# Монитори

# Монитори

- Монитор је скуп сталних променљивих које служе за памћење стања неког ресурса и одговарајућих процедура за имплементацију операција над ресурсом, односно сталним променљивама.
- Приступ сталним променљивама је могућ само преко процедура монитора.
- Сталне променљиве задржавају вредност између два позива мониторских процедура, док се вредности локалних променљивих унутар мониторских процедура не памте.
- Условна синхронизација код монитора се постиже операцијама *signal* и *wait* на некој условној променљивој.

# Монитори

mname: monitor

var декларације сталних променљивих

декларација условних променљивих: **condition**;

procedure p1(параметри);

var декларације локалних варијабли процедуре p1

begin

код који имплементира p1

*cond.**wait***

...

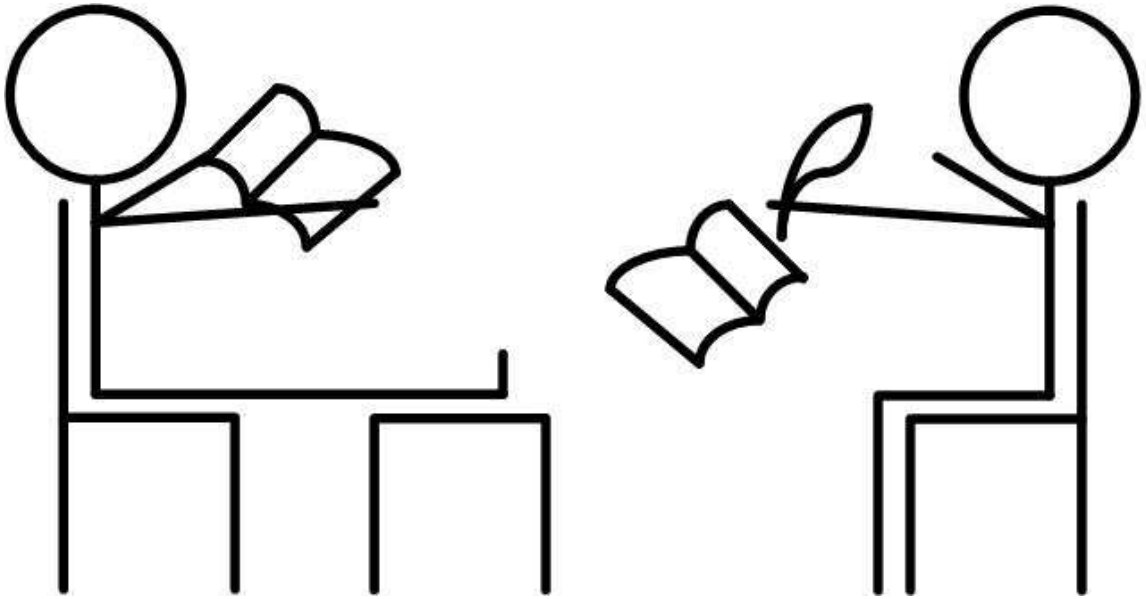
*cond.**signal***

end

...

# Задаци

# Readers – Writers problem



# Readers – Writers problem

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

# Readers – Writers problem

```
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;
```

# Readers – Writers problem

```
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.
```

# Readers – Writers problem

Решити проблем читалаца и писаца (*Readers–Writers Problem*) проблем користећи мониторе који имају дисциплину signal and continue. Приликом решавања задатка потребно је избећи узајамно блокирање.

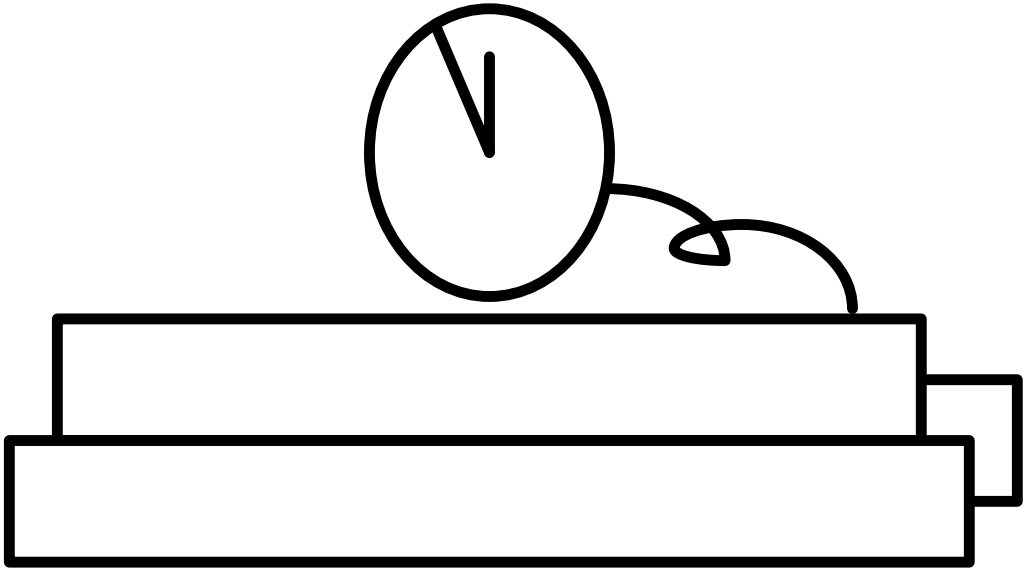
# Readers – Writers problem

```
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;
        OKtoWork.signalAll;
    end;
```

