

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, <http://taceo.eu>



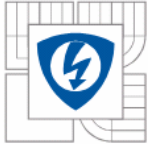
Operační Systémy Reálného Času

VII.

Pavel Kučera

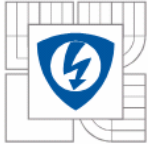
CAK, E112

kucera@feec.vutbr.cz



Obsah

1. Race condition
2. Producent - konzument
3. Semaforey
4. Sdílená paměť

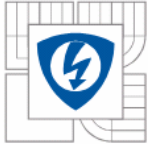


Race condition

```
int sum = 0;

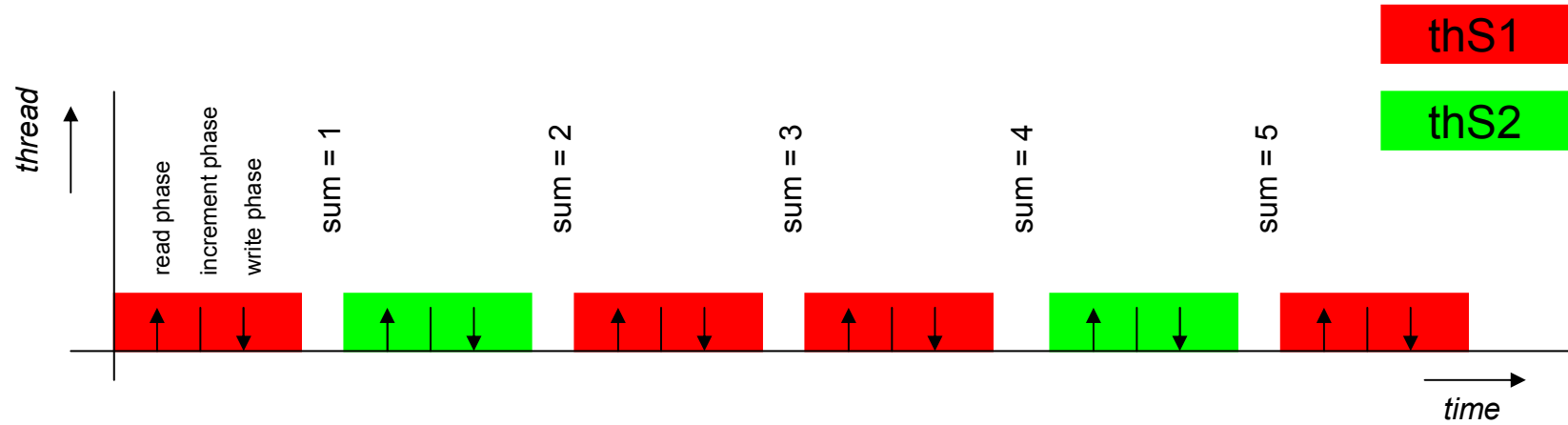
ULONG RTFCNDCL thS1( void *nContext ) {
int is;
    while(TRUE);
        is = sum;
        is ++;
        sum = is;
    }
return 0;
}

ULONG RTFCNDCL thS2( void *nContext ) {
int is;
    while(TRUE);
        is = sum;
        is ++;
        sum = is;
    }
return 0;
}
```

