TÀI LIỆU THÍ NGHIỆM IT4425

PHÁT TRIỂN PHẦN MỀM NHÚNG THÔNG MINH

2021

Bài thí nghiệm 1

GIAO TIẾP GPIO TRONG PHẦN MỀM NHÚNG NỀN TẢNG ARM LINUX

1. Mục đích:

- Nắm được kỹ năng cơ bản về lập trình, phát triển phần mềm trên hệ nhúng nền tảng ARM Linux với các giao tiếp vào ra cơ bản GPIO, thực hiện minh họa trên KIT FriendlyARM hoặc Raspberry Pi 4.

2. Chuẩn bị:

- Bộ KIT FriendlyARM mini/micro2440 hoặc Raspberrry Pi 4
- Máy tính cài đặt Ubuntu (có thể dùng máy ảo), cài đặt trình biên dịch chéo arm-linux-gcc

3. Nội dung thực hành

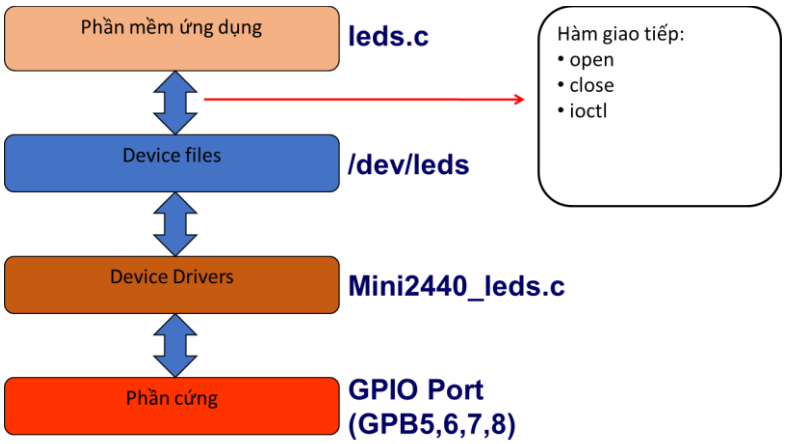
3.1. Giao tiếp GPIO qua driver có sẵn

Bài 1. Giao tiếp với LED driver trên KIT FriendlyARM

Bước 1. Mô tả:

- Dãy 4 led đơn trên KIT ghép nối qua cổng GPIO (GP5, 6, 7, 8) đã có sẵn driver trên Embedded Linux.

- Driver cung cấp hàm điều khiển (ioctl) tắt/bật từng led đơn. Để tạo các hiệu ứng led (nháy, chạy đuổi, đập tường, …) cần xử lý thuật toán lập trình + sử dụng các hàm trễ (delay) tạo hiệu ứng (sử dụng thư viện sys/time.h trên linux, các hàm trễ usleep(us) hoặc sleep(s).
- Mô hình giao tiếp:



Bước 2. Soạn thảo mã nguồn chương trình (tham khảo)

Ví dụ sau minh họa mở file thiết bị leds và điều khiển tắt/bật 1 led có số hiệu led_no (0,3)

