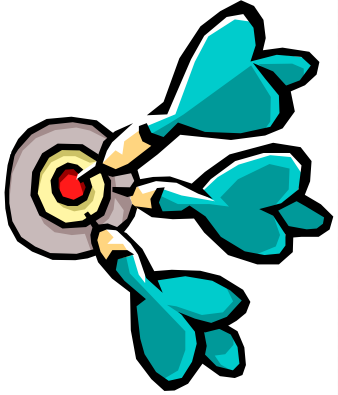
IPC in Windows

Programmazione di Sistema

A.A. 2017-18

© C. Barberis, G. Malnati 2004-18

Argomenti



- Sincronizzazione tra processi
  - Eventi
  - Semafori
  - Mutex
  - Meccanismi congiunti
- Comunicazione tra processi
  - MailSlot
  - Pipe
  - FileMapping
  - Altri meccanismi

© C. Barberis, G. Malnati 2004-18

Sincronizzazione e oggetti kernel

- La piattaforma win32 offre una ricca serie di meccanismi di sincronizzazione
  - Permettono di bloccare un thread fino a quando non si è verificato qualcosa in un altro thread
- Questi meccanismi si basano sull'uso di oggetti kernel condivisi
  - Il S.O. permette un accesso controllato al loro stato
  - Sono utilizzabili da thread appartenenti a processi differenti

