Sistemas Operativos 2

2021/22

Comunicação entre processos com

Ficheiros mapeados em memória / Memória partilhada

1

Tópicos

Ficheiros mapeados em memória

Memória partilhada

Bibliografia específica para este capítulo:

- MSDN Library – PlattformSDK: DLLs, Processes, and Threads
  https://docs.microsoft.com/en-us/windows/win32/memory/creating-named-shared-memory
- Windows System Programming (4th ed.)

2

Ficheiros mapeados em memória

Ficheiros mapeados:

- Trata-se da implementação em Windows de um mecanismo habitual em vários sistemas operativos: mapeamento de ficheiros em memória

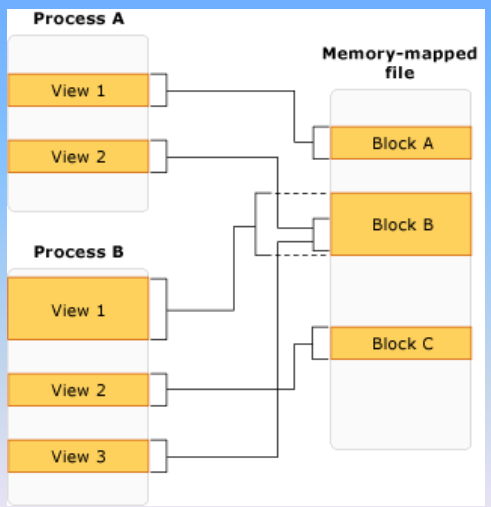
No Windows:

O mecanismo de ficheiros mapeados permite duas funcionalidades:

- Mapear (parte de) um ficheiro em memória, operando sobre ele como se se tratasse de uma matriz de bytes
- Partilhar um bloco de memória entre processos para situações de comunicação entre processos

Estas duas funcionalidades podem ser usadas separadamente, ou em conjunto (mapear um ficheiro em memória e partilhá-lo entre vários processos), dependendo do que o programador pretende.

Ficheiros mapeados em memória



Memória partilhada

- Conseguido em WindowsNT à custa do recurso ***ficheiro mapeado*** (em memória)
- Corresponde a uma zona de memória.
- Essa zona será depois mapeada no espaço de endereçamento dos processos interessados em a usar através de uma ou mais ***vistas***
- Uma vista pode corresponder à totalidade ou apenas a parte da zona total de memória

Ficheiros mapeados em memória

Restrições ao uso de ficheiros mapeados:

Uso concorrente com o API de I/O habitual

- O acesso a um ficheiro mapeado em memória não é garantidamente coerente com o uso em simultâneo do mesmo ficheiro através da API de I/O de ficheiros (*ReadFile* / *WriteFile*): ***usa-se ou um ou outro***

Ficheiros mapeados em memória

Restrições ao uso de ficheiros mapeados:

Manipulação do conteúdo da memória partilhada quando usada para comunicar entre processos (uso em simultâneo em vários processos):

- Seguem-se as restrições habituais usadas em comunicação entre processos (por exemplo, *named pipes*):
  - Apenas se pode colocar informação que faça sentido em todos os processos envolvidos.
  - Ou seja, estão excluídas coisas que só façam sentido no processos que as criou, tais como ***handles*** e **ponteiros** (as coisas apontadas podem não ser partilhadas e a zona de memória partilhada nem sequer é mapeada no mesmo local em cada processo).
  - C++ O uso de objectos polimórficos envolve ponteiros => uso restringido

Ficheiros mapeados em memória

Restrições ao uso de ficheiros mapeados:

Uso em simultâneo (concorrente) da mesma memória partilhada por vários processos/*threads*

- Trata-se de um caso típico de acesso concorrente a dados partilhados
- Normalmente exigirá o uso de ***mutexes*/semáforos** para garantir que duas *threads*/processos não destroem o trabalho uma da outra e para garantir que o conteúdo dos dados permanece coerente

Importante: O mecanismo de memória partilhada/ficheiros mapeados, não inclui de origem nenhum mecanismo de sincronização. Terá que ser o programador a tratar desta questão explicitamente com o API de sincronização (usando *mutexes*, semáforos, outros)

Ficheiros mapeados em memória – API Win32

Padrão de uso de ficheiros mapeados

Sem ser para partilha de dados (apenas aceder ao ficheiro como uma matriz)

