Universita` degli Studi di Bari

C.d.L triennale in Informatica, A.A. 2001/02

Corso di Sistemi Operativi, II semestre

Prof. Impedovo Sebastiano

17/05/2002

OGGETTI KERNEL

Marcello Barnaba <vjt@openssl.it>

Danilo Cassone <cassoned@libero.it>

:: SOMMARIO SEZIONE I ::

PRESENTAZIONE DEGLI OGGETTI KERNEL

Descrizione degli oggetti in Windows

Motivi dell`esistenza degli oggetti kernel

Creazione di un oggetto kernel e conteggio di utilizzo

Protezione degli oggetti kernel

Handle e tabelle di handle

Modifica / Lettura delle proprieta` di un handle

Chiusura di un oggetto kernel

:: SOMMARIO SEZIONE II ::

CONDIVISIONE DI OGGETTI KERNEL

Perche` condividere un oggetto kernel

Metodi di condivisione:

- Ereditarieta` degli handle
- Oggetti denominati (con accenno a NT Terminal Server)
- Duplicazione di handle

:: SEZIONE I ::

descrizione degli oggetti kernel

Gli Oggetti in Windows NT

- Windows NT si basa sui concetti della programmazione ad oggetti.
- Diverse risorse, servizi ed entità all'interno di Windows NT sono rappresentate da oggetti.
- L'**executive** di NT implementa due tipi di oggetti:
  - gli **oggetti kernel**, gestiti dal kernel
  - gli **oggetti executive**

Definizione degli oggetti kernel

- Gli **oggetti kernel** esistono in quanto blocchi di memoria allocati dal kernel in kernel space, memoria cui il processo chiamante non ha accesso
- Per manipolare questi oggetti e` necessario usare delle funzioni apposite per ogni oggetto
- Sono costituiti da un struttura dati con informazioni sull'oggetto: alcuni membri sono comuni a tutti gli oggetti (nome, parametri sulla sicurezza e conteggio di utilizzo), altri sono specifici del tipo di oggetto
- Esempi di oggetti kernel sono FileMapping, NamedPipes, MailSlots, Mutexes, Semaphores, Threads, etc.

Perche` esistono gli oggetti kernel

- Per identificare e poter creare risorse di basso livello del sistema (processi, mutex, semafori)
- Per fornire primitive di controllo e sincronizzazione interni al kernel: non c`e` rischio che un context switch possa vanificare i nostri sforzi di sincronizzazione, in quanto il kernel stesso e` a conoscenza dello stato particolare in cui si trovano i processi
- Per proteggere le stesse risorse che vengono create, in quanto questi oggetti risiedono in kernel space e nessun processo utente puo` alterare il contenuto di questa memoria
- Per fornire un interfaccia strutturata ed astratta a diverse operazioni di sistema

Creazione di un Oggetto Kernel

- Windows mette a disposizione una funzione **`Create*()`** per ogni tipo di oggetto che e` possibile creare
- Ad esempio per creare un oggetto semaforo si utilizza la funzione
  -
```
HANDLE CreateSemaphore(
  PSECURITY_ATTRIBUTES pSA,
  LONG lInitialCount,
  LONG lMaximumCount,
  PCTSTR pszName);
```

La quale accetta come parametri un descrittore di protezione, il valore iniziale del semaforo, il valore massimo che puo` assumere ed una stringa che rappresenta il “nome” che il semaforo stesso possiede in modo da poter venire identificato

Conteggio di utilizzo

- Ogni oggetto kernel contiene un **contatore di utilizzo**: quando viene creato questo contatore viene posto ad **1**, incrementato per ogni processo che ottiene accesso all`oggetto e decrementato per ogni processo che rilascia l`accesso
- Quando il contatore raggiunge il valore **0**, l`oggetto viene distrutto dal kernel.
- Un oggetto kernel puo` quindi sopravvivere al processo che lo ha creato
- I processi non hanno quindi alcun potere di distruggere l`oggetto

Protezione degli oggetti kernel

- E' possibile proteggere un oggetto kernel mediante un descrittore di protezione.
- Quasi tutte le chiamate di creazione hanno come parametro opzionale un puntatore ad una struttura dati di tipo **`SECURITY_ATTRIBUTES`** mediante la quale è possibile definire i diritti di accesso all'oggetto.
- Non definendo le impostazioni di protezione l`oggetto acquisisce le impostazioni predefinite: Accesso completo al creatore dell`oggetto e ai membri del gruppo Administrators, Nessun Accesso a tutti gli altri utenti/gruppi.
- Il meccanismo di protezione non è implementato in Windows 95/98 ma solo in NT.

