

КОНКУРЕНТНО И ДИСТРИБУИРАНО ПРОГРАМИРАЊЕ

ВЕРЗИЈА 2009 1.0

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	6
Задатак 5. Atomic broadcast problem	7
Задатак 6. Atomic broadcast problem B	8
Задатак 7. Dining philosophers problem	9
Задатак 8. The dining savages problem	11
Задатак 9. The H2O problem	12
Задатак 10. The child care problem	13
Региони	15
Задатак 1. Међусобно искључивање	15
Задатак 2. Условна синхронизација	15
Задатак 3. One lane bridge problem	16
Задатак 4. One lane bridge problem	16
Задатак 5. Dining philosophers problem	17
Задатак 6. Dining philosophers problem	17
Задатак 7. Dining philosophers problem	17
Задатак 8. Dining philosophers problem	18
Задатак 9. Readers – Writers problem	18
Задатак 10. Cigarette Smokers' problem	20
Монитори	22
Задатак 1. Readers – Writers problem	22
Задатак 2. Readers – Writers problem	22
Задатак 3. Прихватник	23
Задатак 4. Тајмер	24
Задатак 5. Тајмер	24
Задатак 6. Producer – Consumer/Bounded Buffer problem	24
Задатак 7. Dining philosophers problem	25
Прослеђивање порука	27
Задатак 1. Синхрона, асинхрона и условна комуникација	27
Задатак 2. Dining Philosophers Problem	28
Задатак 3. Broadcast	29
Задатак 4. Broadcast	30
Задатак 5. Readers – Writers Problem	30
Задатак 6. Game of Life	32
CSP	34
Задатак 1. Семафор	34
Задатак 2. Прослеђивање текста	34
Задатак 3. Прослеђивање текста	34
Задатак 4. Читање картица	34
Задатак 5. Обрада текста	34
Задатак 6. Producer – Consumer	35
Задатак 7. Дељење бројева	35
Задатак 8. Факторијел	35
Задатак 9. Скуп	36
Задатак 10. Скуп	36
Задатак 11. Скуп	36

Задатак 12. Множење матрице	37
Задатак 13. Dining philosophers problem	38
Задатак 14. Израчунавање интеграла	39
CONIC	40
Задатак 1. Копирање поруке.....	40
Задатак 2. Бидирекциони прстен	40
Задатак 3. Обрада података	41
Задатак 4. Bounded Buffer	42
Задатак 5. Dining philosophers problem	43
ADA	45
Задатак 1. Једноелементни прихватник	45
Задатак 2. Комуникација са поузданим везама	45
Задатак 3. Клијент-сервер	46
Задатак 4. Conic to Ada	47
Задатак 5. Readers – Writers problem.....	48
Задатак 6. Producer – Consumer/Bounded Buffer	50
Задатак 7. Cigarette Smokers’ problem	51
Linda.....	54
Задатак 1. Семафори	54
Задатак 2. Dining philosophers problem	54
Задатак 3. Клијент-сервер	54
Задатак 4. Клијент-сервер	54
Задатак 5. Cigarette Smokers’ problem	56
Задатак 6. Readers – Writers problem.....	56
Задатак 7. Проблем избора	57
Задатак 8. Проблем лифтова	58
Java	60
Задатак 1. Условна синхронизација	60
Задатак 2. Dining philosophers problem	61
Задатак 3. Условна синхронизација	63
Задатак 4. Прелазак моста.....	65
Задатак 5. Прелазак моста.....	67
Задатак 6. Producer/Consumer problem.....	69
Задатак 7. Reentrant monitor	70
Задатак 8. Readers – Writers problem.....	70
Задатак 9. Readers – Writers problem.....	72
Задатак 10. Зауостављање	74
Задатак 11. The roller coaster problem	75
Задатак 12. The Santa Claus problem	76
Задатак 13. The bus problem	80
Задатак 14. Chat	83
Задатак 15. Клијент-сервер.....	85
Задатак 16. The Savings Account problem	96

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,n²)

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;
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  mutex : semaphore;
     db : semaphore;
     reader_count : integer;

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

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

begin
    init(mutex, 1);
    init (db, 1);
    reader_count := 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;
  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;
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 ticket: semaphore;
    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;
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;
    l: 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

Плеће људождера једе заједничку вечеру из казана који може да прими М 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 куvara и сачека док кувар не напуни казан. Није дозвољено будити куvara уколико се налази бар мало хране у казану. Користећи семафоре написати програм који симулира понашање људождера и куvara.

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;
      signal (savager)
    end;
  end;
end;

procedure Savage(i : integer);
begin
```

```
while (true) do
begin
  wait(mutex);
  if (servings = 0) then
  begin
    signal (cook);
    wait (savage);
    servings := M;
  end;
  servings := servings - 1;
  GetServingFromPot ;
  signal (mutex);
end;
eat;
end;

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

Задатак 9. 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
  signal (hydroSem);
  signal (hydroSem);
  wait (oxyMutex);
  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 10. 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,n²)

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;
  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
  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
    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 ($0 \leq i \leq 4$). (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
        razmislijaj;
        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;
            end;
        forever;
    end;
end;
```

Задатак 6. Dining philosophers problem

Коментаришите следеће решење проблема филозофа који ручају.

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

```

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 R_i , $i = 1, \dots, m$, и писаца W_j , $j = 1, \dots, n$.

