



Rapporttitel

ONDERTITEL RAPPORT

Naam | Cursustitel | Datum

Inhoud

Week 3.....	3
3.3.....	3
B	3
C	3
D	3
3.4.....	4
3.5.....	4
3.6.....	5
3.7.....	8
Week 4.....	9
4.1	9
D	9
4.2.....	10
A	10
B	12
4.3.....	13
A	13
4.4	14
4.5.....	17
Week 5.....	18
5.1	18
A	18
B	18
C	19
D	19
E	20
F	20
G	21
5.2.....	21
A	21
C	25

5.3.....	25
A.....	25
B.....	25
A.....	29
B.....	30
C.....	31
Prioriteit 5 berekening	31
Prioriteit 4 berekening	31
Prioriteit 3 berekening	32
Prioriteit 2 berekening	33
Prioriteit 1 berekening.....	34

Week 3

3.3

B

Code:

```
volatile bool flag = false;

void SysTick_Handler(void){
    flag = true;
}

while (1)
{
    // Wait a moment
    while (!flag);
    flag = false;
    GPIOD->ODR ^= 0xF000;
}
```

C

Omdat de flag verandert moet kunnen worden in de ISR, en de compiler moet deze variabele negeren tijdens het optimaliseren. Als dit niet gebeurt kan de variabele zodanig veranderd worden dat de ISR en niet meer bij kan.

D

Code:

```
void SysTick_Handler(void){
}

while (1)
{
    __asm__("WFI");
    GPIOD->ODR ^= 0xF000;
}
```

3.4

```
while (1)
{
    __WFI();
    GPIOD->ODR ^= 0xF000;
}
```

3.5

```
GPIOD->ODR = 0x4000;
```

```
int interrupt_counter = 0;
int color = Red;
// Do forever:
while (1)
{
    __WFI();
    interrupt_counter++;
    switch (color){
        case Red:
            if (interrupt_counter == 10){
                color = Orange;
                GPIOD->ODR = 0x2000;
            }
        case Orange:
            if (interrupt_counter == 12){
                color = Green;
                GPIOD->ODR = 0x1000;
            }
        case Green:
            if (interrupt_counter == 20){
                color = Red;
                interrupt_counter = 0;
                GPIOD->ODR = 0x4000;
            }
    }
}
```

3.6

Different functions for task management:

```
int Add_task(void (*func)(void), int period, int initial_delay){
    static int nr_tasks = 0;
    // counter voor hoeveelheid taken
    if (nr_tasks > 7){
        return 0;
        // als groter dan 7 (alleen 8
        // mogelijk in dit geval) return 0 en doe niks.
    }
    else{
        task_list[nr_tasks].function = func;
        // vul taak parameters in
        task_list[nr_tasks].period = period;
        task_list[nr_tasks].counter = initial_delay;
        task_list[nr_tasks].initial_delay = initial_delay;
        task_list[nr_tasks].state = WAITING;
        nr_tasks++;
        // verhoog de hoeveelheid in de array
        return 1;
        // return succes
    }
}

int runReadyTasks(void){
    int i = 0;
    for (i = 7; i >= 0; i--){
        // for loop voor maximaal 8 taken
        if (task_list[i].state != NOT_AVAILABLE){
            // check of er uberhaupt een taak is.
            if(task_list[i].state == READY){
                // als de staat READY is
                task_list[i].function();
                // run de functie
                task_list[i].counter = task_list[i].period;
                // reset de counter naar de periode.
            }
        }
    }
    return 1;
}
```

```

void SysTick_Handler(void){
    // takenhandler
    for (int i = 7; i >= 0; i--){
        // maximaal 8 taken.
        if (task_list[i].state != NOT_AVAILABLE){
            // check of er uberhaupt een taak is.
            if(task_list[i].counter > 0){
                // Als de counter groter is dan 0
                task_list[i].counter -= 1;
                // verlaag het met 1 (tick)
            }
            if(task_list[i].counter == 0){
                // Als het wel gelijk is aan 0
                task_list[i].state = READY;
                // Maak de state ready
            }
            else{
                task_list[i].state = WAITING;
                // Als de counter niet gelijk is aan 0,
                // maak de staat WAITING
            }
        }
    }
}

```

Task structure with task functions

```
enum {NOT_AVAILABLE, WAITING, READY};

void Toggle_Green(void){
    GPIOD->ODR ^= 0x1000;
}
void Toggle_Orange(void){
    GPIOD->ODR ^= 0x2000;
}
void Toggle_Red(void){
    GPIOD->ODR ^= 0x4000;
}
void Toggle_Blue(void){
    GPIOD->ODR ^= 0x8000;
}

struct tasks{
    void (*function)(void);
    int period;
    int counter;
    int initial_delay;
    int state;
} task;

struct tasks task_list[8];
```

