

**BRNO UNIVERSITY OF TECHNOLOGY**

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

Kolejní 4, 616 00 Brno

tel.: +420 5 4114 1113 fax: +420 5 4114 1123

E-mail: [kucera@feec.vutbr.cz](mailto:kucera@feec.vutbr.cz) <http://taceo.eu>



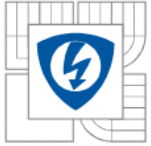
# Operační Systémy Reálného Času

V.

Pavel Kučera

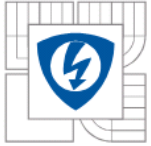
CAK, E112

[kucera@feec.vutbr.cz](mailto:kucera@feec.vutbr.cz)



# Obsah

1. Synchronizace pomocí zámků
2. Synchronizace pomocí mutexů
3. Synchronizace pomocí událostí
4. RTX – Events
5. RTX – Mutex

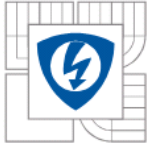


# Synchronizace mezi procesy

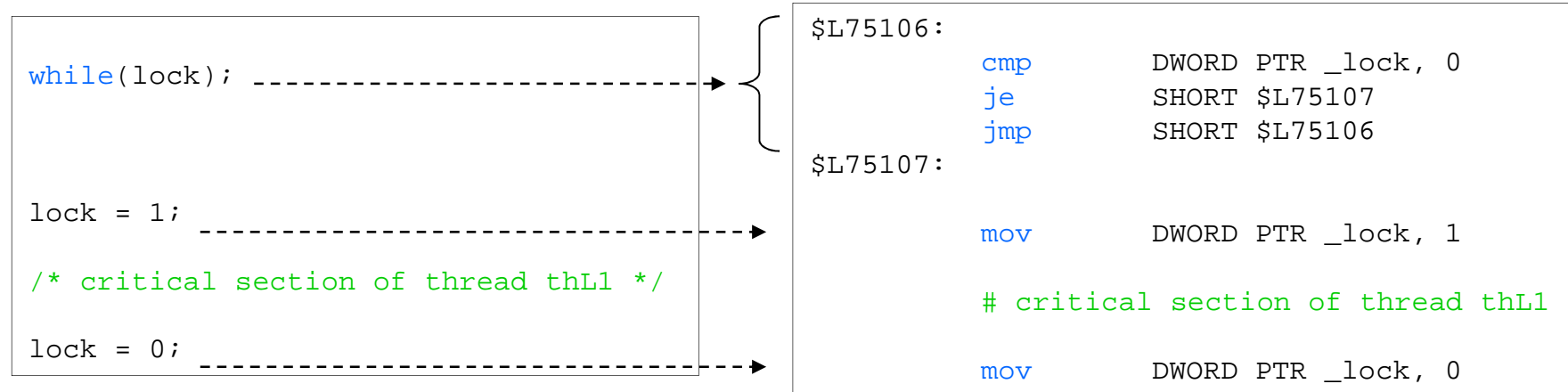
```
int lock = 0;

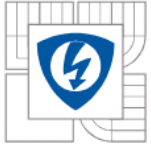
ULONG RTFCNDCL thL1( void *nContext ) {
    while(lock);
    lock = 1;
    /* critical section of thread thL1 */
    lock = 0;
    return 0;
}

ULONG RTFCNDCL thL2( void *nContext ) {
    while(lock);
    lock = 1;
    /* critical section of thread thL2 */
    lock = 0;
    return 0;
}
```



# Synchronizace mezi procesy



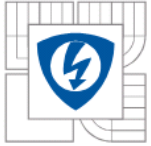


# Synchronizace mezi procesy

```
int lock = 0;

ULONG RTFCNDCL thL1( void *nContext ) {
    while(lock);
    lock = 1;
    /* critical section of thread thL1 */
    lock = 0;
    return 0;
}

ULONG RTFCNDCL thL2( void *nContext ) {
    while(lock);
    lock = 1;
    /* critical section of thread thL2 */
    lock = 0;
    return 0;
}
```

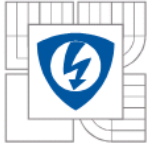


## Kritická sekce

Úsek kódu, který přistupuje ke sdílenému zdroji a který nesmí být v jednom okamžiku využíván více než jedním vláknem.

Je tedy nutné zabránit procesoru opuštění kritické sekce, jakmile do ní jednou vstoupí.

- zakázat přerušení
- použít semaforey
- použít mutexy
- použít zámky



# Spinlocks

```
lock:
    dd    0                # The lock variable. 1 = locked, 0 = unlocked.

spin_lock:
    mov   eax, 1          # Set the EAX register to 1.

loop:
    xchg  eax, [lock]     # Atomically swap the EAX register with
                        # the lock variable.
                        # This will always store 1 to the lock, leaving
                        # previous value in the EAX register.

    test  eax, eax        # Test EAX with itself. Among other things, this will
                        # set the processor's Zero Flag if EAX is 0.
                        # If EAX is 0, then the lock was unlocked and
                        # we just locked it.
                        # Otherwise, EAX is 1 and we didn't acquire the lock.

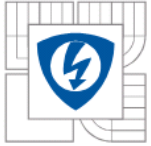
    jnz  loop            # Jump back to the XCHG instruction if the Zero Flag is
                        # not set, the lock was locked, and we need to spin.

    ret                  # The lock has been acquired, return to the calling
                        # function.

spin_unlock:
    mov   eax, 0          # Set the EAX register to 0.

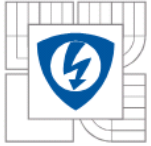
    xchg  eax, [lock]     # Atomically swap the EAX register with
                        # the lock variable.

    ret                  # The lock has been released.
```