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 табlici свакој варијанти одговара по једна колона, и у њу треба за свако од наведених тврђења уписати Т(ачно) или Н(етачно).

Међусобно искључење је осигурано.

Могуће је узајамно блокирање читалаца и писаца.

Могуће је узајамно блокирање писаца (при $r=0$).

Могуће је 'изгладњивање' читалаца.

Могуће је 'изгладњивање' писаца.

а	б	в
Н	Т	Н
Н	Н	Н
Н	Т	Н
Т	Т	Н
Н	Н	Н

а)

" R_i "

repeat

region v do

begin

await ($w = 0$);

$r := r + 1$

end;

read;

region v do $r := r - 1$;

nekritične operacije;

forever

" W_i "

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;
    некритичне операције;
forever
"Wi"
repeat
  region v do
    begin
      w := w + 1;
      await ((r = 0) and (w = 1))
    end;
    write;
    region v do w := w - 1;
    некритичне операције;
forever
B)
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;
      некритичне операције;
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;
```

```
end;  
write;  
region v do  
begin  
    w := w - 1;  
    return := 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;  
end;  
  
procedure smocker_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;
  end;
procedure smocker_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;
  end;
procedure smocker_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;
  end;
begin
  p.paper := false;
  p.tobacco := false;
  p.matches := false;
  p.ok := false;
  cobegin
    Agent;
    smocker_with_Paper;
    smocker_with_Tobacco;
    smocker_with_Matches;
  coend;
end.
```

МОНИТОРИ

Задатак 1. Readers – Writers problem

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

Решење:

```

readers_and_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:

```

readers_and_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 мониторингске 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 - 1) then not_full.wait;
    slots[tail] := first;
    tail := (tail + 1) mod N;
    slots[tail] := second;
    tail := (tail + 1) mod N;
    size := size + 2;
    if(not_empty.queue) then not_empty.signal;
    if(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 > 1) and (not_full.queue)) then not_full.signal
```

end;

procedure cancel;

begin

```
    size := 0; head := 0; tail := 0;
    while((size > 1) 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 DP;  
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

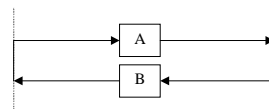
Задатак 1. Синхрона, асинхрона и условна комуникација

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 `ask` 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 ∞ .

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:

<pre> Procedure send(<i>i:integer</i>); var m: msg; begin m := i; mbx_put(m,A); end; </pre>	<pre> procedure receive(var <i>i:integer</i>); var m: msg; st: boolean; begin mbx_get(m,A,8,st); i := m; end; </pre>
---	---

б) Асинхроно слање, условни пријем:

<pre> Procedure send(<i>i:integer</i>); var m: msg; begin m := i; mbx_put(m,A); end; </pre>	<pre> function receive(var <i>i: integer</i>):boolean var m: msg; st: boolean; begin mbx_get(m,A,0,st); if st then i := m; receive := st; end; </pre>
---	--

в) Асинхроно слање, временски условљен пријем:

<pre> procedure send(<i>i:integer</i>); var m: msg; begin m := i; mbx_put(m,A); end; </pre>	<pre> function receive(var <i>i:integer,d:time</i>): boolean; var m: msg; st: boolean; begin mbx_get(m,A,d,st); if st then i := m; receive := st; end; </pre>
---	--

г) Синхроно слање, обичан пријем:

<pre> function send(i: integer):boolean; var m: msg; st: boolean; begin m := i; mbx_put(m,A); mbx_get(m,B,8,st); send := (m = ack); end;</pre>	<pre> procedure receive(var i:integer) var m: msg; st: boolean; begin mbx_get(m,A,8,st); i := m; m := ack; mbx_put(m,B); end;</pre>
--	---

д) Бидирекциона трансакција типа 'захтев-одговор':

<pre> 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;</pre>	<pre> procedure reply; var m: msg; i: integer; st: boolean; begin mbx_get(m,A,8,st); i := m; m := f(i); mbx_put(m,B); end;</pre>
--	--

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, 8, 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], 8, status);
      eat;
      m.id := i;
      m.status := 0;
      mbx_put(m, table);
      mbx_get(n, Phil[i], 8, status)
    end
  end;
begin
  begin
    Philosopher(0);
    Philosopher(1);
    Philosopher(2);
    Philosopher(3);
    Philosopher(4);
    Coordinator
  end
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], 8, 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], 8, 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], 8, 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 : msg;
    status : boolean;

begin
    while (true) do
    begin
        mbx_get(m, operationStart, 8, 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(m, operationEnd, 8, status);
                numReaders := numReaders - 1;
            end;
            mbx_put(m, confirm[m.id]);
            mbx_get(m, operationEnd, 8, status);
        end;
        status := true;
        while(status) do
        begin
            mbx_get(m, operationEnd, 0, status);
        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], 8, 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], 8, status);
  read;
  m.id := i;
  mbx_put(m, operationEnd);
end
end;

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

Задатак 6. Game of Life

Постоји матрица димензија $n \times n$ 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 укоси). 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 msg;

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], 8, 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;
end;
```

CSP

Задатак 1. Семафор

Пројектовати бинарни семафор користећи програмски језик CSP.

Решење:

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

Задатак 2. Прослеђивање текста

Написати програм за процес X који прослеђује знаке добијене од процеса *west* процесу *east*.

