

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

wait(s)

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 **then**

$s=s+1$

else begin

Уклони процес из реда чекања на семафору s

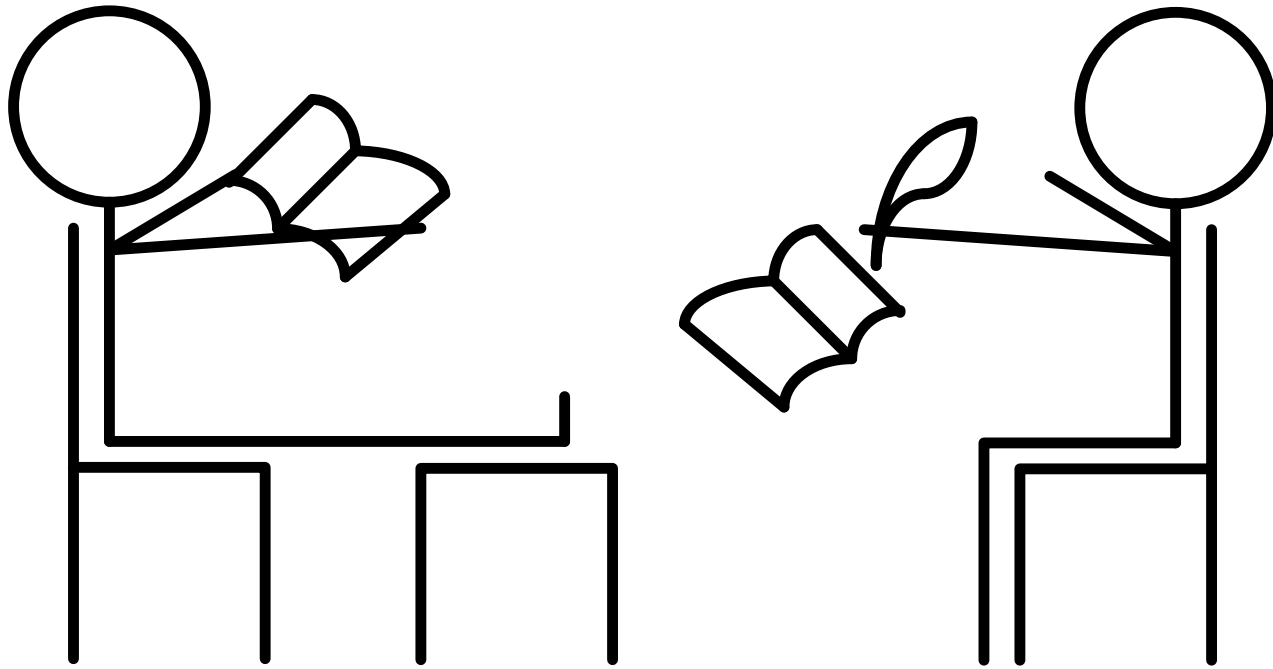
Активирај тај процес код ОС – стави га у ред спремних

end

Задаци



Readers – Writers problem



Readers – Writers problem

Два типа процеса, читаоци и писци, приступају једном запису (у општем случају запис може припадати некој колекцији података - бази података, датотеци, низу, уланчаној листи, табели итд.) Читаоци само читају садржај записа, а писци могу да читају и мењају садржај записа. Да не би дошло до нерегуларне ситуације у којој запису истовремено приступа више писаца или истовремено приступају и писци и читаоци, писци имају право ексклузивног приступа. Са друге стране, дозвољено је да више читалаца истовремено приступа запису (нема ограничења у њиховом броју). Написати програм којим се реализује рад процеса читалаца и писаца.

