Дистрибуирано програмирање

Прослеђивање порука

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

  смешта поруку m у сандуче box
- *mbx_get*(var m: msg, box: mbx, t: time, var status: boolean)

  узима прву поруку из сандучета box и њену вредност додељује променљивој m, постављајући статус на true; ако је сандуче празно током интервала t, статус постаје false, а вредност m је недефинисана. Време t је у опсегу 0..maxtime или је INF.
- Размотримо једноставан систем који садржи само два процеса, S и R.

Прослеђивање порука

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

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

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

Прослеђивање порука

(б) Асинхроно слање, условни пријем:

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

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

Прослеђивање порука

(в) Асинхроно слање, временски условљен пријем:

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

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

Прослеђивање порука

(г) Синхроно слање, обичан пријем:

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

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

Прослеђивање порука

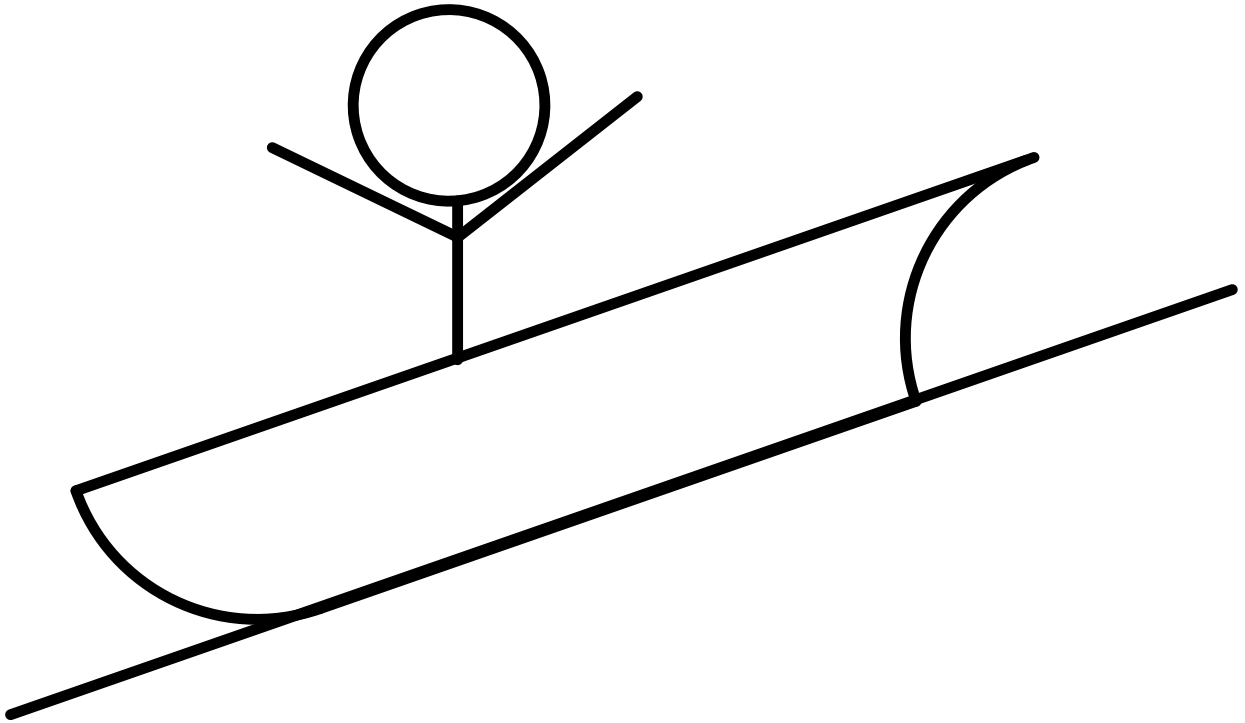
(д) Бидирекциона трансакција типа 'захтев-одговор':

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

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

Задаци

The roller coaster problem



The roller coaster problem

Претпоставити да постоји N путника и једно возило на тобогану (*The roller coaster problem*). 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 користећи асинхрону комуникацију користећи сандучиће који симулира описани систем.