$X::*[c: \text{character}; \text{west}?c \rightarrow \text{east}!c]$

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 потпрограма и који прихвата позитиван делитељ и делилац, и враћа њихов целобројни количник и остатак при дељењу.

Решење:

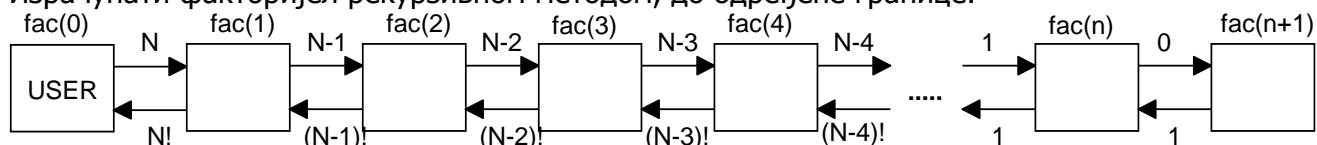
```

[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 (l: 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 $\text{fac}(1)!n$.

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) $S(i)?b \rightarrow \text{skip}$]

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 $IN \times A$. 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.

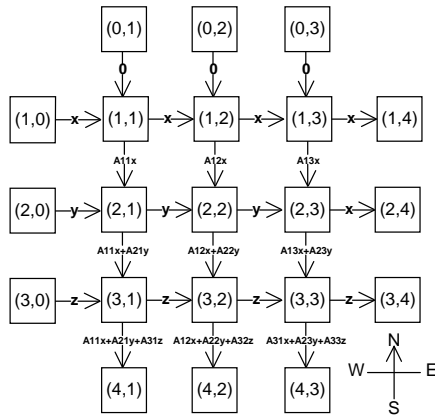
Решење:

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(right)!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,

- креирати три чвора A, B, C и повезати их у бидирекциону прстенасту мрежу,
- специфицирати промену конфигурације под а) насталу креирањем новог чвора D и његовим укључењем у бидирекциони прстен између A и C.
- 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).
- 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;
```

6)

```
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;
```

B)

```
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;
```

r)

```
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;  
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 philosopher 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 ask 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 ∞ 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 C_1, \dots, C_n користи услуге једног сервера S . Клијенти нису међусобно равноправни: ако има више пристиглих захтева, S треба најпре да

услуги клијента C_i 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 C_1 и C_2) решио у језику Conic ?

в) Како би се наведени проблем (за два клијента C_1 и C_2) решио у језику Ada ? (Подразумева се да клијенти упућују захтеве на различите улазе (entry) сервера S : C_1 позива $S.E_1$ (m_1), а C_2 позива $S.E_2$ (m_2)).

Решење:

а)

```
select
  receive  $m_1$  from  $C_1$  => process  $m_1$ ;
or
  receive  $m_2$  from  $C_2$  => process  $m_2$ ;
...
or
  receive  $m_n$  from  $C_n$  => process  $m_n$ ;
```

end select

б)

```
select
  receive  $m_1$  from  $c_1$  → process  $m_1$ ;
or
  receive  $m_2$  from  $c_2$  → process  $m_2$ ;
end select
```

в)

```
select
  accept  $E_1(m_1)$  do process  $m_1$  end;
or
  when  $E_1$ 'count = 0 =>
    accept  $E_2(m_2)$  do process  $m_2$  end;
end select
```

или

```
select
  accept  $E_1(m_1)$  do process  $m_1$  end;
else
  select
    accept  $E_1(m_1)$  do process  $m_1$  end;
  or
    accept  $E_2(m_2)$  do process  $m_2$  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 output
  wait signal => normal_action;
  fail t1 => timeout_action;
end
```

Претпоставимо да је време t_1 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?

6) 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

V (signal): **out** ("sem")

P(wait): **in** ("sem")

Иницијализација на вредност n се врши тако што се out ("sem") изврши n пута.

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);  
        eval (phil (i));  
        if (i<(Num-1)) out ("room ticket");  
    }  
}
```

Задатак 3. Клијент-сервер

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

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 tuple-space-у. 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 *kljenti*, *server* и *sumator* који реализују наведене операције и код за иницијализацију система (ако је потребан) користећи C-Lindu.

Решење:

У tuple space-у се могу наћи следећи tuple-ови:

("broj_servera", **int** n) - укупан број сервера
 ("sumiram") - налази се у tuple space-у када sumator врши сумирање
 ("ne_sumiram") - налази се у tuple space-у када sumator не врши сумирање
 ("index_c", **int** i) - означава редни број следећег захтева који шаље клијент
 ("index_s", **int** i) - означава редни број следећег захтева који узима сервер
 ("zahtev", **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)
 ("odgovor", **int** redni_broj, **int** rez) - одговор на захтев са одговарајућим редним бројем
 ("obradenih", **int** broj_obradenih) - одговор на упит о броју обрађених захтева

```
void client () {
    int i, x, y;
    ...
    in ("index_c", ?i);
    out ("index_c", ++i);
    out ("zahtev", i, x);
    in ("odgovor", i, ?y);
    ...
}

void server () {
    int i=1, obradeni=0, x;
    while (1)
        if (rdp ("sumiram")) {
            out ("obradenih", obradeni);
            rd ("ne_sumiram");
        }
        else {
            in ("index_s", ?i);
            out ("index_s", ++i);
            in ("zahtev", i, ?x);
            out ("odgovor", i, f(x));
            obradeni++;
        }
}

void sumator () {
    int n, ukupno, j;
    ....
    in ("ne_sumiram");
    out ("sumiram");
    rd ("broj_servera", ?n);
    ukupno = 0;
    for (int i = 0; i < n; i++) {
        in ("obradenih", ?j);
        ukupno += j;
    };
    in ("sumiram");
    out ("ne_sumiram")
    ...
}

void inicijalizacija () {
    out ("broj_servera", 10); // na primer
    out ("index_c", 0);
}
```

```
out ("index_s", 0);
out ("ne_sumiram");
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");
    }
}

...
initialize () {
    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);
        in("readers_num", 0);
        writing();
        out("readers_num", 0);
        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 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

