ปฏิบัติการบน RaspberryPi ครั้งที่ 3: การจัดการอินเทอร์รัปต์และสัญญาณในระบบ

ปฏิบัติการ: การตอบสนองอินเทอร์รัปต์จาก GPIO

ขา GPIO ของ raspberry pi สามารถโปรแกรมให้มีการกระตุ้นให้เกิดอินเทอร์รัปต์ และเราสามารถเขียนฟังก์ชันจัดการอินเทอร์รัปต์ (ISR Interrupt Service Routine) เพื่อตอบสนองต่อการกระตุ้นดังกล่าวได้

ไลบรารี pigpio มีฟังก์ชันที่ใช้ในการจัดการอินเทอร์รัพต์ดังนี้

```
int gpioSetISRFunc(unsigned gpio, unsigned edge, int timeout, gpioISRFunc_t f);
int gpioSetISRFuncEx(unsigned gpio, unsigned edge, int timeout,
                     gpioISRFuncEx_t f,void *userdata);
```

pin	ขา GPIO ที่ต้องการจัดการอินเทอร์รัปต์	
edgeType	กำหนดจังหวะการตอบสนองอินเทอร์รัปต์ มีตัวเลือกดังนี้	
	FALLING_EDGE	ตอบสนองที่ขอบขาลง
	RISING_EDGE	ตอบสนองที่ขอบขาขึ้น
	EITHER_EDGE	ตอบสนองทั้งขอบขาขึ้นและลง
timeout	คาบเวลาที่จะเกิดอินเทอร์รัปต์ในกรณีที่หมดเวลา หน่วยเป็นมิลลิวินาที ใช้เพื่อสั่งให้เกิดอินเทอร์รัปต์ หากรอไปจนครบคาบเวลาที่กำหนดแล้วยังไม่เกิดอินเทอร์รัปต์เกิดขึ้น	
f	ค่าอ้างอิงไปยังฟังก์ชันจัดการอินเทอร์รัปต์ที่กำหนดไว้ภายในโปรแกรม โปรโตไทป์และหัวของอินเทอร์รัปต์ฟังก์ชันต้องมีรูปแบบดังนี้	

สำหรับฟังก์ชัน gpioSetISRFunc()

void ชื่ออินเทอร์รัปต์ฟังก์ชัน (int gpio, int level, unit32_t tick);

ค่าที่รับมาเพื่อใช้ทำงานภายในอินเทอร์รัปต์ฟังก์ชันมีดังนี้

gpio	หมายเลขพอร์ต GPIO ที่เกิดอินเทอร์รัปต์	
level	สถานะที่เปลี่ยนไปเมื่อเกิดอินเทอร์รัปต์	
	0	เกิดอินเทอร์รัปต์ที่ขอบขาลง
	1	เกิดอินเทอร์รัปต์ที่ขอบขาขึ้น
	2	เกิดอินเทอร์รัปต์จากการหมดเวลา
tick	จำนวนหน่วยเวลานับจากเมื่อเปิดทำงาน Raspberry Pi หน่วยเป็นไมโครวินาที เนื่องจากขนาดเป็น 32 บิต ค่าจะวนกลับมาเริ่มต้นใหม่ทุกๆ ประมาณ สี่พันล้านหน่วยเวลา หรือประมาณทุกๆ 72 นาที	

สำหรับฟังก์ชัน gpioSetISRFuncEx()

userdata ตัวชี้ไปยังตัวแปรที่จะใช้ผ่านค่าไปยังฟังก์ชันอินเทอร์รัปต์

ส่วนตัวอินเทอร์รัปต์ฟังก์ชันจะต้องมีโปรโตไทป์และหัวฟังก์ชันเป็นดังนี้

void ชื่ออินเทอร์รัปต์ฟังก์ชัน (int gpio, int level, unit32_t tick, void *userdata);