# Readers – Writers problem

```
    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.
```

# Тајмер



# Тајмер

Реализовати монитор који омогућава програму који га позива да чека n јединица времена. Користити signal and wait дисциплину.

# Тајмер

```
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.
```

# 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.

# Тајмер

```
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.
```

# Producer – Consumer problem

# Producer – Consumer problem

Решити проблем Producer – Consumer користећи мониторе који имају signal and wait дисциплину

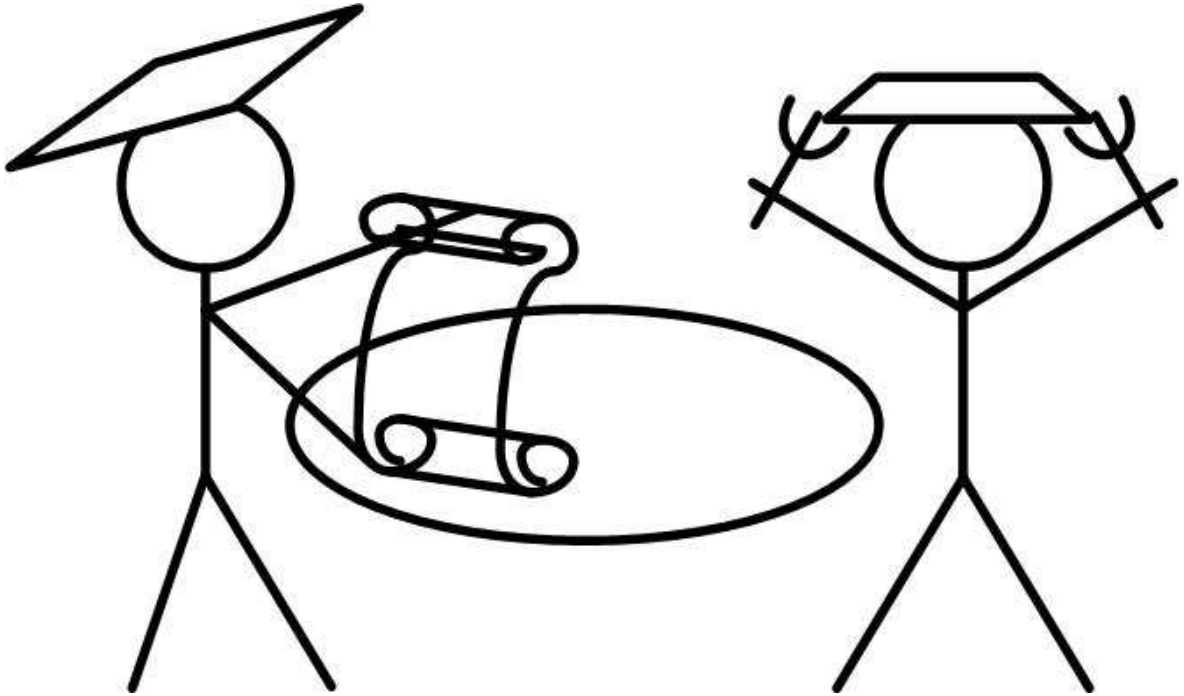
# Producer – Consumer problem

```
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;
```

# Producer – Consumer problem

```
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.
```

# Dining philosophers problem



# Dining philosophers problem

Решити проблем филозофа који ручавају користећи монитоpe који имају дисциплину signal and wait.