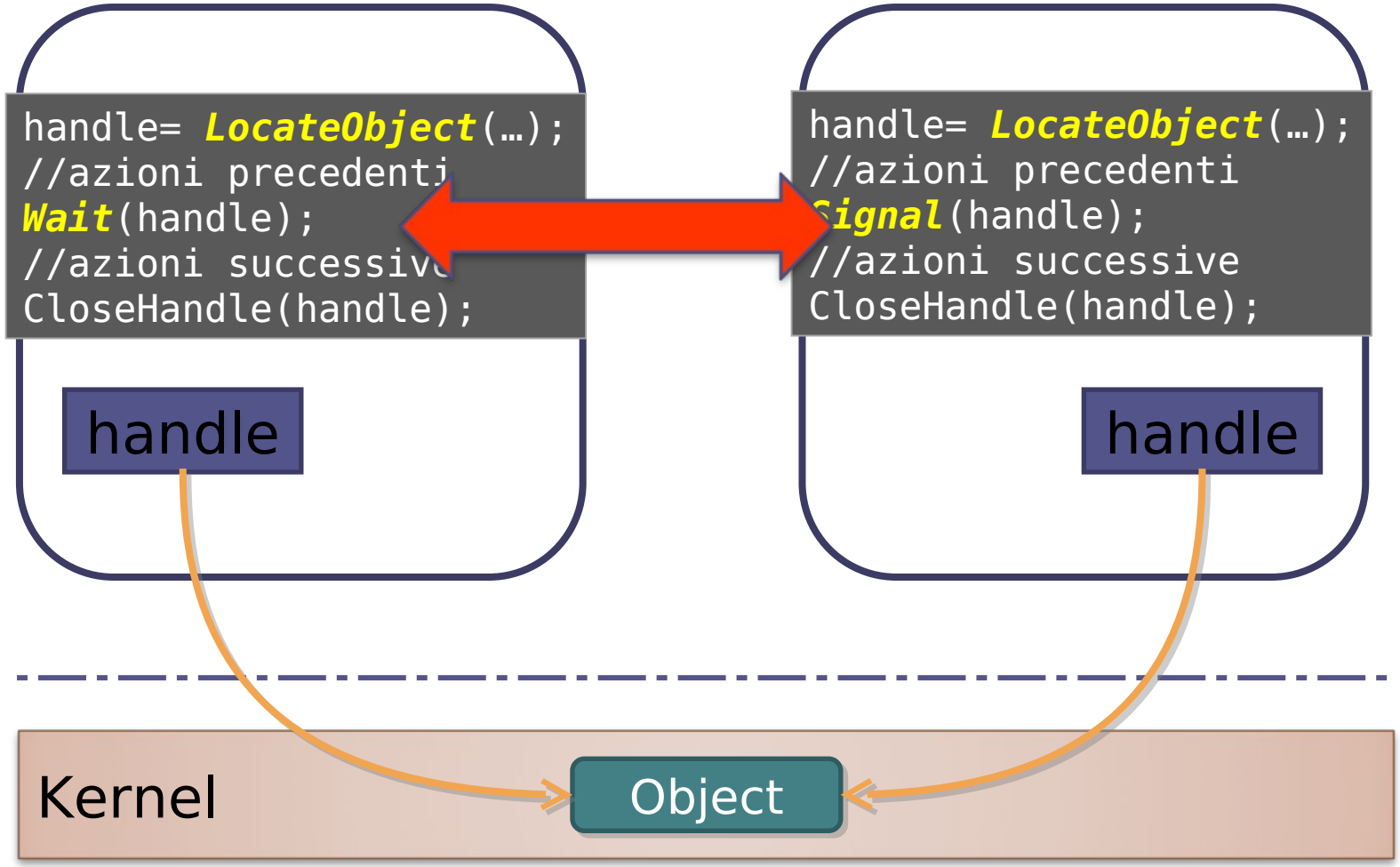
© C. Barberis, G. Malnati 2004-18

Sincronizzazione e oggetti kernel

- Rispetto ai costrutti di sincronizzazione usabili in un singolo processo, sono più generali ma meno efficienti
  - Tempi di sblocco maggiori, in quanto richiedono un passaggio alla modalità supervisore
- Tutte le tecniche si basano sull'uso delle funzioni di attesa offerte da Windows
  - WaitForSingleObject(…)
  - WaitForMultipleObjects(…)
- Ogni tipo di oggetto definisce le proprie politiche di attesa e risveglio
  - Permettendo la realizzazione di comportamenti diversi, adatti alle diverse esigenze

© C. Barberis, G. Malnati 2004-18

Meccanismo generale



© C. Barberis, G. Malnati 2004-18

Stato di Segnalazione

- La maggior parte degli oggetti kernel può trovarsi in due stati differenti
  - Segnalato
  - Non segnalato
- Nel caso di processi e thread
  - Lo stato non segnalato indica che l'elaborazione è ancora in corso
  - Una volta raggiunto lo stato segnalato, non è possibile tornare indietro
- Gli oggetti di sincronizzazione (eventi, semafori, mutex) possono alternare i due stati di segnalazione
  - In funzione delle proprie politiche e delle richieste eseguite nel programma

© C. Barberis, G. Malnati 2004-18

Eventi

- Oggetti kernel che modellano il verificarsi di una condizione
  - Segnalata esplicitamente dal programmatore tramite opportuni metodi
- Gli eventi di tipo manual-reset permettono ad un numero indefinito di thread in attesa di svegliarsi
  - Quelli auto-reset tornano automaticamente allo stato non segnalato se causano lo sblocco di un thread

© C. Barberis, G. Malnati 2004-18

Eventi

- Ci si collega ad un evento tramite
  - CreateEvent(...)
  - OpenEvent(...)
- Due processi possono sincronizzarsi condividendo un evento (tramite il nome)

© C. Barberis, G. Malnati 2004-18

Ciclo di Vita degli Eventi

- Si modifica lo stato di segnalazione di un evento attraverso le primitive
  - SetEvent(…)
  - ResetEvent(…)
  - PulseEvent(…)

	AUTO_RESET	MANUAL_RESET
SetEvent	Un solo thread tra quelli in attesa viene rilasciato. Se nessuno è in attesa, il prossimo che effettuerà un attesa sull'evento sarà rilasciato	Tutti i thread in attesa vengono rilasciati. L'evento resta segnalato fino ad una chiamata a ResetEvent
PulseEvent	Un solo thread tra quelli in attesa viene rilasciato. Se nessuno è in attesa, non capita nulla	Tutti i thread in attesa vengono rilasciati. L'evento viene posto nello stato non segnalato

