Politechnika Warszawska

PSIR: Laboratorium 3

Programowanie mikrokontrolerów z użyciem RiotOS – obsługa wątków i przerwań

Kazimierz Kochan 303704
Monika Lewandowska 303707
2022-05-18

Wstęp

Celem laboratorium jest stworzenie oprogramowania uruchamianego na mikrokontrolerze z użyciem systemu operacyjnego czasu rzeczywistego: RiotOS. Laboratorium jest podzielone na cztery części, w wyniku każdej przygotowany został plik main.c z całym kodem aplikacji.

Fragmenty kodu, które utworzyliśmy lub edytowaliśmy zostały obramowane niebieskim podkreśleniem.

Zadanie 1 - tworzenie wątku

Celem pierwszej części jest utworzenie wątku i uruchomienie w nim funkcjonalności polegającej na miganiu zieloną diodą na płytce. Dodatkowo na laboratorium zostaliśmy poproszeni przez prowadzącego o stworzenie analogicznej funkcjonalności dla diody czerwonej w oddzielnym wątku. Dioda czerwona powinna migać z inną częstotliwością niż dioda zielona.

W rezultacie po kompilacji i wgraniu kodu na płytkę zobaczyliśmy migające z różną częstotliwością diody.

Poniżej znajduje się kod aplikacji do zadania pierwszego (w ramce wprowadzone fragmenty kodu):

```
#include <stdio.h>

#include "board.h"       /* board specific definitions */
#include "periph/gpio.h" /* gpio api */
#include "stm32l072xx.h" /* mcu specific definitions */

/* threading includes */
#include "thread.h"
#include "msg.h"
#include "xtimer.h"

#define ENABLE_DEBUG    (1)
#if ENABLE_DEBUG
#include "debug.h"
#endif

/* button manipulation macro */
#define USER_BUTTON       (BTN_B1_PIN)

/* led manipulation macros */
#define RED_LED_OFF       (LED3_OFF)
#define RED_LED_ON        (LED3_ON)
#define RED_LED_TOGGLE    (LED3_TOGGLE)
#define BLUE_LED_OFF      (LED2_OFF)
#define BLUE_LED_ON       (LED2_ON)
#define BLUE_LED_TOGGLE   (LED2_TOGGLE)
#define GREEN_LED_OFF     (LED1_OFF)
#define GREEN_LED_ON      (LED1_ON)
#define GREEN_LED_TOGGLE  (LED1_TOGGLE)
```

