
Sistemas Operativos 2

2021/22

Sincronização em Windows/Win32

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

1

Tópicos

Sincronização em Win32

Mecanismos e API

Bibliografia específica para este capítulo:

- WindowsNT 4 Programming; Herbert Schildt
- MSDN Library – PlatformSDK: DLLs, Processes, and Threads
<https://docs.microsoft.com/pt-pt/windows/win32/sync/synchronization-objects>
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms687069\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687069(v=vs.85).aspx)
- Windows System Programming (4th ed.)

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

2

Sincronização em Win32

A sincronização em Win32 segue os mesmos moldes gerais aplicáveis em qualquer sistema operativo. É acessível através do API Win32

Aplica-se a:

- Serialização de acções (coordenação / cooperação)
- Competição por acesso a um recurso
- Acesso exclusivo a um recurso (exclusão mútua / secções críticas)

A sincronização em Win32 é feita através de **espera** e **sinalização** em objectos (através dos seus *handles*)

- O *handle* de um objecto pode ser usado numa função **wait**. Se o objecto não estiver **assinalado**, a função bloqueia até que esse objecto seja assinalado.
- Os objectos podem estar apenas nos estados **assinalado** / **não-assinalado**

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

3

Sincronização em Win32

Objectos que podem ser usados em sincronização:

Exclusivamente para efeitos de sincronização:

- Evento (objectos-evento – não confundir com as mensagens WM_....)
- *Mutexes*
- *Critical Sections*
- Semáforos
- *Waitable Timers*

Outros objectos que podem ser usados em sincronização:

- Processos
- *Threads*
- Input de consola
- Notificações de alteração

Não esquecer: o estado sinalizado/não sinalizado **refere-se ao objecto** identificado pelos *handles* e não aos *handles* em si.

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

4

Sincronização em Win32

Objectos de Sincronização no API do Windows família NT

- **Semáforos**
Ferramenta de sincronização quase universal.
Apropriado a quase todos os casos.
Resolve situações de *exclusão mútua*, *competição* e *coordenação*
Em algumas situações não é o mecanismo mais simples ou mais directo
- **Mutexes**
Resolvem situações de *exclusão mútua* entre threads e entre processos.
Melhor mecanismo para situações de exclusão mútua que envolvam processos diferentes.
- **Critical Section**
Versão simplificada de mutexes para uso local a um único processo (com várias *threads*) em situações de *exclusão mútua*
O API destes objectos foge à forma habitual usada nos restantes

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

5

Sincronização em Win32

Objectos de Sincronização no API do Windows família NT

- **Waitable Timers**
Mecanismos apropriados a acontecimentos relacionados com a passagem de intervalos de tempo
- **Eventos**
Objecto de uso não-específico – o significado e forma de utilização depende muito da lógica da aplicação.
Não são apropriados a situações de exclusão mútua. São mais indicados para cenários de cooperação ou coordenação.

Existem mais objectos cujo API não é abordado aqui

One-time initialization – Inicializar um estrutura de dados uma única vez

Interlocked function – Permitem efectuar operações simples de fora atómica (e, ou, comparações, *test and set*) – a lógica destas funções é abordada nos fundamentos teóricos de sincronização

Outros mecanismos (*condition variables, etc.*)

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

6

Sincronização em Win32

Funções de espera

- **WaitForSingleObject / WaitForSingleObjectEx**
Espera que um determinado objecto esteja assinalado ou que um determinado *timeout* se esgote (ou que seja recebido uma notificação de *I/O completo*)
- **SignalObjectAndWait**
Assinala um objecto e espera noutro objecto
Aplicável a semáforos, mutexes e eventos
- **WaitForMultipleObjects / WaitForMultipleObjectEx**
Espera por um conjunto de objectos de uma só vez

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

7

Sincronização em Win32

WaitForSingleObject

```
DWORD WaitForSingleObject(  
    HANDLE hHandle,  
    DWORD dwMilliseconds  
);
```

- O objecto esperado pode ser qualquer um dos indicados atrás
- A função retorna:
 - **WAIT_ABANDONED**
A *thread* que detinha o objecto terminou sem o libertar
 - **WAIT_OBJECT_0**
O objecto de sincronização foi libertado
 - **WAIT_TIMEOUT**
Ocorreu o tempo máximo de espera

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

8

Sincronização em Win32

SignalObjectAndWait

```
DWORD SignalObjectAndWait(  
    HANDLE hObjectToSignal,  
    HANDLE hObjectToWaitOn,  
    DWORD dwMilliseconds,  
    BOOL bAlertable  
);
```