Tabella di Handle

- Quando un processo viene inizializzato, ad esso viene anche assegnata una **tabella di handle**
- La tabella e` un array di strutture dati, ciascuna delle quali include:
  - un puntatore al blocco di memoria dove si trova l'oggetto associato all`handle (`LPVOID`)
  - una bitmask di accesso (`DWORD`)
  - una bitmask di flag (`DWORD`)
- Il termine **handle**, strettamente parlando, si riferisce esclusivamente all'indice della tabella

Handle

- Un **handle** e` un oggetto generico ed astratto con cui si manipolano diversi tipi di risorse in Windows
- Alla creazione di un oggetto kernel, la funzione addetta ritorna un **handle** all`oggetto, il quale non e` altro che un indice della **tabella di handle** del processo chiamante
- Il significato stesso dell`**handle** non e` documentato e soggetto a modifiche. Per questo motivo per utilizzare gli handle e` necessario utilizzare le WinAPI
- Ogni **handle** e` specifico di ogni processo, quindi la trasmissione di questo valore ad un altro processo **non** e` un`operazione corretta
- Per chiudere un **handle** si utilizza la funzione `CloseHandle()`

Modifica delle proprieta` di un handle

- Windows mette a disposizione due funzioni per leggere e scrivere le flag di un handle:
  - `BOOL GetHandleInformation(HANDLE hObj, PDWORD pdwFlags);`
  - `BOOL SetHandleInformation(HANDLE hObj, DWORD dwMask, DWORD dwFlags);`
- Attualmente gli handle possono assumere solo due flag:
  - `HANDLE_FLAG_INHERIT`
  - `HANDLE_FLAG_PROTECT_FROM_CLOSE`
- Funzione `SetHandleInformation()`:
  - Il primo parametro punta ad un handle valido
  - Il secondo indica quali flag si desidera modificare
  - Il terzo indica il valore da impostare nei flag che modifichiamo

Lettura delle proprieta` di un handle

- Funzione **`GetHandleInformation()`:**
  - Il primo parametro e` un puntatore ad un handle valido
  - Il secondo e` un puntatore ad un valore **`DWORD`** che accogliera` le flag dell`handle in esame
- Il flag **`HANDLE_FLAG_INHERIT`** marca l`handle come ereditabile
- Il flag **`HANDLE_FLAG_PROTECT_FROM_CLOSE`** indica al sistema di generare un eccezione se il programma utente cerca di effettuare la **`CloseHandle()`** sull`handle in questione

Chiusura di un Oggetto Kernel

- Per il conteggio di utilizzo, non sempre un oggetto kernel viene eliminato quando un processo che aveva accesso all`oggetto stesso lo rilascia
- E` buona norma chiudere gli handle aperti prima di uscire o quando non sono piu` necessari
- Non chiudere un handle non e` comunque una condizione critica in quanto l`OS garantisce che la liberazione delle risorse allocate da un processo al momento della sua terminazione

:: SEZIONE II ::

condivisione degli oggetti kernel

Perche` condividere gli oggetti kernel

- Condivisione di file (oggetti **`FileMapping`** e **`File`**)
- Trasmissione di dati tra processi in esecuzione su computer diversi (oggetti **`MailSlot`** e **`NamedPipe`**)
- Sincronizzazione di eventi e processi (oggetti **`Semaphore`** e **`Mutex`**)
- Gestione di processi concorrenti da parte di un processo padre (oggetto **`Process`**)