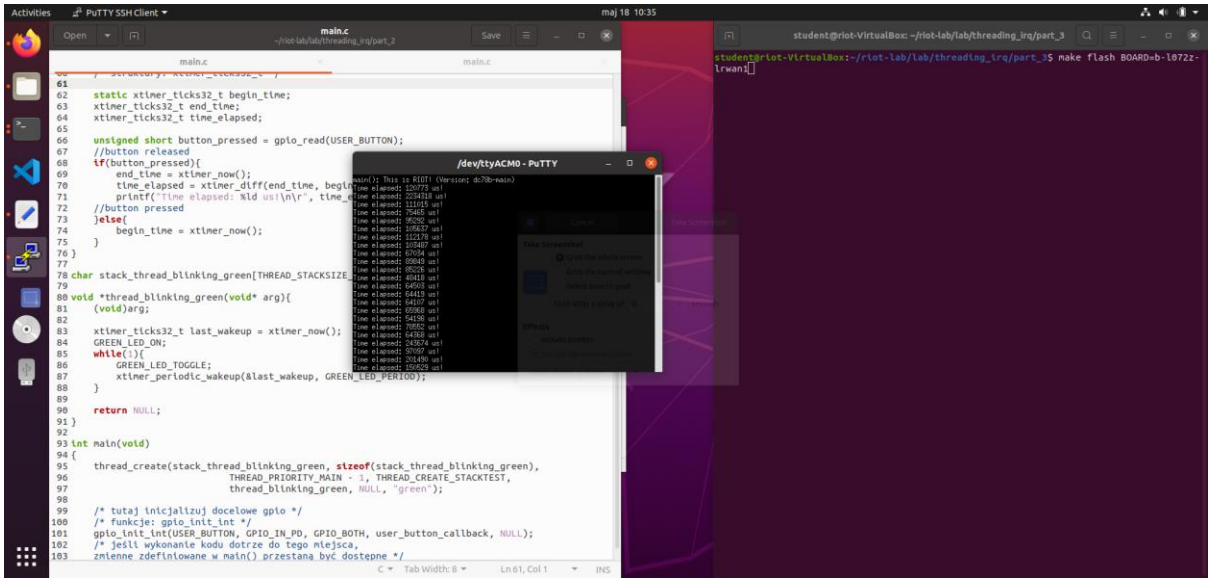
```
/* leds period times (can be changed) */
#define RED_LED_PERIOD         (500000)
#define GREEN_LED_PERIOD     (250000)
#define BLUE_LED_PERIOD     (250000)

char stack_thread_blinking_green[THREAD_STACKSIZE_MAIN];
char stack_thread_blinking_red[THREAD_STACKSIZE_MAIN];

void *thread_blinking_green(void* arg){
    (void)arg;
```

```
    /* tutaj napisz kod, który będzie powodował miganie zieloną diodą na płytce */
    /* funkcje: xtimer_now, xtimer_periodic_wakeup */
    /* struktury: xtimer_ticks32_t */
    xtimer_ticks32_t last_wakeup;
    while(1){
        last_wakeup = xtimer_now();
        GREEN_LED_TOGGLE;
        xtimer_periodic_wakeup(&last_wakeup, GREEN_LED_PERIOD);
    }

    return NULL;
}

void *thread_blinking_red(void* arg){
    (void)arg;
    xtimer_ticks32_t last_wakeup;
    while(1){
        last_wakeup = xtimer_now();
        RED_LED_TOGGLE;
        xtimer_periodic_wakeup(&last_wakeup, RED_LED_PERIOD);
    }

    return NULL;
}

int main(void)
{
    /* tutaj zdefiniuj wątek dla zielonej diody */
    /* funkcje: thread_create */
    thread_create(stack_thread_blinking_green, sizeof(stack_thread_blinking_green),
THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST, thread_blinking_green, NULL,
"thread_green");

    thread_create(stack_thread_blinking_red, sizeof(stack_thread_blinking_red),
THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST, thread_blinking_red, NULL,
"thread_red");

    /* jeśli wykonanie kodu dotrze do tego miejsca,
    zmienne zdefiniowane w main() przestaną być dostępne */
    return 0;
}
```

Zadanie 2 – obsługa przerwań

Celem drugiej części jest napisanie kodu obsługującego przerwanie wywoływane przez naciśnięcie przycisku na płytce mikrokontrolera. W ciele `main` zainicjowaliśmy gpio podając odpowiednie argumenty. Napisaliśmy metodę `user_button_callback` która odczytuje stan GPIO przycisku podczas naciśnięcia i opuszczenia, oraz po opuszczeniu drukuje obliczony czas pomiędzy tymi zdarzeniami.

W rezultacie po wciśnięciu i puszczeniu przycisku na terminalu drukowany jest czas naciśnięcia mierzony w mikrosekundach.



Rysunek 1 Zrzut ekranu prezentujący wynik działania drugiej części laboratorium

Poniżej znajduje się kod aplikacji do zadania drugiego (w ramce wprowadzone fragmenty kodu):

```
#include <stdio.h>

#include "board.h"          /* board specific definitions */
#include "periph/gpio.h" /* gpio api */
#include "stm32l072xx.h" /* mcu specific definitions */

/* threading includes */
#include "thread.h"
#include "msg.h"
#include "xtimer.h"

#define ENABLE_DEBUG    (1)
#if ENABLE_DEBUG
#include "debug.h"
#endif

/* button manipulation macro */
#define USER_BUTTON         (BTN_B1_PIN)

/* led manipulation macros */
#define RED_LED_OFF         (LED3_OFF)
#define RED_LED_ON          (LED3_ON)
#define RED_LED_TOGGLE      (LED3_TOGGLE)
#define BLUE_LED_OFF        (LED2_OFF)
#define BLUE_LED_ON         (LED2_ON)
#define BLUE_LED_TOGGLE     (LED2_TOGGLE)
#define GREEN_LED_OFF       (LED1_OFF)
#define GREEN_LED_ON        (LED1_ON)
```

```
#define GREEN_LED_TOGGLE  (LED1_TOGGLE)

/* leds period times (can be changed) */
#define RED_LED_PERIOD         (250000)
#define GREEN_LED_PERIOD     (250000)
#define BLUE_LED_PERIOD     (250000)
```