ค่าที่รับมาเพื่อใช้ทำงานภายในอินเทอร์รัปต์ฟังก์ชัน นอกเหนือจากรายละเอียดจากที่ใช้กับ gpioSetISRFunc() แล้วมีดังนี้

userdata ตัวชี้ที่ได้รับค่าอ้างอิงของตัวแปรที่ใช้ผ่านค่าเข้ามา

ตัวอย่างต่อไปนี้ เป็นตัวอย่างง่ายๆ สำหรับทดลองการตอบสนองอินเทอร์รัปต์เมื่อมีการเปลี่ยนสถานะของขา GPIO ให้นักศึกษาจัมป์สายจากขา GPIO ที่ต้องการ (ตัวอย่างโปรแกรมใช้ GPIO20) และทดลองการเกิดอินเทอร์รัปต์โดยต่อปลายสายลงกราวนด์ (สถานะปกติให้ปล่อยลอยปลายสายไว้)

<u>อุปกรณ์ที่ต้องการ</u>

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด

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

int key = 27;
int count = 0;

void isrFunc(int gpio,int level,uint32_t tick);

int main(){
    if(gpioInitialise()<0) return -1;

    gpioSetMode(key,PI_INPUT);
    gpioSetPullUpDown(key,PI_PUD_UP);
    printf("Set G27 as input\n");

    gpioSetISRFunc(key,FALLING_EDGE,10000,isrFunc);

    while(count<10){
        printf("Key pressed : %2d\r",count);
        fflush(stdout);
        usleep(250000);
    }

    gpioTerminate();
    return 0;
}

void isrFunc(int gpio,int level,uint32_t tick){
   count++;
}
```

ปฏิบัติการ: การแก้ไขปัญหา bouncing ของสัญญาณอินพุต

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด

จากปฏิบัติการที่แล้ว นักศึกษาจะเห็นว่า เวลาที่ต่อสายลงกราวนด์(หรือเทียบเท่ากับการกดปุ่ม) บ่อยครั้งที่ค่าตัวนับขึ้นมากกว่าหนึ่งในคราวเดียว ซึ่งเกิดมาจากปัญหา key bounce

เราสามารถลดปัญหาโดยการทำ debouncing ได้โดยการอ่านค่าฐานเวลามาเก็บไว้เมื่อกดปุ่มแต่ละครั้ง และเทียบกับครั้งสุดท้ายที่เราเปลี่ยนแปลงค่าจริงๆ ถ้าค่าฐานเวลาครั้งที่เราเปลี่ยนค่าครั้งสุดท้ายไม่แตกต่างจากค่าที่อ่านครั้งหลังสุด เราก็จะไม่ถือว่าเป็นการกดปุ่ม (จากในโปรแกรมตัวอย่าง เราก็จะไม่เพิ่มค่า count เป็นต้น

อาศัยแนวคิดดังกล่าว จงแก้ไขโปรแกรมข้างบนในส่วนของฟังก์ชัน myISR() เพื่อแก้ปัญหา key bouncing

HINT *นักศึกษาลองนำค่าฐานเวลาที่อ่านได้ มาแสดงบนหน้าจอ เพื่อดูว่าค่าความแตกต่างของฐานเวลาควรจะเป็นเท่าใดที่เหมาะสม*

การจัดการสัญญาณเหตุการณ์ในลินุกซ์

ลินุกซ์มีกลไกการจัดการกับสัญญาณหรือเหตุการณ์ต่างๆ ที่เกิดขึ้นภายในระบบโดยอาศัยฟังก์ชัน signal() ซึ่งมีโปรโตไทป์ที่น่าสนใจดังนี้

