Региони

Региони

- Програмска парадигма за приступ критичној секцији
- Увођење посебне синтаксе за експлицитно означавање критичних секција
- Имплицитно обезбеђивање међусобног искључивања процеса
- Условни критични регион је критични регион који поред обезбеђивања међусобног искључивања има и механизам за синхронизацију процеса преко (опционих) *await* наредби

Региони

Декларација ресурса одређеног типа се обавља као и декларација сваке друге дељене променљиве:

res: **shared** *type*;

Условни критични регион се синтаксно дефинише на следећи начин (***await*** наредба је опциона и ако се изостави добијамо обичан критични регион):

region res **do**
begin
...
[**await**(condition);]
...
end;

Региони

region res → позивајући процес има ексклузивно право приступа променљивој res

Уколико је променљива *res* структура, може се имплицитно подразумевати да се њеном пољу *field* унутар региона приступа без експлицитног навођења имена структуре;

field := *value*; ⇔ res.field := *value*;

Услов *condition* у **await** наредби представља Булов израз који мора атомски да се изврши и не сме имате бочне ефекте.

- Уколико услов *condition* није испуњен, позивајући процес се блокира и ослобађа ресурс региона (одриче се ексклузивног приступа), како би неки други процес могао да приступи региону.

Региони

- **Ако процес има ексклузивно право приступа више променљивама, одриче се само последњег добијеног приступа (задржава ексклузивно право приступа за остале променљиве).**
- Процесу који је био блокиран на услову *condition* ће бити омогућен поновни приступ региону када тај услов буде испуњен и ниједан други процес тренутно не приступа региону (нема експлицитних провера услова нити експлицитног буђења одређеног процеса).
- Није гарантован ФИФО.

Задаци

Условна синхронизација

Дат је упоредни програм на проширеном Pascal-у:

```
procedure makepoints;
var i: integer;
begin
        for i := 1 to n do
                x := i;
                y := i*i;
        end
end;
```

```
procedure printpoints;
var i: integer;
begin
        for i := 0 to n do
                write('(', x, ', ', y, ')');
        end
end;
```

Жељени излаз програма је низ парова облика:

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

Отклонити временску зависност у датом програму употребом условних критичних региона.

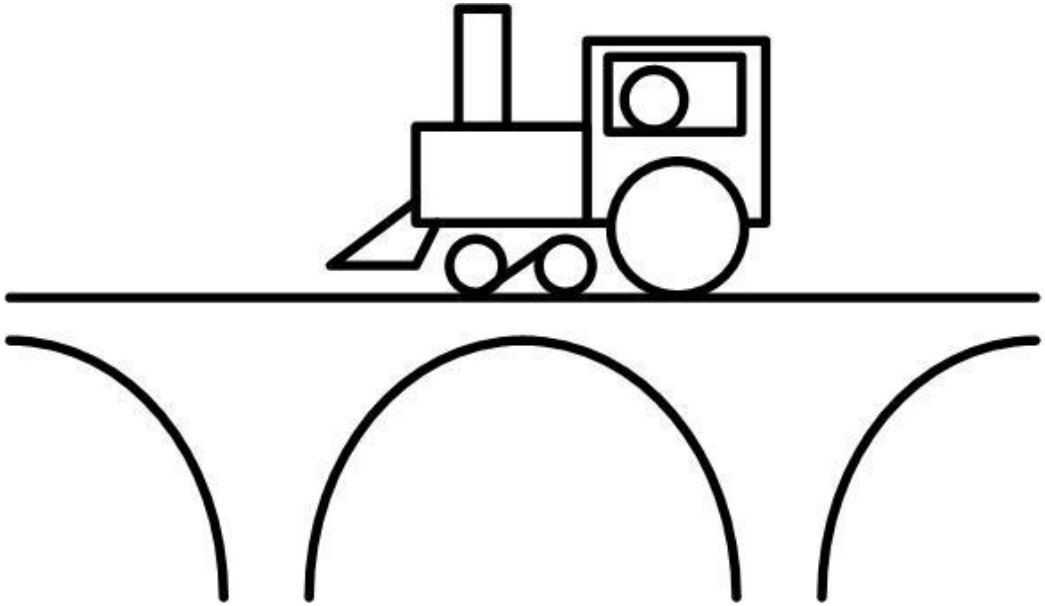
Условна синхронизација

```
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;
```

Условна синхронизација

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


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

One-lane bridge problem



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.

