Opdrachten week 4 lab 2 – Pthreads message queue

In de vorige les hebben we een begin gemaakt met het gebruik van een RTOS. Het voordeel van het gebruik van een RTOS is dat je verschillende stukken code (taken) tegelijk kunt laten draaien op dezelfde microcontroller. Je hebt geleerd om deze taken runtime aan te maken. Dit lab ga je leren hoe je (veilig) kunt communiceren tussen taken.

Je zult deze les leren:

- Hoe pthreads met elkaar kunnen communiceren met behulp van een *message queue* mqueue.

4.2.1 In deze opdrachten gaan we nog niet aan de slag met internet, maar beginnen we simpel met UART-communicatie. Importeer het project `uart2echo_CC3220S_LAUNCHXL_tirtos_ccs` uit de SDK map[1]. Maak de Debug configuratie Active.

Open het bestand `main_tirtos.c`. In dit bestand wordt één pthread aangemaakt en wordt het OS gestart. We leggen de nieuwe aspecten van deze code eerste even aan je uit:

```
/* Set priority, detach state, and stack size attributes */
priParam.sched_priority = 1;
retc = pthread_attr_setschedparam(&attrs, &priParam);
```

In de eerste van de bovenstaande regels wordt in de variabele `priParam` de prioriteit van de aan te maken thread ingesteld. De laagste prioriteit in TI RTOS is 1 en de hoogste prioriteit is configureerbaar (en staat standaard op 15). De returnwaarde van de functieaanroep `pthread_attr_setschedparam` wordt opgeslagen in de variabele `retc`. Als tijdens het uitvoeren van deze functie een fout is opgetreden, wordt een foutcode teruggegeven. De functie geeft `0` terug als er geen fout is opgetreden.

```
retc |= pthread_attr_setdetachstate(&attrs, ↩
 ↪ PTHREAD_CREATE_DETACHED);
```

In de bovenstaande code wordt de thread op 'detached' gezet. Dit is voor de meeste threads die jullie maken de juiste instelling. Detached betekent namelijk dat de thread op zich zelf staat en onafhankelijk is van de andere threads. Ook deze functie geeft een foutcode terug als er een fout is opgetreden en `0` als er geen fout is opgetreden. Door

[1] `C:\ti\simplelink_cc32xx_sdk_6_10_00_05\examples\rtos\CC3220S_LAUNCHXL\drivers\uart2echo\tirtos\ccs`

deze returnwaarde te OR-en met de waarde van de variabele `retc` houden we bij of er een fout is opgetreden bij één van de functies die is aangeroepen. De OR-operator geeft alleen `0` als alle functieaanroepen correct zijn uitgevoerd en alle returnwaarden `0` zijn.

Vervolgens wordt de stacksize van de taak gespecificeerd:

```
retc |= pthread_attr_setstacksize(&attrs, THREADSTACKSIZE);
```

De returnwaarde van de functie wordt weer ge-OR-ed met de variabele `retc`.

Vervolgens wordt de waarde van `retc` gecontroleerd.

```
if (retc != 0) {
    /* failed to set attributes */
    while (1) {}
}
```

Als `retc` ongelijk aan `0` is, dan is bij één van de functieaanroepen iets misgegaan. In dat geval wordt een oneindige **`while`**-loop uitgevoerd en wordt het programma niet vervolgd.

Vervolgens wordt de thread daadwerkelijk aangemaakt:

```
retc = pthread_create(&thread, &attrs, mainThread, NULL);
if (retc != 0) {
    /* pthread_create() failed */
    while (1) {}
}
```

Tot slot wordt het OS gestart:

```
BIOS_start();
```

De functie `mainThread` die door de pthread wordt uitgevoerd kun je vinden in de file `uartecho.c`. In deze thread wordt onder andere de UART geïnitieerd, geconfigureerd en geopend. Daarna wordt de string `"Echoing characters:\r\n"` via de UART verstuurd. Tot slot wordt in een oneindige lus elk via de UART ontvangen karakter weer teruggestuurd via de UART.

Run en test dit programma en zorg ervoor dat je de code begrijpt.