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 - 2

```
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;  
end;
```

Readers – Writers problem - 2

```
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;
```

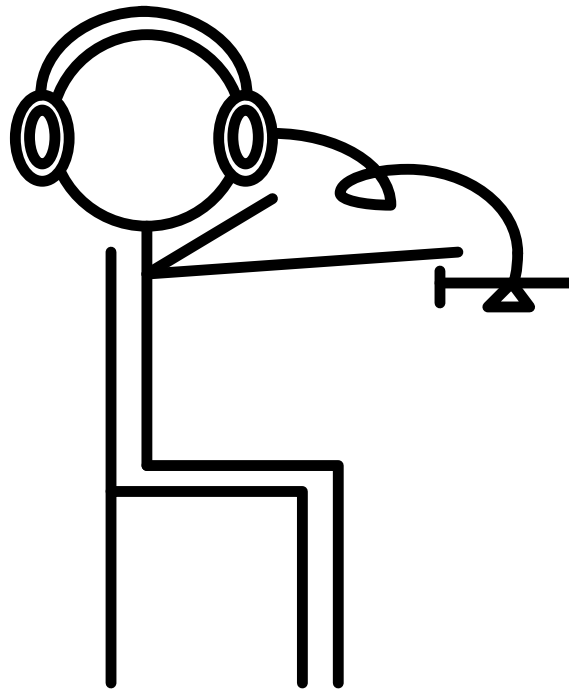
Producer – Consumer problem

```
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;  
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;  
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 на који чекају свих N потрошача, и то само у слободне слотове. Сваки потрошач мора да прими производ у тачно оном редоследу у коме су произведени, мада различити потрошачи могу у исто време да узимају различите производе.

Atomic broadcast problem

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

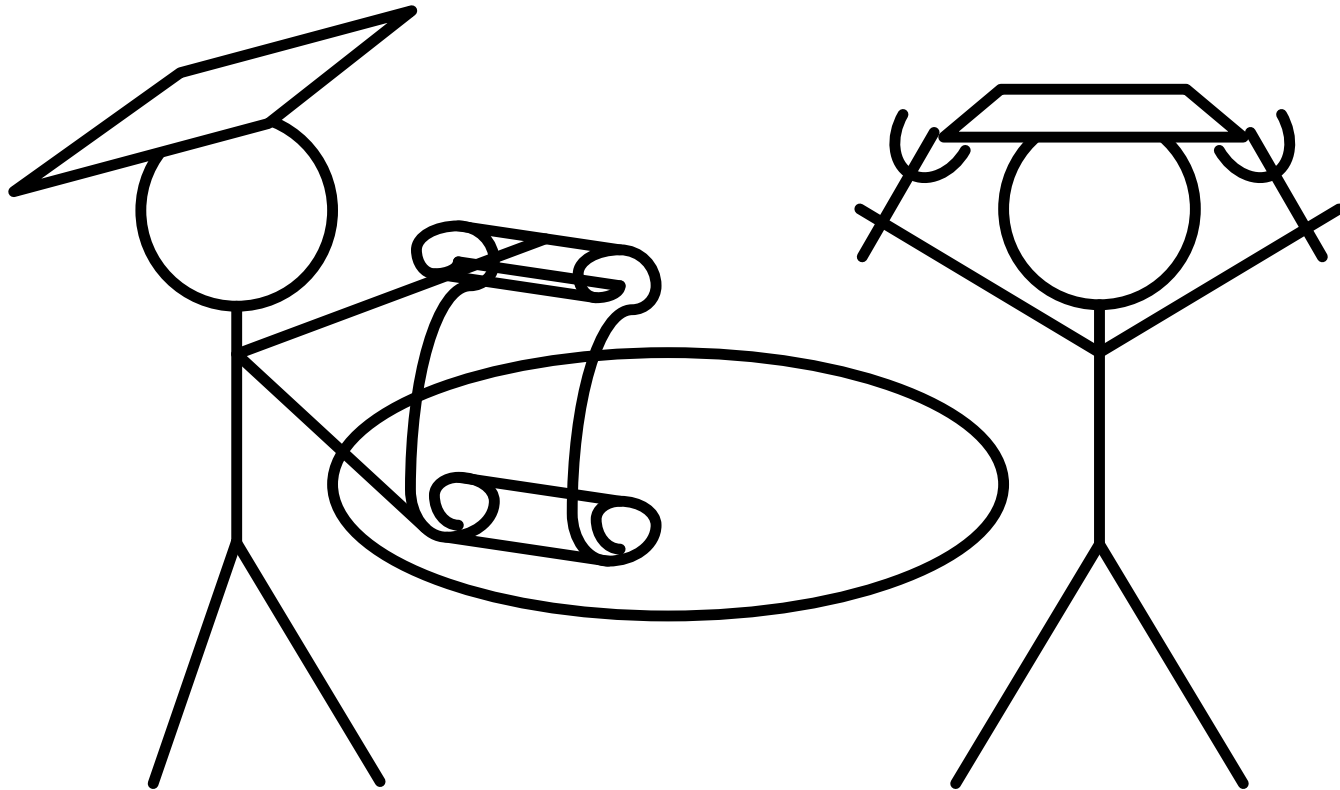
Atomic broadcast problem