- Pode esperar em qualquer dos objectos de sincronização mencionados atrás
- Pode assinalar em: **semaphore**, **mutex**, **event**
- Retorna o mesmo que **WaitForSingleObjectEx**

Esta função é mais eficiente que *signal* + *wait* separados: evita uma mudança de processo desnecessária

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

9

Sincronização em Win32

SignalObjectAndWait – Cenário exemplo de aplicação

- Uma **worker thread** executa um trabalho. Quando o completa assinala trabalho completo e aguarda por mais trabalho (através de eventos)
- Uma outra **thread** (cliente desse trabalho) aguarda pelo trabalho da **worker thread**. Assim que o obtém atribui-lhe novo trabalho, assinalando-o

Código na **worker thread**

```
dwRet = SignalObjectAndWait(hEventWorkerDone,  
                             hEventMoreWorkToDo, INFINITE, FALSE);
```

Código na **thread cliente** (controla a **worker thread**)

```
dwRet = WaitForSingleObject(hEventWorkerDone, INFINITE);  
if( dwRet == WAIT_OBJECT_0)  
    SetEvent(hEventMoreWorkToDo);
```

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

10

Sincronização em Win32

WaitForMultipleObjectsEx

```
DWORD WaitForMultipleObjectsEx(  
    DWORD nCount,           // quantos objectos/handles  
    const HANDLE* lpHandles, // pont para o 1º handle  
    BOOL bWaitAll,           // esperar por todos?  
    DWORD dwMilliseconds,  
    BOOL bAlertable  
);
```

- Permite esperar em mais do que um objecto (dos mencionados atrás) em simultâneo e por notificações *I/O completo*
- Retorna quando um / todos os objectos especificados se encontram no estado assinalado:
 - **WAIT_OBJECT_0** (bWaitAll = TRUE) - todos os objectos foram assinalados
 - **WAIT_OBJECT_0** + índice do *handle* do objecto assinalado (bWaitAll = FALSE)
 - **WAIT_ABANDONED_0** ou **WAIT_ABANDONED_0** + *ind*
 - **WAIT_IO_COMPLETION**
 - **WAIT_TIMEOUT**

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

11

Sincronização em Win32

Obtenção dos *handles* para espera nas funções

Na *thread* que cria os objectos de sincronização

- *Handles* obtidos nas funções que criam os objectos
- Exemplos

- | | |
|--------------------------|------------------|
| - CreateEvent | (para eventos) |
| - CreateMutex | (para mutexes) |
| - CreateSemaphore | (para semáforos) |
- etc.

Em outras *threads*

- *Handles* obtidos por funções **open**
- Exemplos

- | | |
|------------------------|------------------|
| - OpenEvent | (para eventos) |
| - OpenMutex | (para mutexes) |
| - OpenSemaphore | (para semáforos) |
- etc.

Neste caso as flags de acesso especificadas são importantes

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

12

Sincronização em Win32

Exemplo de função *open* para o caso de um semáforo

```
HANDLE WINAPI OpenSemaphore(  
    DWORD dwDesiredAccess,    // descrevem as operações pretendidas  
    BOOL bInheritHandle,  
    LPCTSTR lpName  
);
```

→ Apenas as operações descritas na função *open* poderão posteriormente ser feitas no objecto (como habitualmente)

- Aplicam-se as *flags* habituais a todos os objectos windows. Exemplo:
 - SYNCHRONIZE
- E outras que variam conforme o tipo de objecto (exemplos no slide seguinte)

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

13

Sincronização em Win32

Flags de acesso nas funções *open*

- Semáforos: SEMAPHORE_MODIFY_STATE (SEMAPHORE_ALL_ACCESS)
- Eventos: EVENT_MODIFY_STATE (EVENT_ALL_ACCESS)
- Mutexes: MUTEX_MODIFY_STATE (MUTEX_ALL_ACCESS)
- Timers: TIMER_MODIFY_STATE (TIMER_ALL_ACCESS)

As *flags etc_ALL_ACCESS* permitem usar o objecto de sincronização de uma forma que ultrapassa a lógica de controlo do objecto

- O objecto pode perder as suas características usuais de sincronização
 - Por exemplo modificar o estado assinalado/não assinalado directamente sem ficar bloqueado
- Pode exigir que a aplicação tenha que correr em modo administrador
- Geralmente não se usa esta *flag*

