



## RtCreateMutex(...)

Funkce `RtCreateMutex` vytvoří synchronizační objekt typu mutex.

### Prototyp

HANDLE

`RtCreateEvent (`

**LPSECURITY\_ATTRIBUTES**     *lpEventAttributes,*

**BOOL**                         *bInitialOwner,*

**LPCTSTR**                    *lpName*

`);`



## RtCreateMutex(...)

### Parametry

*lpEventAttributes* v RTSS ignorováno.

*bInitialOwner* určuje, zda-li bude vláknu, které mutex vytváří, mutex po vytvoření touto funkcí vlastnit:  
*TRUE* vlákno bude mutex po jeho vytvoření vlastnit,  
*FALSE* vlákno nebude mutex po jeho vytvoření vlastnit.

*lpName* ukazatel na řetězec označující název objektu. Řetězec může obsahovat libovolný znak kromě znaku backslash '\'. Maximální délka řetězce je *RTX\_MAX\_PATH*. Pokud je tento parametr *NULL*, pak je objekt vytvořen bez názvu. Názvy objektů jsou case-sensitive.



## RtCreateMutex(...)

### Návratová hodnota

Pokud funkce uspěje, návratovou hodnotou je handle na objekt typu událost. Pokud funkce selže, pak vrátí *NULL*; voláním funkce `GetLastError` lze získat další informace o chybě.

### Poznámky

Pokud funkce `GetLastError` vrátí *ERROR\_ALREADY\_EXISTS*, pak mutex tohoto názvu (parametr *lpName*) již existuje a parametr *bInitialOwner* je ignorován, protože již byl nastaven procesem, který mutex vytvořil.

Pokud parametr *lpName* obsahuje jméno již existujícího objektu, který však není typu mutex, pak funkce selže a `GetLastError` vrátí *ERROR\_INVALID\_HANDLE*; objekty, které lze pojmenovat (tj. událost, mutex, semafor a sdílená paměť) totiž sdílejí v RTSS stejný prostor jmen.



## RtCreateMutex(...)

### Poznámky

Libovolné vlákno v daném procesu může pomocí funkcí čekání `RtWaitForSingleObject` nebo `RtWaitForMultipleObjects` čekat na přidělení mutexu.

Stav objekt typu mutex je signalizován tehdy, když mutex **není** vlastněn žádným vláknem. Vlákno, které mutex vytvořilo, si může nárokovat vlastnictví mutexu pomocí parametru `bInitialOwner`. Ostatní vlákna musí vlastnictví mutexu získat pomocí jedné z výše uvedených funkcí čekání.

Pokud je mutex v signálním stavu (tj. není vlastněn žádným vláknem), pak vlákno s **nejvyšší** prioritou, které čeká na přidělení mutexu, tento obdrží. Mutex je pak automaticky převeden do nesignálního stavu. Pokud čeká více vláken se stejnou prioritou na přidělení mutexu, obdrží ho to vlákno, které čeká **nejdéle**. V jednom okamžiku může mutex vlastnit pouze **jedno** vlákno. Vlákno uvolní mutex, a převede tak objekt zpět do signálního stavu, pomocí funkce `RtReleaseMutex`.



## RtCreateMutex(...)

### Poznámky

Vláknem, které již mutex vlastní, může tentýž mutex využít v další funkci čekání. Aby nedošlo k deadlocku, jakékoliv další čekání vlákna na mutex, který již vlastní, nezpůsobí zastavení vlákna. Nicméně při uvolňování mutexu musí být funkce `RtReleaseMutex` zavolána tolikrát, kolikrát vlákno žádalo vlastnictví téhož mutexu ve funkcích čekání.

Procesy mohou mezi sebou sdílet objekty typu mutex. Proces otevře objekt typu mutex, který před tím vytvořil jiný proces, buďto pomocí funkce `RtOpenMutex` nebo pomocí funkce `RtCreateMutex`.

Dva a více procesů mohou ve svých vláknech volat funkci `RtCreateMutex`, aby vytvořily mutex stejného jména. V tom případě první volání funkce `RtCreateMutex` mutex vytvoří a každé další volání jen vrátí handle na tento již existující objekt. Tato technika umožní dvěma a více procesům sdílet stejný mutex, aniž by musel programátor zajistit, aby byl proces, který mutex vytváří spuštěn jako první. V tom případě je však nutné nastavit parametr `bInitialOwner` na `FALSE`, jinak by bylo nemožné rozhodnout o tom, který proces získal vlastnictví mutexu při jeho vytvoření.