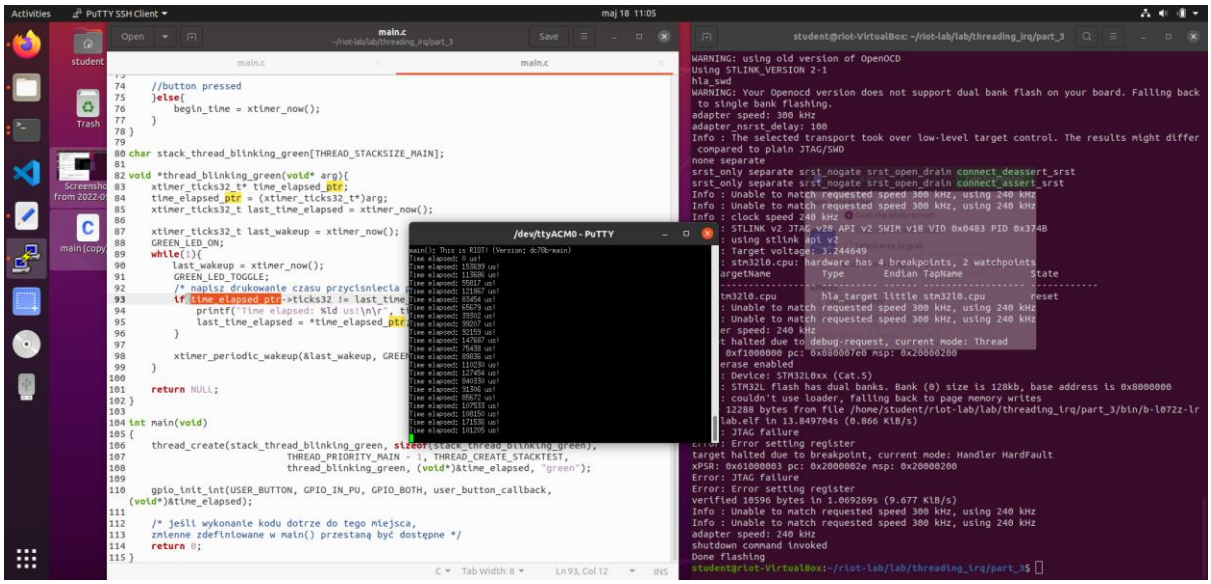
```
static void user_button_callback(void *arg){
    (void)arg;
    /* tutaj napisz obsługę przerwania */
    /* funkcje: xtimer_now, xtimer_diff, DEBUG lub printf */
    /* struktury: xtimer_ticks32_t */

    static xtimer_ticks32_t begin_time;
    xtimer_ticks32_t end_time;
    xtimer_ticks32_t time_elapsed;

    unsigned short button_pressed = gpio_read(USER_BUTTON);
    //button released
    if(button_pressed){
        end_time = xtimer_now();
        time_elapsed = xtimer_diff(end_time, begin_time);
        printf("Time elapsed: %ld us!\n\r", time_elapsed.ticks32);
    //button pressed
    }else{
        begin_time = xtimer_now();
    }
}

char stack_thread_blinking_green[THREAD_STACKSIZE_MAIN];

void *thread_blinking_green(void* arg){
    (void)arg;

    xtimer_ticks32_t last_wakeup = xtimer_now();
    GREEN_LED_ON;
    while(1){
        GREEN_LED_TOGGLE;
        xtimer_periodic_wakeup(&last_wakeup, GREEN_LED_PERIOD);
    }

    return NULL;
}
```

```
int main(void)
{
    thread_create(stack_thread_blinking_green, sizeof(stack_thread_blinking_green),
                            THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST,
                            thread_blinking_green, NULL, "green");

    /* tutaj inicjalizuj docelowe gpio */
    /* funkcje: gpio_init_int */
    gpio_init_int(USER_BUTTON, GPIO_IN_PD, GPIO_BOTH, user_button_callback, NULL);

    /* jeśli wykonanie kodu dotrze do tego miejsca,
    zmienne zdefiniowane w main() przestaną być dostępne */
    return 0;
}
```

