Sistemas Operativos 2

2021/22

Comunicação interprocesso em Win32

com *Named Pipes*

1

Tópicos

Comunicação com *Pipes* e *Named Pipes*

Bibliografia específica para este capítulo:

- Windows System Programming; Johnson M. Hart; 4th Ed.
- Advanced Windows (3rd Edition); Jeffrey Richter
- WindowsNT 4 Programming; Herbert Schildt
- MSDN Library – PlattformSDK: DLLs, Processes, and Threads (disponível online e no ISEC)

2

Windows NT – Comunicação inter-processos Win32

Existem diversas formas de comunicar entre processos em Win32
- Clipboard, DDE, Ficheiros mapeados, mailslots, pipes, sockets

Nestes slides: **Named Pipes**
- Utilização muito comum na programação para Windows
  - Esta matéria é normalmente usada no trabalho prático
- Relacionam-se fortemente com programação *multi-threaded*

Windows NT – Comunicação inter-processos - *Pipes*

Named pipes

Características principais

- Permitem a comunicação entre processos de uma forma simplificada
- Permitem a comunicação em rede (o nome do pipe pode incluir o nome da máquina)
- Muito associados ao modelo cliente-servidor: o API já inclui parte da funcionalidade da lógica e encaminha mesmo para esse modelo

Windows NT – Comunicação inter-processos - *Pipes*

Os named pipes podem ser

De uma só via

O processo ou lê ou escreve (mas não ambos) na sua extremidade do pipe

De duas vias (pipe duplex)

O processo pode ler e escrever na sua extremidade do pipe (mas não ao mesmo tempo: tem que se gerir a sequência de leituras/escritas)

- Os pipes duplex em Windows são apenas ***half-duplex***, o que significa que o processo pode ler e escrever no mesmo pipe, mas não em simultâneo
- É possível ler e escrever em simultâneo (ex., usando duas *threads*), mas *mas apenas* com *operações não bloqueantes* e a complexidade adicional pode não compensar

Windows NT – Comunicação inter-processos - *Pipes*

Named pipes

- Muito associados ao modelo cliente-servidor
- O programador deverá seguir uma lógica típica já planeada de origem para este mecanismo de comunicação.

Operações do lado dos servidor

- Criação: **`CreateNamedPipe`**
- Esperar ligação de um cliente: **`ConnectNamedPipe`**
- Escrita/leitura: **`ReadFile / WriteFile`**

Operações do lado do cliente

- Associação a um pipe existente: **`CreateFile`** ou **`CallNamedPipe`**
- Esperar que um servidor esteja à escuta num pipe: **`WaitNamedPipe`**
- Escrita/leitura: **`ReadFile / WriteFile`**

Windows NT – Named Pipes

Criação do Named Pipe - CreateNamedPipe

```
HANDLE WINAPI CreateNamedPipe(
   LPCTSTR               lpName,
   DWORD                 dwOpenMode,
   DWORD                 dwPipeMode,
   DWORD                 nMaxInstances,
   DWORD                 nOutBufferSize,
   DWORD                 nInBufferSize,
   DWORD                 nDefaultTimeOut,
   LPSECURITY_ATTRIBUTES lpSecurityAttributes
)
```

- **lpName ->** permite usar o pipe em processos diferentes. Formato: **\\.\pipe\nomedopipe**
- **dwOpenMode ->** Modo de abertura PIPE_ACCESS_DUPLEX, PIPE_ACCESS_INBOUND e PIPE_ACCESS_OUTBOUND
- **dwPipeMode ->** Tipo de pipe: PIPE_TYPE_BYTE , PIPE_TYPE_MESSAGE (igual em todas as instâncias), PIPE_READMODE_BYTE, PIPE_READMODE_MESSAGE (depende das anteriores), PIPE_WAIT, PIPE_NOWAIT (pode ser diferente em cada instância), PIPE_ACCEPT_REMOTE_CLIENTS, PIPE_REJECT_REMOTE_CLIENTS (pode ser dif. p/ instânc.)
- **nMaxInstances ->** Número máximo de instâncias abertas em simultâneo (só para a 1º vez). PIPE_UNLIMITED_INSTANCES indica o limite definido no sistema

