Histogram

Gray Image

Histogram Image



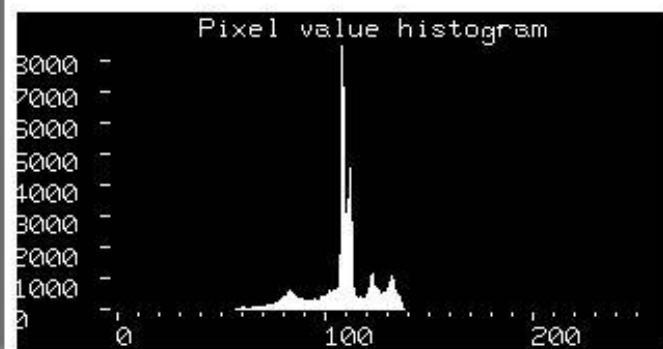
8.3.4. Dãn độ tương phản

- Đây là một kỹ thuật tăng cường chất lượng ảnh thông dụng, nó có tác dụng làm tăng độ tương phản của ảnh bằng cách giãn dải giá trị mức xám của ảnh. Các bước thực hiện
 - Tìm ra giới hạn mức xám nhỏ nhất (a) và lớn nhất (b) của ảnh. Với ảnh đa mức xám thường $a=0$ và $b=255$.
 - Tìm giá trị mức xám nhỏ nhất (c) và lớn nhất (d) trong các điểm ảnh
 - Xác định giá trị mức xám mới theo công thức

$$P_{out} = (P_{in} - c) \left(\frac{b - a}{d - c} \right) + a$$



Dãn độ tương phản



Ảnh trước và sau khi tiến hành dãn độ tương phản



8.3.5. Cân bằng Histogram

- Cân bằng histogram là một phương pháp thay đổi độ tương phản của ảnh bằng cách thay đổi lược đồ phân bố mức xám của chúng.
- Mục đích làm thay đổi biểu đồ phân bố mức xám từ phân bố ban đầu sang sự phân bố hướng tới đều.
- Tác dụng nhằm phát hiện những đối tượng bị che khuất trong ảnh ban đầu. Phép biến đổi này rất có ý nghĩa đối với những bức ảnh chụp trong bóng đêm, đối tượng thường bị mờ, hay bị che khuất bởi bóng tối, áp dụng cân bằng histogram có thể làm nổi rõ đối tượng hơn.