```
procedure Consumer(ID : integer);  
var item : integer;  
begin  
    while (true) do  
        begin  
            wait(full[ID]);  
            wait(mutex[readFromIndex[ID]]);  
  
            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;  
            signal(mutex[readFromIndex[ID]]);  
            readFromIndex[ID] := (readFromIndex[ID] mod B) + 1;  
            consume_item(item);  
        end;  
    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 ($0 \leq i \leq 4$).

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;  
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;
```

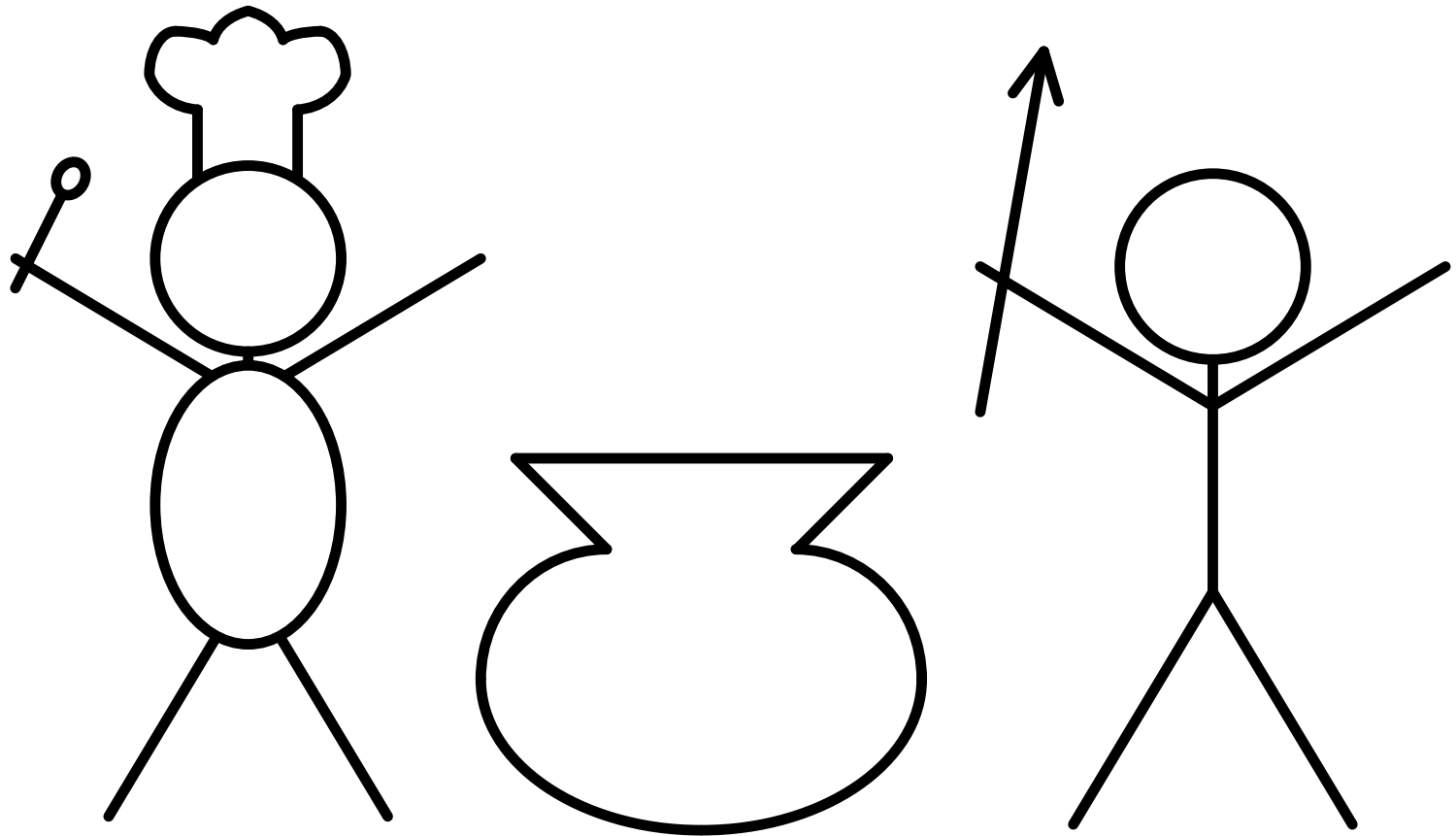
Dining philosophers problem - 2