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

14

Sincronização em Win32

Sincronização com objectos processo e threads

- A função **CreateProcess** cria um objecto que representa o novo processo.
- Inicialmente o objecto está no estado **não-assinalado**
- O estado passa a **assinalado** quando o processo termina
- Idem para **threads**

Funções mais úteis para este tipo de objecto de sincronização:

- **WaitForInputIdle**
Espera que o processo esteja bloqueado à espera de input
- **WaitForSingleObject** / **WaitForSingleObjectEx**
Espera que um processo esteja assinalado (ou seja, tenha terminado)

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

15

Sincronização em Win32

Sincronização com eventos

- Um objecto evento é um objecto de sincronização cujo estado pode ser assinalado explicitamente com a função **SetEvent**

Existem dois tipos de uso de evento:

- **Reset manual (“passam todos até alguém fechar”)**
O estado mantém-se como assinalado até que seja colocado como não assinalado explicitamente através da função **ResetEvent**
Enquanto está assinalado, qualquer *thread* que espere por ele (com as funções de espera vistas atrás) tem permissão para prosseguir
- **Auto-reset (“só passa o primeiro”)**
O evento é colocado no estado não-assinalado automaticamente pela primeira *thread* que espera (e obtém) o evento

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

16

Sincronização em Win32

CreateEvent

```
HANDLE CreateEvent(  
    LPSECURITY_ATTRIBUTES lpEventAttributes,  
    BOOL bManualReset,  
    BOOL bInitialState,  
    LPCTSTR lpName  
);
```

Os eventos podem ter nomes o que permite que outros processos os possam utilizar através da função **OpenEvent**

OpenEvent

```
HANDLE OpenEvent(  
    DWORD dwDesiredAccess,  
    BOOL bInheritHandle,  
    LPCTSTR lpName  
);
```

Obtém um handle para um evento com nome já existente

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

17

Sincronização em Win32

SetEvent

```
BOOL SetEvent(  
    HANDLE hEvent  
);
```

Coloca o evento especificado no estado **assinalado**

ResetEvent

```
BOOL ResetEvent(  
    HANDLE hEvent  
);
```

Coloca o evento especificado no estado **não-assinalado**

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

18

Sincronização em Win32

Sincronização com mutexes

- Um objecto **mutex** permite resolver directamente as situações de **exclusão mútua**
- Encontra-se no estado **assinalado** quando nenhuma *thread* o **possui**
- Assim que uma *thread* **obtem a sua posse** (através das funções de espera), o estado do mutex passa a **não-assinalado**
- Uma *thread* que tenha a posse de um mutex pode libertá-lo através da função **ReleaseMutex**. O mutex em questão passa ao estado **assinalado**
- São criados através da função **CreateMutex**
- Um mutex pode ter um nome e por conseguinte pode ser usado por processos diferentes (através da função **OpenMutex**)

Nota: existem objectos semelhantes (objectos *Critical Section*) mas que só podem ser usados no contexto do mesmo processo

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

19

Sincronização em Win32

CreateMutex

```
HANDLE CreateMutex(  
    LPSECURITY_ATTRIBUTES lpMutexAttributes,  
    BOOL bInitialOwner,  
    LPCTSTR lpName  
);
```

Cria ou (obtem acesso) a um mutex com ou sem nome

- Se **bInitialOwner** for TRUE, a thread que invoca a função obtém automaticamente a posse do mutex
- O nome pode ser NULL

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

20

Sincronização em Win32

OpenMutex

```
HANDLE OpenMutex(  
    DWORD dwDesiredAccess,  
    BOOL bInheritHandle,  
    LPCTSTR lpName  
);
```

Obtém acesso a um mutex

ReleaseMutex

```
BOOL ReleaseMutex(  
    HANDLE hMutex  
);
```

Liberta a posse do mutex

Apenas a *thread* que tem a posse do mutex deve chamar esta função

Nota: lembrar que a posse do **mutex** pode ser obtida através das funções espera já discutidas alguns slides atrás

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

21

Sincronização em Win32

Sincronização com *Critical Sections*

- Semelhante aos mutexes, mas não podem ser partilhados por processos diferentes. Servem apenas para **sincronização de threads dentro do mesmo processo**.
 - Em sistemas com **mais do que um processador**, pode-se indicar o número de vezes que uma *thread* tentará obter a posse do objecto (se este estiver não-assinalado) em ciclo fechado (espera activa) antes (da thread) passar ao estado bloqueado. Esta espera é designada por **Spinning** e o número de vezes que tenta é **Spin Count**. Esta característica pode aumentar a performance
- Criação do recurso critical section: declaração de uma variável do tipo `CRITICAL_SECTION` seguida da sua inicialização com **InitializeCriticalSection** ou **InitializeCriticalSectionAndSpinCount**
- Para obter a posse do objecto: **EnterCriticalSection** ou **TryEnterCriticalSection** (não bloqueante)
- Para libertar (assinalar) o objecto: **LeaveCriticalSection**
- Para destruir o recurso critical section: **DeleteCriticalSection**

