



RtCreateEvent(...)

Funkce `RtCreateEvent` vytvoří synchronizační objekt typu událost (event).

Prototyp

HANDLE

`RtCreateEvent` (

 LPSECURITY_ATTRIBUTES *lpEventAttributes*,

 BOOL *bManualReset*,

 BOOL *bInitialState*

 LPCTSTR *lpName*

);



RtCreateEvent(...)

Parametry

lpEventAttributes v RTSS ignorováno.

bManualReset určuje, jakým způsobem bude probíhat resetování objektu (tj. uvedení objektu do nesignálního stavu):
TRUE objekt musí být resetován manuálně (tzv. manual-reset) zavoláním funkce `RtResetEvent`,
FALSE objekt bude resetován automaticky (tzv. auto-reset) ve chvíli, kdy jeho signalizace uvolní libovolné vlákno z čekání.

bInitialState počáteční stav objektu:
TRUE objekt **bude** po vytvoření v signálním stavu,
FALSE objekt **nebude** po vytvoření v signálním stavu.

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.



RtCreateEvent(...)

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 objekt tohoto názvu (parametr *lpName*) již existuje a parametry *bInitialState* a *bManualReset* jsou ignorovány, protože již byly nastaveny procesem, který objekt vytvořil. Již existující objekt typu událost lze otevřít funkcí `RtOpenEvent`.

Pokud parametr *lpName* obsahuje jméno již existujícího objektu, který však není typu událost, 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.



RtCreateEvent(...)

Poznámky

Libovolné vlákno v daném procesu může pomocí funkcí čekání

`RtWaitForSingleObject` nebo `RtWaitForMultipleObjects` čekat na událost.

Pokud tato nastane, tj. stav objektu se změní na signalizovaný, pak se příslušné vlákno z funkce čekání navrátí a pokračuje dál ve své činnosti.

Funkce `RtSetEvent` převede objekt do signálního stavu. Funkce `RtResetEvent` převede objekt do nesignálního stavu. Počáteční stav objektu po vytvoření je určen parametrem *bInitialState*.

Pokud je objekt typu manual-reset a nachází se v signálním stavu, tak v tomto stavu setrvá dokud není explicitně zavolána funkce `RtResetEvent`. Po celou dobu, kdy je objekt v signálním stavu, uvolní libovolné množství vláken, která buďto čekala na příchod události nebo začala na událost čekat.

Pokud je objekt typu auto-reset a nachází se v signálním stavu, tak v tomto stavu setrvá dokud neuvolní právě jedno vlákno ze stavu čekání; operační systém pak automaticky nastaví objekt do nesignálního stavu. Pokud žádné vlákno na událost nečeká, pak objekt setrvává v signálním stavu.



RtCreateEvent(...)

Poznámky

Procesy mohou mezi sebou sdílet objekty typu událost. Proces otevře objekt typu událost, který před tím vytvořil jiný proces, pomocí funkce `RtOpenEvent`.

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



RtOpenEvent(...)

Funkce `RtOpenEvent` vrátí handle na existující pojmenovaný objekt typu událost (event).

Prototyp

HANDLE

`RtOpenEvent (`

 DWORD *DesiredAccess,*

 BOOL *bInheritHandle,*

 LPCTSTR *lpName*

`);`

Parametry

DesiredAccess v RTSS ignorováno.

bInheritHandle v RTSS ignorováno.

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



RtOpenEvent(...)

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 uspěje pouze tehdy, pokud některý proces objekt typu událost daného názvu již vytvořil pomocí funkce `RtCreateEvent`.

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



RtSetEvent(...)

Funkce `RtSetEvent` nastaví objekt typu událost do signálního stavu.

Prototyp

BOOL

```
RtSetEvent (  
    HANDLE    hEvent  
);
```

Parametry

hEvent

handle na objekt typu událost otevřený buďto funkcí `RtCreateEvent` nebo `RtOpenEvent`.



RtSetEvent(...)

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

Objekt typu manual-reset zůstává v signálním stavu, dokud není explicitně zavolána funkce `RtResetEvent`. Po celou dobu, kdy je objekt v signálním stavu, uvolní libovolné množství vláken, která buďto čekala na příchod události nebo začala na událost čekat.

Objekt typu auto-reset uvolní právě **jedno** vlákno ze stavu čekání; operační systém pak automaticky nastaví objekt do nesignálního stavu.

Pokud na událost čeká více vláken, je uvolněno jen to vlákno, které má **nejvyšší** prioritu ze všech; pokud je takových čekajících vláken více, uvolněno je pouze to, které čeká **nejdéle**.

Pokud žádné vlákno na událost nečeká, pak objekt **setrvává** v signálním stavu.



RtResetEvent(...)

Funkce `RtResetEvent` nastaví objekt typu událost do nesignálního stavu.

Prototyp

BOOL

```
RtResetEvent(  
    HANDLE    hEvent  
);
```

Parametry

hEvent

handle na objekt typu událost otevřený buďto funkcí `RtCreateEvent` nebo `RtOpenEvent`.



RtResetEvent(...)

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

Objekt zůstává v nesignálním stavu, dokud není explicitně zavolána funkce `RtSetEvent` nebo `RtPulseEvent`. Po celou dobu, kdy je objekt v nesignálním stavu, **blokuje** vykonávání všech vláken, které čekají nebo začnou čekat na událost ve funkci `RtWaitForSingleObject` nebo `RtWaitForMultipleObjects`.



RtPulseEvent(...)