Zadanie 3 - wspólne zasoby

Celem trzeciej części laboratorium jest taka modyfikacja programu, żeby to wątek obsługujący migającą diodę drukował czas przyciśnięcia przycisku. Realizowane to będzie poprzez dostęp do wspólnych zasobów. Utworzyliśmy zmienną globalną przechowującą wartość ostatniego czasu pomiędzy naciśnięciem, a opuszczeniem przycisku. Pointer do tej zmiennej przekazywany jest do metod `thread_blinking_green` oraz `user_button_callback`. Druga z tych metod oblicza czas oraz zapisuje do zmiennej globalnej. W metodzie odpowiedzialnej za miganie zieloną diodą pobrany czas porównywany jest z ostatnią wartością, aby nie drukować ciągle niepotrzebnych informacji. Jeżeli czas przyciśnięcia przycisku różni się od ostatniego, jest on wypisywany.

Wynik działania zaprezentowany na poniższym zrzucie ekranu.



Rysunek 2 Zrzut ekranu prezentujący wynik działania trzeciej części laboratorium

Poniżej znajduje się kod aplikacji do zadania trzeciego (w ramce wprowadzone fragmenty kodu):

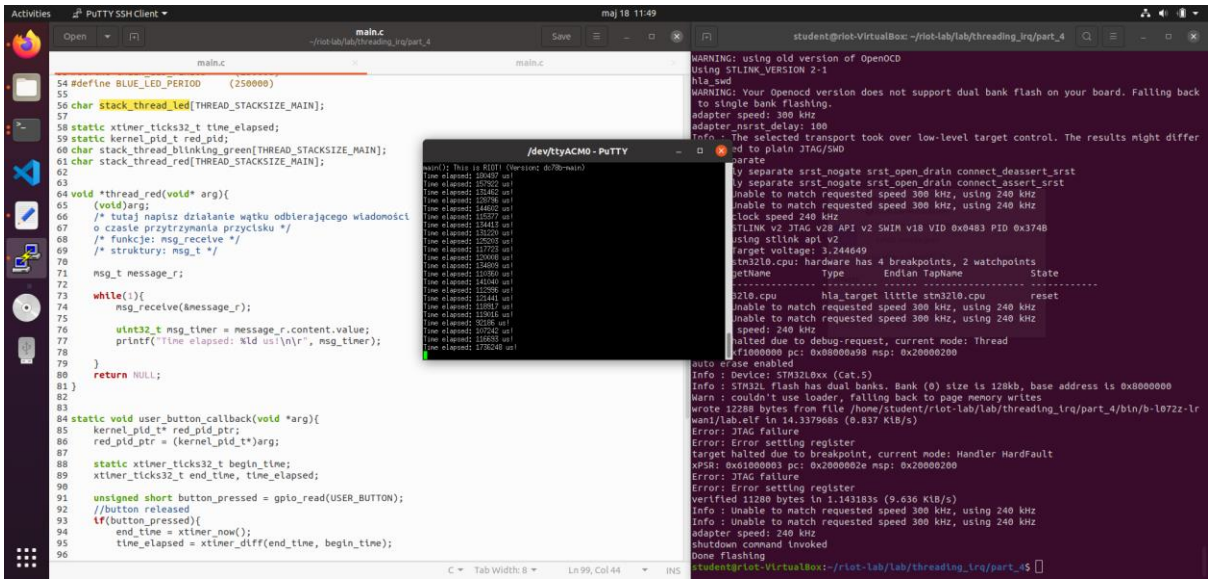
```
#include <stdio.h>

#include "board.h"        /* board specific definitions */
#include "periph/gpio.h" /* gpio api */
#include "stm32l072xx.h" /* mcu specific definitions */

/* threading includes */
#include "thread.h"
#include "msg.h"
#include "xtimer.h"

#define ENABLE_DEBUG    (1)
#if ENABLE_DEBUG
#include "debug.h"
#endif

/* button manipulation macro */
#define USER_BUTTON         (BTN_B1_PIN)

/* led manipulation macros */
#define RED_LED_OFF         (LED3_OFF)
#define RED_LED_ON          (LED3_ON)
#define RED_LED_TOGGLE      (LED3_TOGGLE)
#define BLUE_LED_OFF        (LED2_OFF)
#define BLUE_LED_ON         (LED2_ON)
```

```
#define BLUE_LED_TOGGLE   (LED2_TOGGLE)
#define GREEN_LED_OFF     (LED1_OFF)
#define GREEN_LED_ON      (LED1_ON)
#define GREEN_LED_TOGGLE  (LED1_TOGGLE)

/* leds period times (can be changed) */
#define RED_LED_PERIOD         (250000)
#define GREEN_LED_PERIOD     (250000)
#define BLUE_LED_PERIOD     (250000)

/* tutaj zadeklaruj zmienną globalną, ktora bedzie przechowywać czas
    przez który ostatnio przycisk był wciśnięty */
static xtimer_ticks32_t time_elapsed;

static void user_button_callback(void *arg){
    xtimer_ticks32_t* time_elapsed_ptr;