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

22

Sincronização em Win32

Sincronização com semáforos

- Correspondem ao conceito geral de semáforos, em que é **mantida a contabilização** de operações **esperar/assinalar** que sobre ele são efectuadas (uso mais alargado que mutexes)
- São úteis na gestão de recursos com unidades limitadas
- São criados através da função **CreateSemaphore**
- Podem ter nome o que permite que sejam usados por processos diferentes (função **OpenSemaphore**)
- As funções de **espera** discutidas anteriormente efectuam a operação de espera sobre o semáforo, decrementando o seu contador interno. Se o contador atinge o valor zero, o estado do semáforo passa a **não-assinalado** e a *thread* que efectuou a espera fica bloqueada
- A função **ReleaseSemaphore** permite efectuar a operação **assinalar** num semáforo. Uma das *thread* que estavam bloqueadas à espera nesse semáforo é desbloqueada

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

23

Sincronização em Win32

CreateSemaphore

```
HANDLE CreateSemaphore(  
    LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,  
    LONG lInitialCount,  
    LONG lMaximumCount,  
    LPCTSTR lpName           // 0 nome permite usar o semáforo  
);                          // em processos diferentes
```

Cria (ou obtém acesso a) um semáforo com ou sem nome

OpenSemaphore

```
HANDLE OpenSemaphore(  
    DWORD dwDesiredAccess,  
    BOOL bInheritHandle,  
    LPCTSTR lpName  
);
```

Obtém um *handle* para um semáforo previamente existente

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

24

Sincronização em Win32

ReleaseSemaphore

```
BOOL ReleaseSemaphore(  
    HANDLE hSemaphore,  
    LONG lReleaseCount,  
    LPLONG lpPreviousCount  
);
```

Incrementa o contador interno do semáforo na quantidade especificada no parâmetro **lReleaseCount** (eventualmente levando a que uma outra *thread* desbloqueie)

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

25

Sincronização em Win32

Sincronização com waitable timers

- Correspondem a objectos de sincronização cuja passagem ao estado assinalado está associada à passagem de um intervalo de tempo
- Permitem implementar facilmente temporizadores
- São criados com a função **CreateWaitableTimer**
- Podem ter um nome associado o que permite a sua utilização por processos diferentes (através da função **OpenWaitableTimer**)

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

26

Sincronização em Win32

Sincronização com waitable timers

Existem três tipos de utilização:

- **Reset manual**
O timer permanece no estado assinalado até que seja invocada a função **SetWaitableTimer** para lhe dar um novo intervalo de tempo
- **Timer de sincronização**
Estes timers permanecem no estado assinalado até que uma *thread* complete uma operação de espera sobre ele
- **Timer periódico**
Estes timers são reactivados (passam a não-assinalados) assim que o intervalo de tempo anteriormente dado se esgota

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

27

Sincronização em Win32

CreateWaitableTimer

```
HANDLE CreateWaitableTimer(  
    LPSECURITY_ATTRIBUTES lpTimerAttributes,  
    BOOL bManualReset,  
    LPCTSTR lpTimerName  
);
```

Cria (ou obtém acesso a) um waitable timer

OpenWaitableTimer

```
HANDLE OpenWaitableTimer(  
    DWORD dwDesiredAccess,  
    BOOL bInheritHandle,  
    LPCTSTR lpTimerName  
);
```

Obtém um handle para um waitable timer previamente existente

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

28

Sincronização em Win32

SetWaitableTimer

```
BOOL SetWaitableTimer(  
    HANDLE hTimer,  
    const LARGE_INTEGER* pDueTime,  
    LONG lPeriod,  
    PTIMERAPCROUTINE pfnCompletionRoutine,  
    LPVOID lpArgToCompletionRoutine,  
    BOOL fResume // indica se permite sair do modo power preserve  
);
```

- Activa o waitable timer com o intervalo de tempo especificado. Após esse intervalo de tempo o estado do timer passa a **assinalado**
- **pDueTime** indica o intervalo de tempo após o qual o timer deverá passar ao estado assinalado em unidades de 100 nano-segundos (a precisão é limitada pelo hardware e pode ser inferior)
- Se o **lPeriod** for zero, o timer é **não-periódico** (um valor positivo indica que o timer é **periódico** e o seu período, em milisegundos)