```
sighandler_t signal(int signum, sighandler_t handler);
```

(นิยามใน signal.h)

signum คือหมายเลขสัญญาณที่ต้องการตรวจจับ เลขสัญญาณที่น่าสนใจมีดังนี้

- SIGHUP สัญญาณที่เกิดขึ้นเมื่อเทอร์มินัลที่ใช้เพื่อรับส่งข้อมูลกับโพรเซสหยุดการติดต่อ เช่นกรณีที่เชื่อมต่อเทอร์มินัลผ่านทางระบบเครือข่าย ถ้าระบบเครือข่ายหลุดก็จะเกิดสัญญาณนี้ขึ้น
- SIGTERM สัญญาณที่เกิดขึ้นเมื่อระบบปฏิบัติการสั่งหยุดการทำงานของโพรเซส เช่นการใช้คำสั่ง kill โพรเซสจากระบบปฏิบัติการ
- SIGINT สัญญาณที่เกิดขึ้นเมื่อผู้ใช้กดปุ่ม CTRL-C ในระหว่างการทำงานของโพรเซสเพื่อสั่งหยุดการทำงาน
- SIGUSR1 และ SIGUSR2 สัญญาณที่เกิดขึ้นจากการสั่งงานโดยโพรเซสในระบบ

handler คือค่าอ้างอิงไปยังฟังก์ชันจัดการสัญญาณเหตุการณ์ ซึ่งต้องกำหนดโปรโตไทป์ดังนี้