(File tham khảo: leds.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
int main(int argc, char **argv)
{
	int on, led_no;
	int fd;
	//kiểm tra các tham số truyền từ dòng lệnh có phù hợp
	if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 ||
sscanf(argv[2], "%d", &on) != 1 || on < 0 || on > 1 || led_no < 0 || led_no
> 3) {
		fprintf(stderr, "Usage: leds led_no 0|1\n"); //cách sử dụng
		exit(1);
	}
	fd = open("/dev/leds", 0); //Mở file thiết bị leds để sử dụng
	if (fd < 0) {
		perror("open device leds");
		exit(1);
	}
	ioctl(fd, on, led_no); //Hàm điều khiển từng on/off led đơn
	close(fd); //Đóng file thiết bị sau khi sử dụng
	return 0;
}
```

Bước 3. Biên dịch chương trình, nạp lên KIT, thực thi và quan sát kết quả.

Chú ý: Mặc định trên KIT khởi động xong sẽ chạy ứng dụng led_player điều khiển cụm LED, cần tắt ứng dụng này để tránh xung đột.

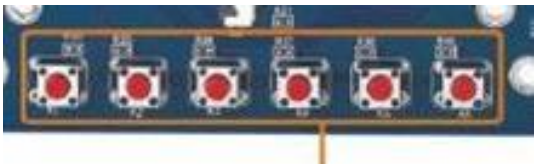
(Sử dụng lệnh ps để tìm tiến trình led_player này, lệnh kill PID để tắt tiến trình này)

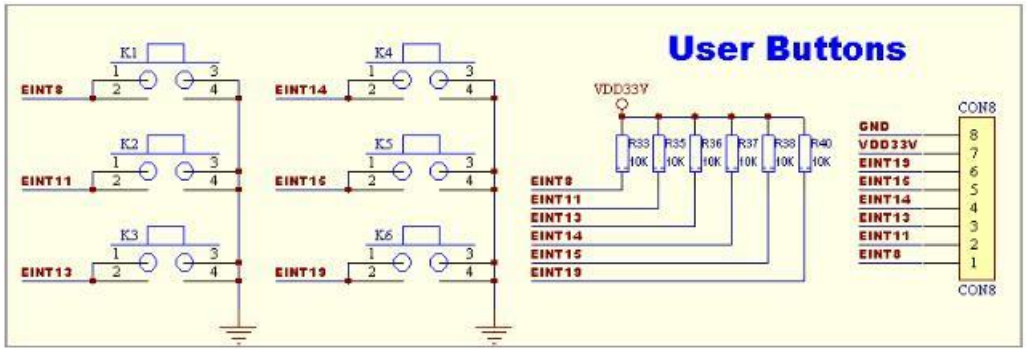
Bước 4. Yêu cầu nâng cao: Dựa trên ví dụ trên, viết chương trình điều khiển tắt, bật dãy led đơn với hiệu ứng khác nhau (nhấp nháy xen kẽ, chạy đuổi, …)

Bài 2. Giao tiếp với Button driver trên KIT FriendlyARM

Bước 1. Mô tả:

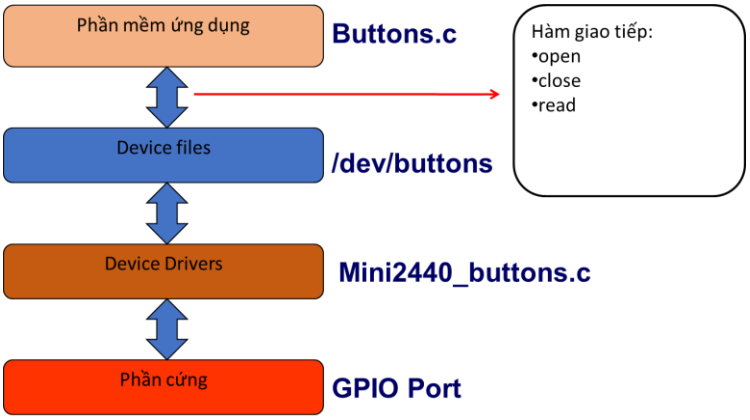
- Dãy nút bấm K1, K2, K3, K4, K5, K6 trên KIT được ghép nối qua GPIO, đã có sẵn driver trên hệ điều hành Linux nhúng.





- Có thể đọc trạng thái các nút bấm này (pressed/release or not ?) và có xử lý thích hợp.

- Mô hình lập trình với nút bấm:



Bước 2. Viết mã nguồn chương trình đọc nút bấm (tham khảo)

- Driver button cung cấp hàm read() cho phép đọc trạng thái của 6 nút bấm (giá trị trả về là '0' hoặc '1' tương ứng nút được nhấn (pressed) hay nhả (released).

- Một cách đơn giản có thể sử dụng cơ chế polling (thăm dò) để đọc trạng thái nút bấm. Với phương pháp này, chương trình sẽ liên tục đọc trạng thái nút bấm để xem có được bấm hay không (sự kiện được bấm tương ứng với giá trị của nút trả về sẽ bị thay đổi '1'-> '0' -> '1').

- Tham khảo mã nguồn sau:

(File buttons.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>
int main(void)
{
	int buttons_fd;
	//mảng chứa giá trị trả về của 6 nút bấm (giá trị là ký tự ‘0’, ‘1’)
	char buttons[6] = { '0', '0', '0', '0', '0', '0' };
	buttons_fd = open("/dev/buttons", 0); //mở file thiết bị button
	if (buttons_fd < 0) {
		perror("open device buttons");
		exit(1);
	}
	for (;;) {//Vòng lặp liên tục đọc trạng thái nút bấm (poll)
		char current_buttons[6]; //mảng chứa giá trị hiện tại đọc được
			int i;
		//liên tục gọi hàm đọc
		if (read(buttons_fd, current_buttons, 6) != 6) {
			perror("read buttons:");
			exit(1);
		}
		//duyệt mảng chứa giá trị 6 nút bấm
		for (i = 0; i < 6; i++) {
			//nếu có sự thay đổi nút tương ứng được bấm
			if (buttons[i] != current_buttons[i]) {
				//cập nhật lại trạng thái nút
				buttons[i] = current_buttons[i];
				//hiện thông tin ra màn hình
				printf("key %d is %s\n", i+1,
					buttons[i] == '0' ? "up" : "down");
			}
		}
	}
	close(buttons_fd);
	return 0;
}
```

Bước 3. Biên dịch, nạp chương trình, thực thi và quan sát kết quả trên KIT

Bài tập nâng cao:

- Viết chương trình kết hợp giao tiếp led, nút bấm. Khi người dùng bấm nút (K1->K4) sẽ bật tắt led tương ứng (led 0->3).

- Kết hợp lập trình đa luồng (luồng chính tạo hiệu ứng led + luồng đọc nút bấm và thay đổi hiệu ứng led phù hợp: tăng/giảm trễ, thay đổi kiểu hiệu ứng, …)

3.2. Giao tiếp GPIO sử dụng /sys/class/gpio

Sử dụng KIT FriendlyARM hoặc Raspberry Pi

Để viết một ứng dụng (trên linux) giao tiếp các chân vào ra (gpio), chúng ta thường sử dụng 2 cách sau:

- Cách 1: Viết gpio driver (trên không gian nhân hệ điều hành, kernel space), ứng dụng giao tiếp qua driver này.

- Cách 2: Sử dụng 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 user space.

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         ktmt        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]#
```

Trong trường hợp này (với linux 2.6.32.2 trên FriendlyArm 2440), gpio sysfs interface gồm các files:

- export
- unexport
- gpiochip0, gpiochip32, gpiochip64, gpiochip96, gpiochip128, gpiochip160, gpiochip192, gpiochip224

- Các gpiochip này tương ứng với các GPIO port của CPU, mỗi file gpiochip quản lý 32 pin GPIOs (32 chân/1 port)
- (GPIOA ↔ gpiochip0, GPIOB ↔ gpiochip32, …. GPIOF ↔ gpiochip160, …)

Việc giao tiếp điều khiển input/output với mỗi chân gpio sẽ trở nên dễ dàng với các thao tác đọc/ghi file tương ứng do giao diện lập trình cung cấp. Có thể sử dụng lập trình c/c++ (hàm read/write), có thể sử dụng lập trình shell linux (echo, cat, ...)

Sử dụng gpiolib bằng các lệnh của Linux (có thể thực hiện trên KIT FriendlyARM hoặc Raspberry Pi)

Bước 1. Exporting a GPIO (đăng ký để sử dụng chân gpio từ user space)

Trước khi có thể sử dụng 1 chân gpio, cần export ra không gian người dùng

Để export 1 chân gpio, ghi số hiệu (ID number) của nó vào file /sys/class/gpio/export

Ví dụ: **Để export chân GPB5: (base on gpiochip32)**

