Operačné systémy

4. Komunikácia procesov (IPC) - vzájomné vylučovanie

Ing. Martin Vojtko, PhD.

2024/2025

IPC

Interprocess communication (IPC)

Procesy a IPC

Proces je abstrakcia, ktorá delí komplexný problém na sekvenčné problémy. Komplexný problém ale nemá iba sekvenčnú povahu. Jednotlivé procesy medzi sebou komunikujú, odovzdávajú si prácu, čakajú na seba a súperia o zdroje výpočtového systému. Pri vzájomnej komunikácií procesov môže dôjsť k:

- Synchronizácii.
- Vzájomnému vylučovaniu (Mutual Exclusion).
- Výmene správ (Message Passing).

Vlákna a IPC

Problematika IPC medzi vláknami sa sústredí predovšetkým na synchronizáciu a vzájomné vylučovanie. Posielanie správ medzi vláknami nie je potrebné pretože vlákna zdieľajú spoločnú pamäť.

Interprocess communication (IPC)

Synchronizácia procesov

je technika riadeného usporiadania vzájomne spolupracujúcich procesov.

Vzájomné vylučovanie (Mutual Exclusion)

je technika synchronizácie procesov, pri ktorej je cieľom limitovať a riadiť prístup procesov do kritickej oblasti.

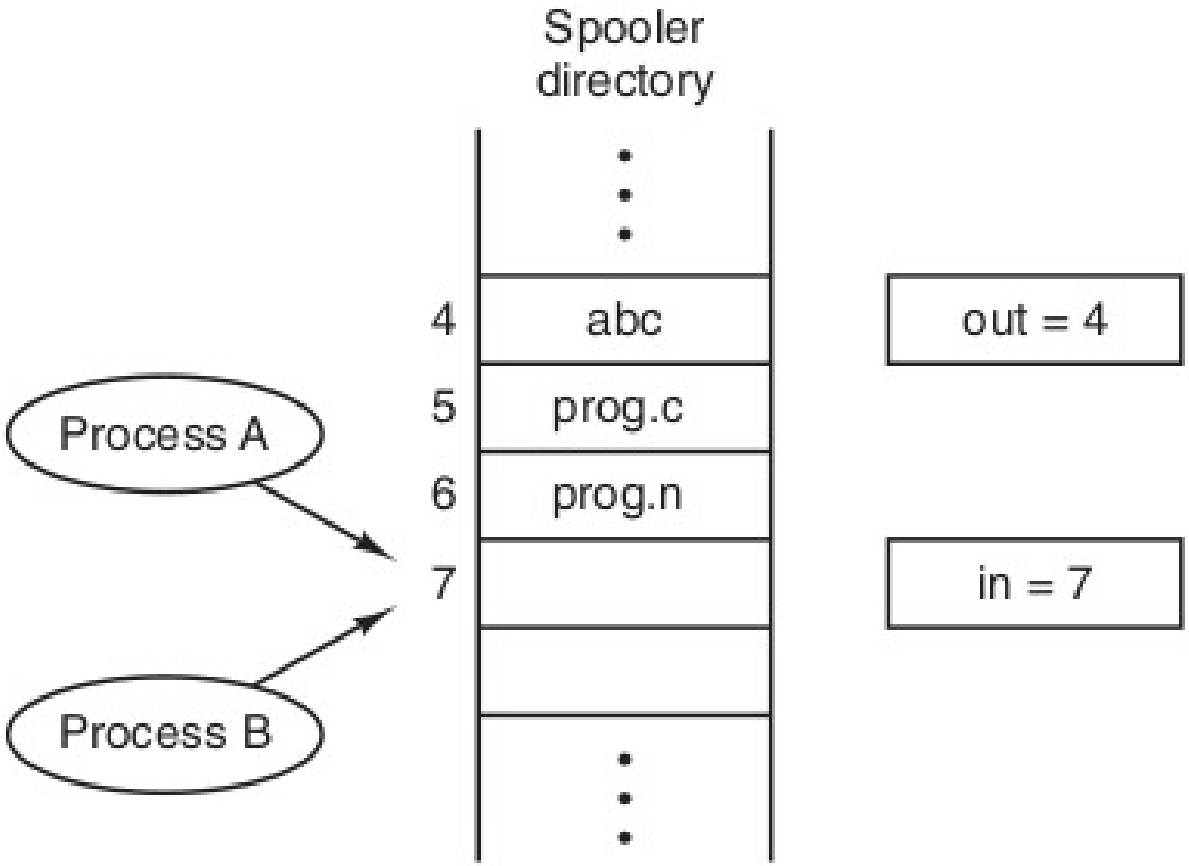
Kritická oblasť (Critical Region)

je časť programu, v ktorom proces pristupuje ku zdieľanému prostriedku. V KO vznikajú podmienky súperenia.