```
void ชื่อฟังก์ชัน(int ชื่อตัวแปร);
```

ฟังก์ชันมีอาร์กิวเมนต์หนึ่งตัว ซึ่งจะรับค่ารหัสสัญญาณที่เกิดขึ้น เพื่อใช้ในการจัดการภายใน โดยเฉพาะในกรณีที่เขียนฟังก์ชันตัวเดียวเพื่อจัดการกับสัญญาณหลายๆ ตัวพร้อมกัน ก็จะสามารถใช้อาร์กิวเมนต์นี้เพื่อแยกแยะกลไกการจัดการกับสัญญาณที่แตกต่างกันไปได้

สำหรับกรณีที่ดักจับสัญญาณ SIGUSR1 หรือ SIGUSR2 นั้น ก็จะต้องอาศัยโพรเซสพ่อแม่ หรือเธรดหลัก ในการสร้างสัญญาณขึ้นมา การส่งสัญญาณจากโพรเซสพ่อแม่ไปยังโพรเซสลูก หรือการส่งสัญญาณจากเธรดหลักไปยังเธรดย่อย ทำได้โดยใช้ฟังก์ชันต่อไปนี้

`int kill (pid_t pid, int signum)`

ใช้ส่งสัญญาณไปยังโพรเซสลูก (ที่สร้างขึ้นด้วย fork()) เพื่อให้โพรเซสลูกสามารถดักสัญญาณแล้วนำไปใช้ อย่างกรณีเพื่อขอให้โพรเซสลูกจบการทำงาน เป็นต้น

pid	หมายเลขโพรเซสไอดีของโพรเซสลูกที่จะส่งสัญญาณไป (เราสามารถใช้ getpid() เพื่อส่งสัญญาณกลับมายังโพรเซสตนเองได้เช่นกัน)
signum	สัญญาณที่จะต้องการจะส่งไป (สามารถส่ง SIGUSR1, SIGUSR2 หรือสัญญาณอื่นๆ อย่างเช่น SIGINT ไปก็ได้เช่นกัน
ค่ากลับคืน	เป็น 0 กรณีสำเร็จ หรือ -1 หากเกิดความผิดพลาด

`int pthread_kill (pid_t tid, int signum)`

ใช้ส่งสัญญาณจากเธรดหลักไปยังเธรดย่อยที่สร้างขึ้น ใช้ในลักษณะคล้ายคลึงกันกับการส่งจากโพรเซสพ่อแม่ไปยังโพรเซสลูก แต่ในกรณีนี้เป็นการส่งสัญญาณไปยังเธรดย่อย เพื่อให้เธรดย่อยทำงานตามที่ต้องการ เช่นการขอให้เธรดย่อยจบการทำงาน เป็นต้น

tid	หมายเลขของเธรดไอดีปลายทางที่จะส่งสัญญาณไป
signum	สัญญาณที่จะต้องการจะส่งไป
ค่ากลับคืน	เป็น 0 กรณีสำเร็จ หรือ -1 หากเกิดความผิดพลาด

`int raise (int signum)`

ใช้ส่งสัญญาณจากโพรเซสไปยังตนเอง หรือจากเธรดใดๆ ไปยังตนเอง

signum	สัญญาณที่จะต้องการจะส่งไป
ค่ากลับคืน	เป็น 0 กรณีสำเร็จ หรือ -1 หากเกิดความผิดพลาด

`int sigemptyset (sigset_t *set)`