```
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;
        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

```
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 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 --;
        bridge.notifyAll();
    }
}
}
```

```
public class Sought extends Car implements Runnable {
```

```
    private Thread thread = null;

    public Sought(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 --;
            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 [] sought = new Car[n];
        Car [] north = new Car[n];

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

        for(int i=0; i < n; i++) {
            sought[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 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 Sought extends Car implements Runnable {

    private Thread thread = null;

    public Sought(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 [] sought = new Car[n];
        Car [] north = new Car[n];

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

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

Задатак 6. Producer/Consumer problem

```
public class OneBuffer {
    private int contents;
    private boolean available = false;
    public synchronized int get() {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        available = false;
        notifyAll();
        return contents;
    }
    public synchronized void put(int value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        contents = value;
        available = true;
        notifyAll();
    }
}

public class Producer extends Thread {
    private OneBuffer onebuffer;
    private int number;
    public Producer(OneBuffer c, int number) {
        onebuffer = c;
        this.number = number;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            onebuffer.put(i);
            System.out.println("Producer #" + this.number + " put: " + i);
        }
    }
}
```

```
        try {
            sleep((int)(Math.random() * 100));
        } catch (InterruptedException e) {}
    }
}
}

public class Consumer extends Thread {
    private OneBuffer onebuffer;
    private int number;
    public Consumer(OneBuffer c, int number) {
        onebuffer = c;
        this.number = number;
    }
    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = onebuffer.get();
            System.out.println("Consumer #" + this.number + " got: " + value);
        }
    }
}

public class ProducerConsumerTest {
    public static void main(String[] args) {
        OneBuffer c = new OneBuffer();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        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 startReading(){
        int PID;
        PID = ID ++;
        while (PID != ok_to_work) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        readers_num ++;
        ok_to_work ++;
        notifyAll();
    }
    public synchronized void endReading(){
        readers_num --;
        notifyAll();
    }
    public synchronized void startWriting(){
        int PID;
        PID = ID ++;
        while ((PID != ok_to_work)|| (readers_num != 0)) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        readers_num = 0;
        notifyAll();
    }
    public synchronized void endWriting(){
        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){
            try {
                Thread.sleep(500+(int)(Math.random()*5000));
            } catch (InterruptedException e) { }
            rw.startReading();
            b.reading(ID);
            rw.endReading();
        }
    }
}

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){
        try {
            Thread.sleep(500+(int)(Math.random()*5000));
        } catch (InterruptedException e) { }
        rw.startWriting();
        b.writing(ID);
        rw.endWriting();
    }
}
}

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 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();  
    }  
}
```

Задатак 11. The roller coaster problem

Проблем вожње тобоганом (*The roller coaster problem*). Претпоставити да постоји n нити које представљају путнике и једна нит каја 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 Elve extends Thread{  
  private SantaClausHouse sc;  
  private int ID;  
  
  public Elve(int ID, SantaClausHouse sc){  
    this.sc = sc;  
    this.ID = ID;  
  }  
  
  private void work(){  
    System.out.println("working " + ID);  
    try {  
      sleep((int)(Math.random() * 10000));  
    } catch (InterruptedException e) { }  
  }  
  
  private void talk(){  
    System.out.println("talking " + ID);  
    try {  
      sleep((int)(Math.random() * 1000));  
    } catch (InterruptedException e) { }  
  }  
}
```

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

```
public class Reindeer extends Thread{
    private SantaClausHouse sc;
    private int ID;

    public Reindeer(int ID, SantaClausHouse sc){
        this.sc = sc;
        this.ID = ID;
    }
    private void rest(){
        System.out.println("on wacation " + ID);
        try {
            sleep((int)(Math.random() * 10000));
        } catch (InterruptedException e) { }
    }
    private void riding(){
        System.out.println("riding " + ID);
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }
    public void run(){
        for(;;){
            rest();
            sc.wakeupSantaR();
            riding();
            sc.exitTheSleigh();
        }
    }
}
```

```
public class SantaClaus extends Thread{
    private int dir;
    private SantaClausHouse sc;
    public static final int elves = 0;
    public SantaClaus(SantaClausHouse sc){
        this.sc = sc;
    }
    private void talk(){
        System.out.println("Santa Claus is talking ");
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }
    private void riding(){
        System.out.println("Santa Claus is riding ");
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
    }
    public void run(){
        for(;;){
            dir = sc.sleeping();
            if(dir == elves){//elves
                talk();
            }
        }
    }
}
```

```
        sc.disposeElves();
    }
    else{
        riding();
        sc.disposeReindeers();
    }
}
}
```

```
public class SantaClausHouse {

    private static final int numReindeer = 9;
    private static final int numElve = 10;
    private static final int minElve = 3;
    private static final int minReindeer = 9;
    private int elvesAtTheDoor;
    private int reindeerAtTheDoor;
    private int elvesInTheRoom;
    private int reindeerInTheSleigh;
    private boolean wakeupE, wakeupR;
    private boolean enterElves, exitElves;
    private boolean enterReindeers, exitReindeers;
    private boolean isSleeping;

    public SantaClausHouse() {
        elvesAtTheDoor = 0;
        reindeerAtTheDoor = 0;
        elvesInTheRoom = 0;
        reindeerInTheSleigh = 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 == minElve){
            wakeupE = true;
        }
        notifyAll();
        while (!enterElves) {
            try {
                wait();
            } catch (InterruptedException e) {; }
        }
        elvesAtTheDoor --;
        elvesInTheRoom ++;
        if(elvesAtTheDoor == 0) notifyAll();
    }

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