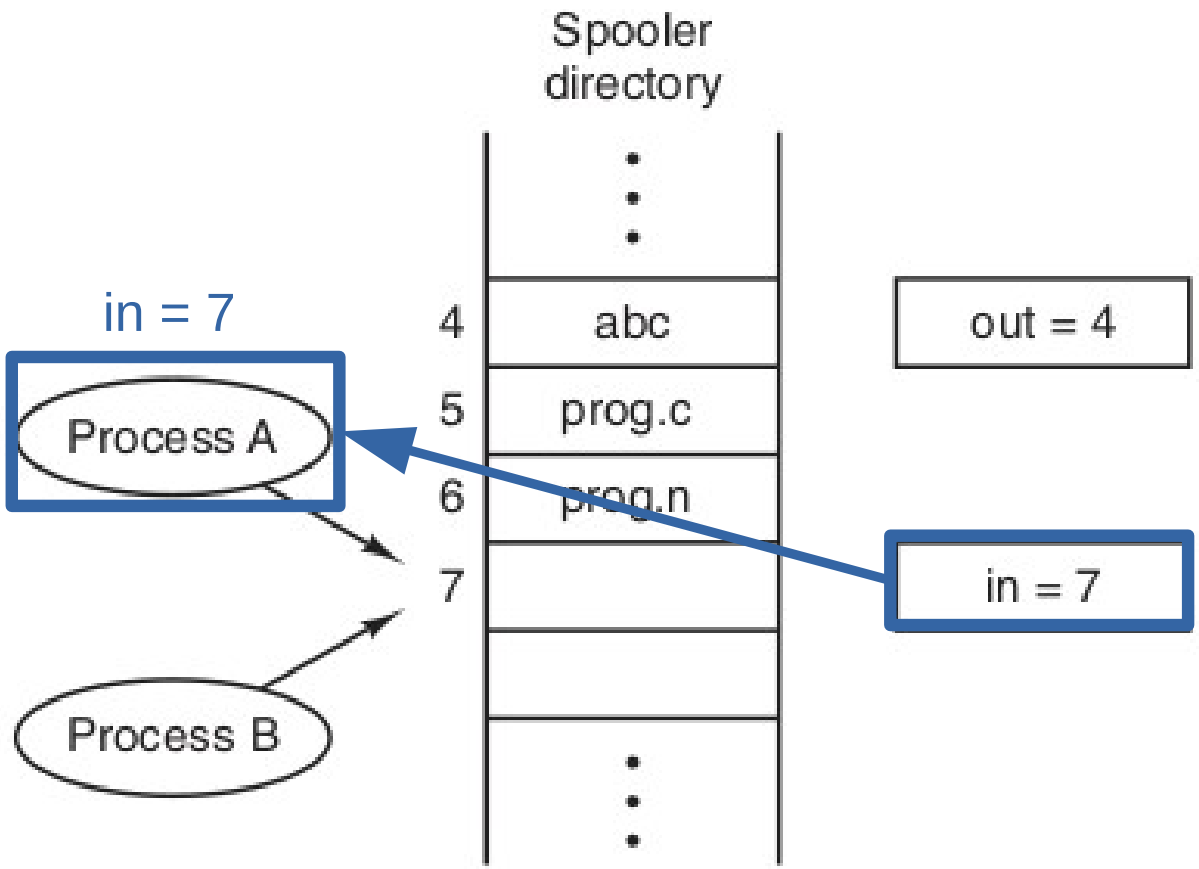
Súperenie (Race conditions)

môže vzniknúť ak dva a viac procesov zdieľajú prostriedok (najčastejšie spoločnú pamäť).