One-lane bridge problem

```
var most: shared record
            juzni, severni: integer
      end
            "u početku 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
```

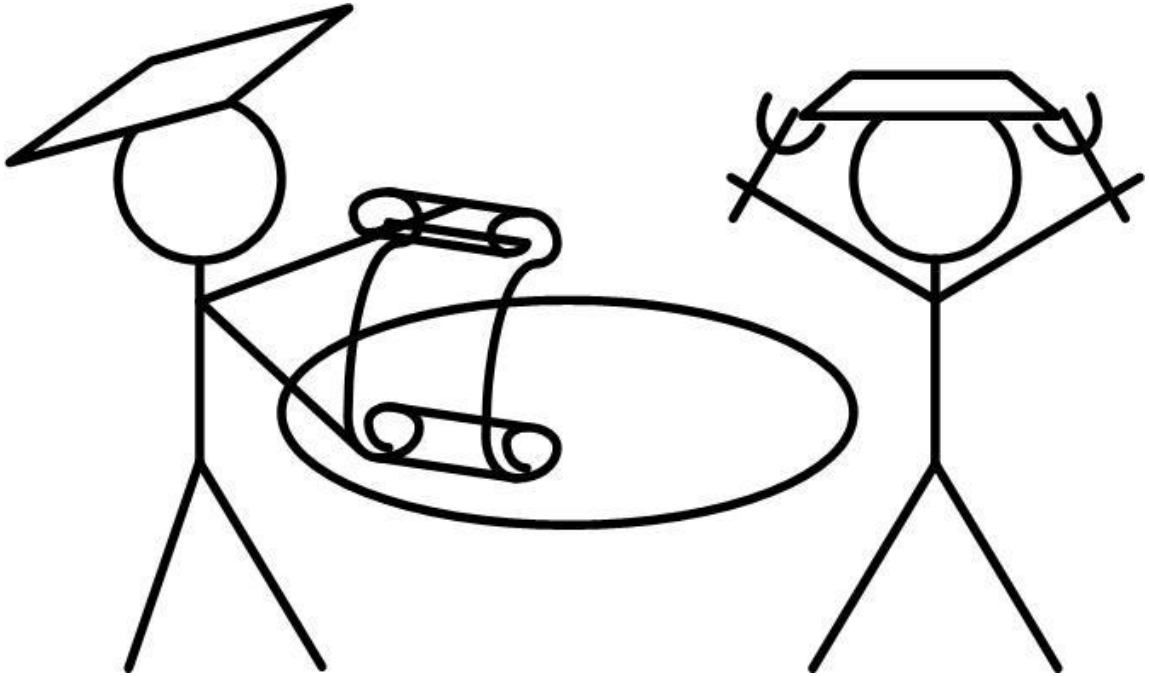
One-lane bridge problem

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.

One-lane bridge problem

```
type smer =          record cekaju, prelaze, ispred: integer; end;
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
```

Dining philosophers problem



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. Прокоментарисати дата решења описаног проблема (исправност, праведност, ...).