Windows NT – Named Pipes

Esperar ligação - ConnectNamedPipe

```
BOOL WINAPI ConnectNamedPipe(
   HANDLE       hNamedPipe,
   LPOVERLAPPED lpOverlapped
);
```

Aguarda que um cliente se ligue (a uma instância) do named pipe

- **hNamedPipe ->** Handle obtido com CreateNamedPipe
- **lpOverlapped ->** Ponteiro para estrutura OVERLAPPED. Só pode ser NULL se não se tiver usado a flag FILE_FLAG_OVERLAPPED em CreateNamedPipe

Windows NT – Named Pipes

Terminar ligação da instância - DisconnectNamedPipe

```
BOOL WINAPI DisconnectNamedPipe(
   HANDLE hNamedPipe
);
```

Liberta a instância do named pipe

- **hNamedPipe ->** Handle obtido com CreateNamedPipe

Windows NT – Comunicação inter-processos - *Pipes*

Named pipes – Padrão de uso

Servidor

- O servidor chama a função **`CreateNamedPipe`** para criar a primeira instância (e seguintes) do named pipe
- Podem existir várias instâncias em simultâneo, permitindo um servidor *multi-threaded* de forma simplificada
- O servidor usa a função **`ConnectNamedPipe`** para aguardar um pedido de ligação à instância do named pipe. O atendimento dessa instância pode ser feito numa *thread* independente, libertando o servidor para criar outra instância e aguardar + processar outro cliente nela
- Para desligar do cliente, o servidor aguarda que os dados já tenham sido lidos pelo cliente usando a função **`FlushFileBuffers`**. Após este passo pode-se desligar a instância do pipe com **`DisconnectNamedPipe`** e de seguida **`CloseHandle`**

Windows NT – Comunicação inter-processos - *Pipes*

Named pipes

Cliente

- O cliente utiliza a função **`CreateFile`** ou **`CallNamedPipe`** para obter um handle (do lado cliente) para uma instância do named pipe
- A função **`WaitNamedPipe`** permite ao cliente aguardar que exista uma instância do servidor do pipe disponível (= aguardar que o servidor faça um **`ConnectNamedPipe`**). Cada cliente que se liga consome (ocupa) uma instância
- O cliente interage com o servidor de acordo com um protocolo qualquer predefinido para essa aplicação cliente-servidor
- O cliente termina a interação fechando os handles (com reflexo no servidor). A lógica de uso é muito semelhante aos ficheiros

Windows NT – Named Pipes

Abertura do Named Pipe - CreateFile

```
HANDLE WINAPI CreateFile(
   LPCTSTR               lpFileName,
   DWORD                 dwDesiredAccess,
   DWORD                 dwShareMode,
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
   DWORD                 dwCreationDisposition,
   DWORD                 dwFlagsAndAttributes,
   HANDLE                hTemplateFile
);
```

- **lpName ->** Nome do ficheiro/named pipe. Formato: **\\maquina\pipe\nomedopipe**
- **dwDesiredAccess ->** Modo de uso (GENERIC_READ, GENERIC_WRITE)
- **dwShareMode ->** 0=acesso exclusivo ou FILE_SHARE_READ, FILE_SHARE_WRITE (exemplos)
- **dwCreatioDisposition ->** CREATE_ALWAYS, CREATE_NEW, OPEN_EXISTING, etc.
- **dwFlagsAndAttributes ->** Atributos de ficheiro (ex., FILE_ATTRIBUTE_NORMAL)
- **hTemplateFile** -> Handle para um ficheiro já aberto; este será criado com os atributos desse (ignorado se este ficheiro já existir. Pode ser NULL)