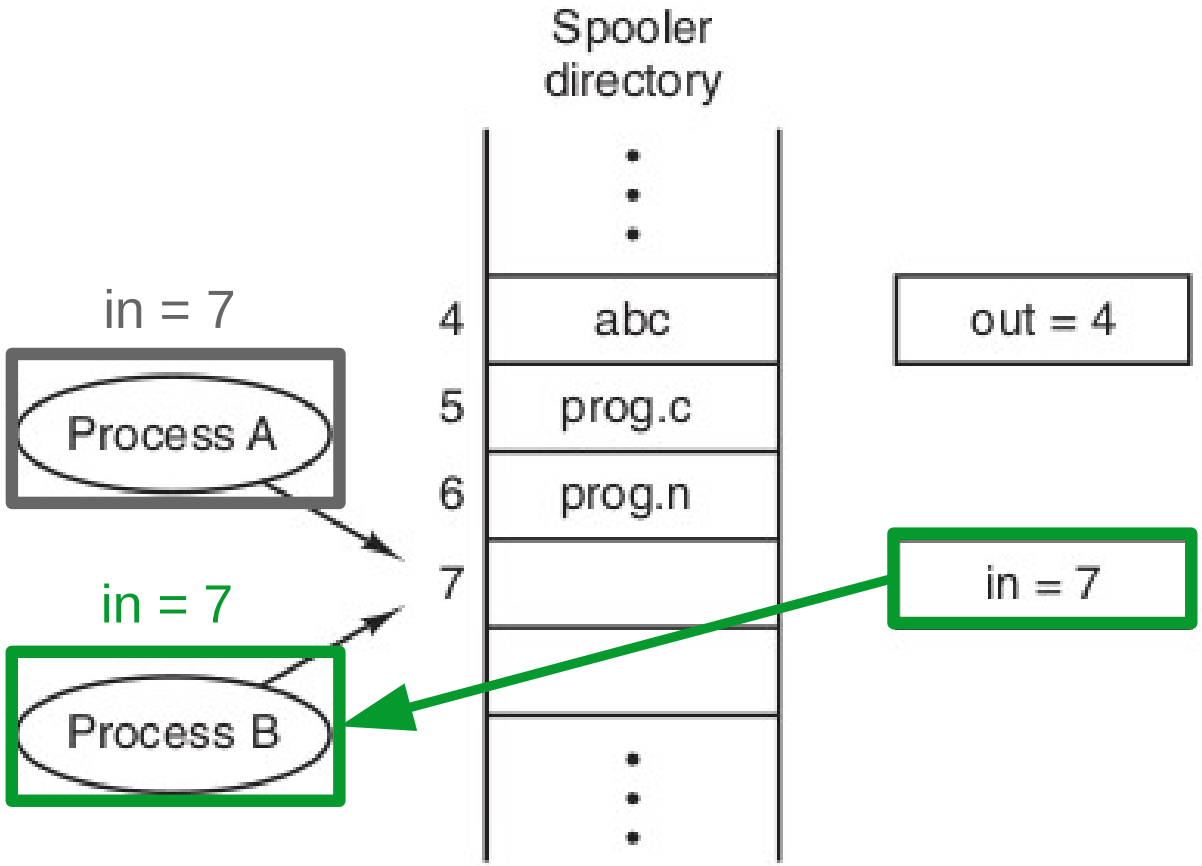
Race conditions - Spooler directory



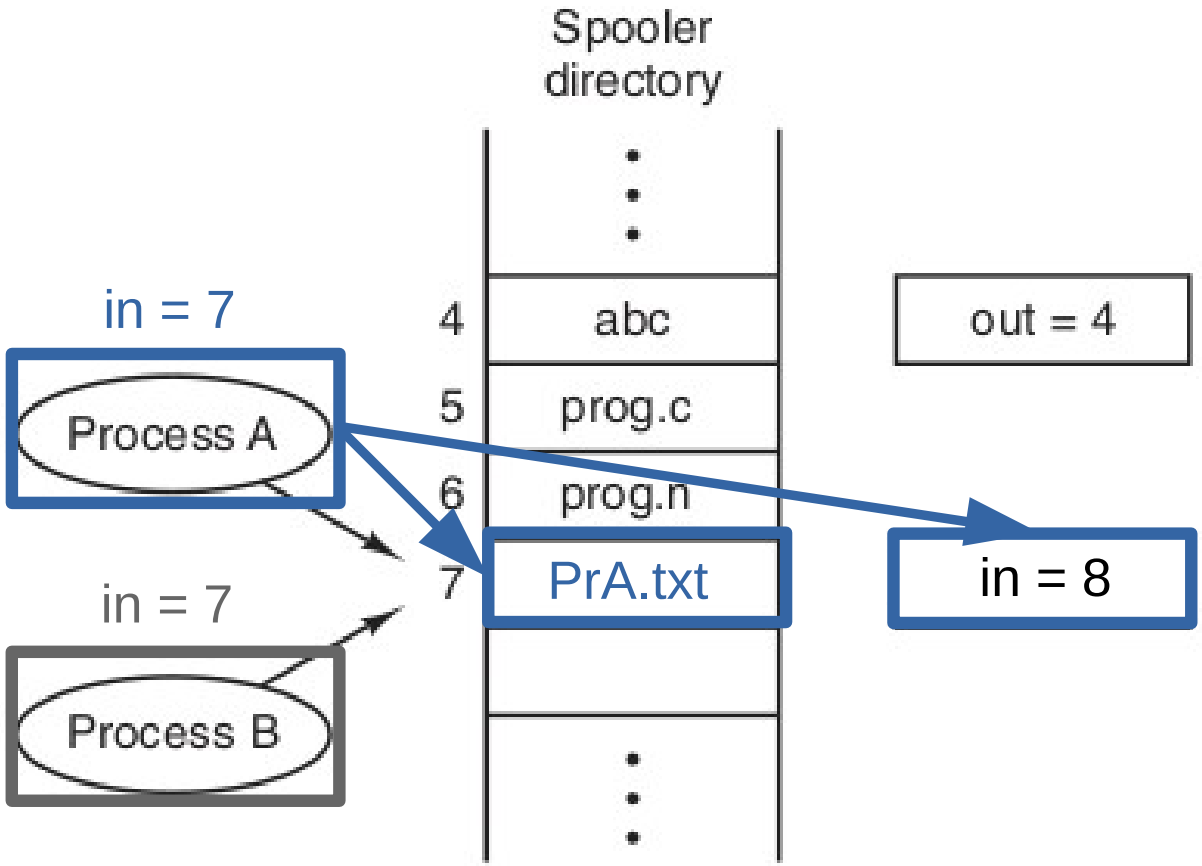
Race conditions - Spooler directory



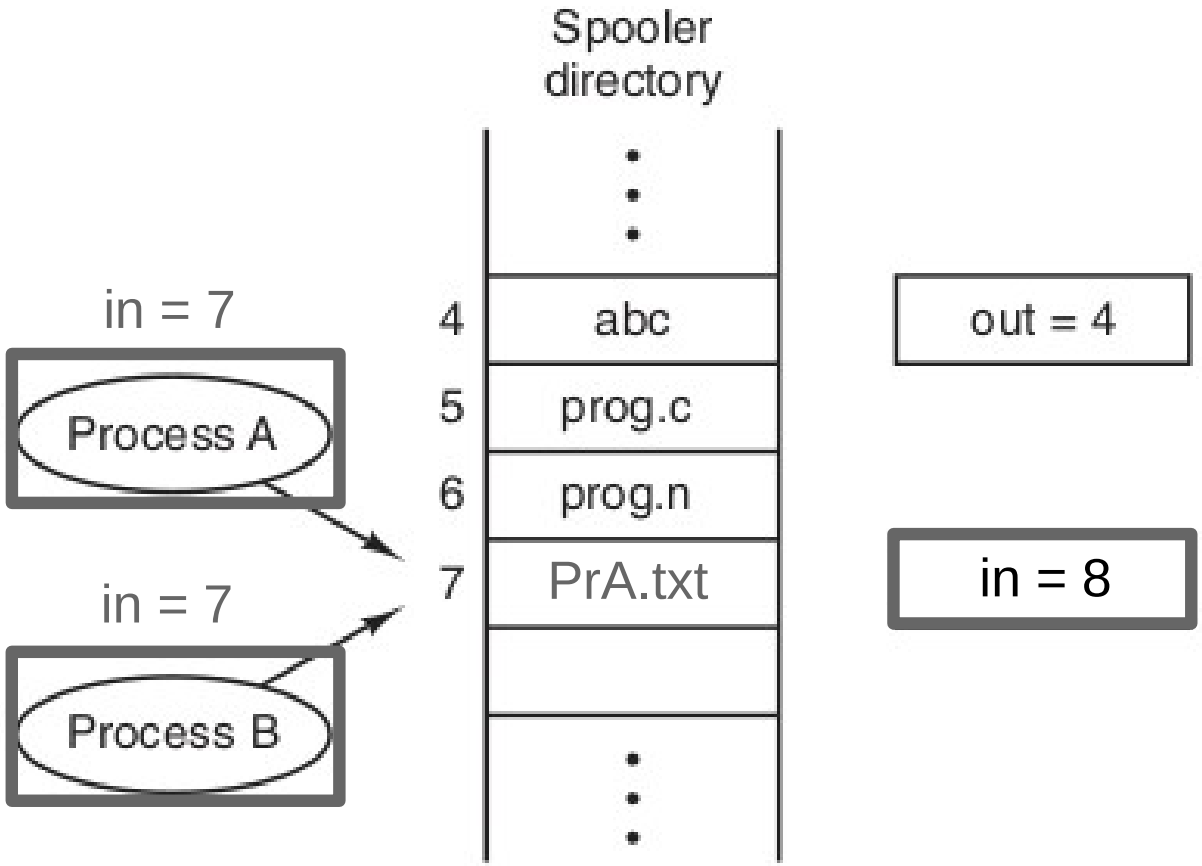
Race conditions - Spooler directory



