КОНКУРЕНТНО И ДИСТРИБУИРАНО ПРОГРАМИРАЊЕ

ВЕРЗИЈА 2015 1.2

САДРЖАЈ

Садржај 2
Семафори 4
Задатак 1. Међусобно искључивање 4
Задатак 2. Условна синхронизација 4
Задатак 3. Readers – Writers problem 6
Задатак 4. Producer – Consumer problem 7
Задатак 5. Atomic broadcast problem 8
Задатак 6. Atomic broadcast problem B 9
Задатак 7. Dining philosophers problem 10
Задатак 8. The dining savages problem 12
Задатак 9. Barrier Synchronization 13
Задатак 10. The H2O problem 14
Задатак 11. The child care problem 15
Региони 17
Задатак 1. Међусобно искључивање 17
Задатак 2. Условна синхронизација 17
Задатак 3. One lane bridge problem 18
Задатак 4. One lane bridge problem 18
Задатак 5. Dining philosophers problem 19
Задатак 6. Dining philosophers problem 19
Задатак 7. Dining philosophers problem 19
Задатак 8. Dining philosophers problem 20
Задатак 9. Readers – Writers problem 20
Задатак 10. Cigarette Smokers' problem 22
Монитори 24
Задатак 1. Readers – Writers problem 24
Задатак 2. Readers – Writers problem 24
Задатак 3. Прихватник 25
Задатак 4. Тајмер 26
Задатак 5. Тајмер 26
Задатак 6. Producer – Consumer/Bounded Buffer problem 26
Задатак 7. Dining philosophers problem 27
Прослеђивање порука 29
Задатак 1. Синхрона, асинхрона и условна комуникација 29
Задатак 2. Dining Philosophers Problem 30
Задатак 3. Broadcast 31
Задатак 4. Broadcast 32
Задатак 5. Readers – Writers Problem 32
Задатак 6. Game of Life 34
CSP 36
Задатак 1. Семафор 36
Задатак 2. Прослеђивање текста 36
Задатак 3. Прослеђивање текста 36
Задатак 4. Читање картица 36
Задатак 5. Обрада текста 36
Задатак 6. Producer - Consumer 37
Задатак 7. Дељење бројева 37
Задатак 8. Факторијел 37
Задатак 9. Скуп 38
Задатак 10. Скуп 38

Задатак 11. Скуп ... 38
Задатак 12. Множење матрице ... 39
Задатак 13. Dining philosophers problem ... 40
Задатак 14. Израчунавање интеграла ... 41
CONIC ... 42
Задатак 1. Копирање поруке ... 42
Задатак 2. Бидирекциони прстен ... 42
Задатак 3. Обрада података ... 43
Задатак 4. Bounded Buffer ... 44
Задатак 5. Dining philosophers problem ... 45
ADA ... 47
Задатак 1. Једноелементни прихватник ... 47
Задатак 2. Комуникација са поузданим везама ... 47
Задатак 3. Клијент-сервер ... 48
Задатак 4. Conic to Ada ... 49
Задатак 5. Readers – Writers problem ... 50
Задатак 6. Producer – Consumer/Bounded Buffer ... 52
Задатак 7. Cigarette Smokers' problem ... 53
Linda ... 56
Задатак 1. Семафори ... 56
Задатак 2. Dining philosophers problem ... 56
Задатак 3. Клијент-сервер ... 56
Задатак 4. Клијент-сервер ... 57
Задатак 5. Cigarette Smokers' problem ... 58
Задатак 6. Readers – Writers problem ... 59
Задатак 7. Проблем избора ... 60
Задатак 8. Проблем лифтова ... 60
Java ... 62
Задатак 1. Условна синхронизација - семафори ... 62
Задатак 2. Dining philosophers problem ... 63
Задатак 3. Условна синхронизација - региони ... 65
Задатак 4. Прелазак моста ... 67
Задатак 5. Прелазак моста ... 69
Задатак 6. Условна синхронизација - монитори ... 71
(Producer/Consumer problem) ... 71
Задатак 7. Reentrant monitor ... 73
Задатак 8. Readers – Writers problem ... 73
Задатак 9. Readers – Writers problem ... 75
Задатак 10. Заустављање ... 77
Задатак 11. The roller coaster problem ... 78
Задатак 12. The Santa Claus problem ... 79
Задатак 13. The bus problem ... 83
Задатак 14. Chat ... 86
Задатак 15. Клијент-сервер ... 88
Задатак 16. The Savings Account problem ... 103

СЕМАФОРИ

Задатак 1. **Међусобно искључивање**

Дат је упоредни програм на проширеном Pascal-у:

```
program timedependent;
var x: shared integer;
begin
    x := 2;
    cobegin
        x := x + 1;
        x := x - 1;
    coend;
    writeln ('x = ', x);
end.
```

а) Навести све могуће излазне резултате наведеног програма.

б) Употребом семафора отклонити временску зависност у програму.

Решење:

б)

Редослед	излазна вредност x
r1 w1 r2 w2	2
r1 r2 w1 w2	1
r1 r2 w2 w1	3
r2 r1 w1 w2	1
r2 r1 w2 w1	3
r2 w2 r1 w1	2

б)

```
program timedependent;
var x: shared integer;
     s: semaphore;
begin
    x := 2;
    init (s,1);
    cobegin
        begin wait (s); x := x + 1; signal (s) end;
        begin wait (s); x := x - 1; signal (s) end;
    coend;
    writeln ('x = ', x);
end.
```

Задатак 2. **Условна синхронизација**

Дат је упоредни програм на проширеном Pascal-у:

```
program graph;
const n =    ...;
var    x: shared integer;
       y: shared integer;
procedure makepoints;
var i: integer;
begin
    for i := 1 to n do
        begin x := i; y := i * i end
end;
procedure printpoints;
var i: integer;
begin
    for i := 0 to n do
        begin write (' (', x, ','); write (y,') ') end
end;
begin
```

```
    x := 0; y:= 0;
    cobegin
        makepoints;
        printpoints
    coend
end. {graph}
```

Жељени излаз програма је низ парова облика:
(0,0) (1,1) (2,4) ... (n,n^2)
а) Одредити све могуће стварне излазе програма за случај n=1.
б) Отклонити временску зависност у датом програму употребом семафора.
Решење:
а)

$x_m\ y_m\ x_{0p}\ y_{0p}\ x_{1p}\ y_{1p}$	(1,1) (1,1)*
$x_m\ x_{0p}\ y_m\ y_{0p}\ x_{1p}\ y_{1p}$	(1,1) (1,1)
$x_m\ x_{0p}\ y_{0p}\ y_m\ x_{1p}\ y_{1p}$	(1,0) (1,1)*
$x_m\ x_{0p}\ y_{0p}\ x_{1p}\ y_m\ y_{1p}$	(1,0) (1,1)
$x_m\ x_{0p}\ y_{0p}\ x_{1p}\ y_{1p}\ y_m$	(1,0) (1,0)*
$x_{0p}\ x_m\ y_m\ y_{0p}\ x_{1p}\ y_{1p}$	(0,1) (1,1)*
$x_{0p}\ x_m\ y_{0p}\ y_m\ x_{1p}\ y_{1p}$	(0,0) (1,1)*
$x_{0p}\ x_m\ y_{0p}\ x_{1p}\ y_m\ y_{1p}$	(0,0) (1,1)
$x_{0p}\ x_m\ y_{0p}\ x_{1p}\ y_{1p}\ y_m$	(0,0) (1,0)*
$x_{0p}\ y_{0p}\ x_m\ y_m\ x_{1p}\ y_{1p}$	(0,0) (1,1)
$x_{0p}\ y_{0p}\ x_m\ x_{1p}\ y_m\ y_{1p}$	(0,0) (1,1)
$x_{0p}\ y_{0p}\ x_m\ x_{1p}\ y_{1p}\ y_m$	(0,0) (1,0)
$x_{0p}\ y_{0p}\ x_{1p}\ x_m\ y_m\ y_{1p}$	(0,0) (0,1)*
$x_{0p}\ y_{0p}\ x_{1p}\ x_m\ y_{1p}\ y_m$	(0,0) (0,0)*
$x_{0p}\ y_{0p}\ x_{1p}\ y_{1p}\ x_m\ y_m$	(0,0) (0,0)

б)

```
program graph;
const n =    ...;
var    x: shared integer;
       y: shared integer;
       full, empty: semaphore;
procedure makepoints;
var i: integer;
begin
    for i := 1 to n do
        begin
            wait (empty);
            x := i; y := i * i;
            signal (full)
        end
end;
procedure printpoints;
var i: integer;
begin
    for i := 0 to n do
        begin
            wait (full);
            write (' (', x, ','); write (y,')');
            signal (empty)
        end
end;
begin
    init (full,1); init (empty,0);
    x := 0; y:= 0;
```

```
    cobegin
        makepoints;
        printpoints
    coend
end. {graph}
```

Задатак 3. **Readers – Writers problem**

(**Reader's preference solution**)

```
program Readers_Writers;
var    mutexR : semaphore;
       db : semaphore;
       readerCount : integer;

procedure Reader(ID : integer);
begin
    while (true) do
    begin
        wait(mutexR);
        readerCount := readerCount + 1;
        if (readerCount = 1) then wait(db);
        signal(mutexR);
        read_db;
        wait(mutexR);
        readerCount := readerCount - 1;
        if (readerCount = 0) then signal(db);
        signal(mutexR);
    end
end;

procedure Writer(ID : integer);
begin
    while (true) do
    begin
        create_data;
        wait(db);
        write_db;
        signal(db);
    end;
end;

begin
    init(mutexR, 1);
    init (db, 1);
    readerCount := 0;
    cobegin
        Writer(0);
        Writer(1);
        Reader(0);
        Reader(1);
        Reader(2);
    coend;
end.
```

(**Fair solution**)

```
program Readers_Writers;
var    mutexR : semaphore;
       db : semaphore;
       in: semaphore;
       readerCount : integer;

procedure Reader(ID : integer);
begin
```

```
    while (true) do
    begin
        wait(in);
        wait(mutexR);
        readerCount := readerCount + 1;
        if (readerCount = 1) then wait(db);
        signal(mutexR);
        signal(in);
        read_db;
        wait(mutexR);
        readerCount := readerCount - 1;
        if (readerCount = 0) then signal(db);
        signal(mutexR);
    end
end;

procedure Writer(ID : integer);
begin
    while (true) do
    begin
        create_data;
        wait(in);
        wait(db);
        write_db;
        signal(db);
        signal(in);
    end;
end;

begin
    init(mutexR, 1);
    init (db, 1);
    init (in, 1);
    readerCount := 0;
    cobegin
        Writer(0);
        Writer(1);
        Reader(0);
        Reader(1);
        Reader(2);
    coend;
end.
```

Задатак 4. **Producer – Consumer problem**

```
program Producer_Consumer;
const BufferSize = 3;
var     mutex : semaphore;
        empty : semaphore;
        full : semaphore;

procedure Producer(ID : integer);
var item : integer;
begin
    while (true) do
    begin
        make_new(item);
        wait(empty);
        wait(mutex);
        put_item(item);
        signal(mutex);
        signal(full);
    end;
end;
```

```
procedure Consumer(ID : integer);
var item : integer;
begin
    while (true) do
    begin
        wait(full);
        wait(mutex);
        remove_item(item);
        signal(mutex);
        signal(empty);
        consume_item(item);
    end;
end;
begin
    init(mutex,1);
    init(empty, BufferSize);
    init(full, 0);
    cobegin
        Producer(0);
        Producer(1);
        Producer(2);
        Consumer(0);
        Consumer(1);
        Consumer(2);
    coend;
end.
```

Задатак 5. **Atomic broadcast problem**

Постоји један произвођач и N потрошача који деле заједнички једноелементни бафер. Произвођач убацује производ у бафер и чека док свих N потрошача не узму исти тај производ. Тада започиње нови циклус производње.

Решење:

```
program AtomicBroadcast;
const N = 5;
var    mutex : semaphore;
    empty : semaphore;
    full : array [1..N] of semaphore;
    num : integer;
    index : integer;
...
procedure Producer;
var item, index : integer;
begin
    while (true) do
    begin
        wait(empty);
        make_new(item);
        for index := 1 to N do signal(full[index]);
    end;
end;

procedure Consumer(ID : integer);
var item : integer;
begin
    while (true) do
    begin
        wait(full[ID]);
        wait(mutex);
        get_item(item);
        num := num + 1;
        if (num = N) then
        begin
```

```
            signal(empty);
            num := 0;
        end;
        signal(mutex);
        consume_item(item);
    end;
end;
begin
    init(mutex,1);
    init(empty, 1);
    for index := 1 to N do init(full[index], 0);
    num := 0;
    cobegin
        Producer;
        Consumer(1);
        ...
        Consumer(N);
    coend;
end.
```

Задатак 6. **Atomic broadcast problem B**

Постоји један произвођач и N потрошача који деле заједнички бафер капацитета B. Произвођач убацује производ у бафер и то само у слободне слотове на који чекају свих N потрошача. Сваки потрошач мора да прими производ у тачно оном редоследу у коме су произведени, мада различити потрошачи могу у исто време да узимају различите приозводе.

Решење:

```
program AtomicBroadcastB;
const N = 5;
    B = 2;
var    mutex : semaphore;
    empty : semaphore;
    full : array [1..N] of semaphore;

    buffer : array [1..B] of integer;
    num : array [1..B] of integer;
    readFromIndex : array [1..N] of integer;
    writeToIndex : integer;

    index : integer;

procedure put_item(var item : integer);
begin
    buffer[writeToIndex] := item;
    writeToIndex := (writeToIndex mod B) + 1;
end;

procedure Producer;
var item, index : integer;
begin
    while (true) do
    begin
        wait(empty);
        make_new(item);
        put_item(data);
        for index := 1 to N do signal(full[index]);
    end;
end;

procedure Consumer(ID : integer);
var item : integer;
begin
```

```
    while (true) do
    begin
        wait(full[ID]);
        wait(mutex);

        item := buffer[readFromIndex[ID]];
        get_item(item, ID);

        num[readFromIndex[ID]] := num[readFromIndex[ID]] + 1;
        if (num[readFromIndex[ID]] = N) then
        begin
            signal(empty);
            num[readFromIndex[ID]] := 0;
        end;
        readFromIndex[ID]:= (readFromIndex[ID] mod B) + 1;
        signal(mutex);
        consume_item(item);
    end;
end;
begin
    init(mutex,1);
    init(empty, B);
    for index := 1 to N do init(full[index], 0);
    for index := 1 to N do readFromIndex[index] := 1;
    for index := 1 to B do num [index] := 0;

    writeToIndex := 1;

    cobegin
        Producer;
        Consumer(1);
        ...
        Consumer(N);
    coend;
end.
```

Задатак 7. **Dining philosophers problem**

Пет филозофа седи око стола. Сваки филозоф наизменично једе и размишља. Испред сваког филозофа је тањир шпагета. Када филозоф пожели да једе, он узима две виљушке које се налазе уз његов тањир. На столу, међутим, има само пет виљушки. Значи, филозоф може да једе само када ниједан од његових суседа не једе. Написати алгоритам за филозофа (0<=i<=4). (Упутство: спречити блокирање).

Решење:

```
procedure Philosopher(i : integer);
begin
    while (true) do
        begin
            think;
            acquire_forks;
            eat;
            release_forks;
        end
end;
```

Прво решење

```
program Dining_Philosophers;
const n =    5;
var    fork: array [0..n-1] of semaphore;
       i: integer;
```

```
procedure think; begin ... end;
procedure eat; begin ... end;

procedure Philosopher(i : integer);

var firstodd, secondeven: 0..n-1;
begin
    if (i mod 2 = 1) then begin
        firstodd := i;
        secondeven := (i+1) mod n
    end
    else
    begin
        firstodd := (i+1) mod n;
        secondeven := i
    end;
    while (true) do
    begin
        think;
        wait(fork[firstodd]);
        wait(fork[secondeven]);
        eat;
        signal(fork[firstodd]);
        signal(fork[secondeven])
    end
end;

begin
    for i:=0 to n-1 do init(fork[i],1);
    cobegin
        Philosopher(0);
        Philosopher(1);
        Philosopher(2);
        Philosopher(3);
        Philosopher(4)
    coend
end.
```

Друго решење

```
program Dining_Philosophers;
const n =    5;
var    ticket: semaphore;
       fork: array [0..n-1] of semaphore;
       I: integer;

procedure think; begin ... end;
procedure eat; begin ... end;

procedure Philosopher(i : integer);
var left, right: integer;
begin
    left := i;
    right := (i + 1) mod n;
    while (true) do
        begin
            think;
            wait (ticket);
            wait (fork[left]);
            wait (fork[right]);
            eat;
            signal (fork[right]);
            signal (fork[left]);
            signal (ticket)
        end
```

```
end;

begin
    init (ticket, n-1 );
    for i:=0 to n-1 do init(fork[i],1);
    cobegin
        Philosopher(0);
        Philosopher(1);
        Philosopher(2);
        Philosopher(3);
        Philosopher(4);
    coend
end.
```

Задатак 8. **The dining savages problem**

Племе људождера једе заједничку вечеру из казана који може да прими M порција куваних мисионара (The dining savages problem). Када људождер пожели да руча онда се он сам послужи из заједничког казана, уколико казан није празан. Уколико је казан празан људождер буди кувара и сачека док кувар не напуни казан. Није дозвољено будити кувара уколико се налази бар мало хране у казану. Користећи семафоре написати програм који симулира понашање људождера и кувара.

Решење:

```
program DiningSavages;
const M =...;
var    cook: semaphore;
       savager: semaphore;
       mutex: semaphore;
       servings: shared integer;

procedure PrepareLunch;
begin
    …
end

procedure GetServingFromPot;
begin
    …
end

procedure SavageCook;
begin
    while (true) do
    begin
        wait (cook);
        PrepareLunch;
        servings := M;
        signal (savager)
    end;
end;

procedure Savage(i : integer);
begin
    while (true) do
    begin
        wait(mutex);
        if (servings = 0) then
        begin
            signal (cook);
            wait (savager);
        end;
        servings := servings - 1;
        GetServingFromPot ;
        signal (mutex);
```

```
    end;
    eat;
end;

begin
    servings := 0;
    init(cook, 0);
    init(savager, 0);
    init(mutex, 1);
    cobegin
        SavageCook();
        Savage(1);
        Savage(2);
        ...
    coend;
end.
```

Задатак 9. **Barrier Synchronization**

Разматра се проблем синхронизације на баријери (*Barrier Synchronization*). Синхронизациона баријера омогућава нитима да на њој сачекају док тачно N нити не достигне одређену тачку у извршавању пре него што било која од тих нити не настави са својим извршавање. Користећи семафоре решити овај проблем. Омогућити да се иста баријера може користити већи број пута.

Решење:

Решење користи расподељене бинарне семафоре и технику предаје штафетне палице

```
program Barrier(input, output);
const   N = ...;

var     barrier1, barrier2 : Semaphore;
var     cnt : Integer;

procedure Pass(id : integer);
begin
        wait(barrier1);
        cnt := cnt + 1;
        if(cnt = N) then
                signal(barrier2)
        else
                signal(barrier1);

        wait(barrier2);
        cnt := cnt - 1;
        if(cnt = 0) then
                signal(barrier1)
        else
                signal(barrier2);
end;

begin
        init(barrier1, 1);
        init(barrier2, 0);
        cnt := 0;

        cobegin
                Pass(0);
                Pass(1);
                ...;
        coend
end.
```

Задатак 10. **The H2O problem**

Постоје два типа атома, водоник и кисеоник, који долазе до баријере (*The H2O problem*). Да би се формирао молекул воде потребно је да се на баријери у истом тренутку нађу два атома водоника и један атом кисеоника. Уколико атом кисеоника дође до баријере на којој не чекају два атома водоника онда он чека да се они сакупе. Уколико атом водоника дође до баријере на којој се не налазе један кисеоник и један водоник он чека на њих. Баријеру треба да напусте два атома водоника и један атом кисеоника. Користећи семафоре написати програм који симулира понашање водоника и кисеоника.