© C. Barberis, G. Malnati 2004-18

Semafori

- Mantengono al proprio interno un contatore
  - Stato segnalato se il contatore è > O
  - Stato non segnalato se il contatore è = O
  - Non può valere mai meno di zero
- WaitForSingleObject decrementa il contatore se >O
  - Altrimenti blocca il thread in attesa che un altro incrementi il contatore
- Il contatore è incrementato quando un thread invoca ReleaseSemaphore(...)

© C. Barberis, G. Malnati 2004-18

Utilizzo dei semafori

- Usati tipicamente quando si hanno più copie di una risorsa disponibili
- CreateSemaphore(…)
  - Crea un semaforo
- OpenSemaphore(…)
  - Recupera un handle di un semaforo precedentemente creato
- WaitForSingleObject / WaitForMultipleObjects
  - Decrementa il contatore o si accoda in attesa che venga incrementato
- ReleaseSemaphore
  - Incrementa il contatore

© C. Barberis, G. Malnati 2004-18

Mutex

- Assicurano a più trhead (anche di processi differenti) l'accesso in mutua esclusione ad una data risorsa
  - Può essere specificato un tempo massimo di attesa
- Conservano l'ID del thread che li ha acquisiti ed un contatore
  - Se ID = O, la risorsa non è stata acquisita ed il mutex è nello stato segnalato
- Il contatore mantiene il numero di volte che il thread ha acquisito la risorsa
  - Se il thread che possiede il mutex termina, il mutex viene resettato

© C. Barberis, G. Malnati 2004-18

Ciclo di vita di un mutex

- CreateMutex, OpenMutex
  - Crea (od ottiene) l'handle ad un mutex
- ReleaseMutex
  - Verifica che il thread corrente sia il possessore
  - Decrementa il contatore
  - Se arriva a O segnala il mutex
- WaitForSingleObject / WaitForMultipleObjects
  - Acquisisce un mutex se è nello stato segnalato o se è già in possesso del thread, e ne incrementa il contatore
  - Altrimenti attende che il mutex diventi segnalato

© C. Barberis, G. Malnati 2004-18

Uso di più primitive

- Due o più oggetti kernel possono essere usati in modo congiunto per implementare particolari politiche di sincronizzazione
  - Nel caso di memoria condivisa, si può usare un Mutex per regolare l'accesso ed uno o più eventi per indicare il completamento di azioni

© C. Barberis, G. Malnati 2004-18

```
hE=CreateEvent("done"
,...);
hMut=CreateMutex("m"
);
hFM=CreateFileMapping(
...);
WaitForSingleObject(h
Mut);

ptr=MapViewOfFile(hFM,
...);
 //write to shared
memory
 SetEvent(hE);
 UnmapViewOfFile(hFM);
ReleaseMutex(hMut);
CloseHandle(...);
```

```
hE=CreateEvent("done"
,...);
hMut=CreateMutex("m"
);
hFM=CreateFileMapping(
...);
WaitForMultipleObject
s(
   [hE, hMult] ,WAIT_ALL,
...);
 ptr=MapViewOfFile(hFM,
...);
 //read  shared memory
 UnmapViewOfFile(hFM);
ReleaseMutex(hMut);
CloseHandle(...);
```

© C. Barberis, G. Malnati 2004-18

Mailslot

- Coda di messaggi asincrona per la comunicazione tra processi
  - Il mittente può essere un processo della macchina stessa…
  - …oppure di un elaboratore appartenente allo stesso dominio di rete
- Un processo può essere contemporaneamente sia un mailslot client che un mailslot server
  - Ciò permette di realizzare canali di comunicazione bidirezionali
- È possibile inviare messaggi broadcast a tutte le mailslot con lo stesso nome nello stesso dominio di rete