Funkce `RtPulseEvent` nastaví objekt typu událost do signálního stavu, uvolní příslušná vlákna, která na událost čekají a poté nastaví objekt do nesignálního stavu.

Prototyp

BOOL

```
RtPulseEvent(  
    HANDLE    hEvent  
);
```

Parametry

hEvent

handle na objekt typu událost otevřený buďto funkcí `RtCreateEvent` nebo `RtOpenEvent`.



RtPulseEvent(...)

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

Pro objekt typu `manual-reset` jsou uvolněna **všechna** vlákna, která na událost čekají a poté je objekt automaticky převeden do nesignálního stavu.

Pro objekt typu `auto-reset` je uvolněno právě **jedno** čekající vlákno; objekt je poté automaticky převeden do nesignálního stavu.

Pokud na událost čeká více vláken, je uvolněno jen to vlákno, které má **nejvyšší** prioritu ze všech; pokud je takových čekajících vláken více, uvolněno je pouze to, které čeká **nejdéle**.

Pokud žádné vlákno na událost nečeká, pak funkce objekt nastaví do nesignálního stavu, aniž by uvolnila jakékoliv vlákno.



MRTS – RTX6.5



```
HANDLE ev;
void _cdecl main( int  argc, char **argv, char **envp ) {
HANDLE th1, th2;

    ev = RtCreateEvent(0, TRUE, FALSE, "EVENT.OBJECT");
    if (ev == NULL) { RtPrintf("ev was NOT created.\n"); ExitProcess(1); }
    th1 = RtCreateThread( 0, 0, thE1, 0, CREATE_SUSPENDED, 0);
    if ( th1 == NULL ) { RtPrintf("Error: thE1 was NOT created!\n"); ExitProcess(1); }
    th2 = RtCreateThread( 0, 0, thE2, 0, CREATE_SUSPENDED, 0);
    if ( th2 == NULL ) { RtPrintf("Error: thE2 was NOT created!\n"); ExitProcess(1); }

    if (RtSetThreadPriority(th1, RT_PRIORITY_MIN + 1))
        RtPrintf("thE1 priority was set to MIN + 1.\n");
    if (RtSetThreadPriority(th2, RT_PRIORITY_MIN + 2))
        RtPrintf("thE2 priority was set to MIN + 2.\n");

    if (RtResumeThread(th1) != 0xFFFFFFFF)
        RtPrintf("thE1 was resumed.\n");

    Sleep(500);

    if (RtResumeThread(th2) != 0xFFFFFFFF)
        RtPrintf("thE2 was resumed.\n");

    Sleep(500);
    RtPrintf("Signaling event.\n");
    RtSetEvent(ev);
    Sleep(1000);
    RtCloseHandle(ev); RtCloseHandle(th1); RtCloseHandle(th2);
    RtPrintf("\n\n\n");
    ExitProcess(0);
}
```



MRTS – RTX6.5



```
ULONG RTFCNDCL thE1( void *nContext ) {
    RtPrintf("thE1 is running.\n");
    RtWaitForSingleObject(ev, INFINITE);
    RtPrintf("thE1 Event is signaled.\n");
    return 0;
}

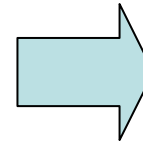
ULONG RTFCNDCL thE2( void *nContext ) {
    RtPrintf("thE2 is running.\n");
    RtWaitForSingleObject(ev, INFINITE);
    RtPrintf("thE2 Event is signaled.\n");
return 0;
}
```



MRTS – RTX6.5

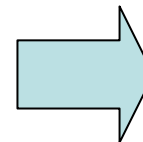


```
ev = RtCreateEvent(0, TRUE, FALSE, "EVENT.OBJECT");  
.  
.  
.  
RtSetEvent(ev);
```



```
thE1 priority was set to MIN + 1.  
thE2 priority was set to MIN + 2.  
thE1 is running.  
thE1 was resumed.  
thE2 is running.  
thE2 was resumed.  
Signaling event.  
thE2 Event is signaled.  
thE1 Event is signaled.
```

```
ev = RtCreateEvent(0, FALSE, FALSE, "EVENT.OBJECT");  
.  
.  
.  
RtSetEvent(ev);
```



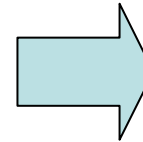
```
thE1 priority was set to MIN + 1.  
thE2 priority was set to MIN + 2.  
thE1 is running.  
thE1 was resumed.  
thE2 is running.  
thE2 was resumed.  
Signaling event.  
thE2 Event is signaled.
```



MRTS – RTX6.5

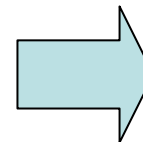


```
ev = RtCreateEvent(0, TRUE, FALSE, "EVENT.OBJECT");  
.  
.  
.  
RtPulseEvent(ev);
```



```
thE1 priority was set to MIN + 1.  
thE2 priority was set to MIN + 2.  
thE1 is running.  
thE1 was resumed.  
thE2 is running.  
thE2 was resumed.  
Signaling event.  
thE2 Event is signaled.  
thE1 Event is signaled.
```

```
ev = RtCreateEvent(0, FALSE, FALSE, "EVENT.OBJECT");  
.  
.  
.  
RtPulseEvent(ev);
```



```
thE1 priority was set to MIN + 1.  
thE2 priority was set to MIN + 2.  
thE1 is running.  
thE1 was resumed.  
thE2 is running.  
thE2 was resumed.  
Signaling event.  
thE2 Event is signaled.
```