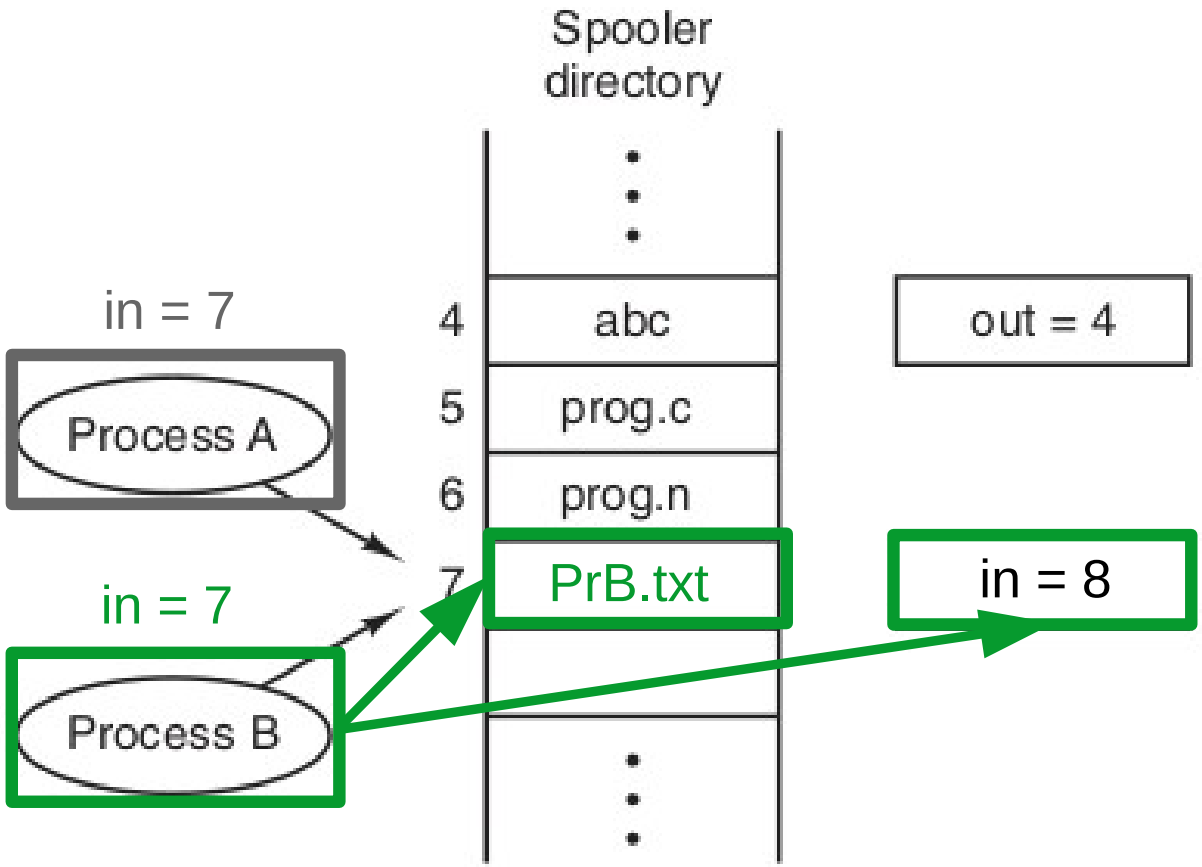
Race conditions - Spooler directory



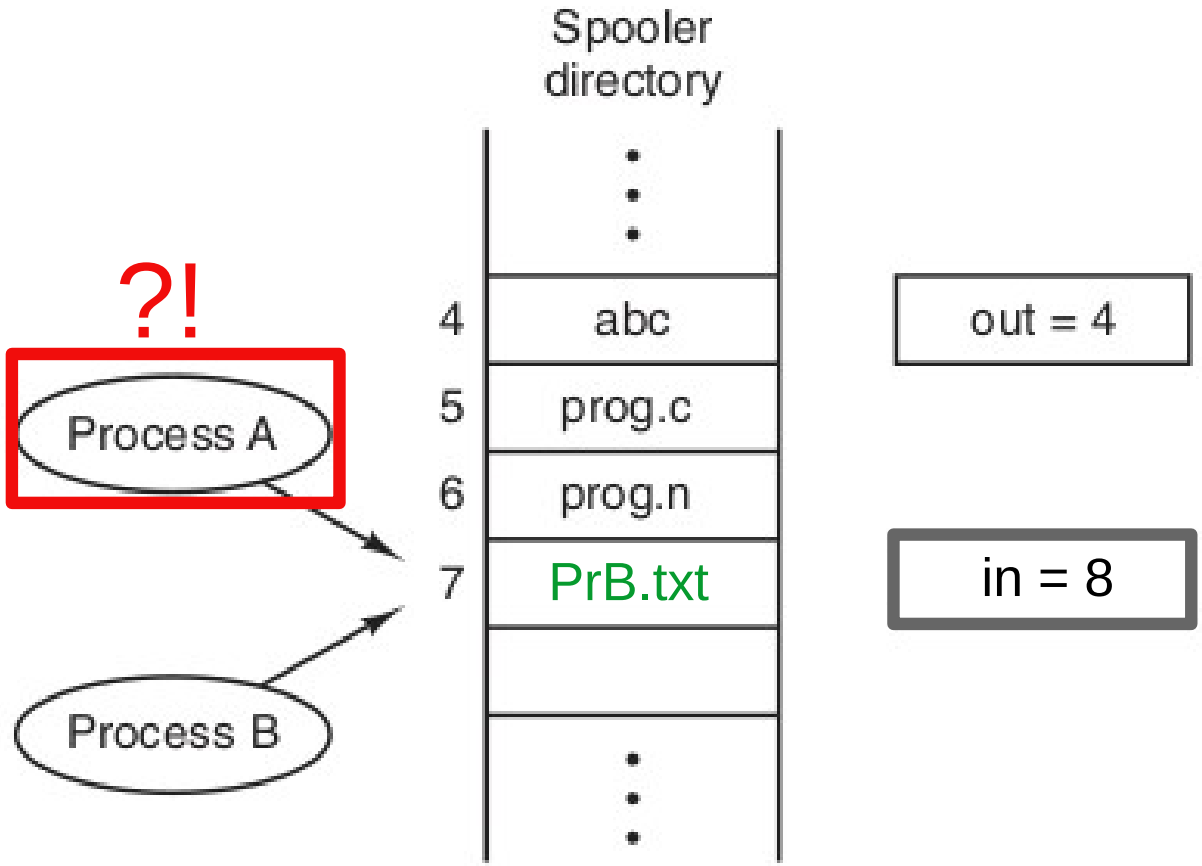
Race conditions - Spooler directory



Race conditions - Spooler directory



Race conditions - Spooler directory



Vzájomné vylučovanie (Mutual Exclusion)

