Comunicação em Windows

Named pipes e mailslots
Windows Sockets (Winsock)
Comunicação entre Janelas

Named Pipes e Mailslots

- Desenvolvidos inicialmente para OS/2 e depois portados para Windows NT
- Named pipes:
  - Suportam comunicação bidireccional
  - A nomeação é feia segundo o Windows 2000 Universal Naming Convention (UNC)
  - UNC é independente dos protocolos de comunicação e permite identificar recursos na rede
- Mailslots:
  - Suportam comunicação unidireccional assim como “broadcast”
  - Ex. de utilização do “broadcast“: serviço horário
- Segurança em named pipes e mailslots:
  - O servidor pode controlar o acesso dos clientes

Named Pipes (1)

- Comunicação efectuada entre um named pipe servidor e clientes:
  - O servidor cria o named pipe
  - Os clientes ligam-se ao named pipe
  - Um named pipe não pode ser criado num computador remoto
- Um named pipe tem um nome com o formato:
  - \\Server\Pipe\PipeName
  - \\Server indica o nome do computador onde o named pipe se encontra
  - O nome pode ser do tipo DNS (ex.:, mspress.microsoft.com), NetBIOS (mspress), ou IP (255.0.0.0).
- Criação do named pipe é feita com função Win32 CreateNamedPipe com argumentos:
  - Descriptor de segurança para controle de acesso
  - Flag que indica se a comunicação é bidireccional ou unidirecional
  - Número que indica o número máximo de ligações simultâneas que o named pipe suporta
  - Flag que indica se o named pipe funciona em byte mode ou message mode

Named Pipes (2)

- Byte mode:
  - Os dados são enviados em stream
  - Implica que o emissor e receptor têm de formatar os dados em causa
- Message mode:
  - Simplifica a programação dos intervenientes
  - Cada operação de “receive” recebe uma mensagem na sua totalidade
- Depois da criação do named pipe, o servidor invoca a função ConnectNamedPipe da Win32:
  - Permite que o named pipe estabeleça ligações com os clientes
  - ConnectNamedPipe pode ser efectuado sincronamente ou asincronamente
  - ConnectNamedPipe só se completa quando um cliente faz um pedido de estabelecimento de ligação (análogo ao *accept* nos sockets)

Named Pipes (3)

- Um cliente usa as funções da Win32 CreateFile ou CallNamedPipe:
  - Permite estabelecer a ligação ao named pipe que foi criado pelo servidor e no qual este já invocou a função ConnectNamedPipe
  - A identificação do cliente e o tipo de acesso requisitado (read ou write) são validados tendo em conta o descriptor de segurança
  - Se o cliente tem permissão para aceder ao named pipe, recebe um descriptor (do tipo HANDLE) no retorno da função
  - Este descriptor representa o "client-side" do named pipe
- Depois da ligação estar estabelecida:
  - O cliente e o servidor podem usar as funções ReadFile e WriteFile para enviar e receber dados via pipe
  - Os named pipes suportam comunicação síncrona e assíncrona

Named Pipes - Exemplo

Cliente

```
#include <windows.h>
#define SZ sizeof(TCHAR)
int _tmain(int argc, TCHAR *argv[]) {
  HANDLE hPipe;
  LPTSTR lpNome= TEXT("\\\\.\\pipe\\serv");
  LPTSTR lpMens= TEXT("mensagem c-->s");
  TCHAR buffer[MAX];
  DWORD nsent, nrecv;
  hPipe = CreateFile( lpNome,
    GENERIC_READ|GENERIC_WRITE,...);
  if (hPipe == INVALID_HANDLE_VALUE)
    perror ("associar");
  if (! WriteFile(hPipe,lpMens,(lstrlen(
    lpMens) +1 ) * SZ, &nsent, NULL)
  || (size!=nsent) )
      perror ("enviar");
  if (! ReadFile(hPipe, buffer, MAX*SZ,
    &nrecv, NULL) perror ("receber");
  CloseHandle(hPipe);
}
```