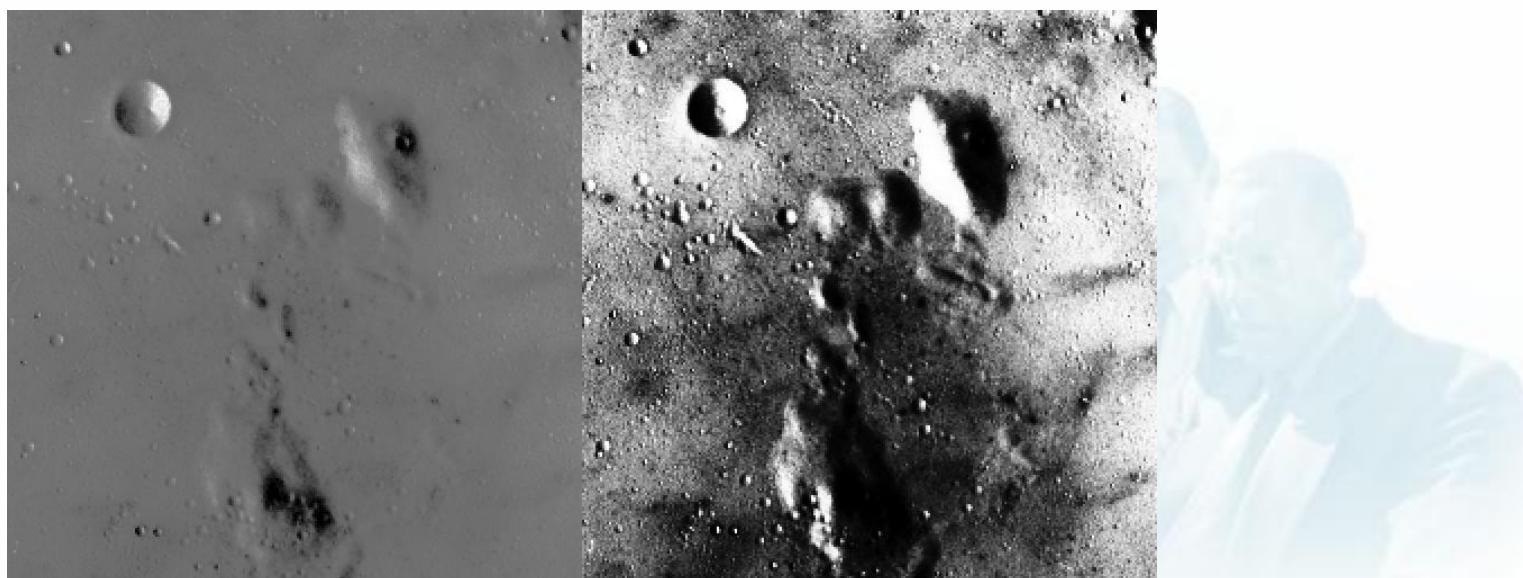


Cân bằng Histogram

- Công thức thực hiện:

$$b_k = (b_{\max} - b_{\min})$$

- Trong đó b_{\min} , b_{\max} là những giá trị được chọn, p_i là xác suất xuất hiện giá trị mức xám i trong ảnh ban đầu, với $i \in [a_{\min}, a_{\max}]$





Hàm cân bằng Histogram

- Sử dụng hàm cvEqualizeHist
 - Src: ma trận ảnh gốc
 - Dst: ma trận ảnh sau khi đã cân bằng Histogram

```
void cvEqualizeHist(  
    const CvArr* src,  
    CvArr*         dst  
)
```