ใช้กำหนดสภาพเริ่มต้น หรือล้างรายชื่อสัญญาณที่จะจัดการ

set	ตัวชี้เก็บค่าอ้างอิงของตัวแปรที่เก็บชุดสัญญาณที่จะจัดการ
ค่ากลับคืน	เป็น 0 กรณีสำเร็จ หรือ -1 หากเกิดความผิดพลาด

`int sigaddset (int signum)`

ใช้เพิ่มสัญญาณที่ต้องการเข้าไปในเซ็ตรายชื่อสัญญาณ

signum	สัญญาณที่จะต้องการจะเพิ่มลงในรายชื่อ
ค่ากลับคืน	เป็น 0 กรณีสำเร็จ หรือ -1 หากเกิดความผิดพลาด

`int sigwait (sigset_t *set,int *signum)`

ใช้หยุดเธรดเพื่อรอสัญญาณตามรายชื่อสัญญาณที่ให้ไว้.3

set	ตัวชี้เก็บค่าอ้างอิงของตัวแปรที่เก็บชุดสัญญาณที่จะจัดการ
signum	สัญญาณที่รับได้
ค่ากลับคืน	เป็น 0 กรณีสำเร็จ หรือ -1 หากเกิดความผิดพลาด

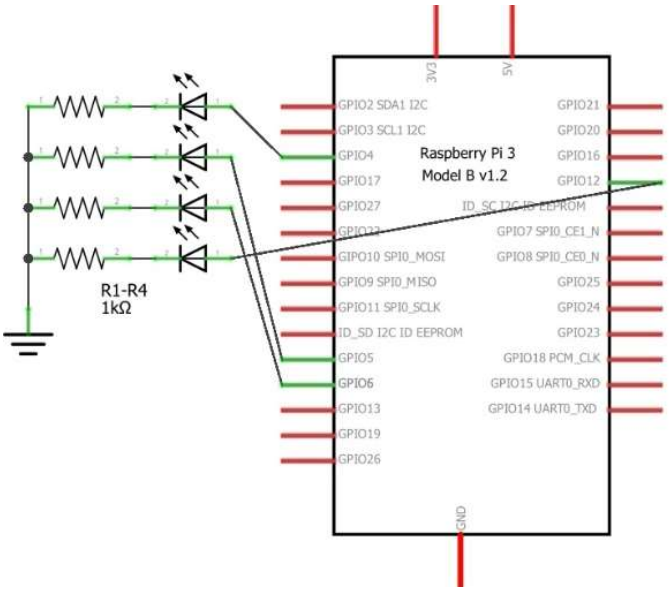
ปฏิบัติการ: การสั่งหยุดการทำงานไลบรารี pigpio เมื่อผู้ใช้กด CTRL-C

ไลบรารี pigpio ถูกออกแบบมาในลักษณะที่ควรให้มีการสั่งจบการทำงานไลบรารี เพื่อจะได้มีการคืนทรัพยากรแก่ระบบอย่างถูกต้อง โดยการเรียกใช้ฟังก์ชัน gpioTerminate() และนอกจากนี้ ในหลายสภาพของการเขียนโปรแกรม เราอาจจะต้องกำหนดให้ผู้ใช้หยุดการทำงานของกลไกบางอย่างของระบบ หรือคืนทรัพยากรให้กับระบบ (หรืออื่นๆ เช่นการปิดไฟล์) ก่อนที่จะหยุดโปรแกรม แต่เมื่อผู้ใช้กดปุ่ม CTRL-C เพื่อหยุดการทำงานของโปรแกรมอย่างกะทันหัน โปรแกรมจะไม่มีโอกาสได้ทำงานต่างๆ ดังที่ได้กล่าวมาก่อนหน้านี้

ระบบปฏิบัติการลินุกซ์ จึงมีชุดฟังก์ชันที่ใช้ในการจัดการสัญญาณเหตุการณ์ต่างๆ ที่เกิดขึ้นในระบบ และเราสามารถอาศัยการดักจับสัญญาณต่างๆ เหล่านี้ เพื่อดำเนินการตามกลไกที่ควรจะเป็นได้

ในตัวอย่างต่อไปนี้ จะเห็นถึงการสร้างฟังก์ชันตอบสนองสัญญาณชื่อ gpio_stop() เพื่อใช้ตอบสนองต่อสัญญาณที่ผู้ใช้กดปุ่ม CTRL-C และนำไปใช้เรียกฟังก์ชัน gpioTerminate() เพื่อให้การจบการทำงานของไลบรารี pigpio เป็นไปอย่างที่ควรจะเป็น