Main function section for task adding and interrupt waking:


```
int ret = Add_task(&Toggle_Green, 200, 100);
if (ret == 0){
    while(1);
}


ret = Add_task(&Toggle_Orange, 500, 200);
if (ret == 0){
    while(1);
}
ret = Add_task(&Toggle_Red, 750, 300);
if (ret == 0){
    while(1);
}
ret = Add_task(&Toggle_Blue, 300, 400);
if (ret == 0){
    while(1);
}


// Do forever:
while (1)
{
    __WFI();
    runReadyTasks();
}
```




```

▶  P0 → Δ1.007 220 500 s (992.83 mHz)

▶  P1 → Δ2.517 854 625 s (397.16 mHz)

▶  P2 → Δ3.778 156 500 s (264.68 mHz)

▶  P3 → Δ1.510 818 125 s (661.89 mHz)


```


Results from logic analyzer


Logic signal	Corresponding function
P ₀	Toggle_Green
P ₁	Toggle_Orange
P ₂	Toggle_Red
P ₃	Toggle_Blue


3.7

```

▶  P0 → Δ504.18275 ms (1.98 Hz)

▶  P1 → Δ1.008 487 375 s (991.58 mHz)

▶  P2 → Δ1.512 623 000 s (661.1 mHz)

▶  P3 → Δ2.016 929 875 s (495.8 mHz)

```

Results from the logic analyzer including the initial delay ticks. Every tick is 0.005s, therefore 100 ticks is equal to 0.5S





Logic signal	LED	Set cycles (ticks)	Corresponding time (s)
P ₀	Green	100	0.5
P ₁	Orange	200	1
P ₂	Red	300	1.5
P ₃	Blue	400	2

Week 4

4.1

D

The times are x4 because every 1ms the tasks switches to the next task. Therefore after 4ms task 1,2,3 and 4 are 1ms further in their own scope. This is an order execution of tasks, without priorities.

```
▶  P0 → Δ400.0375 ms (2.5 Hz)
▶  P1 → Δ800.075 ms (1.25 Hz)
▶  P2 → Δ1.600 149 500 s (624.94 mHz)
▶  P3 → Δ3.200 299 500 s (312.47 mHz)
```

Logic signal	LED	Set cycles (ticks)	Corresponding time (ms)
P ₀	Green	100	400
P ₁	Orange	200	800
P ₂	Red	300	1200
P ₃	Blue	400	1600

4.2

A

Adjustments in code:

New function:

```
void non_blocking_delay(unsigned int ticks){
    extern unsigned int tick_delay;
    extern void taskYield(void);
    tick_delay = ticks;
    taskYield();
}
```

Edited taskYield:

```
void taskYield(void)
{
    currentTask->state = RUNNING;
    currentTask->counter = tick_delay;
    asm(" svc #1");
}
```

Added counter:

```
unsigned int tick_delay;
```

Adjustments in schedule:





```
task * schedule()
{
    task* tempTaskPtr = currentTask;
    task *idleTaskPtr = &taskList[IDLE_TASK];
    //tempTaskPtr->state = READY;
    // current task state = ready

    int teller=0;

    //Find next ready, non idle task.
    // -> do while will always execute once
    do
    // zolang volgende taak niet
    klaar is, en er door de hele lijst is gelopen
    {
        if (tempTaskPtr->counter == 0 && tempTaskPtr != idleTaskPtr){
            tempTaskPtr->state = READY;
        }
        else if (tempTaskPtr != idleTaskPtr){
            tempTaskPtr->counter--;
            tempTaskPtr++;
        }

        if( (tempTaskPtr-1) == idleTaskPtr || tempTaskPtr == idleTaskPtr)
        // eind van de lijst -> naar begin weer gaan
        {
            //since idle task is the last in the list, we've reached the
            end
            //and need to continue at the beginning
            tempTaskPtr = &taskList[0];
        }
    }
}
```

B

▶		P0	→	$\Delta 100.0095$ ms (10 Hz)
▶		P1	→	$\Delta 200.0195$ ms (5 Hz)
▶		P2	→	$\Delta 400.0385$ ms (2.5 Hz)
▶		P3	→	$\Delta 800.0765$ ms (1.25 Hz)

Logic signal	LED	Set cycles (ticks)	Corresponding time (ms)
P0	Green	100	100
P1	Orange	200	200
P2	Red	300	400
P3	Blue	400	800