In de voorbeelden van TI wordt een 'oude' programmeerstijl gebruikt. Alle variabelen worden aan het begin van een functie gedefinieerd. In C89 was dat verplicht maar sinds C99 mag je variabelen definiëren als je ze nodig hebt.

In listing 1 is een meer moderne implementatie van de functie `mainThread` gegeven. Om deze code te kunnen compileren moet je wel de *project properties* aanpassen omdat Code Composer Studio standaard de (verouderde) C89-standaard gebruikt.[2] Open het menu Project > Properties en kies vervolgen Build > ARM Compiler > Advanced Options > Language Options. Kies vervolgens, in de dropdown box *C Dialect*, voor *Compile program in C11 mode (–c11)*.

Je ziet dat de variabelen pas gedefinieerd worden wanneer ze nodig zijn, op regel 6, 10, 20, 21, 26, 27, 29, 37 en 39. Ook zijn enkele regels met overbodig commentaar verwijderd. Tot slot is de aanroep naar `GPIO_setConfig` verwijderd. De GPIO-driver is namelijk al geconfigureerd met behulp van de system configuration tool.

```
1  void *mainThread(void *arg0)
2  {
3      GPIO_init();
4
5      /* Create a UART where the default read and write mode is ↩
   ↪ BLOCKING */
6      UART2_Params uartParams;
7      UART2_Params_init(&uartParams);
8      uartParams.baudRate = 115200;
9
10     UART2_Handle uart = UART2_open(CONFIG_UART2_0, &uartParams);
11
12     if (uart == NULL) {
13         /* UART2_open() failed */
14         while (1);
15     }
16
17     /* Turn on user LED to indicate successful initialization */
18     GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
19
20     const char echoPrompt[] = "Echoing characters:\r\n";
21     size_t bytesWritten;
22     UART2_write(uart, echoPrompt, sizeof(echoPrompt), &bytesWritten);
```

[2] Er zijn verschillende versies van de C-standaard C89, C99, C11 en C18. De laatste versie C18 bevat geen nieuwe features en wordt niet ondersteund door de TI C-compiler. We adviseren je om C11 te gebruiken.

```
23
24      /* Loop forever echoing */
25      while (1) {
26          char input;
27          size_t bytesRead = 0;
28          while (bytesRead == 0) {
29              uint32_t status = UART2_read(uart, &input, 1, ↩
    ↪ &bytesRead);
30
31              if (status != UART2_STATUS_SUCCESS) {
32                  /* UART2_read() failed */
33                   while (1);
34              }
35          }
36
37          size_t bytesWritten = 0;
38          while (bytesWritten == 0) {
39              uint32_t status = UART2_write(uart, &input, 1, ↩
    ↪ &bytesWritten);
40
41              if (status != UART2_STATUS_SUCCESS) {
42                  /* UART2_write() failed */
43                  while (1);
44              }
45          }
46      }
47  }
```

Listing 1: Een modernere implementatie van `mainThread`, zie `uartecho.c`

4.2.2 Pas, als oefening, het programma nu zodanig aan dat de **`while`**`(1)`-loop aan het einde van de functie `mainThread` vervangen wordt door het aanmaken van twee pthreads. Het is de bedoeling dat je in de ene thread de UART-data inleest en in een andere thread de echo terugstuurt. Voor de communicatie tussen de twee threads moet je gebruik maken van een POSIX *message queue* (mqueue). De berichten bevatten **`char`**s.

Voer achtereenvolgens de volgende stappen uit:

- Zorg ervoor dat **`#include`** `<mqueue.h>` aan het bestand `uartecho.c` wordt toegevoegd.

- Maak in de functie `mainThread` een variabele genaamd `postbus` aan van het type `mqd_t`. Deze variabele wordt de koppeling naar de aan te maken mqueue. Later krijgt de variabele `postbus` een waarde.
- Om de mqueue aan te maken, moet eerst wat dingen worden ingesteld. Dit instellen gaat door middel van een **`struct`**. Maak een variabele genaamd `postbus_attr` van het structure type `mq_attr`. Deze struct is al gedefinieerd in de include file `<mqueue.h>` heeft vier velden zoals weergeven in listing 2.

```
typedef struct {
    long mq_flags;
    long mq_maxmsg;  /* Maximum number of messages on queue. */
    long mq_msgsize; /* Maximum message size. */
    long mq_curmsgs; /* Number of messages currently queued. */
} mq_attr;
```

Listing 2: De definitie van `mq_attr`.