Handle na objekt typu mutex je možné uzavřít pomocí funkce `RtCloseHandle`. Operační systém uzavře handle na objekt typu událost automaticky při ukončení procesu, který handle vytvořil. Objekt typu mutex je vymazán, když jsou uzavřeny **všechny** handlers, které se na něho odkazovaly.



## RtOpenMutex(...)

Funkce `RtOpenMutex` vrátí handle na existující pojmenovaný objekt typu mutex.

### Prototyp

HANDLE

`RtOpenMutex(`

    DWORD        *DesiredAccess,*

    BOOL         *bInheritHandle,*

    LPCTSTR     *lpName*

`);`

### Parametry

*DesiredAccess*       v RTSS ignorováno.

*bInheritHandle*     v RTSS ignorováno.

*lpName*             ukazatel na řetězec obsahující název objektu typu mutex, který chceme otevřít. Názvy objektů jsou case-sensitive.



## RtOpenMutex(...)

### Návratová hodnota

Pokud funkce uspěje, návratovou hodnotou je handle na objekt typu událost. Pokud funkce selže, pak vrátí *NULL*; voláním funkce `GetLastError` lze získat další informace o chybě.

### Poznámky

Funkce umožňuje více procesům, sdílet tentýž mutex. Funkce uspěje pouze tehdy, pokud některý proces objekt typu mutex daného názvu již vytvořil pomocí funkce `RtCreateMutex`.

Handle na objekt typu mutex je možné uzavřít pomocí funkce `RtCloseHandle`. Operační systém uzavře handle na objekt typu mutex automaticky při ukončení procesu, který handle vytvořil. Objekt typu mutex je vymazán, když jsou uzavřeny *všechny* handlery, které se na něho odkazovaly.



## RtReleaseMutex(...)

Funkce `RtReleaseMutex` uvolní vlastnictví mutexu.

### Prototyp

BOOL

```
RtReleaseMutex(  
    HANDLE    hMutex  
);
```

### Parametry

*hMutex*

handler na objekt typu mutex otevřený buďto funkcí `RtCreateMutex` nebo `RtOpenMutex`.



## RtReleaseMutex(...)

### Návratová hodnota

Pokud funkce uspěje, návratová hodnota je *TRUE*.

Pokud funkce selže, návratová hodnota je *FALSE*; voláním funkce `GetLastError` lze získat další informace o chybě.

### Poznámky

Funkce selže (vrátí *FALSE*) pokud volající vlákno mutex *ne*vlastní.

Vlákno, které již mutex vlastní, může tentýž mutex využít v další funkci čekání. Aby nedošlo k deadlocku, jakékoliv další čekání vlákna na mutex, který již vlastní, nezpůsobí zastavení vlákna. Nicméně při uvolňování mutexu musí být funkce `RtReleaseMutex` zavolána tolikrát, kolikrát vlákno žádalo vlastnictví téhož mutexu ve funkcích čekání.



## MRTS – RTX6.5



```
#define SHARED_MEMORY_MUTEX_NAME          "sg_SharedmemoryMutex.Name"  
HANDLE hShmMutex = NULL;                //Shared Memory Mutex Handler
```

---

```
/*  
* Create hShmMutex object  
*/  
hShmMutex = RtCreateMutex(NULL, FALSE, TEXT(SHARED_MEMORY_MUTEX_NAME));  
if(GetLastError() == ERROR_ALREADY_EXISTS) {  
    RtPrintf("<ERROR> SG->SGIni::RtCreateMutex() ShmMutex object already exists!\n");  
    RtPrintf("<INFO> SG->Process will be terminated.\n");  
    return FALSE;  
}  
if(hShmMutex == NULL) {  
    RtPrintf("<ERROR> SG->SGIni::RtCreateMutex() Can not create ShmMutex object!\n");  
    RtPrintf("<INFO> SG->Process will be terminated.\n");  
    return FALSE;  
}  
RtPrintf("<INFO> SG->ShmMutex was created, HANDLER = 0x%X.\n", hShmMutex);
```

---

```
DWORD dw2 = RtWaitForSingleObject(hShmMutex, 0);  
if (dw2 == WAIT_OBJECT_0) {  
    /* critical section */  
  
    RtReleaseMutex(hShmMutex);  
}  
}
```