```
echo 37 > /sys/class/gpio/export
```

Để export chân GPB6:

```
echo 38 > /sys/class/gpio/export
```

Sau khi export một chân gpio, nó sẵn sàng sử dụng qua file /sys/class/gpio/gpio[ID] (ví dụ /sys/class/gpio/gpio37).

Nếu không cần sử dụng nữa, có thể giải phóng bằng cách ghi số hiệu ID của nó vào file /sys/class/gpio/export

```
echo 37 > /sys/class/gpio/unexport
```

Bước 2. Cấu hình chân input/output

Để cấu hình chân gpio là input/output bằng cách ghi giá trị in/out đến file

```
/sys/class/gipo/gpio[ID]/direction
```

Ví dụ: Thiết lập gpio37 là chân output

```
$ echo "out" > /sys/class/gpio/gpio37/direction
```

hoặc là chân input

```
$ echo "in" > /sys/class/gpio/gpio37/direction
```

Hoặc sử dụng giá trị:

"high" để cấu hình là chân output với giá trị khởi tạo là 1

"low" để cấu hình là chân output với giá trị khởi tạo là 0

Bước 3. Truy cập giá trị của chân gpio

Giá trị của một chân gpio có thể đọc/ghi bằng cách đọc/ghi file /sys/class/gpio/gpio[ID]/value.

Ví dụ: Cấu hình chân GPB5 output, và xuất giá trị 0 ra chân này

```
echo 37 > /sys/class/gpio/export

echo "out" > /sys/class/gpio/gpio37/direction

echo 0 > /sys/class/gpio/gpio37/value
```

Xây dựng các hàm giao tiếp gpio bằng C

- Tham khảo mã nguồn sau cung cấp 1 thư viện các hàm để truy cập gpio sử dụng gpio sysfs từ không gian người dùng (user space):

gpio.h

```
#ifndef GPIO_H
#define GPIO_H
int gpio_export(unsigned gpio);  //Hàm export pin ra user space
int gpio_unexport(unsigned gpio); //Hàm giải phóng pin khi không còn sử dụng
int gpio_dir_out(unsigned gpio); //Cấu hình pin là output
int gpio_dir_in(unsigned gpio); //Cấu hình pin là input
int gpio_value(unsigned gpio, unsigned value); //Đọc/ghi giá trị của pin
gpio
#endif
```

gpio.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#define GPIO_DIR_IN     0
#define GPIO_DIR_OUT    1
/****************************************************
Hàm đăng ký (export) chân gpio muốn sử dụng ra không gian người dùng
****************************************************/
int gpio_export(unsigned gpio)
{
      int fd, len;
      char buf[11];

      fd = open("/sys/class/gpio/export", O_WRONLY);
      if (fd < 0) {
```

```
            perror("gpio/export");
            return fd;
      }

      len = snprintf(buf, sizeof(buf), "%d", gpio)
            write(fd, buf, len); //Ghi số hiệu (ID) pin muốn sử dụng vào
file /sys/class/gpio/export khi đăng ký sử dụng
      close(fd);
      return 0;
}
/****************************************************
Hàm giải phóng (unexport) chân gpio khi không còn sử dụng
****************************************************/
int gpio_unexport(unsigned gpio)
{
      int fd, len;
      char buf[11];
      fd = open("/sys/class/gpio/unexport", O_WRONLY);
      if (fd < 0) {
            perror("gpio/export");
            return fd;
      }
      len = snprintf(buf, sizeof(buf), "%d", gpio);
      write(fd, buf, len); //Ghi số hiệu (ID) pin muốn sử dụng vào file
/sys/class/gpio/unexport khi giải phóng
      close(fd);
      return 0;
}
/****************************************************
Hàm cấu hình chân gpio là in hay out (dir)
****************************************************/

int gpio_dir(unsigned gpio, unsigned dir)
{
      int fd, len;
      char buf[60];
      len = snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/direction",
gpio);
      fd = open(buf, O_WRONLY);
      if (fd < 0) {
            perror("gpio/direction");
            return fd;
      }
      //Cấu hình pin là input/output bằng cách ghi giá trị (ASCII) in, out
vào file /sys/class/gpio/gpio[ID]/diriection
      if (dir == GPIO_DIR_OUT)
            write(fd, "out", 4);
      else
            write(fd, "in", 3);
      close(fd);
      return 0;
}
/****************************************************
Hàm thiết lập chân gpio out
****************************************************/
```

```
int gpio_dir_out(unsigned gpio)
{
	return gpio_dir(gpio, GPIO_DIR_OUT); //trường hợp là output
}
/************************************************
Hàm thiết lập chân gpio in
************************************************/

int gpio_dir_in(unsigned gpio)
{
	return gpio_dir(gpio, GPIO_DIR_IN); //trường hợp là input
}
/************************************************
Hàm ghi ra trị ra chân gpio out (tương ứng xuất 0 , 1)
************************************************/

int gpio_value(unsigned gpio, unsigned value)
{
	int fd, len;
	char buf[60];
	len = snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/value",
gpio);
	fd = open(buf, O_WRONLY);
	if (fd < 0) {
		perror("gpio/value");
		return fd;
	}
	//Xuất giá trị 1, 0 bằng cách ghi ra file value tương ứng với pin đã
cấu hình
	if (value)
		write(fd, "1", 2);
	else
		write(fd, "0", 2);
	close(fd);
	return 0;
}
/************************************************
Chương trình chính để test các hàm này
************************************************/
#if 1
int main(int argc, char *argv[])
{
	int i = 20;
	int pin_no = 160 //Sử dụng chân 160 (tương ứng với GPF0 trên
FriendlyArm mini/micro 2440)
		gpio_export(pin_no);
	gpio_dir_out(pin_no);
	while (i--) {
		gpio_value(pin_no, i & 1); //toggle on GPF0
		sleep(1);
	}
	gpio_unexport(pin_no);
}
#endif
```