- Geef daarna de velden `mq_maxmsg` en `mq_msgsize` de gewenste waarde. Het veld `mq_flags` moet de waarde `0` krijgen.
- Initialiseer de mqueue door `mq_open()` aan te roepen. Deze functie heeft vier parameters. In volgorde:
  - Een naam van de mqueue als string,
  - `O_RDWR | O_CREAT` om aan te geven dat je wilt lezen en schrijven en dat een nieuwe mqueue aangemaakt moet worden,
  - `0666` met dit octale getal stel je de toegangsrechten in voor de verschillende gebruikers van het systeem[3] en
  - het adres van een variabele van het type `mq_attr` waarin de attributen van aan te maken mqueue worden doorgegeven.

[3] Bij TI-RTOS is het niet mogelijk om verschillende gebruikers en groepen van gebruikers aan te maken. Dus in dit geval kun je ook gewoon `0` invullen. Maar als deze code bijvoorbeeld op Linux wilt draaien, dan zijn de toegangsrechten wel van belang. De octale code `0666` geeft alle gebruikers het recht om deze mqueue te gebruiken. Zie eventueel http://www.cis.rit.edu/class/simg211/unixintro/Access_Permissions.html voor verdere uitleg.

De functie geeft de *handle*[4] van de mqueue terug, van het type `mqd_t`. Deze handle kun je later gebruiken om de mqueueu te benaderen. Zet de returnwaarde van `mq_open` in de variabele genaamd `postbus`.

- Maak nu de twee pthreads aan op de plaats waar nu de `while(1)`-loop staat. De functies die door deze threads worden uitgevoerd kun je het beste definiëren in het bestand `uartecho.c` boven de functie `mainThread`. Deze functies moeten de UART-handle genaamd `uart` en de handle van de mqueue genaamd `postbus` via een parameter van het type `void *` ontvangen. Om dit mogelijk te maken moet je deze twee variabelen `uart` en `postbus` in een `stuct` plaatsen.
- Wacht aan het einde van de functie `mainThread` tot beide pthreads klaar zijn door de functie `pthread_join` twee keer aan te roepen. Bijvoorbeeld als volgt:

```
pthread_join(t1, NULL);
pthread_join(t2, NULL);
```

  Dit is nodig omdat we niet willen dat de lokale variabelen die in de functie `mainThread` gedefinieerd worden, al opgeruimd worden voordat de zojuist aangemaakte pthreads zijn afgelopen.
- De threads kunnen nu berichten versturen en ontvangen met behulp van de functies:

```
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t ↩
 ↪ msg_len, unsigned int *msg_prio);
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t ↩
 ↪ msg_len, unsigned int msg_prio);
```

  Hierin is:

  - `mqdes` de handle van de mqueue;
  - `msg_ptr` de pointer naar het bericht;
  - `msg_len` de lengte van het huidige bericht in bytes;
  - `msg_prio` de prioriteit van het huidige bericht.

4.2.3 Als je netjes hebt geprogrammeerd, dan heb je in het programma *géén* globale variabelen gebruikt maar heb je de 'handles' van de UART en van de message queue netjes via

[4] De term 'handle' betekent letterlijk 'handvat'. In operating systemen wordt de term 'handle' ook gebruikt om te verwijzen naar een specifieke door het operating systeem beheerde resource (zoals bijvoorbeeld een mqueue). Zie https://en.wikipedia.org/wiki/Handle_(computing).

een parameter meegegeven aan de twee threads. Pas je programma aan als je (per ongeluk) toch globale variabelen hebt gebruik. Het in de les besproken voorbeeld vind je hier.

4.2.4 Als je het programma uit opdracht 4.2.2 test, zul je merken dat als je return drukt de cursor wel naar het begin van de regel gaat maar niet naar de volgende regel gaat. Los dit op door bij ontvangst van het karakter `'\r'` twee karakters in de mqueue te plaatsen: `'\r'` en `'\n'`.