НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

# Лабораторна робота № R04

з курсу

”Мікропроцесорні та програмні засоби автоматизації”

# Робота з потоками та подіями у Raspberry Pi за допомогою бібліотеки WiringPi

Тернопіль 2021

Методичні вказівки до лабораторної роботи № R04 “Робота з потоками та подіями у Raspberry Pi за допомогою бібліотеки WiringPi” з курсу "Мікропроцесорні та програмні засоби автоматизації". Медвідь В.Р., Пісьціо В.П.,, Тернопіль: ТНТУ, 20211 - 7 с.

Для студентів напряму підготовки: 151 "Автоматизація та комп'ютерно-інтегровані технології"

Автори: Медвідь В.Р., Пісьціо В.П..

**Тема роботи**

Робота з потоками та подіями у Raspberry Pi за допомогою бібліотеки WiringPi

**Мета роботи**

Ознайомитись з можливостями та роботою із потоками у Raspberry Pi за допомогою бібліотеки WiringPi.

**Пріорітети, переривання і потоки**

WiringPi надає деякі допоміжні функції, що дозволяють керувати пріоритетом програми (або потоку) та допомагають запускати новий потік всередині програми. Волокна виконуються одночасно з вашою основною програмою і можуть використовуватися для різних цілей.

Функція int piHiPri (int priority); намагається перенести програму (або волокно в багатопотокову програму) на більш високий пріоритет і дає змогу планувати виконання програми у режимі реального часу. Параметр пріоритету повинен бути від 0 (за замовчуванням) до 99 (максимум). Це не збільшить швидшої роботи програми, але додасть їй більше квантів часу, у випадку коли запущені інші програми. Параметр пріоритету працює відносно інших - тому можна зробити один потік із програмним пріоритетом 1 та інший пріоритет 2, і він матиме такий же ефект, як встановлення одного на 10, а іншого на 90 (до тих пір, поки жодна інша програма не працює з підвищеними пріоритетами). Функція повертає значення 0 у випадку успішного встановлення пріоритету та -1 при помилці. Якщо повернуто значення -1, програма може звернутися до глобальної змінної errno, котра містить код помилки. Зауважимо, що лише програми, котрі працюють під керуванням root, можуть змінювати свій пріоритет. Якщо викликати функцію з програми не під root нічого не відбувається.

При використанні нового ядра, з'явилась можливість обробки переривань GPIO, і тепер у програмі можна чекати переривань за зміною стану ліній GPIO. Обробка переривань звільняє процесор, для виконання інших завдань, при очікуванні зміни сигналів на лініях GPIO. Переривання може бути встановлені на спрацювання за на зростаючим, падаючий або обидва фронти сигналу на входах.

Функція int wiringPiISR (int pin, int edgeType, void (* func) (void)); реєструє функцію для отриманих переривань на зазначеному контакті. Параметр edgeType набуває значення INT_EDGE_FALLING, INT_EDGE_RISING, INT_EDGE_BOTH або INT_EDGE_SETUP. Значення параметра INT_EDGE_FALLING вказує на спадаючій фронт, INT_EDGE_RISING -на зростаючій, а INT_EDGE_BOTH викликає переривання при отриманні будь-якої зміни сигналу на вході. Параметр INT_EDGE_SETUP указує, що повторна ініціалізація переривань на контакті не відбудеться - передбачається, що режим лінії вже встановлений раніше.

Номер контакту подається в поточному режимі - рідному wiringPi, BCM_GPIO, фізичному або режимі Sys. Ця функція працюватиме в будь-якому режимі і не потребує привілеїв root для роботи. Функція func буде викликана, коли спрацює переривання. При спрацьовуванні переривання воно видаляється у диспетчері перед викликом функції func. Отже якщо при роботі функції func переривання знову виникне переривання воно не буде запущено до того, як функція func закінчіть свою роботу. Однак диспетчер може відслідковувати лише ще одне переривання, і якщо під час обробки спрацьовує більше одного переривання, вони будуть проігноровані.

Дана функція виконується з високим пріоритетом (якщо програма виконується за допомогою sudo або як root) та виконується одночасно з основною програмою. Вона має повний доступ до всіх глобальних змінних, відкритих файлів тощо.

**Одночасна обробка.**

wiringPi має спрощений інтерфейс до впровадження Linux потоків Posix, а також спрощені механізми доступу до мютексів. За допомогою цих функцій можна створити нові потоки, що працюють одночасно з основною програмою і за допомогою механізмів мютексів безпечно передають змінні між ними.

Функція int piThreadCreate (ім'я); створює потік, який працює паралельно із основним потоком програми, на основі раніше оголошеної за допомогою декларації PI_THREAD функції потоку. Функція може бути корисна у випадку коли необхідно виконувати погграму в кілька

потоків: наприклад фоновий потік проводить слідкування за контактами GPIO, у той час коли основний потік програми здійснює обслуговування введення користувача. Потік може вказувати на подію чи дію, використовуючи глобальні змінні для передачі інформації до основної програми чи інших потоків.

Для оголошення функції потоку використовують наступний шаблон:

```
PI_THREAD (myThread)
{
  .. код тут для одночасного запуску
        основна програма, ймовірно, в
        нескінченному циклі}
```

Після чого його новий потік можна запустити в основній програмі за допомогою коду:

```
x = piThreadCreate (myThread);
if (x! = 0)
  printf ("не стартувало")
```

Це насправді не що інше, як спрощений інтерфейс до механізму потоків Posix, який підтримує Linux.