Dining philosophers problem - 1

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

dolazi do izgladnjivanja

Dining philosophers problem - 2

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

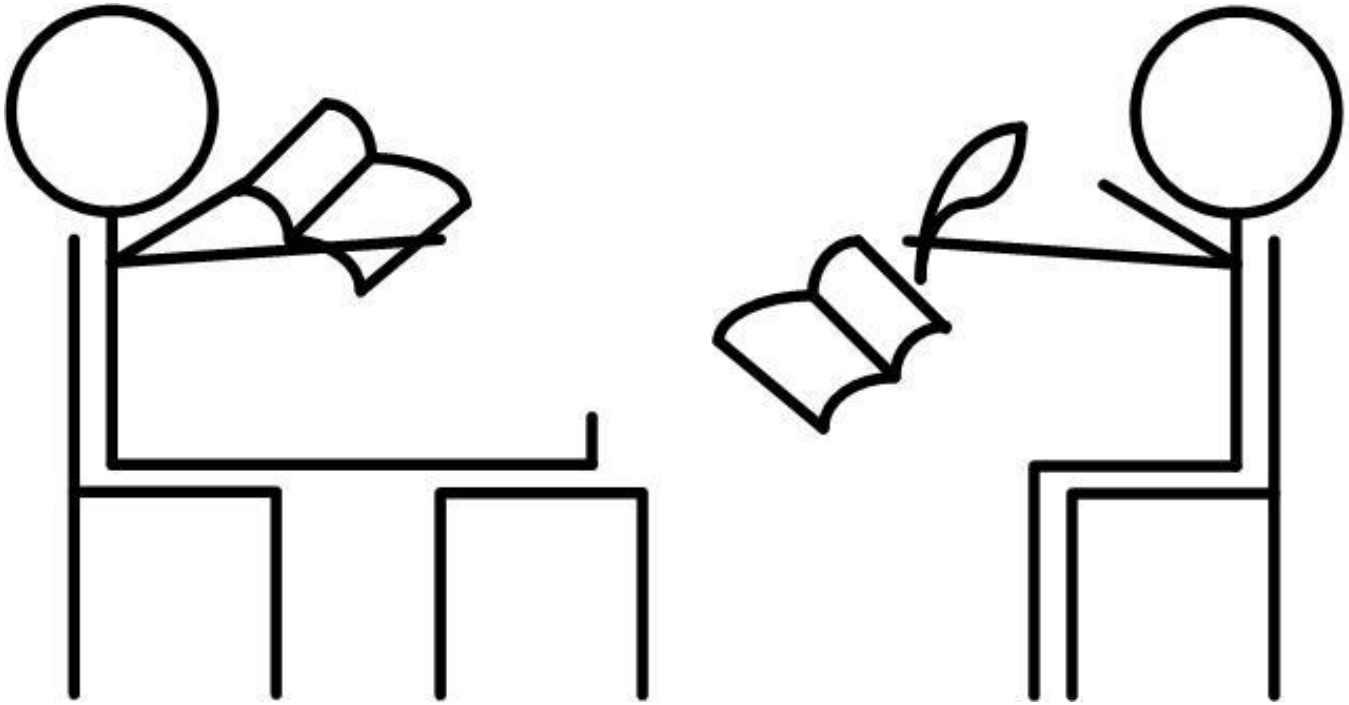
dovodi do deadlocka

Dining philosophers problem - 3

```
var         razmisljanje: shared array [0..4] of boolean;
            "u početku sve je tacno"

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

najbolje bi bilo uvesti FIFO redosled ili uvesti Ticket (ili Petersonov) algoritam - probaj

Readers – Writers problem



Readers – Writers problem

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

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

За сва предложена решења одговорити да ли је:

- Међусобно искључење осигурано.
- Могуће узајамно блокирање читалаца и писаца.
- Могуће узајамно блокирање писаца (при r=0).
- Могуће 'изгладњивање' читалаца.
- Могуће 'изгладњивање' писаца.

Readers – Writers problem - 1

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

Readers – Writers problem - 2

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

Readers – Writers problem - 3

```
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;
```

Readers – Writers problem - 3