Windows NT – Named Pipes

Abertura do Named Pipe – CreateFile

Alguns cuidados a observar após CreateFile

- No lado do cliente o *named pipe* deverá ter as mesmas propriedades de funcionamento que as especificadas no lado do servidor com ***CreateNamedPipe***
- ***CreateFile*** abre (mais) uma instância do **named pipe**. No lado do servidor terá que ser executado (mais) um ***ConnectNamedPipe*** correspondente
- A função ***CreateFile*** poderá abrir o *named pipe* com propriedades por omissão que podem não corresponder ao que foi definido no servidor e é preciso ter cuidado com este aspecto (cuidado com os *defaults*)

Exemplo

- O modo MESSAGE: por omissão CreateFile abre em modo BYTE.
  - Solução: mudar com a função ***SetNamedPipeHandleState***

Windows NT – Named Pipes

Abertura do Named Pipe - WaitNamedPipe

```
BOOL WINAPI WaitNamedPipe(
   LPCTSTR lpNamedPipeName,
   DWORD   nTimeOut
);
```

Aguarda por uma instância disponível ou pelo time-out.

- **nTimeOut ->** é dado em milisegundos (NMPWAIT_FOREVER_WAIT, NMPWAIT_USE_DEFAULT_WAIT)

Windows NT – Named Pipes

Modificação das propriedades do pipe
-> SetNamedPipeHandleState

```
BOOL SetNamedPipeHandleState(
      HANDLE hNamedPipe,              // handle para o pipe (já aberto)
      LPDWORD lpMode,                 // novo modo do pipe - NULL mantém
      LPDOWRD lpMaxCollectionCount,   // novo buffer size - NULL mantém
      LPDOWRD lpCollectDataTimeout);  // novo timeout - NULL mantém
```

- **lpMode ->** PIPE_READMODE_BYTE, PIPE_READMODE_MESSAGE, PIPE_WAIT, PIPE_NOWAIT

Windows NT – Named Pipes

Acerca dos modos

- PIPE_READMODE_BYTE
- PIPE_READMODE_MESSAGE

- Têm que bater certo no lado do servidor e no lado do cliente
  - PIPE_TYPE_BYTE – PYPE_READMODE_BYTE
  - PIPE_TYPE_MESSAGE – PYPE_READMODE_MESSAGE

Nota: o *default* na abertura com *CreateFile* é BYTE e tem que se mudar, caso seja outro o tipo usado no servidor

- Modo BYTE
  - O sistema não se preocupa com a fronteira entre mensagens: é tudo uma *byte stream*
- Modo MESSAGE
  - O sistema percebe que um *write* corresponde a uma mensagem e ajuda a gerir na leitura (se se ler menos que uma mensagem inteira devolve um código de erro)
  - Em situações de rede, o sistema tenta enviar toda a mensagem numa única operação
  - O modo MESSAGE é mais útil no caso em que se enviam mensagens de tipo e tamanho previamente conhecido

Windows NT – *Named pipes* – Exemplo: servidor

Exemplo de aplicação single-threaded (baseado num ex. disponível no MSDN)
Lógica de interação: 1º) Cliente → Servidor, depois
2º) Servidor → Cliente, depois repete, alternando sempre
"*falo eu, falas tu*"

A) Servidor (aplicação consola)

1 – Criação do pipe e espera por clientes

```
#include <windows.h>
#include <stdio.h>

#define BUFSIZE 4096

int _tmain(int argc, TCHAR *argv[]) {
    BOOL fConnected;
    DWORD dwThreadId;
    HANDLE hPipe, hThread;
    LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");
        /* recordar que "\\" transforma-se em um só "\" */
```