The roller coaster problem

```
program RollerCoaster;
const K = ...;
    N = ...;
type msg = record
            ID : integer;
    end;

var     coasterIN, coasterOUT : mbx;
    passengerBox : array [1..N] of mbx;
```

The roller coaster problem

```
procedure Passenger(ID : integer);
begin
    while true do
    begin
        walking(ID);
        boardCar (ID);
        riding(ID);
        leaveCar (ID);
    end;
end;
```

The roller coaster problem

```
procedure boardCar (ID : integer);
var       m : msg;
    status : boolean;
begin
    m.ID := ID;
    mbx_put(m, coasterIN);
    mbx_get(m, passengerBox[ID], INF, status);
end;
procedure leaveCar (ID : integer);
var       m : msg;
    status : boolean;
begin
    mbx_get(m, passengerBox[ID], INF, status);
    m.ID := ID;
    mbx_put(m, coasterOUT);
end;
```

The roller coaster problem

```
procedure Coaster;
var     i : integer;
    boarded : array[1..K] of msg;

begin
    while true do
    begin
        boardingCar;
        riding;
        leaveingCar;
    end;
end;
```

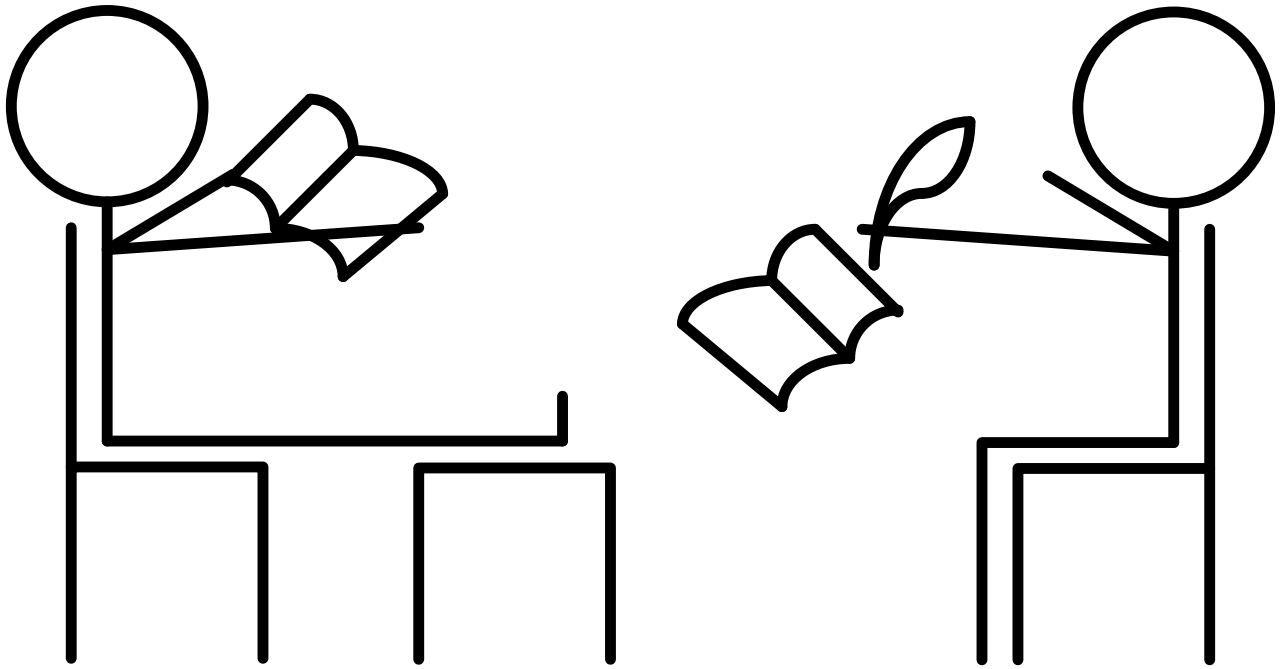
The roller coaster problem

```
procedure boardingCar;
var       i : integer;
    m : msg;
    status : boolean;
begin
    for i := 1 to K do
    begin
            mbx_get(m, coasterIN, INF, status);
            boarded[i] := m;
    end;
    for i := 1 to K do
    begin
            mbx_put(m, passengerBox[boarded[i].ID]);
    end;
end;
```

The roller coaster problem

```
procedure leavingCar;
var i : integer;
    m : msg;
    status : boolean;
begin
    for i := 1 to K do
    begin
        mbx_put(m, passengerBox[boarded[i].ID]);
    end;
    for i := 1 to K do
    begin
        mbx_get(m, coasterOUT, INF, status);
    end;
end;
```

Readers – Writers Problem



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.