Demo cân bằng Histogram

Lab2

Convert to Gray scale image

Display Histogram

Histogram Equalization

Convert to Binary Image

AdaptiveThreshold

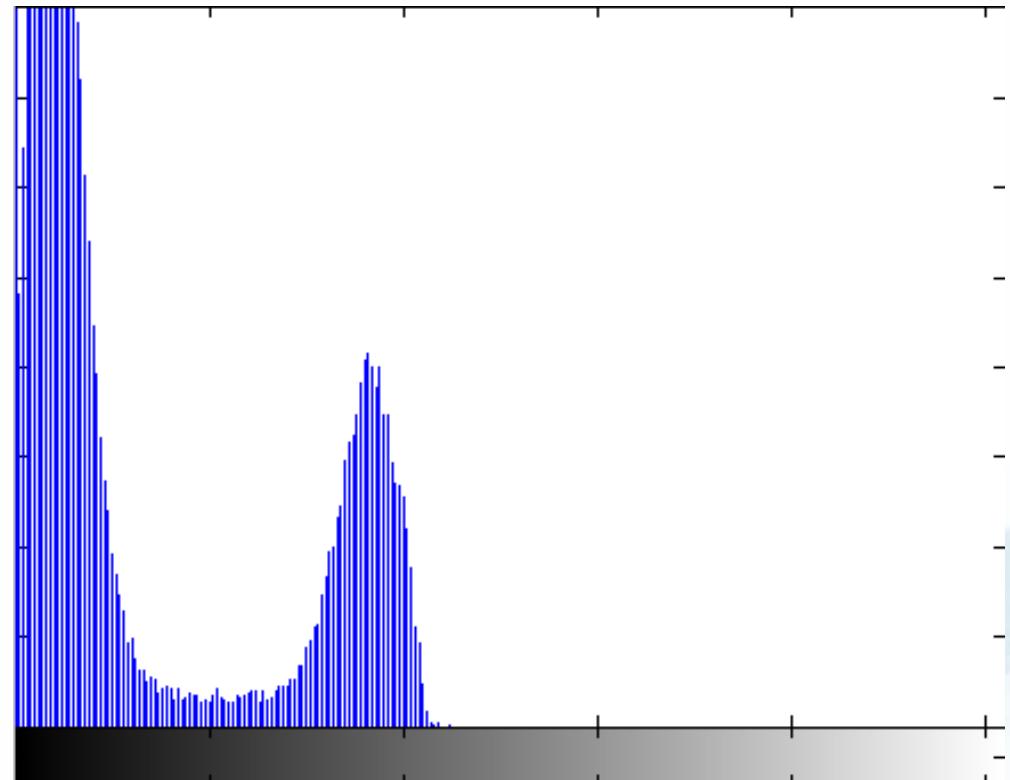
Original Image

Image after Histogram Equalization



8.3.6. Chuyển sang ảnh nhị phân

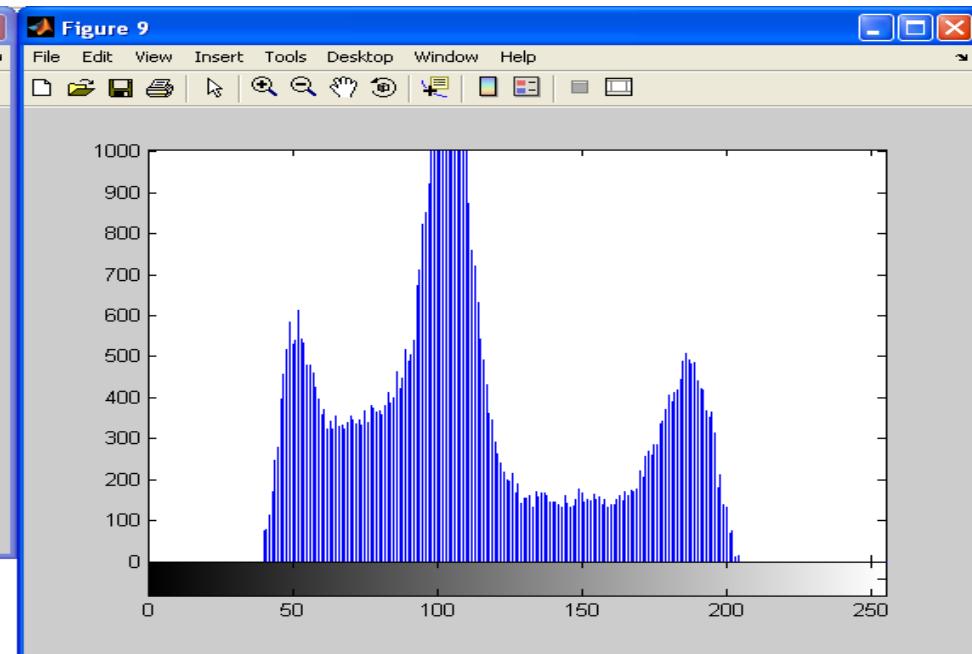
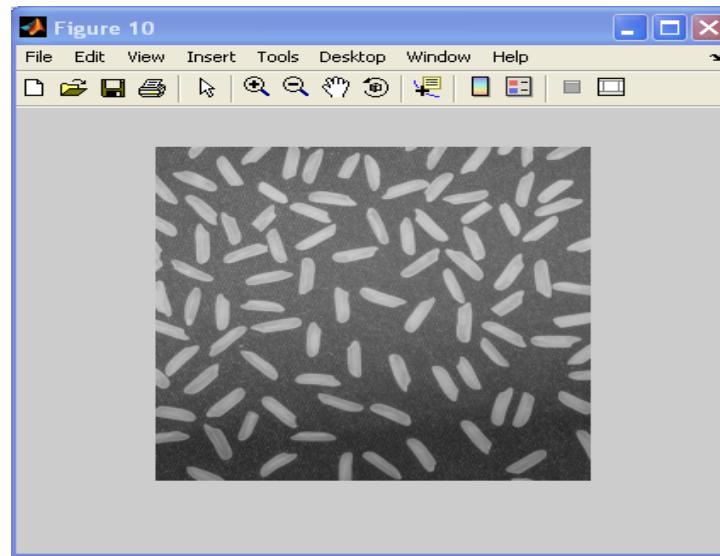
- Chuyển đổi dựa trên phân ngưỡng
 - Phân ngưỡng cố định (fixed threshold): sử dụng khi Histogram phân bố rõ ràng hai vùng sáng, tối với hai đỉnh rõ rệt