Windows NT – *Named pipes* – Exemplo: servidor

```
// O ciclo principal cria (mais uma) instância do named pipe
// e depois espera que um cliente se ligue

    while (1) {
        hPipe = CreateNamedPipe(
            lpszPipename,             // nome do pipe
            PIPE_ACCESS_DUPLEX,       // acesso read/write (duplex)
            PIPE_TYPE_MESSAGE |       // pipe to tipo message
            PIPE_READMODE_MESSAGE |   // modo message-read
            PIPE_WAIT,                // modo "blocking"
            PIPE_UNLIMITED_INSTANCES, // max. instâncias
            BUFSIZE,                  // tam. buffer output
            BUFSIZE,                  // tam. Buffer input
            NMPWAIT_USE_DEFAULT_WAIT, // time-out para o cliente
            NULL);                    // atributos segurança default

        if (hPipe == INVALID_HANDLE_VALUE) {
            _tprintf(TEXT("CreatePipe falhou"));
            return 0;
        }
```

Windows NT – *Named pipes* – Exemplo: servidor

```
    // Aguarda a ligação de um cliente

        fConnected = ConnectNamedPipe(hPipe, NULL);
        if (!fConnected && (GetLastError() == ERROR_PIPE_CONNECTED) )
            fConnected = TRUE;

        if (fConnected) {
            // atender o cliente (apresentado mais adiante)
            AtendeCliente(hPipe)   // feita mais adiante
            // Este exemplo, não sendo multi-thread
            // implica que só trata 1 cliente de cada vez (em série)
        }
        else
           // Este cliente não se conseguiu ligar
           // por isso fecha-se o pipe (continua para o prox cliente)
           CloseHandle(hPipe);
        // Próxima iteração -> mais uma instância e possível cliente
    } // ciclo principal          (apenas depois de ter atendido este cliente)

    return 1;
} // fim do programa servidor
```


Windows NT – *Named pipes* – Exemplo: servidor

Servidor – 2 – Atendimento do cliente – troca de mensagens

Esta função é invocada de dentro do ciclo principal

Podia ser no contexto de uma *thread* independente

```
void AtendeCliente (HANDLE hPipe) {
    char chRequest[BUFSIZE];
    char chReply[BUFSIZE];
    DWORD cbBytesRead, cbReplyBytes, cbWritten;
    BOOL fSuccess;

    while (1) {  // ciclo p/ permitir haver N x pergunta-resposta)
    // Lê dados do cliente via pipe.  // mas cliente só vai fazer 1x
        fSuccess = ReadFile(
        hPipe,                  // handle para o pipe
        chRequest,              // buffer para receber os dados
        BUFSIZE * sizeof(char), // tam do buffer (bytes a ler)
        & cbBytesRead,          // num. de bytes lidos
        NULL);                  // não é overlapped I/O

   // se não houver mais dados sai
   if (! fSuccess || cbBytesRead == 0) break;
```

Windows NT – *Named pipes* – Exemplo: servidor

```
      // Escreve a resposta para o pipe
      fSuccess = WriteFile(
           hPipe,        // handle para o pipe
           chReply,      // buffer com os dados
           cbReplyBytes, // num. de bytes a escrever
           & cbWritten,  // num. de bytes escritos
           NULL);        // não é overlapped I/O

      // verifica operação efectuada
      if (! fSuccess || cbReplyBytes != cbWritten)
          break;
  } // fim do ciclo de escrita

  // faz flush ao pipe para garantir que o cliente já leu tudo
  // antes de desligar esta instância do pipe

   FlushFileBuffers(hPipe);
   DisconnectNamedPipe(hPipe);
   CloseHandle(hPipe);

} // fim do atendimento deste cliente
```


Windows NT – *Named pipes* – Exemplo: cliente

B) Cliente de named pipe (aplicação consola)

```
#include <windows.h>
#include <stdio.h>

#define BUFSIZE 512

int _tmain(int argc, TCHAR *argv[]) {
    HANDLE hPipe;
    LPTSTR lpvMessage;
    TCHAR chBuf[BUFSIZE];
    BOOL fSuccess;
    DWORD cbRead, cbWritten, dwMode;
    LPTSTR lpszPipename = "\\\\.\\pipe\\mynamedpipe";

    lpvMessage = argv[1];   // msg dada na linha de comando
```