```
"Ri"
repeat
    region v do
    begin
        if (rturn) then
        begin
            r := r + 1; await (w = 0)
        end
        else
        begin
            await (w = 0); r := r + 1
        end;
    end;
    read;
    region v do
    begin
        r := r - 1;   rturn := false
    end;
    nekritične_operacije;
forever
```

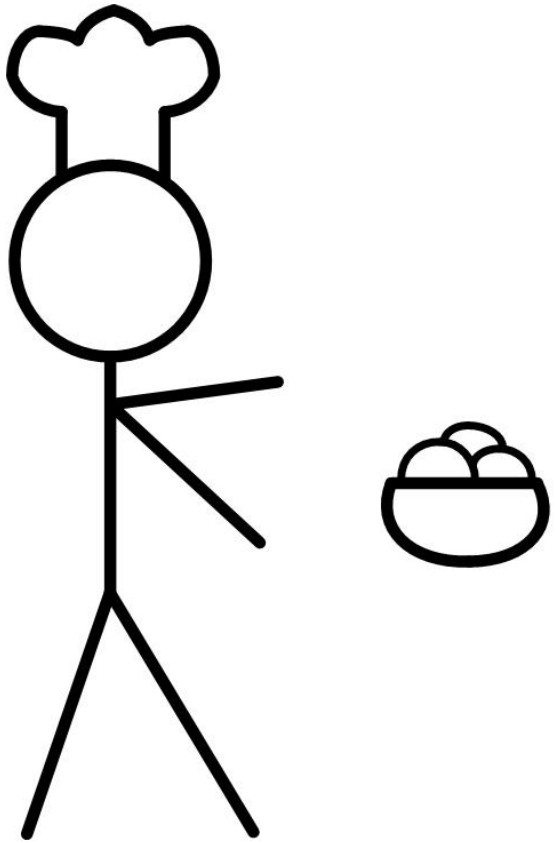
Readers – Writers problem - 3

"Wi"

```
repeat
    region v do
    begin
        if (rturn) then
        begin
            await (r = 0); w := w + 1
        end
        else
        begin
            w := w + 1; await (r = 0)
        end;
    end;
    write;
    region v do
    begin
        w := w - 1; rturn := true
    end;
    nekritične_operacije;
forever
```

moze vise writera??????

Ice Cream Makers problem



Ice Cream Makers problem

Користећи условне критичне регионе написати програм који решава проблем и симулира систем прављења сладоледа (*Ice Cream Makers problem*). Постоји један снабдевач и три сладолеџије. Сладолеџији су потребна три састојка да би направио сладолед – слатка павлака, шећер и ванила. Један сладолеџија има бесконачне залихе слатке павлаке, други шећера и трећи ваниле. Снабдевач има бесконачне залихе сва три састојка. Он на заједнички сто ставља два од три потребна састојка. Сладолеџија коме баш та два састојка фале их узима са стола, прави сладолед, и након што је завршио о томе обавести снабдевача. Снабдевач након тога поново ставља нека два састојка на сто, и циклус се понавља.

Ice Cream Makers problem

```
program IceCreamMaker(input, output);
type table = record
                    cream, sugar, vanilla : boolean;
                    ok : boolean;
             end;
var p: shared table;
```

Ice Cream Makers problem

```
procedure Provider;
var n : integer;
begin
    while (true) do begin
            n := RANDOM(0, 2);
            region p do
            begin
              case n of
               0: begin p.cream := false; p.sugar := true;  p.vanilla := true;  end;
               1: begin p.cream := true;  p.sugar := false; p.vanilla := true;  end;
               2: begin p.cream := true;  p.sugar := true;  p.vanilla := false; end;
               else ;
             await(p.ok);
             p.ok := false;
            end;
    end;
end;
```

Ice Cream Makers problem

```
procedure ice_cream_maker_with_cream;
begin
    while (true) do
        region p do
        begin
            await(p.sugar and p.vanilla);
            p.sugar := false;
            p.vanilla := false;
            enjoy;
            p.ok := true;
        end;
    end;
end;
```

Ice Cream Makers problem

