DE HAAGSE HOGESCHOOL

Technology, Innovation & Society Delft

VOORBLAD SCHRIFTELIJKE TOETSEN

OPLEIDING	**: ELEKTROTECHNIEK**
TOETSCODE	**: RTSYST-co1**
GROEP	**: ECV**
TOETSDATUM	**: 12 JUNI 2012**
TIJD	**: 11.00 – 12.30 uur**
AANTAL PAGINA'S (incl. voorblad)	**: 6**
DEZE TOETS BESTAAT UIT	**: 3 open vragen**
GEBRUIK HULPMIDDELEN	**: JA**
TOEGESTANE HULPMIDDELEN	**: Tijdens dit tentamen mogen alle boeken, dictaten, aantekeningen enz. worden gebruikt.**
OVERIGE OPMERKINGEN	**: Geen**
OPSTELLER VAN DEZE TOETS	**: Harry Broeders**
TWEEDE LEZER VAN DEZE TOETS	**: Jesse op den Brouw**

BELANGRIJKSTE PUNTEN UIT ARTIKEL 12 VAN DE ONDERWIJS- EN EXAMENREGELING:

- je dient je via Osiris ingeschreven te hebben voor deze toets
- schrijf je naam, je studentnummer, de toetscode en de naam van de docent meteen op het tentamenpapier
- leg je identiteitsbewijs op de hoek van de tafel
- zet alle elektronische communicatiemiddelen (mobiele telefoon, PDA, etc.) uit en stop deze in je tas; deze mogen niet als calculator of klok worden gebruikt
- je mag het lokaal het eerste halfuur niet verlaten
- volg de instructies op het toetsvoorblad
- steek je hand op als je een vraag hebt

HHS TIS Delft

DE HAAGSE HOGESCHOOL

KLAS(SEN)	:	ECV		BLAD	: 1 van 5 BLADEN
TOETS	:	real-time systemen (TOETS01)		DOCENT	: Harry Broeders
CODE	:	**RTSYST-co1**		DATUM	: 12 juni 2012
KWARTAAL	:	4	TYPE : tentamen	TIJD	: 11:00 – 12:30

Tijdens dit tentamen mogen **alle** boeken, dictaten, aantekeningen enz. worden gebruikt.

Bij elke opgave staat tussen haakjes het maximale aantal te behalen punten vermeld.
Eindcijfer = (aantal behaalde punten + 10) / 10.

1. Op pagina 4 en 5 van dit tentamen vind je een vereenvoudigde versie van het programma dat je bij practicumopgave 2 hebt geanalyseerd.

(20) Wat is de uitvoer van dit programma? Verklaar je antwoord door duidelijk stap voor stap aan te geven wat er gebeurt.

2. Een *barrier* is een synchronisatie mechanisme met de volgende werking:

- Als een barrier wordt aangemaakt dan wordt een integer *n* opgegeven.
- Een barrier heeft slecht één memberfunctie genaamd *wait*. Als deze functie door een thread wordt aangeroepen dan moet deze thread wachten tot in totaal *n* threads de functie *wait* hebben aangeroepen.
- Als in totaal *n* (of meer) threads de *wait* functie hebben aangeroepen hoeft geen enkele thread nog te wachten.

In de C++11 standaard is **geen** barier opgenomen. In het onderstaande (onvolledige) programma is een barier geïmplementeerd met de wel in C++11 beschikbare *mutex* en *condition_variable*.

```
#include <thread>
#include <mutex>
#include <condition_variable>
#include <iostream>

using namespace std;

class barrier {
public:
    barrier(int n): teller(n) {
    }
    void wait() {
        /* Deze code moet jij invullen! */
    }
private:
    int teller;
    mutex m;
    condition_variable c;
};
```

Zie volgende blad ⇨

```
barrier b(3);

void task(int d) {
     cout << "Thread met delay " << d << " gestart" << endl;
     this_thread::sleep_for(chrono::seconds(d));
     cout << "Thread met delay " << d << " wordt wakker" << endl;
     b.wait();
     cout << "Thread met delay " << d << " passeert barrier" << endl;
}

int main() {
     thread t1(&task, 1), t2(&task, 2), t3(&task, 3);
     t1.join(); t2.join(); t3.join();
     cout << "Einde!" << endl;
     return 0;
}
```

A. (15) Geef de **exacte** uitvoer van het bovenstaande programma ervan uitgaande dat de functie `wait` correct wordt ingevuld.