Readers – Writers Problem

```
program ReadersWriters;

const STARTREAD = 0;
const STARTWRITE = 1;

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

Readers – Writers Problem

```
procedure Reader(i : integer);
var         m: msg;
    status : boolean;
procedure read; begin  end;
begin
    while (true) do
    begin
            m.id := i;
            m.operation := STARTREAD;
            mbx_put(m, operationStart);
            mbx_get(m, confirm[i], INF, status);
            read;
            m.id := i;
            mbx_put(m, operationEnd);
    end
end;
```

Readers – Writers Problem

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

Readers – Writers Problem

```
program ReadersWriters;

const STARTREAD = 0;
const STARTWRITE = 1;

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

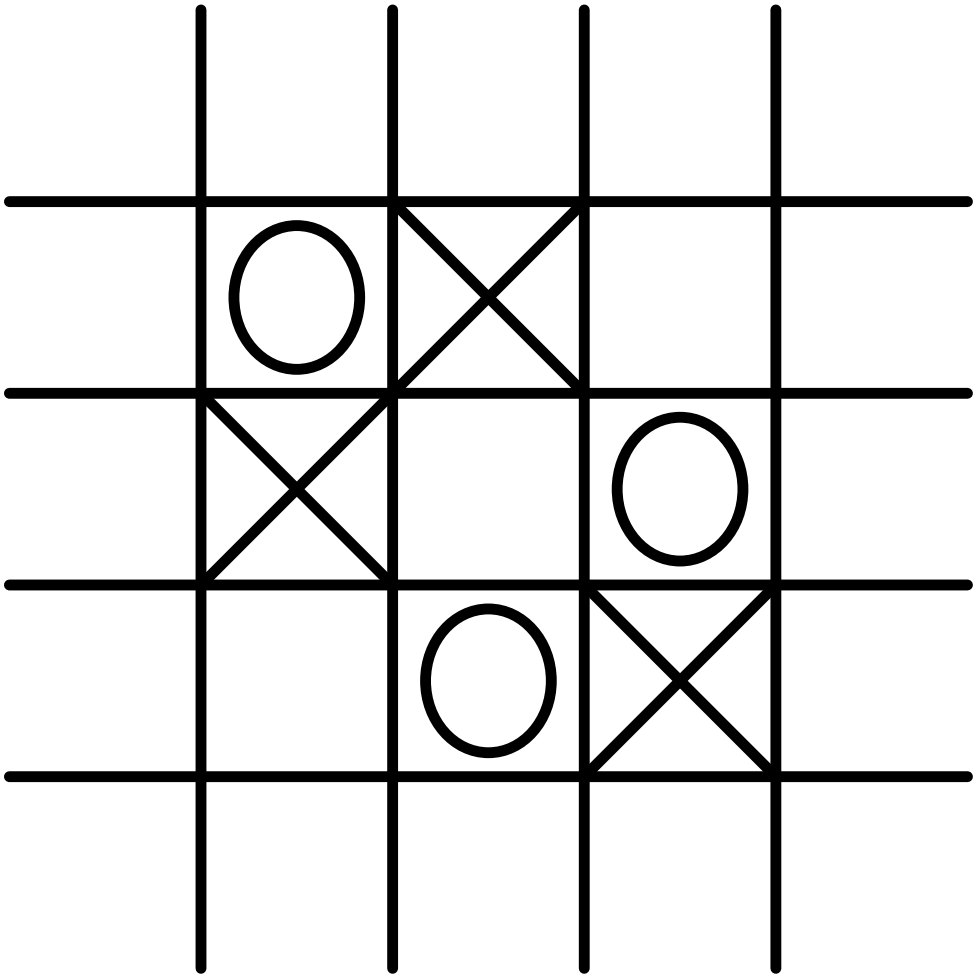
Readers – Writers Problem

```
procedure Coordinator;
    var
        numReaders: integer;
        numWriters: integer;
        m, n : msg;
      status : boolean;
begin
    while (true) do
    begin
        mbx_get(m, operationStart, INF, status);
        if(m.operation = STARTREAD) then
        begin
            numReaders := numReaders + 1;
            mbx_put(m, confirm[m.id]);
        end
```

Readers – Writers Problem

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

Game of Life



Game of Life

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

Жив организам који има мање од два жива суседа умире од усамљености

Жив организам који има више од три жива суседа умире од пренатрпаности

Жив организам са два или три жива суседа преживљава и формира следећу генерацију

Мртав организам са три жива суседа оживљава

Користећи сандучиће написати програм који симулира организам.