```
procedure ice_cream_maker_with_sugar;
begin
    while (true) do
        region p do
        begin
            await(p.cream and p.vanilla);
            p.cream := false;
            p.vanilla := false;
            enjoy;
            p.ok := true;
        end;
    end;
end;
```

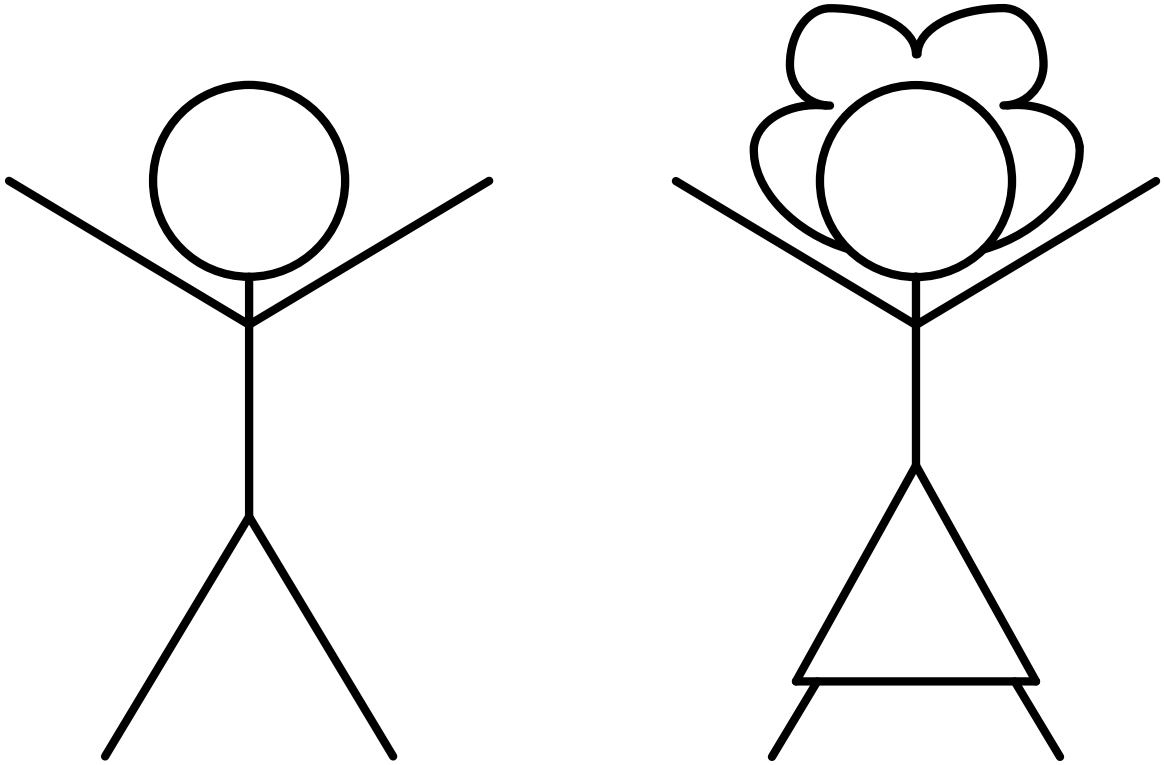
Ice Cream Makers problem

```
procedure ice_cream_maker_with_vanilla;
begin
    while (true) do
        region p do
        begin
            await(p.cream and p.sugar);
            p.cream := false;
            p.sugar := false;
            enjoy;
            p.ok := true;
        end;
    end;
end;
```

Ice Cream Makers problem

```
begin
    p.cream := false;
    p.sugar := false;
    p.vanilla := false;
    p.ok := false;
    cobegin
        Provider;
        ice_cream_maker_with_cream;
        ice_cream_maker_with_sugar;
        ice_cream_maker_with_vanilla;
    coend;
end.
```

Unisex bathroom problem



Unisex bathroom problem

Постоји тоалет капацитета N (N > 1) који могу да користе жене и мушкарци, такав да се у исто време у тоалету не могу наћи и жене и мушкарци. Написати програм за жене и мушкарце који долазе до тоалета, користе га и напуштају га користећи условне критичне регионе. Избећи изгладњавање.

Unisex bathroom problem

```
program UnisexBathroom
var wc: shared record
            turn: integer; // 0 - niko nema prednost, 1 - muskarci, 2 – zene
            cntW, cntM, waitingW, waitingM: integer;
      end;
```

Unisex bathroom problem

```
procedure Women;
begin
    region wc do
    begin
        waitingW := waitingW + 1;
        await (cntM = 0 AND turn != 1 AND cntW < N);
        waitingW := waitingW - 1;
        cntW := cntW + 1;
        if (waitingM > 0) then turn = 1;
    end

    usingToilet;
    region wc do
    begin
        cntW := cntW - 1;
        if (cntW = 0 AND waitingM = 0) then turn := 0;
    end
end
```

Unisex bathroom problem