```c
#include <pigpio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int ledGPIO[4]={4,5,6,12};

void initGPIO();
void blinkingLED();

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    gpioTerminate();
    exit(0);
}

int main(){

    initGPIO();
    signal(SIGINT,gpio_stop);
    blinkingLED();
    gpioTerminate();
    return 0;
}

void initGPIO(){
    int i;
    if(gpioInitialise() < 0) exit(1);

    for(i=0;i<4;i++)
        gpioSetMode(ledGPIO[i],PI_OUTPUT);
}

void blinkingLED(){
   int i,j,k;
    int pattern[2][4]={{1,0,1,0},{0,1,0,1}};

    for(k=20;k>0;k--){
        for(i=0;i<2;i++){
            for(j=0;j<4;j++)
                gpioWrite(ledGPIO[j],pattern[i][j]);
                    usleep(250000);
        }
        printf(" %d \r",k);
        fflush(stdout);
```



รูปประกอบการทำงานของโปรแกรม
และโปรแกรมถัดๆไปในปฏิบัติการครั้งนี้

```
    }
    for(j=0;j<4;j++)
        gpioWrite(ledGPIO[j],0);
    printf("Program exits normally.");
}
```

จากโปรแกรมตัวอย่าง มีกลไกการทำงานที่น่าสนใจดังนี้

- ในฟังก์ชัน main() หลังการเริ่มต้นเรียกใช้งาน gpioInitilise() (จากภายในฟังก์ชัน initGPIO() ที่เราสร้างขึ้น) เราเรียกใช้ฟังก์ชัน signal() โดยดักจับสัญญาณ SIGINT (ผู้ใช้กดปุ่ม CTRL-C) และส่งผ่านค่าอ้างอิงฟังก์ชัน gpio_stop() เป็นอาร์กิวเมนตีที่สอง
- ภายในฟังก์ชัน gpio_stop() ที่ใช้ตอบสนองสัญญาณ SIGINT เราเรียกใช้ฟังก์ชัน gpioTerminate() เพื่อจบการทำงานไลบรารี pigpio
- สังเกตการนิยามหัวฟังก์ชัน gpio_stop() ที่มีอาร์กิวเมนต์หนึ่งตัว อาร์กิวเมนต์นี้จะรับค่าหมายเลขสัญญาณ ซึ่งในกรณีนี้ก็จะได้ค่าสัญญาณ SIGINT เข้ามา แต่ในที่นี้เราไม่ได้นำมาใช้งาน อาร์กิวเมนต์ตัวนี้จะมีประโยชน์ในกรณีที่เราใช้ฟังก์ชันตอบสนองสัญญาณนี้ต่อการดักจับสัญญาณมากกว่าหนึ่งตัวพร้อมๆ กัน เราสามารถตรวจสอบค่าตัวแปรอาร์กิวเมนต์นี้เพื่อจะได้เขียนคำสั่งตอบสนองเฉพาะต่อเหตุการณ์ที่แตกต่างกันออกไปได้

ปฏิบัติการ: การใช้สัญญาณ INTUSR (1/2)

ตัวอย่างต่อไปนี้เป็นการนำเอา SIGUSR1 มาใช้เพื่อทำหน้าที่กำหนดสถานะเพื่อให้เธรดจบการทำงานอย่างเรียบร้อย โดยในที่นี้เมื่อเธรดฟังก์ชันตรวจสอบสถานะตัวแปร stop ว่าเป็นจริง (จากการเซ็ตโดยฟังก์ชันตอบสนองสัญญาณ SIGUSR1) ก็จะหลุดออกจากวนรอบ while(1) และมาดับ LED ทั้งสี่ดวงก่อนจบการทำงาน

อนึ่ง ในตัวอย่างนี้เป็นการสร้างสัญญาณแบบง่ายๆ โดยอาศัยการดักจับจากสัญญาณ CTRL-C ซึ่งมีฟังก์ชันตอบสนองสัญญาณ gpio_stop() ตอบสนองต่อสัญญาณ ซึ่งจริงๆ แล้วในที่นี้สามารถนำชุดข้อความสั่งใน thread_stop() ไปใส่ไว้ใน gpio_stop() ก็ได้ แต่ในทางปฏิบัตินั้น เราอาจจะรับสัญญาณอินเทอร์รัปต์มาจากแหล่งอื่น เช่นจากขา GPIO ที่กำหนดไว้ ในกรณีเช่นนี้เราก็อาจจะเขียนฟังก์ชันจัดการอินเทอร์รัปต์ ที่มีการสร้างสัญญาณ INTUSR ต่อไปยังโพรเซสหรือเธรดในส่วนต่างๆ ต่อไปได้อีก

```
#include <pigpio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

int ledGPIO[4]={4,5,6,12};
int stop = false;
void initGPIO();

void *blinkingLED(void *param);

void gpio_stop(int sig){
    printf("User pressing CTRL-C\n");
       kill(getpid(),SIGUSR1);
}

void thread_stop(int sig){
    printf("Recieving USR signal\n");
    stop = true;
}

int main(){
    pthread_t tid;
    pthread_attr_t attr;
```

```
    int i;

    initGPIO();

    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,blinkingLED,NULL);

    printf("Waiting all threads to stop...\n");
    pthread_join(tid,NULL);
    pthread_attr_destroy(&attr);

       gpioTerminate();
    return 0;
}

void initGPIO(){
    int i;
    if(gpioInitialise() < 0) exit(1);
    signal(SIGINT,gpio_stop);
       signal(SIGUSR1,thread_stop);
    for(i=0;i<4;i++)
        gpioSetMode(ledGPIO[i],PI_OUTPUT);
}

void *blinkingLED(void *param){
    int i,j;
    int pattern[2][4]={{1,0,1,0},{0,1,0,1}};

    while(1){
        for(i=0;i<2;i++){
            for(j=0;j<4;j++)
                gpioWrite(ledGPIO[j],pattern[i][j]);
                    usleep(250000);
        }
        if(stop) break;
    }
    for(j=0;j<4;j++)
        gpioWrite(ledGPIO[j],0);
    pthread_exit(0);
}
```

ปฏิบัติการ: การกระตุ้นให้เกิดสัญญาณในระบบ

ตัวอย่างต่อไปนี้เป็นการทดลองใช้ pthread_kill() เพื่อส่งสัญญาณเจาะจงลงไปยังเธรดปลายทางที่ต้องการ (ในกรณีที่ใช้ kill() จะส่งสัญญาณให้ทั้งโพรเซสได้รับ) และมีการใช้ raise() เพื่อกระตุ้นให้เกิดสัญญาณตามที่ต้องการ ซึ่งในที่นี้เราเขียนชุดคำสั่งวนรอบรอเวลา 10 วินาที หากผู้ใช้ยังไม่กดปุ่ม CTRL-C ก็จะสร้างสัญญาณ INTUSR1 ขึ้นมาเองจากเธรดหลัก (จากฟังก์ชัน main())

