Семафори

Семафори

- Семафор је ненегативна целобројна променљива на којој су дефинисане операције:
- **wait(*s*)** испитује да ли вредност семафора *s* задовољава услов *s>0* и ако је услов задовољен онда се декрементира вредност семафора (*s=s-1*), а ако није, онда се чека док се тај услов не испуни. Провера задовољености услова *s>0* и декрементирање вредности семафора се обављају атомски.
- **signal(*s*)** инкрементира вредност семафора (*s=s+1*). Инкрементирање вредности семафора се обавља атомски.
- **init(*s, val*)** поставља почетну вредност (*s= val*)

Семафори – једна имплементација

<u>wait(s)</u>

if s>0 **then**

s=s-1

else begin

Заустави процес и стави га у стање чекања код ОС

Стави процес у ред чекања на семафору s

Ослободи процесор

end

Семафори – једна имплементација

signal(s)

if ред чекања на семафору s празан (односно s==0) **then**

s=s+1

else begin

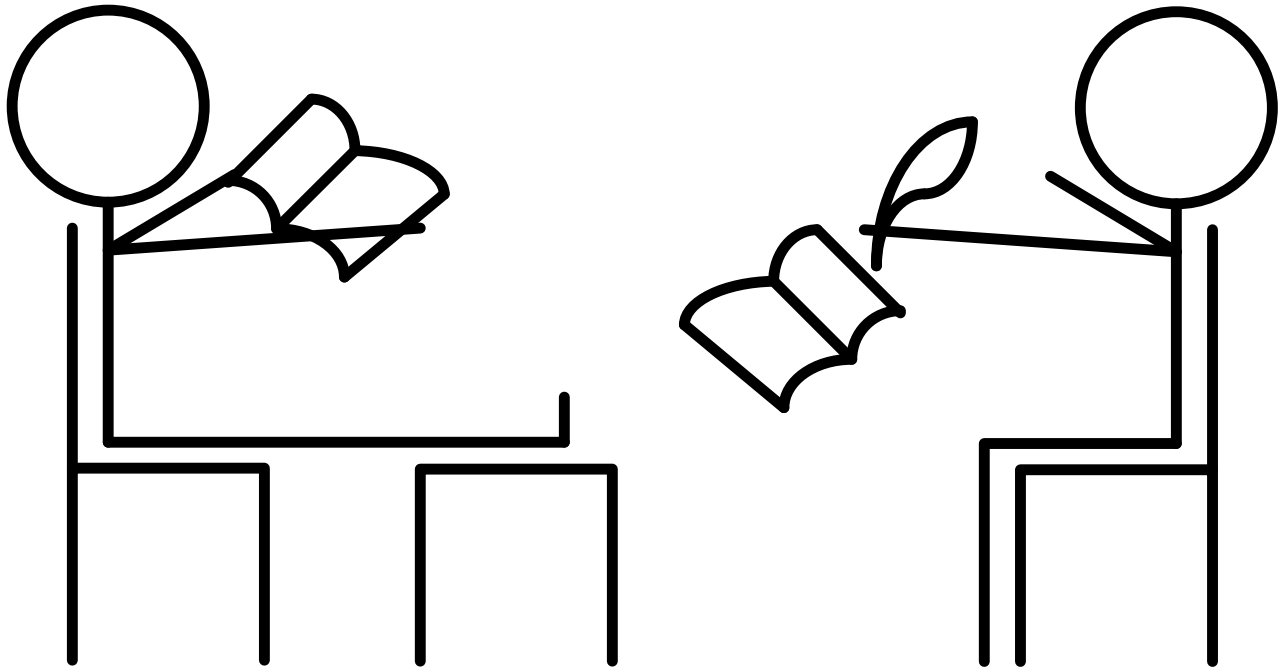
Уклони први процес из реда чекања на семафору s

Активирај тај процес код ОС

end

Задаци

Readers – Writers problem



Readers – Writers problem - 1

```
program Readers_Writers;
var db, mutexR : semaphore;
    readerCount : integer;
procedure Reader(ID : integer);
begin
    while (true) do
    begin

        wait(mutexR);
         readerCount := readerCount + 1;
        if (readerCount = 1) then wait(db);
        signal(mutexR);

        readData();

        wait(mutexR);
         readerCount := readerCount - 1;
        if (readerCount = 0) then signal(db);
        signal(mutexR);
    end
end;
```

Readers – Writers problem - 1

```
procedure Writer(ID : integer);
begin
    while (true) do
    begin
        createData();

        wait(db);

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

Readers – Writers problem - 1