    time_elapsed_ptr = (xtimer_ticks32_t*)arg;

    static xtimer_ticks32_t begin_time;
    xtimer_ticks32_t end_time;


    unsigned short button_pressed = gpio_read(USER_BUTTON);
    //button released
    if(button_pressed){
        end_time = xtimer_now();
        *time_elapsed_ptr = xtimer_diff(end_time, begin_time);

    //button pressed
    }else{
        begin_time = xtimer_now();
    }
}

char stack_thread_blinking_green[THREAD_STACKSIZE_MAIN];

void *thread_blinking_green(void* arg){
    xtimer_ticks32_t* time_elapsed_ptr;
    time_elapsed_ptr = (xtimer_ticks32_t*)arg;
    xtimer_ticks32_t last_time_elapsed = xtimer_now();

    xtimer_ticks32_t last_wakeup = xtimer_now();
    GREEN_LED_ON;
    while(1){
       last_wakeup = xtimer_now();
        GREEN_LED_TOGGLE;
        /* napisz drukowanie czasu przycisniecia przycisku w tym watku */
        if(time_elapsed_ptr->ticks32 != last_time_elapsed.ticks32){
            printf("Time elapsed: %ld us!\n\r", time_elapsed_ptr->ticks32);
            last_time_elapsed = *time_elapsed_ptr;
        }

        xtimer_periodic_wakeup(&last_wakeup, GREEN_LED_PERIOD);
    }

    return NULL;
}

int main(void)
{
    thread_create(stack_thread_blinking_green, sizeof(stack_thread_blinking_green),
                            THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST,
                            thread_blinking_green, (void*)&time_elapsed, "green");

    gpio_init_int(USER_BUTTON, GPIO_IN_PU, GPIO_BOTH, user_button_callback,
(void*)&time_elapsed);

    return 0;}
```

Zadanie 4 - przesyłanie wiadomości między wątkami

Celem tej części laboratorium jest stworzenie systemu, w którym czas przyciśnięcia przycisku będzie wyświetlany przez wątek, do którego wartość tego czasu zostanie przesłana poprzez system wiadomości systemu RIOT. Taką funkcjonalność uzyskaliśmy przekazując metodzie `user_button_callback` pointer na pid czerwonego wątku. Dzięki temu metoda ta może wysłać do drugiego wątku wiadomość msg_t zawierającą pole ticks32 struktury xtimer_ticks32_t przechowujące mierzony czas. Czerwony wątek taką wiadomość odbiera i następnie odmierzony czas zostaje drukowany.

Wynik działania zaprezentowany na poniższym zrzucie ekranu.