Решење:

```
program H2O(input, output);
var
    hydroSem : Semaphore;
    hydroSem2 : Semaphore;
    hydroMutex : Semaphore;
    oxySem : Semaphore;
    oxyMutex : Semaphore;
    count : integer;
procedure Oxygen(i : integer);
begin
    wait (oxyMutex);
    signal (hydroSem);
    signal (hydroSem);
    wait (oxySem);
    bond (i);
    signal (oxyMutex);
end;
procedure Hydrogen(i : integer);
begin
    wait (hydroSem);
    wait (hydroMutex);
    count := count + 1;
    if (count = 2) then
    begin
        signal (oxySem);
        signal (hydroSem2);
        signal (hydroSem2);
        count := 0
    end;
    signal (hydroMutex);
    wait (hydroSem2);
    bond (i)
end;
begin
    init(hydroSem, 0);
    init(hydroSem2, 0);
    init(hydroMutex, 1);
    init(oxySem, 0);
    init(oxyMutex, 1);
    count := 0;
    cobegin
        Oxygen(1);
        Oxygen(2);
        ...
        Hydrogen(1);
        Hydrogen(2);
        ...
    coend;
end.
```

Задатак 11. **The child care problem**

У неком забавишту постоји правило које каже да се на свака три детета мора наћи барем једна васпитачица. Родитељ доводи једно или више деце у забавиште. Уколико има места оставља их, уколико не одводи их. Васпитачица сме да напусти забавиште само уколико то не нарушава правило. Написати процедуре, користећи семафоре, за родитеље који доводе и одводе децу и васпитачици и иницијализовати почетне услове.

Решење:

```
program ChildCare;
const C = 3;

var    numChild : integer;
    numNann : integer;
    numWaiting : integer;
    mutex : semaphore;
    confirm : semaphore;
    toLeave : semaphore;
function bringUpChildren ( num : integer) : boolean;
begin
    wait(mutex);
    if((numChild + num) <= C * numNann) then
    begin
        numChild := numChild + num;
        bringUpChildren := true;
    end
    else
        bringUpChildren := false;
    signal(mutex);
end;

procedure bringBackChildren ( num : integer);
var out, i : integer;
begin
    wait(mutex);
    numChild := numChild - num;
    out : = numNann - (numChild + C - 1) / C;
    if(out > numWaiting) then out := numWaiting;
    for i := 1 to out do
    begin
        signal(toLeave);
        wait(confirm);
    end
    signal(mutex);
end;

procedure nannEnter();
begin
    wait(mutex);
    numNann := numNann + 1;
    if(numwaiting > 0) then
    begin
        signal(toLeave);
        wait(confirm);
    end
    signal(mutex);
end;

procedure nannExit();
begin
    wait(mutex);
    if ((numChild) <= C * ( numNann - 1)) then
    begin
        numNann := numNann - 1;
        signal(mutex);
```

```
    end
    else
    begin
        numWaiting := numWaiting + 1
        signal(mutex);
        wait(toLeave);
        numNann := numNann - 1;
        numWaiting := numWaiting - 1
        signal(confirm);
    end
end;

begin
    numChild := 0;
    numNann := 0;
    numwaiting := 0;
    init(mutex, 1);
    init(confirm, 0);
    init(toLeave, 0);
    cobegin
    ...
    coend;
end.
```

Региони

Задатак 1. **Међусобно искључивање**

Дат је упоредни програм на проширеном Pascal-у:

```
program timedependent;
var x: shared integer;
begin
    x := 1;
    cobegin
        x := x + 1;
        x := x + 3;
    coend;
    writeln ('x = ', x);
end.
```

а) Навести све могуће излазне резултате наведеног програма.

б) Употребом критичних области отклонити временску зависност у програму.

Решење:

б)

```
program timedependent;
var x: shared integer;
begin
    x := 1
    cobegin
        region x do x := x + 1;
        region x do x := x + 3;
    coend;
    writeln ('x = ', x);
end.
```

Задатак 2. **Условна синхронизација**

Дат је упоредни програм на проширеном Pascal-у:

```
program graph;
...
end. {graph}
```

Жељени излаз програма је низ парова облика:

(0,0) (1,1) (2,4) ... (n,n^2)

Отклонити временску зависност у датом програму употребом условних критичних области облика

```
region <deljena promenljiva> do
            begin <akcija1>; await <uslov>; <akcija2> end
```

Решење:

```
program graph;
const n = ...;
type point = record
                x, y:integer;
                full: boolean
             end;
var p: shared point;

procedure makepoints;
var i: integer;
begin
    for i := 1 to n do
        region p do
            begin
                await(not p.full);
                p.x := i;
                p.y := i*i;
                p.full := true
```

```
        end
end;

procedure printpoints;
var i: integer;
begin
    for i := 0 to n do
        region p do
            begin
                await(p.full);
                write('(',p.x,',',p.y,')');
                p.full := false
            end
end;

begin
    p.x := 0; p.y := 0; p.full := true;
    cobegin
        makepoints;
        printpoints;
    coend
end.
```

Задатак 3. **One lane bridge problem**

Аутомобили који долазе са севера и југа морају да пређу реку преко моста. На мосту, на жалост, постоји само једна возна трака. Значи, у било ком тренутку мостом може да прође један или више аутомобила који долазе из истог смера (али не и из супротног смера). Написати алгоритам за аутомобил са севера и аутомобил са југа који долазе на мост, прелазе га и напуштају га са друге стране.

Решење:

```
var most: shared record juzni, severni: integer end
                        "u pocetku oba su nula"
"automobil sa juga"
begin
    region most do
        begin
            await severni = 0;
            juzni := juzni + 1;
        end
    predji most;
    region most do
        juzni := juzni - 1;
end
```

Задатак 4. **One lane bridge problem**

Усавршити решење претходног задатка тако да се смер саобраћаја мења сваки пут након што га пређе 10 аутомобила из истог смера, ако су за то време један или више аутомобила чекали да га пређу из супротног смера.

Решење:

```
type smer =     record;
                    cekaju, prelaze, ispred: integer;
                end
                    "u pocetku svi su nula"
var most:   shared record
                juzni, severni: smer;
            end
"automobil sa juga"
begin
    region most do
        with juzni do
        begin
            cekaju := cekaju + 1;
            await severni.prelaze = 0 AND ispred < 10;
```

```
            cekaju := cekaju - 1;
            prelaze := prelaze + 1;
            if (severni.cekaju > 0) then
                ispred := ispred + 1;
        end;
    predji most;
    region most do
        with juzni do
        begin
            prelaze := prelaze - 1;
            if (prelaze = 0) then
                severni.ispred := 0;
        end
end
```

Задатак 5. **Dining philosophers problem**

Пет филозофа седи око стола. Сваки филозоф наизменично једе и размишља. Испред сваког филозофа је тањир шпагета. Када филозоф пожели да једе, он узима две виљушке које се налазе уз његов тањир. На столу, међутим, има само пет виљушки. Значи, филозоф може да једе само када ниједан од његових суседа не једе. Написати алгоритам за филозофа и виљушку (0<=i<=4). (Упутство: спречити блокирање).

Решење:

```
var    viljuske: shared array [0..4] of 0..2;
procedure filozof (i:0..4);
var    levi, desni: 0..4;
begin
    levi := (i-1) mod 5;
    desni := (i+1) mod 5;
    repeat
        razmisljaj;
        region viljuske do
        begin
                await viljuske [i] = 2;
                viljuske [levi] := viljuske [levi] - 1;
                viljuske [desni] := viljuske [desni] - 1;
        end;
        jedi;
        region viljuske do
        begin
                viljuske [levi] := viljuske [levi] + 1;
                viljuske [desni] := viljuske [desni] + 1;
        end;
    forever;
end;
```

Задатак 6. **Dining philosophers problem**

Коментаришите следеће решење проблема филозофа који ручају.

```
var viljuska: array [0..4] of shared boolean;
"filozof i"
repeat
    razmisljaj;
    region viljuska [i] do
        region viljuska [(i+1) mod 5] do jedi;
forever
```

Задатак 7. **Dining philosophers problem**

Коментаришите следеће решење проблема филозофа који ручају.

```
var    razmisljanje: shared array [0..4] of boolean;
       "u pocetku sve je tacno"
"filozof i"
repeat
    razmisljaj;
```

```
    region razmisljanje do
    begin
        await razmisljanje [(i-1) mod 5] AND
            razmisljanje [(i+1) mod 5];
        razmisljanje [i] := false;
    end;
    jedi;
    region razmisljanje do razmisljanje [i] := true;
forever;
```

Задатак 8. **Dining philosophers problem**

Коментаришите следеће решење проблема филозофа који ручају: гладни филозоф прво узима виљушку са своје леве стране; ако је виљушка са његове десне стране слободна, узима је и почиње да једе; у супротном, одлаже виљушку са леве стране и понавља циклус.

Задатак 9. **Readers – Writers problem**

Група упоредих процеса који приступају заједничком средству састоји се од читалаца Ri, i = 1,... ,m, и писаца Wj, j = 1,...,n.

```
v: shared record r,w: integer end;
v1: shared integer;
begin
    v.r := 0; v.w := 0;
    cobegin R1; ... Rm; W1; ... Wn coend
end
```

Исправна контрола приступа мора обезбедити међусобно искључивање процеса према уобичајеном правилу за читаоце и писце, и спречити узајамно блокирање (deadlock) и 'изгладњивање' (starvation). Предложене су различите варијанте решења. У датој таблици свакој варијанти одговара по једна колона, и у њу треба за свако од наведених тврђења уписати Т(ачно) или Н(етачно).

	а	б	в
Међусобно искључење је осигурано.	Н	Т	Н
Могуће је узајамно блокирање читалаца и писаца.	Н	Н	Н
Могуће је узајамно блокирање писаца (при r=0).	Н	Т	Н
Могуће је 'изгладњивање' читалаца.	Т	Т	Н
Могуће је 'изгладњивање' писаца.	Н	Н	Н

а)

```
"Ri"
repeat
    region v do
    begin
        await (w = 0);
        r := r + 1
    end;
    read;
    region v do r := r - 1;
    nekritične operacije;
forever
"Wi"
repeat
    region v do
    begin
        w := w + 1;
        await (r = 0)
    end;
    write;
    region v do w := w - 1;
    nekritične operacije;
forever
```

б)

```
"Ri"
repeat
    region v do
    begin
            await (w = 0);
            r := r + 1
    end;
    read;
    region v do r := r - 1;
    nekritične operacije;
forever
"Wi"
repeat
    region v do
    begin
            w := w + 1;
            await ((r = 0) and (w = 1))
    end;
    write;
    region v do w := w - 1;
    nekritične operacije;
forever
```

в)

```
var v: shared record r,w: integer; rturn:boolean end;
    begin
    v.r := 0; v.w := 0; v.rturn := false;
    cobegin R1; ... Rm; W1; ... Wn coend
end;
"Ri"
repeat
    region v do
    begin
        if (rturn) then
            begin
                r := r + 1;
                await (w = 0)
            end
        else
            begin
                await (w = 0);
                r := r + 1
            end;
    end;
    read;
    region v do
    begin
            r := r - 1;
            rturn := false
    end;
    nekritične operacije;
forever
"Wi"
repeat
    region v do
    begin
        if (rturn) then
            begin
                await (r = 0);
                w := w + 1
            end
        else
            begin
                w := w + 1;
                await (r = 0)
            end;
```

```
    end;
    write;
    region v do
    begin
            w := w - 1;
            rturn := true
    end;
    nekritične operacije;
forever
```

Задатак 10. **Cigarette Smokers' problem**

Користећи условне критичне регионе написати програм који решава проблем и симулира систем "нервозних пушача" (*Cigarette Smokers' problem*). Постоји један агент и три нервозна пушача. Агент поседује резерве три неопходна предмета за лечење нервозе: папир, дуван и шибице. Један од пушача има бесконачне залихе папира, други – дувана, а трећи - шибица. Агент почиње тако што два различита предмета ставља на сто, један по један. Пушач, коме баш та два предмета фале, узима их, завија и пали цигарету и ужива. Након тога обавештава агента да је завршио, а агент онда ставља два нова предмета на сто, итд.

Решење:

```
program CigaretteSmokers(input, output);
type table = record
            paper, tobacco, matches : boolean;
            ok : boolean;
        end;
var p: shared table;

procedure Agent;
var n : integer;
begin
    while (true) do
    begin
        n := RANDOM(0, 2);
        region p do
        begin
            case n of
            0: begin
                p.paper := false;
                p.tobacco := true;
                p.matches := true;
            end;
            1: begin
                p.paper := true;
                p.tobacco := false;
                p.matches := true;
            end;
            2: begin
                p.paper := true;
                p.tobacco := true;
                p.matches := false;
            end;
            else ;
        end;
        region p do
            await(p.ok);
            p.ok := false;
        end;
    end;
end;

procedure Smoker_with_Matches;
```

```
begin
    while (true) do
    begin
        region p do
        begin
            await(p.paper and p.tobacco);
            p.paper := false;
            p.tobacco := false;
        end;
        enjoy;
        region p do
            p.ok := true;
    end;
end;
procedure Smoker_with_Tobacco;
begin
    while (true) do
    begin
        region p do
        begin
            await(p.paper and p.matches);
            p.paper := false;
            p.matches := false;
        end;
        enjoy;
        region p do
            p.ok := true;
    end;
end;

procedure Smoker_with_Paper;
begin
    while (true) do
    begin
        region p do
        begin
            await(p.matches and p.tobacco);
            p.matches := false;
            p.tobacco := false;
        end;
        enjoy;
        region p do
            p.ok := true;
    end;
end;

begin
    p.paper := false;
    p.tobacco := false;
    p.matches := false;
    p.ok := false;
    cobegin
        Agent;
        Smoker_with_Paper;
        Smoker_with_Tobacco;
        Smoker_with_Matches;
    coend;
end.
```

Монитори

Задатак 1. **Readers – Writers problem**

Реализовати проблем писаца и читача помоћу монитора. Користити signal and wait дисциплину.

Решење:

```
Readers_Writers: monitor;
    var:   readCount: integer;
           busy: boolean;
           OKtoread, OKtowrite: condition;
    procedure startread;
        begin
            if (busy or OKtowrite.queue) then OKtoread.wait;
            readCount := readCount + 1;
            OKtoread.signal
        end;
    procedure endread;
        begin
            readCount:= readCount - 1;
            if (readCount = 0) then OKtowrite.signal
        end;
    procedure startwrite;
        begin
            if (readCount <> 0 or busy) then OKtowrite.wait;
            busy := true
        end;
    procedure endwrite;
        begin
            busy := false;
            if (OKtoread.queue)   then OKtoread.signal
                                  else OKtowrite.signal
        end;
    begin
        readCount:= 0;
        busy := false
    end.
```

Задатак 2. **Readers – Writers problem**

Решити проблем читалаца и писаца користећи мониторе који имају дисциплину signal and continue. Решење треба да обезбеди да процес који је пре стигао пре и започне операцију читања односно уписа.

Решење:

```
Rreaders_Writers: monitor;
var
    ID, ok_to_work, readCount: integer;
    OKtoWork: condition;

    procedure startread;
    var PID : integer;
    begin
        PID := ID;
        ID := ID + 1;
        while (PID <> ok_to_work) do OKtoWork.wait;
        readCount:= readCount + 1;
        ok_to_work := ok_to_work + 1;
        OKtoWork.signalAll;
    end;

    procedure endread;
    begin
```

```
        readers_num := readers_num - 1;
        if(readers_num = 0) then OKtoWork.signalAll;
    end;
    procedure startwrite;
    var PID : integer;
    begin
        PID := ID;
        ID := ID + 1;
        while ((PID <> ok_to_work) or (readCount <> 0)) do OKtoWork.wait;
    end;
    procedure endwrite;
    begin
        ok_to_work := ok_to_work + 1;
        OKtoWork.signalAll;
    end;
    begin
        ID := 0;
        ok_to_work := 0;
        readCount := 0;
    end.
```

Задатак 3. **Прихватник**

Реализујте монитор за FIFO прихватник који функционише на следећи начин: читање се обавља у бајтима само када постоји бар један бајт у прихватнику; упис се обавља у речима (по два бајта истовремено), када постоје бар два празна бајта; повремено се брише садржај целокупног прихватника на позив мониторске процедуре. Користити signal and wait дисциплину.

Решење:

```
program BufferCancel;
const  N = ...;

Buffer : monitor;
        var
                slots : array [0 .. N-1] of byte;
                head, tail : 0..N-1;
                size : 0..N;
                not_full, not_empty : condition;

        procedure put(first, second : byte);
        begin
                if(size > N - 2) then not_full.wait;
                slots[tail] := first;
                tail := (tail + 1) mod N;
                slots[tail] := second;
                tail := (tail + 1) mod N;
                size := size + 2;
                if((size > 0) and (not_empty.queue)) then not_empty.signal;
                if((size > 0) and (not_empty.queue)) then not_empty.signal;
        end;

        procedure get(var data: byte);
        begin
                if(size = 0) then not_empty.wait;
                data := slots[head];
                size := size - 1;
                head := (head + 1) mod N;
                if((size <= N - 2) and (not_full.queue)) then not_full.signal
        end;

        procedure cancel;
        begin
                size := 0; head := 0; tail := 0;
                while((size <= N - 2) and (not_full.queue)) do not_full.signal
```

```
        end;

begin
        size := 0; head := 0; tail := 0
end;
```

Задатак 4. **Тајмер**

Реализовати монитор који омогућава програму који га позива да чека n јединица времена. Користити signal and wait дисциплину.

Решење:

```
Alarmclock: monitor;
    var:    now: integer;
            wakeup: condition;
    procedure wakeme (n: integer);
        var alarmsetting: integer;
        begin
            alarmsetting := now + n;
            while (now < alarmsetting) do
                wakeup.wait (alarmsetting);
            wakeup.signal;
        end;
    procedure tick;
        begin
            now := now + 1;
            wakeup.signal
        end;
    begin
        now := 0
    end.
```

Задатак 5. **Тајмер**

Реализовати монитор који омогућава програму који га позива да чека n јединица времена. Користити signal and wait дисциплину, трудити се да буди што мање процеса.