Servidor

```
#include <windows.h>
#define SZ sizeof(TCHAR)
int _tmain(int argc, TCHAR *argv[]) {
/* Declaração de variáveis */
  hPipe = CreateNamedPipe( lpNome,
    PIPE_ACCESS_DUPLEX, PIPE_TYPE_BYTE |
    PIPE_READMODE_BYTE | PIPE_WAIT,
    PIPE_UNLIMITED_INSTANCES,
    MAX,MAX,0,NULL);
  if (hPipe == INVALID_HANDLE_VALUE)
    perror ("criar");
  if ( ! ConnectNamedPipe(hPipe, NULL) &&
    (GetLastError()!= ERROR_PIPE_CONNECTED)
   perror("aceitar ligação");
  if (! ReadFile(hPipe, buffer, MAX*SZ,
    &nrecv, NULL) perror ("receber");
  /* Processa mensagem em buffer */
  if (! WriteFile(hPipe,lpMens,(lstrlen(
    buffer) +1 ) * SZ, &nsent, NULL) ||
    nsent!=size)
   perror ("enviar");
  FlushFileBuffers(hPipe);
  DisconnectNamedPipe(hPipe);
  CloseHandle(hPipe);
}
```

Named Pipes (4)

- Leitura não destrutiva/sondagem (**PeekNamedPipe**)
  - Permite aferir da existência de dados no named pipe disponíveis para serem lidos sem que para isso seja obrigado a efectuar uma operação de leitura.
  - Permite aceder ao conteúdo dos dados no named pipe, dados de forma não-destrutiva, para que não sejam removidos do *named pipe* e continuem disponíveis para uma operação de leitura subsequente.
  - Dois argumentos adicionais permitem devolver, por referência, número total de bytes disponíveis no pipe, e na primeira mensagem disponível (quando em *message mode*).

```
BOOL PeekNamedPipe(
      HANDLE hNamedPipe,
      LPVOID lpBuffer,
      DWORD nBufferSize,
      LPDWORD lpBytesRead,
      LPDWORD lpTotalBytesAvail,
      LPDWORD lpBytesLeftThisMessage
);
```


Named Pipes (5)

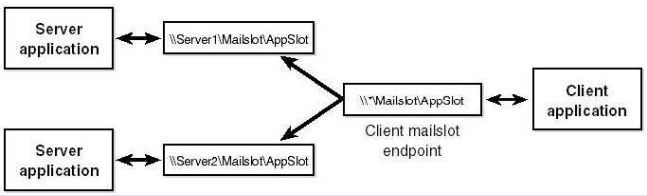
- Agrupamento de operações sobre named pipes (baseados em mensagens)
  - Permite realizar a programação da interação entre processos produtor e consumidor de forma mais resumida
  - **TransactNamedPipe** no contexto de um named pipe já ligado
    - encapsula, numa única invocação, uma operação de escrita, e de leitura, (chamada remota de procedimento) tendo como argumentos a reunião dos das funções ReadFile e WriteFile.
    - comunicação usando named pipes baseados em mensagens definida através de sequências de interacções completas (escrita de um pedido e leitura da resposta a este).
  - **CallNamedPipe** que abrange toda a comunicação
    - Encapsula: 1) o estabelecimento da ligação a um *named pipe* indicado pelo seu nome, com um período máximo de espera, 2) a operação de escrita, 3) operação de leitura, e 4) o fecho do pipe no fim da interação.