```
elvesInTheRoom --;
if(elvesInTheRoom == 0) notifyAll();

}

public synchronized void wakeupSantaR(){
    while (!isSleeping) {
        try {
            wait();
        } catch (InterruptedException e) {;}
    }
    reindeerAtTheDoor ++;
    if(reindeerAtTheDoor == minReindeer){
        wakeupR = true;
    }
    notifyAll();
    while (!enterReindeers) {
        try {
            wait();
        } catch (InterruptedException e) {;}
    }
    reindeerAtTheDoor --;
    reindeerInTheSleigh ++;
    if(reindeerAtTheDoor == 0) notifyAll();
}

public synchronized void exitTheSleigh(){
    while (!exitReindeers) {
        try {
            wait();
        } catch (InterruptedException e) {;}
    }
    reindeerInTheSleigh --;
    if(reindeerInTheSleigh == 0) notifyAll();
}

public synchronized int sleeping(){
    while (!(wakeupR || wakeupE)) {
        try {
            wait();
        } catch (InterruptedException e) {;}
    }
    isSleeping = false;
    if(wakeupR){
        wakeupR = false;
        enterReindeers = true;
        notifyAll();
        while (reindeerAtTheDoor != 0) {
            try {
                wait();
            } catch (InterruptedException e) {;}
        }
        enterReindeers = false;
        notifyAll();//

        return 1;
    }
    else{
        wakeupE = false;
        enterElves = true;
        notifyAll();
        while (elvesAtTheDoor != 0) {
            try {
                wait();
            } catch (InterruptedException e) {;}
        }
        enterElves = false;
        notifyAll();//
    }
}
```

```
        return 0;
    }

}

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

public synchronized void disposeReindeers(){
    exitReindeers = true;
    notifyAll();
    while (reindeerInTheSleigh != 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 возње 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 К 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 дање. 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) {System.out.println(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 StationLlst 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 StationLlst();
    }
    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();
        }
    }
}
```

```
        permissionToExit();
        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) {
            System.out.println(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) {
            System.out.println(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) {
            ex.printStackTrace();
        } finally {
            try {
                pin.close();
                in.close();
                client.close();
            } catch (IOException ex1) {
                ex1.printStackTrace();
            }
        }
    }
}

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) {
        ex.printStackTrace();
    } finally {
        try {
            pout.close();
            out.close();
            client.close();
        } catch (IOException ex1) {
            ex1.printStackTrace();
        }
    }
}
```

Задатак 15. Клијент-сервер

```
public interface Message {
    public void setData(Object data);
    public Object getData();
    public void setMessageID(long messageID);
    public long getMessageID();
}

public interface Buffer {
    public static final int MAXBUFFERSIZE = 150;

    public Object get();
    public void put(Object data);
    public void remove(Object data);
    public int size();
    public int capacity();
}

import java.util.*;

public class ArrayBuffer implements Buffer{
    private int capacity;
    ArrayList arrayBuffer;
    public ArrayBuffer(){
        capacity = MAXBUFFERSIZE;
        arrayBuffer = new ArrayList();
    }
    public ArrayBuffer(int newCapacity){
        if((newCapacity > 0) && (newCapacity <= MAXBUFFERSIZE )) capacity = newCapacity;
        else capacity = MAXBUFFERSIZE;
        arrayBuffer = new ArrayList();
    }
}
```

```
}

public synchronized Object get() {
    while (arrayBuffer.size() == 0) {
        try {
            wait();
        } catch (InterruptedException ex ) { System.out.println(ex); ex.printStackTrace();}
    }
    Object o = arrayBuffer.remove(0);

    notifyAll();
    return o;
}

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

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

}

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

public synchronized int capacity(){
    return capacity;
}

}

public interface Protocol {
    public TextMessage conversation(TextMessage msg) throws NextMsgException;
    public boolean endConvresation();
    public boolean startConvresation();
    public void conversation();
    public void addCommunicator(Communicator c);
    public boolean removeCommunicator(Communicator c);
    public Communicator getCommunicator(int id);
}

public interface Communicator {

    public boolean init();
    public boolean close();
    public TextMessage readMessage();
    public void writeMessage(TextMessage m);
    public Object readObject();
    public void writeObject(Object m);
    public String readString();
    public void writeString(String s);
    public void flush();

}

public class NextMsgException extends Throwable{

}

public class TextMessage implements Serializable, Message{
    static long ID = 0;
```