Bài tập nâng cao: Tìm hiểu mã nguồn giao tiếp gpio sử dụng giao diện /sys/class/gpio bằng ngôn ngữ Java trên Android

```
public int openDoor(String openType) {
	String gpioNo = "";
	if (openType.equals("IN")) {
		gpioNo = "gpio947";
	}
	else if (openType.equals("OUT")) {
		gpioNo = "gpio1000";
	}
	try {
		FileOutputStream gpioFile = new FileOutputStream(new
File("/sys/class/gpio/" + gpioNo + "/value"));
		gpioFile.write(new byte[]{ 49 }); //out 1
		Thread.sleep(500);
		gpioFile.write(new byte[]{ 48 }); //out 0
		Thread.sleep(3000);
		gpioFile.write(new byte[]{ 49 }); //out 1
		gpioFile.flush();
		gpioFile.close();
	}
	catch (Exception e) {
		e.printStackTrace();
	}
	return 0;
}
```

Bài thí nghiệm 2

GIAO TIẾP CỔNG NỐI TIẾP TRONG PHẦN MỀM NHÚNG NỀN TẢNG ARM LINUX

1. Mục đích:

- Hiểu cơ chế giao nối tiếp serial trên hệ thống nhúng có hệ điều hành (ví dụ nền tảng ARM Linux), biết cách thực hiện kỹ thuật trình giao tiếp nối tiếp trên thiết bị nhúng Raspberry Pi hoặc Friendly Arm

2. Chuẩn bị:

- Bộ KIT FriendlyARM mini/micro2440 hoặc Raspberrry Pi 3/4
- Máy tính cài đặt Ubuntu, cài đặt trình biên dịch chéo arm-linux-gcc

3. Nội dung thực hành

Viết ứng dụng giao tiếp qua cổng COM giữa máy tính và KIT, sử dụng driver và thư viện đã có sẵn trên Linux. File thư viện **termios.h**

Cần kiểm tra tên file thiết bị trên Linux (host hoặc KIT), sử dụng lệnh **ls /dev**

3.1. Chương trình giao tiếp UART trên KIT FriendlyARM

KIT FriendlyARM sử dụng: COM1, hoặc COM2.

HOST Linux (Laptop): sử dụng COM ảo (USB2COM): ví dụ /dev/ttyUSB0)

Mã nguồn tham khảo:

Chương trình write.c ghi dữ liệu ra cổng com

```
#include<stdio.h>// standard input / output functions
#include<stdlib.h>
#include<string.h>// string function definitions
#include<unistd.h>// UNIX standard function definitions
#include<fcntl.h>// File control definitions
#include<errno.h>// Error number definitions
#include<termios.h>// POSIX terminal control definitionss
#include<time.h>// time calls
int main(int argc, char** argv)
{
      char result;
      int n;
      struct termios port_settings;      // structure to store the port
settings in
      int fd = open (argv[1],  O_RDWR | O_NOCTTY | O_NDELAY);
      fcntl(fd, F_SETFL, FNDELAY);                 /* Configure port
reading */
```

```
	printf("\nfd=%d",fd);
	//Setup configuration (9600, 8N1)
	/* Set the baud rates (ingoing/outgoing) to 9600 */
	cfsetispeed(&port_settings, B9600);
	cfsetospeed(&port_settings, B9600);

	port_settings.c_cflag &= ~PARENB;    // set no parity
	port_settings.c_cflag &= ~CSTOPB;     // set stop bit
	port_settings.c_cflag &= ~CSIZE;
	port_settings.c_cflag |= CS8;         // set 8 bit data
	port_settings.c_cflag &= ~CRTSCTS; //No hadware hanshaking
	// apply the settings to the port
	tcsetattr(fd, TCSANOW, &port_settings);
	while(1)
	{
		char key=getchar();
		result[0]=key;
		write(fd,result,1);
	}
	close (fd);
	return 0;
}
```

Chương trình read.c đọc dữ liệu từ cổng com

```
#include<stdio.h>// standard input / output functions
#include<stdlib.h>
#include<string.h>// string function definitions
#include<unistd.h>// UNIX standard function definitions
#include<fcntl.h>// File control definitions
#include<errno.h>// Error number definitions
#include<termios.h>// POSIX terminal control definitionss
#include<time.h>// time calls
int main(int argc, char** argv)
{
	char result;
	int n;
	struct termios port_settings;      // structure to store the port
settings in
	int fd = open (argv[1],  O_RDWR | O_NOCTTY | O_NDELAY);
	fcntl(fd, F_SETFL, FNDELAY);    /* Configure port reading */
	printf("\nfd=%d",fd);
	//Setup configuration
	/* Set the baud rates (ingoing, outgoing) to 9600 */
	cfsetispeed(&port_settings, B9600);
	cfsetospeed(&port_settings, B9600);
```

```
	port_settings.c_cflag &= ~PARENB;  // set no parity
	port_settings.c_cflag &= ~CSTOPB;  // stop bits
	port_settings.c_cflag &= ~CSIZE;   // 8 bit data
	port_settings.c_cflag |= CS8;
	port_settings.c_cflag &= ~CRTSCTS; //No hadware hanshaking
	// apply the settings to the port
	tcsetattr(fd, TCSANOW, &port_settings);
	while(1)
	{
		n=read(fd,&result,sizeof(result));
		if(n>0)
		{
			printf("%c",result);
		}
	}
	close (fd);
	return 0;
}
```