Windows NT – *Named pipes* – Exemplo: cliente

```
// tenta repetidamente abrir o named pipe (em ciclo)
// espera se for preciso

  while (1) { // o ciclo é só para abrir ligação com servidor
      hPipe = CreateFile( // tenta obter um handle para o pipe
          lpszPipename,   // nome do pipe
          GENERIC_READ |  // acesso read & write
          GENERIC_WRITE,
          0,              // sem "sharing"
          NULL,           // attributos segurança default
          OPEN_EXISTING,  // deve abrir um pipe já existente
          0,              // atributos default
          NULL);          // sem "ficheiro" "template"


      // se obteve um handle válido, sai do ciclo
      if (hPipe != INVALID_HANDLE_VALUE)
          break;  // pipe aberto -> sai do ciclo
```


Windows NT – *Named pipes* – Exemplo: cliente

```
      // erro = outro que não PIPE_BUSY -> desiste
      if (GetLastError() != ERROR_PIPE_BUSY) {
          printf("Não foi possível abrir o pipe");
          return 0; // num programa real: não terminar logo a
      }       // aplicação. Pode dar para fazer outras tarefas

      // ERROR_PIPE_BUSY – todas as instâncias estavam ocup.
      // tenta mais 1 (ou N) vezes mas espera entre cada tent.
      // (espera 20 segundos "porque sim" – é só um exemplo )
      // isto é uma estratégia definida pelo programador
      // e podia ser diferente

      if (!WaitNamedPipe(lpszPipename, 20000)) {
          printf("não foi possivel abrir o pipe");
          return 0;
      }// O ciclo serve para tentar várias vezes porque o serv
       // pode estar ocupado. Pode-se sair após N tentativas.
       // Com serv multi-threaded este ciclo é menos importante
```

Windows NT – *Named pipes* – Exemplo: cliente

```
        // o ciclo termina quando consegue abrir o pipe
        // sai do ciclo via break (mais atrás)

    } // fim do ciclo while em que tenta ligar ao pipe

    // Neste momento o pipe está ligado ao servidor
    // mudar para modo de leitura de mensagem
    // porque foi esse o modo (MESSAGE) usado no servidor

    dwMode = PIPE_READMODE_MESSAGE;
    fSuccess = SetNamedPipeHandleState(
        hPipe,          // handle para o pipe
        & dwMode,       // novo pipe mode
        NULL,           // maximum bytes = NULL (não altera)
        NULL);          // maximum time = NULL (não altera)
    if (!fSuccess) {
       printf("SetNamedPipeHandleState falhou");
       return 0;
    }
```


Windows NT – *Named pipes* – Exemplo: cliente

```
// lógica de interação nesta aplicação (servidor é diferente)
// -> cliente escreve 1 msg, servidor responde, fim
// 1º: cliente escreve,
// 2º cliente lê

// 1º Escrever uma mensagem no pipe

  fSuccess = WriteFile(     // no exemplo: envia sempre chars
      hPipe,                // handle para o pipe
      lpvMessage,           // mensagem
      (lstrlen(lpvMessage)+1)*sizeof(TCHAR), // tam da msg
      &cbWritten,           // bytes escritos
      NULL);                // não é uma operação overlapped

  if (!fSuccess) {
      _tprintf(TEXT("WriteFile falhou"));
      return 0;
  }
```

Windows NT – *Named pipes* – Exemplo: cliente

```
  // 2º Ler dados do pipe

    do {
        fSuccess = ReadFile(
            hPipe,     // handle para o pipe
            chBuf,     // buffer para receber os dados
            BUFSIZE * sizeof(TCHAR),  // tam. do buffer
            & cbRead,  // num. bytes a ler
            NULL);     // não é overlapped

        // se não houver mais dados sai
        if (! fSuccess && GetLastError() != ERROR_MORE_DATA)
            break;

        printf("%s\n", chBuf ); // mostra informação lida
    } while (!fSuccess);  // repete até não haver mais dados
    // = "até a resposta terminar" (o serv não envia mais nada)

    CloseHandle(hPipe);
    return 0;
}  // fim do exemplo
```