Решење:

```
Alarmclock: monitor;
    var:    now: integer;
            wakeup: condition;
    procedure wakeme (n: integer);
        var alarmsetting: integer;
        begin
            alarmsetting := now + n;
            wakeup.wait (alarmsetting);
        end;
    procedure tick;
        begin
            now := now + 1;
            while (NOT wakeup.empty AND wakeup.minrank <= now) do
                wakeup.signal
        end;
    begin
        now := 0
    end.
```

Задатак 6. **Producer – Consumer/Bounded Buffer problem**

Решити проблем Producer – Consumer користећи мониторе који имају signal and wait дисциплину.

```
BoundedBuffer: monitor;
    var:
        buffer: array [0..k-1] of items;
        nextin, nextout, count: integer;
        notfull, notempty: condition;
```

```
    procedure Append(v: items);
        begin
            if (count = k) then notfull.wait;
            buffer[nextin] := v;
            nextin = (nextin + 1) mod k;
            count :=count + 1;
            notempty.signal;
        end;

    procedure Take(var v: items):
        begin
            if (count = 0) then notempty.wait;
            v := buffer[nextout];
            nextout := (nextout + 1) mod k;
            count :=count - 1;
            notfull.signal;
        end;

    begin
        nextin := 0;
        nextout := 0;
        count := 0;
    end.
```

Задатак 7. **Dining philosophers problem**

Користити signal and wait дисциплину.

```
program Dining_Philosophers;
const num_phils = 5;
    num_philsminusone = 4;
monitor data;
    var
        can_eat : array [0..num_philsminusone] of condition;
        state : array [0..num_philsminusone] of integer; {(thinking=0, hungry=1, eating=2) ;}
        index : integer;

    procedure test (k: integer);
    begin
        if ((state[(k+4) mod 5] <> 2) and (state[k] = 1) and (state[(k+1) mod 5] <> 2)) then
        begin
            state[k] := 2;
            can_eat[k].signal;
        end;
    end;
    procedure pickup(ID : integer);
    begin
        state[ID] := 1;
        test(ID);
        if (state[ID] <> 2) then can_eat[ID].wait;
    end;

    procedure putdown (ID : integer);
    begin
        state[ID] := 0;
        test((ID+4) mod 5);
        test((ID+1) mod 5);
    end;
    begin
        for index := 0 to 4 do state[index] := 0;
    end;
procedure philosopher(id : integer);
procedure think ;
begin
```

```
end;
procedure eat;
begin

end;
    begin
        while (true) do
        begin
            think;
            pickup(id);
            eat;
            putdown(id);
        end;
    end;
begin
    cobegin
        philosopher(0);
        philosopher(1);
        philosopher(2);
        philosopher(3);
        philosopher(4);
    coend;
end.
```

ПРОСЛЕЂИВАЊЕ ПОРУКА

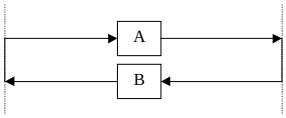
Задатак 1. **Синхрона, асинхрона и условна комуникација**

Комуникациони сервиси у једном дистрибуираном систему реализовани су на бази сандучића типа mbx за пренос порука типа msg. Претпоставићемо да тип msg обухвата целе бројеве и специјални симбол ack за потврђивање пријема. Основне операције на сандучићима су следеће:

mbx_put(m: msg, box: mbx) смешта поруку m у сандуче box

mbx_get(var m: msg, box: mbx, t: time, var status: boolean) узима прву поруку из сандучета box и њену вредност додељује променљивој m, постављајући статус на true; ако је сандуче празно током интервала t, статус постаје **false**, а вредност m је недефинисана. Време t је у опсегу 0..maxtime или је INF.

Размотримо једноставан систем који садржи само два процеса, S и R. Процеси комуницирају преко сандучића A и B као на слици.



Реализовати следеће интеракције између процеса:

а) S асинхроно шаље целобројну вредност i, а R извршава обичан пријем (basic receive).

б) S асинхроно шаље целобројну вредност i, а R извршава условни пријем (conditional receive).

в) S асинхроно шаље целобројну вредност i, а R извршава временски условљен пријем (receive on timeout) са интервалом d.

г) S синхроно шаље целобројну вредност i, а R извршава обичан пријем.

д) У бидирекционој трансакцији типа 'захтев-одговор' (request-reply), S шаље целобројну вредност i и добијени резултат j додељује променљивој x; у случају да трансакција не успе током интервала d, x добија вредност 0. R обрађује захтев тако што за дати аргумент и враћа вредност функције f(i).

Решење:

а) Асинхроно слање, обичан пријем:

```
Procedure send(i:integer);
var m: msg;
begin
    m := i;
    mbx_put(m,A);
end;
```

```
procedure receive(var i:integer);
var m: msg; st: boolean;
begin
    mbx_get(m,A,INF,st);
    i := m;
end;
```

б) Асинхроно слање, условни пријем:

```
Procedure send(i:integer);
var m: msg;
begin
    m := i;
    mbx_put(m,A);
end;
```

```
function receive(var i: integer):boolean
var m: msg; st: boolean;
begin
    mbx_get(m,A,0,st);
    if st then i := m;
    receive := st;
end;
```

в) Асинхроно слање, временски условљен пријем:

```
procedure send(i:integer);
var m: msg;
begin
    m := i;
    mbx_put(m,A);
end;
```

```
function receive(var i:integer,d:time):
boolean;
var m: msg; st: boolean;
begin
    mbx_get(m,A,d,st);
    if st then i := m;
    receive := st;
end;
```

г) Синхроно слање, обичан пријем:

```
function send(i: integer):boolean;
var m: msg; st: boolean;
begin
    m := i;
    mbx_put(m,A);
    mbx_get(m,B,INF,st);
    send := (m = ack);
end;
```

```
procedure receive(var i:integer)
var m: msg; st: boolean;
begin
    mbx_get(m,A,INF,st);
    i := m;
    m := ack;
    mbx_put(m,B);
end;
```

д) Бидирекциона трансакција типа 'захтев-одговор':

```
procedure rq(i:integer; var x:integer,d:time);
var m: msg; st: boolean;
begin
    m := i;
    mbx_put(m,A);
    mbx_get(m,B,d,st);
    if st then x := m
    else x := 0;
end;
```

```
procedure reply;
var m: msg; i: integer; st: boolean;
begin
    mbx_get(m,A,INF,st);
    i := m;
    m := f(i);
    mbx_put(m,B);
end;
```

Задатак 2. **Dining Philosophers Problem**

Користећи размену порука написати програм који решава проблем филозофа који ручавају (*The Dining Philosophers*). Написати решење код кога филозофи комуницирају само са заједничким столом (централизовано решење).

Решење:

```
program DiningPhilosophers;
const N = 5;

var    table : mbx;
    Phil : array [0..N-1] of mbx;

procedure Coordinator;
    var
        state : array [0.. N-1] of integer; {(thinking = 0, hungry = 1, eating = 2) ;}
        index : integer;
        m : msg;
        status : boolean;
    procedure test (k: integer);
    begin
        if ((state[(k + N-1) mod N] <> 2)
            and (state[k] = 1)
            and (state[(k+1) mod N] <> 2)) then
        begin
            state[k] := 2;
            mbx_put(m, Phil[k]);
        end;
    end;
    procedure pickup (ID : integer);
    begin
        state[ID] := 1;
        test(ID);
    end;

    procedure putdown (ID : integer);
    begin
        state[ID] := 0;
        test((ID + N-1) mod N);
        test((ID + 1) mod N);
        mbx_put(m, Phil[ID]);
    end;

begin
    for index := 0 to N-1 do state[index] := 0;
    while (true) do
```

```
    begin
        mbx_get(m, table, INF, status);
        if(m.status = 1) then pickup(m.id)
        else if(m.status = 0) then putdown (m.id);

    end
end;


procedure Philosopher(i : integer);

var     m, n : msg;
    status : boolean;
procedure think; begin ... end;
procedure eat; begin ... end;

begin
    while (true) do
    begin
        think;
        m.id := i;
        m.status := 1;
        mbx_put(m, table);
        mbx_get(n, Phil[i], INF, status);
        eat;
        m.id := i;
        m.status := 0;
        mbx_put(m, table);
        mbx_get(n, Phil[i], INF, status)
    end
end;
begin
    begin
        Philosopher(0);
        Philosopher(1);
        Philosopher(2);
        Philosopher(3);
        Philosopher(4);
        Coordinator
    end
end.
```

Задатак 3. **Broadcast**

Постоји повезан граф који се састоји из n чворова. Чворови могу да комуницирају само са суседним чворовима. Користећи сандучиће написати програм који поруку коју шаље један чвор прослеђује свим осталим чворовима у графу. Сваки чвор има информације само о својим суседима.

Решење:

```
program Broadcast;
const n = ...;
var     probe : array [1..n] of mbx;

procedure Node(p : 1..n);
var     links : array [1..n] of boolean; //neighbors of node p;
    m : msg;
    num : integer; //number of neighbors;
    q : integer;
    st : boolean;
begin
    init(p);
    mbx_get(m, probe[p], INF, st) ;
    //send m to all neighbors
    for q := 1 to n do
        if(links[q]) then mbx_put(m, probe[q]);
```

```
    //receive num-1 redundant copies of m
    for q = 1 to num-1 do
        mbx_get(m, probe[p], INF, st) ;
end;

procedure Initiator; //executed on source node S
var    m : msg; //message to broadcast;
    S : integer;
begin
    mbx_put(m, probe[S]);
end;
```

Задатак 4. **Broadcast**

Постоји повезан граф који се састоји из n чворова. Чворови могу да комуницирају само са суседним чворовима. Користећи сандучиће написати програм који поруку коју шаље један чвор прослеђује свим осталим чворовима у графу. У овом решењу претпоставите да почетни чвор има информације о комплетној топологији графа.

Решење:

```
program BroadcastTree;
const n = ...;
type graph = array [1..n, 1..n] of boolean;
type msg = record
        data : message;
        spanningTree : graph;
    end;
var    probe : array [1..n] of mbx;

procedure Node(p : 1..n);
var    t : graph;
    m : msg;
    q : integer;
    st : boolean;
begin
    init(p);
    mbx_get(m, probe[p], INF, st) ;
    t := m.spanningTree;
    //send m to all children
    for q := 1 to n do
        if(t[p, q]) then mbx_put(m, probe[q]); //q is a child of p in t
end;

procedure Initiator; //executed on source node S
var    m : msg; //message to broadcast;
    S : integer;
    topology : graph; //network topology;
    t : graph; //spanning tree of topology;
begin
    initTopology(topology, t);
    m.spanningTree := t;
    mbx_put(m, probe[S]);
end;
```

Задатак 5. **Readers – Writers Problem**

Решити проблем читалаца и писаца (*Readers – Writers Problem*) користећи поштанске сандучиће. Дозвољено је да само један процес чита поруке из једног сандучета.

Решење:

```
program ReadersWriters;

const STARTREAD = 0;
const STARTWRITE = 1;
```

```
var    operationStart : mbx;
    operationEnd : mbx;
    confirm : array [0..N-1] of mbx;

procedure Coordinator;
    var
        numReaders: integer;
        numWriters: integer;
        m, n : msg;
 status : boolean;

begin
    while (true) do
    begin
        mbx_get(m, operationStart, INF, status);
        if(m.operation = STARTREAD) then
        begin
            numReaders := numReaders + 1;
            mbx_put(m, confirm[m.id]);
        end
        else if(m.operation = STARTWRITE) then
        begin
            while(numReaders > 0) do
            begin
                mbx_get(n, operationEnd, INF, status);
                numReaders := numReaders - 1;
            end;
            mbx_put(m, confirm[m.id]);
            mbx_get(m, operationEnd, INF, status);
        end;
        status := true;
        while(status) do
        begin
            mbx_get(m, operationEnd, 0, status);
            if(status) then numReaders := numReaders - 1;
        end;
    end
end;


procedure Reader(i : integer);

var    m: msg;
    status : boolean;
procedure read; begin end;

begin
    while (true) do
    begin
        m.id := i;
        m.operation := STARTREAD;
        mbx_put(m, operationStart);
        mbx_get(m, confirm[i], INF, status);
        read;
        m.id := i;
        mbx_put(m, operationEnd);
    end
end;

procedure Writer(i : integer);

var    m: msg;
    status : boolean;
procedure write; begin end;
```

```
begin
    while (true) do
    begin
        m.id := i;
        m.operation := STARTWRITE;
        mbx_put(m, operationStart);
        mbx_get(m, confirm[i], INF, status);
        write;
        m.id := i;
        mbx_put(m, operationEnd);
    end
end;

begin
    begin
        Reader(0);
        Reader(1);
        Reader(2);
        Writer(3);
        Writer(4);
        Coordinator
    end
end.
```

Задатак 6. **Game of Life**

Постоји матрица димензија nxn таква да свака њена ћелија представља један организама који може да буде жив или мртав. Организми могу да комуницирају само са својим суседима (горе, доле, лево, десно и укосо). Организми у средини ће имати 8 суседа, док ће они у угловима имати само 3. Правила која важе за сваки организам су следећа:

Жив организам који има мање од два жива суседа умире од усамљености
Жив организам који има више од три жива суседа умире од пренатрпаности
Жив организам са два или три жива суседа преживљава и формира следећу генерацију
Мртав организам са три жива суседа оживљава

Користећи сандучиће написати програм који симулира организам.

Решење:

```
program GameOfLIfe;
const numGenerations = ...;
    N = ...;
type msg = record
        status : boolean;
        i, j, index : integer;
end;
var     box : array [0..N-1, 0..N-1] of mbx;

function xStart(i : integer) : integer;
begin
    i := i - 1;
    if(i < 0) then xStart := 0
    else xStart := i;
end;
function yStart(i : integer) : integer;
begin
    i := i - 1;
    if(i < 0) then yStart := 0
    else yStart := i;
end;
function xEnd(i : integer) : integer;
begin
    i := i + 1;
    if(i >= N) then xEnd := N-1
    else xEnd := i;
end;
```

```
function yEnd(i : integer) : integer;
begin
    i := i + 1;
    if ( i >= N) then yEnd := N-1
    else yEnd := i;
end;
function numOfNeighbours(i, j : integer) : integer;
begin
    numOfNeighbours := (xEnd(i) - xStart(i) + 1)*(yEnd(i) - yStart(i) + 1) - 1;
end;
procedure calculateState(i, j, k : integer; neighbours : array[0..1, 0..7] of msg; var status : boolean);
var index, num : integer;
begin
    num := 0;
    for index := 0 to numOfNeighbours(i, j) -1 do
    begin
        if neighbours[k mod 2, index].status then num := num + 1;
    end;
    if (status and ((num < 2 ) or (num > 3))) then status := false
    else if ((not status) and (num = 3)) then status := true;
end;
procedure Node(i : integer; j : integer);
var    p, q, k : integer;
    status, st : boolean;
    neighbours : array[0..1, 0..7] of msg;
    num : array[0..1] of integer;
    m : msg;
begin
    num [0] := 0;     num [1] := 0;
    for k := 1 to numGenerations do
    begin
        m.status := status;
        m.i := i;
        m.j := j;
        m.index := k;

        for p := xStart(i) to xEnd(i) do
            for q := yStart(j) to yEnd(j) do
                begin
                    if((p <> i) or (q <> j)) then
                        mbx_put(m, box[p, q]);
                end;

        while (num [k mod 2] < numOfNeighbours(i, j)) do
        begin
            mbx_get(m, box[i, j], INF, st);
            neighbours[m.index mod 2, num[m.index mod 2]] := m;
            num[m.index mod 2] := num[m.index mod 2] + 1;
        end;
        num[k mod 2] := 0;
        calculateState(i, j, k, neighbours, status);
    end;
end;
```

CSP

Задатак 1. **Семафор**

Пројектовати бинарни семафор користећи програмски језик CSP.
Решење:

```
s : integer; s : = 0;
*[
    s > 0; (i:1..100) X(i)?wait()→
                    s := s - 1
    []
    (i:1..100) X(i)?signal()→
                    S : = s + 1
]
```

Задатак 2. **Прослеђивање текста**

Написати програм за процес X који прослеђује знаке добијене од процеса *west* процесу *east*.
X::*[c: **character**; west?c → east!c]

Задатак 3. **Прослеђивање текста**

Модификовати претходно решење тако да свака две суседне звездице "**" замени са "^". Сматрати да последњи знак није звездица.
Решење:

```
X::* [  c: character;
        west?c →
            [   c <> '*' → east!c;
            []
                c = '*' → west?c;
                [   c <> '*' → east!'*'; east!c;
                []
                    c = '*' → east!'^'
                ]
            ]
]
```

Задатак 4. **Читање картица**

Написати програм који добија картице од процеса *cardfile* и прослеђује процесу *X* низ знакова које садржи. На крају треба додати још један бланко знак.
Решење:

```
*[  cardimage: (1..80)character;
    cardfile?cardimage →
        i:integer; i := 1;
        *[  i ≤ 80 → X!cardimage(i); i := i + 1]
        X!' '
]
```

Задатак 5. **Обрада текста**

Написати програм који добија низ знакова од процеса *X* и штампа их на штампачу по 125 знакова у реду. Последњу линију допунити бланковима по потреби.
Решење:

```
lineimage: (1..125)character;
i: integer; i := 1;
lineimage: (1..125)character;
i: integer; i := 1;
*[  c:character; X?c →
```

```
        lineimage(i) := c;
        [   i ≤ 124 → i := i + 1
        []
            i = 125 → lineprinter!lineimage; i := 1
        ]
];
[   i = 1 → skip
[]
    i > 1 → *[i ≤ 125 → lineimage(i) := ' '; i := i + 1]
            lineprinter!lineimage
]
```

Задатак 6. **Producer - Consumer**

Решење:

```
[Producer (i:1..5)::PRODUCER  || Consumer (i:1..6)::CONSUMER
|| Buffer::BUFFER]

BUFFER:: [ buffer: (0..9) portion;
    in, out : integer; in := 0; out := 0;
    *[
        in<out + 10; (i: 1..5) Producer (i)?buffer(in mod 10) →
            in:=in + 1
    []
        out < in; (i: 1..6) Consumer (i)?more() →[
            Consumer (i)!buffer(out mod 10) -> out := out + 1]
    ]
]

PRODUCER:: *[ true →[data : portion;
        PRODUCE;
        Buffer!data
    ]
]

CONSUMER:: *[ true →[data : portion;
    Buffer!more();
        Buffer?data;
        CONSUME
    ]
]
```

Задатак 7. **Дељење бројева**

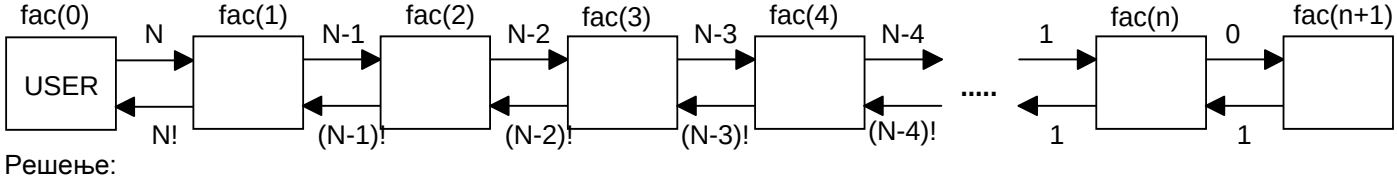
Конструисати процес на CSP-у који представља функцијски потпрограм и који прихвата позитиван делитељ и делилац, и враћа њихов целобројни количник и остатак при дељењу.

Решење:

```
 [DIV:: *[  x, y : integer;
          X?(x,y) → quot, rem:integer; quot := 0; rem := x;
                *[rem ≥ y →  rem := rem - y;
                                quot := quot + 1
                ];
                X!(quot, rem)
            ]
|| X:: USER]
```

Задатак 8. **Факторијел**

Израчунати факторијел рекурзивном методом, до одређене границе.



Решење:

```
 [fac (I: 1..LIMIT):
    *[N: integer; fac (i-1)?n →
```

```
        [   n = 0 →       fac (i-1)!1
        []
            n > 0 →  fac (i + 1)!n-1;
                     r: integer; fac(i + 1)?r;
                     fac (i - 1)!(n*r)
        ]
    ]
|| fac (0):: USER
]
```

У USER-у има fac(1)!n.

Задатак 9. **Скуп**

Представити у CSP-у скуп од највише 100 целих бројева као процес S, који прихвата два типа инструкција од позивајућег процеса X:
1) S!*insert (n)*, убацује цео број n у скуп, и
2) S!*has (n);...;S?b*, где је b истинито ако је n у скупу, односно неистинито у супротном случају.
У почетку је скуп празан.

