

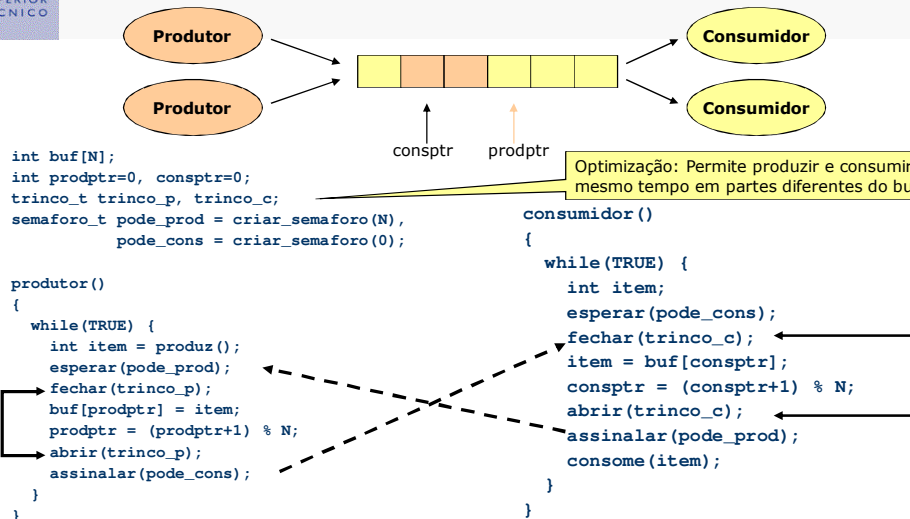
Comunicação entre Processos

Canal de comunicação
Arquitectura da comunicação
Modelos de comunicação

Necessidade da Comunicação

- A sincronização entre processos permitiu que diversas actividades possam cooperar na execução de um algoritmo.
- Contudo, logo se alertou para o facto de em muitas situações a cooperação implicar para além da sincronização a transferência de informação
- A comunicação entre processos ou IPC de InterProcess Communication é um dos aspectos do modelo computacional do sistema operativo que maior importância tem na programação de aplicações

Exemplo de Cooperação entre Processos: Produtor - Consumidor



8/28/2003

José Alves Marques

3

Comunicação entre Processos

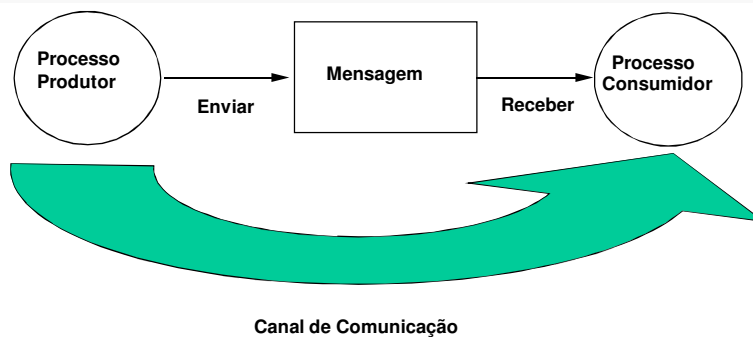
- Para interagirem os processos necessitam de sincronizar-se e de trocar dados
 - Generalização do modelo de interacção entre processos em que para além da sincronização existe transferência de informação
 - A transferência de informação é suportada por um canal de comunicação disponibilizado pelo sistema operativo
- Protocolo e estrutura das mensagens
 - Os processos que comunicam necessitam de estabelecer a estrutura das mensagens trocadas, bem como o protocolo que rege a troca das mesmas
 - Normalmente o sistema operativo considera as mensagens como simples sequências de octetos
 - Numa visão OSI podemos ver estes mecanismo como correspondendo às camadas de Transporte e Sessão

8/28/2003

José Alves Marques

4

Comunicação entre Processos



- generalização do modelo de cooperação entre processos

Exemplos

- A comunicação entre processos pode realiza-se no âmbito:
 - de uma única aplicação,
 - uma máquina
 - Entre máquinas interligadas por uma rede de dados
- Ex.. Outlook e exchange, servidores de base de dados, WWW, FTP, Telnet, SSH, MAIL, P2P

Modelo do canal de comunicação

- Um canal de comunicação é um objecto do sistema operativo que tem pelo menos a seguinte interface funcional
 - Criar e Eliminar
 - Associar - criar uma sessão
 - Enviar (mensagem)
 - Receber (*mensagem)

Características do Canal

- Nomes dos objectos de comunicação
- Tipo de ligação entre o emissor e o receptor
- Estrutura das mensagens
- Capacidade de armazenamento
- Sincronização
 - no envio
 - na recepção
- Segurança – protecção envio/recepção
- Fiabilidade

Ligação

- Antes de usar um canal de comunicação um processo tem de saber se existe e depois indicar ao sistema que se pretende associar
- Este problema decompõe-se em dois
 - Nomes dos canais de comunicação
 - Funções de associação e respectivo controlo de segurança

Nomes dos objectos de comunicação

- Podemos ter duas soluções para os nomes
- Dar nomes explícitos aos canais
 - o espaço de nomes é gerido pelo sistema operativo e pode assumir diversas formas (cadeias de caracteres, números inteiros, endereços estruturados, endereços de transporte das redes)
 - Enviar (IdCanal, mensagem)
 - Receber (IdCanal, *buffer)
 - É o mais frequente e muitas vezes baseia-se na gestão de nomes do sistema de ficheiros
- Os processos terem implicitamente associado um canal de comunicação
 - o canal é implicitamente identificado usando os identificadores dos processos
 - Enviar (IdProcessoConsumidor, mensagem)
 - Receber (IdProcessoProdutor, *buffer)
 - Pouco frequente – ex.: enviar mensagens para janelas em Windows

Ligação – função de associação

- Para usar um canal já existente um processo tem de se lhe associar
- Esta função é muito semelhante ao *open* de um ficheiro
- Tal como no *open* o sistema pode validar os direitos de utilização do processo, ou seja, se o processo pode enviar (escrever) ou receber (ler) mensagens

Sincronização

- Sincronização (envio de mensagem):
 - **assíncrona** – o cliente envia o pedido e continua a execução
 - **síncrona** (rendez-vous) – o cliente fica bloqueado até que o processo servidor leia a mensagem
 - **cliente/servidor** – o cliente fica bloqueado até que o servidor envie uma mensagem de resposta
- Sincronização (recepção de mensagem):
 - bloqueante na ausência de mensagens, a mais frequente
 - Testa se há mensagens e retorna
- Capacidade de Armazenamento de Informação do canal
 - um canal pode ou não ter capacidade para memorizar várias mensagens
 - o armazenamento de mensagens num canal de comunicação permite desacoplar os ritmos de produção e consumo de informação, tornando mais flexível a sincronização

Estrutura da informação trocada

- Fronteiras das mensagens
 - mensagens individualizadas
 - sequência de octetos (*byte stream*, vulgarmente usada nos sistemas de ficheiros e interfaces de E/S)
- Formato
 - Opacas para o sistema - simples sequência de octetos
 - Estruturada - formatação imposta pelo sistema
 - Formatada de acordo com o protocolo das aplicações

Direccionalidade da comunicação

- A comunicação nos canais pode ser unidireccional ou bidireccional
 - Unidireccional o canal apenas permite enviar informação num sentido que fica definido na sua criação
 - Normalmente neste tipo de canais são criados dois para permitir a comunicação bidireccional. Ex.: *pipes*
 - Bidireccional o canal permite enviar mensagens nos dois sentidos
 - Ex.: *sockets*

Resumo do Modelo Computacional

- IDCanal = **CriarCanal** (Nome, Dimensão)
- IDCanal = **AssociarCanal** (Nome, Modo)
- **EliminarCanal** (IDCanal)
- **Enviar** (IDCanal, Mensagem, Tamanho)
- **Receber** (IDCanal, buffer, TamanhoMax)