© C. Barberis, G. Malnati 2004-18

Mailslot server

- Si crea una mailslot
  - Associandole un nome univoco

```
HANDLE hSlot;
LPTSTR SlotName = TEXT("\\\\.\\mailslot\\ms1");

hSlot = CreateMailslot(SlotName,
             0, // no maximum message size
             MAILSLOT_WAIT_FOREVER, // no time-out
             (LPSECURITY_ATTRIBUTES)NULL); //
default                                    //
security
```

© C. Barberis, G. Malnati 2004-18

Mailslot server

- I messaggi vengono letti come normali file
  - Con le API ReadFile(…) e ReadFileEx(…)
- Il messaggio viene conservato finché non viene letto
- GetMailslotInfo(…)
  - Restituisce il numero di messaggi accodati e la dimensione del primo da leggere

© C. Barberis, G. Malnati 2004-18

Mailslot server

```
DWORD cbMessage, cMessage, cbRead;
LPTSTR lpszBuffer;

BOOL fResult = GetMailslotInfo(hSlot,
(LPDWORD)NULL, &cbMessage,
       &cMessage,(LPDWORD)NULL);

if (fResult && cbMessage!= MAILSLOT_NO_MESSAGE) {

  fResult = ReadFile(hSlot,lpszBuffer,cbMessage,
                     &cbRead,NULL);
               //…
             }
```

© C. Barberis, G. Malnati 2004-18

Mailslot Client

- Accoda i messaggi ad una mailslot
  - CreateFile(...) apre la mailslot
  - WriteFile(...) e WriteFileEx(...) scrivono atomicamente un messaggio

© C. Barberis, G. Malnati 2004-18

Mailslot Client

```
LPTSTR Slot = TEXT("\\\\.\\mailslot\\ms1");
HANDLE hSlot = CreateFile(Slot, GENERIC_WRITE,
                  FILE_SHARE_READ,
                  (LPSECURITY_ATTRIBUTES)NULL,
                  OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
                  (HANDLE)NULL);
//…
LPTSTR lpszMessage = TEXT("Message one");

BOOL fResult = WriteFile(hSlot, lpszMessage,
   (DWORD)(lstrlen(lpszMessage) +
1)*sizeof(TCHAR),
    &cbWritten,(LPOVERLAPPED)NULL);
```

© C. Barberis, G. Malnati 2004-18

Rilascio delle risorse

- CloseHandle(…)
  - Chiude la mailslot e rilascia le relative risorse nel momento in cui tutti i suoi handle siano stati chiusi

© C. Barberis, G. Malnati 2004-18

Pipe

- Possono essere di tipo
  - Anonimo
  - Dotato di nome (named pipe)

© C. Barberis, G. Malnati 2004-18

Anonymous pipe

- Mezzo efficiente di comunicazione tra 2 processi “parenti”
  - Ridirezione dello standard input e/o dello standard output tra processo padre e figlio
  - Sono solo monodirezionali
- Un processo padre, dopo aver creato una pipe, può passare uno dei due handle al figlio

© C. Barberis, G. Malnati 2004-18

Anonymous Pipe

- Una handle può anche essere duplicata nel processo con cui si intende comunicare
  - Tramite DuplicateHandle
- La sua identità può anche essere passata attraverso un segmento di memoria condivisa
  - Facendoglielo ereditare all'atto della CreateProcess
- ReadFile e WriteFile
  - Servono a leggere e scrivere dalla pipe
  - Le pipe anonime non supportano l'input-output asincrono

© C. Barberis, G. Malnati 2004-18

Named Pipe

- Permettono comunicazioni tra due processi qualsiasi
  - Che risiedono sia sulla stessa macchina che su macchine differenti
- Possono essere create bidirezionali
- Il nome di una pipe è così specificato
  - \\ServerName\pipe\PipeName
  - PipeName è case-insensitive e deve essere unico all'interno del S.O.
  - "\\.\"Indica il calcolatore locale

© C. Barberis, G. Malnati 2004-18

Named Pipe

- Creazione/apertura
  - CreateNamedPipe(…), OpenFile(…), CallNamedPipe(…)
- Si opera sulle named pipe come sui file
  - ReadFile(…), WriteFile(…), ecc.