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

29

Sincronização em Win32

SetWaitableTimer

- **pfnCompletionRoutine** indica uma função que é invocada automaticamente quando o timer é assinalado (pode ser NULL) cujo protótipo é:

```
VOID CALLBACK TimerAPCProc(  
    LPVOID lpArgToCompletionRoutine,  
    DWORD dwTimerLowValue,  
    DWORD dwTimerHighValue  
);
```

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

30

Sincronização em Win32

CancelWaitableTimer

```
BOOL CancelWaitableTimer(  
    HANDLE hTimer  
);
```

Coloca o waitable timer como **inactivo**

Nota: Esta função **não muda** o estado assinalado/não-assinalado do timer. Se existirem *threads* à espera do timer, estas *threads* não são automaticamente desbloqueadas

Obs.: a utilização desta função exige que o símbolo (macro) `_WIN32_WINNT` seja **0x0400** ou maior

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

31

Resumo dos mecanismos de sincronização

• Mutexes

- Permitem a duas ou mais *threads* (ou processos) aguardarem por acesso a uma zona de código ("seção crítica") que manipula um recurso (ex., dados) partilhados e cujo acesso simultâneo poria em causa a coerência desse recurso.
- Os *mutexes* podem ser criados com um nome, ficando acessíveis a mais do que um processo. Se forem criados sem um nome, o ficam restringidos a *threads* do mesmo processo.

DEIS/SEC

Sistemas Operativos 2 – 2021/22

João Durães

32

Resumo dos mecanismos de sincronização

- **Critical sections**

- Não confundir com o conceito de zona de código – “secção crítica”.
- São uma forma otimizada de *mutexes* que apenas servem para *threads* dentro do mesmo processo. Permitem a mesma funcionalidade dos *mutexes* com menos custo de performance, apesar dessa diferença de performance depender de diversos fatores e não ser linear.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

33

Resumo dos mecanismos de sincronização

- **Critical sections**

- Permitem a especificação de um valor de *spin count*. É o número de vezes que a *thread* tenta repetidamente e em espera ativa obter acesso, antes de desistir e ficar bloqueada caso a *critical section* esteja ocupada.

Espera ativa é, em teoria, indesejável, mas para valores de *spin count*, baixos, tem-se a possibilidade real de um ganho de performance pois é menos custoso tentar entrar “algumas vezes” repetidamente e conseguir entrar do que a *thread* ficar logo bloqueada e mais tarde ser reativada. Isto faz sentido quando se sabe à partida que o recurso que se pretende obter fica bloqueado por períodos de tempo muito curtos.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

34

Resumo dos mecanismos de sincronização

• Semáforos

- Generalização do conceito de *mutex* – Permite acesso a mais do que um *processo/thread* em simultâneo
- Usado para gerir o acesso a um recurso – não está necessariamente associado ao conceito de secção crítica (por poder permitir o acesso a mais do que um *processo/thread*).
- Normalmente usado para gerir acesso/uso a recursos finitos
- Os padrões de uso são diferentes dos *mutexes* e a semântica também é mais flexível: as operações de assinalar podem ser feitas por *threads* diferentes daquelas que tinham feito a operação de esperar.
- Podem ser criados com um nome, ficando acessíveis a mais do que um processo, sendo essa uma situação comum.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

35

Resumo dos mecanismos de sincronização

• Eventos

- Servem para uma *thread* indicar que “algo” aconteceu a uma ou mais *threads* que aguardavam por esse algo.
 - Exemplo de aplicação: aguardar que uma *thread* conclua uma tarefa qualquer mas continua em execução, não se podendo assim aguardar que essa *thread* termine.
 - Outro exemplo: indicar a várias *threads* que “podem começar” a fazer algo.
- Os eventos são bastante versáteis e permitem: deixar passar apenas uma das *threads* que aguardam ou todas. A evitar: confundir o cenário de aplicação com cenários de exclusão mútua, pois são completamente diferentes.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

36

Resumo dos mecanismos de sincronização

- **Waitable timers**

- Trata-se de um mecanismo semelhante aos eventos mas que permitem definir um período de tempo. Permitem implementar mecanismos de temporização (“despertadores”) de forma bastante simples.
- Permitem definir uma função que é chamada automaticamente no final do período definido (“*completion routine*”). A chamada a essa função é automática e assíncrona