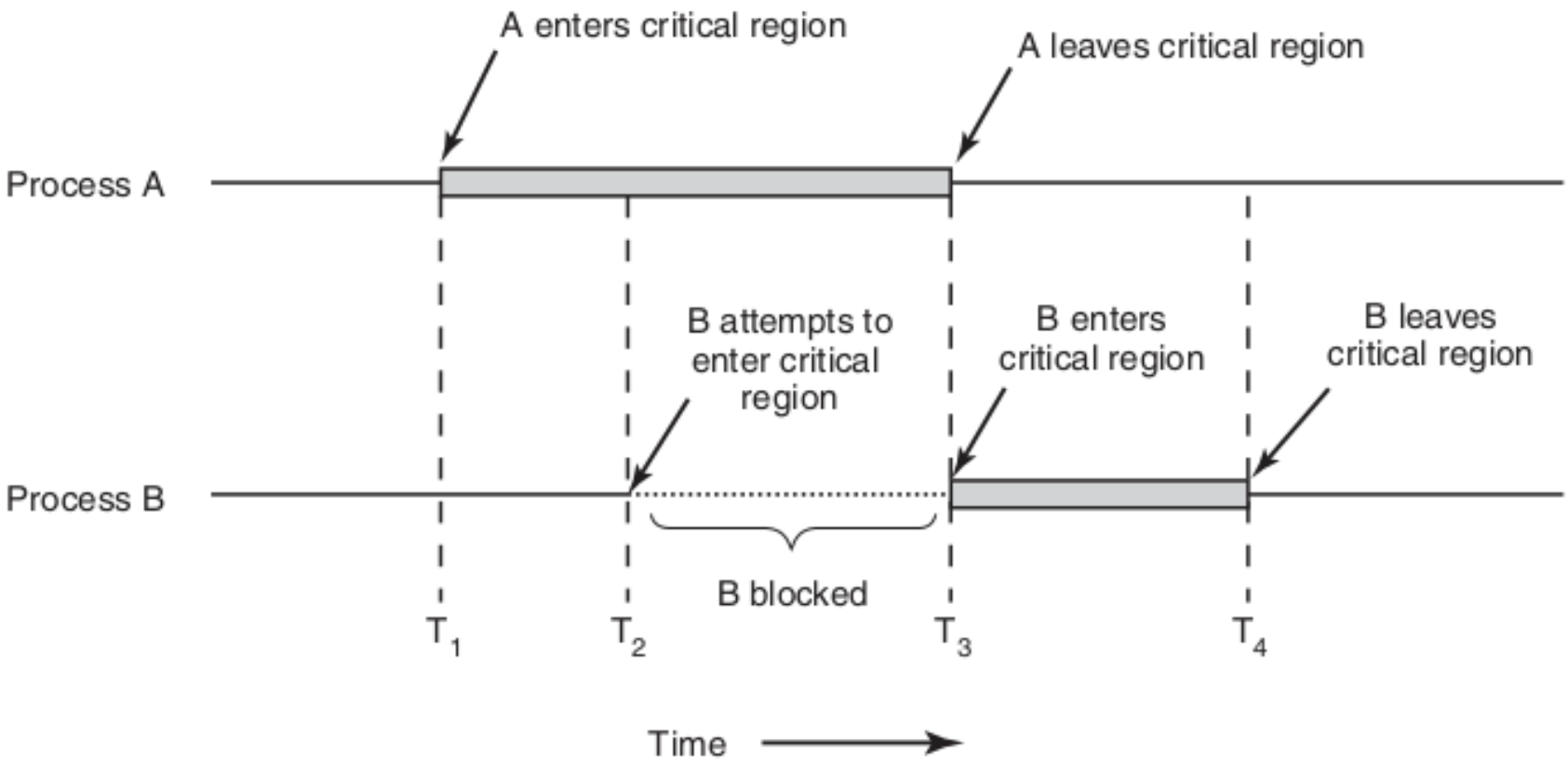
- Má zaručiť aby dva a viac procesov sa neocitli v KO súčasne.
- Na to aby sme zabezpečili fungujúce vzájomné vylučovanie je nutné splniť podmienky vzájomného vylučovania.

Podmienky riešenia vzájomného vylučovania

- V KO môže byť najviac 1 proces.
- Rýchlosť a počet CPU sa nesmie brať do úvahy.
- Proces mimo KO nesmie brániť inému procesu vstupovať do KO.
- Čas čakania procesu na vstup do KO musí byť konečný.

Vzájomné vylučovanie (Mutual Exclusion)

- Príkladom KO je zápis a/alebo čítanie z globálnej premennej.



Vzájomné vylučovanie (Mutual Exclusion)

Možné riešenia vzájomného vylučovania

- Zakázanie prerušení
- Busy Waiting (Obsadzujúce čakanie)
  - Softvérové riešenia
  - Špeciálne inštrukcie
- Sleep and WakeUp (blokovanie a zobúdzanie)
  - Semafór
  - Monitor

Vzájomné vylučovanie - Zakázanie prerušení

```
1 void process()
2 {
3     while(true) {
4         disable_interrupts();
5         critical_region();
6         enable_interrupts();
7         noncritical_region();
8     }
9 }
```

- Porušuje 2. podmienku vzájomného vylučovania. Rýchlosť a počet CPU sa nesmie brať do úvahy.

Vzájomné vylučovanie - Zakázanie prerušení

- Proces, ktorý je v KO nemôže byť prerušený a nahradený iným.
- Zakázanie prerušení pred vstupom zabráni preplánovaniu procesu.
- Veľmi časté riešenie problému vo vnorených systémoch.
- Kernel používa túto metódu ale len na veľmi krátke úseky kódu.

Problémy riešenia

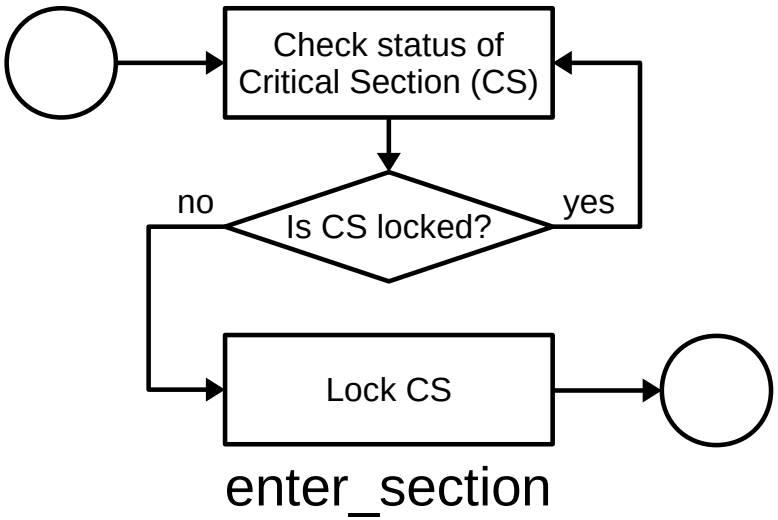
- Proces by nemal mať právo zakázať prerušenia.
- Znižuje priepustnosť výpočtového systému.
- Aplikovateľné len na procesory s jedným CPU.

