Opdrachten week 4 lab 1 – RTOS

Op het moment dat de applicaties complexer worden, zullen meerdere taken (schijnbaar) parallel uitgevoerd moeten worden. Om dit makkelijker te maken kan je gebruik maken van een Operating System (OS). Een OS kennen jullie vaak in de vorm van Windows, Linux, MacOS of Android. Dit zijn hele grote en zware OSen die heel veel kunnen, maar daardoor niet geschikt zijn om op een microcontroller te draaien. Toch willen we vaak wel gebruik maken van een OS. Zeker op de wat geavanceerdere microcontrollers zoals de CC3220S.

Hiervoor zijn embedded OSen gemaakt. Veel daarvan zijn zogeheten Real-Time Operating Systems (RTOS), het doel van deze RTOSen is dat het taken in een voorspelbare tijd kan afhandelen. In dit vak wordt het RTOS van TI gebruikt (TI-RTOS) waarbij we gebruik maken van de POSIX interface. TI-RTOS is speciaal ontwikkeld voor microcontrollers van Texas Instruments en wordt ondersteund voor heel veel van hun processoren. Het voordeel van dit RTOS is dat het gratis is (voor gebruik op TI processoren) en netjes is geïntegreerd in Code Composer Studio (CCS). Een andere optie is het veelgebruikte FreeRTOS, dat Texas Instruments ook ondersteund. Er bestaan nog vele andere real-time operating systems[1].

Je gaat in dit lab leren:

- wat een Operating System (OS) voor je kan doen;
- wat het verschil is tussen een General Purpose OS (GPOS) en een Real-Time OS (RTOS);
- waarom het handig is om taken concurrent uit te voeren;
- hoe je taken met behulp van POSIX threads concurrent kan uitvoeren;
- Wat concurrent uitvoeren voor gevolgen heeft

4.1.1 Pthreads is een door POSIX[2] gestandaardiseerde methode om nieuwe taken aan te maken. Dit werkt in bijna alle moderne OSen[3]. Met pthreads kun je dynamisch (runtime[4]) nieuwe threads of taken aanmaken.

Een POSIX thread bestaat uit een variabele (handle of handvat) om de taak aan te geven. Dit handvat is van het type `pthread_t`. Dit handvat kan worden doorgegeven

[1] Zie: https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems.

[2] Portable Operating System Interface, zie https://en.wikipedia.org/wiki/POSIX.

[3] Niet alleen in RTOSen maar ook in General Purpose OSen zoals Windows, Linux en MacOS.

[4] runtime betekent tijdens het uitvoeren van de code, in tegenstelling tot tijdens het compileren.

aan pthread_create() om een functie te koppelen aan de thread en de thread te starten. De taak zelf is gewoon een C functie.

```c
// Voor sleep en printf
#include <unistd.h>
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>

/* POSIX Header files */
#include <pthread.h>

/* RTOS header files */
#include <ti/sysbios/BIOS.h>

/* Driver Header files */
#include <ti/drivers/Board.h>
#include <ti/drivers/GPIO.h>
#include "ti_drivers_config.h"

void *mijnTaakFunctie(void *arg) {
    while (1) {
        //Doe iets
        puts("Hallo!\n"); //put string (domme versie van printf)
        sleep(1); //slaap 1s
    }
}

int main() {
    pthread_t mijnTaakHandle;

    Board_init(); //TI Drivers
    pthread_create(&mijnTaakHandle, NULL, &mijnTaakFunctie, NULL);
    BIOS_start(); //TI-RTOS

    return 0;
}
```

Listing 1: Basisprogramma voor het gebruik pthread: opdr41.c

Board_init() is deel van de TI drivers. De functie BIOS_start() start het RTOS en moet dus altijd aanwezig zijn!

A Kies Project > Import CCS Projects... en klik op Browse. Navigeer naar het directory: `C:\ti\simplelink_cc32xx_sdk_6_10_00_05\examples\rtos\CC3220S_LAUNCHXL\drivers\empty\tirtos\ccs`. Klik op Finish. Verwijder het bestand `empty.c` en vervang de inhoud van `main_tirtos.c` met listing 1. Zorg ervoor dat de Debug configuratie als actief staat ingesteld. Wanneer je de code uitvoert zou je elke seconde `Hallo!` moeten zien verschijnen.

B Maak een nieuwe functie aan, waarin oneindig de rode led wordt geschakeld met een periodetijd van 2 seconden. Maak gebruik van `pthread_create()` om de nieuwe functie te koppelen aan een taak.

C Debug het project en kijk of de led inderdaad knippert en nog steeds `Hallo!` elke seconde wordt afgedrukt.

4.1.2 Natuurlijk wordt een RTOS interessant met meerdere taken. We gaan ook een derde taak aanmaken die de groene led elke 3 seconden schakelt. Schrijf eerst de functie die dit schakelen kan doen en maak een nieuwe `pthread_t` variabele aan voor de nieuwe taak. Maak wederom gebruik van `pthread_create()` om de taak uit te laten voeren. Test nu of beide leds knipperen.