```
procedure Men;
begin
    region wc do
    begin
        waitingM := waitingM + 1;
        await (cntW = 0 AND turn != 2 AND cntM < N);
        waitingM := waitingM - 1;
        cntM := cntM + 1;
        if (waitingW > 0) then turn = 2;
    end
    usingToilet;
    region wc do
    begin
        cntM := cntM - 1;
        if (cntM = 0 AND waitingW = 0) then turn := 0;
    end
end
```

Unisex bathroom problem

```
Begin
    wc.turn := 0;
    wc.cntW := 0;
    wc.cntM := 0;
    wc.waitingW := 0;
    wc.waitingM := 0;
    cobegin
        Women1;
        ...
        WomenM;
        Man1;
        ...
        ManN;
    coend;
end.
```

Unisex bathroom problem

Усавршити решење претходног задатка тако да особа која пре стигне до тоалета пре и уђе у њега.

Unisex bathroom problem

```
program UnisexBathroom
var wc_fifo: shared record
                cntW, cntM, ticket, next: integer;
     end;
```

Unisex bathroom problem

```
procedure Women;
var
    myTicket: integer;
begin
    region wc_fifo do
    begin
        myTicket := ticket;
        ticket := ticket + 1;
        await (cntM = 0 AND myticket = next AND cntW < N);
        cntW := cntW + 1;
        next := next + 1;
    end

    usingToilet;

    region wc_fifo do
    begin
        cntW := cntW - 1;
    end
end
```

Unisex bathroom problem

```
procedure Men;
var
    myTicket: integer;
begin
    region wc_fifo do
    begin
        myTicket := ticket;
        ticket := ticket + 1;
        await (cntW = 0 AND myticket = next AND cntM < N);
        cntM := cntM + 1;
        next := next + 1;
    end

    usingToilet;

    region wc_fifo do
    begin
        cntM := cntM - 1;
    end
end
```

Unisex bathroom problem

```
Begin
    wc.cntW := 0;
    wc.cntM := 0;
    wc. ticket := 0;
    wc.next := 0;
    cobegin
        Women1;
        ...
        WomenM;
        Man1;
        ...
        ManN;
    coend;
end.
```

Savings Account problem

У банци постоји N (N > 1) различитих рачуна. Рачун у банци може да дели више различитих корисника. Сваки корисник може да види стање рачуна, да уплаћује или подиже одређену суму новца са рачуна, под условом да на рачуну има довољно средстава (никада се не може отићи у минус). Уколико нема довољно новца на рачуну, корисник чека док се новац не уплати. Ниједан корисник не може бити исплаћен док год сви који су пре њега тражили исплату са тог рачуна не добију свој новац. Решити проблем коришћењем региона.

Savings Account problem

```
program SavingsAccount
type account = record
        value, ticket, next: integer;
    end
var
    accounts: shared array [1 .. N] of account

procedure Withdraw(accNum: 1..N, amount: integer);
var
    myTicket: integer;
begin
    region accounts[accNum] do
    begin
        myTicket := ticket;
        ticket := ticket + 1;
        await (amount <= value AND myticket = next);
        value := value - amount;
    end
end
```

Savings Account problem

```
procedure Payment(accNum: 1..N, amount: integer);
begin
    region accounts[accNum] do
    begin
            value := value + amount;
    end
end

function CheckBalance(accNum: 0..N): integer;
begin
    region accounts[accNum] do
    begin
            balance := value;
    end
     CheckBalance := balance;
end
```

Savings Account problem

```
begin
    for i := 0 to N do
    begin
        accounts[i].value := 0;
        accounts[i].ticket := 0;
        accounts[i].next := 0;

    end;
    cobegin
        Payment( p1, a1);
        ...
        Payment( pm, am);
        Withdraw( p1, a1);
        ...
        Withdraw( pm, am);
        CheckBalance( p1);
        ...
        CheckBalance( pm);
    coend;
end.
```

Питања?

Захарије Радивојевић, Сања Делчев, Тамара Шекуларац

Електротехнички Факултет

Универзитет у Београду

zaki@etf.rs, sanjad@etf.rs, tasha@etf.bg.ac.rs