Bài thí nghiệm 2

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 nền tảng 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** Ubuntu (hoặc máy ảo Ubuntu)
- **KIT** FriendlyArm mini/micro2440 hoặc Raspberry Pi

3. Nội dung thực hành

3.1. 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).

3.2. Lập trình đa luồng

3.2.1. 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.

3.2.2. 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

3.2.3. 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;
}
```