```
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 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.

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;  
end;
```

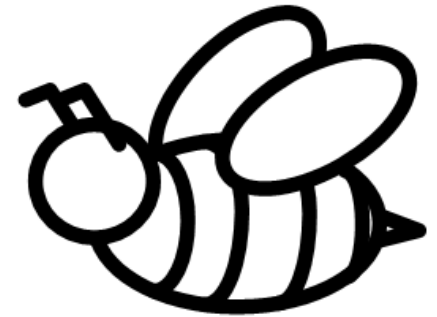
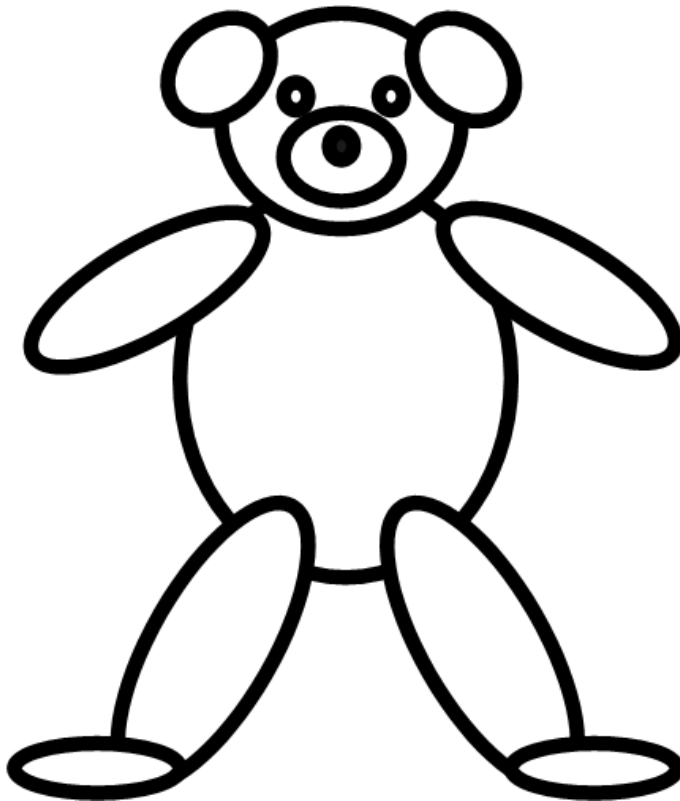
The dining savages problem