4.3

A

```
void toggleGreen(void)
{
    while(1)
    {
        SysTick->LOAD = 2 * CLOCK_FREQ_IN_KHz - 1;           //
2ms    GPIO->ODR ^= 1 << GREEN;
        non_blocking_delay(100);
    }
}
```

Code for changing period to 2ms

```
void startVersdOS(uint16_t sysTickPeriodIn_ms) {
    // Configure SysTick of 1 ms
    SysTick->LOAD = sysTickPeriodIn_ms * CLOCK_FREQ_IN_KHz - 1;
    SysTick->VAL = 0;
    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk //Clock source selection =
Processor clock (AHB)
                    | SysTick_CTRL_TICKINT_Msk //Counting down to zero
to asserts the SysTick exception request
                    | SysTick_CTRL_ENABLE_Msk; //Counter enable

    //set systick and pendsv interrupt priority to lowest.
    //svc will be highest.
    SCB->SHP[2] |= 0xFF<<24;
    SCB->SHP[2] |= 0xFF<<16;

    // Create Idle task
    addTaskToListAtIndex(idleTask, 128, -1, IDLE_TASK, UNPRIVILEGED);
    __set_CONTROL(1); // set
control bit (npriv) to (un)privileged mode

    __asm(" wfi"); // Sleep until next SysTick
}
```

Added a set Control function to set the nPRIV bit high.

4.4

```
void SVC_Handler(void)
{
    // Get svc and systickperiod from psp stack
    uint32_t *psp_address;
    uint32_t *pc_address;
    uint32_t svc_number;
    uint32_t sysTickPeriodInMs;
    psp_address = (uint32_t *)__get_PSP(); //
    retrieve address of psp
    sysTickPeriodInMs = *psp_address;
    // send value of address of psp to variable
    pc_address = (uint32_t *)*(psp_address + 6);
    svc_number = *(pc_address-1);
    // try to filter out the value 2.
    svc_number &= (0x00FF0000);
    svc_number = svc_number >> 16;
    switch(svc_number)
    {
        case 1:
            currentTask->state = RUNNING;
            currentTask->counter = tick_delay + 1;
            taskToExecute = schedule();
            SCB->ICSR |= (1<<28);
            break;
        case 2:
            if (sysTickPeriodInMs < 10){
                SysTick->CTRL = 0;

                SysTick->LOAD = sysTickPeriodInMs * CLOCK_FREQ_IN_KHz - 1;
                SysTick->VAL = 0;
                SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk //Clock source
                selection = Processor clock (AHB)
                | SysTick_CTRL_TICKINT_Msk
            //Counting down to zero to asserts the SysTick exception request
                | SysTick_CTRL_ENABLE_Msk;
            //Counter enable

            }
            break;
    }

    //determine what to do based on the svc number and systick period.
}
```

```

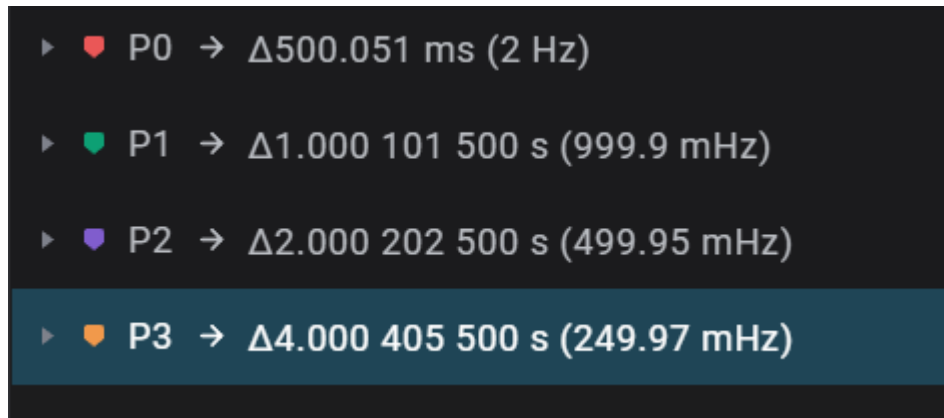
void taskYield(void)
{
    asm(" svc #1");
    // Calls function SVC_Handler

    // interrupt returns to msp with pushing certain
    registers to stack
}

void ChangeSystick(uint32_t sysTickPeriodIn_ms)
{
    asm(" svc #2");
    // Calls function SVC_Handler
}

void toggleGreen(void)
{
    while(1)
    {
        //SysTick->LOAD = 2 * CLOCK_FREQ_IN_KHz - 1; // 2ms
        extern void ChangeSystick(uint32_t sysTickPeriodIn_ms);
        ChangeSystick(5);
        GPIOD->ODR ^= 1 << GREEN;
        non_blocking_delay(100);
    }
}