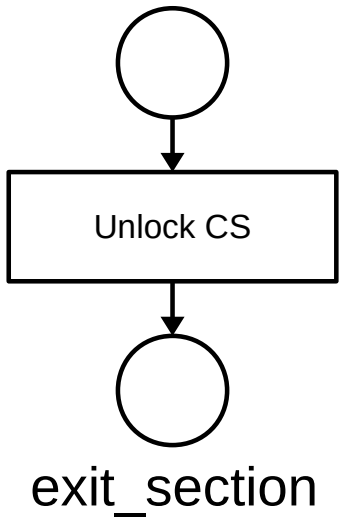
Busy Waiting

Vzájomné vylučovanie - Busy Waiting

Busy Waiting

Proces neustále v cykle kontroluje či je možné vstúpiť do KO. Ak je to možné, proces ukončí cyklus a vstúpi do KO.





Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  0
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

- Prvý pokus - skúsme riadiť prístup do KO pomocou premennej.

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  0
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  0
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  0
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  0
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock = 1
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  1
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock = 1
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

Busy Waiting - Lock Variable

```
int lock = 0;

void procesA
{
  while(true) {
    while (lock != 0) { ; }
    lock = 1;
    critical_region();
    lock = 0;
    noncritical_region();
  }
}
```

```
1  //state of lock =  1
2
3    void procesB
4    {
5      while(true) {
6        while (lock != 0) { ; } //enter_section
7        lock = 1;               //
8        critical_region();
9        lock = 0;               //exit_section
10       noncritical_region();
11     }
12   }
```

- Porušuje 1. podmienku vzájomného vylučovania.
  V KO môže byť najviac 1 proces.
- **Toto vôbec nefunguje!!**

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

- Druhý pokus - skúsme mať premenú, ktorou riadime striedanie. Vždy je niekto na rade.

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;                //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;                //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;                //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = A
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;                //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Strict Alternation

```
#define A 0
#define B 1
int turn = A;

void procesA
{
  while(true) {
    while (turn != A) { ; }
    critical_region();
    turn = B;
    noncritical_region();
  }
}
```

```
1  //state of turn = B
2
3
4
5    void procesB
6    {
7      while(true) {
8        while (turn != B) { ; } //enter_section
9        critical_region();
10       turn = A;               //exit_section
11       noncritical_region();
12     }
13   }
```

- Porušuje 3. podmienku vzájomného vylučovania.
  Proces mimo KO nesmie brániť inému procesu vstupovať do KO.
- **Riešenie z núdze!!??** Závisí od toho čo je v noncritical_region().

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =false
2  //state of inB =false
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

- Tretí pokus - každý proces deklaruje záujem o vstup do KO. Ak druhý proces má záujem tak prvý čaká.

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =false
2  //state of inB =false
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =true
2  //state of inB =false
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =true
2  //state of inB =false
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =true
2  //state of inB =true
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =true
2  //state of inB =true
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

Busy Waiting - Manifest Interest

```
int inA = false;
int inB = false;

void procesA
{
  while(true) {
    inA = true;
    while (inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA =true
2  //state of inB =true
3
4    void procesB
5    {
6      while(true) {
7        inB = true;
8        while (inA == true) {;} //enter_section
9        critical_region();
10       inB = false;            //exit_section
11       noncritical_region();
12     }
13   }
```

- Porušuje 4. podmienku vzájomného vylučovania.
  Čas čakania procesu na vstup do KO musí byť konečný.
- **Toto vôbec nefunguje!!**

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = false
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

- Štvrtý pokus - skombinujme druhý a tretí pokus. Budeme striedať procesy a zároveň proces musí mať záujem.

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = false
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = B
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = B
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = true
2  //state of inB  = true
3  //state of turn = B
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = false
2  //state of inB  = true
3  //state of turn = B
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

Busy Waiting - Peterson

```
#define A 0
#define B 1
int inA, inB = false;
int turn = A;

void procesA {
  while(true) {
    inA = true;
    turn = A;
    while (turn == A
        && inB == true) {;}
    critical_region();
    inA = false;
    noncritical_region();
  }
}
```

```
1  //state of inA  = false
2  //state of inB  = false
3  //state of turn = A
4
5
6    void procesB {
7      while(true) {
8        inB = true;
9        turn = B;
10       while (turn == B
11           && inA == true) {;} //enter_section
12       critical_region();
13       inB = false;            //exit_section
14       noncritical_region();
15     }
16   }
```

- Funkčné riešenie ... **len pre dva procesy!!**

Busy Waiting - Špeciálne inštrukcie

- Inštrukcie zaručujú atomickosť:
  - na CPU: z princípu spracovania inštrukcií.
  - v pamäti: zablokovaním prístupu ostatných CPU na zbernicu.
- Funkčné riešenie ... **nie každý procesor má špeciálne inštrukcie!!**

TSL - Test and Set Lock

```
enter_section: TSL REG, LOCK
               CMP REG, #0
               JNE enter_section
               RET

exit_section:  MOV LOCK, #0
               RET
```

XCHG - Exchange

```
enter_section: MOV  REG, #1
               XCHG REG, LOCK
               CMP  REG, #0
               JNE  enter_section
               RET

exit_section:  MOV LOCK, #0
               RET
```

Busy Waiting - Nevýhody

- Cyklus čakania procesu na vstup do KO plytvá zdrojmi CPU.
- Problém inverzie priorít

Priority Inversion (inverzia priorít)