Implementação do Canal de Comunicação

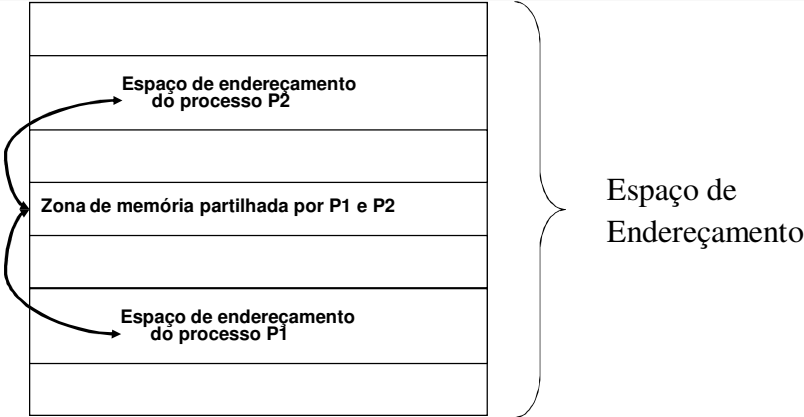
- Para implementar o canal de comunicação é fundamental definir como são transferidos os dados entre os espaços de endereçamento dos processos
- O canal de comunicação pode ser implementado com dois tipos de mecanismos:
 - **Memória partilhada**: os processos acedem a uma zona de memória que faz parte do espaço de endereçamento dos processos comunicantes
 - **Transferidos através do núcleo do sistema operativo**; os dados são sempre copiados para o núcleo antes de serem transferidos



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

Arquitectura da Comunicação: memória partilhada



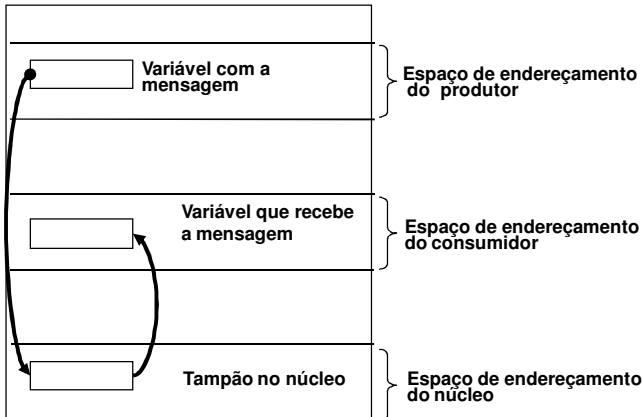
8/28/2003
José Alves Marques
17



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

Arquitectura da Comunicação: cópia através do núcleo



8/28/2003
José Alves Marques
18

Comparação

- A memória partilhada pode ser considerada como uma extensão à gestão da memória virtual, permitindo ultrapassar os mecanismos de protecção dos espaços de endereçamento
 - Num sistema paginado corresponde a acrescentar uma tabela de páginas que descreve a região de memória partilhada
- A comunicação no núcleo é semelhante a gestão de outros objectos do sistema em particular os ficheiros
 - Na execução da chamada sistema `Enviar` a mensagem é copiada para o núcleo e na chamada `Receber` é copiada do núcleo para o espaço de endereçamento do processo.

8/28/2003

José Alves Marques

19

Memória Partilhada

- `Apont = CriarRegião (Nome, Tamanho)`
- `Apont = AssociarRegião (Nome)`
- `EliminarRegião (Nome)`

São necessários mecanismos de sincronização para:

- Garantir exclusão mútua sobre a zona partilhada
- Sincronizar a cooperação dos processos produtor e consumidor (ex. produtor-consumidor ou leitores-escritores)

8/28/2003

José Alves Marques

20

Objecto de Comunicação do Sistema

- `IdCanal = CriarCanal(Nome)`
- `IdCanal = AssociarCanal (Nome)`
- `EliminarCanal (IdCanal)`
- `Enviar (IdCanal, Mensagem, Tamanho)`
- `Receber (IdCanal, *Buffer, TamanhoMax)`

Não são necessários mecanismos de sincronização adicionais porque são implementados pelo núcleo do sistema operativo

Comparação: memória partilhada vs. cópia através do núcleo

- | | |
|--|--|
| <ul style="list-style-type: none">• Memória partilhada:<ul style="list-style-type: none">– mecanismo mais eficiente– a sincronização tem de ser explicitamente programada– programação complexa | <ul style="list-style-type: none">• Objecto de Comunicação Sistema:<ul style="list-style-type: none">– velocidade de transferência limitada pelas duas cópias da informação e pelo uso das chamadas sistema para Enviar e Receber– sincronização implícita– fácil de utilizar |
|--|--|



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

Modelos de Comunicação

8/28/2003José Alves Marques23



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

Modelos de Comunicação

- Com as funções do modelo computacional poderíamos criar qualquer tipo de estrutura de comunicação entre os processos.
- Contudo existem algumas que, por serem mais frequentes, correspondem a padrões que os programadores utilizam ou que o sistema operativo oferece directamente como canais nativos

8/28/2003José Alves Marques24

Modelos de Comunicação

- **Um-para-Um (fixo)- Mestre/escravo:**
 - O processo consumidor (escravo) tem a sua acção totalmente controlada por um processo produtor (mestre)
 - A ligação entre produtor consumidor é fixa
- **Um-para-Muitos - Difusão:**
 - Envio da mesma informação a um conjunto de processos consumidores
- **Muitos-para-Um (caixa de correio, canal sem ligação):**
 - Transferência assíncrona de informação (mensagens), de vários processos produtores, para um canal de comunicação associado a um processo consumidor
 - Os produtores não têm qualquer controlo sobre os consumidores/receptores
- **Um-para-Um de vários (diálogo, canal com ligação):**
 - Um processo pretende interactivar com outro, negociam o estabelecimento de um canal dedicado, mas temporário, de comunicação entre ambos. Situação típica de cliente servidor
- **Muitos-para-Muitos**
 - Transferência assíncrona de informação (mensagens) de vários processos produtores para um canal de comunicação associado a múltiplos processos consumidor

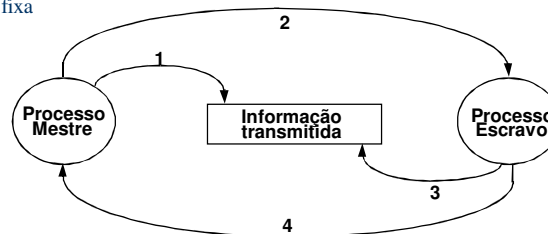
8/28/2003

José Alves Marques

25

Comunicação Mestre-Escravo

- o mestre não necessita de autorização para utilizar o escravo
- a actividade do processo escravo é controlada pelo processo mestre
- a ligação entre emissor e receptor é fixa



- **Etapas:**
 - 1 - informação para o processo escravo
 - 2 - assinalar ao escravo a existência de informação para tratar
 - 3 - leitura e eventualmente escrita de informação para o processo mestre
 - 4 - assinalar ao mestre o final da operação

8/28/2003

José Alves Marques

26

Mestre Escravo com Memória Partilhada

```

#define DIMENSAO 1024
char* adr;
int Mest, Esc;
semaforo SemEscravo, SemMestre;

main() {
    SemEscravo = CriarSemaforo(0);
    SemMestre = CriarSemaforo(0);
    Mest = CriarProcesso(Mestre);
    Esc = CriarProcesso(Escravo);
}
  
```

```

void Mestre () {
    adr = CriarRegiao ("MemPar", DIM);
    for (; ;) {
        ProduzirInformação();
        EscreverInformação();
        Assinalar (SemEscravo);
        /* Outras acções */
        Esperar (SemMestre);
    }
}

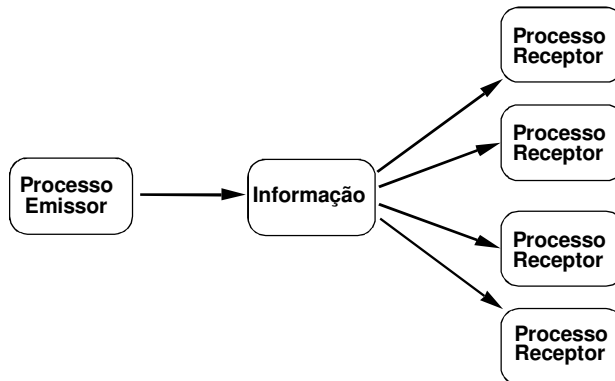
void Escravo() {
    adr = AssociarRegiao ("MemPar", DIM);
    for (; ;) {
        Esperar (SemEscravo);
        TratarInformação();
        Assinalar (SemMestre);
    }
}
  
```

8/28/2003

José Alves Marques

27

Difusão da Informação



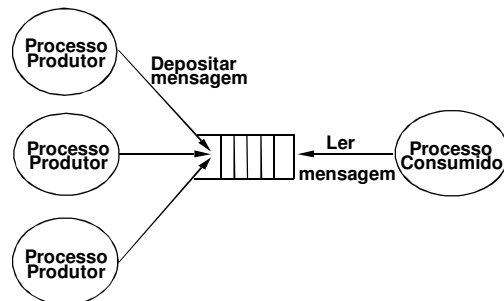
8/28/2003

José Alves Marques

28

Correio (canal sem ligação)