```

With 1 tick being 5ms (x5 from previous assignment) this is the result

Logic signal	LED	Set cycles (ticks)	Corresponding time (ms)
P ₀	Green	100	500
P ₁	Orange	200	1000
P ₂	Red	400	2000
P ₃	Blue	800	4000

4.5

Deze opdracht hebben we niet gemaakt

Week 5

5.1

A

downloaden van het project

B

importeren van het project, en checken of deze werkt. Hierbij krijgt de consument een prioriteit van 3 (hoogste), de frikandel producent een prioriteit van 2, en de kroket producent een prioriteit van 1 (laagste). Dit is de output die we krijgen op ons console:

```
Output for Consumer priority = 3 frikandel Producer priority = 2 Krokot
Producer priority = 1
Thread: 0x200015e8 starts
Thread: 0x20001b80 with argument: F starts
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Thread: 0x20002118 with argument: K starts
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
Thread: 0x200015e8 stops
Thread: 0x20002118 stops
```

Eerst wordt er gevraagd naar de prioriteiten. Hierbij krijgt de consument een prioriteit van 3 (hoogste), de frikandel producent een prioriteit van 2, en de kroket producent een prioriteit van 1 (laagste). Hierna start het de consumenten thread (0x200015e8) als eerste, want deze heeft de hoogste prioriteit. Omdat de frikandellen producent prioriteit 2 heeft, start deze thread (0x20001b80) hierna. Direct hierna start de consument met het consumeren van de frikandellen dat te zien is aan “FFFFFFFFFFFF...”. Als de frikandellen allemaal gemaakt zijn, stopt de frikandellen producent. Omdat de consument de hoogste prioriteit heeft, is deze eerder klaar met alle frikandellen consumeren dan dat de frikandel producent thread stopt. Omdat de kroket producent prioriteit 1 heeft, start deze thread (0x20002118) hierna. Direct hierna start de consument met het consumeren van de krocketten dat te zien is aan “KKKKKKKK...”. Als de krocketten allemaal gemaakt zijn, stopt de krocketten producent. Omdat de consument de hoogste prioriteit heeft, is deze eerder klaar met alle krocketten consumeren dan dat de kroket producent thread stopt.

Er zal maar één snack in de buffer terecht komen voordat de eerste snack is geconsumeerd, omdat de consument de hoogste prioriteit heeft.

C

sem_wait zorgt er voor dat maar één taak tegelijkertijd gedaan kan worden. Normaal zou eerst de get() functie uitgevoerd worden, en daarna de rest van de consument functie. Nu wordt er eerst begonnen aan de consument functie, en deze is nog niet afgemaakt, en kan dus niet naar de get() functie. Hierdoor loopt stalls het programma.

D

compileer het originele programma met prioriteiten: consument = 1, frikandel
 producent =2 en kroket producent =3. Dit genereerd de volgende output:

[illegible]

Aangezien de kroket producent (0x20002118) de hoogste prioriteit heeft, start deze eerst. Hierna start de frikandel producent (0x20001b80) meteen. Nadat deze twee zijn gestart, start de consument (0x200015e8) die de laagste prioriteit heeft. De kroketten worden als eerst geconsumeerd. Nadat de kroket producent klaar is, worden er nog kroketten geconsumeerd door de consument, omdat er nog kroketten in de buffer zitten. `sem_Empty = -1`. Dit betekent dat de buffer helemaal vol zit `+1`, en er dus nog 9 kroketten geconsumeerd worden nadat de kroket producent is gestopt. Hierna worden de frikandellen geconsumeerd. Nadat de frikandellen producent is gestopt, worden er nog 9 frikandellen geconsumeerd, omdat de buffer nog vol zit, en `sem_Empty = -1`. Hierna stopt ook de consumenten thread, en is het programma klaar.

E

Ik verwacht dat het programma het zelfde draait als in opdracht D, alleen met de frikandel producer en kroket producer omgedraaid.

```
Output for Consumer priority = 1 frikandel Producer priority = 3 Kroket
Producer priority = 2
Thread: 0x20001b80 with argument: F starts
Thread: 0x20002118 with argument: K starts
Thread: 0x200015e8 starts
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFThread: 0x20001b80 stops
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
KKKKKKKKKKKKKKKKKKKKKKKKKKKKThread: 0x20002118 stops
KKKKKKKKKKThread: 0x200015e8 stops
```

Het programma draait inderdaad zoals we hadden verwacht.