Mailslot (1)

- O servidor cria uma mailslot usando a função CreateMailslot:
  - Recebe como argumento um nome do tipo "\\.\Mailslot\MailslotName"
  - As mailslots não podem ser criadas em computadores remotos
  - Recebe como argumento um descriptor de segurança que permite efectuar controle de acesso dos clientes
  - Devolve um descriptor (do tipo HANDLE)
- Depois de criada a mailslot:
  - O servidor simplesmente aguarda que lhe sejam enviadas mensagens
  - Para tal invoca a função ReadFile sobre o *handle* respectivo

Mailslot (2)

- O esquema de nomeação das mailslots é análogo ao dos named pipes:
  - Diferença que permite o broadcast para um grupo de mailslots
- Cliente invoca *CreateFile* indicando:
  - Nome da mailsot em particular (ex.: \\Server\Mailslot\MailslotName), ou
  - Conjunto de mailslots (ex.: *\Mailslot\MailslotName ou \\DomainName\Mailslot\MailslotName)
  - Esta função retorna um *handle*
- Envio de mensagens:
  - Cliente invoca *WriteFile* sobre o *handle* antes obtido com *CreateFile*



Mailslots: exemplo

CLIENTE

```
#include <windows.h>
#define SZ sizeof(TCHAR)
int _tmain(int argc, TCHAR *argv[]) {
  HANDLE hSlotCli, hSlotSrv;
  LPTSTR pSrv=TEXT("\\\\.\\mailslot\\serv");
  LPTSTR lpCli=TEXT("\\\\.\\mailslot\\cli");
  LPTSTR lpMens;
  TCHAR buffer[MAX];
  DWORD nsent, nrecv;
  hSlotCli = CreateMailslot(lpCli, 0,
      MAILSLOT_WAIT_FOREVER, NULL);
  if (hSlotCli == INVALID_HANDLE_VALUE)
    perror ("criar");
  hSlotSrv = CreateFile( lpSrv,
         GENERIC_READ|GENERIC_WRITE,...);
  if (hSlotSrv == INVALID_HANDLE_VALUE)
    perror ("associar");
/* Cria string contendo texto a enviar e */
/* nome do mailslot cliente para resposta */
  if (! WriteFile(hSlotSrv,lpMens,(lstrlen(
    lpMens) +1 ) * SZ, &nsent, NULL) ||
  (size!=nsent) )
    perror ("enviar");
  if (! ReadFile(hSlotCli, buffer, MAX*SZ,
    &nrecv, NULL) perror ("receber");
  CloseHandle(hSlotSrv);
  CloseHandle(hSlotCli);
}
```

SERVIDOR

```
#include <windows.h>
#define SZ sizeof(TCHAR)
int _tmain(int argc, TCHAR *argv[]) {
  HANDLE hSlotCli, hSlotSrv;
LPTSTR lpSrv; /* mailslot\serv */
LPTSTR lpCli; /* a preencher */
  TCHAR buffer[MAXMENS];
  DWORD nsent, nrecv;
  hSlotSrv = CreateMailslot(lpSrv, 0,
     MAILSLOT_WAIT_FOREVER, NULL);
  if (hSlotSrv == INVALID_HANDLE_VALUE)
    perror ("criar");

/* GetMailSlotInfo: inspecção mailslot */
  if(! ReadFile(hSlotSrv,buffer,MAX*SZ,
    &nrecv,NULL) perror ("receber");
/* Processa mensagem em buffer para lpMens*/
/* Extrai nome mailslot cliente da mensagem e
   coloca em hSlotCli */
/* Associa-se a mailslot do cliente usando
   CreateFile e retorno em hSlotCli */

  if(!WriteFile(hSlotCli,lpMens,(lstrlen(
    buffer) +1) * SZ, &nsent, NULL) ||
      (nsent!=size) )
   perror ("enviar");
  CloseHandle(hSlotCli);
  CloseHandle(hSlotSrv);
}
```


Mailslots (4)

- Leitura não destrutiva/sondagem:
  - No caso de não haver mensagens para leitura no mailslot, a execução de uma operação de leitura bloqueia o processo que pretende efectuar a leitura.
  - A tarefa pode optar por verificar, previamente, se existem mensagens em espera no mailslot, antes de executar uma operação de leitura.
  - Isto é realizado através da função **GetMailSlotInfo** que permite e inspeccionar o estado do mailslot:
    - número de mensagens disponíveis para leitura
    - dimensão da próxima mensagem.
    - dimensão máxima permitida para mensagens
    - limite de tempo que uma operação de leitura espera para que haja mensagens disponíveis.