- os processos emissores não controlam directamente a actividade do receptor ou receptores
- a ligação efectua-se indirectamente através das caixas de correio não existe uma ligação directa entre os processos
- a caixa de correio permite memorizar as mensagens quando estas são produzidas mais rapidamente do que consumidas



```

idCC = CriarCCorreio (Nome, parametros)
idCC = AssociarCCorreio (Nome, modo)
EliminarCCorreio (Nome)
  
```

8/28/2003

José Alves Marques

29

Programação da Caixa de Correio

cliente

```

#define NMax 10

typedef char TMensagem[NMax];
IdCC CCliente, CServidor;
TMensagem Mens;

/* Cria a mensagem de pedido do serviço o qual
contém o identificador da caixa de correio de resposta */
void PreencheMensagem(TMensagem MS) { }

/* Processa a mensagem de resposta */
void ProcessaResposta(TMensagem MS) { }

void main() {
  CCliente=CriarCorreio("Cliente");
  CServidor=AssociarCorreio("Servidor");

  for (;;) {
    PreencheMensagem (Mens);
    Enviar (CServidor, Mens);
    Receber (CCliente, Mens);
    ProcessaResposta (Mens);
  }
}
  
```

servidor

```

#define NMAX 10
#define NCNome 64

typedef char TMensagem[NMAX];
typedef char Nome[NCNome];
IdCC CResposta, CServidor;
TMensagem Mens;
Nome NomeCliente;

/* Trata a mensagem e devolve o nome da caixa de
correio do cliente enviada na mensagem inicial */
void TrataMensagem (TMensagem Ms, Nome NomeCliente) { }

void main () {
  CServidor=CriarCorreio("Servidor");

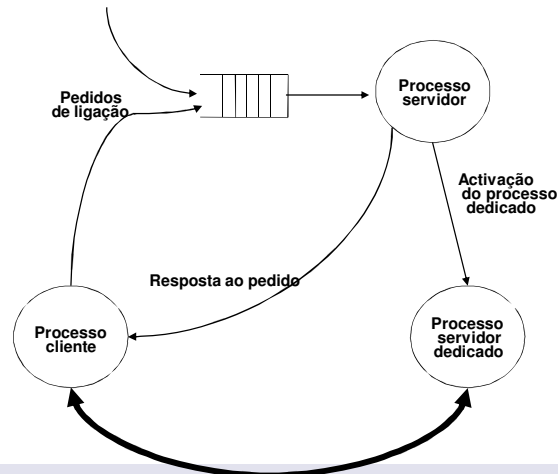
  for (;;) {
    Receber (CServidor, Mens);
    TrataMensagem (Mens, NomeCliente);
    CResposta=AssociarCCorreio (NomeCliente);
    Enviar (CResposta, Mens);
    EliminarCC(CResposta);
  }
}
  
```

8/28/2003

José Alves Marques

Canal com ligação Modelo de Diálogo

- É estabelecido um canal de comunicação entre o processo cliente e o servidor
- O servidor pode gerir múltiplos clientes, mas dedica a cada um deles uma actividade independente
- O servidor pode ter uma política própria para atender os clientes



8/28/2003

José Alves Marques

Canal de diálogo

31

Diálogo

Servidor

- Primitiva para Criação de Canal
`IdCanal = CriarCanal (Nome);`
- Primitivas para Aceitar/Desligar/Eliminar Ligações
`IdCanal= AceitarLigacao (IdCanServidor);`
`Desligar (IdCanal);`
`Eliminar (Nome);`

Cliente

- Primitivas par Associar/Desligar ao Canal
`IdCanal:= PedirLigacao (Nome);`
`Desligar (IdCanal);`

8/28/2003

José Alves Marques

32

Modelo de Diálogo - Canal com ligação

Cliente

```
IdCanal Canal;
int Ligado;

void main() {
    while (TRUE) {
        Canal=PedirLigacao("Servidor");
        Ligado = TRUE;

        while (Ligado) {
            ProduzInformacao(Mens);
            Enviar(Canal, Mens);
            Receber(Canal, Mens);
            TratarInformacao(Mens);
        }
        TerminarLigacao(Canal);
    }
    exit(0);
}
```

Servidor

```
IdCanal CanalServidor, CanalDialogo;

void main() {
    CanalPedido=CriarCanal("Servidor");

    for (;;) {
        CanalDialogo=AceitarLigacao(CanalPedido);
        CriarProcesso(TrataServico, CanalDialogo);
    }
}
```

8/28/2003

José Alves Marques

Unix– Modelo Computacional - IPC

pipes
sockets
IPC sistema V

8/28/2003

José Alves Marques

34

Mecanismos de Comunicação em Unix

- No Unix houve uma tentativa de uniformização da interface de comunicação entre processos com a interface dos sistemas de ficheiros.
- Para perceber os mecanismos de comunicação é fundamental conhecer bem a interface com o sistema de ficheiros.

Sistema de Ficheiros

- Sistema de ficheiros hierarquizado
- Tipos de ficheiros:
 - Normais – sequência de octetos (bytes) sem uma organização em registos (records)
 - Ficheiros especiais – periféricos de E/S, pipes, sockets
 - Ficheiros directório
- Quando um processo se começa a executar o sistema abre três ficheiros especiais
 - `stdin` – input para o processo (fd – 0)
 - `stdout` – Output para o processo (fd – 1)
 - `stderr` – periférico para assinalar os erros (fd – 2)
- Um file descriptor é um inteiro usado para identificar um ficheiro aberto (os valores variam de zero até máximo dependente do sistema)

Sistema de Ficheiros

```

main (argc, argv)
int argc;
char *argv[];
{
    int origem, destino, n;
    char tampao[1024];

    origem = open (argv[1], O_RDONLY);
    if (origem == -1) {
        printf ("Nao consigo abrir %s \n", argv[1]);
        exit(1);
    }
    destino = creat (argv[2], 0666);
    if (destino == -1) {
        printf ("Nao consigo criar %s \n", argv[2]);
        exit(1);
    }
    while ((n = read (origem, tampao, sizeof(tampao))) > 0)
        write (destino, tampao, n);
    exit(0);
}
  
```

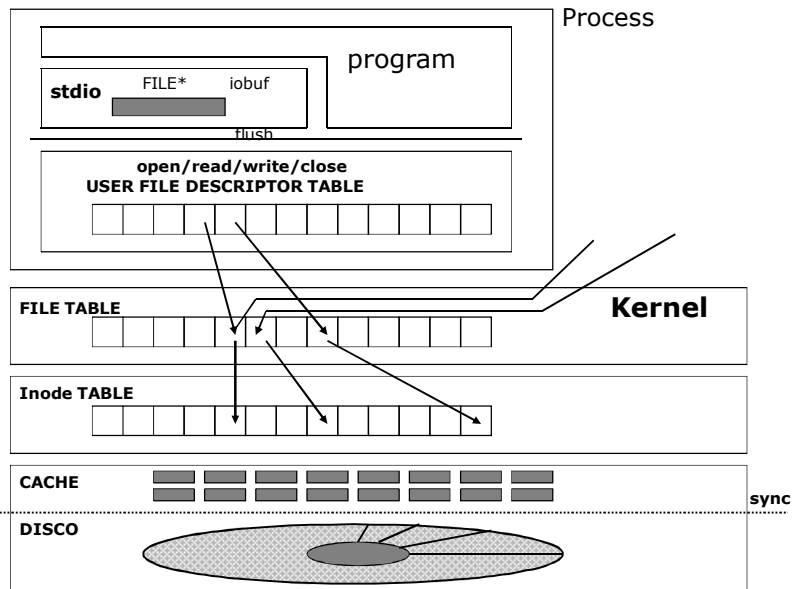
8/28/2003

José Alves Marques

37

Sistema de Ficheiros UNIX

Process



IPC no UNIX

- Mecanismo inicial:
 - pipes
- Extensão dos pipes:
 - pipes com nome
- Evolução do Unix BSD 4.2:
 - sockets
- Unix sistema V:
 - regiões de memória partilhada
 - semáforos
 - caixas de correio

Bibliografia:

- *Unix Network Programming*, Stevens, (Caps. 3 e 6)
- Fundamentos de Sistemas Operativos

