CHALMERS LINDHOLMEN

Institutionen för data- och elektroteknik

TENTAMEN

KURSNAMN	**Parallellprogrammering**
PROGRAM: namn åk / läsperiod	**Dataingenjörslinjen** **Inbyggda system Åk 3 / Lp 2**
KURSBETECKNING	**LEU 290 0199**
EXAMINATOR	**Roger Johansson**
TID FÖR TENTAMEN	**Torsdag 20 december 2001 8:30 – 12:30**
HJÄLPMEDEL	**Inga**
ANSV LÄRARE: namn telnr besöker tentamen kl	**Anders Franzén** **031 – 772 5715** **9:45 och 11:00**
DATUM FÖR ANSLAG av resultat samt av tid och plats för granskning	**Resultat anslås senast torsdag 10 januari 2002** **Granskning sker hos Anders Franzén**
ÖVRIG INFORM.	• **Motivera de val du gjort väl. Vissa uppgifter kanske inte tillhandahåller alla uppgifter som behövs för att lösa problemet. I så fall får du göra rimliga antaganden om dessa detaljer. Ange tydligt vilka antagande du gör. Om dina lösningar endast fungerar under vissa förutsättningar måste du klart tala om vilka dessa förutsättningar är.** • **Poängavdrag kan ges för onödigt långa, komplicerade, eller ostrukturerade lösningar**

NAMN (tentand): _____

Uppgift 1: Har du gjort laboration 2? (6p)

Uppgift 2: Vad menas med *liveness* och *safety*? Förklara och exemplifiera! (4p)

Uppgift 3: Betrakta de fyra **select**-satserna:

(a)

```
select
   T.Call;
   Flag := A;
or
   delay 10.0;
   Flag := B;
   delay 2.0;
end select;
```

(b)

```
select
   T.Call;
   Flag := A;
else
   delay 10.0;
   Flag := B;
   delay 2.0;
end select;
```

(c)

```
select
   T.Call;
   Flag := A;
then abort
   delay 10.0;
   Flag := B;
   delay 2.0;
end select;
```

(d)

```
select
   delay 10.0;
   Flag := A;
then abort
   T.Call;
   Flag := B;
   delay 2.0;
end select;
```

Antag att exekvering av rendezvous med T.Call tar 5 sekunder. Visa hur Flag kan få värdena A respektive B då **select**-satserna avslutats. Om något fall inte är möjligt, förklara varför. (8p)

Uppgift 4: Följande kodfragment illustrerar två olika sätt att implementera heltalssemaforer i Ada:

```
package Semaphore_Package_1 is
   protected type Semaphore(Initial: Natural := 1) is
      entry Wait;
      procedure Signal;
   private
      Value : Natural :=  Initial ;
   end Semaphore;
end Semaphore_Package_1;

package Semaphore_Package_2 is
   task type Semaphore(Initial: Natural := 1) is
      entry Wait;
      entry Signal;
```

```
    end Semaphore;
end Semaphore_Package_2;
```

(a) Visa hur paketet Semaphore_Package_1 kan implementeras. (8p)

(b) Visa hur paketet Semaphore_Package_2 kan implementeras. (8p)

(c) Diskutera för- och nackdelar med de olika lösningarna. (4p)

Uppgift 5: I problemet "Dinerande filosofer" finns det många möjliga lösningar för att kontrollera ätpinnarna. En lösning ser ut så här:

```
type Phil_ID is new Integer range 1..5;
type Allocation is array (Phil_ID) of Boolean;

protected Chopstick_Control is
    entry Take(Left, Right: Phil_ID);
    procedure Drop(Left, Right: Phil_ID);
private
    Free : Allocation := ( others => True);
    entry Retry(Left, Right: Phil_ID);
    Waiting : Natural := 0;
end Chopstick_Control;

protected body Chopstick_Control is
    entry Take(Left, Right: Phil_ID) when True is
    begin
        if Free(Left) and Free(Right) then
            Free(Left) := False;
            Free(Right) := False;
        else
            requeue Retry;
        end if;
    end Take;

    procedure Drop(Left, Right: Phil_ID) is
    begin
        Free(Left) := True;
        Free(Right) := True;
        Waiting := Retry'Count;
    end Drop;

    entry Retry(Left, Right: Phil_ID) when Waiting>0 is
    begin
        Waiting := Waiting-1;
        if Free(Left) and Free(Right) then
            Free(Left) := False;
            Free(Right) := False;
        else
            requeue Retry;
```

```
            end if;
        end Retry;
    end Chopstick_Control;
```

(a) Ett problem med ovanstående lösning är dock att filosofer kan utsättas för svält. Visa hur. (4p)

(b) Förklara hur en lösning på problemet med svält kan se ut, utan att förändra det publika gränssnittet. (8p)

Uppgift 6: Följande kodskelett är en primitiv lösning av producent/konsument-problemet i POSIX och C, där en eller flera producenter producerar data som en eller flera konsumenter konsumerar. Systemet använder en buffer med plats för ett enda element, lagrat i variabeln value.

```
static int value;

void produce(int i);
int consume();

void *producer(void *dummy) {
    int i;
    for (;;) {
        for (i=0; i<1000; i++) {
            produce(i);
        }
    }
    return 0;
}

void *consumer(void *dummy) {
    int i;
    for (;;) {
        i = consume();
        printf("%d\n", i);
    }
    return 0;
}
```

Uppgiften består i att implementera funktionerna produce och consume

- produce skall lägga värdet av parametern i value.
- consume skall hämta och returnera värdet av value.
- Samma meddelande skall aldrig komma fram mer än en gång.
- Meddelanden skall aldrig tappas bort.
- Alla meddelanden är heltal.

(a) Implementera funktionerna med hjälp av POSIX-semaforer. (8p)

(b) Implementera funktionerna med hjälp av villkorsvariabler. (8p)

Urval av POSIX-API

Funktioner från pthread.h:

```
/* Pthread functions */
int  pthread_create(pthread_t * thread,
                    pthread_attr_t * attr,
                    void * (* start_routine )( void *),
                    void * arg);
int pthread_join(pthread_t th, void **thread_return);
void pthread_exit(void *retval );

/* Mutex functions */
pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex));
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

/* Condition variable functions */
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_init(pthread_cond_t *cond,
                      pthread_condattr_t *cond_attr);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond,
                           pthread_mutex_t *mutex,
                           const struct timespec *abstime);
int pthread_cond_destroy(pthread_cond_t *cond);
```

Funktioner från semaphore.h:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t * sem);
int sem_trywait(sem_t * sem);
int sem_post(sem_t * sem);
int sem_getvalue(sem_t * sem, int * sval);
int sem_destroy(sem_t * sem);
```