4.1.3 Beide leds knipperen terwijl beide ledfuncties een `while(1)` loop toepassen. Hoe kan dit? Op het moment dat een taak gaat slapen met behulp van de `sleep()` functie, vertelt de taak tegen het OS dat andere taken aan de beurt mogen komen.

Test wat er gebeurt als je de `sleep()` functie in de rode led taak vervangt met de `busySleep()` functie van listing 2. Kun je verklaren wat er gebeurt met de leds?

```
void busySleep(uint8_t n)
{   //8e6 is ongeveer een seconde
    volatile uint32_t counter = 8e6;
    int i;
    for(i=0; i<n; i++){
        while(counter--);
        counter = 8e6;
    }
}
```

Listing 2: Een actieve wacht functie: busysleep.c

De tweede parameter van de `pthread_create()` functie wordt gebruikt om de taak in te stellen. Denk aan instellingen als

- Hoeveel geheugen mag de taak gebruiken? (de *stack size*)
- Hoe belangrijk is de taak? (de prioriteit)

Dit instellen wordt gedaan via een `pthread_attr_t` variabele. Dit type is een struct waarin alle verschillende instellingen worden bijgehouden. Om de variabele te initializeren met standaard instellingen kun je de `pthread_attr_init()` functie toepassen.

```
pthread_attr_t instellingen;
pthread_attr_init(&instellingen); //initialiseren
```

Om de standaard instelling van het beschikbare geheugen aan te passen kun je de `pthread_attr_setstacksize()` functie gebruiken. Dit is dus niet noodzakelijk.

```
pthread_attr_setstacksize(&instellingen, 1024); //1kb voor de taak
```

Om de standaard prioriteit aan te passen kun je de `pthread_attr_setschedparam()` functie toepassen.

```
struct sched_param prioriteit ;
prioriteit.sched_priority = 1;
pthread_attr_setschedparam(&instellingen, &prioriteit);
```

Hierbij wordt een extra type `sched_param` toegepast waarin nòg meer instellingen te vinden zijn. Uiteindelijk kun je deze instellingen koppelen aan de taak doormiddel van call-by-reference.

```
pthread_create(&mijnHandle, &instellingen, &mijnFunctie, NULL);
```

4.1.4 In opdracht 4.1.3 ben je erachter gekomen dat als een taak niet gaat slapen, de andere taken niet meer aan de beurt komen. Om te voorkomen dat belangrijke processen niet vastlopen kunnen prioriteiten worden gebruikt. Elke taak heeft standaard een prioriteit van 1. Geef de groene taak nu een hogere prioriteit dan de rode taak. Leg uit wat er gebeurt met de leds.

Het vierde parameter van de `pthread_create()` functie wordt gebruikt om parameters door te geven. Het type van de parameter is een `void *` (void pointer). Dit betekent dat je een pointer naar elk willekeurig data type kunt doorgeven, zolang aan de ontvangende kant het type maar bekend is. Een voorbeeld zonder OS:

```
typedef struct{
    char *naam;
    char waarde;
}mijnType;

void functie(void *argumenten)
{
    mijnType args = *(mijnType*)argumenten
    printf("De functie zegt \%s en heeft de waarde \%d ↩
 ↪ ontvangen\n", args.naam, args.waarde);
}

int main()
{
    mijnType argsVoorFunctie = {"Hallo", 10};
    functie(&argsVoorFunctie);
}
```

4.1.5 Maak een nieuwe taak aan waaraan je zowel de slaaptijd als de led om te schakelen door kunt geven met behulp van een **void** * argument. Verzin zelf een passende **struct** om hiervoor te gebruiken. Vervang nu de functies van de rode en groene taken met de nieuwe functie, waarbij dus de parameters aangeven om welke led en frequentie het gaat. Het is dus niet erg om twee threads te maken die aan dezelfde functie zitten gekoppeld.

4.1.6 In listing 3 zijn twee taken geven die een globale variabele ophogen. Vervang alle taken uit je project met deze twee. Geef `Tel0p2()` een hogere prioriteit dan `Tel0p()`. Wat verwacht je dat de waarde van counter is wanneer het wordt afgedrukt? Voer het project een aantal keer uit. Wat zijn de uitkomsten? Wat zou het probleem zijn?

Volgende les wordt gekeken hoe gecommuniceert kan worden tussen taken zonder problemen.

```c
volatile int counter = 0;

void *TelOp(void *arg) {
    int i;
    for (i = 0; i < 1e6; i++) {
        counter++;
    }
}

void *TelOp2(void *arg) {
    int i;
    usleep(1000);
    for (i = 0; i < 1e6; i++)
    {
        counter++;
    }

    sleep(1);
    printf("counter: %d\n", counter);
}
```

Listing 3: Twee taken: conflict.c