# Dining philosophers problem

```
program DP;
const     num_phils = 5;
    num_philsminusone = 4;
procedure philosopher(id : integer);
    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.
```

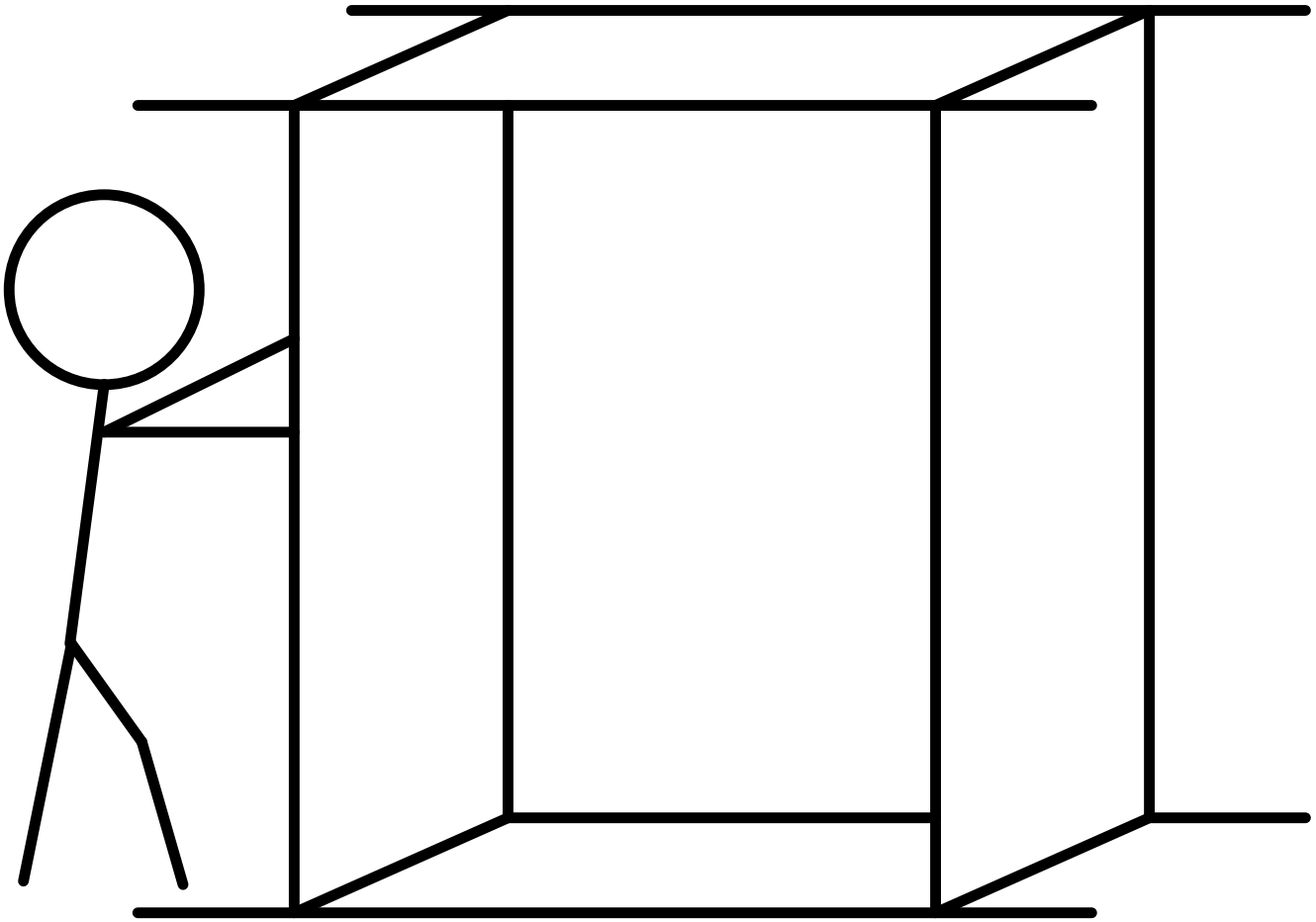
# Dining philosophers problem

```
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 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;
```

# Dining philosophers problem

```
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;

    begin
            for index := 0 to 4 do state[index] := 0;
    end;
```

# Прихватник



# Прихватник

Реализујте монитор за 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 мониторске процедуре. Користити signal and wait дисциплину.

# Прихватник

```
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 - 2) then not_full.wait;
            slots[tail] := first;
            tail := (tail + 1) mod N;
            slots[tail] := second;
            tail := (tail + 1) mod N;
            size := size + 2;
            if((size > 0) and (not_empty.queue)) then not_empty.signal;
            if((size > 0) and (not_empty.queue)) then not_empty.signal;
    end;
```

# Прихватник

```
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 <= N - 2) and (not_full.queue)) then not_full.signal
    end;
    procedure cancel;
    begin
        size := 0; head := 0; tail := 0;
        while((size <= N - 2) and (not_full.queue)) do not_full.signal
    end;
begin
    size := 0; head := 0; tail := 0
end;
```

# Питања?

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