Tentamen
Datorteknik och realtidssystem, TSEA81

Datum	2021-03-15
Lokal	Distanstentamen
Tid	14-18
Kurskod	TSEA81
Provkod	TEN1
Kursnamn	Datorteknik och realtidssystem
Institution	ISY
Antal uppgifter	4
Antal sidor (inklusive denna sida)	15
Kursansvarig	Anders Nilsson
Telefon under skrivtiden	013-28 2635
Kursadministratör	Maria Hamnér, 013-28 5715
Tillåtna hjälpmedel	Inga, förutom dator för att skriva lösningar på digital form samt mobiltelefon/scanner för att digitalisera handskrivna lösningar
Betygsgränser	**Poäng** **Betyg** 41-50 5 31-40 4 21-30 3 0-20 U

Viktig information

- Alla svar ska ha en motivering om inget annat anges. Om du svarar med programkod räknas kommentarer i programkoden som motivering. Svar som ej är motiverade kan leda till poängavdrag.
- Om inget annat anges ska du anta att schemaläggningsmetoden som används är *priority based preemptive scheduling*.
- Om inget annat anges antas semaforer vara starka.
- Om du är osäker på det exakta namnet för en viss funktion, skriv en kommentar om vad funktionen gör så kommer vi troligtvis att förstå vad du menar. (Detsamma gäller syntaxen för programspråket C.)
- Tänk igenom din lösning NOGGRANT och använd dig av de lösningsprinciper som kursen förevisar. Okonventionella och tvetydiga lösningar ger poängavdrag.
- Svara ALDRIG med pseudokod, om det inte specifikt efterfrågas. Pseudokod blir lätt tvetydig och därmed inte bedömningsbar.
- Lämna INTE in denna tentamen tillsammans med lösningarna. En inlämnad tentamen med eventuella anteckningar kommer inte att beaktas som en lösning.
- Skriv läsbart! Oläsbar text kan inte bedömas och ger därmed inga poäng.

Lycka till!

Uppgift 1: Schemaläggning(10p)

Ett realtidssystem med ett antal processer ska schemaläggas på en dator som enbart har en processor/kärna, dvs endast en process åt gången kan exekvera. Följande krav gäller:

- *P1* ska arbeta/köra under 5 tidsenheter i tidsintervallet [*i***x*, (*i*+1)**x*]
- *P2* ska arbeta/köra under 2 tidsenheter i tidsintervallet [*i**8, (*i*+1)*8]
- *P3* ska arbeta/köra under 1 tidsenhet i tidsintervallet [*i**3, (*i*+1)*3]

där i är ett heltal och $i \geq 0$, och x är ett heltal och $x \geq 1$.

Tidräkningen startar vid t=0 för alla processer. Du kan anta att uppstart av processer inte tar någon tid samt att processbyte inte tar någon tid. Eventuellt missat arbete under något tidsintervall ackumuleras *inte* till kommande tidsintervall.

(a) (5p) Antag att schemaläggningsmetoden Earliest Deadline First (EDF) används. Man vill att *P1* ska köra så ofta som möjligt, dvs beräkna minsta möjliga värde på x så att specifikationerna uppfylls. Visa sedan vad som händer när programmet körs genom att rita ett tidsdiagram.
Hur stor blir den faktiska utnyttjandegraden?

(b) (5p) Antag att schemaläggningsmetoden Rate Monotonic Scheduling (RMS) används. Antag samma värde på x som i a-uppgiften.
Visa vad som händer när programmet körs genom att rita ett tidsdiagram.
Kommer processerna att uppfylla kraven?
Hur stor blir den faktiska utnyttjandegraden?
Kan sambandet för RMS som säger att om $U_e \leq n(2^{1/n} - 1)$ så uppfylls kraven, användas i detta fall för att avgöra om kraven uppfylls eller inte? Motivera svaret.

Uppgift 2: Teori(12p)

(a) (3p) *Static Cyclic Scheduling* är en schemaläggningsmodell.
-Beskriv kort hur den fungerar.
-Beskriv en fördel som den har jämfört med prioritetsbaserad påtvingad schemaläggning.
-Beskriv en nackdel som den har jämfört med prioritetsbaserad påtvingad schemaläggning.