```
#include <pigpio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

int ledGPIO[4]={4,5,6,12};
int stop = false;
void initGPIO();
pthread_t tid;

void *blinkingLED(void *param);
```

```
void gpio_stop(int sig){
    printf("User pressing CTRL-C\n");
       pthread_kill(tid,SIGUSR1);
}

void thread_stop(int sig){
    printf("Recieving USR signal\n");
    stop = true;
}

int main(){
    pthread_attr_t attr;
    int i;

    initGPIO();

    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,blinkingLED,NULL);

    printf("Waiting LED thread to stop (using CTRL-C or waiting for time out)...\n");
    for(i=10;i>1;i--){
        printf("in %d second(s) \r",i);
        fflush(stdout);
        sleep(1);
    }
    raise(SIGUSR1);
    pthread_join(tid,NULL);
    pthread_attr_destroy(&attr);

       gpioTerminate();
    return 0;
}

void initGPIO(){
    int i;
    if(gpioInitialise() < 0) exit(1);
    signal(SIGINT,gpio_stop);
    signal(SIGUSR1,thread_stop);
    for(i=0;i<4;i++)
        gpioSetMode(ledGPIO[i],PI_OUTPUT);
}

void *blinkingLED(void *param){
    int i,j;
    int pattern[2][4]={{1,0,1,0},{0,1,0,1}};

    while(1){
        for(i=0;i<2;i++){
            for(j=0;j<4;j++)
                gpioWrite(ledGPIO[j],pattern[i][j]);
                    usleep(250000);
        }
        if(stop) break;
    }
    for(j=0;j<4;j++)
        gpioWrite(ledGPIO[j],0);
    pthread_exit(0);
}
```

ปฏิบัติการ: การสั่งให้เธรดหยุดรอสัญญาณ

ตัวอย่างสุดท้ายในปฏิบัติการครั้งนี้ เป็นการทดลองใช้ฟังก์ชัน sigwait() เพื่อให้เธรดรอรับสัญญาณที่จะเกิดขึ้นภายในระบบ ก่อนที่จะทำงานในขั้นตอนต่อไป ทั้งนี้เนื่องจากในการเขียนโปรแกรมหลายเธรด เรามักจะแบ่งหน้าที่การทำงานของเธรดให้รองรับการทำงานเป็นส่วนๆ ดังนั้นเธรดหลายเธรดที่สร้างขึ้นจึงมีลักษณะการทำงานเป็นวนรอบ และเมื่อจบการทำงานหนึ่งชุดแล้ว ก็จะวนรอการทำงานในชุดต่อไป การวนรอตามปกติเราอาจจะใช้หลักการ polling หรือการเขียนชุดคำสั่งทำงานเพื่อวนรอสถานะไปจนกว่าจะเกิดสถานะดังกล่าวขึ้น แล้วจึงค่อยทำงานต่อไป แต่ในที่นี้เราอาศัยการใช้ sigwait() เพื่อให้เกิดการรอคอยของเธรด จนกว่าจะได้รับสัญญาณที่กำหนด จึงค่อยเกิดการเริ่มต้นการทำงานในรอบใหม่

การใช้ sigwait() เริ่มต้นจากการนิยามตัวแปรสำหรับเก็บชุดข้อมูลสัญญาณโดยนิยามให้มีแบบชนิดเป็น sigset_t จากนั้นใช้ฟังก์ชัน sigemptyset() เพื่อล้างลิสต์/กำหนดสภาวะเริ่มต้นของลิสต์ แล้วใช้ sigaddset() เพื่อเพิ่มสัญญาณที่ต้องการดักจับเข้าไป ซึ่งนั่นหมายความว่า สามารถกำหนดให้การรอคอยด้วย sigwait() นั้นรอคอยสัญญาณได้มากกว่าหนึ่งประเภทสัญญาณไปในขั้นตอนเดียวกัน จากนั้นจึงใช้ sigwait() ในจุดที่ต้องการให้เธรดหยุดการทำงานไปเป็นการชั่วคราว (ซึ่งการหยุดการทำงานในลักษณะเช่นนี้จะไม่เสียเวลาการทำงานของซีพียู แตกต่างจากการเขียนข้อความสั่งวนรอบคอยตรวจสอบสถานะ)

โปรแกรมตัวอย่างข้างล่างนี้ จะแสดงไฟกระพริบสลับกันเป็นจำนวนสามรอบแล้วหยุดลง เพื่อรอให้ผู้ใช้กด CTRL-C ที่คีย์บอร์ด ก่อนที่จะกระพริบไฟใหม่อีกหนึ่งชุด และจะทำเช่นนี้ไปจำนวน 5 ครั้ง จึงค่อยหยุดเธรดไฟกระพริบและหยุดโปรแกรมในที่สุด