- Biên dịch và chạy ứng dụng write.c trên máy host (dùng gcc)

- Biên dịch và chạy ứng dụng read.c trên KIT (dùng arm-linux-gcc)

Chạy 2 ứng dụng trên host và KIT (chú ý tham số tên cổng com truyền từ dòng lệnh)

Chú ý

- Tìm hiểu chi tiết về thư viện COM trên Linux (termios.h): Cấu trúc termios, thiết lập thông số, các hàm đọc ghi, …

https://www.cmrr.umn.edu/~strupp/serial.html

- Dựa trên tài liệu mô tả và ví dụ minh họa ở trên. Viết cặp ứng dụng trên Host và KIT thực hiện truyền/nhận dữ liệu (chuỗi ký tự) qua cổng COM.

3.2. Chương trình giao tiếp UART trên Raspberry Pi

```
// Compile : gcc -Wall uart-send.c -o uart-send -lwiringPi

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>

int main() {
	int fd;
	printf("Raspberry's sending : \n");
```

```
        while(1) {
                if((fd = serialOpen ("/dev/ttyAMA0", 9600)) < 0 ){
                        fprintf (stderr, "Unable to open serial
device: %s\n", strerror (errno));
                }
                serialPuts(fd, "hello");
                serialFlush(fd);
                printf("%s\n", "hello");
                fflush(stdout);
                delay(1000);
        }
        return 0;
}
```

Chương trình trên Pi sẽ gửi chuỗi ký tự "hello" thông qua uart tới arduino. Nó sẽ gửi liên tục sau 1s.

Thư viện sử dụng : #include <wiringSerial.h>

http://wiringpi.com/reference/serial-library/

- Mở cổng kết nối Serial.

  int **serialOpen** (char *device, int baud);

  Luôn nhớ rằng cổng uart có tên là ttyAMA0. Hàm này sẽ trả về file-descriptor. Nếu file-descritor lỗi sẽ có giá trị là -1.

- Hàm gửi dữ liệu

  Void **serialPutchar** (int fd, unsigned char c) ;

  Hàm này gửi sẽ gửi 1 byte. Nếu muốn gửi một mảng (hoặc chuỗi ký tự) có thể sử dụng hàm sau.

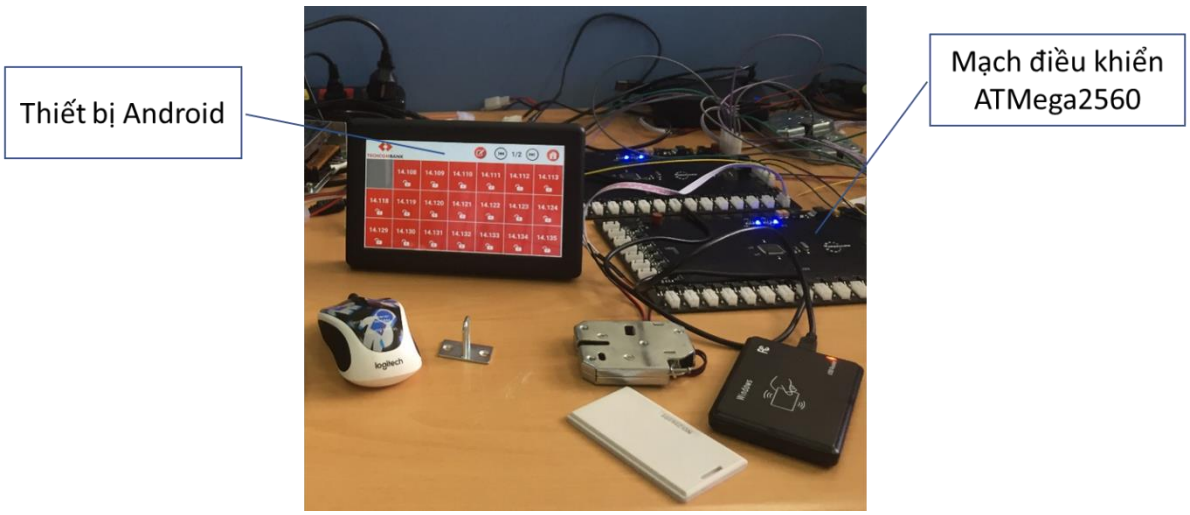
  void **serialPuts** (int fd, char *s) ;

- Hàm void serialFlush (int fd) : chờ đến khi dữ liệu được gửi xong hoặc bỏ qua tất cả dữ liệu được gửi tới. Hàm này rất cần với uart, nếu không có sẽ không thể gửi chính xác được. Bạn nên bỏ hàm này đi và xem thử. Có lẽ đây là một bug nhỏ đối với chương trình này vì đáng lý ra chương trình sau khi thực hiện lệnh gửi dữ liệu có thể thực hiện ngay lập tức lệnh tiếp theo, dữ liệu sẽ được uart tự động ngầm gửi đi. Chương trình không cần phải chờ.