Windows Sockets - Winsock

- É a implementação dos Sockets BSD pela Microsoft.
  - Suporta integralmente a interface dos sockets
- Interface WSA* tem algumas funcionalidades que estendem sockets:
  - I/O assíncrono
  - Negociação e monitorização de QoS (latência e largura de banda) por parte das aplicações, caso a rede o suporte
  - Extensibilidade no sentido em que Winsock pode usar vários protocolos de transporte por baixo
  - Vários espaços de nomes:
    - Ex.: Active Directory
  - Comunicação multiponto, i.e. pode enviar mensagens para vários receptores ao mesmo tempo
    - Ex: *multicast*, videoconferência, comunicação em grupo

Windows Sockets - Winsock (2)

- Suporta:
  - Transmissão de dados associada ao estabelecimento e fecho de uma ligação
    - Reduz a latência na interacção cliente-servidor. Mensagens trocadas no estabelecimento da ligação (normalmente opacas às aplicações) também carregam dados.
  - Aceitação de uma ligação avaliada dinamicamente, delegada numa função *callback* que é invocada sempre que ocorre um pedido de ligação.
  - Envio e/ou recepção em paralelo utilizando múltiplos *buffers*
    - O sistema operativo garante sequencialidade dos dados (*Scatter and Gather)*
  - Partilha de sockets entre processos não relacionados hierarquicamente
    - Ex: Duplicação de *handles*

Windows Sockets
Aceitação dinâmica e transmissão de dados no estabelecimento de ligações

- Pedido de Ligação:
  1. Inclusão de dados no próprio pedido de ligação
  2. Recepção de dados incluídos na resposta de aceitação.
  3. Negociação QoS
- Aceitação de Ligações
  1. Função predicado que decide aceitação da ligação
  2. Dados incluídos no pedido pelo cliente são passados como parâmetro à função
  3. Parâmetros adicionais
     - Ex: estado do servidor

```
int WSAConnect(
  SOCKET s,
  const struct sockaddr* name,
  int namelen,
  LPWSABUF lpCallerData,
  LPWSABUF lpCalleeData,
  LPQOS lpSQOS, ...);
```

```
SOCKET WSAAccept(
  SOCKET s,
  struct sockaddr* addr,
  LPINT addrlen,
  LPCONDITIONPROC lpfnCondition,
  DWORD dwCallbackData );
```

```
int CALLBACK ConditionFunc(
  IN LPWSABUF lpCallerId,
  IN LPWSABUF lpCallerData,
  IN OUT LPQOS lpSQOS,
  …
  IN LPWSABUF lpCalleeId,
  OUT LPWSABUF lpCalleeData,
  …
  IN DWORD_PTR dwCallbackData
);
```


Windows Sockets
Envio paralelo de múltiplos *buffers* (*Scatter and Gather*)

- **WSASend** com parâmetros:
- Vector de estruturas do tipo **WSABUF**
  - Cada **WSABUF** inclui apontador para zona tampão e sua dimensão
- Número de estruturas no vector a enviar
- Sistema assegura a sequencialidade dos dados no receptor
  - Independentemente da ordem de envio e recepção de cada *buffer*
  - Disposição dos *buffers* no receptor respeitam a original no emissor

```
int WSASend(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
  LPDWORD lpNumberOfBytesSent,
  DWORD dwFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);
```

Winsock - Comunicação Assíncrona

- A API Winsock API está integrada com a sincronização Windows (**`Events`**)
- Uma aplicação que use Winsock pode então:
  - Efectuar operações assíncronas sobre os sockets
  - Receber notificações da terminação de uma dada operação através de:
    - **Assinalar objecto sincronização (`WSAEvent` incluído em `lpOverlapped`)**
    - **Invocação de uma função "callback"**
- Facilita a implementação das aplicações:
  - A aplicação não precisa de ser *multithreaded* ou gerir objectos de sincronização (que fazem I/O rede e I/O via teclado e terminal)

```
int WSASend(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
  LPDWORD lpNumberOfBytesSent,
  DWORD dwFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);
```