Функції piLock (int keyNum); piUnlock (int keyNum) дозволяють синхронізувати оновлення змінних між основною програмою і будь-яких потоків, що працюють у програмі. keyNum - це число від 0 до 3 і являє собою "ключ". Коли інший процес спробує заблокувати той самий ключ, він буде затримуватися, поки перший процес не розблокує той ключ.

Ці функції слід використовувати для забезпечення отримання дійсних даних під час обміну даними між основною програмою та потоком - інакше можливо, що поток може прокинутися у процесі копіювання даних та змінити дані - тоді, дані, що отримує основний потік будуть не неповними або недійсними.

**Приклад програми**

Програма роботи з перериваннями

```
/*  * isr.c:
 *   Wait for Interrupt test program - ISR method
 *   How to test:
 *     Use the SoC's pull-up and pull down resistors that are
avalable
 *   on input pins. So compile & run this program (via sudo),
then
 *   in another terminal:
 *        gpio mode 0 up
 *        gpio mode 0 down
 *   at which point it should trigger an interrupt. Toggle the
pin
 *   up/down to generate more interrupts to test. */
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <wiringPi.h>
//Global variable to count interrupts Should be declared
volatile to make sure the compiler doesn't cache it.
static volatile int globalCounter [8] ;
/* myInterrupt: ********************/
void myInterrupt0 (void) { ++globalCounter [0] ; }
void myInterrupt1 (void) { ++globalCounter [1] ; }
void myInterrupt2 (void) { ++globalCounter [2] ; }
void myInterrupt3 (void) { ++globalCounter [3] ; }
void myInterrupt4 (void) { ++globalCounter [4] ; }
void myInterrupt5 (void) { ++globalCounter [5] ; }
```

```
void myInterrupt6 (void) { ++globalCounter [6] ; }
void myInterrupt7 (void) { ++globalCounter [7] ; }
/* main ***************************/
int main (void)
{ int gotOne, pin ;
  int myCounter [8] ;
  for (pin = 0 ; pin < 8 ; ++pin)
    globalCounter [pin] = myCounter [pin] = 0 ;
  wiringPiSetup () ;
  wiringPiISR (0, INT_EDGE_FALLING, &myInterrupt0) ;
  wiringPiISR (1, INT_EDGE_FALLING, &myInterrupt1) ;
  wiringPiISR (2, INT_EDGE_FALLING, &myInterrupt2) ;
  wiringPiISR (3, INT_EDGE_FALLING, &myInterrupt3) ;
  wiringPiISR (4, INT_EDGE_FALLING, &myInterrupt4) ;
  wiringPiISR (5, INT_EDGE_FALLING, &myInterrupt5) ;
  wiringPiISR (6, INT_EDGE_FALLING, &myInterrupt6) ;
  wiringPiISR (7, INT_EDGE_FALLING, &myInterrupt7) ;
  for (;;)
  {
    gotOne = 0 ;
    printf ("Waiting ... ") ; fflush (stdout) ;
    for (;;)
    {
      for (pin = 0 ; pin < 8 ; ++pin)
      {
     if (globalCounter [pin] != myCounter [pin])
     {
      printf (" Int on pin %d: Counter: %5d\n", pin,
globalCounter [pin]) ;
       myCounter [pin] = globalCounter [pin] ;
       ++gotOne ;
     }
      }
      if (gotOne != 0)
     break ;
    }
  }

  return 0 ;
}
```

Програма роботи з потоками

```
/*
 * blink-thread.c:
 */
#include <stdio.h>
#include <wiringPi.h>
// LED Pin - wiringPi pin 0 is BCM_GPIO 17.
#define   LED  0
PI_THREAD (blinky)
{ for (;;)
  {
    digitalWrite (LED, HIGH) ;     // On
    delay (500) ;        // mS
    digitalWrite (LED, LOW) ; // Off
    delay (500) ;
```

```
  }
}
int main (void)
{  printf ("Raspberry Pi blink\n") ;
   wiringPiSetup () ;
   pinMode (LED, OUTPUT) ;
   piThreadCreate (blinky) ;
  for (;;)
  {
    printf ("Hello, world\n") ;
    delay (600) ;
  }
  return 0 ;
}
```

**Завдання**

1 Завантажити та зібрати програму, наведені у прикладі.

2 Модифікувати програму таким чином, щоб підраховувалась різниця імпульсів на входах.

3 Спробувати при запущеній програмі, що працює з апаратним та програминим ШІМ запустити відносно тяжку програму. Визначити чи не змінюється сигнал ШІМ.

4. Створити і запустити власну програму, що створює потік і виконує там, наприклад, мигання світлодіодом.

5 Скласти звіт з лабораторної роботи.

**Контрольні запитання**

1. Для чого можна використати різні потоки у програмі ?
2. Чи є можливість обслуговувати переривання від GPIO у програмі користувача?
3. Як декларувати функцію потоку?
4. Як можна синхронізувати роботу потоків у WiringPi?

**Література**

1. Иго Т. Arduino, датчики и сети для связи устройств: Пер. с англ. -СПб.: БХВ-Петербург, 2016. - 544 с.

2. Петин В.А. Arduino и Raspberry Pi в проектах Internet of Things. -СПб.: БХВ-Петербург, 2016.-464 с.

3. Петин В.А. Микрокомпьютеры Raspberry Pi.Практическое руководство. СПб.: БХВ-Петербург, 2015. -240 с.

4. Петин В.А. Проекты с использованием контроллера Arduino - СПб.: БХВ-Петербург, 2016. - 464 с.

**Зміст**