```
long messageId;
String text;

public TextMessage(){
    this("", ID++);
}
public TextMessage(String s){
    this(s, ID++);
}
public TextMessage(String s, long sentID){
    messageId = sentID;
    text = s;
}
public TextMessage(TextMessage m){
    this(m.getText(), m.getMessageID());
}

public String toString() {
    String s = "";
    s += "Message ID " + messageId + " message: " + text ;
    return s;
}
public void setData(Object data) {
    if(data instanceof String){
        text = (String)data;
    }
}
public Object getData() {
    return text;
}
public void setMessageID(long messageId) {
    this.messageID = messageId;
}
public long getMessageID() {
    return messageId;
}
public String getText() {
    return text;
}
public void setText(String text) {
    this.text = text;
}
}
import java.net.*;
import java.io.*;

public class SocketCommunicator implements Communicator {

    Socket client;

    InputStream in;
    OutputStream out;

    ObjectOutputStream oout;
    ObjectInputStream oin;

    PrintWriter pout;
    BufferedReader pin;

    public SocketCommunicator(Socket client) {
        this.client = client;
    }

    public SocketCommunicator() {
```

```
}

public boolean init() {
    boolean ok = true;
    try {
        out = client.getOutputStream();
        oout = new ObjectOutputStream(out);
        pout = new PrintWriter(out, true);
    } catch (Exception e) {ok = false;}

    try {
        in = client.getInputStream();
        oin = new ObjectInputStream(in);
        pin = new BufferedReader(new InputStreamReader(in));
    } catch (Exception e) {ok = false;}
    return ok;
}

public boolean close() {
    boolean ok = true;
    try {
        pin.close();
        oin.close();
        in.close();
    } catch (IOException ex) { ok = false; }
    try {
        pout.close();
        out.close();
        oout.close();
        client.close();
    } catch (IOException ex) { ok = false;}
    return ok;
}

public TextMessage readMessage() {
    try {
        TextMessage m = (TextMessage) oin.readObject();
        return m;
    } catch (Exception ex) {
        System.out.println(ex);
        ex.printStackTrace();
    }
    return null;
}

public void writeMessage(TextMessage m) {
    try {
        oout.writeObject(m);
    } catch (Exception ex) {
        System.out.println(ex);
        ex.printStackTrace();
    }
}

public Object readObject() {
    try {
        Object m = (Object) oin.readObject();
        return m;
    } catch (Exception ex) {
        System.out.println(ex);
        ex.printStackTrace();
    }
    return null;
}

public void writeObject(Object m) {
    try {
```



```
        oout.writeObject(m);
    } catch (Exception ex) {
        System.out.println(ex);
        ex.printStackTrace();
    }
}

public String readString() {
    try {
        String s = pin.readLine();
        return s;
    } catch (Exception ex) {
        System.out.println(ex);
        ex.printStackTrace();
    }
    return null;
}

public void writeString(String s) {
    try {
        pout.println(s);
    } catch (Exception ex) {
        System.out.println(ex);
        ex.printStackTrace();
    }
}

public void flush() {}

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

import java.net.*;

public class Server implements Runnable{
    public static final int DEFAULTPORT = 88;
    static int ID = 0;
    int port = 88;
    private Thread thread = null;

    public Server(){
        port = DEFAULTPORT;
    }
    public Server(int port){
        this.port = port;
    }

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

    public void run() {
        try {
            ServerSocket listener = new ServerSocket( port );
            Thread myThread = Thread.currentThread();
            while (thread == myThread) {
                try {
                    Socket client = listener.accept();
                    proceedRequest(client);
                }
            }
        }
    }
}
```

```
        }catch (Exception ex1 ) { System.out.println(ex1); ex1.printStackTrace();}
    }
    listener.close();
}catch (Exception ex ) { System.out.println(ex); ex.printStackTrace();}
}

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

public void proceedRequest(Socket client){
    new TServer(client, ID++).start();
}

public int getPort() {
    return port;
}

public void setPort(int port) {
    this.port = port;
}
}
import java.net.*;

public class OneThreadServer extends Server{

    public OneThreadServer(){

    }

    public void proceedRequest(Socket client){
        try{
            TServer tserver = new TServer(client, ID++);
            tserver.start();
            tserver.join();
        }catch (Exception ex ) { System.out.println(ex); ex.printStackTrace();}
    }
}
import java.net.*;

public class BufferServer extends Server{
    public static final int numThreads = 10;
    Buffer buffer;

    public BufferServer(){
        this(numThreads, new KnockKnockServerProtocol(), DEFAULTPORT);
    }
    public BufferServer(int num){
        this(num, new KnockKnockServerProtocol());
    }
    public BufferServer(Protocol protocol){
        this(numThreads, protocol, DEFAULTPORT);
    }
    public BufferServer(int num, Protocol protocol){
        this(numThreads, protocol, DEFAULTPORT);
    }

    public BufferServer(int num, Protocol protocol, int port){
        super(port);
        buffer = new ArrayBuffer();
        for(int i = 0; i < num; i++){
            TServer t = new BufferTServer(buffer, protocol);
            t.setDaemon(true);
            t.start();
        }
    }
}
```