Để compile chương trình C trên Rasberry Pi thực hiện lệnh

```
gcc -Wall uart-send.c -o uart-send -lwiringPi
```

Bài tập nâng cao: Tìm hiểu giao tiếp serial port trên thiết bị Android



Mở cổng COM:

```
File mSerialPort;
FileInputStream mSerR;
FileOutputStream mSerW;
try{
    mSerialPort = new File("/dev/ttyHSL1");
    mSerR = new FileInputStream(mSerialPort);
    mSerW = new FileOutputStream(mSerialPort);
}
catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Gửi dữ liệu ra cổng COM:

```
public void sendCommandToLocker(char cmdType, int portno){
    byte[] data = new byte[5];
    data[0] = 0x01;  //start byte
    data[1] = (byte)cmdType;
    data[2] = (byte)(portno + 0x40);
    data[3] = 0x00;
    data[4] = 0x02;  //end byte
    try {
        if(mSerW != null) mSerW.write(data);
    }
    catch(Exception e){
        e.printStackTrace();
    }
}
```

Nhận dữ liệu từ cổng COM:

```
byte[] recvdata = new byte[5];
int index = 0;
while (mSerR.available() > 0) {
    byte tmp = (byte) mSerR.read();
    if (index < 5) {
        recvdata[index++] = tmp;
    } else {
        for (int i = 0; i < 4; i++) {
```

```
            recvdata[i] = recvdata[i + 1];
        }
        recvdata[4] = tmp;
    }
}
```

Bài thí nghiệm 3

KỸ THUẬT LẬP TRÌNH NÂNG CAO CHO PHẦN MỀM NHÚNG: PROCESS, THREAD

1. Mục đích:

- Tìm hiểu về process (tiến trình) và thread (luồng) trên Linux, cơ chế giao tiếp giữa các tiến trình. Biết cách lập trình giao tiếp giữa 2 process, lập trình với ứng dụng đa luồng (mutlithreads).

2. Chuẩn bị:

- **PC** Linux (Ubuntu) with arm-linux-gcc
- **KIT** FriendlyArm mini/micro2440 hoặc Raspberry Pi

3. Nội dung thực hành

3.1. Tìm hiểu quản lý tiến trình trên Linux

Trên hệ điều hành Linux, tiến trình (process) được nhận biết thông qua số hiệu (định danh) tiến trình **pid.** Một tiến trình có thể nằm trong một nhóm nào đó, có thể nhận biết thông qua số hiệu nhóm **pgrp.** Một số hàm của C cho phép lấy các thông số này:

int getpid() : trả về giá trị int là pid của tiến trình hiện tại
int getppid() : trả về giá trị int là pid của tiến trình cha của tiến trình hiện tại
int getpgrp() : trả về giá trị int là số hiệu của nhóm tiến trình
int setpgrp() : trả về giá trị int là số hiệu nhóm tiến trình mới tạo ra

Ví dụ:

Lệnh *:* printf("Toi la tien trinh %d thuoc nhom %d",getpid(),getgrp());

Kết quả sẽ là: Toi là tien trinh 235 thuoc nhom 231

Hiển thị thông tin tiến trình – lệnh ps

Để biết thông tin các tiến trình hiện hành ta sử dụng lệnh: **ps [option]**

-e: hiển thị thông tin về mỗi tiến trình.

-l: hiển thị thông tin đầy đủ tiến trình.

-f: hiển thị thông tin về tiến trình cha.

-a: hiển thị tất cả các tiến trình.

Lưu ý: dòng lệnh **ps –aux**: liệt kê danh sách các tiến trình đang chạy cùng các thông tin của nó như: Chủ nhân của tiến trình (owner), mã số nhận diện tiến trình (PID), thời gian

hiện sử dụng CPU (%CPU), mức chiếm dụng bộ nhớ của tiến trình (%MEM), trạng thái tiến trình (STAT) và các thông tin khác.

Một số trạng thái của tiến trình thường gặp: R-đang thi hành, S-đang bị đóng, Z-ngừng thi hành, W-không đủ bộ nhớ…

Dừng một tiến trình – lệnh kill

Lệnh kill thường được sử dụng để ngừng thi hành một tiến trình.

kill [signal] <PID>

Trong đó: *signal:* là một số hay tên của tín hiệu được gửi tới tiến trình.

PID: mã số nhận diện tiến trình muốn dừng.

- Lệnh kill có thể gửi bất kỳ tín hiệu signal nào tới một tiến trình, nhưng theo mặc định nó gửi tín hiệu 15, TERM (là tín hiệu kết thúc tiến trình).
- Lệnh kill -9 PID: ngừng thi hành tiến trình mà không bị các tiến trình khác can thiệp (tín hiệu 9, KILL).
- Super-user mới có quyền dừng tất cả các tiến trình, còn người sử dụng chỉ được dừng các tiến trình của mình.

3.2. Lập trình với Process (tiến trình).

Bài 1. Xem thông tin số hiệu tiến trình.

Bước 1. Viết một tiến trình đơn giản, tham khảo mã nguồn sau (File **process1.c**):

```
#include<stdio.h>// standard input / output functions
#include <unistd.h>
#include <time.h>
int main(int argc, char** argv)
{
      printf("\nMa tien trinh dang chay : %d", (int)getpid());
      printf("\nMa tien trinh cha : %d", (int)getppid());
      while (1)
      {
            printf("\nRunning…");
            usleep(500);
      }
}
```

Trong tiến trình này, sử dụng các hàm getpid(), getppid() để lấy định danh của tiến trình đang chạy (chứa lời gọi hàm) và định danh của tiến trình cha của nó. Các định danh tiến trình là tham số quan trọng để cho phép các tiến trình giao tiếp với nhau qua cơ chế signal.

Bước 2. Biên dịch, thực hiện chương trình trên máy tính hoặc KIT, quan sát kết quả.

gcc –o process1 process1.c (Chạy trên máy tính)

Hoặc arm-linux-gcc –o process1 process1.c (Chạy trên KIT)

Bước 3. Dùng lệnh kill gửi signal tới tiến trình. Ví dụ kết thúc tiến trình:

kill SIGTERM <PID>

kill -9 <PID>

Bài 2. Tạo lập tiến trình mới dùng hàm system()