```
program Readers_Writers;
var db, mutexR, in : semaphore;
    readerCount : integer;
procedure Reader(ID : integer);
begin
    while (true) do
    begin
        wait(in);
        wait(mutexR);
         readerCount := readerCount + 1;
        if (readerCount = 1) then wait(db);
        signal(mutexR);
        signal(in);
        readData();

        wait(mutexR);
         readerCount := readerCount - 1;
        if (readerCount = 0) then signal(db);
        signal(mutexR);
    end
end;
```

Readers – Writers problem - 1

```
procedure Writer(ID : integer);
begin
    while (true) do
    begin
        createData();
        wait(in);
        wait(db);
        writeData();
        signal(db);
        signal(in);
    end;
end;
begin
    init (db, 1);
    init(mutexR, 1);
    readerCount := 0;
    init (in, 1);
    cobegin
        Writer(0);
        ...
        Reader(0);
        ...
    coend;
end.
```

Producer – Consumer problem

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;
```

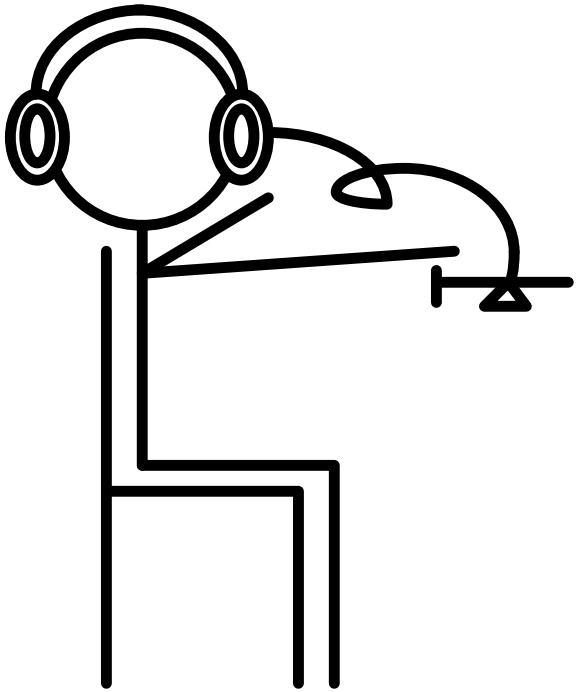
Producer – Consumer problem

```
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;
```

Producer – Consumer problem

```
begin
    init(mutex,1);
    init(empty, BufferSize);
    init(full, 0);
    cobegin
        Producer(0);
        ...
        Consumer(0);
        ...
    coend;
end.
```

Atomic broadcast problem



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.

Atomic broadcast problem

```
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;
```

Atomic broadcast problem

```
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;
```

Atomic broadcast problem

```
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.
```

Atomic broadcast problem

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.

Atomic broadcast problem

```
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;
```

Atomic broadcast problem

```
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;
```

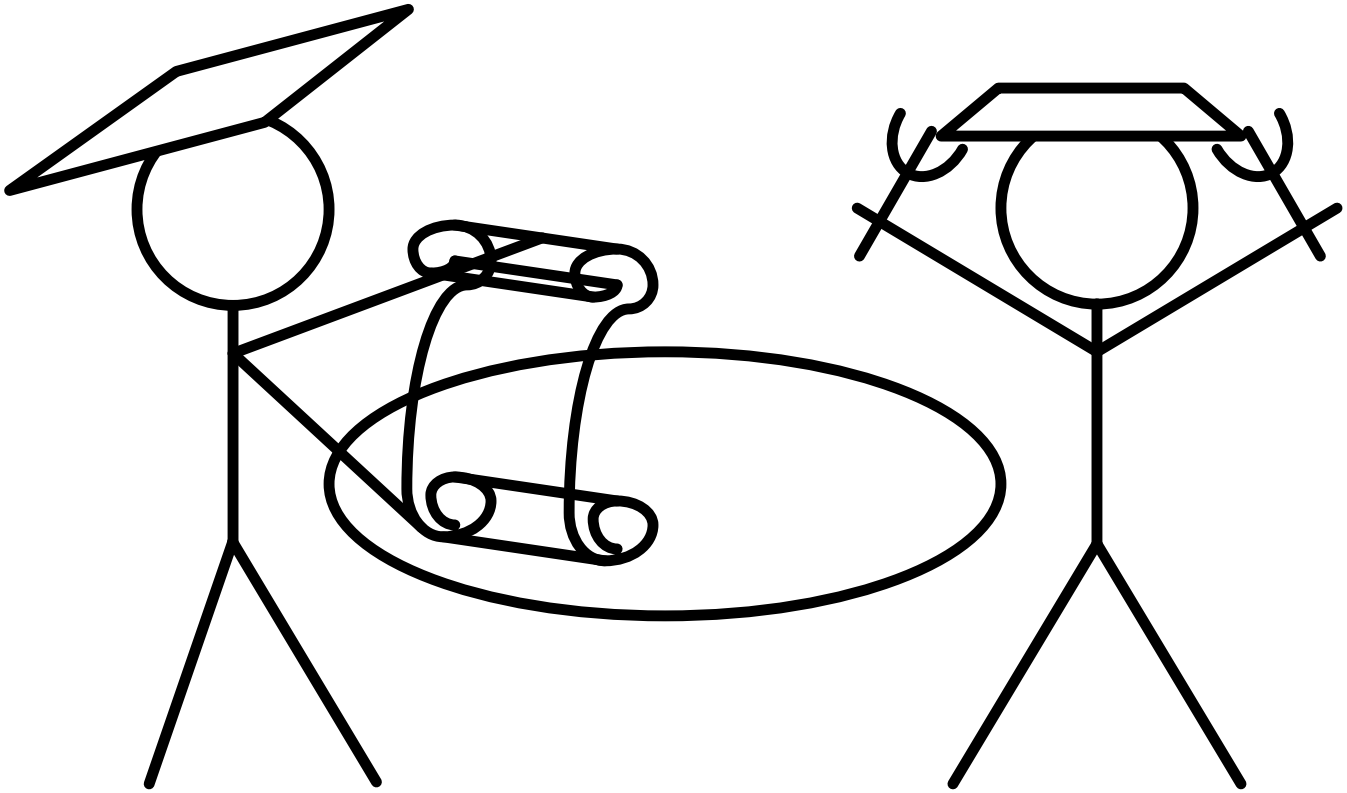
Atomic broadcast problem

```
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;
```