Mecanismos de Comunicação entre Janelas no Windows

Window Messages

Clipboard

Comunicação entre Janelas (Windows)

- Windows suporta janelas IU de forma nativa
  - Entidade principal para o utilizador
    - Uma aplicação em execução é vista como um conjunto de janelas
  - Cada janela tem um descritor único, do tipo HWND
    - Um tipo de HANDLE específico para janelas
    - Cada componente da interface têm uma janela associada
    - O utilizador apenas se apercebe das *top-level windows* (janelas com título)
  - Cada janela pertence a uma classe (tipo de janela e código associado)
- Windows oferece mecanismos de comunicação *originais*
  - baseados na troca de mensagens entre janelas que representam eventos
  - apenas estes eram disponibilizados inicialmente no Windows
  - não existem noutros SO que apenas têm noção de processo

Comunicação entre Janelas (Windows)

- Interlocutores na comunicação são as próprias janelas das aplicações em execução
  - *Input* do utilizador (teclado, rato) é mapeado em mensagens enviadas a janelas
  - Controlo das aplicações pelo utilizador (redimensionamento, minimização de janelas)
  - Gestão da interface é regida pela troca de mensagens entre janelas
  - Aplicações podem enviar mensagens às suas próprias janelas
  - Aplicações podem criar janelas escondidas exclusivamente para comunicação
  - Utilizador pode coordenar a comunicação entre janelas
    - *Ex.: copy-paste, drag-and-drop*

Comunicação entre Janelas no Windows
Window Messages

- Vista global: janelas, mensagens e procedimentos
  - As mensagens são enviadas a uma janela, pelo SO, ou por aplicações
    - é usada uma API específica (ex: função *SendMessage*)
    - Contêm HWND janela-destino e código de classe de mensagem (o seu significado)
  - As mensagens enviadas são colocadas numa fila global ao sistema (*system queue*)
  - Cada mensagem é remetida para a fila dedicada (*message queue*) da tarefa que criou a janela a quem a mensagem é dirigida (pode gerir mais do que uma janela)
  - Cada tarefa reenvia (i.e., despacha) a mensagem invocando uma rotina de tratamento específica (***window procedure***) associada à janela-destino
    - A recepção de mensagens é assim implícita, através da invocação da *window procedure*
- Tipos de mensagens
  - Sistema: pré-definidas, e globais; utilizadas para controlar o funcionamento das aplicações ou para fornecer informação à aplicação (ex: WM_PAINT, WM_QUIT)
  - Aplicacionais: definidas dinamicamente pelas aplicações, podem ser de âmbito privado à aplicação (ex: WM_USER), ou global ao sistema

Comunicação entre Janelas no Windows
Window Messages (2)

```
BOOL PostMessage(
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam);
```

- Envio e encaminhamento de Mensagens
  - Encaminhamento por filas (*queued messages):* função `PostMessage`
    - mensagens são inseridas no fim da fila privada da tarefa que criou a janela
      - ex: informação de *input* do utilizador (ex: `WM_LBUTTONDBLCLK` p/ botão do rato)
    - certas mensagens são mantidas no fim da fila até que não haja outras
      - Ex: redesenho do interface (WM_PAINT) que permite combinar várias mensagens
    - semântica assíncrona: retorno indica apenas inserção na fila
  - Parâmetros:
    - Identificação da janela destino (hWnd)
      - `HWND` conhecido ou obtido através de `FindWindow, GetParent, EnumWindows`
    - Classe da mensagem a enviar (Msg)
      - Ex: `WM_MOUSEMOVE` para notificar movimento do rato
    - Parâmetros de significado específico a cada classe de mensagem
      - Ex: `wParam` indica se alguma tecla do rato foi premida
      - Ex: `lParam` indica as coordenadas da posição do cursor do rato no ecrã

Comunicação entre Janelas no Windows
Window Messages (3)

```
BOOL SendMessage(
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam);
```