Chuyển sang ảnh nhị phân

- **Phân ngưỡng thích nghi (adaptive threshold)**
 - Trong trường hợp lược đồ mức xám của ảnh có tới ≥ 3 đỉnh chóp, ví dụ trong ảnh dưới đây tương ứng với số điểm ảnh có giá trị mức xám tương ứng là 50, 110 và 180.





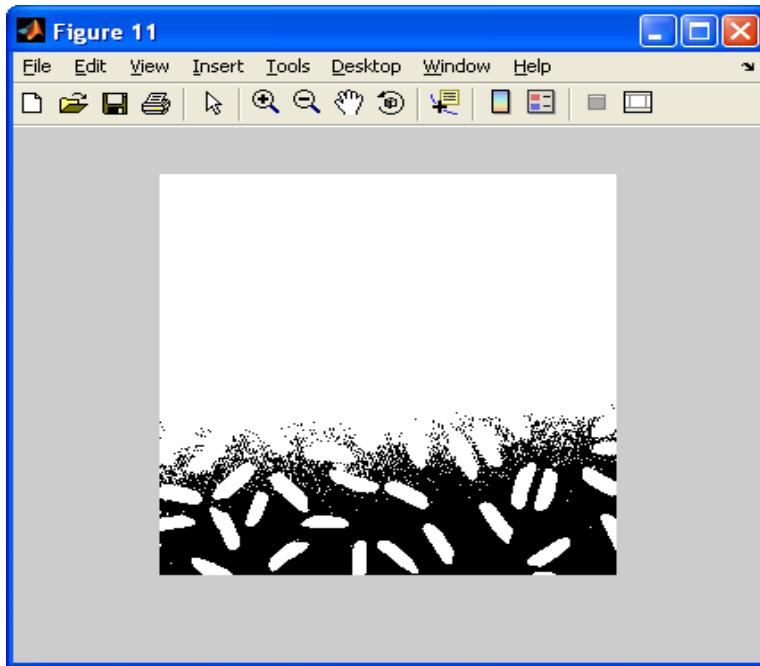
Chuyển sang ảnh nhị phân

▪ Phân ngưỡng thích nghi:

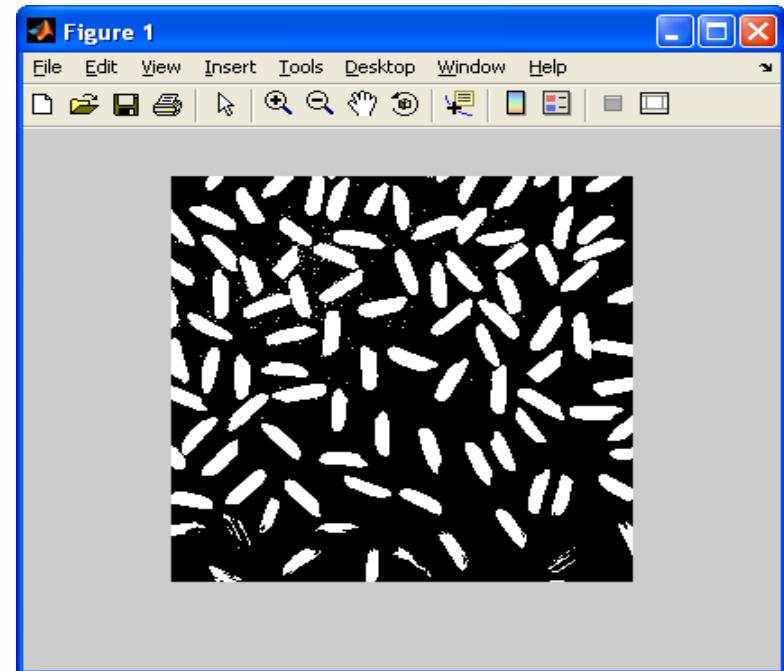
- Sử dụng ngưỡng động cho các điểm ảnh khác nhau
- Kỹ thuật này cho phép chúng ta có thể điều tiết, thích nghi với sự thay đổi về điều kiện sáng của ảnh ví dụ như ảnh có sử dụng các hiệu ứng rọi sáng (illumination) hay đổ bóng(shadow).



Chuyển sang ảnh nhị phân



Kết quả
(Sử dụng ngưỡng cứng)



Kết quả
(Sử dụng ngưỡng thích nghi)



Hàm chuyển sang ảnh nhị phân

▪ Phân ngưỡng cứng

- Src: Ảnh ban đầu, dst: ảnh kết quả
- Threshold: ngưỡng được chọn
- maxValue: giá trị lớn nhất
- thresholdType: kiểu phân ngưỡng

```
void cvThreshold( const CvArr* src, CvArr* dst, double threshold,  
                  double maxValue, int thresholdType );
```