(b) (2p) `Await` är en operation som utförs på en händelsevariabel. Vilka problem kan uppstå om man använder `Await` utan att först ha reserverat den associerade semaforen? Du kan inte ge ett generellt svar, genom att t ex säga att det kan uppstå deadlock eller race condition. Ditt svar måste vara specifikt, med tanke på vad `Await` gör. Det finns i huvudsak två grundläggande problem.

(c) (3p) Vilken kontext, alltså vilken sorts information, finns lagrad i processens stack efter ett processbyte av en körande process?
I vilken ordning lagras informationen på stacken?
Varför behöver informationen lagras i just den ordningen?

(d) (2p) Kärnfunktioner såsom `Wait` och `Signal` i ett realtidssystem börjar med att stänga av avbrott och avslutar med att slå på avbrott. Däremellan kan det tänkas att schemaläggaren `schedule()` anropas och byter process, dvs utan att avbrott slagits på igen. Så när slås avbrott på igen? Det finns i huvudsak två tänkbara lägen.

(e) (2p) Ge två exempel på situationer där det finns grund för processbyte (dvs schemaläggaren `schedule()` anropas), men som av något skäl *inte* leder till ett processbyte.

Uppgift 3: Stark/svag semafor(14p)

Betrakta följande program i Simple-OS. Antag att semaforerna fungerar på så sätt som beskrivits under kursens föreläsningar. Antag att prioritetsbaserad påtvingad schemaläggning används.

```
si_semaphore S1;            // define semaphore

void p1(void)               // Task P1
{
    si_sem_wait(&S1);       // wait on semaphore
    si_wait_n_ms(4000);     // sleep for 4000 ms
    si_sem_signal(&S1);     // signal on semaphore

    si_wait_n_ms(2000);     // sleep for 2000 ms

    si_sem_wait(&S1);       // wait on semaphore
    si_wait_n_ms(2000);     // sleep for 2000 ms
    si_sem_signal(&S1);     // signal on semaphore

    while(1);               // wait for ever
}

void p2(void)               // Task P2
{
    si_wait_n_ms(2000);     // sleep for 2000 ms

    si_sem_wait(&S1);       // wait on semaphore
    si_wait_n_ms(4000);     // sleep for 4000 ms
    si_sem_signal(&S1);     // signal om semaphore

    while(1);               // wait for ever
}
```

För de följande två deluppgifterna, beskriv vad som händer från det att processerna `P1` och `P2` är körklara. Var noggrann med att tala om vilken process som är körande, vilka listor processerna ligger i vid olika tillfällen, semaforens värde samt motivera varför olika händelser sker. Listornas exakta namn är inte viktigt, bara det framgår vad deras syfte är. Redovisa detta tills det inte händer nåt mer. Redovisa gärna stegvis i tabellform där det framgår vad som är orsak och verkan. Ge en kommentar för varje steg. Antag att huvudprogrammet initierar semaforen `S1` till `1`.

(a) (8p) Antag att programmet ovan använder sig av starka semaforer. Antag att `P1` har större prioritet än `P2`.

(b) (6p) Antag att programmet ovan använder sig av svaga semaforer. Antag att `P2` har större prioritet än `P1`.

Uppgift 4: Campus-bussen med semaforer(14p)

På campus finns en automatiserad självkörande buss. Din uppgift är att skriva två programtrådar som simulerar situationen dels för själva bussen (`bus_thread`), dels för en person som ska åka med bussen (`person_thread`).

Bussen antas köra i en rundslinga med `STOPS` antal hållplatser. Hållplatsernas nummer är alltså från `0` till `STOPS-1`, där bussen åker i ordningen `0`, `1`, ..., `STOPS-1` och sedan börjar om på `0`. Bussen kan ta ett visst maximalt antal passagerare, `MAX_PASSENGERS`.
Bussen stannar vid varje hållplats, och de personer som väntar där kan då gå på, och bussen åker inte vidare förrän alla som kan gå på har gjort det. Bussen åker sedan till nästa hållplats, och där kan de personer som ska gå av göra det, och bussen åker inte vidare förrän alla som ska gå av vid den hållplatsen har gjort det.
Personer dyker upp slumpmässigt på de olika hållplatserna och ställer sig att vänta där tills bussen kommer. När bussen anlänt kan så många personer som får plats i bussen gå på. Efter att en person gjort sin resa och gått av bussen väntar personen en slumpmässig tid innan personen påbörjar en ny resa.