Решење:

```
S:: content: (0..99) integer;
    size: integer; size := 0;
    *[  n: integer;
        X?has (n) → SEARCH; X!(i < size)
    []
        n: integer;
        X?insert (n) → SEARCH;
            [   i < size → skip
            []
                i = size; size < 100 →
                    content (size) := n; size := size + 1
            ]
    ]
SEARCH:
i: integer; i := 0;
*[i < size; content (i) ≠ n → i := i + 1]
```

Задатак 10. **Скуп**

Проширити претходно решење, обезбеђујући брзи метод за сканирање свих елемената скупа, без мењања вредности. Кориснички програм садржи команду типа:

```
S!scan (); more: boolean; more := true;
*[more; x: integer; S?next (x); → ...radi sa x...
[] more; S?noneleft () → more := false
]
```

где S!*scan ()* служи да постави скуп у сканирајући режим.

Решење:

Ова команда ради као ***while*** петља која учитава редом све елементе из скупа, ради нешто са њима и тако док их год има. При раду са елементима скупа није дозвољено да комуницира са скупом на било који начин.
Додаћемо трећу заштићену команду у спољну петљу претходног решења:

```
[] X?scan () →  i: integer; i := 0;
                *[i < size →  X!next (content (i));
                                  i := i + 1];
                X!noneleft ()
```

Задатак 11. **Скуп**

Решити проблем скупа од максимално 100 бројева, са две операције (из претпоследњег задатка) помоћу низа процеса, од којих сваки садржи највише један број. Када процес не садржи ниједан број, на сваки упит о садржини треба да одговори са "**false**".

Решење:

Процес се налази у почетном стању када нема садржај. По првом убацивању елемента у тај

процес, он мења стање у коме сада прима поруке од претходног процеса и, евентуално, шаље податак следећем.
Позивајући процес је S(0) који, када хоће да пошаље податак у скуп, шаље га ка процесу S(1), а када испитује да ли је податак у скупу ради следеће:
S(1)!has (n);...;[(i: 1..100) S(i)?b → **skip**]
Због ефикасности, скуп треба да буде сортиран.

```
S (i:1..100):
*[  n: integer; S (i-1)?has (n) → S(0)!false
[]
    n: integer; S (i-1)?insert (n) →
        *[  m: integer; S (i-1)?has (m) →
                [   m ≤ n → S(0)!(m = n)
                []
                    m > n → S(i + 1)!has (m)
                ]
        []
            m: integer; S (i - 1)?insert (m) →
                [   m < n → S (i + 1)!insert (n); n := m
                []
                    m = n → skip
                []
                    m > n → S (i + 1)!insert (m)
                ]
        ]
]
```

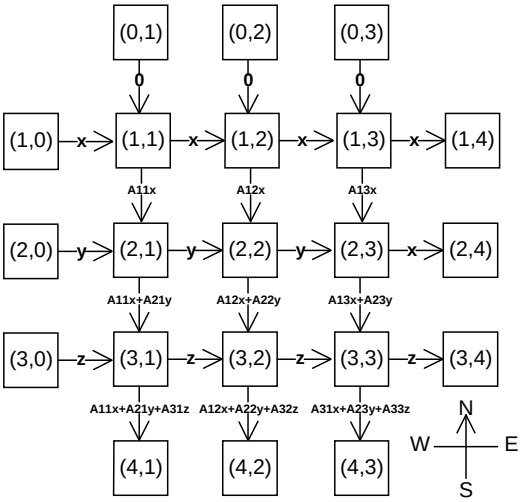
Задатак 12. **Множење матрице**

Дата је квадратна матрица A реда 3. На улаз долазе три низа података, од којих сваки представља по један елемент вектора IN. Три низа података треба да се појаве на излазу, од којих сваки представља по један елемент производа IN×A. После почетног кашњења, резултати треба да се производе истом брзином којом стиже улаз. Матрица A је фиксна.
Решење:
Овако висок ниво паралелизма се може постићи скупом процеса са слике. Сваки не-ивични процес узима компоненту вектора са запада и парцијалну суму са севера. Сваки од њих прослеђује компоненту вектора на исток, а ажурирану суму на југ. Улазни подаци долазе из западних ивичних чворова, а резултати се шаљу у јужне ивичне чворове. Северни чворови служе као извор нула, а источни као рупа без дна.
Решење:
Имамо 5 група чворова:

```
[ M (i:1..3, 0)::WEST   || M (0, j:1..3)::NORTH || M (i:1..3, 4)::EAST || M (4, j:1..3)::SOUTH
|| M (i:1..3, j:1..3)::CENTER]
```

Процеси WEST и SOUTH су кориснички програми. а преостали су:

```
NORTH:: *[true → M (1, j)!0]
EAST::      *[x: real; M (i, 3)?x → skip]
CENTER::    *[x: real; M (i, j-1)?x →
                    M (i, j + 1)!x
                    sum: real;
                    M (i - 1, j)?sum;
                    M (i + 1, j)!(A (i, j)*x + sum)
                ]
```



Задатак 13. **Dining philosophers problem**

Решење:

```
[ Pfil (i:0..4) :: PFILOSOPHER || Fork (i:0..4) :: FORK || Room :: ROOM ]

PFILOSOPHER :: [ left, right : integer; left := i; right := (i +1) mod 5;
    *[true → THINK;
              room!ticket();
              fork(rihgt)!in();
              fork(left)!in();
              EAT;
              fork(left)!out();
              fork(right)!out();
              room!back();
    ]
]

FORK:: [ left, right : integer; left := (i – 1) mod 5; right := i;
    *[  Pfil(left)?in();→ Pfil(left)?out();
        []
        Pfil(right)?in();→ Pfil(right)?out();
    ]
]

ROOM::[ v : integer; v : = 4;
    *[
        v > 0; (i:0..4) Pfil(i)? ticket()→
                  v := v - 1
        []
        (i:0..4) Pfil(i)?out()→
                  v: = v + 1
    ]

]
```

Друго решење

```
[ Pfil (i:0..4) :: PFILOSOPHER || Room :: ROOM ]

PFILOSOPHER :: [ left, right : integer; left := i; right := (i + 1) mod 5;
    *[true → [
            THINK;
            room!getForks();
            EAT;
            room!putForks();
        ]
```

```
    ]
]

ROOM::[ forks : (0..4) integer;
    forks(0) := 2; forks(1) := 2; forks(2) := 2; forks(3) := 2; forks(4) := 2;
    *[
        (i:0..4) forks(i) = 2; Pfil(i)?getForks()→ [
                    forks((i + 1) mod 5) := forks((i + 1) mod 5) - 1;
                    forks((i + 5 - 1) mod 5) := forks((i + 5 - 1) mod 5) - 1;
                ]
        []
        (i:0..4) Pfil(i)?putForks()→ [
                    forks((i + 1) mod 5) := forks((i + 1) mod 5) + 1;
                    forks((i + 5 - 1) mod 5) := forks((i + 5 - 1) mod 5) + 1;
                ]
    ]
]
```

Задатак 14. **Израчунавање интеграла**

Написати програм на језику CSP који израчунава интеграл функције на интервалу XMIN, XMAX у N корака користећи "Торбу Послова" (Bag of Tasks).

Решење:

```
[Node(p : 1..n) :: NODE || Bag :: BAG]

NODE::[ left, right, data : double;
    *[Bag!getTask(); Bag?getData (left, right) →
        CALCULATE;
        Bag!putResult(data);
    ]
]
BAG:: [ Xmin, Xmax, dx, x, F : double; F :=0;
    N, i : integer; i := 0;
    INIT;
    *[  i < N, x < Xmax, (j:1..n)Node(j)?getTask()→ [
            Node(j)!getData (x, x+dx); x := x + dx]
        []
        i < N; (j:1..n)Node(j)?putResult(data) → [
            F := F + data; i := i + 1]
    ]
    STOP;
]
INIT::…
STOP::…
```

CONIC

Задатак 1. **Копирање поруке**

Написати програм на CONIC-у који прихвата улазну поруку и шаље њене копије на сваки од два излазна порта.

Решење:

```
task module splitter;
    entryport    input: string reply signaltype;
    exitport  output1: string reply signaltype;
                  output2: string reply signaltype;
    const waitperiod = 150; {3 seconds}
    var buffer: string;
    begin
        loop
            receive buffer from input reply signal =>
                send buffer to output1
                    wait signal
                    timeout waitperiod => {do nothing}
                end;
                send buffer to output2
                    wait signal
                    timeout waitperiod => {do nothing}
                end;
            end;
end.
```

Задатак 2. **Бидирекциони прстен**

На језику CONIC дефинисан је тип модула

```
TASK MODULE node;
ENTRYPORT   in1: mess;
            in2: mess;
EXITPORT    out1;
            out2;
BEGIN...END.
```

Конкретне инстанце оваквих модула могу се дефинисати и уклањати командама конфигурацијског језика, као што су

```
CREATE A, B: node;
DELETE A;
```

а везе између њих могу се успостављати и раскидати командама облика

```
LINK A.out1 TO B.in1;
UNLINK A.out1 FROM B.in1;
```

У овом задатку модули типа node употребљени су за симулацију чворова локалне мреже која има топологију бидирекционог прстена, као што је нпр. FDDI. Коришћењем наведених команди конфигурацијског језика система CONIC,

а) креирати три чвора A, B, C и повезати их у бидирекциону прстенасту мрежу,

б) специфицирати промену конфигурације под а) насталу креирањем новог чвора D и његовим укључењем у бидирекциони прстен између A и C.

в) специфицирати промену конфигурације која настаје у мрежи под б) приликом отказа чвора B (приказати реконфигурацију повратним превезивањем, као у FDDI, на чворовима суседним чвору B, а затим уклонити чвор B).

г) специфицирати промену конфигурације којом ће мрежа под в) постати бидирекциони прстен са чворовима A, C и D.

Решење:

а)

```
CREATE A, B, C: node;
LINK A.out1 TO B.in1;
     B.out1 TO C.in1;
```

```
        C.out1 TO A.in1;
        A.out2 TO C.in2;
        C.out2 TO B.in2;
        B.out2 TO A.in2;
```

б)

```
CREATE D: node;
UNLINK  C.out1 FROM A.in1;
            A.out2 FROM C.in2;
LINK  C.out1 TO D.in1;
        D.out1 TO A.in1;
        A.out2 TO D.in2;
        D.out2 TO C.in2;
```

в)

```
UNLINK      A.out1 FROM B.in1;
                C.out2 FROM B.in2;
                B.out1 FROM C.in1;
                B.out2 FROM A.in2;
LINK  A.out1 TO A.in2;
            C.out2 TO C.in1;
DELETE B;
```

г)

```
UNLINK  A.out1 FROM A.in2;
            C.out2 FROM C.in1;
LINK  A.out1 TO C.in1;
            C.out2 TO A.in2;
```

Задатак 3. **Обрада података**

Написати програм на CONIC-у који прихвата улазни податак за обраду, обрађује га и шаље одговор на излазни порт. Поред овога програм треба да враћа и број захтева који се тренутно обрађују. Од формираног модула направити нов модул који садржи n модула за обраду података, један модул за прихватање и прослеђивање захтева.

Решење:

```
task module Machine;
    entryport    Orders, Finished: Order;
     Status: Inquiry reply integer;
    exitport
     Confirmation, Start: Order;
    var     number_of_orders: integer;
        production_order : Order;
        working : boolean;
    begin
        number_of_orders := 0;
        working := false;
            loop
                select
                    when (not working)
                    receive production_order from Orders =>
                    number_of_orders:= number_of_orders + 1;
                    send production_order to Start;
                    working := true;
                or
                    when (working)
                    receive production_order from Finished =>
                    number_of_orders := number_of_orders - 1;
                    send production_order to Confirmation;
                    working := false;
                or receive Inquiry from Status
                    reply number_of_orders;
            end;
    end;
end.

task module Worker();
```

```
    entryport    Start: Order;
    exitport  Finished: Order;

    var production_order: Order;
begin
    loop
        receive production_order from Start =>
                work();
                send production_order to Finished;
    end;
end.

task module Distributor(n: integer := 2);
    entryport    In: Order;
    exitport  Out[1..n]: Order;
    var number_of_orders: integer;
    var production_order: Order;
begin
    number_of_orders := 1;
    loop
        receive production_order from In =>
            number_of_orders := (number_of_orders) mod n + 1;
            send production_order to Out[number_of_orders];
    end;
end.

group module Manufacturing (n: integer := 2);
    entryport
        Orders: Order;
        Status [1..n]: Inquiry reply integer;
    exitport
        Confirmation: Order;
    use
        Distributor, Machine, Worker;
    create
        Distributor at Knot_X;
    create family k:[1..n]
        Machine [k]: Machine at Knot[k];
        Worker[k]: Worker at W[k];
    link
        Orders to Distributor.In
    link family k:[1..n]
        Distributor.Out[k] to Machine[k].Orders;
        Status[k] to Machine[k].Status;
        Machine[k].Confirmation to Confirmation;
        Machine[k].Start to Worker[k].Start;
        Worker [k].Finished to Machine[k].Finished;
end.
```

Задатак 4. **Bounded Buffer**

```
task module bound;
    entryport    put_char : char reply signaltype;
                 get_char : signaltype reply char;
    const max_size = 100;
    var    inp,
           outp,
           contents : natural;
           buffer : array [1..max_size] of char;
    begin
        inp := 1;
        outp := 1;
        contents := 0;
        loop
            select
```

```
            when (contents < max_size) {buffer not full}
            receive buffer[inp] from put_char reply signal
            =>      inp := (inp mod max_size) + 1;
                    contents := contents + 1;
                end;
        or
            when (contents > 0) {buffer not empty}
            receive signal from get_char reply buffer[outp]
            =>      outp := (outp mod max_size) + 1;
                    contents := contents - 1;
                end;
        end;
    end {loop}
end {task}.
```

Задатак 5. **Dining philosophers problem**

```
define philos:forktype;
    type forktype = (pickup, putdown);
end.

task module phil (thinktime, eattime : integer);
    use philos:forktype;
    exitport rightfork : forktype reply signaltype;
             leftfork: forktype reply signaltype;
             leavetable : signaltype reply signaltype;
             sittable : signaltype reply signaltype;
    type activitytype = (thinking, sitting, eating);
    var  request, ok : signaltype;
         pickupreq, putdownreq : forktype;

    begin
        request := signal;
        pickupreq := pickup; putdownreq := putdown;
        loop
            {thinking}
            delay (thinktime);
            send request to sittable wait ok;
            {sitting}
            send pickupreq to leftfork wait ok;
            send pickupreq to rightfork wait ok;
            {eating}
            delay (eattime);
            send request to leavetable wait ok;
            send putdownreq to leftfork wait ok;
            send putdownreq to rightfork wait ok;
        end;
    end.

task module fork;
    use philos : forktype;
    entryport   rightphil : forktype reply signaltype;
                leftphil : forktype reply signaltype;

    type allocationtype = (none, left, right);
    var  allocated : allocationtype;
         request : forktype;
    begin
        allocated := none;
        loop
            select
                when ((allocated=none) or (allocated=left))
                    receive request from leftphil reply signal
                    => case request of
```

```
                                pickup: allocated := left;
                                putdown: allocated := none;
                            end;
            or
                when ((allocated=none) or (allocated=right))
                        receive request from rightphil reply signal
                            => case request of
                                pickup: allocated := right;
                                putdown: allocated := none;
                            end;
            end {select};
        end {loop};
    end.

task module table (n:integer); {number of philosophers}

    entryport   leave : signaltype reply signaltype;
                sit : signaltype reply signaltype;
    var     sitting : integer;
            request, ok : signaltype;
    begin
        sitting := 0;
        ok:= signal;
        loop
            select
                when sitting > 0
                    receive request from leave reply ok
                        => sitting := sitting - 1;
                or
                when sitting < (n-1)
                    receive request from sit reply ok
                        => sitting := sitting + 1;
            end {select};
        end {loop};
    end.

group module diners(n:integer);
    use phil; fork; table; {dining philospher modules}
    create table : table(n=n);
    create    family k:[0..n-1]
              phil[k] : phil(n-k+2,2);
              fork[k] : fork;
    link family k : [0..n-1]
              phil[k].sittable to table.sit;
              phil[k].leavetable to table.leave;
              phil[k].rightfork to fork[k].leftphil;
              phil[(k+1) mod n].leftfork to fork[k].rightphil;
end.
```

ADA

Задатак 1. **Једноелементни прихватник**

Процедура P у језику Ada садржи локални једноелементни прихватник облика

```
task BOX is
    entry PUT(X: in integer);
    entry GET(X: out integer);
end;
task body BOX is
        V : INTEGER
begin
    loop
        accept PUT(X: in integer) do
            V := X;
        end;
        accept GET(X: out integer) do
            X := V;
        end;
    end loop;
end BOX;
```

Шта ће се десити са таском BOX када се заврше све остале упоредне активности настале извршавањем процедуре P и треба остварити повратак из процедуре?

Како треба модификовати BOX да би се омогућио коректан завршетак процедуре P?

Решење:

Таск BOX ће и даље постојати, што значи да се извршење процедуре P не може нормално завршити. (таск се може принудно прекинути споља, командом аборт).

```
task BOX is
entry PUT(X: in integer);
entry GET(X: out integer);
end;
task body BOX is
    V : INTEGER
begin
loop
    select
        accept PUT(X: in integer) do
            V := X;
        end;
    or
        terminate;
    end select;
    select
        accept GET(X: out integer) do
            X := V;
        end;
    or
        terminate;
    end select;
end loop;
end BOX;
```

Задатак 2. **Комуникација са поузданим везама**

Један дистрибуирани оперативни систем обезбеђује за интерпроцесну комуникацију поуздане операције асинхроног слања и временски условљеног пријема облика

```
procedure snd(P: process, m: msg)
function rcv(P: process, var m: msg, t: duration): boolean
```

где тип порука msg обухвата целе бројеве и специјалне симболе као што су ack и nack за позитивне односно негативне потврде (по потреби можете увести и друге симболе), временски тип duration обухвата вредности у опсегу 0..maxtime и специјалну вредност $\propto$ која означава

неограничено чекање, а rcv враћа булову вредност која показује да је ли порука примљена у назначеном интервалу времена t.

Користећи Pascal проширен наведеним операцијама, реализовати следеће бидирекционе трансакције типа rendez-vous, са семантиком као у језику Ada, између процеса S који тражи услугу SERV и процеса R који му је нуди. Именовање не треба решавати на генералан начин, већ сматрати да су ово једини процеси и да знају један за другог.