```
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);  
      eat;  
    end;  
  end;  
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.
```

The Bear and the Honey bees problem



The Bear and the Honey bees problem

Постоји N пчела и један гладан медвед. Они користе заједничку кошницу. Кошница је иницијално празна, а може да прими N напрстака меда. Медвед спава док се кошница не напуни медом, када се напуни медом, он поједе сав мед након чега се враћа на спавање. Пчелице непрестано лете од цвета до цвета и сакупљају мед. Када прикупе један напрстак долазе и стављају га у кошницу. Она пчела која је попунила кошницу буди медведа. Користећи семафоре решити проблем.

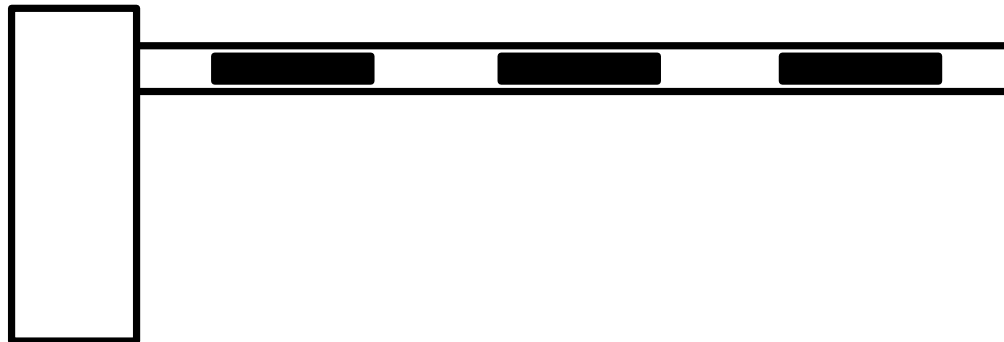
The Bear and the Honey bees problem

```
program BearAndHoneybees;  
const    N = ...;  
          H = ...;  
var      hive : semaphore;  
          full : semaphore;  
          pot : shared integer;  
  
procedure Bear;  
procedure sleep; begin ... end;  
procedure eat; begin ... end;  
begin  
    while (true) do  
    begin  
        sleep;  
        wait(full);  
        eat;  
        pot := 0;  
        signal(hive);  
    end;  
end;
```

The Bear and the Honey bees problem

```
procedure Honeybee (id : integer);
procedure collect; begin ... end;
begin
    while (true) do
        begin
            collect;
            wait(hive);
            pot := pot + 1;
            if (pot = H) then signal(full)
            else signal(hive);
        end;
    end;
end;
begin
    pot := 0;
    init(hive, 1);
    init(full, 0);
    cobegin
        Bear;
        Honeybee(1);
        Honeybee(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 достигну одређену тачку у извршавању, пре него што било која од тих нити не настави са својим извршавањем. Користећи семафоре решити овај проблем. Омогућити да се иста баријера може користити већи број пута.

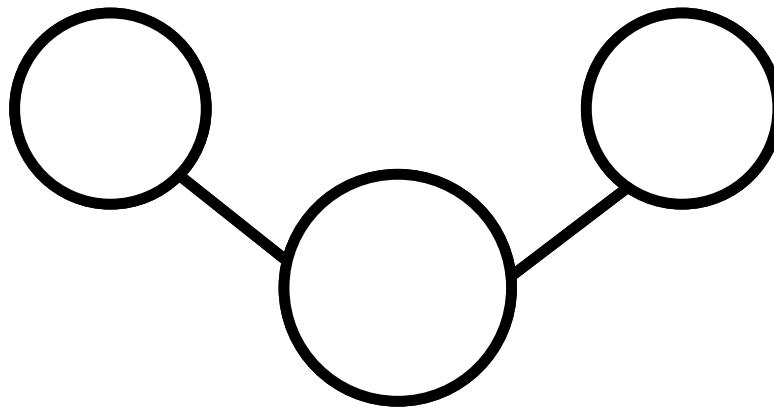
Barrier Synchronization

```
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 – 1

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 – 1