Din uppgift är att fullfärdiga programkoden för de båda trådarna, så att de uppfyller beskrivningen ovan.

Observera, att du i din lösning *endast får använda semaforer* som synkroniseringsmekanism för de båda trådarna, och du får ***inte*** använda händelsevariabler.

Det finns en föreslagen struct, som du får använda i din lösning om du vill, alternativt hitta på egna nödvändiga variabler:

```
struct{
  int trav[STOPS];  // number of travellers waiting at a certain stop
  int dest[STOPS];  // number of travellers going to a certain destination
  int boarders;     // number of persons that can and will board the bus
  int passengers;   // number of passengers in the bus
  int current_stop; // the current stop [0..STOPS-1]
} bus;
```

Du måste själv deklarera och initiera de semaforer du behöver använda. Du behöver *inte* skriva programkod för initieringen av semaforerna. Det räcker med att du kommenterar eller talar om vilket värde dom initieras till. Lösningen behöver *inte* hantera rättvisa, dvs den som är först på en hållplats måste inte få åka först, och först att gå på bussen måste inte få gå av först.

Huvudprogrammet antas starta en buss-tråd, `bus_thread`, och ett okänt antal person-trådar, `person_thread`.

Programkoden fortsätter på nästa sida

Tråden `bus_thread`:

```
void *bus_thread(void *arg)
{
    ...     // complete code here
}
```

Tråden `person_thread`:

```
void *person_thread(void *arg)
{
    int from_dest;  // start destination of journey
    int to_dest;    // end destination of journey

    while (1)
    {
        // generate random journey
        do
        {
            from_dest = get_random_value(STOPS);
            to_dest = get_random_value(STOPS);
        }
        while (from_dest == to_dest);

        ...     // complete code here

    // sleep a random time
    sleep_random();
    }
}
```

Tips : Med tanke på hur semaforer fungerar kan de sägas vara multifunktionella, dvs de kan var och en användas till olika ändamål.

Lösningsförslag fråga 1

Följande notation gäller:

```
# = process running
_ = process not running
. = no process running (unused time slot)
| = deadline (met)
/ = deadline (missed)
```

1a

För EDF gäller att kraven uppfylls om utnyttjandegraden $U_e \leq 1$, dvs:
$5/x + 2/8 + 1/3 \leq 1$
$5/x + 6/24 + 8/24 \leq 24/24$
$5/x \leq 10/24$
$12 \leq x$
Alltså, minsta värde på x är 12.

Med EDF schemaläggs alltid den process som har kortast tid kvar till deadline. Då flera processer har lika lång tid kvar och en av dom redan kör låter man den processen fortsätta för att slippa ett processbyte.

```
P1_ _ _ _ # # _ # # # _ _|_ _ # _ # # _ # # _ _ _|
P2_ # # _ _ _ _ _|_ _ _ # _ # _ _|_ _ _ _ _ # # _|
P3# _ _|# _ _|# _ _|_ # _|# _ _|# _ _|# _ _|_ _ #|
   0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2
   1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
```

Alla processer klarar sina deadlines och den faktiska utnyttjandegraden blir 1 (ty x=12 ger Ue=1). Vid tidpunkten t=24 blir samtliga processer samtidigt redo att köra igen, dvs där börjar förloppet om.

(Kommentar: Lösningen behöver visa hela tidsförloppet (inklusive deadlines) fram till t=24, att körande process fortsätter (i förekommande fall), korrekt dra slutsatsen om vilka processer som klarar sina deadlines, att värdet på x=12, samt att den faktiska utnyttjandegraden blir 1)

1b

MED RMS får processerna prioriteter utefter hur ofta de ska köras, dvs ju kortare tidsintervall ju högre prioritet. Prioriteten bli alltså P3 > P2 > P1. Därefter används schemaläggningsmetoden priority based preemptive scheduling.

```
P1_ _ _ _ # # _ # _ _ _ #/_ # # _ _ _ _ # # _ # .|
P2_ # # _ _ _ _ _|# _ # _ _ _ _ _|# # _ _ _ _ _ .|
P3# _ _|# _ _|# _ _|# _ _|# _ _|# _ _|# _ _|# _ .|
   0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2
   1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
```

P2 och *P3* klarar sina deadlines, men *P1* missar sin deadline vid *t=12*. Den faktiska utnyttjandegraden blir 23/24. Vid tidpunkten t=24 blir samtliga processer samtidigt redo att köra igen, dvs där börjar förloppet om.
Sambandet $U_e \leq n(2^{1/n} - 1)$ kan i detta fall inte användas för att avgöra om processerna klarar sina krav. Även om sambandet inte uppfylls (ty beräknad U_e=1) så kan processerna fortfarande klara sina krav, även om det inte var så i detta fall.