F

Ik verwacht dat het programma de twee producers om en om laat draaien, en dus steeds van producer wisselt. daarom zal de consument ook om en om een kroket en een frikandel krijgen.

```
Output for Consumer priority = 2 frikandel Producer priority = 1 Kroket
Producer priority = 1
Thread: 0x200015e8 starts
Thread: 0x20001b80 with argument: F starts
FThread: 0x20002118 with argument: K starts
KFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFK
KFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFK
KFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFKFK
FFFFThread: 0x20002118 stops
FFFFThread: 0x200015e8 stops
Thread: 0x20001b80 stops
```

Het programma draait zoals we hadden verwacht.

G

Ik verwacht dat het programma de twee producers om en om laat draaien, en dus steeds van producer wisselt. daarom zal de consument ook om en om een kroket en een frikandel krijgen. We krijgen daarnaast ook weer hetzelfde geval als in opdracht D, waar de buffer nog niet leeg is nadat de producer is gestopt.

[illegible]

Het programma draait zoals we hadden verwacht.

5.2

A

maken van het project, en check of het voorbeeld werkt.

B vervang het globale variabel buffer en de semaforen semMutialExclusive, semEmpty en semFilled met een message queue.

```
#include "main.h"
#include <pthread.h>
#include <semaphore.h>
#include <mqueue.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

mqd_t mqdes;
struct mq_attr mqAttrs;

void check_errno(int error)
{
    if (error < 0)
    {
        perror("Error");
        while (1);
    }
}

void check(int error)
{
    if (error != 0)
    {
        printf("Error: %s\n", strerror(error));
        while (1);
    }
}

sem_t semPrintf; // binary semaphore: used for mutual exclusive use of printf

void *producer(void *arg) // function for producer thread
{
    char c = *(char *)arg;
    check_errno( sem_wait(&semPrintf) );
    check_errno( printf("Thread: %p with argument: %c starts\n",
pthread_self(), c) );
    check_errno( sem_post(&semPrintf) );
    for (int i = 0; i < 100; ++i)
    {
        check_errno( mq_send(mqdes, (char *)&c, sizeof(c), 0) );
    }
    check_errno( sem_wait(&semPrintf) );
    check_errno( printf("Thread: %p stops\n", pthread_self()) );
    check_errno( sem_post(&semPrintf) );
    return NULL;
}

void *consumer(void) // function for consumer thread
{

```

```

check_errno( sem_wait(&semPrintf) );
check_errno( printf("Thread: %p starts\n", pthread_self()) );
check_errno( sem_post(&semPrintf) );
for (int i = 0; i < 200; ++i)
{
    int msg;
    check_errno( mq_receive(mqdes, (char *)&msg, sizeof(msg), NULL) );

    check_errno( sem_wait(&semPrintf) );
    check_errno( printf("%c", msg) );
    check_errno( fflush(stdout) );
    check_errno( sem_post(&semPrintf) );
}
check_errno( sem_wait(&semPrintf) );
check_errno( printf("Thread: %p stops\n", pthread_self()) );
check_errno( sem_post(&semPrintf) );
return NULL;
}

int read_prio(char *process_name)
{
    int prio;
    do
    {
        check_errno( printf("Enter priority for process %s [1..15]: ",
process_name) );
        check_errno( fflush(stdout) );
    }
    while (scanf("%d", &prio) != 1 || prio < 1 || prio > 15);
    return prio;
}

void *main_thread(void *arg)
{
    int prioc = read_prio("Consumer");
    int priop1 = read_prio("Frikandel Producer");
    int priop2 = read_prio("Kroket Producer");

    check_errno( printf("Output for Consumer priority = %d ", prioc) );
    check_errno( printf("frikandel Producer priority = %d ", priop1) );
    check_errno( printf("Kroket Producer priority = %d\n", priop2) );

    check_errno( sem_init(&semPrintf, 0, 1) ); // allow one thread
exclusively to use printf

    // Zie UM2609 6.2.1.2 Add to registry
    // https://www.st.com/resource/en/user_manual/dm00629856-stm32cubeide-
user-guide-stmicroelectronics.pdf
    vQueueAddToRegistry((QueueHandle_t) &semPrintf.xSemaphore,
"semPrintf");

    mqAttrs.mq_maxmsg = 3;
    mqAttrs.mq_msgsize = sizeof(int);
    mqAttrs.mq_flags = 0;
    check_errno((int) (mqdes = mq_open("/ints", O_RDWR | O_CREAT, 0666,
&mqAttrs) ));

    pthread_attr_t ptac, ptap1, ptap2;
    check( pthread_attr_init(&ptac) );