а)

```
{ proces S }
A,B: integer:= 1;
…
R.SERVE(A,B);
```

```
{ proces R }
Accept SERVE(X:in integer;
Y:out integer) do
Y := F(X);
end SERVE;
```

б)

```
{ process S }
A,B: integer := 1;
…

select
    R.SERVE(A,B);
    NORMAL_ACTION;
or
    delay t1;
    TIMEOUT_ACTION;
end select;
```

```
{ proces R }
select
accept SERVE(X:in integer; Y:out
integer) do
        Y := F(X);
    end SERVE;
else
    DEFAULT_ACTION;
end select;
```

Решење:

а)

```
var a, b: integer;
    st: boolean
begin
    …
    a := 1;
    snd(R, a);
    st := rcv(R, b, ∝);
    …
end
```

```
var x, y: integer;
    st: boolean;
begin
    …
    st := rcv(S, x, ∝);
    y := f(x);
    snd(S, y);
    …
end
```

б)

```
var a, b: integer;
    st: boolean;
    rep: message;
begin
    …
    a := 1;
    snd(R, a);
if rcv(R,rep,t1) and rep=accept
then
    begin
        snd(R, ack);
        st := rcv(R, b, ∝);
        NORMAL_ACTION
    end
    else
    begin
        snd(R, nack);
        TIMEOUT_ACTION
    end
    …
```

```
var x, y: integer;
    st: boolean;
    rep: message;
begin
    …
    if rcv(S, x, 0) then
    begin
        snd(S, accept);
        st := rcv(S,rep, ∝);
        if rep = ack then
            snd(S, f(x));
        else
        DEFAULT_ACTION
        end
    else
        DEFAULT_ACTION;
    …
end
```

Задатак 3. **Клијент-сервер**

У једној дистрибуираној апликацији већи број клијената C1,...,Cn користи услуге једног сервера S. Клијенти нису међусобно равноправни: ако има више пристиглих захтева, S треба најпре да

услужи клијента Ci са најмањим индексом и.
а) Увести погодну нотацију за селективни пријем порука у складу са наведеном дисциплином, и објаснити значење уведене нотације.
б) Како би се наведени проблем (за два клијента C1 и C2) решио у језику Conic ?
в) Како би се наведени проблем (за два клијента C1 и C2) решио у језику Ada ? (Подразумева се да клијенти упућују захтеве на различите улазе (entry) сервера S: C1 позива S.E1 (m1), а C2 позива S.E2 (m2)).

Решење:

а)

```
select
    receive m1 from C1 => process m1;
or
    receive m2 from C2 => process m2;
...
or
    receive mn from Cn => process mn;
end select
```

б)

```
select
    receive m1 from c1 → process m1;
or
    receive m2 from c2 → process m2;
end select
```

в)

```
select
    accept E1(m1) do process m1 end;
or
    when E1'count = 0 =>
        accept E2(m2) do process m2 end;
end select
```

или

```
select
    accept E1(m1) do process m1 end;
else
    select
        accept E1(m1) do process m1 end;
    or
        accept E2(m2) do process m2 end;
    end select
end select
```

Задатак 4. **Conic to Ada**

Дат је програмски фрагмент на језику Ada:

```
select
    buffer.write(x);
    normal_action;
or
    delay t1;
    timeout_action;
end select
```

и сличан фрагмент на језику Conic:

```
send x to outport
    wait signal => normal_action;
    fail t1 => timeout_action;
end
```

Претпоставимо да је време t1 истекло у тренутку када је пријемни процес прихватио захтев и почео да га обрађује, али још није одговорио.
а) Да ли ће се, у случају Ade, извршити normal_action или timeout_action? Шта је предност оваквог решења?

б) Исто за Conic.

Задатак 5. **Readers – Writers problem**

```
with ada.text_io, ada.integer_text_io;
use ada.text_io, ada.integer_text_io;

procedure rw is
   n : constant integer := 2;

   task control is
      entry start_read;
      entry stop_read;
      entry start_write;
      entry stop_write;
   end control;

   task body control is
      done : boolean;
      readers : integer range 0 .. n;
      writer : boolean;
   begin
      readers := 0;
      writer := false;
      loop
         select
            when (not writer) =>
               accept start_read;
               Put_Line ("start_read");
               readers := readers + 1;
               null;
         or
            accept stop_read;
            Put_Line ("stop_read");
            readers := readers - 1;
            null;
         or
            when (not writer) and (readers = 0) =>
               accept start_write;
               writer := true;
               Put_Line ("start_write");
               null;
         or
            accept stop_write;
            writer := false;
            Put_Line ("stop_write");
            null;
         or
            terminate;
         end select;
         exit when done;
      end loop;
   end control;

   task reader_1 is
   end reader_1;

   task body reader_1 is
      done : boolean;
      i : integer :=0;
   begin
      loop
         control.start_read;
         Put_Line ("reader_1 " & i'Img);
```

```
            control.stop_read;
            delay 1.0;
            i := i + 1;
            if i > 10 then
                done := true;
            end if;
            exit when done;
        end loop;
    end reader_1;

    task reader_2 is
    end reader_2;

    task body reader_2 is
        done : boolean;
        i : integer :=0;
    begin
        loop
            control.start_read;
            Put_Line ("reader_1 " & i'Img);
            control.stop_read;
            delay 1.0;
            i := i + 1;
            if i > 10 then
                done := true;
            end if;
            exit when done;
        end loop;
    end reader_2;

    task writer_1 is
    end writer_1;

    task body writer_1 is
        done : boolean;
        i : integer :=0;
    begin
        loop
            control.start_write;
            Put_Line ("writer_1 " & i'Img);
            control.stop_write;
            delay 1.9;
            i := i + 1;
            if i > 10 then
                done := true;
            end if;
            exit when done;
        end loop;
    end writer_1;

    task writer_2 is
    end writer_2;

    task body writer_2 is
        done : boolean;
        i : integer :=0;
    begin
        loop
            control.start_write;
            Put_Line ("writer_2 " & i'Img);
            control.stop_write;
            delay 1.7;
            exit when done;
            i := i + 1;
            if i > 10 then
```

```
                done := true;
            end if;
            exit when done;
        end loop;
    end writer_2;

    begin
        delay 10.0;
end rw;
```

Задатак 6. **Producer – Consumer/Bounded Buffer**

```
with ada.text_io, ada.integer_text_io;
use ada.text_io, ada.integer_text_io;
procedure Producer_Consumer is
    task buffer is
        entry PUT(coin: in integer);
        entry GET(goods: out integer);
    end buffer;
    task body buffer is
    size: constant := 500;
        data: array(1 .. size) of integer;
        indata: integer:=1;
        outdata: integer:= 1;
        num: integer:=0;
    begin
        loop
            select
                when (num < size) =>
                    accept PUT(coin: in integer) do
                        data(indata) := coin;
                        indata := (indata) mod size + 1;
                        num:= num + 1;
                    end PUT;
                    null;
            or
                when (num > 1) =>
                    accept GET(goods: out integer) do
                        goods := data(outdata);
                        outdata := (outdata) mod size + 1;
                        num:= num - 1;
                    end GET;
                    null;
            end select;
        end loop;
    end buffer;

    task producer1 is
    end producer1;
    task body producer1 is
    i: integer:=0;
    begin
        loop
            buffer.PUT(i);
            Put_Line ("buffer.PUT(i)" & i'Img);
            i := i + 1;
            delay 1.0;
        end loop;
    end producer1;

    task producer2 is
    end producer2;
    task body producer2 is
    i: integer:=0;
```

```
    begin
        loop
            buffer.PUT(i);
            Put_Line ("buffer.PUT(i)" & i'Img);
            i := i + 1;
            delay 1.0;
        end loop;
    end producer2;

    goods: integer := 0;
    i: integer :=0;
    begin
        loop
            buffer.GET(goods);
            Put_Line ("buffer.GET(goods)" & goods'Img & " " & i'Img);
            i := i + 1;
            delay 1.0;
        end loop;
end Producer_Consumer;
```

Задатак 7. **Cigarette Smokers' problem**

```
with ada.text_io, ada.integer_text_io, ADA.NUMERICS.DISCRETE_RANDOM;
use ada.text_io, ada.integer_text_io;

procedure CS is
    n : constant integer := 2;
    MinNum :constant integer := 1;
    MaxNum :constant integer := 3;

    task Table is
        entry yes_matches;
        entry yes_tobacco;
        entry yes_paper;
        entry goods(inMatches, inTobacco, inPaper: in boolean);

    end Table;

    task body Table is
        done : boolean;
        matches, tobacco, paper: boolean;
    begin
        done := false;
        matches := false;
        tobacco := false;
        paper := false;
        loop
            select
                when (matches AND tobacco) =>
                    accept yes_paper do
                        matches := false;
                        tobacco := false;
                        Put_Line ("start yes_paper");
                        null;
                    end yes_paper;
            or
                when (tobacco AND paper) =>
                    accept yes_matches do
                        tobacco := false;
                        paper := false;
                        Put_Line ("start yes_matches");
                        null;
                    end yes_matches;
            or
```

```
            when (matches AND paper) =>
                accept yes_tobacco do
                    matches := false;
                    paper := false;
                    Put_Line ("start yes_tobacco");
                    null;
                end yes_tobacco;
        or
            accept goods(inMatches, inTobacco, inPaper: in boolean) do
                matches := inMatches;
                tobacco := inTobacco;
                paper := inPaper;
                Put_Line ("goods");
                null;
            end goods;
        or
            terminate;
        end select;
        exit when done;
    end loop;
end Table;

task Agent is
    entry OK;
end Agent;

task body Agent is
subtype myNumber is INTEGER range MinNum .. MaxNum;
package myRandomNumbers is new ADA.NUMERICS.DISCRETE_RANDOM(myNumber);

    use myRandomNumbers;

    done : boolean;
    i: integer;
    RandomNumber : myNumber;
    Seed : GENERATOR;

begin
    RESET(Seed);
    loop
        RandomNumber := RANDOM(Seed);
        i:= RandomNumber;
        Put_Line ("ALIVE");
        if (i = 1) Then
            Table.goods(false, true, true);
            Put_Line ("Agent.goods(false, true, true) " & i'Img);
        elsif (i = 2) Then
            Table.goods(true, false, true);
            Put_Line ("Agent.goods(true, false, true) " & i'Img);
        elsif (i = 3) Then
            Table.goods(true, false, true);
            Put_Line ("Agent.goods(true, true, false) " & i'Img);
        end if;

        select
            accept OK;
            Put_Line ("OK");
        or
            terminate;
        end select;

        delay 1.0;
    end loop;
end Agent;
```

```
    task smoker_with_paper is
    end smoker_with_paper;

    task body smoker_with_paper is
        done : boolean;
        i : integer :=0;
    begin
        loop
            Table.yes_paper;
            Put_Line ("yes_paper " & i'Img);
            delay 1.0;
            Agent.OK;
            i := i + 1;
        end loop;
    end smoker_with_paper;

    task smoker_with_matches is
    end smoker_with_matches;

    task body smoker_with_matches is
        done : boolean;
        i : integer :=0;
    begin
        loop
            Table.yes_matches;
            Put_Line ("yes_matches " & i'Img);
            delay 1.0;
            Agent.ok;
            i := i + 1;
        end loop;
    end smoker_with_matches;

    task smoker_with_tobacco is
    end smoker_with_tobacco;

    task body smoker_with_tobacco is
        done : boolean;
        i : integer :=0;
    begin
        loop
            Table.yes_tobacco;
            Put_Line ("yes_tobacco " & i'Img);
            delay 1.0;
            Agent.ok;
            i := i + 1;
        end loop;
    end smoker_with_tobacco;

    begin
        delay 1.0;
end CS;
```

Превођење:
gcc -c imefajla.adb
gnatbind imefajla
gnatlink imefajla
gnatmake imefajla.adb

LINDA

Задатак 1. **Семафори**

Користећи библиотеку C-Linda приказати како се могу креирати методе еквивалентне онима приликом рада са семафорима.

Решење:

signal: **out** (“sem”)
wait: **in** (“sem”)

Иницијализација на вредност n се врши тако што се out (“sem”) изврши n пута.
Или у виду метода:

```
void signal(String sem){
    out(sem);
}
void wait(String sem){
    in(sem);
}
void init(String sem, unsigned int va){
    for(int I = 0; i < val; i++){
        out(sem);
    }
}
```

Задатак 2. **Dining philosophers problem**

```
void phil (int i) {
    int i;
    while (1) {
        think ();
        in (“room ticket”);
        in (“chopstick”, i);
        in (“chopstick”, (i+1)%Num);
        eat ();
        out (“chopstick”, i);
        out (“chopstick”, (i+1)%Num);
        out (“room ticket”);
    }
}
void initialize () {
    int i;
    for (i=0; i<Num; i++) {
        out (“chopstick”, i);
        if (i<(Num-1)) out (“room ticket”);
        eval (phil (i));
    }
}
```

Задатак 3. **Клијент-сервер**

У једном дистрибуираном рачунарском систему има више клијената и један сервера. Број клијената је произвољан. Клијент шаље захтев, а сервер га опслужује и шаље одговор клијенту. Захтев који је раније послат има предност над оним који је послат касније.

Решење:

```
void server () {
    int index = 1;
    ...
    while (1) {
        in (“request”, index, ?req);
        ...
```

```
        out (“response”, index++, response);
    }
}
void client (int id) {
    int index;
    ...
    in (“client index”, ?index);
    out (“client index”, index+1);
    ...
    out (“request”, index, request);
    in (“response”, index, ?response);
    ...
}
void init () {
    out ("server index ", 1);
    eval (server());
    for (int i=0; i<10; i++)
        eval (client(i));
}
```

Задатак 4. **Клијент-сервер**

У једном дистрибуираном рачунарском систему има више клијената и више сервера. Број клијената је произвољан, а број сервера се налази у простору торки. Клијент шаље захтев, а један од сервера (било који) га опслужује и шаље одговор клијенту. Захтев који је раније послат има предност над оним који је послат касније. Постоји и процес *sumator* који с времена на време захтева од свих сервера да му пошаљу број обрађених захтева до тог тренутка и то тако што сви сервери заврше са обрадом захтева на коме тренутно раде, шаљу том процесу тражени податак и не узимају нови захтев све док овај процес не добије податке од свих сервера. Написати функције *klijent*, *server* и *sumator* који реализују наведене операције и код за иницијализацију система (ако је потребан) користећи библиотеку C-Linda.

Решење:

У prostoru torki (tuple space) се могу наћи следеће торке:

("number of servers", **int** n) - укупан број сервера
("summing") - налази се у простору торки када sumator врши сумирање
("not summing") - налази се у простору торки када sumator не врши сумирање
("client index", **int** i) - означава редни број следећег захтева који шаље клијент
("server index", **int** i) - означава редни број следећег захтева који узима сервер
("request", **int** redni_broj, **int** param) - захтев са одговарајућим редним бројем (није потребно рећи од ког је клијента, јер је редни број јединствен, а није битно ни ком серверу је упућен)
("response", **int** redni_broj, **int** rezultat) - одговор на захтев са одговарајућим редним бројем
("number of processed", **int** broj_obradenih) - одговор на упит о броју обрађених захтева

```
void client (int id) {
    int index;
    ...
    in (“client index”, ?index);
    out (“client index”, index+1);
    ...
    out (“request”, index, request);
    in (“response”, index, ?response);
    ...
}

void server () {
    int i=1, cnt=0, x;
    in ("number of servers", ?x);
    out ("number of servers", x + 1);
    while (1)
        if (rdp ("summing")) {
            out ("number of processed", cnt);
```

```
            rd ("not summing");
        }
        else  {
            in ("server index", ?i);
            out ("server index", ++i);
            in ("request", i, ?x);
            out ("response", i, f(x));
            cnt++;
        }
}
void sumator () {
    int n, ukupno, j;
    ....
    in (“not summing”);
    out ("summing");
    rd ("number of servers", ?n);
    cnt = 0;
    for (int i = 0; i < n; i++) {
        in ("number of processed", ?j);
        cnt += j;
    };
    in ("summing");
    out (“not summing”)
    ...
}
void init () {
    out ("number of servers", 0);
    out ("client index", 0);
    out ("server index", 0);
    out (“not summing”);
    for (int i = 0; i < 10; i++)
        eval (server ());
}
```

Задатак 5. **Cigarette Smokers’ problem**

Користећи C-Lindu или Pascal-Lindu написати програм који решава проблем и симулира систем “нервозних пушача”. Постоји један агент и три нервозна пушача. Агент поседује резерве три неопходна предмета за лечење нервозе: папир, дуван и шибице. Један од пушача има бесконачне залихе папира, други – дувана, а трећи - шибица. Агент почиње тако што два различита предмета ставља на сто, један по један. Пушач, коме баш та два предмета фале, узима их, завија и пали цигарету и ужива. Након тога обавештава агента да је завршио, а агент онда ставља два нова предмета на сто, итд.

Решење:

```
void agent(){
    int n;
    while(1){
        n = (int)((rand()*3)/RAND_MAX);
        switch(n){
            case 0: out("Paper");
                    out("Tobacco");
                    break;
            case 1: out("Tobacco");
                    out("Matches");
                    break;
            case 2: out("Matches");
                    out("Paper");
                    break;
        }
        in("OK");
    }
}
```

```
void smocker_with_matches(){
    while(1){
        in("Watch");
        if(rdp("Paper") && rdp("Tobacco")){
            in("Paper");
            in("Tobacco");
            enjoy();
            out("OK");
        }
        out("Watch");
    }
}
...
void init () {
    eval(agent());
    eval(smocker_with_matches ());
    eval(smocker_with_paper ());
    eval(smocker_with_tobacco ());
    out("Watch");
}
```

Задатак 6. **Readers – Writers problem**

Користећи C Lindu решити проблем читалаца и писаца. Решење треба да обезбеди да процес који је пре стигао пре и започне операцију читања односно уписа.

```
void reader(){
    int id, num;
    while(1){
        in("id", ?id);
        out("id", id + 1);
        in("ok_to_work", id);
        in("readers_num", ?num);
        out("readers_num", num + 1);
        out("ok_to_work", id + 1);
        reading();
        in("readers_num", ?num);
        out("readers_num", num - 1);
    }
}

void writer(){
    int id;
    while(1){
        in("id", ?id);
        out("id", id + 1);
        in("ok_to_work", id);
        rd("readers_num", 0);
        writing();
        out("ok_to_work", id + 1);
    }
}

void init () {
    int i;
    out("id", 0);
    out("ok_to_work", 0);
    out("readers_num", 0);
    for (i=0; i<10; i++) {
        eval (reader ());
        eval (writer ());
    }
}
```

Задатак 7. **Проблем избора**

Користећи C-Lindu написати програм који решава следећи проблем: Постоје три особе међу којима треба изабрати једну. Свака од тих особа поседује новчић који има две стране. Избор особе се одиграва тако што свака особа независно баца свој новчић. Уколико постоји особа којој је новчић пао на другу страну у односу на преостале две онда се та особа изабира. Уколико све три особе имају исто постављен новчић поступак се понавља све док се не изабере једна.

Решење:

```
void Player(int id){
    int PID;
    int coin, coinl, coinr;
    int end, winner;
    in("PID", ?PID);
    do{
        PID ++;
        coin = (int)((rand()*2)/RAND_MAX);
        out("RESULT", id, PID, coin);
        out("RESULT", id, PID, coin);
        in("RESULT", (id + 1)%3, PID, ?coinr);
        in("RESULT", (id + 2)%3, PID, ?coinl);
        end = !((coin == coinl) && (coin == coinr) && (coinl == coinr));
        winner = (coin != coinl) && (coin != coinr);
    }while(!end);
}
void init () {
    out("PID", 0);
    out("PID", 0);
    out("PID", 0);
    eval (Player(0));
    eval (Player(1));
    eval (Player(2));
}
```

Задатак 8. **Проблем лифтова**

Користећи C-Lindu написати програм који решава проблем путовања лифтом. Путник позива лифт са произвољног спрата. Када лифт стигне на неки спрат сви путници који су изразили жељу да сиђу на том спрату обавезно изађу. Након изласка путника сви путници који су чекали на улазак уђу у лифт и кажу на који спрат желе да пређу. Тек када се сви изјасне лифт прелази даље. Није потребно оптимизовати пут лифта и путника.