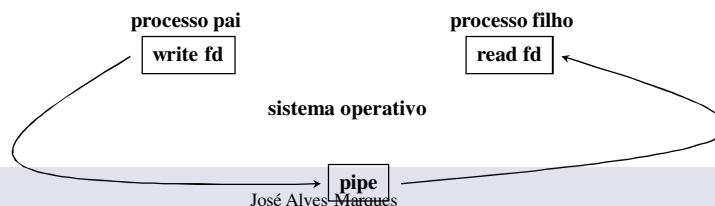
8/28/2003

José Alves Marques

39

Pipes

- Mecanismo original do Unix para comunicação entre processos.
- Têm uma interface idêntica à dos ficheiros
- Constitui um dos conceitos unificadores na estrutura do interpretador de comandos
- Canal (*byte stream*) ligando dois processos
- Permite um fluxo de informação unidireccional, um processo escreve num pipe e o correspondente lê na outra extremidade – modelo um para um
- Não tem nome lógico associado
- As mensagens são sequências de bytes de qualquer dimensão



8/28/2003

José Alves Marques

40

Pipes (2)

```
int pipe (int *fds);
```

fds[0] - descritor aberto para leitura
fds[1] - descritor aberto para escrita

- Os descritores de um pipe são análogos ao dos ficheiros
- As operações de read e write sobre ficheiros são válidas para os pipes
- Os descritores são locais a um processo podem ser transmitidos para os processos filhos através do mecanismo de herança
- O processo fica bloqueado quando escreve num pipe cheio
- O processo fica bloqueado quando lê de um pipe vazio

Pipes (3)

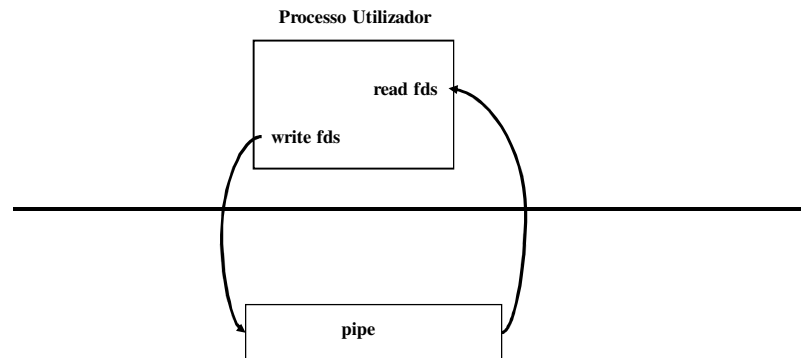
```
char msg[] = "utilizacao de pipes";

main() {
    char tampao[1024];
    int fds[2];

    pipe(fds);

    for (;;) {
        write (fds[1], msg, sizeof (msg));
        read (fds[0], tampao, sizeof (msg));
    }
}
```

Pipes (4)



8/28/2003

José Alves Marques

43

Comunicação pai-filho

```

#include <stdio.h>
#include <fcntl.h>

#define TMSG 100
char msg[] = "mensagem de teste";
char tmp[TMSG];

main() {
    int fds[2], pid_filho;

    if (pipe (fds) < 0) exit(-1);
    if (fork () == 0) {
        /* lê do pipe */
        read (fds[0], tmp, sizeof (msg));
        printf ("%s\n", tmp);
        exit (0);
    }
  
```

```

    }
    else {
        /* processo pai */
        /* escreve no pipe */
        write (fds[1], msg, sizeof (msg));
        pid_filho = wait();
    }
  
```

8/28/2003

José Alves Marques

44

Redirecção de Entradas/saídas

8/28/2003

José Alves Marques

45

DUP – System Call

NAME

dup - duplicate an open file descriptor

SYNOPSIS

```
#include <unistd.h>  
int dup(int fildes);
```

DESCRIPTION

The dup() function returns a new file descriptor having the following in common with the original open file descriptor fildes:

- same open file (or pipe)
- same file pointer (that is, both file descriptors share one file pointer)
- same access mode (read, write or read/write)

The new file descriptor is set to remain open across exec functions (see fcntl(2)).

The file descriptor returned is the lowest one available.

The dup(fildes) function call is equivalent to: fcntl(fildes, F_DUPFD, 0)

8/28/2003

José Alves Marques

46

Redireccionamento de Entradas/Saídas

```

#include <stdio.h>
#include <fcntl.h>

#define TMSG 100
char msg[] = "mensagem de teste";
char tmp[TMSG];

main() {
    int fds[2], pid_filho;

    if (pipe (fds) < 0) exit(-1);
    if (fork () == 0) {
        /* processo filho */
        /* liberta o stdin (posição zero) */
        close (0);

        /* redirecciona o stdin para o pipe de
        leitura */
        dup (fds[0]);
    }

```

```

    /* fecha os descritores não usados pelo
    filho */
    close (fds[0]);
    close (fds[1]);

    /* lê do pipe */
    read (0, tmp, sizeof (msg));
    printf ("%s\n", tmp);
    exit (0);
}
else {
    /* processo pai */
    /* escreve no pipe */
    write (fds[1], msg, sizeof (msg));
    pid_filho = wait();
}
}

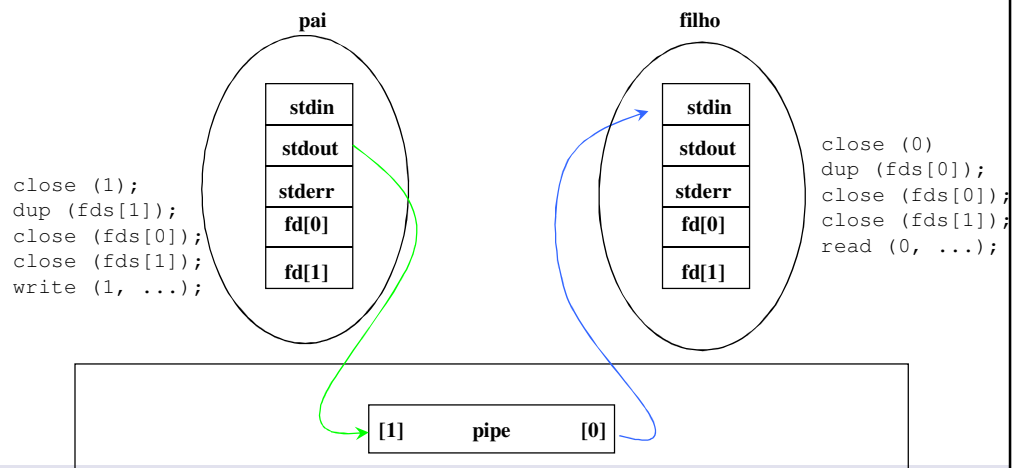
```

8/28/2003

José Alves Marques

47

Redireccionamento de Entradas/Saídas (2)



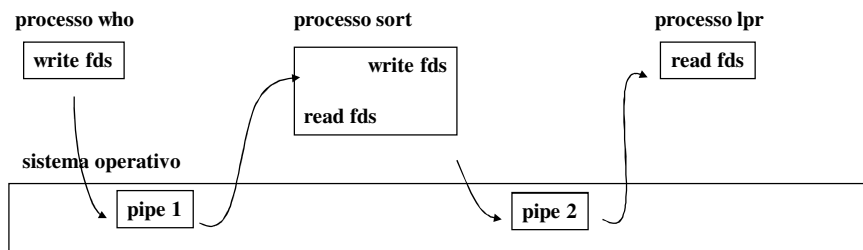
8/28/2003

José Alves Marques

48

Redireccionamento de Entradas/Saídas no Shell

exemplo:
who | sort | lpr



8/28/2003

José Alves Marques

49

popen

NAME

popen, pclose - initiate a pipe to or from a process

SYNOPSIS

```
#include <stdio.h>
FILE *popen(const char *command, const char *mode);
int pclose(FILE *stream);
```

DESCRIPTION

The `popen()` function creates a pipe between the calling program and the command to be executed. The arguments to `popen()` are pointers to null-terminated strings. The `command` argument consists of a shell command line.

The `mode` argument is an I/O mode, either `r` for reading or `w` for writing. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is `w`, by writing to the file stream (see `intro(3)`); and one can read from the standard output of the command, if the I/O mode is `r`, by reading from the file stream. Because open files are shared, a type `r` command may be used as an input filter and a type `w` as an output filter.

The environment of the executed command will be as if a child process were created within the `popen()` call using `fork()`. The child is invoked with the call: `execl("/usr/bin/ksh", "ksh", "-c", command, (char *)0)`; otherwise, the child is invoked with the call:

A stream opened by `popen()` should be closed by `pclose()`, which closes the pipe, and waits for the associated process to terminate and returns the termination status of the process running the command language interpreter.