Atomic broadcast problem

```
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;
    writeToIndex := 1;

    cobegin
        Producer;
        Consumer(1);
        ...
        Consumer(N);
    coend;
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. Написати алгоритам за филозофа (0<=i<=4).

Dining philosophers problem - 1

```
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;
```

Dining philosophers problem - 1

```
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;
```

Dining philosophers problem - 1

```
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.
```

Dining philosophers problem - 2

```
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;
```

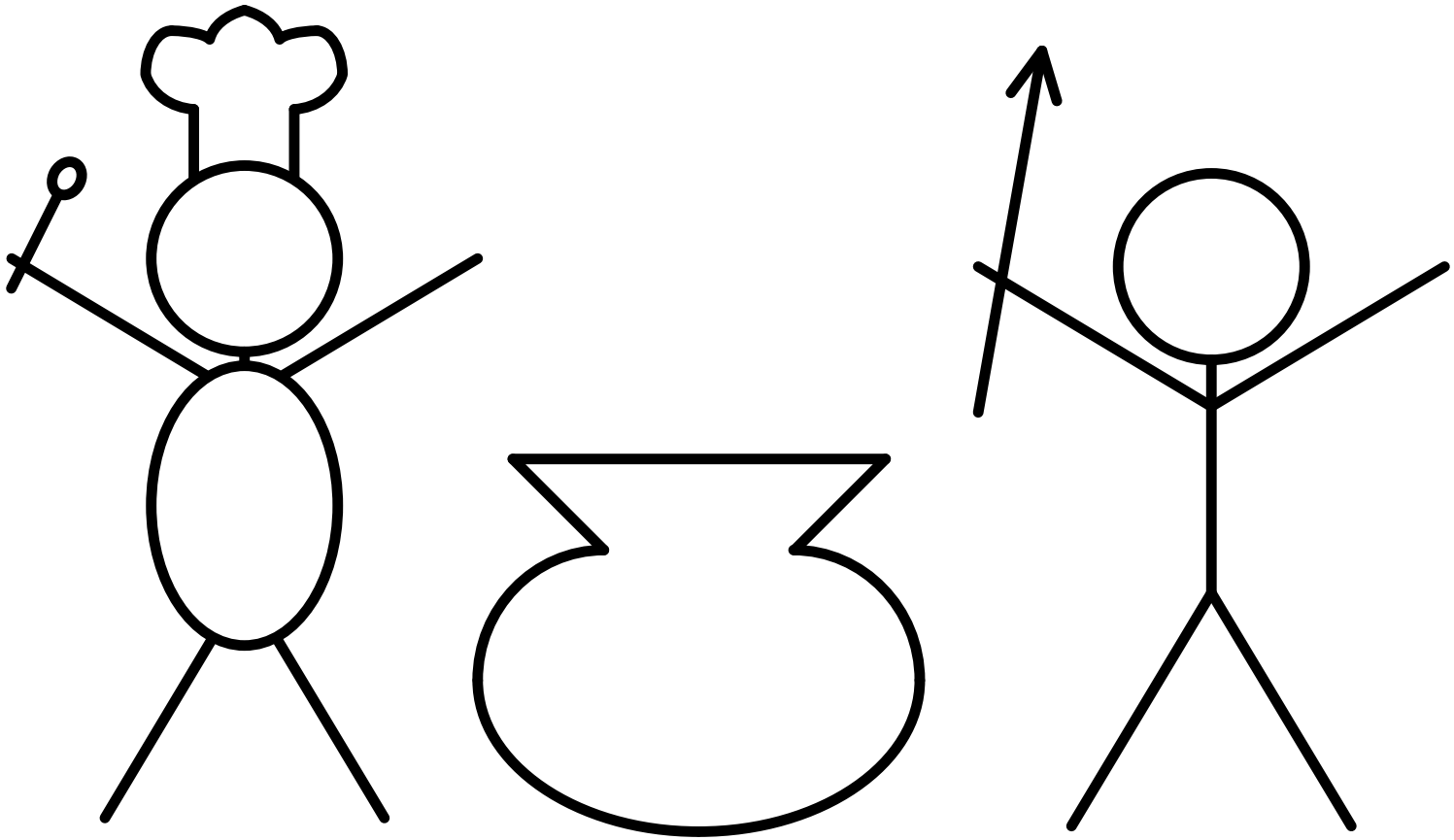
Dining philosophers problem - 2