Metodi di condivisione degli oggetti kernel

- Ereditarieta` degli handle
- Oggetti denominati
- Duplicazione di handle

Ereditarieta` degli handle:
definizione

- Un handle puo` essere marcato come ereditable se al momento della sua creazione il parametro **`bInheritHandle`** della struttura **`SECURITY_ATTRIBUTES`** e` settato a **`TRUE`**.
- L`handle immesso nella tabella degli handle del processo che ha creato l`oggetto ha il bit di flag che identifica l`ereditabilita` settato ad 1
- Quando il processo che ha creato l`oggetto crea un nuovo processo con la **`CreateProcess()`**, essa analizza la tabella degli handle e copia nella tabella processo figlio qualsiasi handle ereditabile presente nella tabella del padre conservando i numeri di indice.
- Gli handle ereditabili vengono di fatto ereditati solo al momento della **`CreateProcess()`**

Ereditarieta` degli handle (2): particolarita`

- Un processo che ha ereditato degli handle non e` consapevole di questo evento
- I valori degli handle sono assegnati dal sistema e non c`e` modo di richiedere un handle con un indice specifico
- Il processo padre deve quindi notificare al figlio l`indice della tabella in cui si trova(no) l`(gli) handle ereditato(i)
- Possibili metodi per la trasmissione del valore di handle:
  - Tramite riga di comando (`GetCommandLine()`);
  - Tramite variabili di ambiente (`GetEnvironmentVariable()`);
  - Utilizzando primitive di IPC, a patto che il processo padre utilizzi `WaitForInputIdle()` per attendere il completamento dell`inizializzazione del figlio

Oggetti denominati:
definizione

- Quasi ogni funzione per la creazione di oggetti kernel possiede un parametro di tipo **`PCTSTR`**, ossia una stringa di caratteri terminata da uno **`'\0'`**
- Questa stringa e` usata per denominare in maniera univoca un oggetto kernel
- Un processo che voglia creare un mutex denominato potrebbe eseguire:
  - **`HANDLE hMutex = CreateMutex(NULL, FALSE, "myMutex");`**
- E` possibile quindi per un altro processo accedere al mutex, a patto che possegga i diritti di accesso, utilizzando la **`CreateMutex()`** con lo stesso nome oppure utilizzando la **`OpenMutex()`**.

Oggetti denominati (2):
`create*()` e `open*()`

- Non possono esistere due oggetti di tipo diverso con nome uguale
- Differenze nell`apertura di oggetti denominati tra `Create*()` e `Open*()`:

`Evento`	`Create*()`	`Open*()`
"Apertura" di un oggetto inesistente	Crea l`oggetto e ritorna l`handle	Ritorna **`NULL`** e setta **`ERROR_FILE_NOT_FOUND`**
Apertura di un oggetto esistente cui si ha accesso	Ritorna l`handle all`oggetto e setta **`ERROR_ALREADY_EXISTS`**	Ritorna l`handle all`oggetto
Apertura di un oggetto cui non si ha accesso	Ritorna **`NULL`** e setta **`ERROR_ACCESS_DENIED`**	Ritorna **`NULL`** e setta **`ERROR_ACCESS_DENIED`**

Oggetti denominati (3):
consigli sulla nominazione degli oggetti condivisi

- Se si utilizzano le **`Open*()`** per aprire oggetti esistenti non e` possibile passare **`NULL`** come parametro
- Per evitare conflitti di nomi nella creazione di oggetti, e` consigliabile generare un **`CLSID`** univoco dell`applicazione ed utilizzare la rappresentazione in stringa del **`CLSID`** per rappresentare gli oggetti
- Utilizzando oggetti denominati con un **`CLSID`** e` possibile ad esempio impedire multiple istanze di un`applicazione, semplicemente creando un nuovo oggetto e verificando il valore che ritorna **`GetLastError()`**

Oggetti denominati (4):
Spazi nome di NT Terminal Server

- Su Windows NT Terminal Server gli oggetti kernel non condividono i namespace (spazi nome) come indicato precedentemente: ogni sessione utente ha un suo spazio di nomi privato
- Per forzare un oggetto a rientrare nel namespace globale bisogna anteporre al suo nome il prefisso "**`Global\`**", per forzarlo a rientrare nel namespace locale bisogna anteporre al nome il prefisso "**`Local\`**"
- "**`Local`**", "**`Global`**" e "**`Session`**" sono case-sensitive, sono considerate keyword riservate da Microsoft e sono ignorate se sulla macchina non e` in esecuzione Terminal Server