(Kommentar: Lösningen behöver visa hela tidsförloppet (inklusive deadlines) fram till t=24, tala om de inbördes prioriteterna för P1, P2 och P3 vid RMS, korrekt dra slutsatsen om vilka processer som klarar sina deadlines (och visa var och vilka som inte gör det), visa outnyttjade tidsintervall, samt att den faktiska utnyttjandegraden blir 23/24, samt motivera varför sambandet för U_e inte kan användas.)

Lösningsförslag fråga 2

2a

De ingående processerna har en förbestämd cyklisk körordning.
Fördelar: Den är enkel att implementera, då det bara är att följa körordningen utan att behöva ta hänsyn till något annat. Den är snabb vid processbyte, då själva bytet har lite overhead.
Nackdelar: Den anpassar sig inte efter situationen. T ex om en process som kör inte kan fortsätta för att en gemensam resurs är upptagen så går det körintervallet förlorat. Med PBPS kan en annan process schemaläggas istället.

2b

Det korta svaret är att Await riskerar att ge andra processer tillgång till en kritisk region, när den inte får göra det. Om en process väntar på semaforen kommer Await då att flytta den till ready-listan och den processen kan bli körande. Om ingen process väntar på semaforen kommer Await att räkna upp semaforen till ett för högt värde. Båda dessa saker kan leda till att flera processer samtidigt får tillgång till den kritiska regionen, vilket sedan skulle kunna orsaka t ex deadlock eller race condition.

2c

Eventuella temporärdata, återhoppsadress (från PC i CPU), registerinnehåll (från CPU), i den ordningen.
När processen kör kan den tänkas spara temoprärdata. Vid processbyte lagras sedan återhoppsadressen (innan registerinnehållet) för att återhoppsadressen ska återhämtas sist då det utgör hoppet till själva processen när den återupptar exekveringen.

2d

Antingen när exekveringen återkommer till ett tidigare anrop av schedule() (varpå avbrott sedan slås på i slutet på den kärnfunktionen), eller när en annan kärnfunktion anropas (typiskt av en annan process) men inte orsakar processbyte via schedule(), varpå avbrott slås på i slutet igen.

2e

En ny process skapas, men den har inte högre prioritet än körande process.
En process har väntat färdigt på en timer, men den processen har inte högre prioritet än körande process.
En process gör Signal på en semafor som en annan process väntar på, men den väntande processen har inte högre prioritet än den körande processen.

Lösningsförslag fråga 3:

Här finns flera tillfällen som kan (men inte nödvändigtvis måste) orsaka ett processbyte. När en process gör sleep (och en annan process är körklar), när en process anropar wait (och semaforens värde är 0), när en process kör signal (och en högre prioriterad process är körklar) eller när en process vaknar från sleep (och den processen har högre prioritet än körande process).

Tre listor blir aktuella, en time-list för då sleep anropas (T), en wait-list för semaforen (W) och en ready-lista för körklara processer (R).

3a

Här används en stark semafor och P1 har större prioritet än P2.