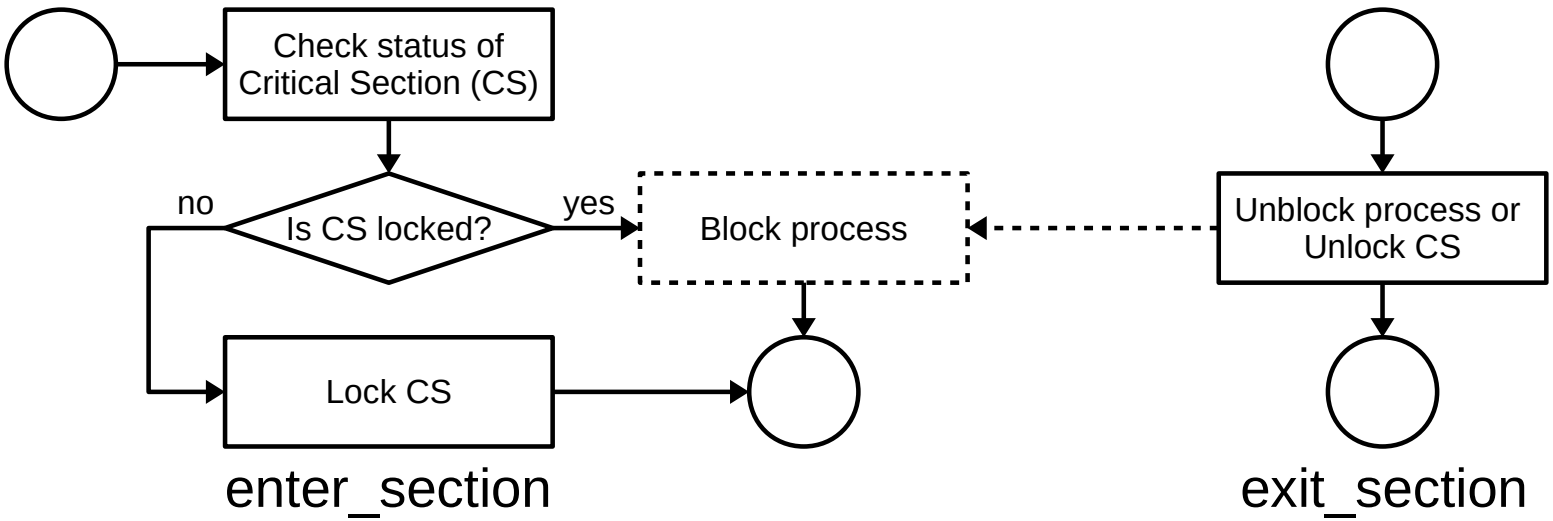
vzniká ak dva procesy H a L zdieľajú KO pričom proces H má vyššiu prioritu ako proces L. V prípade, že proces L sa dostane do KO ako prvý môže dôjsť k jeho preplánovaniu za proces H. Proces H následne už nemôže vstúpiť do KO a čaká v cykle na vstup. Nakoľko proces H má vyššiu prioritu môže dôjsť k situácií kedy sa proces L sporadicky alebo aj nikdy nedostane k CPU.

Sleep and WakeUp

Vzájomné vylučovanie - Sleep and WakeUp

Sleep and WakeUp

V prípade, že proces nemôže vstúpiť do KO je blokovaný (uspaný). Proces je prebudený po uvoľnení KO.



Sleep and WakeUp - Semafór

Semafór

je univerzálny prostriedok synchronizácie procesov.

Semafór

je abstraktný dátový typ so zadefinovanými operáciami init(), up() a down() pre ktoré platí, že počet **ukončených** operácií down() musí byť vždy menší alebo rovný počtu **ukončených** operácií up(). Operácia init() definuje počiatočný počet ukončených operácií up().

Sleep and WakeUp - Semafór

Semafór - po lopate

- semafór je kladné celé číslo s a rad blokovaných procesov r
- semafór je zamknutý ak $s = 0$, inak je odomknutý
- *init*() nastaví počiatočnú hodnotu s a inicializuje prázdny r
- *down*() vkladá proces do r ak $s = 0$, inak zamyká semafór $s--$
- *down*() pre aktuálny proces neskončí lebo proces je blokovaný v r
- *up*() vyberá z r ak je neprázdny, inak odomyká semafór $s++$
- *init*(), *up*() a *down*() sú systémovými volaniami OS
- *up*() a *down*() musia byť atomické lebo sú to KO
  - *up*() a *down*() sú malé funkcie a preto atomickosť je zaručená použitím busy waiting.
  - zakázanie prerušení, špeciálne inštrukcie alebo softvérové riešenie

Sleep and WakeUp - Semafór

- Príklad ako by mohla vyzerať implementacia v kerneli po uspešnom spracovani systémového volania down alebo up.

```
sem_down:   TSL  REG, LOCK
            CMP  REG, #0
            JNE  sem_down       //check if other proces is doing up or down
            CMP  S, #0          //check if we are locked i.e. S is zero
            JNE  dec
            MOV  LOCK, #0       //unlock
            CALL process_sleep
            JMP  end
dec:        DEC  S
            MOV  LOCK, #0       //unlock
end:        RET                 //return to caller; process entered KO

sem_up:     TSL  REG, LOCK
            CMP  REG, #0
            JNE  sem_up         //check if other proces is doing up or down
            CMP  S, #0          //check if we are locked i.e. S is zero
            JNE  inc
            CALL wakeup_any
            JZE  end            //wakeup returns 1 if any proces was waiting
inc:        INC  S
end:        MOV  LOCK, #0       //unlock
            RET                 //return to caller; process left KO
```

Sleep and WakeUp - Binárny semafór

Binárny Semafór

je semafór, ktorého počiatočná hodnota je 0 alebo 1 pričom táto hodnota nikdy nie je väčšia ako 1.

- Vhodné riešenie pri vzájomnom vylučovaní.
- Inicializácia by mala nastať mimo procesov, ktoré semafór používajú.

```
int main()
{
  tSem lock;

  //sem_init(lock, 1); podla paradigmy
  lock.init(1);

  create_processes(lock);
}
```

```
1  void processFn(tSem & lock)
2  {
3    while (true) {
4      lock.down();
5      critical_region();
6      lock.up();
7      noncritical_region();
8    }
9  }
```

Sleep and WakeUp - Mutex

- Operácie nad semafórom vyžadujú systémové volanie jadra OS.
- V prípade, že KO je zdielaná len medzi thread-mi jedného procesu:
  - Semafór môže spôsobovať predčasné prepnutie procesu (trpí latencia procesu).
  - V prípade, že máme len user thread-y. Semafór nezablokuje thread ale celý proces (semafór nefunguje).

Mutex (**Mut**ual **Ex**clusion)

je softvérový, knižničný ekvivalent semafóru určený pre vzájomné vylučovanie thread-ov.