```
#include <pigpio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

int ledGPIO[4]={4,5,6,12};
void initGPIO();
pthread_t tid;
int count=5;

void *blinkingLED(void *param);

void gpio_stop(int sig){
    printf("User pressing CTRL-C (%d)\n",count);
       pthread_kill(tid,SIGUSR1);
       count--;
}

int main(){
    pthread_attr_t attr;
    int i;

    initGPIO();

    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,blinkingLED,NULL);

    printf("Waiting LED thread to stop (using CTRL-C for 5 times)...\n");

    pthread_join(tid,NULL);
    pthread_attr_destroy(&attr);

    gpioTerminate();
    return 0;
}

void initGPIO(){
    int i;
```

```
    if(gpioInitialise() < 0) exit(1);
    signal(SIGINT,gpio_stop);
    for(i=0;i<4;i++)
        gpioSetMode(ledGPIO[i],PI_OUTPUT);
}

void *blinkingLED(void *param){
    int i,ii,j;
    int pattern[2][4]={{1,0,1,0},{0,1,0,1}};
    sigset_t signal_set;
    int sig;

    sigemptyset(&signal_set);
    sigaddset(&signal_set,SIGUSR1);

    while(count>0){
        for(ii=0;ii<3;ii++){
            for(i=0;i<2;i++){
                for(j=0;j<4;j++)
                    gpioWrite(ledGPIO[j],pattern[i][j]);
                usleep(250000);
            }
        }
        sigwait(&signal_set,&sig);
        printf("Getting signal %d\n",sig);
    }
    for(j=0;j<4;j++)
        gpioWrite(ledGPIO[j],0);
    pthread_exit(0);
}
```

คำเตือน *เนื่องจากเนื้อหาในปฏิบัติการครั้งนี้และต่อๆ ไป เราได้ดึงเอาสัญญาณ SIGINT (ผู้ใช้กดปุ่ม CTRL-C) มาจัดการเอง ดังนั้นหากเกิดความผิดพลาดใดๆ ในโปรแกรม (เช่นการเขียนโปรแกรมแบบหลายเธรด และใช้สัญญาณ CTRL-C มาสั่งการหยุดเธรดต่างๆ แล้วเราลืมสั่งหยุดบางเธรด) เราอาจจะไม่สามารถกดปุ่ม CTRL-C เพื่อหยุดโปรแกรมได้ตามปกติ*

ในกรณีเช่นนี้ ให้เปิด terminal ขึ้นมาใหม่หนึ่งตัว แล้วใช้คำสั่ง

ps -ef เพื่อค้นดูว่าโพรเซสที่เรากำลังสั่งทำงานนั้นมีเลข process ID อะไร จากนั้นให้ใช้คำสั่ง

sudo kill -9 ตามด้วย process ID เพื่อสั่งหยุดการทำงาน