8/28/2003

José Alves Marques

50

popen (1)

EXAMPLE

The following program will print on the standard output (see `stdio(3S)`) the names of files in the current directory with a `.c` suffix.

```
#include <stdio.h>
#include <stdlib.h>
main() {
    char *cmd = "/usr/bin/ls *.c";
    char buf[BUFSIZ];
    FILE *ptr;

    if ((ptr = popen(cmd, "r")) != NULL)
        while (fgets(buf, BUFSIZ, ptr) != NULL)
            printf("%s", buf);
    return 0;
}
```

8/28/2003

José Alves Marques

51

popen (2)

```
#include <stdio.h>

#define READ 0
#define WRITE 1
#define tst (a,b) (mode == READ ? (b) : (a))
static int popen_pid;
```

```
FILE *popen (char* cmd, int mode) {
    int p[2];
    if (pipe(p) < 0) return (NULL);
```

```
    if ((popen_pid = fork()) == 0) {
        close (tst (p[WRITE], p[READ]));
        close (tst (0, 1));
        dup (tst (p[READ], p[WRITE]));
        close (tst (p[READ], p[WRITE]));
        execl ("/bin/sh", "sh", "-c", cmd, 0);
        exit (1);
    }
```

```
    if (popen_pid == -1) return (NULL);
    close (tst (p[READ], p[WRITE]));
    return (tst (p[WRITE], p[READ]));
}
```

mode = READ
mode = WRITE

o filho lê o que
o pai envia

o pai envia
para o filho
os argumentos
do comando

8/28/2003

José Alves Marques

52

Named Pipes ou FIFO

- Para dois processos (que não sejam pai e filho) comunicarem é preciso que o pipe seja identificado por um nome
- Atribui-se um nome lógico ao pipe. **O espaço de nomes usado é o do sistema de ficheiros**
- Um named pipe comporta-se externamente como um ficheiro, existindo uma entrada na directoria correspondente
- Um named pipe pode ser aberto por processos que não têm qualquer relação hierárquica

8/28/2003

José Alves Marques

54

Named Pipes

- um named pipe é um canal :
 - unidireccional
 - interface sequência de caracteres (byte stream)
 - um processo associa-se com a função open
 - é eliminado com a função unlink
 - o envio de informação é efectuado com a função write
 - a leitura da informação é efectuada com a função read
- A função mknod ou mkfifo permite criar ficheiros com características especiais e serve para criação dos named pipes.
`int mknod (char *pathname, int mode)`

8/28/2003

José Alves Marques

55

<pre> /* Cliente */ #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> #define TAMMSG 1000 void produzMsg (char *buf) { strcpy (buf, "Mensagem de teste"); } void trataMsg (buf) { printf ("Recebeu: %s\n", buf); } main() { int fcli, fserv; char buf[TAMMSG]; if ((fserv = open ("/tmp/servidor", O_WRONLY)) < 0) exit (-1); if ((fcli = open ("/tmp/cliente", O_RDONLY)) < 0) exit (-1); produzMsg (buf); write (fserv, buf, TAMMSG); read (fcli, buf, TAMMSG); trataMsg (buf); close (fserv); close (fcli); } </pre>	Departamento de Engenharia Informática <pre> /* Servidor */ #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> #define TAMMSG 1000 main () { int fcli, fserv, n; char buf[TAMMSG]; unlink("/tmp/servidor"); unlink("/tmp/cliente"); if (mkfifo ("/tmp/servidor", 0777) < 0) exit (-1); if (mkfifo ("/tmp/cliente", 0777) < 0) exit (-1); if ((fserv = open ("/tmp/servidor", O_RDONLY)) < 0) exit (-1); if ((fcli = open ("/tmp/cliente", O_WRONLY)) < 0) exit (-1); for (;;) { n = read (fserv, buf, TAMMSG); if (n <= 0) break; trataPedido (buf); n = write (fcli, buf, TAMMSG); } close (fserv); close (fcli); unlink("/tmp/servidor"); unlink("/tmp/cliente"); } </pre>
---	--

 INSTITUTO SUPERIOR TÉCNICO	Departamento de Engenharia Informática
Sockets	
• Interface de programação para comunicação entre processos introduzida no Unix 4.2 BSD • Objectivos: – independente dos protocolos – transparente em relação à localização dos processos – compatível com o modelo de E/S do Unix – eficiente	
8/28/2003	57

Domínio e Tipo de Sockets

- Domínio do socket - define a família de protocolos associada a um socket:
 - Internet: família de protocolos Internet
 - Unix: comunicação entre processos da mesma máquina
 - outros...
- Tipo do socket - define as características do canal de comunicação:
 - stream: canal com ligação, bidireccional, fiável, interface tipo sequência de octetos
 - datagram: canal sem ligação, bidireccional, não fiável, interface tipo mensagem
 - raw: permite o acesso directo aos níveis inferiores dos protocolos (ex: IP na família Internet)

8/28/2003

José Alves Marques

58

Domínio e Tipo de Sockets (2)

- Relação entre domínio, tipo de socket e protocolo:

tipo \ domínio	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	SIM	TCP	SPP
SOCK_DGRAM	SIM	UDP	IDP
SOCK_RAW	-	IP	SIM
SOCK_SEQPACKET	-	-	SPP

8/28/2003

José Alves Marques

59

Interface Sockets: definição dos endereços

```
/* ficheiro <sys/socket.h> */
struct sockaddr {
    u_short family; /* definição do domínio (AF_XX) */
    char sa_data[14]; /* endereço específico do domínio */
};

/* ficheiro <sys/un.h> */
struct sockaddr_un {
    u_short family; /* definição do domínio (AF_UNIX) */
    char sun_path[108]; /* nome */
};
```

```
/* ficheiro <netinet/in.h> */
struct in_addr {
    u_long addr; /* Netid+Hostid */
};

struct sockaddr_in {
    u_short sin_family; /* AF_INET */
    u_short sin_port; /* número do porto - 16 bits */
    struct in_addr sin_addr; /* Netid+Hostid */
    char sin_zero[8]; /* não utilizado */
};
```

struct sockaddr_un

family
pathname (up to 108 bytes)

struct sockaddr_in

family
2-byte port
4-byte net ID, host ID
(unused)

8/28/2003

José Alves Marques

60

Interface Sockets: criação de um socket e associação de um nome

- Criação de um socket:

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int dominio, int tipo, int protocolo);
```

- domínio: AF_UNIX, AF_INET
- tipo: SOCK_STREAM, SOCK_DGRAM
- protocolo: normalmente escolhido por omissão
- resultado: identificador do socket (sockfd)

- Um socket é criado sem nome

- A associação de um nome (endereço de comunicação) a um socket já criado é feito com a chamada bind:

```
int bind(int sockfd, struct sockaddr *nome, int dim)
```

8/28/2003

José Alves Marques

61

Sockets com e sem Ligação

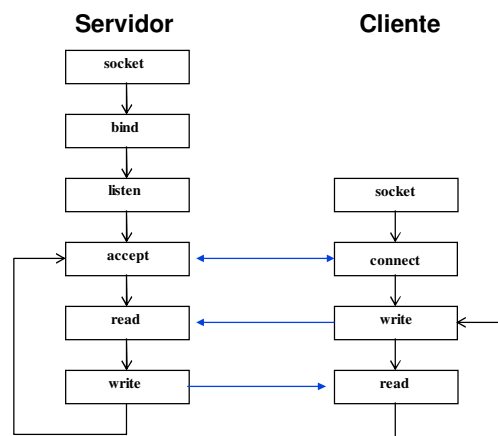
- Sockets com ligação:
 - Modelo de comunicação tipo diálogo
 - Canal com ligação, bidireccional, fiável, interface tipo sequência de octetos
- Sockets sem ligação:
 - Modelo de comunicação tipo correio
 - Canal sem ligação, bidireccional, não fiável, interface tipo mensagem

8/28/2003

José Alves Marques

62

Sockets com Ligação



8/28/2003

José Alves Marques

63

Sockets com Ligação

- **listen** - indica que se vão receber ligações neste socket:
 - `int listen (int sockfd, int maxpendentes)`
- **accept** - aceita uma ligação:
 - espera pelo pedido de ligação
 - cria um novo socket
 - devolve:
 - identificador do novo socket
 - endereço do interlocutor
 - `int accept(int sockfd, struct sockaddr *nome, int *dim)`
- **connect** - estabelece uma ligação com o interlocutor cujo endereço é nome:
 - `int connect (int sockfd, struct sockaddr *nome, int dim)`

8/28/2003

José Alves Marques

64

unix.h e inet.h

unix.h

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define UNIXSTR_PATH
"/tmp/s.unixstr"
#define UNIXDG_PATH
"/tmp/s.unixdgx"
#define UNIXDG_TMP
"/tmp/dgXXXXXXX"
```

inet.h

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

#define SERV_UDP_PORT 6600
#define SERV_TCP_PORT 6601
#define SERV_HOST_ADDR "193.136.128.20"
/* endereço do servidor */
#define SERV_HOSTNAME "mega"
/* nome do servidor */
```

8/28/2003

José Alves Marques

65

Exemplo

- Servidor de eco
- Sockets no domínio Unix
- Sockets com ligação

Cliente STREAM AF_UNIX