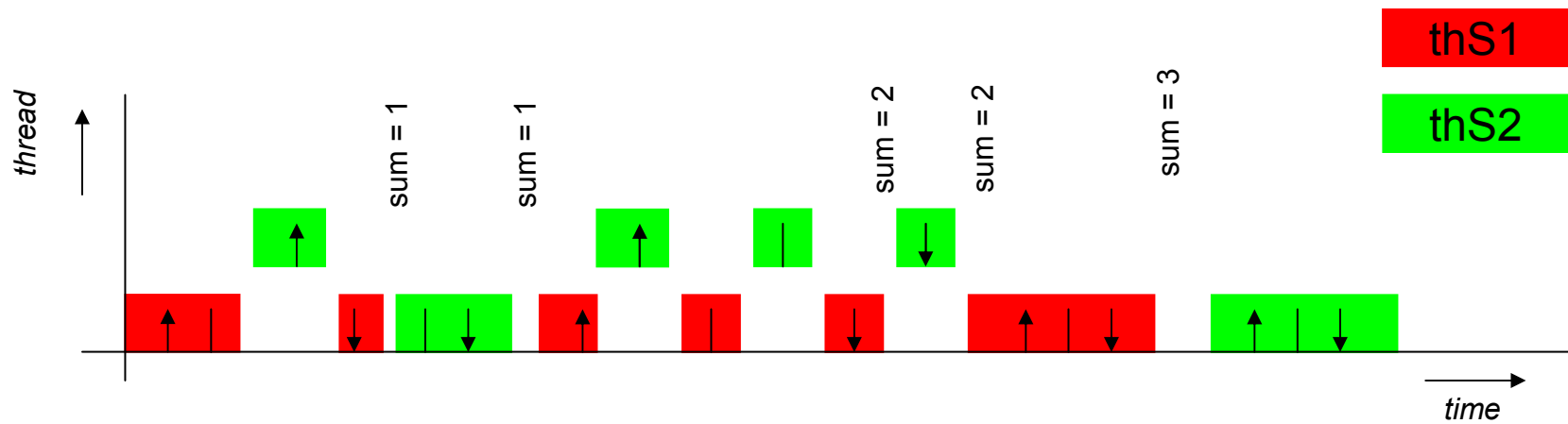


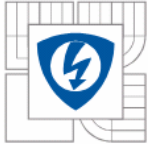
Race condition

Scenario 1



Scenario 2





Race condition

```
int sum = 0;

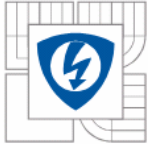
ULONG RTFCNDCL thS1( void *nContext ) {
    int is;
    while(TRUE);
        is = sum;
        is ++;
        sum = is;
    }
    return 0;
}

ULONG RTFCNDCL thS2( void *nContext ) {
    int is;
    while(TRUE);
        is = sum;
        is ++;
        sum = is;
    }
    return 0;
}
```

```
int sum = 0;

ULONG RTFCNDCL thS1( void *nContext ) {
    int is;
    while(TRUE);
        RtWaitForSignleObject(hMutex, INFINITE);
        is = sum;
        is ++;
        sum = is;
        RtReleaseMutex(hMutex);
    }
    return 0;
}

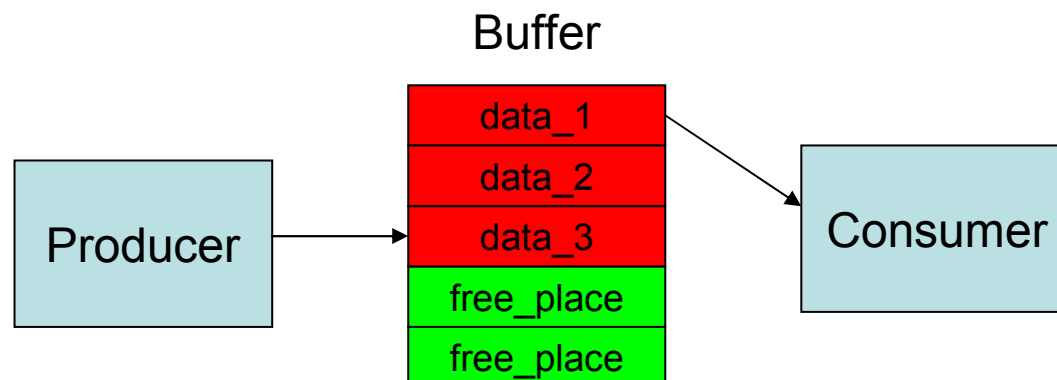
ULONG RTFCNDCL thS2( void *nContext ) {
    int is;
    while(TRUE);
        RtWaitForSignleObject(hMutex, INFINITE);
        is = sum;
        is ++;
        sum = is;
        RtReleaseMutex(hMutex);
    }
    return 0;
}
```

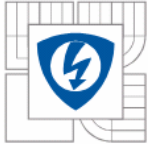


Producent - konzument

Producent vytváří data ke zpracování a konzument je zpracovává. Data jsou mezi producentem a konzumentem předávána přes buffer fixní délky. Je nutné zajistit:

1. Aby producent nevkládal další data do plného bufferu.
2. Aby konzument nečetl data, když je buffer prázdný.



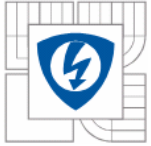


Produzent - konzument

```
int buffer[BUFFER_SIZE];
int buffer_count = 0;

ULONG RTFCNDCL Producer( void *nContext ) {
int value;
    while(TRUE);
        value = GetValue();
        if (buffer_count == BUFFER_SIZE) RtWaitForSingleObject(hEventBufferEmpty, INFINITE);
        buffer[buffer_count] = value;
        buffer_count ++;
        if (buffer_count == 1) RtPulseEvent(hEventBufferContainData);
    }
    return 0;
}

ULONG RTFCNDCL Consumer( void *nContext ) {
int value;
    while(TRUE);
        if (buffer_count == 0) RtWaitForSingleObject(hEventBufferContainData, INFINITE);
        value = buffer[buffer_count - 1];
        buffer_count --;
        if (buffer_count == BUFFER_SIZE - 1) RtPulseEvent(hEventBufferEmpty);
        ProcessValue(value);
    }
    return 0;
}
```