```
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 – 1

```
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 H2O problem – 2

```
program H2O;  
type Molecule = record ids : array[0..2] of integer end;  
  
var  
    hydroSem : semaphore;  
    oxySem : semaphore;  
    door1, door2 : semaphore;  
    count : integer;  
    mol : Molecule;  
  
procedure bond(mol : Molecule); begin ... end;
```


The H2O problem – 2

```
procedure Oxygen(id : integer);  
var mol : Molecule;  
begin  
    wait(oxySem);  
    mol := barrier(id);  
    bond(mol);  
end;
```

```
procedure Hydrogen(id : integer);  
var mol : Molecule;  
begin  
    wait(hydroSem);  
    mol := barrier(id);  
    bond(mol);  
end;
```

The H2O problem – 2

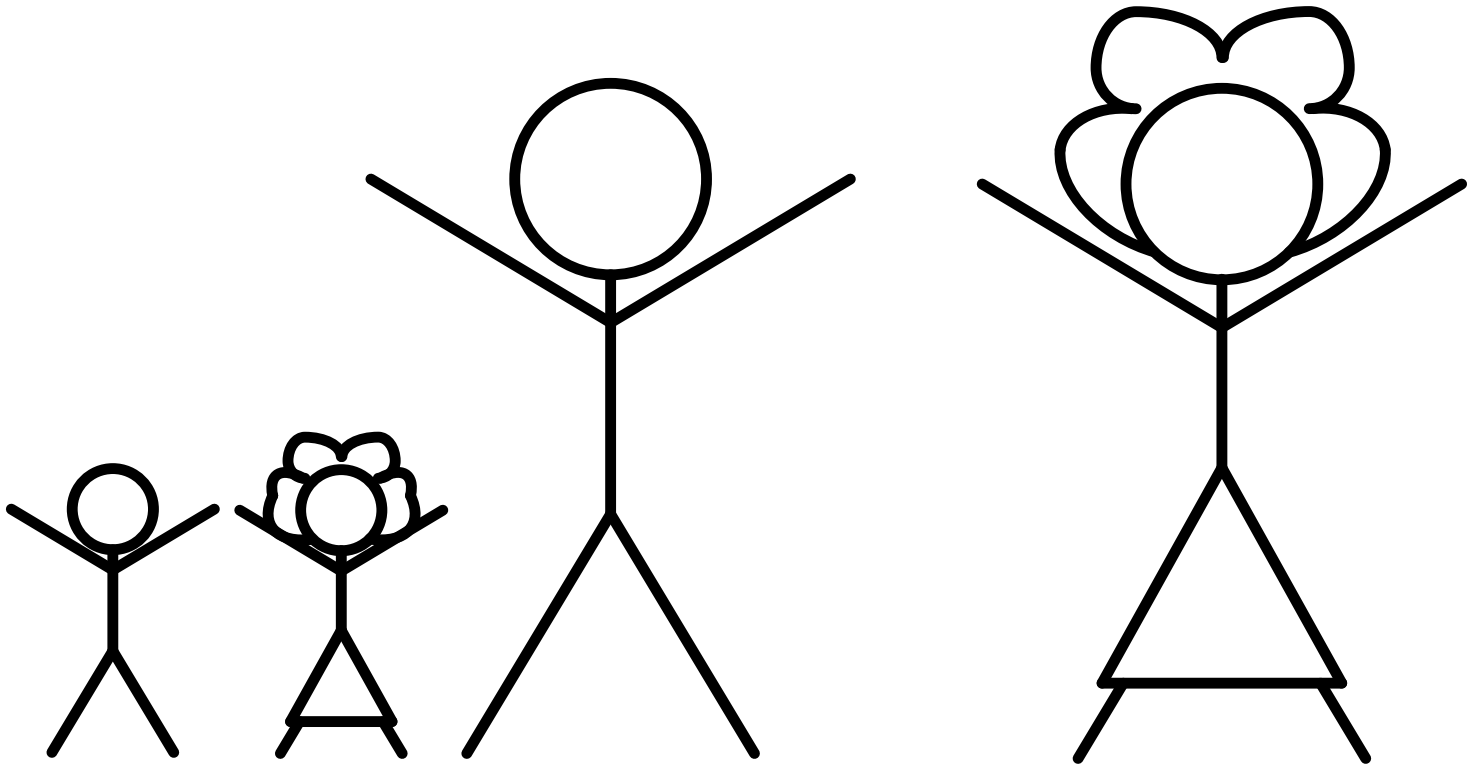
```
function barrier(id : integer) : Molecule;  
begin  
    wait(door1);  
    mol.ids[count] := id;  
    count := count + 1;  
    if (count = 3) then  
        begin  
            count := 0; signal(door2);  
        end  
    else  
        signal(door1);  
    wait(door2);  
    barrier := mol;  
    count := count + 1;  
    if (count = 3) then  
        begin  
            count := 0;  
            signal(oxySem); signal(hydroSem); signal(hydroSem);  
            signal(door1);  
        end  
    else  
        signal(door2);  
  