```
#include <stdlib.h>
#include <stdio.h>
int main(){
    printf("Hello!\n");
    system("ps -al");
    printf("Goodbye!\n");
}
```

Bài 3. Thay thế tiến trình mới bằng hàm exec()

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
    printf("Start\n\n");
    execlp("ps", "ps", "ax", 0);
    printf("Done.\n");
    exit(0);
}
```

Bài 4. Đón bắt và xử lý tín hiệu gửi tới tiến trình

Bước 1: Tìm hiểu cơ chế nhận signal, lập trình bắt signal với cấu trúc sigaction

Bước 2: Lập trình ứng dụng bắt các tín hiệu được gửi tới tiến trình (thử nghiệm với hai tín hiệu là SIGINT và SIGUSR1)

Viết chương trình như minh họa sau đây (file **process2.c**)

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
int count = 0;
//Hàm xử lý được gọi khi có tín hiệu SIGUSR1 gửi tới tiến trình
void handler(int signal_number)
{
      FILE *fid = fopen("~/output.txt", "a");  //Ghi thông tin ra tệp
      fprintf(fid, "\nNhan duoc tin hieu SIGUSR1 lan thu %  d", count++);
      fclose(fid);
}
int main()
{
```

```
	struct   sigaction   sa; //Khai báo một biến cấu trúc sigaction
	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));
	//Gán con trỏ hàm xử lý signal cho trường sa_handler của biến cấu trúc
sa
	sa.sa_handler = &handler;
	//Đăng ký cấu trúc sa cho xử lý tín hiệu (signal) SIGUSR1
	sigaction(SIGUSR1, &sa, NULL);
	//Chương trình chính liên tục in ra chữ A
	while (1)
	{
		printf("A");
		sleep(1);
	}
	return  0;
}
```

Bước 3. Biên dịch và chạy chương trình trên máy tính:

gcc –o process2 process2.c

Thực thi tiến trình này, quan sát kết quả, xem số hiệu pid

Mở cửa sổ terminal khác, gửi tín hiệu SIGUSR1 đến tiến trình này bằng lệnh linux:

kill SIGUSR1 pid

(pid là số hiệu tiến trình process2 đang chạy)

Thử gửi tín hiệu này vài lần và mở file output.txt (Thư mục $HOME) quan sát kết quả.

Ghi chú: Có thể sử dụng hàm kill() trong lập trình C để gửi tín hiệu đến một tiến trình.

Thư viện: #include <signal.h>

int kill(pid_t pid, int sig);

(Viết một chương trình khác sử dụng hàm này, tham số pid của tiến trình muốn gửi signal đến được truyền từ dòng lệnh).

3.3. Lập trình đa luồng

3.3.1. Cơ bản về tạo luồng

Viết ứng dụng đơn giản thực hiện tạo luồng (dùng hàm pthread_create)

(**File: thread1.c**)

```
#include <stdio.h>
#include <pthread.h>
//Hàm xử lý của luồng thực hiện liên tục in chữ x
void* printx(void* unused)
{
	while (1)
	{
```

```
            fputc('x', stdout);
      }
      return NULL;
}
int main(int argc, char** argv)
{
      pthread_t thread_id;
      //Tạo ra một luồng mới với hàm xử lý luồng là printx
      pthread_create(&thread_id, NULL, &printx, NULL);
      //Chương trình chính liên tục in chữ o
      while (1)
      {
            fputc('o', stdout);
      }
      return 0;
}
//Trong ví dụ này cả chương trình chính là luồng đều chạy lặp vô hạn
```

Biên dịch và chạy ví dụ này trên PC hoặc trên KIT và quan sát kết quả.

3.3.2. Truyền dữ liệu cho luồng

Chú ý: Hàm xử lý của luồng có kiểu tham số là void*, vì vậy để truyền nhiều thành phần dữ liệu cho luồng cần tạo một cấu trúc (struct), và truyền cho hàm xử lý của luồng tham số là biến cấu trúc này.

Ví dụ sau minh họa tạo ra 2 luồng cùng sử dụng hàm xử lý in một số lượng ký tự ra màn hình, với tham số truyền vào là ký tự và số lượng muốn in.

File: **thread2.c**

```
#include <pthread.h>
#include <stdio.h>
//Cau truc la tham so cho ham xu ly luong (ham char_print)
struct char_print_parms
{
      char character; //Ky tu muon in
      int count; //So lan in
};
//Ham xu ly cua thread
//In ky tu ra man hinh, duoc cho boi tham so la mot con tro den cau truc du
lieu tren
void* char_print(void* params)
{
      //Tham so truyen vao la kieu void* duoc ep thanh kieu nhu struct da
khai bao
      struct char_print_parms* p = (struct char_print_parms*) params;
      int i;
      int n = p->count; //Bien chua so lan in ra
      char c = p->character; //Bien chua ma ky tu muon in ra
      for (i = 0; i < n; i++)
            fputc(c, stdout); //Ham in 1 ky tu ra thiet ra chuan
      return NULL;
}
```