	Orsak	Verkan					
		Kör	S	R	W	T	Kommentar
1)	Init	P1	1	P1,P2			P1 högst prio
2)	P1:Wait	P1	0	P1,P2			S– ty ingen i W
3)	P1:sleep(4)	-	0	P2		P1	P1 -> T
4)	Sched	P2	0	P2		P1	Sched väljer P2
5)	P2:sleep(2)	-	0			P1,P2	P2 -> T, Sched väljer idle
6)	P2:awake	-	0	P2		P1	P2 vaknar -> R
7)	Sched	P2	0	P2		P1	Sched väljer P2
8)	P2:Wait	-	0		P2	P1	P2->W ty S=0, Sched väljer idle
9)	P1:awake	-	0	P1	P2		P1 vaknar -> R
10)	Sched	P1	0	P1	P2		Sched väljer P1
11)	P1:Signal	P1	0	P1,P2			S oför. ty P2 i W, P2->R
12)	Sched	P1	0	P1,P2			Sched väljer P1, ty högst prio
13)	P1:sleep(2)	-	0	P2		P1	P1 -> T
14)	Sched	P2	0	P2		P1	Sched väljer P2
15)	P2:sleep(4)	-	0			P1,P2	P2 -> T, Sched väljer idle
16)	P1:awake	-	0	P1		P2	P1 vaknar -> R
17)	Sched	P1	0	P1		P2	Sched väljer P1
18)	P1:Wait	-	0		P1	P2	P1->W, ty S=0, Sched väljer idle
19)	P2:awake	-	0	P2	P1		P2 vaknar -> R
20)	Sched	P2	0	P2	P1		Sched väljer P2
21)	P2:Signal	P2	0	P1,P2			S oför. ty P1 i W, P1->R
22)	Sched	P1	0	P1,P2			Sched väljer P1, t högst prio
23)	P1:sleep(2)	-	0	P2		P1	P1 -> T
24)	Sched	P2	0	P2		P1	Sched väljer P2
25)	P2:while(1)	P2	0	P2		P1	P2 i while(1)
26)	P1:awake	P2	0	P1,P2			P1 vaknar -> R
27)	Sched	P1	0	P1,P2			Sched väljer P1, ty högst prio
28)	P1:Signal	P1	1	P1,P2			S++, ty ingen i W
29)	P1:while(1)	P1	1	P1,P2			Fastnar här ty P1 högst prio

3b

Här används en svag semafor och P2 har större prioritet än P1.

	Orsak	Verkan					
		Kör	S	R	W	T	Kommentar
1)	Init	P2	1	P1,P2			P2 högst prio
2)	P2:sleep(2)	-	1	P1		P2	P2 ->T
3)	Sched	P1	1	P1		P2	Sched väljer P1
4)	P1:Wait	P1	0	P1		P2	S–, ty ingen i W
5)	P1:sleep(4)	-	0			P1,P2	P1 ->T, Sched väljer idle
6)	P2:awake	-	0	P2		P1	P2 vaknar ->R
7)	Sched	P2	0	P2		P1	Sched väljer P2
8)	P2:Wait	-	0		P2	P1	P2->W, ty S=0, Sched väljer idle
9)	P1:awake	-	0	P1	P2		P1 vaknar ->R
10)	Sched	P1	0	P1	P2		Sched väljer P1
11)	P1:Signal	P1	1	P1,P2			S++ (svag sem), P2->R
12)	Sched	P2	1	P1,P2			Sched väljer P2, ty högst prio
13)	P2:Wait	P2	0	P1,P2			S– ty ingen i W
14)	P2:sleep(4)	-	0	P1		P2	P2->T
15)	Sched	P1	0	P1		P2	Sched väljer P1
16)	P1:sleep(2)	-	0			P1,P2	P1->T, Sched väljer idle
17)	P1:awake	-	0	P1		P2	P1 vaknar ->R
18)	Sched	P1	0	P1		P2	Sched väljer P1
19)	P1:Wait	-	0		P1	P2	P1->W, ty S=0, Sched väljer idle
20)	P2:awake	-	0	P2	P1		P2 vaknar ->R
21)	Sched	P2	0	P2	P1		Sched väljer P2
22)	P2:Signal	P2	1	P1,P2			S++ (svag sem), P1->R
23)	Sched	P2	1	P1,P2			Sched väljer P2, ty högst prio
24)	P2:while(1)	P2	1	P1,P2			Fastnar här ty P2 högst prio

(Kommentar: Lösningarna i tabellerna ovan är tämligen detaljerade. En inlämnad lösning behöver inte vara fullt så detaljerad, men måste ända ta med alla händelser och även visa vad som händer när en process vaknar från sleep)

Lösningsförslag fråga 4:

Svårigheten i uppgiften ligger nog främst i att hitta lämpliga semaforer. På något sätt behövs dels någon semafor för att agera mutex för gemensamma variabler, STOPS antal semaforer per destination, dels för att synkronisera/räkna antal personer som ska kunna gå på bussen (boardQueue), dels för att räkna/synkronisera antal resande som ska gå av bussen (unboardQueue). Dessutom behövs ett par semaforer för att signalera från person_thread till bus_thread när sista person går på bussen och när sista person går av bussen.