```
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;
```

Dining philosophers problem - 2

```
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.
```

The dining savages problem



The dining savages problem

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

The dining savages problem

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

procedure PrepareLunch; begin …end
procedure GetServingFromPot; begin …end

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

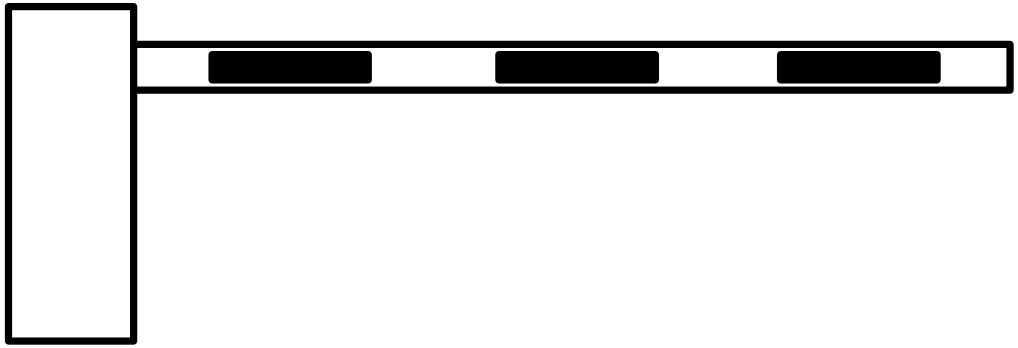
The dining savages problem

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

The dining savages problem

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

Barrier Synchronization



Barrier Synchronization

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

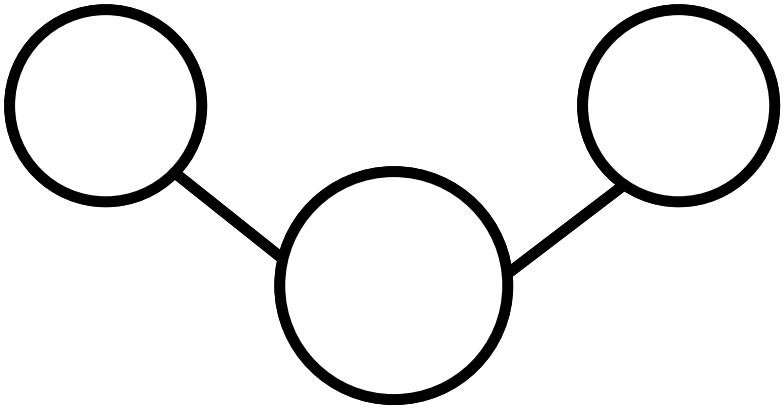
Barrier Synchronization

```
program Barrier(input, output);
const      N = ...;
var barrier1, barrier2 : Semaphore;
     cnt : Integer;
procedure Pass(id : integer);
begin
     wait(barrier1);
     cnt := cnt + 1;
     if(cnt = N) then
              signal(barrier2)
     else
              signal(barrier1);

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

Barrier Synchronization

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

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

The H2O problem



The H2O problem

Постоје два типа атома, водоник и кисеоник, који долазе до баријере. 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.