Решење:

```
void elevator(){
    int x;
    while(1){
        x = getFloor();

        out("off", x);
        in("stop", x, 0);
        in("off", x);
        out("stop", x, 0);

        out("on", x);
        in("start", x, 0);
        in("on", x);
        out("start", x, 0);
    }
}

int getFloor(){
    int x;
    in("floor", ?x);
    return x;
```

```
}

void passenger(unsigned int ID, x, y){
    int s1, s2;
    if(x != y){
        in("start", x, ?s1);
        if(s1 == 0) out("floor", x);
        out("start", x, s1+1);

        in("on", x);
        in("start", x, ?s1);
        in("stop", y, ?s2);
        if(s2 == 0) out("floor", y);
        inp("floor", x);
        out("stop", y, s2+1);
        out("start", x, s1-1);
        out("on", x);

        in("off", y);
        in("stop", y, ?s2);
        inp("floor", y);
        out("stop", y, s2-1);
        out("off", y);
    }
}

void init(){
    int floorNum = N;
    int i;

    for(i=0; i < floorNum; i++){
        out("start", i, 0);
        out("stop", i, 0);
    }
    eval( elevator());
    ...
}
```

JAVA

Задатак 1. **Условна синхронизација - семафори**

```
public class Semaphore{
    private int s = 0;

    public synchronized void initS(int i){
        s = i;
    }
    public synchronized void waitS(){
        while (s == 0) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        s = s - 1;
//      notifyAll();
    }
    public synchronized void signalS(){
        s = s + 1;
        notify();
//      if ( s == 1) notify();
//      if ( s == 1) notifyAll();
    }
}

public class Tacka{

    public int x;
    public int y;
    public Tacka(int x, int y){
        this.x = x;
        this.y = y;
    }
}

public class Makepoints extends Thread{
    int i;
    int n;
    Tacka t;
    Semaphore full, empty;

    public Makepoints(Tacka t, int n, Semaphore full, Semaphore empty){
        this.t = t;
        this.n = n;
        this.full = full;
        this.empty = empty;
    }

    public void run(){
        for( i = 1; i < n; i ++){
            empty.waitS();
            t.x = i;
            t.y = i*i;
            full.signalS();
        }
    }
}

public class Printpoints extends Thread{
    int i;
```

```
    int n;
    Tacka t;
    Semaphore full, empty;

    public Printpoints(Tacka t, int n, Semaphore full, Semaphore empty){
        this.t = t;
        this.n = n;
        this.full = full;
        this.empty = empty;
    }

    public void run(){
        for( i = 0; i < n; i ++){
            full.waitS();
            System.out.println(t.x + " " + t.y);
            empty.signalS();
        }
    }
}

public class Graph{
    public static final int N = 89;
    public static void main(String []args){
        int n = N;
        Tacka t;
        t = new Tacka(0,0);
        Semaphore full = new Semaphore();
        Semaphore empty = new Semaphore();

        Makepoints makepoints = new Makepoints(t, n, full, empty);
        Printpoints printpoints = new Printpoints(t, n, full, empty);

        full.initS(1);
        empty.initS(0);
        makepoints.start();
        printpoints.start();
    }
}
```

Задатак 2. **Dining philosophers problem**

Прво решење:

```
public class Philosopher extends Thread {
    int id;
    int firstodd, secondeven;
    Semaphore [] fork;
    public Philosopher (int i, int n , Semaphore [] fork){
        id = i;
        this.fork = fork;
        if((i % 2) == 1){
            firstodd = i;
            secondeven = (i + 1) % n;
        }
        else{
            firstodd = (i + 1) % n;
            secondeven = i;
        }
    }

    private void think(){
        System.out.println("think " + id);
        try {
```

```
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }

    private void eat(){
        System.out.println("eat " + id);
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }

    public void run(){
        while(true){
            think();
            fork[firstodd].waitS();
            fork[secondeven].waitS();
            eat();
            fork[secondeven].signalS();
            fork[firstodd].signalS();
        }
    }
}
public class Dining_philosophers{
    public static final int N = 5;
    public static void main(String [] vpar){
        int n = N;
        Semaphore [] fork = new Semaphore[n];
        Philosopher [] philosopher = new Philosopher[n];
        int i;

        for( i=0; i < n; i++) fork[i] = new Semaphore();

        for( i=0; i < n; i++) philosopher[i] = new Philosopher(i, n , fork);

        for( i=0; i < n; i++) fork[i].initS(1);

        philosopher[0].start();
        philosopher[1].start();
        philosopher[2].start();
        philosopher[3].start();
        philosopher[4].start();
    }
}
```

Друго решење:

```
public class Philosopher extends Thread {
    int id;
    int left, right;
    Semaphore [] fork;
    Semaphore ticket;
    public Philosopher (int i, int n , Semaphore [] fork, Semaphore ticket){
        id = i;
        this.fork = fork;
        this.ticket = ticket;
        left = i;
        right = (i + 1) % n;
    }

    private void think(){
        System.out.println("think " + id);
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }
```

```
    private void eat(){
        System.out.println("eat " + id);
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }

    public void run(){
        while(true){
            think();
            ticket.waitS();
            fork[left].waitS();
            fork[right].waitS();
            eat();
            fork[right].signalS();
            fork[left].signalS();
            ticket.signalS();
        }
    }
}

public class Dining_philosophers{
    public static final int N = 5;
    public static void main(String [] vpar){
        int n = N;
        Semaphore ticket = new Semaphore();
        Semaphore [] fork = new Semaphore[n];
        Philosopher [] philosopher = new Philosopher[n];
        int i;

        for( i=0; i < n; i++) fork[i] = new Semaphore();

        for( i=0; i < n; i++) philosopher[i] = new Philosopher(i, n , fork, ticket);

        ticket.initS( n-1 );
        for( i=0; i < n; i++) fork[i].initS(1);

        philosopher[0].start();
        philosopher[1].start();
        philosopher[2].start();
        philosopher[3].start();
        philosopher[4].start();
    }
}
```

Задатак 3. **Условна синхронизација - региони**

```
public class TackaP{

    public int x;
    public int y;
    public boolean full;
    public TackaP(int x, int y, boolean full){
        this.x = x;
        this.y = y;
        this.full = full;
    }
}
public class Makepoints extends Thread{

    int i;
    int n;
    TackaP p;
```

```
    public Makepoints(TackaP p, int n){
        this.p = p;
        this.n = n;
    }

    public void run(){
        for( i = 1; i < n; i ++){
            synchronized(p){
                while (p.full) {
                    try {
                        p.wait();
                    } catch (InterruptedException e) { }
                }
                p.x = i;
                p.y = i*i;
                p.full = true;
                p.notifyAll();
            }
        }
    }
}

public class Printpoints extends Thread{

    int i;
    int n;
    TackaP p;

    public Printpoints(TackaP p, int n){
        this.p = p;
        this.n = n;
    }


    public void run(){
        for( i = 0; i < n; i ++){
            synchronized(p){
                while (!p.full) {
                    try {
                        p.wait();
                    } catch (InterruptedException e) { }
                }
                System.out.println(p.x + " " + p.y);
                p.full = false;
                p.notifyAll();
            }
        }
    }
}

public class Graph{
    public static final int N = 89;

    public static void main(String []args){
        int n = N;

        TackaP p;
        p = new TackaP(0,0, true);

        Makepoints makepoints = new Makepoints(p, n);
        Printpoints printpoints = new Printpoints(p, n);

        makepoints.start();
        printpoints.start();
```

```
    }
}
```

Задатак 4. **Прелазак моста**

```
public class Bridge{

    public int north;
    public int south;
    public Bridge(int north, int south){
        this.north = north;
        this.south = south;
    }
}

public class Car{

    Bridge bridge = null;
    int ID;

    public Car(Bridge bridge, int ID){
        this.bridge = bridge;
        this.ID = ID;
    }
    public void crossing(){
        System.out.println("Crossing " + ID);
        try {
            Thread.sleep(5000+(int)(Math.random()*1000));
        } catch (InterruptedException e) { }
        System.out.println("Crossed " + ID);

    }
    public void starting(){
        try {
            Thread.sleep(5000+(int)(Math.random()*1000));
        } catch (InterruptedException e) { }
    }
    public void start() {
    }
}

public class North extends Car implements Runnable {

    private volatile Thread thread = null;

    public North(Bridge bridge, int ID){
        super(bridge, ID);
    }

    public void start() {
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
    }

    public void run() {

        starting();
        synchronized(bridge){
            while (bridge.south != 0) {
                try {
                    bridge.wait();
                } catch (InterruptedException e) { }
```

```
            }
            bridge.north ++;
            //bridge.notifyAll();
        }
        crossing();

        synchronized(bridge){
            bridge.north --;
            if(bridge.north == 0)
            bridge.notifyAll();
        }
    }

}

public class South extends Car implements Runnable {

    private volatile Thread thread = null;

    public South(Bridge bridge, int ID){
        super(bridge, ID);
    }

    public void start() {
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
    }
    public void run() {

        starting();
        synchronized(bridge){
            while (bridge.north != 0) {
                try {
                    bridge.wait();
                } catch (InterruptedException e) { }
            }
            bridge.south ++;
            //bridge.notifyAll();
        }

        crossing();

        synchronized(bridge){
            bridge.south --;
            if(bridge.south == 0)
            bridge.notifyAll();
        }
    }
}
public class BridgeCrossing{
    public static final int N = 89;
    public static void main(String []args){
        int n = N;
        Bridge bridge = new Bridge(0,0);
        Car [] south = new Car[n];
        Car [] north = new Car[n];

        for(int i=0; i < n; i++) {
            south[i] = new South(bridge, 2*i);
            north[i] = new North(bridge, 2*i+1);
        }

        for(int i=0; i < n; i++) {
```

```
            south[i].start();
            north[i].start();
        }
    }
}
```

Задатак 5. **Прелазак моста**

```
public class Direction{

    public int wait;
    public int cross;
    public int ahead;
    public Direction(){
        wait = 0;
        cross = 0;
        ahead = 0;
    }
}

public class Bridge{

    public Direction north;
    public Direction south;
    public Bridge(){
        north = new Direction();
        south = new Direction();
    }
}

public class Car{

    Bridge bridge = null;
    int ID;

    public Car(Bridge bridge, int ID){
        this.bridge = bridge;
        this.ID = ID;
    }
    public void crossing(){
        System.out.println("Crossing " + ID);
        try {
            Thread.sleep(5000+(int)(Math.random()*1000));
        } catch (InterruptedException e) { }
        System.out.println("Crossed " + ID);
    }
    public void starting(){
        try {
            Thread.sleep(5000+(int)(Math.random()*1000));
        } catch (InterruptedException e) { }
    }
    public void start() {

    }
}

public class North extends Car implements Runnable {

    private volatile Thread thread = null;

    public North(Bridge bridge, int ID){
        super(bridge, ID);
    }
```

```
    public void start() {
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
    }

    public void run() {

        starting();
        synchronized(bridge){
            bridge.north.wait++;
            while (!((bridge.south.cross == 0) && (bridge.north.ahead<10))) {
                try {
                    bridge.wait();
                } catch (InterruptedException e) { }
            }
            bridge.north.wait--;
            bridge.north.cross++;
            if (bridge.south.wait > 0) bridge.north.ahead ++;
            //bridge.notifyAll();
        }

        crossing();

        synchronized(bridge){
            bridge.north.cross--;
            if (bridge.north.cross == 0){
                bridge.south.ahead = 0;
                bridge.notifyAll();
            }
        }
    }
}

public class South extends Car implements Runnable {

    private volatile Thread thread = null;

    public South(Bridge bridge, int ID){
        super(bridge, ID);
    }

    public void start() {
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
    }
    public void run() {

        starting();
        synchronized(bridge){
            bridge.south.wait++;
            while (!((bridge.north.cross == 0) && (bridge.south.ahead<10))) {
                try {
                    bridge.wait();
                } catch (InterruptedException e) { }
            }
            bridge.south.wait--;
            bridge.south.cross++;
            if (bridge.north.wait > 0) bridge.south.ahead ++;
            //bridge.notifyAll();
        }
```

```
        crossing();

        synchronized(bridge){
            bridge.south.cross--;
            if (bridge.south.cross == 0){
                bridge.north.ahead = 0;
                bridge.notifyAll();
            }
        }
    }
}

public class BridgeCrossing{

    public static void main(String []args){
        int n = 89;
        Bridge bridge = new Bridge();
        Car [] south = new Car[n];
        Car [] north = new Car[n];

        for(int i=0; i < n; i++) {
            south[i] = new South(bridge, 2*i);
            north[i] = new North(bridge, 2*i+1);
        }

        for(int i=0; i < n; i++) {
            south[i].start();
            north[i].start();
        }
    }
}
```

Задатак 6. **Условна синхронизација - монитори (Producer/Consumer problem)**

```
public class Tacka {

    public int x;
    public int y;

    public Tacka(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Buffer {
    private Tacka contents;
    private boolean available = false;

    public synchronized Tacka get() {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        available = false;
        notifyAll();
        return contents;
    }
```

```
    public synchronized void put(Tacka value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        contents = value;
        available = true;
        notifyAll();
    }
}
public class Makepoints extends Thread {
    // Producer

    private Buffer buffer;
    private int n;

    public Makepoints(Buffer buffer, int n) {
        this.buffer = buffer;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i < n; i++) {
            Tacka t = new Tacka(i, i * i);
            onebuffer.put(t);
            work();
        }
    }

    public void work() {
        try {
            sleep((int) (Math.random() * 100));
        } catch (InterruptedException e) {
        }
    }
}
public class Printpoints extends Thread {
    // Consumer
    private Buffer buffer;
    private int n;

    public Printpoints(Buffer buffer, int n) {
        this.buffer = buffer;
        this.n = n;
    }

    public void run() {
        Tacka value = new Tacka(0, 0);
        for (int i = 0; i < n; i++) {
            value = onebuffer.get();
            System.out.println(this.n + " got: " + value.x + ", " + value.y);
        }
    }
}
public class ProducerConsumerTest {
    public static void main(String[] args) {
        OneBuffer c = new OneBuffer();
        Makepoints p1 = new Makepoints(c, 10);
        Printpoints c1 = new Printpoints(c, 10);
        p1.start();
        c1.start();
    }
}
```

Задатак 7. **Reentrant monitor**

```
public class Reentrant {
    public synchronized void a() {
        b();
        System.out.println("here I am, in a()");
    }
    public synchronized void b() {
        System.out.println("here I am, in b()");
    }
}
```

Задатак 8. **Readers – Writers problem**

```
public class Readers_Writers{
    public static final int numR = 5;
    public static final int numW = 4;
    public static void main(String arg[]){

        Semaphore mutex = new Semaphore();
        Semaphore db = new Semaphore();
        Num reader_count = new Num();
        Num base = new Num();
        Readers [] r = new Readers [numR];
        Writers [] w = new Writers [numW];

        base.set(0);
        reader_count.set(0);
        mutex.initS(1);
        db.initS(1);
        for(int i = 0; i < numW; i ++){
            w[i] = new Writers(i, db, base);
            w[i].start();
        }
        for(int i = 0; i < numR; i ++){
            r[i] = new Readers(i, reader_count, mutex, db, base);
            r[i].start();
        }

    }
}

public class Readers extends Thread{
    private int ID;
    Semaphore mutex;
    Semaphore db;
    Num reader_count;
    Num base;

    public Readers(int ID, Num reader_count, Semaphore mutex, Semaphore db, Num base){
        this.ID = ID;
        this.reader_count = reader_count;
        this.mutex = mutex;
        this.db = db;
        this.base = base;
    }

    public void run(){
        while(true){
            prepare();
            mutex.waitS();
            reader_count.set(reader_count.get() + 1);
            if (reader_count.get() == 1) db.waitS();
            mutex.signalS();
```

```
            read_db();
            mutex.waitS();
            reader_count.set(reader_count.get() - 1);
            if (reader_count.get() == 0) db.signalS();
            mutex.signalS();
        }
    }
    public void read_db (){
        System.out.println(ID + " read_db " + base.get());
        try {
            sleep((int)(Math.random() * 10000 + 500));
        } catch (InterruptedException e) { }

    }
    public void prepare (){
        try {
            sleep((int)(Math.random() * 10000 + 500));
        } catch (InterruptedException e) { }

    }
}

public class Writers extends  Thread{
    private int ID;
    Semaphore db;
    Num base;

    public Writers(int ID, Semaphore db, Num base){
        this.ID = ID;
        this.db = db;
        this.base = base;
    }

    public void run(){
        while(true){
            create_data();
            db.waitS();
            write_db();
            db.signalS();
        }
    }
    public void create_data (){
        try {
            sleep((int)(Math.random() * 100 + 50));
        } catch (InterruptedException e) { }
    }
    public void write_db (){
        System.out.println("write_db " + ID);
        base.set(ID);
        try {
            sleep((int)(Math.random() * 10000 + 10));
        } catch (InterruptedException e) { }

    }
}
public class Num{
    int s = 0;

    public void set(int i){
        s = i;
    }
    public int get(){
        return s;
    }
}
```

Задатак 9. **Readers – Writers problem**

Решити проблем читалаца и писаца користећи програмски језик Јава. Решење треба да обезбеди да процес који је пре стигао пре и започне операцију читања односно уписа.

```
public class Book{

    public Book(){
    }
    public void reading(int ID){
        System.out.println("Reading " + ID);
        try {
            Thread.sleep(500+(int)(Math.random()*1000));
        } catch (InterruptedException e) { }
        System.out.println("End reading " + ID);
    }
    public void writing(int ID){
        System.out.println("Writing " + ID);
        try {
            Thread.sleep(5000+(int)(Math.random()*1000));
        } catch (InterruptedException e) { }
        System.out.println("End writing " + ID);
    }
}

public class ReadersWriters{
    private int ID;
    private int ok_to_work;
    private int readers_num;

    public ReadersWriters(){
        ID = 0;
        ok_to_work = 0;
        readers_num = 0;
    }
    public synchronized void startRead(){
        int PID;
        PID = ID ++;
        while (PID != ok_to_work) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        readers_num ++;
        ok_to_work ++;
        notifyAll();
    }
    public synchronized void endRead(){
        readers_num --;
        notifyAll();
    }
    public synchronized void startWrite(){
        int PID;
        PID = ID ++;
        while ((PID != ok_to_work)||(readers_num != 0)) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        readers_num = 0;
        notifyAll();
    }
    public synchronized void endWrite(){
        readers_num = 0;
        ok_to_work ++;
```

```
        notifyAll();
    }
}

public class Readers extends Thread{
    private Book b;
    private ReadersWriters rw;
    private int ID;
    public Readers(int ID, Book b, ReadersWriters rw){
        this.ID= ID;
        this.b = b;
        this.rw = rw;
    }

    public void run() {

        while(true){
            work();
            rw.startRead();
            b.reading(ID);
            rw.endRead();
        }
    }
}

public class Writers extends Thread{
    private Book b;
    private ReadersWriters rw;
    private int ID;
    public Writers(int ID, Book b, ReadersWriters rw){
        this.ID= ID;
        this.b = b;
        this.rw = rw;
    }

    public void run() {

        while(true){
            work();
            rw.startWrite();
            b.writing(ID);
            rw.endWrite();
        }
    }
}

public class rwTest {
    public static final int numR = 10;
    public static final int numW = 2;
    public static void main(String []args){

        Book b = new Book();
        ReadersWriters rw = new ReadersWriters();
        Readers [] r = new Readers[numR];
        Writers [] w = new Writers[numW];

        for(int i=0; i < numR; i++) {
            r[i] = new Readers(i, b, rw);
        }
        for(int i=0; i < numW; i++) {
            w[i] = new Writers(i, b, rw);
        }
        for(int i=0; i < numR; i++) {
            r[i].start();
        }
```

```
        for(int i=0; i < numW; i++) {
            w[i].start();
        }
    }
}
```

Задатак 10. **Заустављање**

```
import java.util.*;
import java.text.DateFormat;

public class Clock implements Runnable {
    private volatile Thread thread = null;
    public void start() {
        if (thread == null) {
            thread = new Thread(this, "Clock");
            thread.start();
        }
    }
    public void run() {
        Thread myThread = Thread.currentThread();
        while (thread == myThread) {
            paint();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e){
                System.out.println("Interrupted");
            }
        }
    }

    public void paint() {
        Calendar cal = Calendar.getInstance();
        Date date = cal.getTime();
        DateFormat dateFormatter = DateFormat.getTimeInstance();
        System.out.println(dateFormatter.format(date));
    }

    public void stop() {
        Thread stopThread = thread;
        thread = null;
        stopThread.interrupt();
    }
}


public class CS {

    public static void main(String []args){
        Clock c = new Clock();
        c.start();
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e){}
        c.stop();
    }
}

import java.util.*;
import java.text.DateFormat;
public class Clock implements Runnable {
    private volatile boolean running = false;
    private Thread thread = null;
    public void start() {
        if (!running) {
```

```
            thread = new Thread(this, "Clock");
            running = true;
            thread.start();
        }
    }

    public void run() {
        while (running) {
            paint();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e){
                System.out.println("Interrupted");
            }
        }
    }

    ...

    public void stop() {
        Thread stopThread = thread;
        thread = null;
        running = false;
        stopThread.interrupt();
    }
}
```

Задатак 11. **The roller coaster problem**

Проблем вожње тобоганом (*The roller coaster problem*). Претпоставити да постоји n нити које представљају путнике и једна нит која представља возило на тобогану. Путници се наизменично шетају по луна парку и возе на тобогану. Тобоган може да прими највише K путника при чему је K < n. Вожња тобоганом може да почне само уколико се сакупило тачно K путника. Написати програм на језику Java који симулира описани систем.

