Cvičení 9 - Monitory

- na rozdíl od semaforů je monitor jazyková konstrukce
- monitor = Pascalský blok podobný proceduře nebo fci
- uvnitř monitoru jsou definovány proměnné, procedury a fce
- proměnné monitoru nejsou viditelné zvenčí, jsou dostupné pouze procedurám a fcím monitoru
- procedury a fce jsou viditelné a volatelné zvenčí (vně monitoru)
- v BACI nemohou být monitory vnořené, v monitoru nelze definovat fci

```
monitor m;
var proměnné ...

  procedure p;
  begin
     ...
  end;

begin
  inicializace;
end;
```

- kód mezi begin a end se spustí před spuštěním hlavního programu
- používá se pro inicializaci proměnných monitoru
- monitory se používají pro vzájemné vyloučení a synchronizaci
- v jednu chvíli může v monitoru běžet pouze jedna procedura nebo fce
- výhoda - strukturovaný popis, kód zajišťující vzájemné vyloučení není rozptýlen
- kritická sekce programu se přesune do monitoru

Úloha 4 (ošetření KS pomocí monitoru)

Program demonstrující časový souběh ošetřete pomocí monitoru.

- proměnná x bude proměnnou monitoru
- vytvoříme procedury monitoru "nastav", "zvětši" a "prozraď" - nastaví hodnotu x, zvětší x o 1, vrátí hodnotu x
- monitor chápeme jako objekt, procedury jako jeho metody

Řešení:

```
// demonstrace monitoru
monitor m {
    int x;

    void setx(int val)
    {
        x = val;
    }

    int getx()
    {
        return x;
    }

    void incx()
    {
        x++;
    }
}

void a()
{
    int i;

    for (i=0; i<50; i++)
        incx();
}

int main() {
    setx(0);
        cobegin {
              a(); a();
        }
        cout << "x=" << getx() << endl;
}
```

synchronizace

- pouze vzájemné vyloučení - nestačí pro synchronizaci (např. nevyřešili bychom problém producent / konzument)
- pro synchronizaci v monitoru se používají podmínky (condition variables)
- mohou být definovány pouze v monitoru, neobsahují hodnotu
- nad podmínkami definovány 2 operace - wait a signal - v BACI procedury:

`PROCEDURE WAITC(cond : CONDITION);`
WAITC zablokuje proces v monitoru; pokud je některý proces připraven vstoupit do monitoru, bude mu to dovoleno

`PROCEDURE SIGNALC(cond : CONDITION);`

- pokud existuje jeden nebo více procesů spících nad podmínkou, jeden vzbudí
- pokud nad podmínkou nespí žádný proces, nedělá nic (na rozdíl od operace V(sem) nad semaforem, která si "zapamatuje" že byla zavolána)
- Hoarova definice monitoru - v BACI spící je vzbuzen ihned, proto je proces volající SIGNALC pozastaven

Co pokusit se řešit mutexy pomocí monitorů?

```
monitor uuu;
var c: condition;
    l: boolean:=false;

procedure mylock;
begin
   if l then waitc(c);
   l:=true;
end;

procedure myunlock;
begin
    l:=false;
    signalc(c);
end;
```

Implementace spravedlivých semaforů

Fce empty

- oproti tradičním monitorům v BACI navíc fce empty(cond: condition)
  - vrací true pokud nad podmínkou cond nikdo nespí
  - jinak vrací false

Prioritní čekání

PROCEDURE WAITC(cond: CONDITION);

WAITC zablokuje proces v monitoru; pokud je některý proces připraven vstoupit do monitoru, bude mu to dovoleno

PROCEDURE WAITC(cond: CONDITION; prio: INTEGER);

- jako WAITC(cond), ale procesu je přiřazena priorita pro vzbuzení (tj. pořadí, ve kterém budou procesy vzbuzeny); defaultní priorita je 10
- nižší číslo = vyšší priorita
- většina reálných implementací monitorů (např. v Javě, Delphi apod.) nemá prioritní WAIT
- prioritní schéma lze použít pro implementaci FIFO pro buzení čekajících procesů
- zavedeme proměnnou monitoru s aktuální prioritou
- prio := prio+1; WAITC(cond, prio);

PROCEDURE SIGNALC(cond: CONDITION);

- pokud existuje jeden nebo více procesů spících nad podmínkou, vzbudí proces s nejnižším číslem "prio" (proces s nejvyšší prioritou)
- pokud nad podmínkou nespí žádný proces, nedělá nic (na rozdíl od operace V(sem) nad semaforem, která si "zapamatuje" že byla zavolána)
- v BACI spící je vzbuzen ihned - proto je proces volající SIGNALC pozastaven => procesy volající SIGNALC to provádějí typicky jako svou poslední instrukci

Úloha 1 (večeřící filosofové)

- 5 filosofů sedí kolem kulatého stolu
- každý filosof před sebou talíř se špagetami
- mezi každými dvěma talíři je vidlička
- špagety jsou tak klouzavé, že filosof potřebuje 2 vidličky, aby mohl jíst