# Spinlocks

- Atomická struktura umožňující synchronizaci mezi procesy.
- V podstatě nelze realizovat ve vyšších programovacích jazycích.
- Ve strojovém kódu lze realizovat při použití atomických instrukcí (tj. takových, které nemohou být přerušeny) typu test-and-set, fetch-and-add, compare-and-swap.
- Nelze v jednoúlohových RTOS.
- Jednoduchá implementace, časté použití v kernelech a driverech.

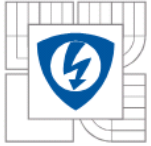


## Mutex (mutual exclusion)

Mutual exclusion – v jednom okamžiku zdroj patří pouze jednomu tasku.

Pokud task čeká na zdroj chráněný mutexem, čeká tak dlouho, dokud zdroj nebude uvolněn. **Ovšem jen pokud použijeme k přístupu mutex!**

Pokud je task, který mutex vlastní, předčasně ukončen (čímž sdílený zdroj řádně neuvolní), jsou o tom ostatní úlohy v procesu informovány (WAIT\_ABANDONED); stav sdíleného prostředku je pak nekonzistentní!

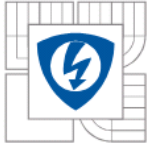


## Mutex (mutual exclusion)

Pokud na vlastnictví mutexu čeká více tasků, získá ho ten s nejvyšší prioritou. Pokud mají tasky stejnou prioritu, získá mutex ten task, který požádal o mutex nejdříve (tj. čeká nejdéle), tj. round-robin policy.

Problémy s mutexy:

- Vytváření mutexu a okamžité vlastnění mutexu (race).
- Mutex není uvolněn a task skončil -> deadlock.
- Blokují části kódu (režije).



# Mutex – Dekkersův algoritmus

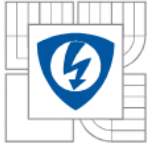
```
bool flag_0 = false; //proces 0 chce vstoupit do kritické sekce
bool flag_1 = false; //proces 1 chce vstoupit do kritické sekce
int turn = 0;        //id procesu
//int turn = 1;
```

## process\_0

```
flag_0 = true;
while(flag_1) {
    if(turn != 0) {
        flag_0 = false;
        while(turn != 0) {
        }
        flag_0 = true;
    }
}
// critical section
turn = 1;
flag_0 = false;
```

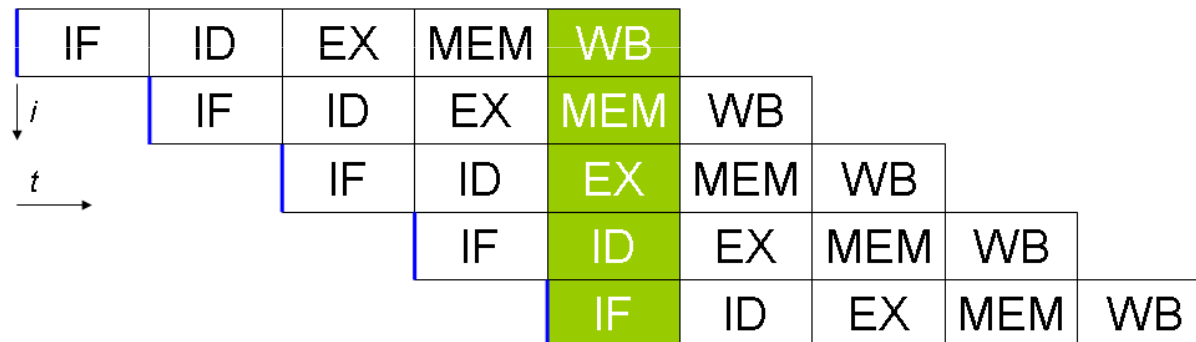
## process\_1

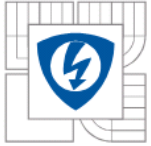
```
flag_1 = true;
while(flag_0) {
    if(turn != 1) {
        flag_1 = false;
        while(turn != 1) {
        }
        flag_1 = true;
    }
}
// critical section
turn = 0;
flag_1 = false;
```



# Mutex – problémy

- out-of-order řetězení
- pipelining





## Event – událost

Události jsou synchronizačním prostředkem mezi tasky IPC. Neblokují části kódu jako mutexy – jsou bodové.

RTOS obvykle poskytují operace pro vytvoření události, čekání na událost a signalizace události.

RTOS obvykle umožňují nastavit event jako manual-reset nebo auto-reset a pulse nebo set.

Problémy s eventy:

- Správný výběr eventu (manual-reset/auto-reset). První uvolněný task nastaví při auto-reset event na non-signaled).
- Správná signalizace eventu (pulse/set). Pulse nastaví stav event do signaled, uvolní všechny čekající tasky a poté se sám zruší.