Duplicazione di handle:
definizione della `DuplicateHandle()`

- Il modo piu` flessibile per condividere oggetti kernel e` la duplicazione degli handle, effettuata mediante la seguente primitiva:

- ```
  BOOL DuplicateHandle(
      HANDLE hSourceProcessHandle,
      HANDLE hSourceHandle,
      HANDLE hTargetProcessHandle,
      PHANDLE phTargetHandle,
      DWORD dwDesiredAccess,
      BOOL bInheritHandle,
      DWORD dwOptions);
  ```

- Ritorna TRUE in caso di successo, FALSE in caso di insuccesso
  ```

Duplicazione di handle (2):
lista parametri

- La primitiva effettua una copia tra due processi di un handle valido ed aperto nel momento in cui viene chiamata
- Il **primo** parametro e` un handle al processo dalla cui tabella l`OS prendera` l`handle richiesto
- L`handle richiesto e` all`indice specificato dal **secondo** parametro
- Il **terzo** parametro e` un handle al processo di destinazione
- Il **quarto** parametro e` un puntatore ad un tipo handle che verra` riempito dalla **`DuplicateHandle()`** in modo da contenere un handle valido per il processo di destinazione: poiche` questo non riceve alcuna notifica della duplicazione, e` compito del processo che effettua la **`DuplicateHandle()`** notificare mediante IPC i processi destinatari della duplicazione

Duplicazione di handle (3):
lista parametri

- Il **quinto** parametro identifica il valore della bitmask di accesso dell`handle che sara` posto nella tabella del processo destinatario
- Il **sesto** parametro identifica il flag di ereditarieta` che la **`DuplicateHandle()`** dovra` o non dovra` apporre nella nuova riga della tabella del processo destinatario
- Il **settimo** parametro puo` essere zero oppure una combinazione delle due seguenti opzioni:
  - **`DUPLICATE_SAME_ACCESS`** per replicare le informazioni di accesso dall`handle originario all`handle di destinazione, ignorandp il valore di **`dwDesiredAccess`**
  - **`DUPLICATE_CLOSE_SOURCE`** per chiudere l`handle di origine dopo aver copiato l`handle di destinazione

Duplicazione di handle (4):
note sui processi coinvolti nella duplicazione

- Per trasmettere un handle ad un altro processo bisogna avere i diritti di apertura ed utilizzare la **`OpenProcess()`** passando il PID del processo di destinazione
- Dopo la trasmissione di un handle ad un altro processo aperto e` necessario notificare il processo destinazione dell`avvenuta duplicazione mediante IPC
- Sebbene sia possibile trasmettere handle coinvolgendo tre processi **`S`**, **`X`** e **`Y`**, in cui **`X`** possiede l`handle da copiare, **`Y`** riceve l`handle ed **`S`** chiama la **`DuplicateHandle()`**, piu` in generale il processo che effettua la copia e` lo stesso che chiama la **`DuplicateHandle()`**, utilizzando come pseudo-handle sorgente il valore ritornato dalla **`GetCurrentProcess()`**.

Duplicazione di handle (5):
note sul quarto parametro e sul conteggio di utilizzo

- Quando un handle viene copiato da un processo ad un altro, l`handle ritornato dalla **`DuplicateHandle()`** nel quarto parametro e` un handle valido per il processo di destinazione ma non per il chiamante: chiamare la **`CloseHandle()`** su questo handle puo` produrre effetti inaspettati in quanto non c`e` nessun modo per prevedere se l`handle del processo di destinazione sia valido per il chiamante, e comunque se ci sia necessita` di chiuderlo o meno
- Ogni chiamata con successo alla **`DuplicateHandle()`** che non contenga il flag **`DUPLICATE_CLOSE_SOURCE`** incrementa il conteggio di utilizzo dell`oggetto in causa di uno

Duplicazione di handle (6):
duplicazione all`interno dello stesso processo

- Un handle puo` essere duplicato anche all`interno dello stesso processo
- Ogni handle e` identificato da una bitmask di accesso: se un processo volesse avere due accessi differenti allo stesso oggetto puo` chiamare la **`DuplicateHandle()`** con una bitmask differente per ottenere un nuovo handle con accesso diverso allo stesso oggetto
- In questo caso gli handle del processo sorgente e di destinazione saranno entrambi degli pseudo-handle ritornati da **`GetCurrentProcess()`**