Решење:

```
public class Coaster extends Thread{
    ...
    public void run(){
        while(true){
            for( i = 0; i < K; i++) car.signalS ();
            allAboard.waitS ();
            depart ();
            for( i = 0; i < K; i ++) lastStop.signalS ();
            allOut.waitS ();
        }
    }
    public void depart (){
        …
    }
}
public class RollerCoaster{
    public static void main(String arg[]){
        …
        car.initS(0);
        allAboard.initS(0);
        lastStop.initS( 0);
        mutex.initS(1);
        allOut.initS(0);
        coaster.start();
        for(int i = 0; i < numP; i ++){
            pas[i] = new Passenger(…);
            pas[i].start();
        }
    }
}
```

```
public class Passenger extends Thread{
    …
    public void run(){
        while(true){
            car.waitS ();
            boardCar ();
            lastStop.waitS ();
            leaveCar ();
        }

    }
    public void boardCar (){
        mutex.waitS ();
        coaster.passengers ++;
        if (coaster.passengers == K){
            allAboard.signalS ();
        }
        mutex.signalS ();
    }
    public void leaveCar (){
        mutex.waitS ();
        coaster.passengers --;
        if (coaster.passengers == 0){
            allOut.signalS ();
        }
        mutex.signalS ();
    }
}
```

Задатак 12. **The Santa Claus problem**

Деда Мраз који живи на северном полу већи део свог времена проводи спавајући (*The Santa Claus Problem*). Могу га пробудити или уколико се испред врата појаве свих 9 његових ирваса или 3 од укупно 10 патуљака. Када се Деда Мраз пробуди он ради једну од следећих ствари: Уколико га је пробудила група ирваса одмах се спрема и креће на пут да подели деци играчке. Када се врати са пута свим ирвасима даје награду. Уколико га је пробудила група патуљака онда их он уводи у своју кућу, разговара са њима и на крају их испрати до излазних врата. Група ирваса треба да буде опслужена пре групе патуљака. Написати програм на језику Java који симулира описани систем.

Решење:

```
public class Elf extends Thread {
    private SantaClausHouse sc;
    private int id;

    public Elf(int i, SantaClausHouse sc) {
        id = i;
        this.sc = sc;
    }

    public void run() {
        for (;;) {
            work();
            sc.wakeupSantaE();
            talk();
            sc.exitTheRoom();
        }
    }


    private void talk() {
        ...
    }
```

```
    private void work() {
        ...
    }
}

public class Reindeer extends Thread {
    private SantaClausHouse sc;
    private int id;

    public Reindeer(int id, SantaClausHouse sc) {
        this.id = id;
        this.sc = sc;
    }

    public void run() {
        for (;;) {
            rest();
            sc.wakeupSantaR();
            riding();
            sc.exitTheSleigh();
        }
    }

    private void riding() {
        ...
    }

    private void rest() {
        ...
    }
}

public class SantaClaus extends Thread {
    private int dir;
    private SantaClausHouse sc;

    public SantaClaus(SantaClausHouse sc) {
        this.sc = sc;
    }

    public void run() {
        for (;;) {
            dir = sc.sleeping();
            if (dir == SantaClausHouse.ELVES) {
                talk();
                sc.sendoffElves();
            } else {
                riding();
                sc.outspanReindeers();
            }
        }
    }

    private void riding() {
        ...
    }

    private void talk() {
        ...
    }
}

public class SantaClausHouse {
    public static final int ELVES = 0;
```

```
    public static final int REINDEERS = 1;
    private static final int minElves = 3;
    private static final int minReindeers = 9;
    private int elvesAtTheDoor;
    private int reindeersAtTheDoor;
    private int elvesInTheRoom;
    private int reindeersInTheSleigh;
    private boolean wakeupE, wakeupR;
    private boolean enterElves, exitElves;
    private boolean enterReindeers, exitReindeers;
    private boolean isSleeping;

    public SantaClausHouse() {
        elvesAtTheDoor = 0;
        reindeersAtTheDoor = 0;
        elvesInTheRoom = 0;
        reindeersInTheSleigh = 0;
        wakeupE = false;
        wakeupR = false;
        enterElves = false;
        exitElves = false;
        enterReindeers = false;
        exitReindeers = false;
        isSleeping = true;
    }

    public synchronized void wakeupSantaE() {
        while (!isSleeping) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        elvesAtTheDoor++;
        if (elvesAtTheDoor == minElves) {
            wakeupE = true;
        }
        notifyAll();
        while (!enterElves) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        elvesAtTheDoor--;
        elvesInTheRoom++;
        notifyAll();
    }

    public synchronized void exitTheRoom() {
        while (!exitElves) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        elvesInTheRoom--;
        notifyAll();
    }

    public synchronized void wakeupSantaR() {
        while (!isSleeping) {
```

```
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        reindeersAtTheDoor++;
        if (reindeersAtTheDoor == minReindeers) {
            wakeupR = true;
        }
        notifyAll();
        while (!enterReindeers) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        reindeersAtTheDoor--;
        reindeersInTheSleigh++;
        notifyAll();
    }
    public synchronized void exitTheSleigh() {
        while (!exitReindeers) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        reindeersInTheSleigh--;
        notifyAll();
    }
    public synchronized int sleeping() {
        while (!(wakeupR || wakeupE)) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        isSleeping = false;
        if (wakeupR) {
            wakeupR = false;
            enterReindeers = true;
            notifyAll();
            while (reindeersAtTheDoor != 0) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    ;
                }
            }
            enterReindeers = false;
            notifyAll();
            return SantaClausHouse.REINDEERS;
        }
        else {
            wakeupE = false;
            enterElves = true;
            notifyAll();
            while (elvesAtTheDoor != 0) {
                try {
                    wait();
                } catch (InterruptedException e) {
```

```
                    ;
                }
            }
            enterElves = false;
            notifyAll();
            return SantaClausHouse.ELVES;
        }
    }

    public synchronized void sendoffElves() {
        exitElves = true;
        notifyAll();
        while (elvesInTheRoom != 0) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        exitElves = false;
        enterElves = false;
        isSleeping = true;
        notifyAll();
    }

    public synchronized void outspanReindeers() {
        exitReindeers = true;
        notifyAll();
        while (reindeersInTheSleigh != 0) {
            try {
                wait();
            } catch (InterruptedException e) {
                ;
            }
        }
        exitReindeers = false;
        enterReindeers = false;
        isSleeping = true;
        notifyAll();
    }
}
public class SantaClausTest{
    public static final int numReindeer = 9;
    public static final int numElve = 10;
    public static void main(String [] vpar){
        SantaClausHouse sc = new SantaClausHouse();
        SantaClaus santa = new SantaClaus(sc);
        Elve [] Elves = new Elve[numElve];
        Reindeer [] Reindeers = new Reindeer[numReindeer];
        for(int i = 0; i < numElve; i ++) {
            Elves[i] = new Elve(i, sc);
            Elves[i].start();
        }
        for(int i = 0; i < numReindeer; i ++){
            Reindeers[i] = new Reindeer(i, sc);
            Reindeers[i].start();
        }
        santa.start();
    }
}
```

Задатак 13. **The bus problem**

Проблем вожње аутобусом (*The bus problem*). Путници долазе на аутобуску станицу и чекају приви аутобус који наиђе. Када аутобус наиђе сви путници који су били на станици пробају да

уђу у аутобус. Уколико има места у аутобусу путници улазе у њега. Капацитет аутобуса је K места. Путници који су дошли док је аутобус био на станици чекају на следећи аутобус. Када сви путници који су били на станици у тренутку доласка аутобуса провере да ли могу да уђу и уђу уколико има места аутобус креће. Уколико аутобус дође на празну станицу одмах продужава даље. Сви путници силазе на завршној станици. Написати програм на језику Java који симулира описани систем.

Решење:

```
public class Station{
    private String name;
    private int ID;
    private int numP, numFP;
    private boolean busIN;
    private Bus bus;
    public Station(String name){
        this.name = name;
        bus = null;
        busIN = false;
    }
    public String toString(){return name;}

    public synchronized Bus waitBus(){
        while(true){
            while (busIN) {
                try {
                    wait();
                } catch (InterruptedException e) { }
            }
            numP ++;
            while (!busIN) {
                try {
                    wait();
                } catch (InterruptedException e) { }
            }
            numP--;
            if(numP == 0) notifyAll();
            if(numFP > 0){
                numFP--;
                return bus;
            }
        }
    }
    public synchronized int busEnter(int numFree, Bus b) {
        while (busIN) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        busIN = true;
        numFP = numFree;
        bus = b;
        notifyAll();
        while (numP !=0) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        busIN = false;
        bus = null;
        notifyAll();
        return numFP;
    }
}
public class Passenger extends Thread{
    public static int ID = 0;
```

```
    int PID;
    Station start, end;
    Bus bus;
    public Passenger(Station start, Station end){
        PID = ID ++;
        this.start = start;
        this.end = end;
    }
    private void trawel(){
        try {
            sleep((int)(Math.random() * 1000 + 4000));
        } catch (InterruptedException e) { }
    }
    public void run(){
        while(true){
            bus = start.waitBus();
            trawel();
            bus.exitTheBus(end);
        }
    }
}
public class Bus extends Thread{
    private static final int maxNumPasage = 50;
    private StationLIst sl;
    private Station nextStation;
    private int numFree;
    private boolean toExit = false;
    private static int ID = 0;
    private int BID = ID++;
    public Bus(){
        numFree = maxNumPasage;
        sl = new StationLIst();
    }
    private void travel(){
        try {
            sleep((int)(Math.random() * 1000 + 5000));
        } catch (InterruptedException e) { }
    }
    public void putStation(Station newStation){
        sl.put(newStation);
    }
    private Station getLastStation(){
        return sl.getLast();
    }
    private synchronized Station getNextStation(){
        notifyAll();
        nextStation = sl.getNext();
        return nextStation;
    }
    public int getID(){ return BID;}
    public synchronized void exitTheBus(Station exitStation){
        while (!toExit || exitStation != nextStation) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        numFree++;
    }
    public synchronized void permitionToExit(){
        toExit = true;
        notifyAll();
        try {
            wait(500);
        } catch (InterruptedException e) { }
        toExit = false;
```

```
    }
    public void run(){
        while(true){
            nextStation = getNextStation();
            travel();
            permitionToExit();
            numFree = nextStation.busEnter(numFree, this);
        }
    }
}
```

Задатак 14. **Chat**

У програмском језику Јава је потребно написати скуп класа за интерактивну комуникацију између корисника (CHAT). Потребно је реализовати решење код кога корисник може да пошаље већи број порука без чекања одговора на претходне.

```
package chat;

import java.net.*;

public class Server {

    public static void main(String[] args) {
        try {
            int port = Integer.parseInt(args[0]);
            ServerSocket listener = new ServerSocket(port);
            Socket client = listener.accept();
            Chat c = new Chat(client);
            c.communicate();
            listener.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

}
package chat;

import java.net.*;

public class Client {

    public static void main(String[] args) {
        try {
            int port = Integer.parseInt(args[1]);
            String host = args[0];
            Socket server = new Socket(host, port);
            Chat s = new Chat(server);
            s.communicate();
            server.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
package chat;

import java.net.*;

public class Chat {
    Socket client;

    public Chat(Socket client) {
```

```
        this.client = client;
    }

    public void communicate() {

        try {
            ReadThread read = new ReadThread(client);
            WriteThread write = new WriteThread(client);
            read.start();
            write.start();
            read.join();
            write.join();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
}
```

```
package chat;

import java.io.*;
import java.net.*;

public class ReadThread extends Thread {
    Socket client;

    InputStream in;

    BufferedReader pin;

    public ReadThread(Socket client) {
        try {
            this.client = client;
            this.in = client.getInputStream();
            pin = new BufferedReader(new InputStreamReader(in));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public void run() {
        try {
            String s;
            while ((s = pin.readLine()) != null) {
                System.out.println("> " + s);
            }
        } catch (Exception ex) {}
        finally {
            try {
                pin.close();
                in.close();
                client.close();
            } catch (IOException ex1) {}
        }
    }
}
```

```
package chat;

import java.io.*;
import java.net.*;

public class WriteThread extends Thread {
    Socket client;

    OutputStream out;
```

```
    PrintWriter pout;

    public WriteThread(Socket client) {
        try {
            this.client = client;
            this.out = client.getOutputStream();
            pout = new PrintWriter(out, true);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public void run() {
        try {
            InputStream is = System.in;
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            String s;
            while (!pout.checkError() && ((s = br.readLine()) != null)) {
                pout.println(s);
            }
        } catch (Exception ex) {}
        finally {
            try {
                pout.close();
                out.close();
                client.close();
            } catch (IOException ex1) {}
        }
    }
}
```