Game of Life

```
program GameOfLIfe;
const    numGenerations = ...;
     n = ...;
var        box : array [0..n-1, 0..n-1] of mbx;

function xStart(i : integer) : integer;
begin
     i := i - 1;
     if(i < 0) then xStart := 0;
     else xStart := i;
end;
...
function numOfNeighbours(i, j : integer) : integer;
begin
     numOfNeighbours := (xEnd(i) - xStart(i) + 1)*(yEnd(i) - yStart(i) + 1) - 1;
end;
```

Game of Life

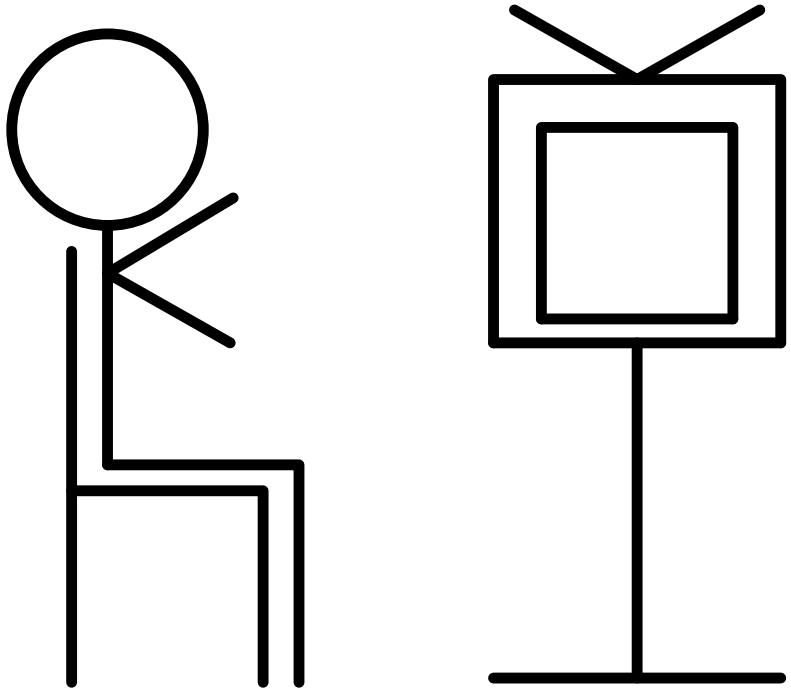
```
procedure Node(i : 1..n, j : 1..n);
var         p, q, k : integer;
      status, st : boolean;
      neighbours : array[0..1, 0..7] of msg;
      num : array[0..1] of integer;
      m : msg;
begin
      num [0] := 0;    num [1] := 0;
      for k := 1 to numGenerations do
      begin
            m.status := status;
            m.i := i;
            m.j := j;
            m.index := k;
```

Game of Life

```
            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, g]);
                                    end;

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

Broadcast



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.

Broadcast

```
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;
```

Broadcast

```
begin
    init(p);
    mbx_get(m, probe[p], IN, st) ;
    //send m to all neighbors
    for q := 1 to n do
        if(links[q]) then mbx_put(m, probe[q]);
    //receive num-1 redundant copies of m
    for q = 1 to num-1 do
        mbx_get(m, probe[p], INF, st) ;
end;
procedure Initiator; //executed on source node S
var     m : msg; //message to broadcast;
    S : integer;
begin
    mbx_put(m, probe[S]);
end;
```

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.

Broadcast

```
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;
```

Broadcast

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

Broadcast

```
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;
```

Питања?

Захарије Радивојевић
Електротехнички Факултет
Универзитет у Београду
zaki@etf.rs