```



```

check( pthread_attr_init(&ptap1) );
check( pthread_attr_init(&ptap2) );

check( pthread_attr_setstacksize(&ptac, 1024) );
check( pthread_attr_setstacksize(&ptap1, 1024) );
check( pthread_attr_setstacksize(&ptap2, 1024) );

struct sched_param spc, spp1, spp2;
check( pthread_attr_getschedparam(&ptac, &spc) );
check( pthread_attr_getschedparam(&ptap1, &spp1) );
check( pthread_attr_getschedparam(&ptap2, &spp2) );

spc.sched_priority = prioc;
spp1.sched_priority = priop1;
spp2.sched_priority = priop2;

check( pthread_attr_setschedparam(&ptac, &spc) );
check( pthread_attr_setschedparam(&ptap1, &spp1) );
check( pthread_attr_setschedparam(&ptap2, &spp2) );

pthread_t ptc, ptp1, ptp2;
char frikandel = 'F', kroket = 'K';

check( pthread_create(&ptc, &ptac, consumer, &mqdes) );
check( pthread_create(&ptp1, &ptap1, producer, &frikandel) );
check( pthread_create(&ptp2, &ptap2, producer, &kroket) );

check( pthread_join(ptc, NULL) );
check( pthread_join(ptp1, NULL) );
check( pthread_join(ptp2, NULL) );

check_errno( sem_destroy(&semPrintf) );

check( pthread_attr_destroy(&ptac) );
check( pthread_attr_destroy(&ptap1) );
check( pthread_attr_destroy(&ptap2) );

check( mq_close(mqdes) );
check( mq_unlink("/ints") );

return NULL;
}

int main(void)
{
    Board_Init();

    pthread_attr_t pta;
    check( pthread_attr_init(&pta) );
    check( pthread_attr_setdetachstate(&pta, PTHREAD_CREATE_DETACHED) );
    check( pthread_attr_setstacksize(&pta, 1024) );

    struct sched_param sp;
    check( pthread_attr_getschedparam(&pta, &sp) );
    // The main thread must have the highest priority because this thread
    will start
    // the other threads and we want to study the interaction between those
    other threads
    sp.sched_priority = 15;
    check( pthread_attr_setschedparam(&pta, &sp) );

```

```

pthread_t pt;
check( pthread_create(&pt, &pta, main_thread, NULL) );

printf("\n");
vTaskStartScheduler();
/* We should never get here as control is now taken by the scheduler
*/

check( pthread_attr_destroy(&pta) );

return EXIT_SUCCESS;
}

```

C

welke prioriteit zou je de *messages* moeten geven om real-time gedrag te implementeren?

Op het moment dat een bepaald bericht een deadline heeft, die korter is als de ander, zal deze een hogere prioriteit moeten krijgen.

5.3

A

maken van het project, en check of het voorbeeld werkt.

B

vervang de semafoor *semPrintf* met een mutex.

```

#include "main.h"
#include <pthread.h>
#include <semaphore.h>
#include <mqueue.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

mqd_t mqdes;
struct mq_attr mqAttrs;
pthread_mutex_t m;

void check_errno(int error)
{
    if (error < 0)
    {
        perror("Error");
        while (1);
    }
}

void check(int error)
{
    if (error != 0)
    {
        printf("Error: %s\n", strerror(error));
        while (1);
    }
}

void *producer(void *arg) // function for producer thread
{
    char c = *(char *)arg;
    check( pthread_mutex_lock(&m) );
    check_errno( printf("Thread: %p with argument: %c starts\n",
pthread_self(), c) );
    check( pthread_mutex_unlock(&m) );
    for (int i = 0; i < 100; ++i)
    {
        check_errno( mq_send(mqdes, (char *)&c, sizeof(c), 0) );
    }
    check( pthread_mutex_lock(&m) );
    check_errno( printf("Thread: %p stops\n", pthread_self()) );
    check( pthread_mutex_unlock(&m) );
    return NULL;
}

void *consumer(void *arg) // function for consumer thread
{