Sleep and WakeUp - Mutex

```
mutex_down: TSL  REG, MUTEX     //copy variable MUTEX to REG and do set 1
            CMP  REG, #0        //was MUTEX 0?
            JZE  ok             //MUTEX = 0 -> jump to label ok
            CALL thread_yield   //MUTEX = 1 -> schedule another thread
            JMP  mutex_down     //once we are scheduled again repeat test
ok:         RET                 //return to caller; critical section entered

mutex_up:   MOV MUTEX, #0       //store 0 to mutex
            RET                 //return to caller
```

Sleep and WakeUp - POSIX Mutex

Thread call	Description
Pthread_mutex_init	Create a mutex
Pthread_mutex_destroy	Destroy an existing mutex
Pthread_mutex_lock	Acquire a lock or block
Pthread_mutex_trylock	Acquire a lock or fail
Pthread_mutex_unlock	Release a lock

Sleep and WakeUp - Monitor

- Problémom semafórov je náchylnosť na chyby programátora.
- Takéto chyby sa veľmi ťažko odhaľujú nakoľko vznikajú náhodne.
- V mnohých ohľadoch je semafór považovaný za príliš nízku abstrakciu synchronizácie.

Monitor

je konceptom rozšírenej syntaxe[1] programovacieho jazyka, ktorý definuje KO ako objekt. Monitor udržuje svoj interný stav a definuje operácie, ktoré je nad ním možné vykonať. V jednom časovom bode iba jedna z definovaných operácií sa môže vykonávať.

[1] Tannenbaum uvádza aj teoretické programové riešenie Monitora. Vo všeobecnosti sa pod monitorom rozumie syntaktické rozšírenie programovacieho jazyka.

Sleep and WakeUp - Monitor

- Programátor definuje kritickú oblasť vytvorením objektu monitora.
- Vzájomné vylučovanie procesov je zabezpečené kompilátorom.
- Monitor vyžaduje zavedenie syntaxe do jazyka.
- Java je jedným z takých programovacích jazykov. (viď. synchronized)

SW riešenia pre N procesov

Vzájomné vylučovanie - N procesov

Funguje bez nutných zmien

- Busy Waiting - špeciálne inštrukcie
- Sleep and WakeUp - semafór, mutex, monitor

Problem

- Busy Waiting - softvérové riešenia

Peterson 2 procesy

```
1  #define A 0
2  #define B 1
3  int inA, inB = false;
4  int turn = A;
5
6  void procesA()
7  {
8      inA = true;                              //
9      turn = A;                                //enter_section
10     while (turn == A && inB == true) { ; } //
11     critical_region();
12     inA = false;                             //exit_section
13     noncritical_region();
14 }
```

Peterson 2 procesy

```
1  #define A 0
2  #define B 1
3  int inA, inB = false;
4  int turn = A;
5
6  void exit_section()
7  {
8      inA = false;
9  }
10
11 void enter_section()
12 {
13     inA = true;
14     turn = A;
15     while (turn == A && inB == true) { ; }
16 }
```

- Čo bude treba spraviť aby sme podporovali N procesov?

Peterson N procesov - pokus

```
1  #define N 100
2
3  int proc_in[N];                //int inA, inB = false;
4  int proc_turn = 0;             //int turn = A;
5
6  void exit_section(int pid)
7  {
8      proc_in[pid] = false;      //inA = false;
9  }
10
11 void enter_section(int pid)
12 {
13     proc_in[pid] = true;       //inA = true;
14     proc_turn = pid;           //turn = A;
15     while (proc_turn == pid && other_in(pid)) { ; }
16 }
```

- *other_in()* - loop: ma nejaky iny proces zaujem?
- Nefunguje to! Prečo?

Peterson N procesov

```
1  #define N 100
2  #define MAX_LEVEL N - 1
3  int proc_in_level[N];
4  int proc_turn[MAX_LEVEL];
5
6  void exit_section(int pid)
7  {
8      proc_in_level[pid] = -1;
9  }
10 ...
```

Peterson N procesov

```
11 void enter_section(int pid)
12 {
13     for (int level = 0; level < MAX_LEVEL; level++) {
14         proc_in_level[pid] = level;
15         proc_turn[level] = pid;
16 
17         while (proc_turn[level] == pid &&
18                other_proc_has_higher_level(pid, level)) {;}
19     }
20 }
21 
22 bool other_proc_has_higher_level(int pid, int level)
23 {
24     for (int oPid = 0; oPid < N; oPid++) {
25         if (oPid == pid)
26             continue;
27 
28         if (proc_in_level[oPid] >= level)
29             return true;
30     }
31     return false;
32 }
```

- Tak toto funguje ale za akú cenu?
  - s rastúcim N rastie čas vykonania kvadraticky (N^2)

Bakery Algorithm

```
1  #define N = 100
2  int ticketN = 0; //zanedbajme ze tiket moze pretiect
3  int choosing[N];
4  int ticket[N];
5
6  void enter_section(int pid)
7  {
8      choosing[pid] = true;    //cisc OK, risc ?
9      ticket[pid] = ++ticketN; //sanca, ze viac procesov dostane rovnake cislo
10     choosing[pid] = false;
11
12     for (int oPid = 0; oPid < N; oPid++)
13     {
14         while (choosing[oPid] == true) {;}
15         while (ticket[oPid] != 0 && (ticket[oPid] < ticket[pid] ||
16                                      (ticket[oPid] == ticket[pid] && oPid < pid)))
17         {;}
18     }
19 }
20
21 void exit_section(int pid)
22 {
23   ticket[pid] = 0;
24 }
```

Zhrnutie

Zhrnutie

- Zdieľanie prostriedkov výpočtového systému je jeho neoddeliteľnou súčasťou.
- Len ťažko nájdeme systém, v ktorom by nebolo potrebné riadiť prístup ku zdrojom.
- OS sú tým softvérom, ktoré nutne musia poskytnúť programátorovi prostriedky, ktoré odstránia súperiace podmienky.
- OS ako manažér tieto prostriedky využíva sám.
- Bežnými prostriedkami IPC sú:
  - Zakázanie prerušení - bežne používané v jadre OS (ak jedno CPU)
  - Špeciálne inštrukcie TSL, XCHG - bežne používané pre riadenie prístupu ku krátkym KO (ako je down() a up() semafóru).
  - Softvérové riešenia - bežne používané pre riadenie prístupu ku krátkym KO ak CPU nemá špeciálne inštrukcie.
  - Semafór - univerzálne použitie na synchronizáciu procesov.
  - Mutex - používané na synchronizáciu thread-ov.
  - Monitory - ako súčasť vyšších programovacích jazykov (java).

Čo robiť do ďalšej prednášky

- Znova prečítať kapitoly 2.3, 2.5 a 2.7 z Tanenbauma