© C. Barberis, G. Malnati 2004-18

Named pipe server

```
TSTR lpszPipename = TEXT("\\\\.\\pipe\\
mynamedpipe");

hPipe = CreateNamedPipe( lpszPipename
          PIPE_ACCESS_DUPLEX,
          PIPE_TYPE_MESSAGE |
          PIPE_READMODE_MESSAGE | PIPE_WAIT,
          PIPE_UNLIMITED_INSTANCES,
          BUFSIZE, BUFSIZE,
          0, NULL);
//...
BOOL fSuccess = ReadFile(
         hPipe,        // handle to pipe
         pchRequest,    // buffer to receive data
         BUFSIZE*sizeof(TCHAR), // size of buffer
         &cbBytesRead, // number of bytes read
         NULL);        // not overlapped I/O
```

© C. Barberis, G. Malnati 2004-18

Named pipe client

```
TSTR lpszPipename = TEXT("\\\\.\\pipe\\
mynamedpipe");

HANDLE hPipe = CreateFile(
         lpszPipename, GENERIC_READ |
GENERIC_WRITE,
         0, NULL, OPEN_EXISTING, 0 NULL);
//...
LPTSTR lpvMessage=TEXT("Default message from
client.");
BOOL fSuccess = WriteFile(
      hPipe,                    // pipe handle
      lpvMessage,               // message
      cbToWrite,                // message length
      &cbWritten,               // bytes written
      NULL);                    // not overlapped
```

© C. Barberis, G. Malnati 2004-18

Pipe

- Modalità di lettura
  - All'atto della creazione si può specificare la modalità di lettura
    - Stream di byte o a messaggio
- Modalità di attesa
  - Determina il comportamento delle operazioni di lettura, scrittura e di connessione
    - ReadFile, WriteFile, ConnectNamedPipe
  - In modalità bloccante
    - La funzione attende per un tempo indefinito che il processo con cui si vuole comunicare termini le operazioni sulla pipe
  - In modalità non bloccante
    - La funzione ritorna immediatamente nelle situazioni che richiederebbero un'attesa indefinita

© C. Barberis, G. Malnati 2004-18

File Mapping

- Meccanismo adatto per condividere ampie zone di memoria e a realizzare aree condivise persistenti
- Permette di accedere al contenuto di un file come se fosse un blocco di memoria
  - Allocato nello spazio di indirizzamento del processo
  - Scritture in tale blocco comportano la modifica del file
- Occorre sincronizzare accessi concorrenti allo stesso file-mapping
- È un meccanismo efficiente per condividere dati tra più processi sullo stesso computer

© C. Barberis, G. Malnati 2004-18

Ciclo di vita di un File Mapping

- CreateFileMapping(...)
  - Crea un file mapping
  - Se esiste già un file mapping con il nome passato, ne ritorna l'handle
  - Consente di specificare i diritti di accesso al file
- MapViewOfFile
  - Crea una vista del file mapping nello spazio di indirizzamento del processo
  - Ritorna un puntatore che può essere direttamente usato per accedere alla area di memoria condivisa
- UnmapViewOfFile
  - Rilascia la vista sul file mapping
- CloseHandle
  - Chiude la handle al file mapping

© C. Barberis, G. Malnati 2004-18

Altri meccanismi

- Socket
  - Permettono la comunicazione tra macchine dotate di sistemi operativi differenti
  - Supportano direttamente solo il trasferimento di array di byte
- RPC – Remote Procedure Call
  - Tecnica basata sui socket
  - Fornisce un meccanismo di alto livello per il trasferimento di dati strutturati

© C. Barberis, G. Malnati 2004-18