```
/* Cliente do tipo socket stream.
#include "unix.h"
main(void) {
    int sockfd, servlen;
    struct sockaddr_un serv_addr;

    /* Cria socket stream */
    if ((sockfd= socket(AF_UNIX, SOCK_STREAM, 0) ) < 0)
        err_dump("client: can't open stream socket");

    /* Primeiro uma limpeza preventiva */
    bzero((char *) &serv_addr, sizeof(serv_addr));

    /* Dados para o socket stream: tipo + nome que
    identifica o servidor */
    serv_addr.sun_family = AF_UNIX;
    strcpy(serv_addr.sun_path, UNIXSTR_PATH);
    servlen = strlen(serv_addr.sun_path) +
    sizeof(serv_addr.sun_family);
```

Cliente STREAM AF_UNIX(2)

```

/* Estabelece uma ligação. Só funciona se o socket
   tiver sido criado e o nome associado*/

   if(connect(sockfd, (struct sockaddr *) &serv_addr,
             servlen) < 0)
       err_dump("client: can't connect to server");

/* Envia as linhas lidas do teclado para o socket
*/
   str_cli(stdin, sockfd);

/* Fecha o socket e termina */
   close(sockfd);
   exit(0);
}

```

8/28/2003

José Alves Marques

68

Cliente STREAM AF_UNIX (3)

```

#include <stdio.h>
#define MAXLINE 512

/*Lê string de fp e envia para
sockfd. Lê string de sockfd e envia
para stdout*/

str_cli(fp, sockfd)
FILE *fp;
int sockfd;
{
    int n;
    char sendline[MAXLINE],
    recvline[MAXLINE+1];

    while(fgets(sendline, MAXLINE, fp)
        != NULL) {

```

```

/* Envia string para sockfd.
   Note-se que o \0 não é enviado */
    n = strlen(sendline);
    if (writen(sockfd, sendline, n) != n)
        err_dump("str_cli:writen error on socket");

```

```

/* Tenta ler string de sockfd.
   Note-se que tem de terminar a string com \0 */
    n = readline(sockfd, recvline, MAXLINE);
    if (n<0) err_dump("str_cli:readline error");
    recvline[n] = 0;

```

```

/* Envia a string para stdout */
    fputs(recvline, stdout);
}
if (ferror(fp))
    err_dump("str_cli: error reading file");
}

```

8/28/2003

José Alves Marques

69

Servidor STREAM AF_UNIX

```

/* Recebe linhas do cliente e reenvia-as para o cliente */
#include "unix.h"

main(void) {
    int sockfd, newsockfd, clilen, childpid, servlen;
    struct sockaddr_un cli_addr, serv_addr;

    /* Cria socket stream */
    if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
        err_dump("server: can't open stream socket");

    /* Elimina o nome, para o caso de já existir.
    /* O nome serve para que os clientes possam identificar o servidor */
    unlink(UNIXSTR_PATH);
    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sun_family = AF_UNIX;
    strcpy(serv_addr.sun_path, UNIXSTR_PATH);
    servlen = strlen(serv_addr.sun_path) + sizeof(serv_addr.sun_family);
    if (bind(sockfd, (struct sockaddr *) &serv_addr, servlen) < 0)
        err_dump("server, can't bind local address");

    listen(sockfd, 5);

```

8/28/2003

José Alves Marques

70

Servidor STREAM AF_UNIX (2)

```

for (;;) {
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
    &clilen);
    if (newsockfd < 0) err_dump("server: accept error");

    /* Lança processo filho para tratar do cliente */
    if ((childpid = fork()) < 0) err_dump("server: fork error");
    else if (childpid == 0) {
        /* Processo filho.
        Fecha sockfd já que não é utilizado pelo processo filho
        Os dados recebidos do cliente são reenviados para o cliente
        */
        close(sockfd);
        str_echo(newsockfd);
        exit(0);
    }

    /* Processo pai. Fecha newsockfd que não utiliza */
    close(newsockfd);
}

```

8/28/2003

José Alves Marques

71

Servidor STREAM AF_UNIX (3)

```

#define MAXLINE 512
/* Servidor do tipo socket stream. Reenvia as linhas recebidas para o cliente*/

str_echo(int sockfd)
{
    int n;
    char line[MAXLINE];

    for (;;) {
        /* Lê uma linha do socket */
        n = readline(sockfd, line, MAXLINE);
        if (n == 0) return;
        else if (n < 0) err_dump("str_echo: readline error");

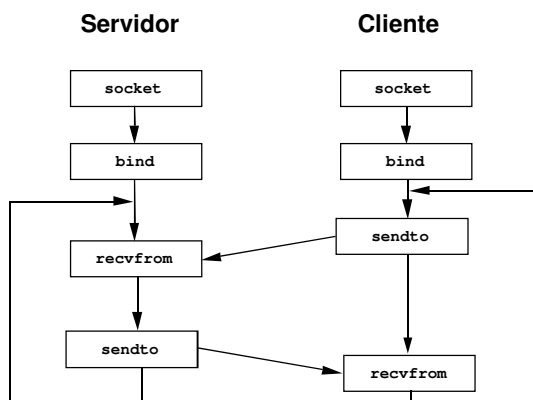
        /* Reenvia a linha para o socket. n conta com o \0 da string,
        caso contrário perdia-se sempre um caracter! */
        if (writen(sockfd, line, n) != n)
            err_dump("str_echo: writen error");
    }
}
  
```

8/28/2003

José Alves Marques

72

Sockets sem Ligação



8/28/2003

José Alves Marques

73

Sockets sem Ligação

- **sendto**: Envia uma mensagem para o endereço especificado

```
int sendto(int sockfd, char *mens, int dmens,  
           int flag, struct sockaddr *dest, int *dim)
```

- **recvfrom**: Recebe uma mensagem e devolve o endereço do emissor

```
int recvfrom(int sockfd, char *mens, int dmens,  
             int flag, struct sockaddr *orig, int *dim)
```

8/28/2003

José Alves Marques

74

Cliente DGRAM AF_UNIX

```
#include "unix.h"  
main(void) {  
    int sockfd, clilen, servlen;  
    char *mktemp();  
    struct sockaddr_un cli_addr, serv_addr;  
  
    /* Cria socket datagram */  
    if(( sockfd = socket(AF_UNIX, SOCK_DGRAM, 0) ) < 0)  
        err_dump("client: can't open datagram socket");  
    /* O nome temporário serve para ter um socket para resposta do  
    servidor */  
    bzero((char *) &cli_addr, sizeof(cli_addr));  
    cli_addr.sun_family = AF_UNIX;  
    mktemp(cli_addr.sun_path);  
    clilen = sizeof(cli_addr.sun_family) + strlen(cli_addr.sun_path);  
  
    /* Associa o socket ao nome temporário */  
    if (bind(sockfd, (struct sockaddr *) &cli_addr, clilen) < 0)  
        err_dump("client: can't bind local address");  
}
```

8/28/2003

José Alves Marques

75

Cliente DGRAM AF_UNIX(2)

```
/* Primeiro uma limpeza preventiva!
bzero((char *) &serv_addr, sizeof(serv_addr));

serv_addr.sun_family = AF_UNIX;
strcpy(serv_addr.sun_path, UNIXDG_PATH);
servlen=sizeof(serv_addr.sun_family) +
        strlen(serv_addr.sun_path);

/* Lê linha do stdin e envia para o servidor. Recebe a linha do
servido e envia-a para stdout */
dg_cli(stdin, sockfd, (struct sockaddr *) &serv_addr, servlen);

close(sockfd);
unlink(cli_addr.sun_path);
exit(0);
}
```

8/28/2003

José Alves Marques

76

Cliente DGRAM AF_UNIX (3)

```
#include <stdio.h>
#define MAXLINE 512

/* Cliente do tipo socket datagram.
Lê string de fp e envia para sockfd.
Lê string de sockfd e envia para stdout */

#include <sys/types.h>
#include <sys/socket.h>
```

8/28/2003

José Alves Marques

77

Cliente DGRAM AF_UNIX (4)

```
while (fgets(sendline, MAXLINE, fp) != NULL) {
    n = strlen(sendline);