The H2O problem

```
program H2O;
var
    hydroSem : Semaphore;
    hydroSem2 : Semaphore;
    hydroMutex : Semaphore;
    oxySem : Semaphore;
    oxyMutex : Semaphore;
    count : integer;

procedure Oxygen(i : integer);
begin
    wait (oxyMutex);
    signal (hydroSem);
    signal (hydroSem);
    wait (oxySem);
    bond (i);
    signal (oxyMutex);
end;
```

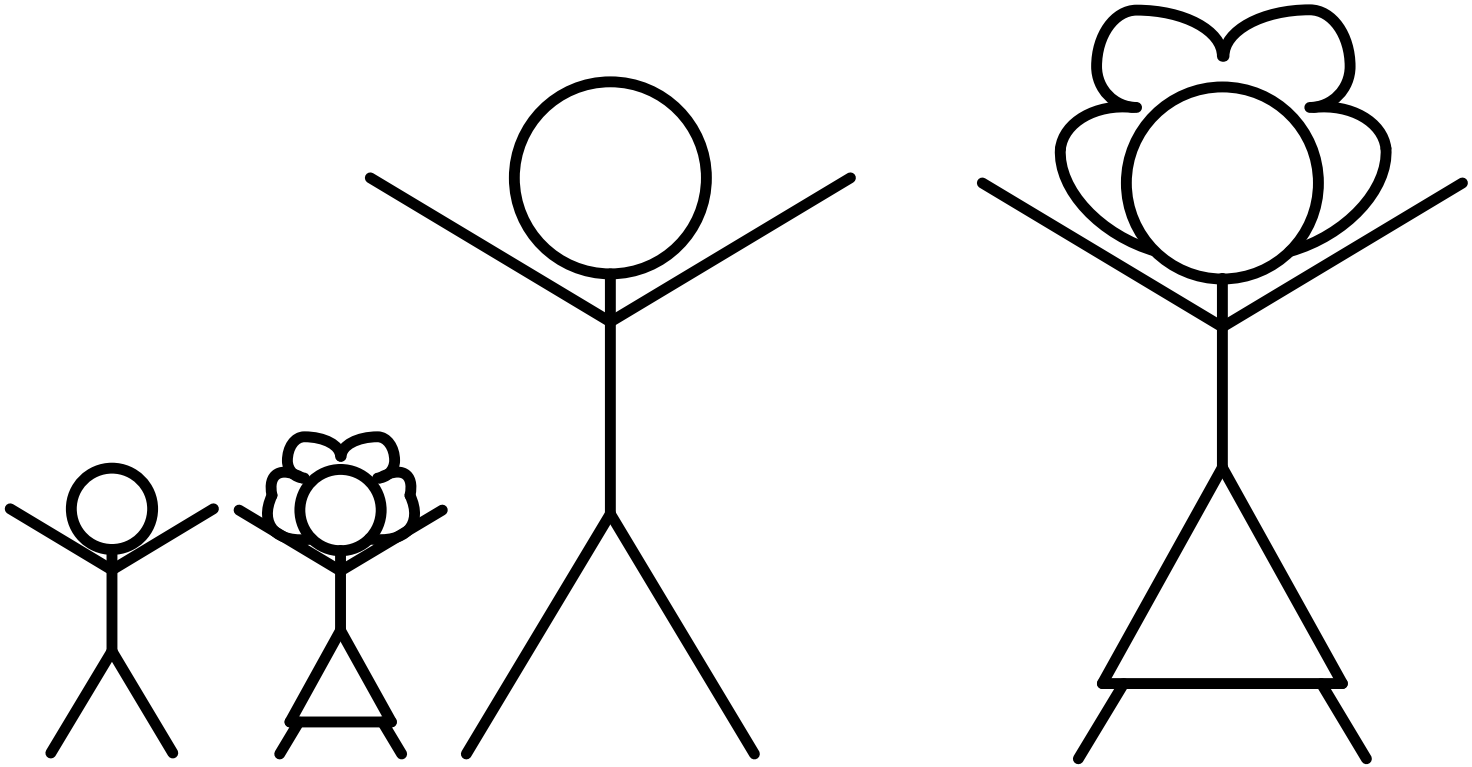
The H2O problem

```
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;
```

The H2O problem

```
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.
```

The child care problem



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.

The child care problem

```
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;
```

The child care problem

```
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 := 0 to out do
    begin
        signal(toLeave);
        wait(confirm);
    end
    signal(mutex);
end;
```

The child care problem

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

The child care problem

```
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;
```

The child care problem

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

Питања?

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