1 – Obtém handle para o ficheiro
CreateFile *(indica-se o ficheiro)*
2 – Criar o objecto FileMapping usando o handle anterior (handle do ficheiro)
CreateFileMapping *(indica-se o handle e o tamanho)*
3 – Mapear uma vista do ficheiro no seu espaço de endereçamento
MapViewOfFile *(indica-se a zona pretendida e obtém-se um ponteiro)*
4 – Usa a memória partilhada através da vista (sintaxe habitual ponteiros **** -> [ ]***)

5 - Desmapeia a vista
UnmapViewOfFile *(indica-se o ponteiro)*
6 – Fecha o *handle do objeto Ficheiro Mapeado*
CloseHandle *(handle do filemapping)*
7 – Fecha o *handle do ficheiro*
CloseHandle *(handle do ficheiro)*

Ficheiros mapeados em memória – API Win32

Padrão de uso de ficheiros mapeados para comunicação entre processos

Um processo, inicialmente:

1 – Cria um objecto memória partilhada, obtendo um *handle*
CreateFileMapping

2 – Mapeia uma vista da memória partilhada no seu espaço de endereçamento
MapViewOfFile

3 – Usa a memória partilhada através da vista (***ponteiros*** ou ***[ ]***)

4 - Desmapeia a vista, eventualmente no final do processo
UnmapViewOfFile

4 – Fecha o handle
CloseHandle

Ficheiros mapeados em memória – API Win32

Padrão de uso de ficheiros mapeados para comunicação entre processos

Outro/restantes processos:

1 – Obtém handle para o objecto memória partilhada
OpenFileMapping

2 – Mapeia uma vista da memória partilhada no seu espaço de endereçamento
MapViewOfFile

3 – Usa a memória partilhada através da vista (***ponteiros*** ou ***[ ]***)

4 – Desmapeia a vista, eventualmente no final do processo
UnmapViewOfFile

5 – Fecha o *handle*
CloseHandle

Ficheiros mapeados em memória – API Win32

CreateFileMapping

```
HANDLE WINAPI CreateFileMapping (
    HANDLE hFile,                   // Ficheiro a usar
    LPSECURITY_ATTRIBUTES lpAttributes,
    DWORD flProtect,                // flags para: escrita/leitura/execução
    DWORD dwMaximumSizeHigh,        // Tamanho dado em duas DWORDS
    DWORD dwMaximumSizeLow,         // (mais significativo e menos significativo)
    LPCTSTR lpName                  // Nome a dar ao recurso (fich. mapeado)
);
```

- Se hFile for INVALID_HANDLE_VALUE, então obter-se-á apenas memória partilhada (não estará associado a nenhum ficheiro)
- Esta função é usada para criar um objecto de memória partilhada.
- Posteriormente terá que se mapear uma porção dessa memória (uma ***view***) para a poder usar
- Deve ser chamada a função CloseHandle para remover a memória partilhada. Se existir um ficheiro associado, deve ser fechado/removido à parte

Ficheiros mapeados em memória – API Win32

OpenFileMapping

```
HANDLE WINAPI OpenFileMapping(
    DWORD dwDesiredAccess,      // acesso pretendido
    BOOL bInheritHandle,
    LPCTSTR lpName              // nome dado ao recurso (ficheiro mapeado)
);
```

- Esta função é usada para obter acesso ao objecto de memória partilhada (dado o seu nome) criado (normalmente) por outro processo/*thread*.
- De seguida é necessário mapear uma parte (bloco) → uma ***View*** ← dessa memória partilhada através da função MapViewOfFile

Ficheiros mapeados em memória – API Win32

MapViewOfFile

```
LPVOID WINAPI MapViewOfFile (
    HANDLE hFileMappingObject,     // Handle do ficheiro mapeado
    DWORD dwDesiredAccess,         // Flags de acesso (ler, escrever)
    DWORD dwFileOffsetHigh,        // Início dentro do bloco pretendido
    DWORD dwFileOffsetLow,         // dentro do ficheiro (+signific., -signific.)
    SIZE_T dwNumberOfBytesToMap    // Tamanho da view pretendida
);
```

Esta função mapeia uma porção da memória partilhada, devolvendo o ponteiro para onde ficou mapeada a vista

- **Handle:** obtido com CreateFileMapping ou OpenFileMapping
- **Acesso desejado:** é necessário indicar o tipo de acesso pretendido
  Exemplos: FILE_MAP_ALL_ACCESS, FILE_MAP_READ, FILE_MAP_WRITE
- **Offset:** início do bloco pretendido dentro do objecto. Indicado através de duas DWORDS (parte mais e menos significativa)