Hàm chuyển sang ảnh nhị phân

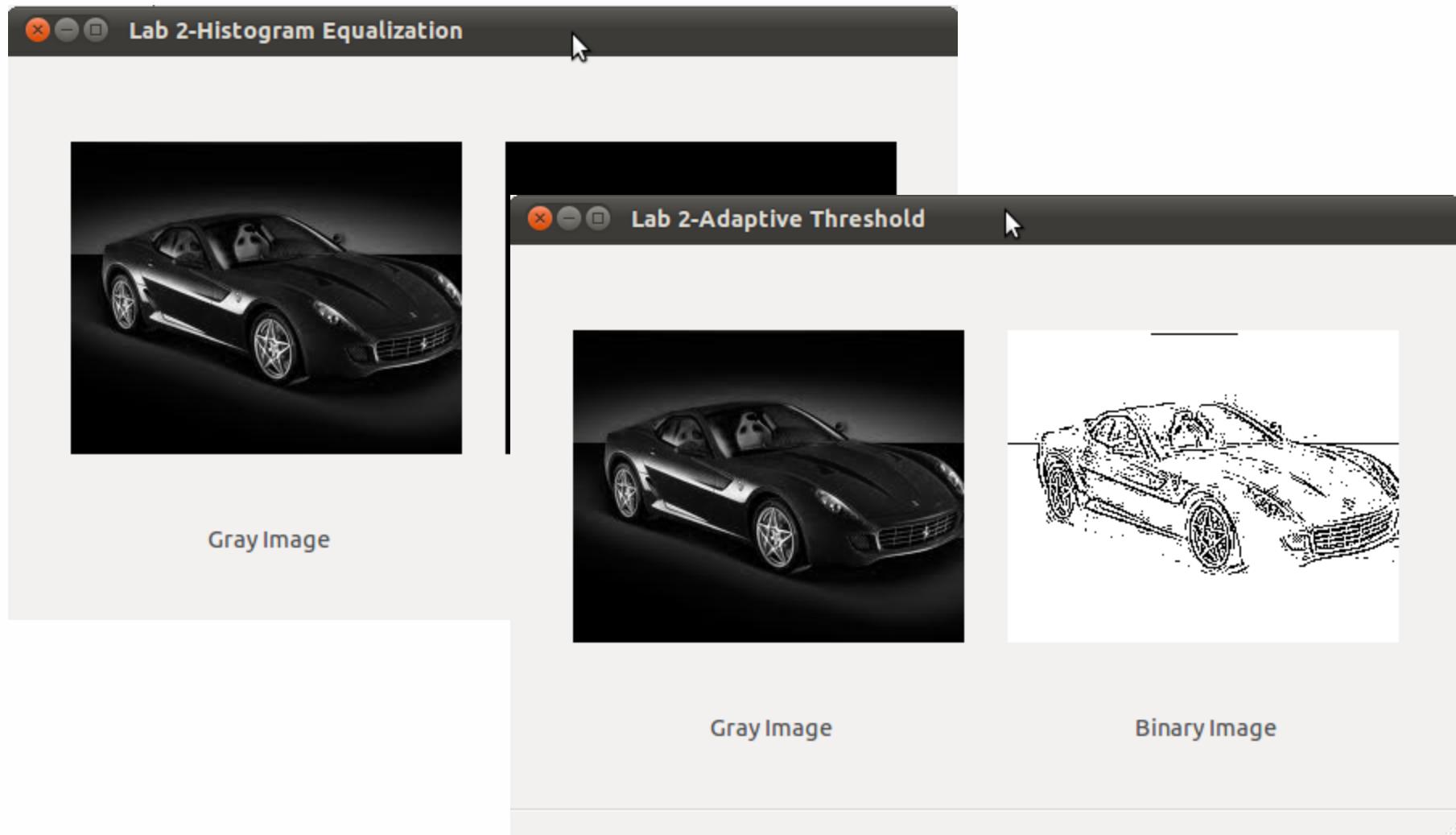
▪ Phân ngưỡng thích nghi

- Src: ảnh ban đầu, dst: ảnh kết quả
- maxValue: giá trị lớn nhất
- adaptiveMethod: phương pháp tính ngưỡng thích nghi (CV_ADAPTIVE_THRESH_MEAN_C hay CV_ADAPTIVE_THRESH_GAUSSIAN_C)

```
void cvAdaptiveThreshold( const CvArr* src, CvArr* dst, double maxValue,  
                         int adaptiveMethod, int thresholdType,  
                         int blockSize, double param1 );
```



Demo chuyển sang ảnh nhị phân





Phụ lục A – Các lệnh Linux

- Lệnh hiển thị thông tin các file trong thư mục
ls -al //hiển thị danh sách với đầy đủ thông tin
- Lệnh thay đổi quyền cho một file hay thư mục
chmod

vd: **chmod +x Filename** //Cấp thêm quyền thực thi

- Lệnh để xem danh sách các file thiết bị
ls -al /dev
- Lệnh để xem tất cả các tiến trình đang chạy
ps



Phụ lục A – Các lệnh Linux

- Lệnh cài đặt một phần mềm từ kho chứa của Linux

sudo apt-get install Tên_gói_phần_mềm

- Xem danh sách các major id tương ứng với các device driver đang active

cat /proc/devices

- Tìm kiếm file chứa một dòng text bất kỳ

grep vd: **grep -r “Hello” .**

//Tìm tất cả các file và hiển thị ra các dòng chứa từ khóa

//Hello trong thư mục hiện tại và các thư mục con



Phụ lục B – Website quan trọng

- <http://www.friendlyarm.net //download>
- [http://www.thaieeasyelec.com/FriendlyARM //mua KIT, download tài liệu](http://www.thaieasyelec.com/FriendlyARM //mua KIT, download tài liệu)
- <http://dientuvietnam.net //forum chia sẻ>
- <http://eetimes.com //Tin tức công nghệ>
- <http://www.kernel.org //download mã kernel>
- <http://qt.nokia.com //Hỗ trợ QT SDK>
- qtforum.org
- qtcenter.org