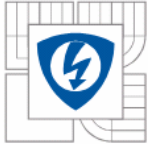


Producent - konzument

```
int buffer[BUFFER_SIZE];
int buffer_count = 0;

ULONG RTFCNDCL Producer( void *nContext ) {
int value;
    while(TRUE);
        value = GetValue();
        if (buffer_count == BUFFER_SIZE) RtWaitForSingleObject(hEventBufferEmpty, INFINITE);
        buffer[buffer_count] = value;
        buffer_count ++;
        if (buffer_count == 1) RtPulseEvent(hEventBufferContainData);
    }
    return 0;
}

ULONG RTFCNDCL Consumer( void *nContext ) {
int value;
    while(TRUE);
        if (buffer_count == 0) RtWaitForSingleObject(hEventBufferContainData, INFINITE);
        value = buffer[buffer_count - 1];
        buffer_count --;
        if (buffer_count == BUFFER_SIZE - 1) RtPulseEvent(hEventBufferEmpty);
        ProcessValue(value);
    }
    return 0;
}
```



Semaforey

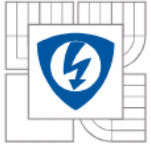
Synchronizační objekt obsahující vnitřní čítač, jenž se může nacházet v rozmezí $\langle 0, MAX \rangle$.

Pokud je stav čítače > 0 (minimálně jeden sdílený zdroj je k dispozici), pak je semafor v signálním stavu a libovolné vlákno, které na čeká na semaforem chráněný zdroj, může být uvolněno.

Je-li stav čítače roven 0 (žádné sdílené zdroje nejsou k dispozici), pak je objekt v nesignálním stavu a vlákna, která na zdroj čekají jsou blokována.

Vlákno, které je semaforem uvolněno, automaticky sníží hodnotu vnitřního čítače semaforu o 1.

Vlákno **může** zvětšit hodnotu čítače semaforu o definovanou hodnotu, která odpovídá počtu sdílených zdrojů, které jsou k dispozici. Hodnota čítače však nesmí překročit hodnotu MAX.

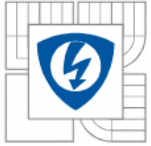


Semafor

```
int buffer[BUFFER_SIZE];
int buffer_count = 0;

ULONG RTFCNDCL Producer( void *nContext ) {
int value;
while(TRUE);
value = GetValue();
RtWaitForSingleObject(hSemaphoreFull, INFINITE);
buffer[buffer_count] = value;
buffer_count ++;
RtReleaseSemaphore(hSemaphoreEmpty, 1, NULL)
}
return 0;
}

ULONG RTFCNDCL Consumer( void *nContext ) {
int value;
while(TRUE);
RtWaitForSingleObject(hSemaphoreEmpty, INFINITE);
value = buffer[buffer_count - 1];
buffer_count --;
ProcessValue(value);
RtReleaseSemaphore(hSemaphoreFull, 1, NULL)
}
return 0;
}
```

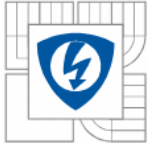


Sdílená paměť

Sdílená paměť je objekt, který slouží k IPC jak mezi RTSS procesy navzájem, tak mezi RTSS procesy a Win32 aplikacemi.

Sdílená paměť je vždy vytvořena v nestránkované paměti a je přístupná jako virtuální paměť daného procesu (který ji vytvořil nebo otevřel).

Již vytvořená sdílená paměť je přístupná ostatním procesům jejím handlem, virtuální adresou začátku (pointer).



Sdílená paměť

Aby mohl být proces úspěšně ukončen MUSÍ uzavřít všechny své handly na sdílenou paměť.

Pokud všechny procesy uzavřou svoje handly na příslušnou paměť, tak je tato navracena do nestránkované paměti jádra a objekt sdílené paměti přestane existovat.

Vlákno každého procesu musí mít **VLASTNÍ** handle na sdílenou paměť a vlastní pointer na její umístění ve virtuálním prostoru procesu.