    /* Envia string para sockfd. Note-se que o \0 não é enviado */
    if (sendto(sockfd, sendline, n, 0, pserv_addr, servlen) != n)
        err_dump("dg_cli: sendto error on socket");

    /* Tenta ler string de sockfd. Note-se que tem de
       terminar a string com \0 */
    n = recvfrom(sockfd, recvline, MAXLINE, 0,
        (struct sockaddr *) 0, (int *) 0);
    if (n < 0) err_dump("dg_cli: recvfrom error");
    recvline[n] = 0;

    /* Envia a string para stdout */
    fputs(recvline, stdout);
}
if (ferror(fp)) err_dump("dg_cli: error reading file");
}
```

8/28/2003

José Alves Marques

78

Servidor DGRAM AF_UNIX

```
/* Servidor do tipo socket datagram. Recebe linhas do cliente e devolve-as para o
   cliente */
#include "unix.h"
main (void) {
    int sockfd, servlen;
    struct sockaddr_un serv_addr, cli_addr;

    /* Cria socket datagram */
    if ((sockfd = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0)
        err_dump("server: can't open datagram socket");

    unlink(UNIXDG_PATH);
    /* Limpeza preventiva */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sun_family = AF_UNIX;
    strcpy(serv_addr.sun_path, UNIXDG_PATH);
    servlen = sizeof(serv_addr.sun_family) + strlen(serv_addr.sun_path);
    /* Associa o socket ao nome */
    if (bind(sockfd, (struct sockaddr *) &serv_addr, servlen) < 0)
        err_dump("server: can't bind local address");

    /* Fica à espera de mensagens do cliente e reenvia-as para o cliente */
    dg_echo(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr));
} 8/28/2003
```

José Alves Marques

79

Servidor DGRAM AF_UNIX (3)

```
#define MAXLINE 512

/* Servidor do tipo socket datagram.
Manda linhas recebidas de volta
para o cliente */

#include <sys/types.h>
#include <sys/socket.h>
#define MAXMSG 2048

/* pcli_addr especifica o cliente */
dg_echo(sockfd, pcli_addr, maxclilen)
{
    int sockfd;
    struct sockaddr *pcli_addr;
    int maxclilen;
    {
        int n, clilen;
        char msg[MAXMSG];

        for (;;) {
            clilen = maxclilen;

            /* Lê uma linha do socket */
            n = recvfrom(sockfd, msg, MAXMSG,
                        0, pcli_addr, &clilen);
            if (n < 0)
                err_dump("dg_echo:recvfrom error");

            /*Manda linha de volta para o socket */
            if (sendto(sockfd, msg, n, 0,
                      pcli_addr, clilen) != n)
                err_dump("dg_echo: sendto error");
        }
    }
}
```

8/28/2003

José Alves Marques

80

Espera Múltipla com Select

```
#include <sys/select.h>
#include <sys/time.h>
int select (int maxfd, fd_set* leitura, fd_set*
            escrita, fd_set* excepcao, struct timeval*
            alarme)
```

select:

- espera por um evento
- bloqueia o processo até que um descritor tenha um evento associado ou expire o alarme
- especifica um conjunto de descritores onde espera:
 - receber mensagens
 - receber notificações de mensagens enviadas (envios assíncronos)
 - receber notificações de acontecimentos excepcionais

8/28/2003

José Alves Marques

81

Select

- exemplos de quando o select retorna:
 - Os descritores (1,4,5) estão prontos para leitura
 - Os descritores (2,7) estão prontos para escrita
 - Os descritores (1,4) têm uma condição excepcional pendente
 - Já passaram 10 segundos

Espera Múltipla com Select (2)

```
struct timeval {  
    long tv_sec; /* seconds */  
    long tv_usec; /* microseconds */  
}
```

- esperar para sempre → parâmetro efectivo é null pointer)
- esperar um intervalo de tempo fixo → parâmetro com o tempo respectivo
- não esperar → parâmetro com o valor zero nos segundos e microsegundos
- as condições de excepção actualmente suportadas são:
 - chegada de dados out-of-band
 - informação de controlo associada a pseudo-terminais

Manipulação do fd_set

- Definir no select quais os descritores que se pretende testar
 - void FD_ZERO (fd_set* fdset) - clear all bits in fdset
 - void FD_SET (int fd, fd_set* fd_set) - turn on the bit for fd in fdset
 - void FD_CLR (int fd, fd_set* fd_set) - turn off the bit for fd in fdset
 - int FD_ISSET (int fd, fd_set* fd_set) - is the bit for fd on in fdset?
- Para indicar quais os descritores que estão prontos, a função select modifica:
 - fd_set* leitura
 - fd_set* escrita
 - fd_set* excepcao

8/28/2003

José Alves Marques

84

Servidor com Select

```
/* Servidor que utiliza sockets stream e
   datagram em simultâneo.
   O servidor recebe caracteres e envia-os
   para stdout */

#include <stdio.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <errno.h>

#define MAXLINE 80
#define MAXSOCKS 32

#define ERRORMSG1 "server: cannot open stream
socket"
#define ERRORMSG2 "server: cannot bind stream
socket"
#define ERRORMSG3 "server: cannot open
datagram socket"
#define ERRORMSG4 "server: cannot bind
datagram socket"
#include "names.h"
```

8/28/2003

José

```
int main(void) {
    int strmfd, dgrmfd, newfd;
    struct sockaddr_un
        servstrmaddr, servdgrmaddr, clientaddr;
    int len, clientlen;
    fd_set testmask, mask;

    /* Cria socket stream */
    if ((strmfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
        perror(ERRORMSG1);
        exit(1);
    }
    bzero((char*)&servstrmaddr,
          sizeof(servstrmaddr));
    servstrmaddr.sun_family = AF_UNIX;
    strcpy(servstrmaddr.sun_path, UNIXSTR_PATH);
    len = sizeof(servstrmaddr.sun_family)
        + strlen(servstrmaddr.sun_path);
    unlink(UNIXSTR_PATH);
    if (bind(strmfd, (struct sockaddr*)&servstrmaddr,
             len) < 0)
    {
        perror(ERRORMSG2);
        exit(1);
    }
}
```

Servidor com Select (2)

```

/*Servidor aceita 5 clientes no socket stream*/
listen(strmfd, 5);
/* Cria socket datagram */
if((dgrmfd = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0)
{
    perror(ERRORMSG3);
    exit(1);
}
/*Inicializa socket datagram: tipo + nome */
bzero((char
*) &servdgrmaddr, sizeof(servdgrmaddr));
servdgrmaddr.sun_family = AF_UNIX;
strcpy(servdgrmaddr.sun_path, UNIXDG_PATH);
len = sizeof(servdgrmaddr.sun_family) +
        strlen(servdgrmaddr.sun_path);
unlink(UNIXDG_PATH);
if(bind(dgrmfd, (struct
*) &servdgrmaddr, len) < 0)
{
    perror(ERRORMSG4);
    exit(1);
}

```

/*

- Limpa-se a máscara
- Marca-se os 2 sockets - stream e datagram.
- A máscara é limpa pelo sistema de cada vez que existe um evento no socket.
- Por isso é necessário utilizar uma máscara auxiliar

*/

```

FD_ZERO(&testmask);
FD_SET(strmfd, &testmask);
FD_SET(dgrmfd, &testmask);

```

8/28/2003

José Alves Marques

86

Servidor com Select (3)

```

for(;;) {
    mask = testmask;

    /* Bloqueia servidor até que se dê um evento. */
    select(MAXSOCKS, &mask, 0, 0, 0);

    /* Verificar se chegaram clientes para o socket stream */
    if(FD_ISSET(strmfd, &mask)) {
        /* Aceitar o cliente e associa-lo a newfd. */
        clientlen = sizeof(clientaddr);
        newfd = accept(strmfd, (struct sockaddr*)&clientaddr, &clientlen);
        echo(newfd);
        close(newfd);
    }

    /* Verificar se chegaram dados ao socket datagram. Ler dados */
    if(FD_ISSET(dgrmfd, &mask))
        echo(dgrmfd);