```
}
public void proceedRequest(Socket client){
    try{
        buffer.put(client);
    }catch (Exception ex ) { System.out.println(ex); ex.printStackTrace();}
}
}
import java.net.*;

public class TServer extends Thread {
    protected long ID;
    protected Protocol protocol;
    protected Communicator communicator;
    protected Socket client;

    public TServer() {
        this(null, 0, new ServerProtocol());
    }
    public TServer(Socket client, long ID) {
        this(client, ID, new ServerProtocol());
    }
    public TServer(Socket client, long ID, Protocol protocol) {
        super("Thread Server" + ID);
        this.ID = ID;
        this.client = client;
        this.communicator = new SocketCommunicator(client);
        this.protocol = protocol;
    }
    public void run() {
        work();
    }

    public void work() {
        try {
            if (protocol == null) return;
            Communicator c = this.getCommunicator();
            c.init();
            protocol.addCommunicator(c);
            protocol.conversation();
            c.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 SocketCommunicator(getClient());
    }
}
import java.net.*;

public class BufferTServer extends TServer{

    Buffer buffer;

    public BufferTServer(){
        buffer = new ArrayBuffer();
    }
    public BufferTServer(Buffer buffer){
```

```
        this.buffer = buffer;
    }
    public BufferTServer(Buffer buffer, Protocol protocol){
        this.buffer = buffer;
        this.protocol = protocol;
    }
    public void run() {
        while(true){
            work();
        }
    }
    public Socket getClient(){
        System.out.println("getClient " + this.getClass());
        return (Socket)buffer.get();
    }
}
import java.net.*;

public class TClient extends TServer{
    public TClient() {
        super(null, 0, new ClientProtocol());
    }
    public TClient(Socket client, long ID) {
        super(client, ID, new ClientProtocol());
    }
    public TClient(Socket client, long ID, Protocol protocol) {
        super(client, ID, protocol);
    }
}

public class OneCommunicatorProtocol implements Protocol{
    Communicator c;

    public OneCommunicatorProtocol(){

    }

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

    public TextMessage conversation(TextMessage msg) throws NextMsgException{ return null;}
    public boolean endConvresation(){return true;}
    public boolean startConvresation(){ return false;}

    public void conversation(){

    }

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

    public boolean removeCommunicator(Communicator c) {
        if(this.c.equals(c)){
            c = null;
            return true;
        }
        return false;
    }
    public Communicator getCommunicator(int id){
        return c;
    }
}

public class ServerProtocol extends OneCommunicatorProtocol{

    public ServerProtocol(){

    }

    public ServerProtocol(Communicator c){
```

```
        this.c = c;
    }

    public void conversation(){

        String response = c.readString();
        System.out.println("received " + response);

        c.writeString("Goodbye!");
        System.out.println("sent Goodbye! ");

        TextMessage m = (TextMessage)c.readObject();
        System.out.println(m);

    }
}

public class ClientProtocol extends OneCommunicatorProtocol{

    public ClientProtocol(){

    }

    public ClientProtocol(Communicator c){
        this.c = c;
    }

    public void conversation(){

        c.writeString("Hello!");
        System.out.println("sent Hello!");

        String response = c.readString();
        System.out.println("received " + response);
        TextMessage m = new TextMessage("Mesage teks" + response);
        c.writeObject(m);
    }

}

import java.io.*;

public class KnockKnockClientProtocol extends OneCommunicatorProtocol{

    public KnockKnockClientProtocol(){
        super();
    }

    public void conversation(){
        String fromServer;
        String fromUser;
        try{
            BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("BufferedReader " + stdIn);
            while ((fromServer = c.readString()) != null) {
                System.out.println("Server: " + fromServer);
                if (fromServer.equals("Bye.")) break;
                fromUser = stdIn.readLine();
                if (fromUser != null) {
                    System.out.println("Client: " + fromUser);
                    c.writeString(fromUser);
                }
            }
        } catch (Exception ex) { System.out.println(ex); ex.printStackTrace();}
    }
}

public class KnockKnockServerProtocol extends OneCommunicatorProtocol{
    private static final int WAITING = 0;
    private static final int SENTKNOCKKNOCK = 1;
```

```
private static final int SENTCLUE = 2;
private static final int ANOTHER = 3;

private static final int NUMJOKES = 5;

private int state = WAITING;
private int currentJoke = 0;

private String[] clues = { "Turnip", "Little Old Lady", "Atch", "Who", "Who" };
private String[] answers = { "Turnip the heat, it's cold in here!",
    "I didn't know you could yodel!",
    "Bless you!",
    "Is there an owl in here?",
    "Is there an echo in here?" };

public KnockKnockServerProtocol(){
    super();
}

public void conversation(){
    String outputLine = processInput(null);
    c.writeString(outputLine);
    String inputLine = null;
    while ((inputLine = c.readString()) != null) {
        System.out.println("Client: " + inputLine);
        outputLine = processInput(inputLine);
        c.writeString(outputLine);
        System.out.println("Server: " + outputLine);
        if(outputLine.equals("Bye.")) break;
    }
}