Rysunek 3 Zrzut ekranu prezentujący wynik działania czwartej części laboratorium

Poniżej znajduje się kod aplikacji do zadania czwartego (w ramce wprowadzone fragmenty kodu):

```
#include <stdio.h>

#include "board.h"        /* board specific definitions */
#include "periph/gpio.h" /* gpio api */
#include "stm32l072xx.h" /* mcu specific definitions */

/* threading includes */
#include "thread.h"
#include "msg.h"
#include "xtimer.h"

#define ENABLE_DEBUG    (1)
#if ENABLE_DEBUG
#include "debug.h"
#endif

/* button manipulation macro */
#define USER_BUTTON         (BTN_B1_PIN)

/* led manipulation macros */
#define RED_LED_OFF         (LED3_OFF)
#define RED_LED_ON          (LED3_ON)
#define RED_LED_TOGGLE      (LED3_TOGGLE)
#define BLUE_LED_OFF        (LED2_OFF)
#define BLUE_LED_ON         (LED2_ON)
#define BLUE_LED_TOGGLE     (LED2_TOGGLE)
#define GREEN_LED_OFF       (LED1_OFF)
```

```
#define GREEN_LED_ON      (LED1_ON)
#define GREEN_LED_TOGGLE  (LED1_TOGGLE)

/* leds period times (can be changed) */
#define RED_LED_PERIOD         (250000)
#define GREEN_LED_PERIOD     (250000)
#define BLUE_LED_PERIOD     (250000)

char stack_thread_led[THREAD_STACKSIZE_MAIN];
```

```
static xtimer_ticks32_t time_elapsed;
static kernel_pid_t red_pid;
char stack_thread_blinking_green[THREAD_STACKSIZE_MAIN];
char stack_thread_red[THREAD_STACKSIZE_MAIN];
```

```
void *thread_red(void* arg){
    (void)arg;
    /* tutaj napisz działanie wątku odbierającego wiadomości
    o czasie przytrzymania przycisku */
    /* funkcje: msg_receive */
    /* struktury: msg_t */

    msg_t message_r;

    while(1){
        msg_receive(&message_r);

        uint32_t msg_timer = message_r.content.value;
        printf("Time elapsed: %ld us!\n\r", msg_timer);

    }
    return NULL;
}
```

```
static void user_button_callback(void *arg){
    kernel_pid_t* red_pid_ptr;
    red_pid_ptr = (kernel_pid_t*)arg;

    static xtimer_ticks32_t begin_time;
    xtimer_ticks32_t end_time, time_elapsed;

    unsigned short button_pressed = gpio_read(USER_BUTTON);
    //button released
    if(button_pressed){
        end_time = xtimer_now();
        time_elapsed = xtimer_diff(end_time, begin_time);

        msg_t message_s;
        message_s.content.value = (uint32_t) time_elapsed.ticks32;
        msg_send(&message_s, *red_pid_ptr);
    //button pressed
    }else{
        begin_time = xtimer_now();
    }
}

void *thread_blinking_green(void* arg){
    (void)arg;
    xtimer_ticks32_t last_wakeup = xtimer_now();
    GREEN_LED_ON;
    while(1){
       last_wakeup = xtimer_now();
        GREEN_LED_TOGGLE;
        xtimer_periodic_wakeup(&last_wakeup, GREEN_LED_PERIOD);
    }
    return NULL;
```

```
}


int main(void)
{
    thread_create(stack_thread_blinking_green, sizeof(stack_thread_blinking_green),
                            THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST,
                            thread_blinking_green, NULL, "green");

    /* zmodyfikuj callback w przerwaniu generowanym przez przycisk */
    gpio_init_int(USER_BUTTON, GPIO_IN_PU, GPIO_BOTH, user_button_callback,
(void*)&red_pid);

    /* tutaj napisz uruchomienie czerwonego wątku, przekaż pid tego wątku do
user_button_callback
    poprzez callback argument lub zmienną globalną*/
    red_pid = thread_create(stack_thread_red, sizeof(stack_thread_red),
                            THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST,
                            thread_red, (void*)&time_elapsed, "green");
    /* jeśli wykonanie kodu dotrze do tego miejsca,
    zmienne zdefiniowane w main() przestaną być dostępne */
    return 0;
}
```

Podsumowanie

Wszystkie zadania udało się zrealizować zgodnie z poleceniem. Laboratorium pozwoliło nam zapoznać się z programowaniem programowania mikrokontrolerów z użyciem systemu operacyjnego czasu rzeczywistego oraz przećwiczyć wykorzystywanie w programach mechanizmów wielowątkowości oraz przerwań systemu.