- program by měl vypisovat: "Filosof #N ji"
- pozor - syntaxe BACI:
  - místo konstant se nedají používat konstantní výrazy
  - problém s fcí modulo: (-1 mod 3) je -1

```
monitor večeřící_filosofové;

const N=5;

var
   f: array [0 .. N-1] of integer;   { vidličky }
   a: array [0 .. N-1] of condition; { vidličky jsou k
dispozici }
   i: integer;

procedure chci_jíst(i: integer)
begin
   if f[i] < 2 then a[i].wait;
   f[(i-1) mod N] -= 1;
   f[(i+1) mod N] -= 1;
end;

procedure mám_dost(i: integer)
begin
   f[(i-1) mod N] += 1;
   f[(i+1) mod N] += 1;
   if f[(i-1) mod N] = 2 then a[(i-1) mod N].signal;
   if f[(i+1) mod N] = 2 then a[(i+1) mod N].signal
end;

begin
   for i:=0 to 4 do
      f[i] := 2;
end;
```

Úloha 2 (implementace semaforů pomocí monitorů)

Na přednášce jsme si řekli, že stejnou vyjadřovací sílu semaforů a monitorů můžeme ukázat tím, že budeme semafory implementovat pomocí monitorů a naopak.

Na přednášce bylo uvedeno řešení semaforů pomocí monitorů.

Jak bychom implementovali monitory pomocí semaforů?

operace nad semafory jsou definovány takto:

operace P(sem):
pokud sem>0, sníží sem o 1
jinak pozastaví proces, který chtěl provést operaci P.

operace V(sem):
pokud je nad semaforem sem zablokovaný jeden nebo více procesů, vzbudí jeden z procesů; proces pro vzbuzení je vybrán náhodně jinak zvýší sem o 1

implementace bude obsahovat následující procedury monitoru:

- monP - provede operaci P nad jedním semaforem implementovaným v monitoru
- monV - provede operaci V

Řešení:

```
monitor monSem;
var
  sem:     integer;
  blocked: condition;

  procedure monP;
  begin
      if sem > 0 then
          sem := sem - 1
      else
          waitc(blocked)
  end;
```

```
  procedure monV;
  begin
      if not empty(blocked) then
          signalc(blocked)
      else
          sem := sem + 1
  end;

  procedure monIni(val: integer);
  begin
       sem := val
  end;

begin { inicializace }
    sem:=1
end;
```

Rozšíření 2-1 (implementace bez fce empty)

- jak byste implementovali semafory pomocí monitorů, pokud by nebyla fce empty()?

```
monitor monSem2;
var
  sem:     integer;
  blocked: condition;

  procedure monP;
  begin
       if sem=0 then waitc(con);
       sem:=sem-1;
  end;

  procedure monV;
  begin
       signalc(con);
       sem:=sem+1
  end;
```

```
begin { inicializace }
    sem:=1
end;
```

Rozšíření 2-2 (implementace množiny semaforů)

dovolte uživateli označit semafor const hodnotou v rozsahu 1 až 3 (tj. číslo bude sloužit jako "handle" pro přístup k semaforu, resp. ve vaší implementaci bude indexem pro přístup k odpovídajícím proměnným)

* procedura implementující
 - operaci P bude monP(s: integer)
 - operaci V bude monV(s: integer)
 - inicializaci semaforu monIni(s: integer, val: integer)

Poznámka:

Řešení není maximálně paralelní, protože vzájemné vyloučení při operacích P a V se odehrává i v případě, že se operace odehrávají nad různými semafory.

V reálných programovacích jazycích bývají monitory součástí konstrukce typu objekt - s každým objektem je sdružen monitor (v BACI by bylo podobné chování možné emulovat pomocí samostatného monitoru pro každý semafor).

Úloha 3 (spravedlivé semafory pomocí monitorů)

Semafor v BACI je implementován podle Dijkstrovy definice, tj. procesy spící nad semaforem jsou buzeny v náhodném pořadí. Pro nekončící procesy však v sobě náhodné buzení skrývá nebezpečí, že dojde k vyhladovění některého procesu.

Ve většině reálných OS se proto používají tzv. spravedlivé semafory, kde jsou procesy buzeny ve FIFO pořadí.

- pomocí prioritního čekání rozšířit na "spravedlivé" semafory.
- procedury FIFO_P(sem: integer), FIFO_V(sem: integer).

- ověřit, že je funkční a pracuje jako FIFO

Úloha 4 (producent/konzument pomocí monitorů)

Implementujte řešení zjednodušeného problému producent/konzument pomocí monitoru. Velikost vyrovbávací paměti bude pouze jeden znak nebo integer.

```
program simple_prod_cons;

monitor m;

var  e, f:    condition;
     isempty: boolean;
     buffer: integer;

procedure insert(x: integer);
begin
  if not isempty then waitc(e);
  buffer:=x;
  isempty:=false;
  signalc(f)
end;

procedure remove(var x: integer);
begin
  if isempty then waitc(f);
  x:=buffer;
  isempty:=true;
  signalc(e);
end;

begin
end;

procedure Producer;
var i: integer;
begin
     for i:=1 to 99 do
```

```
        insert(i);
end;

procedure Consumer;
var x: integer;
begin
     repeat
           remove(x);
           write(x, ', ');
     until x=99;
     writeln
end;

begin
     cobegin
            Producer; Consumer
     coend
end.
```