Följande semaforer har deklarerats:

```
// Queue for passengers waiting at a certain stop,
// signal as many times as available places in bus.
// Initiate all to 0.
sem_t boardQueue[STOPS];
// signal to bus from last person boarding bus that all
// passengers (possible and willing) has boarded.
// Inititate to 0
sem_t allAboard;
// Queue for passengers leaving at a certain stop,
// signal as many times as passengers to unboard.
// Initiate all to 0
sem_t unboardQueue[STOPS];
// Signal to bus from last person leaving bus, that all
// who should leave have left.
// Inititate to 0
sem_t allLeft;
// Mutex for mutual variables.
// Inititate to 1
sem_t mutex;
```

Tråden `bus_thread` skulle kunna lösas enligt följande:

```
void *bus_thread(void *arg)
{
    // any_boarders=1 if anyone will board the bus at current stop
    int any_boarders = 0;
    // any_unboarders=1 if anyone will unboard tha bus at current stop
    int any_unboarders = 0;
    // loop variable
    int i;

    while (1)
    {

        // reserve common variables
        sem_wait(&mutex);
        // calculate number of persons that can board the bus
        bus.boarders = MAX_PASSENGERS - bus.passengers;
        // adjust if less number of persons wants to board
        if (bus.trav[bus.current_stop] < bus.boarders)
        {
            bus.boarders = bus.trav[bus.current_stop];
        }
        // signal to bus.boarders number of travellers waiting at this stop
        for (i=0; i<bus.boarders; i++)
        {
            sem_post(&boardQueue[bus.current_stop]);
        }
        // any_boarders=1 if anyone will board the bus at current stop
        any_boarders = (bus.boarders>0);
```

```
            // release common variables
            sem_post(&mutex);


            // wait until last passengers has boarded bus
            if (any_boarders)
            {
                sem_wait(&allAboard);
            }


            // reserve common variables
            sem_wait(&mutex);
            // drive to next stop
            bus.current_stop++;
            if (bus.current_stop == STOPS) {
                bus.current_stop = 0;
            }
            // any_unboarders=1 if any passenger will unboard the bus at current
            any_unboarders = (bus.dest[bus.current_stop] > 0);
            // signal to all passengers leaving at current stop
            for (i=0; i<bus.dest[bus.current_stop]; i++)
            {
                sem_post(&unboardQueue[bus.current_stop]);
            }
            // release common variables
            sem_post(&mutex);


            // wait until last passenger who should leave, has left
            if (any_unboarders)
            {
                sem_wait(&allLeft);
            }

        }
    }
```

Tråden `person_thread` skulle kunna lösas enligt följande:

```
    void *person_thread(void *arg)
    {
        int from_dest;
        int to_dest;

        while (1)
        {

            // generate random journey
            do
```

```
{
    from_dest = get_random_value(STOPS);
    to_dest = get_random_value(STOPS);
}
while (from_dest == to_dest);


// reserve common variables
sem_wait(&mutex);
// one more traveller waiting at from_dest
bus.trav[from_dest]++;
// release common variables
sem_post(&mutex);


// wait to board the bus at from_dest
sem_wait(&boardQueue[from_dest]);


// reserve common variables
sem_wait(&mutex);
// one less traveller waiting at from_dest
bus.trav[from_dest]--;
// one less traveller to board the bus
bus.boarders--;
// one more passenger in bus
bus.passengers++;
// one more passenger going to to_dest
bus.dest[to_dest]++;
// if last passenger boarding
if (bus.boarders == 0)
{
    // signal allAboard
    sem_post(&allAboard);
}
// release common variables
sem_post(&mutex);


// wait to arrive at to_dest
sem_wait(&unboardQueue[to_dest]);


// reserve common variables
sem_wait(&mutex);
// unboard the bus
bus.dest[current_stop]--;
// if last passenger to unboard
if (bus.dest[current_stop] == 0)
{
```

```
            // signal that all who should, have unboarded
            sem_post(&allLeft);
        }
        // release common variables
        sem_post(&mutex);


        // sleep a random time
        sleep_random();
    }
}
```

(Kommentar: Korrektioner som måste göras i inlämnad lösning för att den ska fungera ger olika mycket poängavdrag beroende på hur mycket som måste korrigeras för att lösningen ska fungera. Om lösningen inte har tillräcklig stomme som ger någon grundläggande funktion alls, ges inga poäng)