```
int main(int argc, char** agrv)
{
	pthread_t  thread1_id, thread2_id; //Khai báo 2 biến định danh luồng
	struct char_print_parms  p1, p2; //2 bien tham so truyen cho ham xu ly
cua thread
	//Tao 1 thread in 30000 chu 'x'
	p1.character = 'x';
	p1.count = 30000;
	pthread_create(&thread1_id, NULL, &char_print, &p1);
	//Tao 1 thread khac in ra 20000 chu 'o'
	p2.character = 'o';
	p2.count = 20000;
	pthread_create(&thread2_id, NULL, &char_print, &p2);
	//Dam bao thread1 da ket thuc
	pthread_join(thread1_id, NULL);
	//Dam bao thread2 da ket thuc
	pthread_join(thread2_id, NULL);
	// Now we can safely return.
	return 0;
}
```

Tìm hiểu, biên dịch và chạy ví dụ trên trên máy tính hoặc KIT

3.3.3. Bài tập nâng cao: Chương trình giao tiếp button, led sử dụng đa luồng

Mục đích: Áp dụng cơ chế luồng, viết một ứng dụng trên KIT thực hiện hiệu ứng led đuổi, sử dụng nút bấm K1, K2 để thay đổi (giảm/tăng) tốc độ của hiệu ứng led đuổi.

Chú ý:

- Chương trình chính (master thread) thực hiện hiệu ứng led đuổi trong một vòng lặp vô hạn, hiệu ứng dựa trên viết bật/tắt từng led với thời gian trễ (delay) thích hợp (giả sử là t milisecond, ban đầu mặc định là 1000ms=1s)
- Thời gian trễ nói trên có thể điều chỉnh (tăng/giảm thích hợp) khi bấm nút K1, K2.
- Vì button device driver cho phép giao tiếp kiểu thăm dò (polling), cần sử dụng một luồng riêng để thực hiện công việc này (song song với công việc chính là điều khiển hiệu ứng nháy led), hàm xử lý của luồng sẽ đọc K1, K2 có được ấn để thay đổi giá trị t tương ứng.

Tiến hành: Tham khảo mã nguồn ví dụ sau, bổ sung các chỗ còn thiếu biên dịch và chạy ứng dụng trên KIT

(**File: threadbuttonled.c**)

```
/************************************************************
* main:    Thuc hien dieu khien led chay duoi
* thread:  Thuc hien doc (polling) trang thai nut bam de thay doi toc do led
************************************************************/
```

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>
#define ON  1
#define OFF 0
/* Bien luu thoi gian delay, sẽ thay doi khi K1, K2 duoc an */
static int t = 1000; //don vi la milisecond, ban dau mac dinh la 1000 ms
//Nếu sử dụng cơ chế mutex
//pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
/* Ham sleep ms su dung usleep cua linux */
void sleepms(int ms)
{
	//usleep in us, sleep in second
	usleep(1000 * ms); //convert to microseconds
	return;
};
/* Cau truc du lieu se truyen tham so cho thread */
struct thread_parms
{
	char btn[6];
	int btn_fd;
};

/* Ham xu ly cua thread thuc hien doc nut K1, K2 (polling) */
void* btn_polling(void* param);
int main(int argc, char** agrv)
{
	int led_fd, button_fd; //định danh file thiết bị (led, button)
	struct thread_parms  p; //Bien cau truc chua tham so se truyen cho ham
xu ly cua thread
	pthread_t    thread_id;   //định danh luồng
	int led_no; //So hieu led 0-4
	//Mang chua gia tri trang thai 6 button se doc
	char buttons[6] = { '0', '0', '0', '0', '0', '0' };
	int i;
	//Mo thiet bi (led port), can kiem tra chinh xac ten trong /dev
	Led_fd = open(“ / dev / leds”, 0);
	if (led_fd < 0) {
		perror("open device leds");
		exit(1);
	}
	else printf("open device led ok\n");

	//Mo thiet bi (button port)
	button_fd = open(“ / dev / buttons”, 0); //mo button port

	if (button_fd < 0)
```

```
	{
		perror("open device buttons");
		exit(1);
	}
	else printf("open device button ok\n");

	//Chuan bi tham so truyen vao cho ham xu ly cua thread
	p.btn_fd = button_fd; // button device file number
	for (i = 0; i < 6; i++) p.btn[i] = '0'; //buffer to read button status
(K1-K6)

	//Tao thread thuc hien polling button
	thread_id = pthread_create(&thread_id, NULL, &btn_polling, &p);;
	//Chuong trinh chinh (master thread) thuc hien hieu ung led duoi
	//voi thoi gian delay chua trong bien t
	//Ban dau tat ca cac led deu off
	for (i = 0; i < 4; i++) ioctl(led_fd, OFF, i);
	led_no = 0;
	while (1)
	{
		//Bat led so hieu led_no
		ioctl(led_fd, ON, led_no);
		//Sleep in t ms
		sleepms(t);
		//Tat led so hieu led_no
		ioctl(led_fd, OFF, led_no);
		led_no++;
		if (led_no == 4) led_no = 0;
	}
	close(button_fd);
	close(led_fd);
	return 0;
}
void* btn_polling(void* param)
{
	struct thread_parms* p = (struct thread_parms*)param;
	char cur_btn[6], old_btn[6];
	int btn_fd;
	int i;
	for (i = 0; i < 6; i++) old_btn[i] = p->btn[i];
	btn_fd = p->btn_fd;
	for (;;)//Lien tuc tham do trang thai nut bam (K1, K2 co duoc an)
	{
		int num = read(btn_fd, cur_btn, sizeof(cur_btn)); //doc button
device file
		if (num != sizeof(cur_btn))
		{
			perror("read buttons:");
			exit(1);
		}
		//Chi can doc K1, K2 tuong ung voi tang/giam led speed
		//Doc K1
		if (old_btn[0] != cur_btn[0]) //Nếu K1 được ấn
		{
			old_btn[0] = cur_btn[0];
```

```
                printf("K1 is pressed/released\n");
                //Neu dung mutex cho bien t
                //pthread_mutex_lock( &mutex1 );
                t = t + 50;//tang thoi gian delay (so với t hiện tại)
                //pthread_mutex_unlock( &mutex1 );
            }
            //Doc K2
            if (old_btn[1] != cur_btn[1]) //Nếu K2 được ấn
            {
                old_btn[1] = cur_btn[1];
                printf("K2 is pressed/released\n");
                //pthread_mutex_lock( &mutex1 );
                t = t - 50;//giam thoi gian delay (so với t hiện tại)
                if (t < 100) t = 100;
                //pthread_mutex_unlock( &mutex1 );
            }
        }
        return NULL;
}
```