```

```

    check( pthread_mutex_lock(&m) );
    check_errno( printf("Thread: %p starts\n", pthread_self()) );
    check( pthread_mutex_unlock(&m) );
    for (int i = 0; i < 200; ++i)
    {
        int msg;
        check_errno( mq_receive(mqdes, (char *)&msg, sizeof(msg), NULL) );
        check( pthread_mutex_lock(&m) );
        check_errno( printf("%c", msg) );
        check_errno( fflush(stdout) );
        check( pthread_mutex_unlock(&m) );
    }
    check_errno( printf("Thread: %p stops\n", pthread_self()) );
    return NULL;
}

int read_prio(char *process_name)
{
    int prio;
    do
    {
        check_errno( printf("Enter priority for process %s [1..15]: ",
process_name) );
        check_errno( fflush(stdout) );
    }
    while (scanf("%d", &prio) != 1 || prio < 1 || prio > 15);
    return prio;
}

void *main_thread(void *arg)
{
    pthread_mutexattr_t ma;
    check( pthread_mutexattr_init(&ma) );
    check( pthread_mutex_init(&m, &ma) );

    int prioc = read_prio("Consumer");
    int priop1 = read_prio("Frikandel Producer");
    int priop2 = read_prio("Kroket Producer");

    check_errno( printf("Output for Consumer priority = %d ", prioc) );
    check_errno( printf("frikandel Producer priority = %d ", priop1) );
    check_errno( printf("Kroket Producer priority = %d\n", priop2) );

    mqAttrs.mq_maxmsg = 3;
    mqAttrs.mq_msgsize = sizeof(int);
    mqAttrs.mq_flags = 0;
    check_errno((int) (mqdes = mq_open("/ints", O_RDWR | O_CREAT, 0666,
&mqAttrs) ));

    pthread_attr_t ptac, ptap1, ptap2;
    check( pthread_attr_init(&ptac) );
    check( pthread_attr_init(&ptap1) );
    check( pthread_attr_init(&ptap2) );

    check( pthread_attr_setstacksize(&ptac, 1024) );

```

```

check( pthread_attr_setstacksize(&ptap1, 1024) );
check( pthread_attr_setstacksize(&ptap2, 1024) );

struct sched_param spc, spp1, spp2;
check( pthread_attr_getschedparam(&ptac, &spc) );
check( pthread_attr_getschedparam(&ptap1, &spp1) );
check( pthread_attr_getschedparam(&ptap2, &spp2) );

spc.sched_priority = prioc;
spp1.sched_priority = priop1;
spp2.sched_priority = priop2;

check( pthread_attr_setschedparam(&ptac, &spc) );
check( pthread_attr_setschedparam(&ptap1, &spp1) );
check( pthread_attr_setschedparam(&ptap2, &spp2) );

pthread_t ptc, ptp1, ptp2;
char frikandel = 'F', kroket = 'K';

check( pthread_create(&ptc, &ptac, consumer, &mqdes) );
check( pthread_create(&ptp1, &ptap1, producer, &frikandel) );
check( pthread_create(&ptp2, &ptap2, producer, &kroket) );

check( pthread_join(ptc, NULL) );
check( pthread_join(ptp1, NULL) );
check( pthread_join(ptp2, NULL) );

check( pthread_attr_destroy(&ptac) );
check( pthread_attr_destroy(&ptap1) );
check( pthread_attr_destroy(&ptap2) );

check( mq_close(mqdes) );
check( mq_unlink("/ints") );

return NULL;
}

int main(void)
{
    Board_Init();

    pthread_attr_t pta;
    check( pthread_attr_init(&pta) );
    check( pthread_attr_setdetachstate(&pta, PTHREAD_CREATE_DETACHED) );
    check( pthread_attr_setstacksize(&pta, 1024) );

    struct sched_param sp;
    check( pthread_attr_getschedparam(&pta, &sp) );

    sp.sched_priority = 15;
    check( pthread_attr_setschedparam(&pta, &sp) );

    pthread_t pt;
    check( pthread_create(&pt, &pta, main_thread, NULL) );

    printf("\\n");
    vTaskStartScheduler();
}

```

```

    /* We should never get here as control is now taken by the scheduler
*/
    check( pthread_attr_destroy(&pta) );

    return EXIT_SUCCESS;
}

```

Week 6 rekenopdracht

A

Bepaal voor elke taak i de prioriteit P_i als DMPA (Deadline Monotonic Priority Assignment) wordt gebruikt. De hoogste prioriteit die je mag gebruiken is 5 en de laagste prioriteit die je mag gebruiken is 1.