Задатак 15. **Клијент-сервер**

```
import java.io.*;
import java.net.*;

public abstract class Server implements Runnable {
    static int id = 0;

    public static final int DEFAULTPORT = -1;
    public static final String DEFAULTPROTOCOL = "ServerProtocol1";

    protected String protocol;
    protected String host;
    protected int port;

    public Server() {
        this(DEFAULTPROTOCOL, DEFAULTPORT);
    }

    public Server(int port) {
        this(DEFAULTPROTOCOL, port);
    }

    public Server(String protocol, int port) {
        this.port = port;
        this.protocol = protocol;
        thread = null;
        running = false;
    }

    protected Thread thread;
    protected boolean running;
```

```
    public void start() {
        if (thread == null) {
            thread = new Thread(this, "Server");
            running = true;
            thread.start();
        }
    }

    ServerSocket listener = null;

    public void run() {
        try {
            listener = new ServerSocket(port);
            while (running) {
                try {
                    Socket client = listener.accept();
                    processRequest(client);
                } catch (Exception e) {
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }finally{
            close();
        }
    }

    public void stop() {
        running = false;
        thread.interrupt();
        close();
    }

    public void close() {
        try {
            listener.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public abstract void processRequest(Socket client);

    public boolean isRunning() {
        return running;
    }
}

import java.net.*;

public class OneThreadServer extends Server {

    public OneThreadServer(String protocol, int port) {
        super(protocol, port);
    }

    public void processRequest(Socket client) {
        try {
            WorkingThread tserver = new WorkingThread(id++, client, protocol);
            // tserver.start();
            // tserver.join();
            tserver.run();
        } catch (Exception ex) {
        }
    }
```

```
}


import java.net.*;

public class MultiThreadsServer extends Server {

    public MultiThreadsServer(String protocol, int port) {
        super(protocol, port);
    }

    public void processRequest(Socket client) {
        new WorkingThread(id++, client, protocol).start();
    }

}

import java.net.*;
import java.util.concurrent.*;

public class ExecutorServer extends Server {
    public static final int MAXTHREADS = 10;

    protected ExecutorService pool;

    public ExecutorServer() {
        this(MAXTHREADS, "", DEFAULTPORT);
    }

    public ExecutorServer(int numOfThreads, String protocol, int port) {
        super(protocol, port);
        pool = Executors.newFixedThreadPool(numOfThreads);
    }

    public void processRequest(Socket client) {
        pool.execute(new WorkingThread(0, client, protocol));
    }

    public void stop() {
        super.close();

        pool.shutdown();
        try {
            if (!pool.awaitTermination(60, TimeUnit.SECONDS)) {
                pool.shutdownNow();
                if (!pool.awaitTermination(60, TimeUnit.SECONDS))
                    System.err.println("Pool did not terminate");
            }
        } catch (InterruptedException ie) {
            pool.shutdownNow();
            Thread.currentThread().interrupt();
        }
    }
}

import java.net.*;
import java.util.concurrent.*;

public class ExecutorFutureServer extends ExecutorServer {

    public ExecutorFutureServer(int num, String protocol, int port) {
        super(num, protocol, port);
    }

    public void processRequest(final Socket client) {
```

```
        Future<Void> future = pool.submit(new Callable<Void>() {
            public Void call() {
                WorkingThread wt = new WorkingThread(0, client, protocol);
                wt.work();
                return null;
            }
        });
//      try {
//          future.get();
//      } catch (Exception ex) {
//
//      }
    }
}

public class BufferServer extends Server {
    public static final int MAXTHREADS = 10;

    protected Buffer<Runnable> buffer;

    public BufferServer(int num, String protocol, int port) {
        super(port);
        buffer = new ArrayBuffer<Runnable>();
        for (int i = 0; i < num; i++) {
            Thread t = new BufferWorker(i, buffer);
            t.setDaemon(true);
            t.start();
        }
    }

    public void processRequest(Socket client) {
        buffer.put(new WorkingThread(id++, client, protocol));
    }
}

public class BufferWorker extends Thread {
    Buffer<Runnable> buffer;

    public BufferWorker(int id, Buffer<Runnable> buffer) {
        super("BufferWorker" + id);
        this.buffer = buffer;
    }

    public void run() {
        while (true) {
            Runnable r = buffer.get();
            r.run();
        }
    }
}

public interface Buffer<T> {
    public static final int MAXBUFFERSIZE = 150;
    public T get();
    public void put(T data);
    public void remove(T data);
    public int size();
    public int capacity();
}

import java.util.*;

public class ArrayBuffer<T> implements Buffer<T> {
    private int capacity;
    protected ArrayList<T> arrayBuffer;
```

```
    public ArrayBuffer() {
        this(MAXBUFFERSIZE);
    }

    public ArrayBuffer(int newCapacity) {
        if ((newCapacity > 0) && (newCapacity <= MAXBUFFERSIZE))
            capacity = newCapacity;
        else
            capacity = MAXBUFFERSIZE;
        arrayBuffer = new ArrayList<T>();
    }

    public synchronized T get() {
        while (arrayBuffer.size() == 0) {
            try {
                wait();
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
        T data = arrayBuffer.remove(0);
        notifyAll();
        return data;
    }

    public synchronized void put(T value) {
        while (arrayBuffer.size() == capacity) {
            try {
                wait();
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
        arrayBuffer.add(value);
        notifyAll();
    }

    public synchronized void remove(T data) {
        try {
            int index = arrayBuffer.indexOf(data);
            if (index < 0)
                return;
            arrayBuffer.remove(index);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public synchronized int size() {
        return arrayBuffer.size();
    }

    public synchronized int capacity() {
        return capacity;
    }
}

import java.util.concurrent.*;

public class BlockingBuffer<T> implements Buffer<T> {
    protected BlockingQueue<T> buffer;
    protected int capacity;

    public BlockingBuffer() {
```

```
        this(MAXBUFFERSIZE);
    }

    public BlockingBuffer(int capacity) {
        if ((capacity < 0) || (capacity > MAXBUFFERSIZE)) {
            capacity = MAXBUFFERSIZE;
        }
        buffer = new LinkedBlockingQueue<T>(capacity);
        this.capacity = capacity;
    }

    public T get() {
        while (true) {
            try {
                return buffer.take();
            } catch (InterruptedException e) {
            }
        }
    }

    public void put(T value) {
        while (true) {
            try {
                buffer.put(value);
                return;
            } catch (InterruptedException e) {
            }
        }

    }

    public void remove(T data) {
        buffer.remove(data);
    }

    public int size() {
        return buffer.size();
    }

    public int capacity() {
        return capacity;
    }
}


public interface Message<T> {
    public void setData(T data);
    public T getData();
    public void setMessageID(long messageID);
    public long getMessageID();
}


public class TextMessage implements Message<String> {
    private static final long serialVersionUID = 1L;

    static long id = 0;

    long messageID;
    String text;

    public TextMessage() {
        this("", createID());
    }
```

```
    public TextMessage(String s) {
        this(s, createID());
    }

    public TextMessage(String s, long sentID) {
        messageID = sentID;
        text = s;
    }

    public TextMessage(TextMessage m) {
        this(m.getData(), m.getMessageID());
    }

    public String toString() {
        String s = "";
        s += "Message ID " + messageID + " message: " + text;
        return s;
    }

    public void setData(String data) {
        text = (String) data;
    }

    public String getData() {
        return text;
    }

    public void setMessageID(long messageID) {
        this.messageID = messageID;
    }

    public long getMessageID() {
        return messageID;
    }

    public static synchronized long createID() {
        return id++;
    }

}


import java.net.*;

public class WorkingThread extends Thread {
    protected long id;
    protected String protocol;
    protected Socket client;

    public WorkingThread() {
        this(0, null, "");
    }

    public WorkingThread(long id, Socket client) {
        this(id, client, "");
    }

    public WorkingThread(long id, Socket client, String protocol) {
        super("Working Thread " + id);
        this.id = id;
        this.client = client;
        this.protocol = protocol;
    }

    public void run() {
```

```
        work();
    }

    public void work() {
        try {
            Protocol prot = pf.createProtocol(protocol);
            if (protocol == null)
                return;
            Communicator communicator = getCommunicator();
            communicator.init();
            prot.addCommunicator(communicator);
            prot.conversation();
            communicator.close();

        } catch (Exception ex) {
            System.out.println(ex);
            ex.printStackTrace();
        }
    }

    public Socket getClient() {
        // System.out.println("getClient " + this.getClass());
        return client;
    }

    public Communicator getCommunicator() {
        return new ObjectSocketCommunicator(getClient());
    }

    static ProtocolFactory pf;
    private static Object protocolHandlerLock = new Object();

    static void setProtocolFactory(ProtocolFactory protocol) {
        synchronized (protocolHandlerLock) {
            if (pf != null) {
                throw new Error("factory already defined");
            }
            SecurityManager security = System.getSecurityManager();
            if (security != null) {
                security.checkSetFactory();
            }
            pf = protocol;
        }
    }

    static {
        setProtocolFactory(new ProtocolFactory());
    }
}


public interface Communicator {
    public boolean init();
    public boolean close();
    public <T> Message<T> readMessage() throws CommunicationException;
    public <T> void writeMessage(Message<T> data)
            throws CommunicationException;
    public Object readObject() throws CommunicationException;
    public void writeObject(Object data) throws CommunicationException;
    public String readString() throws CommunicationException;
    public void writeString(String data) throws CommunicationException;
    public byte[] read() throws CommunicationException;
    public void write(byte[] data) throws CommunicationException;
    public void flush() throws CommunicationException;
    public void reset() throws CommunicationException;
```

```
}

public class CommunicationException extends  Exception {
    private static final long serialVersionUID = 1L;

    public CommunicationException() {
        super();
    }

    public CommunicationException(String error) {
        super(error);
    }
}

import java.net.*;
import java.io.*;

import rs.ac.bg.etf.kdp.networking.concurrency.*;

public class ObjectSocketCommunicator implements Communicator {

    protected Socket client;
    protected ObjectOutputStream oout;
    protected ObjectInputStream oin;

    public ObjectSocketCommunicator() {
    }
    public ObjectSocketCommunicator(Socket client) {
        this.client = client;
    }
    public void init(Socket client) {
        this.client = client;
        init();
    }

    public boolean init() {
        boolean ok = true;
        ok = getObjectOutputstream();
        ok |= getObjectInputStream();
        return ok;
    }

    private boolean getObjectInputStream() {
        try {
            InputStream in = client.getInputStream();
            oin = new ObjectInputStream(in);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    private boolean getObjectOutputstream() {
        try {
            OutputStream out = client.getOutputStream();
            oout = new ObjectOutputStream(out);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    public boolean close() {
        boolean ok = true;
        try {
```

```
            oin.close();
        } catch (IOException ex) {
            ok = false;
        }
        try {
            oout.close();
        } catch (IOException ex) {
            ok = false;
        }
        try {
            client.close();
        } catch (IOException ex) {
            ok = false;
        }
        return ok;
    }

    public Object readObject() throws CommunicationException {
        try {
            Object m = oin.readObject();
            return m;
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new CommunicationException();
        }
    }

    public void writeObject(Object m) throws CommunicationException {
        try {
            oout.writeObject(m);
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new CommunicationException();
        }
    }

    @SuppressWarnings("unchecked")
    public <T> Message<T> readMessage() throws CommunicationException {
        return (Message<T>) this.readObject();
    }

    public <T> void writeMessage(Message<T> m) throws CommunicationException {
        this.writeObject(m);
    }

    public String readString() throws CommunicationException {
        return (String) this.readObject();
    }

    public void writeString(String s) throws CommunicationException {
        this.writeObject(s);
    }

    public void flush() throws CommunicationException {
        try {
            oout.flush();
        } catch (Exception e) {
            e.printStackTrace();
            throw new CommunicationException();
        }
    }

    public void reset() throws CommunicationException {
        try {
            oout.reset();
```

```
            oin.reset();
        } catch (Exception e) {
            e.printStackTrace();
            throw new CommunicationException();
        }
    }

    public byte[] read() throws CommunicationException {
        byte[] b = new byte[1024];
        try {
            int num = oin.read(b);
            b = Arrays.copyOf(b, num);
        } catch (Exception e) {
            e.printStackTrace();
            throw new CommunicationException();
        }
        return b;
    }

    public void write(byte[] data) throws CommunicationException {
        try {
            oout.write(data);
        } catch (Exception e) {
            e.printStackTrace();
            throw new CommunicationException();
        }
    }
}

public interface Protocol {
    public boolean endConvresation();

    public boolean startConvresation();

    public void conversation();

    public void addCommunicator(Communicator communicator);

    public boolean removeCommunicator(Communicator communicator);

    public Communicator getCommunicator(int id);
}


public class ProtocolFactory {

    public Protocol createProtocol(String protocolName) {
        Protocol protocol = null;
        String protocolPackageRoot = Protocol.class.getPackage().getName();
        try {
            String clsName = protocolPackageRoot + ".protocol." + protocolName;
            Class<?> cls = null;
            try {
                cls = Class.forName(clsName);
            } catch (ClassNotFoundException e) {
                ClassLoader cl = ClassLoader.getSystemClassLoader();
                if (cl != null) {
                    cls = cl.loadClass(clsName);
                }
            }
            if (cls != null) {
                protocol = (Protocol) cls.newInstance();
            }
        } catch (Exception e) {
```

```
        }
        return protocol;
    }

}


public abstract class OneCommunicatorProtocol implements Protocol {
    protected Communicator communicator;

    public OneCommunicatorProtocol() {
    }

    public OneCommunicatorProtocol(Communicator communicator) {
        this.communicator = communicator;
    }

    public abstract void conversation();

    public boolean endConvresation() {
        return true;
    }

    public boolean startConvresation() {
        return false;
    }


    public void addCommunicator(Communicator communicator) {
        this.communicator = communicator;
    }

    public boolean removeCommunicator(Communicator communicator) {
        if (this.communicator.equals(communicator)) {
            this.communicator = null;
            return true;
        }
        return false;
    }

    public Communicator getCommunicator(int id) {
        return communicator;
    }
}


import rs.ac.bg.etf.kdp.networking.Communicator;
import rs.ac.bg.etf.kdp.networking.concurrency.TextMessage;

public class ServerProtocol1 extends  OneCommunicatorProtocol {

    public ServerProtocol1() {
    }

    public ServerProtocol1(Communicator c) {
        this.communicator = c;
    }

    public void conversation() {

        try {
            String response = communicator.readString();
            System.out.println("received " + response);

            communicator.writeString("Goodbye!");
```

```
            System.out.println("Sent Goodbye! ");

            TextMessage m = (TextMessage) communicator.readObject();
            System.out.println(m);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            communicator.close();
        }

    }
}
import rs.ac.bg.etf.kdp.networking.*;
import rs.ac.bg.etf.kdp.networking.concurrency.*;

public class ClientProtocol1 extends  OneCommunicatorProtocol {

    public ClientProtocol1() {
    }

    public ClientProtocol1(Communicator communicator) {
        this.communicator = communicator;
    }

    public void conversation() {

        try {
            communicator.writeString("Hello!");
            System.out.println("Sent Hello!");

            String response = communicator.readString();
            System.out.println("Received " + response);
            TextMessage msg = new TextMessage("Mesage tekst " + response);
            communicator.writeObject(msg);
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            communicator.close();
        }
    }

}

public class ServerProtocolKnockKnock extends  OneCommunicatorProtocol {
    private static final int WAITING = 0;
    private int state = WAITING;

    private String[] eggs = { "Belo", "Plavo", "Crveno", "Zeleno", "Zuto",
            "Crno" };
    private String[] questions = { "Kuc Kuc", "Djavo 's neba", "Jedno jaje" };
    private String[] answers = { "Koje?", "Sta Vam treba?", "Koje boje?" };
    private String yesAnswer = "Ima";
    private String noAnswer = "Nema ";
    private String byeAnswer = "Zdravo";

    public ServerProtocolKnockKnock() {
        super();
    }

    public void conversation() {
        try {
            String outputLine = null;
            String inputLine = null;
            while ((inputLine = communicator.readString()) != null) {
                System.out.println("Klient: " + inputLine);
```

```
                outputLine = processInput(inputLine);
                communicator.writeString(outputLine);
                System.out.println("Server: " + outputLine);
                if (outputLine.equals("Zdravo"))
                    break;
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            communicator.close();
        }
    }

    public String processInput(String theInput) {
        String theOutput = null;
        if (theInput == null)
            theInput = "";
        switch (state) {
        case 0:
        case 1:
        case 2:
            if (theInput.equalsIgnoreCase(questions[state])) {
                theOutput = answers[state];
                state++;
            } else {
                theOutput = byeAnswer;
                state = 0;
            }
            break;
        case 3:
            theOutput = noAnswer + theInput + " " + answers[2];
            for (int i = 0; i < eggs.length; i++) {
                if (theInput.equalsIgnoreCase(eggs[i])) {
                    theOutput = yesAnswer;
                    state++;
                    break;
                }
            }
            break;
        default:
            theOutput = byeAnswer;
            break;
        }
        return theOutput;
    }
}
import java.io.*;

public class ClientProtocolKnockKnock extends  OneCommunicatorProtocol {

    public ClientProtocolKnockKnock() {
        super();
    }

    public void conversation() {
        String fromServer = "";
        String fromUser = "";
        try {
            BufferedReader stdIn = new BufferedReader(new InputStreamReader(
                    System.in));
            System.out.print("Klient: ");
            fromUser = stdIn.readLine();
            communicator.writeString(fromUser);
            while ((fromServer = communicator.readString()) != null) {
                System.out.println("Server: " + fromServer);
```

```
                if (fromServer.equals("Zdravo"))
                    break;
                System.out.print("Klient: ");
                fromUser = stdIn.readLine();
                if (fromUser != null) {
                    communicator.writeString(fromUser);
                }
            }
            System.out.println("----------------------------------------------");

        } catch (Exception ex) {

        } finally {
            communicator.close();
        }
    }
}


import java.net.*;

public class BufferWorkingServer extends Server {
    public static final int MAXTHREADS = 10;

    protected Buffer<Socket> buffer;

    public BufferWorkingServer() {
        this(MAXTHREADS, "", DEFAULTPORT);
    }

    public BufferWorkingServer(String protocol, int port) {
        this(MAXTHREADS, protocol, port);
    }

    public BufferWorkingServer(int num, String protocol, int port) {
        super(port);
        buffer = new ArrayBuffer<Socket>();
        for (int i = 0; i < num; i++) {
            WorkingThread t = new BufferWorkingThread(i, buffer, protocol);
            t.setDaemon(true);
            t.start();
        }
    }

    public void processRequest(Socket client) {
        buffer.put(client);
    }
}


import java.net.*;

public class BufferWorkingThread extends WorkingThread {

    protected Buffer<Socket> buffer;

    public BufferWorkingThread(long id, Buffer<Socket> buffer, String protocol) {
        super(id, null, protocol);
        this.buffer = buffer;
    }

    public void run() {
        while (true) {
            work();
        }
```

```
    }

    public Socket getClient() {
        return buffer.get();
    }
}

import rs.ac.bg.etf.kdp.networking.*;

public class ServerTest1 {

    public static void main(String[] args) {
        Server server = new BufferServer(10, "ServerProtocolKnockKnock", 8080);
        server.start();
    }
}
import java.net.*;

import rs.ac.bg.etf.kdp.networking.*;

public class ClientTest1 {
    static final int port = 8080;
    static final String host = "127.0.0.1";

    public static void main(String[] args) {
        try {
            for (int i = 0; i < 10; i++) {
                try {
                    Socket server = new Socket(host, port);
                    System.out.println("Iteration " + i);
                    WorkingThread tc = new WorkingThread(i, server,
                            "ClientProtocolKnockKnock");
                    tc.start();
                    tc.join();
                    server.close();
                    Thread.sleep(5000 + (int) (Math.random() * 1000));
                } catch (InterruptedException e) {
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Задатак 16. **The Savings Account problem**

Рачун у банци може да дели више корисника. Сваки корисник може да уплаћује и подиже новац са рачуна под условом да салдо на рачуну никада не буде негативан, као и да види тренутно стање рачуна. Решити проблем користећи удаљене позиве метода у Јави.

```
import java.rmi.*;

public interface Bank extends Remote {

    public UserAccount getUserAccount(String name) throws RemoteException;

}

import java.rmi.*;

public interface UserAccount extends Remote {

    public float getStatus() throws RemoteException;
```

```
    public void transaction(float value) throws RemoteException;

}


import java.rmi.*;
import java.util.*;

public class BankImpl implements Bank {

    private static final long serialVersionUID = 1L;
    private static transient Map<String, UserAccount> users;

    public BankImpl() {
        users = new HashMap<String, UserAccount>();
    }

    public synchronized UserAccount getUserAccount(String name) {
        UserAccount user = users.get(name);
        if (user != null) {
            return user;
        }
        try {
            user = new UserAccountImpl(name);
        } catch (RemoteException e) {
        }
        users.put(name, user);
        return user;
    }
}

import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class UserAccountImpl extends UnicastRemoteObject
        implements UserAccount, Serializable {

    private static final long serialVersionUID = 1L;
    private float status;
    private String name;

    public UserAccountImpl(String name) throws RemoteException {
        this.status = 0;
        this.name = name;
    }

    private void work() {
        try {
            Thread.sleep(500 + (int) (Math.random() * 1000));
        } catch (InterruptedException e) {
        }
    }

    public synchronized float getStatus() {
        work();
        return status;
    }

    public synchronized void transaction(float value) {
        work();
        while (status + value < 0) {
            try {
                wait();
            } catch (Exception ex) {
```

```
            }
        }
        status += value;
        notifyAll();
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

import java.rmi.registry.*;
import java.rmi.server.*;

public class Server {

    public static final String host = "127.0.0.1";
    public static final int port = 8080;

    public static void main(String[] args) {

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        try {
            Bank bank = new BankImpl();
            System.out.println("Kreiran objekat BankImpl");

            Bank stub = (Bank) UnicastRemoteObject.exportObject(bank, 0);
            System.out.println("Kreiran stub BankImpl koji slusa bilo koji port");

            String urlString = "/Bank";
            System.out.println(urlString);

            Registry registry = LocateRegistry.createRegistry(port);
//          Registry registry = LocateRegistry.getRegistry(host, port);
            System.out.println("Definisan Registry port");

            registry.rebind(urlString, stub);
            System.out.println("Baknka je povezana sa imenom");

        } catch (Exception ex) {
            System.out.println("Desila se greska");
            ex.printStackTrace();
        }
    }
}


import java.rmi.registry.*;

public class Client {

    public static final String host = "127.0.0.1";
    public static final int port = 8080;

    public static void main(String[] args) {
        try {
            Bank bank = null;
            UserAccount userAcount = null;
```

```
            String name = args[0];

            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());
            }

            String urlString = "/Bank";
            System.out.println("url = " + urlString);

            Registry registry = LocateRegistry.getRegistry(host, port);
            bank = (Bank) registry.lookup(urlString);
            System.out.println("Bakna ucitana");

            userAcount = bank.getUserAccount(name);
            System.out.println("UserAcount ucitan");

            for (int m = 0; m < 100; m++) {
                float nstatus = (float) (50 + m - (int) (Math.random() * 100));
                try {
                    System.out.println("Status: " + userAcount.getStatus());
                    System.out.println("Promena statusa za " + nstatus);
                    userAcount.transaction(nstatus);
                    System.out
                            .println("Novi status: " + userAcount.getStatus());
                } catch (Exception e) {
                    System.err.println("Greska pri transakciji za " + name);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

java.policy

```
grant {
    // Allow everything for now
    permission java.security.AllPermission;
};
```

За верзије јаве пре 1.5 је неопходно додатно превести удаљене објекте користећи посегам преводилац rmic, који је обично инсталиран у истом директоријуму где и извршно јава окружење, користећи команду: **rmic UserAcountImpl BankImpl**

Покретање:

Уколико је на серверској страни присту регистру био са getRegistry потребно је покренути посебан програм rmiregistry који је обично инсталиран у истом директоријуму где и извршно јава окружење, користећи команду: **rmiregistry [port]**

Серверска страна:
java -Djava.security.policy=putanjaDo/java.policy
 [-Djava.rmi.server.codebase=file:\path/] Server

Клијентска страна:
java -Djava.security.policy=putanjaDo/java.policy
[-Djava.rmi.server.codebase=file:\path/] Client pera

-Djava.security.policy=putanjaDo/java.policy представља подешавање за коришћену полису постављањем путање до места где се налази фајл са додељеним привилегијама. Обавезно ју је потребно поставити.

-Djava.rmi.server.codebase=file:\path/ приликом покретања потребно је назначити, користећи аргумент java.rmi.server.codebase, где су класе којима ће серверска апликација (registry) моћи преко мреже да приступи. Путања треба да обухвати дефиниције удаљених интерфејса, као и свих класа које се референцирају у датим интерфејсима (и тако рекурзивно).