end;
```



The H2O problem

```
begin  
  count := 0;  
  init(hydroSem, 2);  
  init(oxySem, 1);  
  init(door1, 1);  
  init(door2, 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 васпитачице 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.

The child care problem

Решити проблем уколико родитељ мора да сачека да остави децу све док не буде било места да их све остави.

The child care problem

```
program ChildCare;  
const C = 3;  
var numChild : integer;  
    numNann : integer;  
    numWaitingC : integer;  
    numWaitingN : integer;  
    mutex, entry, toLeave : semaphore;  
procedure bringUpChildren (num : integer);  
begin  
    wait(entry)  
    wait(mutex);  
    if((numChild + num) > C * numNann) then  
        begin  
            numWaitingC := num;  
            signal(mutex);  
            wait(toLeave);  
            numWaitingC := 0;  
        end  
    numChild := numChild + num;  
    signal(mutex);  
    signal(entry);  
end;
```


The child care problem

```
procedure bringBackChildren (num : integer);  
var out, i : integer;  
begin  
    wait(mutex);  
    numChild := numChild - num;  
    if ((numWaitingC > 0) and (numChild + numWaitingC <= C * numNann)) then  
        signal(toLeave)  
    else if ((numWaitingN > 0) and (numChild <= C * (numNann - 1))) then  
        signal(toLeave);  
    else  
        signal(mutex);  
end;
```

The child care problem

```
procedure nannEnter;  
begin  
    wait(mutex);  
    numNann := numNann + 1;  
    if ((numWaitingC > 0) and (numChild + numWaitingC <= C * numNann)) then  
        signal(toLeave)  
    else if ((numWaitingN > 0) and (numChild <= C * (numNann - 1))) then  
        signal(toLeave);  
    else  
        signal(mutex);  
end;
```

The child care problem

```
procedure nannExit;  
begin  
    wait(entry);  
    wait(mutex);  
    if (numChild > C * (numNann - 1)) then  
        begin  
            numWaitingN := numWaitingN + 1;  
            signal(mutex);  
            wait(toLeave);  
            numWaitingN := numWaitingN - 1;  
        end  
        numNann := numNann - 1;  
        signal(mutex);  
        signal(entry);  
end;
```

The child care problem

begin

```
numChild := 0;  
numNann := 0;  
numWaitingC := 0;  
numWaitingN := 0;  
init(mutex, 1);  
init(entry, 1);  
init(toLeave, 0);
```

cobegin

...

coend;

end.

Питања?

Захарије Радивојевић, Сања Делчев
Електротехнички Факултет
Универзитет у Београду
zaki@etf.rs, sanjad@etf.rs