B. (20) Geef de implementatie (code) van de memberfunctie `wait` van de bovenstaande class `barrier`.

Zie volgende blad ⇨

3. Een programma bestaat uit 5 threads T1 t/m T5. Deze threads gebruiken 4 gedeelde resources R1 t/m R4. Elke resource is mutual exclusive beveiligd met een mutex M1 t/m M4. Deze mutexen gebruiken het **priority inheritance protocol**. Als een thread een resource toegewezen heeft gekregen dan kan deze thread de resource voor een bepaalde maximale tijd bezetten. In de onderstaande tabel staat *k* voor het nummer van de resource en *C(k)* voor de maximale tijd dat resource *k* bezet kan worden.

k	***C(k)***
1	20
2	15
3	10
4	5

In de volgende tabel staat aangegeven welke resources elke thread gebruikt. In de onderstaande tabel staat *i* voor het nummer van de thread, *T(i)* voor de periodetijd van thread *i* en *C(i)* voor de maximale executietijd van thread *i*. Gegeven is dat de deadline van elke taak gelijk is aan zijn periodetijd.

i	***T(i)***	***C(i)***	**Gebruikt**
1	100	50	R1, R3
2	200	45	R2, R3, R4
3	350	20	R2
4	300	15	-
5	400	25	R1

Alle gegeven tijden zijn in ms.

A. (5) Bepaal de prioriteit van de 5 processen als RMPA (**Rate** Monotonic Priority Assignment) wordt gebruikt. De hoogste prioriteit is 5 en de laagste prioriteit is 1.

B. (15) Bereken voor elke thread *i* de blocking time *B(i)*.

C. (15) Bereken voor thread 1, 3 en 5 of de deadline wordt gehaald en geef, indien de deadline wordt gehaald, de response tijd R_i.

Succes!

HHS TIS Delft

DE HAAGSE HOGESCHOOL

KLAS(SEN) : ECV		BLAD	: 4 van 5 BLADEN
TOETS : real-time systemen (TOETS01)		DOCENT	: Harry Broeders
CODE : **RTSYST-co1**		DATUM	: 12 juni 2012
KWARTAAL : 4	TYPE : tentamen	TIJD	: 11:00 – 12:30

Tijdens dit tentamen mogen **alle** boeken, dictaten, aantekeningen enz. worden gebruikt.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>

#define SIZE 4

void check_errno(int error) {
     if (error < 0) {
          perror("Error");
          exit(EXIT_FAILURE);
     }
}

void check(int error) {
     if (error != 0) {
          fprintf(stderr, "Error: %s\n", strerror(error));
          exit(EXIT_FAILURE);
     }
}

char buffer[SIZE];
int indexGet = 0;
int indexPut = 0;
sem_t semEmpty;
sem_t semFilled;

void put(char c) {
     check_errno( sem_wait(&semEmpty) );
     buffer[indexPut] = c;
     indexPut++;
     if (indexPut == SIZE) {
          indexPut = 0;
     }
     check_errno( sem_post(&semFilled) );
}

char get(void) {
     char c;
     check_errno( sem_wait(&semFilled) );
     c = buffer[indexGet];
     indexGet++;
     if (indexGet == SIZE) {
          indexGet = 0;
     }
     check_errno( sem_post(&semEmpty) );
     return c;
}

void* producer(void* arg) {
     char c = *(char*)arg;
     int i;
     check_errno( putchar('1') );
     for (i = 0; i < 10; ++i) {
          put(c);
     }
     check_errno( putchar('2') );
}

void* consumer(void* arg) {
     int i;
```

```
      check_errno( putchar('3') );
      for (i = 0; i < 20; ++i) {
            check_errno( putchar(get()) );
      }
      check_errno( putchar('4') );
}

int main(int argc, char *argv[]) {
      int p;
      struct sched_param sp, spc, spp1, spp2;
      pthread_attr_t ptac, ptap1, ptap2;
      pthread_t ptp1, ptp2, ptc;
      char bamihap = 'B', nasischijf = 'N';

      check( pthread_getschedparam(pthread_self(), &p, &sp) );
      sp.sched_priority = 60;
      check( pthread_setschedparam(pthread_self(), SCHED_FIFO, &sp) );

      check_errno( sem_init(&semEmpty, 0, SIZE) );
      check_errno( sem_init(&semFilled, 0, 0) );

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

      check( pthread_attr_setinheritsched(&ptac, PTHREAD_EXPLICIT_SCHED) );
      check( pthread_attr_setinheritsched(&ptap1, PTHREAD_EXPLICIT_SCHED) );
      check( pthread_attr_setinheritsched(&ptap2, PTHREAD_EXPLICIT_SCHED) );

      check( pthread_attr_setschedpolicy(&ptac, SCHED_RR) );
      check( pthread_attr_setschedpolicy(&ptap1, SCHED_RR) );
      check( pthread_attr_setschedpolicy(&ptap2, SCHED_RR) );

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

      spc.sched_priority = 21;
      spp1.sched_priority = 20;
      spp2.sched_priority = 22;

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

      check( pthread_create(&ptc, &ptac, consumer, 0) );
      check( pthread_create(&ptp1, &ptap1, producer, &bamihap) );
      check( pthread_create(&ptp2, &ptap2, producer, &nasischijf) );

      check( pthread_join(ptc, 0) );
      check( pthread_join(ptp1, 0) );
      check( pthread_join(ptp2, 0) );

      check_errno( sem_destroy(&semEmpty) );
      check_errno( sem_destroy(&semFilled) );

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

      return EXIT_SUCCESS;
}
```