- Envio e encaminhamento de Mensagens (cont.)
  - Envio Directo de mensagens
    - Realizado através da função **`SendMessage`**
    - Utilizada para classes de mensagens de que a aplicação deve ser avisada rapidamente
      - Ex: activação de janelas, gestão do foco da interface, selecção e movimentação de janelas
    - Parâmetros idênticos a **`PostMessage`**
    - Semântica **síncrona**
      - Apenas retorna depois da mensagem ter sido processada pela rotina de tratamento (*window procedure*) da janela-destino
      - Retorno da função indica resultado do processamento
      - Pode causar interblocagem se ocorrer envio directo entre duas janelas, nos dois sentidos

Comunicação entre Janelas no Windows
Window Messages (5)

```
BOOL GetMessage(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax);
```

- Tratamento e Recepção de Mensagens
  - Realizado através de um ciclo executado pela tarefa que gere a interface e que criou a janela. Em cada iteração:
    - É lida/retirada uma mensagem da cabeça da fila (message queue)
    - A mensagem é adaptada caso envolva teclas (TranslateMessage),
    - E agulhada para a rotina de tratamento respectiva (DispatchMessage)
  - Código gerado automaticamente em aplicações baseadas nas MFC
  - Leitura de mensagens recorre a duas funções:
    - **GetMessage**: fica bloqueada enquanto não houver mensagens na fila
    - **PeekMessage**: retorna imediatamente, haja ou não mensagens em espera, e permite a leitura não destrutiva da mensagem (sondagem)
    - A mensagem é devolvida por referência em lpMsg.O retorno das funções é diferente de zero quando é retirada uma mensagem da fila.

Comunicação entre Janelas no Windows
Window Messages (6)

```
BOOL PeekMessage(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg);
```

- Filtragem de Mensagens
  - É possível através dos parâmetros de **`Get / PeekMessage`**
  - Quando são todos nulos/zero, é devolvida a primeira mensagem da fila
  - Quando são especificados valores
    - Indicam que deve ser devolvida a primeira mensagem que satisfizer condições como:
    - especificar a janela destino das mensagens (**`hWnd`**)
      - Ex: janela principal da aplicação
    - restrições dos valores passados como dados adicionais da mensagem (**`wMsgFilterMin`** e **`wMsgFilterMax`**).
      - Ex: coordenadas do rato.

Comunicação entre Janelas no Windows
Clipboard (1)

- ***Clipboard***: zona de memória global ao sistema
  - também podem existir *clipboards* privados de aplicações.
- Comunicação entre janelas sem associação prévia
  - *de-coupled*.
- Permite qualquer número de interlocutores
  - modelo de comunicação muitos-para-muitos
- Documenta os formatos dos dados trocados
  - ex: texto, bitmap, etc.
  - Permite comunicação entre aplicações diferentes
- Orquestrada pelo Utilizador
  - que escolhe as janelas produtoras e consumidoras (*copy/cut* e *paste*)
  - muito diferente do modelo computacional subjacente ao Unix
  - maior flexibilidade na comunicação

Comunicação entre Janelas no Windows
Clipboard (2)

- Mensagens Trocadas:
  - `WM_COPY, WM_CUT, WM_PASTE, WM_CLEAR, EM_UNDO`
  - enviadas às janelas por iniciativa do utilizador
- Formato dos dados
  - suporta diferentes formatos identificados explicitamente
    - pré-definidos (ex: `CF_TEXT`, `CF_BITMAP`)
    - definidos pelas aplicações (nomes geridos como átomos)
- Capacidade de Armazenamento
  - a cada momento o *clipboard* apenas contém um conjunto de dados (*copy* ou *cut*) que pode estar representado em mais do que um formato
  - clipboard tem de ser esvaziado antes de serem colocados novos dados
- Sincronização
  - a cada momento, existe um ***owner*** que pode efectuar escritas ou remoções no *clipboard.*
  - restantes janelas apenas podem efectuar leituras.
  - O ***owner*** é última janela que colocou dados no clipboard ou uma nova que assinalou a intenção de o fazer