Ficheiros mapeados em memória – API Win32

MapViewOfFile

- Parâmetro offset (deslocamento)
  - O valor do deslocamento (combinação dos dois parâmetros dwFileOffsetHigh e dwFileOffsetLow) tem que ser ***múltiplo do valor da granularidade*** mínima do sistema
  - Se o deslocamento não coincidir exactamente com um múltiplo da granularidade mínima, terá que se indicar o valor imediatamente abaixo e contar com a diferença explicitamente no algoritmo que manipula a *view* obtida
  - O valor da granularidade pode ser obtido com a função **GetSystemInfo** (ver adiante o uso desta função)

  Se o deslocamento for 0 (a *view* começa no início da memória partilhada), não é necessário esta preocupação: 0 é sempre múltiplo desse valor)

Ficheiros mapeados em memória – API Win32

MapViewOfFile

Aspectos importantes - 1

- O sistema decide o ponto, dentro do espaço de endereçamento, onde mapeia a vista
  - Diferentes processos irão ver a mesma zona de memória em diferentes localizações
  - Qualquer informação que dependa de um endereço em particular não será compatível como uso em memória partilhada porque irá parecer que está em locais distintos
    - Regra geral: um ponteiro só faz sentido no contexto do processo onde foi obtido
  - Alguns tipos de dados podem ter ponteiros internamente (exemplo: string e vector de c++) e não devem ser colocados dentro de uma zona de memória partilhada

Ficheiros mapeados em memória – API Win32

MapViewOfFile

Aspectos importantes - 2

- "mallocs" não alocam dentro de vistas de memória partilhada
  - De facto, "mallocs" usam a zona *Heap*, que é uma zona de memória distinta das vistas de memória partilhada
  - Significa que qualquer zona de memória alocada dinamicamente terá que ser copiada para dentro da zona de memória partilhada
  - Num caso destes , apenas se está a mover informação de/para memória partilhada. A alocação propriamente dita já foi feita (indirectamente) com o mapeamento da vista
  - Cuidado com o uso de estruturas de dados (em memória partilhada) que usem informação contida em memória dinâmica
    - A informação é alocada fora da zona de memória partilhada e os outros processos não irão ver essa informação
    - Mesmo que vissem, iriam aparentar estar em outros endereços e continuaria a não funcionar
    - Exemplo: listas ligadas

Ficheiros mapeados em memória – API Win32

GetSystemInfo

```
void GetSystemInfo(
  LPSYSTEM_INFO lpSystemInfo
);
```

Esta função é muito simples de usar.

Recebe um ponteiro para uma estrutura **SYSTEM_INFO** (slide seguinte) que é preenchida com vários aspectos sobre o sistema, incluindo o valor da **granularidade mínima de alocação** (importante para o *offset* em MapViewOfFile)

Ficheiros mapeados em memória – API Win32

Campos da estrutura SYSTEM_INFO

```
union {
   DWORD dwOemId;
   struct {
     WORD wProcessorArchitecture;
     WORD wReserved;
   } DUMMYSTRUCTNAME;
 } DUMMYUNIONNAME;
 DWORD     dwPageSize;
 LPVOID    lpMinimumApplicationAddress;
 LPVOID    lpMaximumApplicationAddress;
 DWORD_PTR dwActiveProcessorMask;
 DWORD     dwNumberOfProcessors;
 DWORD     dwProcessorType;
 DWORD     dwAllocationGranularity;
 WORD      wProcessorLevel;
 WORD      wProcessorRevision;
```

Ficheiros mapeados em memória – API Win32

UnmapViewOfFile

```
LPVOID WINAPI UnmapViewOfFile(
    LPCVOID lpBaseAddress      // Ponteiro para o inicio da view
);                             // (obtido com MapViewOfFile)
```

Esta função *desmapeia* uma vista do espaço de endereçamento do processo que a invoca –> o processo deixa de "ver" essa zona de memória partilhada

O único parâmetro é o ponteiro obtido quando se mapeou a vista

Nota: o handle usado para aceder ao objecto de memória partilhada deverá ser fechado com a função CloseHandle

Ficheiros mapeados em memória – API Win32

FlushViewOfFile

```
BOOL WINAPI FlushViewOfFile(
    LPCVOID lpBaseAddress,          // endereço base da vista
    SIZE_T  dwNumberOfBytesToFlush  // 0 = todos
)
```

Esta função pode ser usada antes de fechar a vista para garantir que as alterações são efectivamente escritas no ficheiro associado à memória partilhada (se existir)