public String processInput(String theInput) {
    String theOutput = null;
    if(theInput == null) theInput = "";
    if (state == WAITING) {
        theOutput = "Knock! Knock!";
        state = SENTKNOCKKNOCK;
    } else if (state == SENTKNOCKKNOCK) {
        if (theInput.equalsIgnoreCase("Who's there?")) {
            theOutput = clues[currentJoke];
            state = SENTCLUE;
        } else {
            theOutput = "You're supposed to say \"Who's there?!\" " +
                "Try again. Knock! Knock!";
        }
    } else if (state == SENTCLUE) {
        if (theInput.equalsIgnoreCase(clues[currentJoke] + " who?")) {
            theOutput = answers[currentJoke] + " Want another? (y/n)";
            state = ANOTHER;
        } else {
            theOutput = "You're supposed to say \"\" " +
                clues[currentJoke] +
                " who?\"\" " +
                "! Try again. Knock! Knock!";
            state = SENTKNOCKKNOCK;
        }
    } else if (state == ANOTHER) {
        if (theInput.equalsIgnoreCase("y")) {
            theOutput = "Knock! Knock!";
            if (currentJoke == (NUMJOKES - 1))
                currentJoke = 0;
            else
                currentJoke++;
            state = SENTKNOCKKNOCK;
        } else {
            theOutput = "Bye.";
        }
    }
}
```

```
        state = WAITING;
    }
}
return theOutput;
}
}
import java.net.*;

public class ClientTest1{
    static final int port = 88;
    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);
                    TClient tc = new TClient(server, i, new ClientProtocol());
                    tc.start();
                    tc.join();
                    server.close();
                    Thread.sleep(5000+(int)(Math.random()*1000));
                } catch (InterruptedException e) {System.out.println(e); e.printStackTrace();}
            }
        } catch (Exception ex ) { System.out.println(ex); ex.printStackTrace();}
    }
}

public class ServerTest1{

    public static void main(String [] args) {
        Server s = new BufferServer(new ServerProtocol());
        s.start();
    }
}

import java.net.*;

public class ClientTest2{
    static final int port = 88;
    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);
                    TClient tc = new TClient(server, i, new KnockKnockClientProtocol());
                    tc.start();
                    tc.join();
                    server.close();
                    Thread.sleep(5000+(int)(Math.random()*1000));
                } catch (InterruptedException e) {System.out.println(e); e.printStackTrace();}
            }
        } catch (Exception ex ) { System.out.println(ex); ex.printStackTrace();}
    }
}

public class ServerTest2{

    public static void main(String [] args) {
        Server s = new BufferServer(new KnockKnockServerProtocol());
        s.start();
    }
}
```

```
}  
}
```

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.rmi.server.*;  
import java.util.HashMap;  
  
public class BankImpl extends UnicastRemoteObject implements Bank {  
    private static transient HashMap users = new HashMap();  
    public BankImpl() throws RemoteException {}  
    public UserAccount getUserAccount(String name) throws RemoteException {  
        synchronized(this){  
            UserAccountImpl user = (UserAccountImpl)users.get(name);  
            if (user != null) {  
                return user;  
            }  
            user = new UserAccountImpl(name);  
            users.put(name, user);  
            return user;  
        }  
    }  
}  
  
import java.rmi.*;  
import java.rmi.server.*;  
import java.io.Serializable;  
  
public class UserAccountImpl extends UnicastRemoteObject implements UserAccount, Serializable {  
    private float status = 0;  
    private String name;  
    public UserAccountImpl (String name) throws RemoteException {  
        this.name = name;  
    }  
    private void work(){  
        try {
```



```
        Thread.currentThread().sleep(500+(int)(Math.random()*1000));
    } catch (InterruptedException e) {}
}

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

public void transaction(float value) throws RemoteException{
    synchronized(this){
        work();
        while(status + value < 0){
            try{
                wait();
            } catch (Exception ex){}
        }
        status += value;
        notifyAll();
    }
}

import java.rmi.*;
import java.rmi.registry.LocateRegistry;

public class Server {

    public static final int port = 8181;
    public static final String server = "127.0.0.1";

    public static void main(String []args){

        System.setSecurityManager(new RMISecurityManager());

        try {

            LocateRegistry.createRegistry(port);
            System.out.println("Definisani Registry port");

            System.out.println("BankImpl: kreirani Bank");
            Bank dli = new BankImpl();

            System.out.println("BankImpl: povezuje ga sa imenom");
            String urlString = "/" + server + ":" + port + "/Bank";
            System.out.println(urlString);
            Naming.rebind(urlString, dli);
            System.out.println("Bank je sada spremna");
        } catch (Exception ex) { System.out.println("Desila se greska" + ex); ex.printStackTrace();}
    }
}

import java.rmi.*;

public class Client {

    public static final int port = 8181;
    public static final String server = "127.0.0.1";

    public static void main(String []args){
        Bank dl = null;
        UserAccount ua = null;
        String name = args[0];
    }
}
```

```
System.setSecurityManager(new RMISecurityManager());

try {
    String url = "/" + server + ":" + port + "/Bank";
    System.out.println("url = " + url);
    dl = (Bank)Naming.lookup(url);
} catch (Exception e) {
    System.err.println("ERROR Naming.lookup" + e);
    System.exit(-1);
}

try {
    UserAccount = dl.getUserAccount(name);
    System.out.println("Nadjeni podaci za " + name);
} catch (Exception e) {
    System.err.println("ERROR reaching " + name);
    System.exit(-1);
}

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

java.policy

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

Превођење:

```
javac -d . *.java
```

```
rmic UserAccountImpl BankImpl
```

Покретање:

```
start rmiregistry
```

```
java -Djava.security.policy=java.policy Server
```

```
java -Djava.security.policy=java.policy Client pera
```