Bij DMPA wordt niet gekeken naar de periodetijd maar naar de deadline. Hierbij is de kleinste deadline de grootste prioriteit. Hierdoor krijg je:

Taak (i)	Prioriteit (P)
1	2
2	4
3	5
4	3
5	1

B

Bereken voor elke taak i de blocking time B_i .

$$B_i = \sum_{k=1}^K usage(k, i) C_k$$

- B_i = Maximale blocking time voor taak i
- K = totaal aantal gebruikte resources
- $usage(k, i)$ = Boleaanse functie met de volgende statements:
 - Als een lagere prioriteit resource k gebruikt
 - En als taak i of een hogere prioriteit resource k gebruikt
- Als beide statements waar zijn wordt dit stukje van de functie 1, anders 0.
- C_k = maximale vergrendeltijd van resource k

Als je dit invult bij elke taak krijg je de volgende resultaten:

Blocking time	Resource 1	Resource 2	Resource 3	Resource 4	Totaal blocking time (B_i)
B_1	0	100	0	0	100
B_2	0	100	50	20	170
B_3	25	100	0	0	125
B_4	0	100	50	0	150
B_5	0	0	0	0	0

C

Bereken voor elke taak i of de deadline wordt gehaald en geef, indien de deadline wordt gehaald, de response tijd R_i .

Om antwoord te geven op deze vraag moeten alle response tijden van de taken geanalyseerd en berekend worden. Dit is hieronder te zien van hoogste naar laagste prioriteit:

Prioriteit 5 berekening

$$R_3 = C_3 + B_3 = 250 + 125 = 375$$

De volgende functies van berekening 2 tm 5 worden geïtereerd tot $R_i = R_{temp}$, hieronder is de code ook te vinden in python per berekening

Prioriteit 4 berekening

$$R_{2temp} = C_2 + B_2 + \left\lceil \frac{R_2}{T_3} \right\rceil * C_3$$

```
import math
```

```
def looped_function():
    R2 = 0
    for i in range(20):
        R2t = 225 + 175 + (math.ceil(R2 / 500) * 250)

        if (R2 == R2t):
            return
        R2 = R2t
    print(R2)
    return
```

```
looped_function()
```

Resultaat:

```
400
650
900
```

```
>>> R2 is dus 900.
```

$R_3 = D_3$. Hierdoor zal de deadline behaald worden.

Prioriteit 3 berekening

$$R_{4temp} = C_4 + B_4 + \left\lceil \frac{R_4}{T_2} \right\rceil * C_2 + \left\lceil \frac{R_4}{T_3} \right\rceil * C_3$$

```
def looped_function():  
    R2 = 0  
    for i in range(20):  
        R2t = 75 + 150 + (math.ceil(R2 / 500) * 250) + (math.ceil(R2 / 1000)  
        if (R2 == R2t):  
            return  
        R2 = R2t  
    print(R2)  
return
```

looped_function()

Resultaat:

225

700

950

>>>

R4 is dus 950

$R_3 < D_3$. Hierdoor zal de deadline behaald worden.

Prioriteit 2 berekening

$$R_{1temp} = C_1 + B_1 + \left\lceil \frac{R_1}{T_4} \right\rceil * C_4 + \left\lceil \frac{R_1}{T_2} \right\rceil * C_2 + \left\lceil \frac{R_1}{T_3} \right\rceil * C_3$$

```
def looped_function():
    R2 = 0
    for i in range(20):
        R2t = 100 + 100 + (math.ceil(R2 / 500) * 250) + (math.ceil(R2 / 1000)
* 225) + (math.ceil(R2 / 2000) * 75)
        if (R2 == R2t):
            return
        R2 = R2t
    print(R2)
    return
```

looped_function()

Resultaat:

```
200
750
1000

>>> R1 = 1000
```

$R_3 < D_3$. Hierdoor zal de deadline behaald worden.

Prioriteit 1 berekening

$$R_{5temp} = C_5 + B_5 + \left\lceil \frac{R_5}{T_1} \right\rceil * C_1 + \left\lceil \frac{R_5}{T_4} \right\rceil * C_4 + \left\lceil \frac{R_5}{T_2} \right\rceil * C_2 + \left\lceil \frac{R_5}{T_3} \right\rceil * C_3$$

```
def looped_function():
    R2 = 0
    for i in range(20):
        R2t = 125 + 0 + (math.ceil(R2 / 500) * 250) + (math.ceil(R2 / 1000) *
225) + (math.ceil(R2 / 2000) * 75) + (math.ceil(R2 / 1750) * 100)
        if (R2 == R2t):
            return
        R2 = R2t
    print(R2)
    return
```

looped_function()

Resultaat:

```
125
775
1025
1500
R5 = 1500
```

$R_3 > D_3$. Hierdoor zal de deadline niet behaald worden.