    /*Voltar ao ciclo mas não esquecer da máscara! */
}

```

8/28/2003

José Alves Marques

87

IPC no Sistema V

caixas de correio
memória partilhada
semáforos

8/28/2003

José Alves Marques

88

IPC no Sistema V

⇒ **três tipos de primitivas:**

- novos mecanismos:
 - caixas de correio
 - memória partilhada
 - semáforos

- ♦ get
- ♦ ctl
- ♦ op

	caixas de correio	semáforos	memória partilhada
<i>header</i>	<sys/msg.h>	<sys/sem.h>	<sys/shm.h>
<i>create or open</i>	msgget	semget	shmget
<i>control</i>	msgctl	semctl	shmctl
<i>operations</i>	msgsnd msgrcv	semop	shmat shmdt

8/28/2003

José Alves Marques

89

IPC no Sistema V (II)

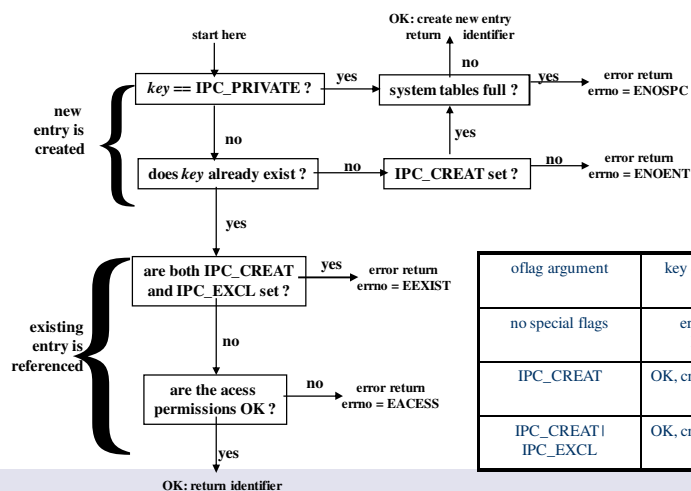
- cada objecto é identificado por uma key
- o espaço de nomes é separado do sistema de ficheiros
- os nomes são locais a uma máquina
- as permissões de acesso são idênticas às de um ficheiro (r/w para user/group/other)
- os processos filho herdam os objectos abertos

8/28/2003

José Alves Marques

90

IPC no Sistema V (cont.)



oflag argument	key does not exist	key already exist
no special flags	error, errno = ENOENT	OK, references existing object
IPC_CREAT	OK, creates new entry	OK, references existing object
IPC_CREAT IPC_EXCL	OK, creates new entry	error, errno = EEXIST

8/28/2003

José Alves Marques

92

Caixas de Correio (Sistema V)

8/28/2003

José Alves Marques

93

IPC no SV - Caixas de Correio

- as caixas de correio são listas “fifo” de mensagens
- no envio a mensagem é colocada na cauda da fila
- na recepção é retirada a primeira mensagem da fila ou a primeira mensagem de um tipo particular
- uma mensagem tem a estrutura:

```
struct msgbuf {
    long type;
    char text[1];
}
```
- obtenção ou criação de uma fila de mensagens:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget (key_t key, int msgflg)
```

- envia uma mensagem apontada por msgp de dimensão msgsz:

```
int msgsnd (int msqid, struct msgbuf * msgp, int msgsz,
            int msgflg)
```
- recebe uma mensagem no tampão apontado por msgp

```
int msgrcv (int msqid, struct msgbuf * msgp, int msgsz,
            long msgtyp, int msgflg)
```
- a mensagem pode ser truncada a msgsz
- se msgtyp=0 é recebida a primeira mensagem da fila
- se msgtyp>0 é recebida a primeira mensagem desse tipo
- se msgtyp<0 é recebida a primeira mensagem de menor tipo inferior ao módulo de msgtyp

8/28/2003

José Alves Marques

94

Caixas de Correio - Função de Controlo

- **sintaxe:**

```
int msgctl (int msqid, int cmd,
            struct msqid_ds * buf)
```
- **estrutura msqid_ds** (mantida no núcleo para cada caixa de correio) contém:

ipc_perm	msg_perm;	/*permissões*/
ushort	msg_cbytes;	/* # de bytes actual*/
ushort	msg_qnum;	/*# de mens. na fila*/
ushort	msg_qbytes;	/*# max de bytes*/
ushort	msg_lspid;	/*pid do último msgnd*/
ushort	msg_lrpid;	/*pid do último msgrcv*/
time_t	msg_stime;	/*data último msgsnd*/
time_t	msg_rtime;	/*data último msgrcv*/
time_t	msg_ctime;	/*data última modif.*/

⇒ **comandos possíveis:**

- ♦ IPC_STAT preenche *buf* com estado actual
- ♦ IPC_SET inicializa parametros a partir de *buf*
- ♦ IPC_RMID elimina a fila de mensagens

8/28/2003

José Alves Marques

95

Caixas de Correio - Cliente

```
/* Cliente */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define TAMMSG 100
#define SERVIDOR 10
#define CLIENTE 11

struct msgTeste {
    long    mtype;
    int     ident;
    char    texto [TAMMSG];
} msg;
```

```
main () {
    int CCcliente, CCServ;

    if ((CCcliente = msgget (CLIENTE, 0777 | IPC_CREAT)) < 0)
        perror("msgget CLIENTE");
    if ((CCServ = msgget (SERVIDOR, 0)) < 0)
        perror("msgget SERVIDOR");
    msg.ident = CLIENTE;
    msg.mtype = 1;
    ProduzMensagem (msg.texto);

    /* Envia pedido e recebe resposta */
    if (msgsnd (CCServ, &msg, sizeof(msg), 0) < 0)
        erro ("msgsnd");
    if (msgrcv (CCcliente, &msg, sizeof(msg), 0, 0) < 0)
        erro ("msgrcv");

    TrataMensagem (msg.texto);

    if(msgctl (CCcliente, IPC_RMID, (struct msqid_ds *) 0) < 0)
        perror("msgctl");
}
```

8/28/2003

José Alves Marques

96

Caixas de Correio - Servidor

```
main () {
    int CCcliente;
    key_t ChaveCli;

    if ((CCServ = msgget (SERVIDOR, 0777 | IPC_CREAT) ) < 0)
        perror("msgget servidor");

    for (;;) {
        if (msgrcv (CCServ, &msg, sizeof(msg), 0, 0) < 0)
            perror("msgrcv");
        ChaveCli = msg.ident;
        if ((CCcliente = msgget (ChaveCli, 0)) < 0)
            perror("msgget cliente");

        TrataMensagem (msg.ident, msg.texto);

        msg.ident = SERVIDOR;
        if (msgsnd (CCcliente, &msg, sizeof(msg), 0) < 0)
            perror("msgsnd");
    }
}
```

8/28

Memória Partilhada (Sistema V)

IPC no SV - Memória Partilhada

- permite o acesso de vários processos a uma zona de memória comum
- a dimensão do segmento não pode ser alterada depois da criação
- cada processo pode “ver” o segmento em endereços distintos do seu espaço de endereçamento
- criação de uma região:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget (key_t key, int size, int shmflg)
```

- size especifica a dimensão da região em bytes

IPC no SV - Memória Partilhada

- associação a uma região:

```
char* shmat (int shmid, char *shmaddr, int shmflg)
```
- devolve o endereço base da região
- o endereço pode:
 - ser especificado por shmaddr
 - se shmaddr for zero, o endereço é calculado pelo sistema
- se shmflg = SHM_RDONLY o acesso fica restrito a leitura
- eliminação da associação:

```
int shmdt (char *shmaddr);
```

Memória Partilhada - Controlo

- **sintaxe:**

```
int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;
```
- **a estrutura shmid_ds (mantida no núcleo para cada região de memória partilhada) contém:**

ipc_perm	shm_perm;	/*permissões*/
int	shm_segsz;	/*dimensão em bytes*/
ushort	shm_cpid;	/*pid do criador*/
ushort	shm_lpid;	/*pid do último shmop*/
ushort	shm_nattch;	/*#actual de ligações*/
time_t	shm_atime;	/*data último shmat*/
time_t	shm_dtime;	/*data último shmdt*/
time_t	shm_ctime;	/*data última modif.*/

⇒ **comandos possíveis:**

- ♦ IPC_STAT preenche *buf* com estado actual
- ♦ IPC_SET inicializa parametros a partir de *buf*
- ♦ IPC_RMID elimina a memória partilhada

8/28/2003

José Alves Marques

101

Exemplo: Memória Partilhada

```
/* produtor */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define CHAVEMEM 10
int IdRegPart;
int *Apint;
int i;

main () {
    IdRegPart = shmget (CHAVEMEM, 1024, 0777| IPC_CREAT);
    if (IdRegPart<0) perror(" shmget:");

    printf (" criou uma regio de identificador %d \n",
            IdRegPart);

    Apint = (int *) shmat (IdRegPart, (char *) 0, 0);
    if (Apint == (int *) -1) perror("shmat:");

    for (i = 0; i<256; i++) *Apint++ = i;
}
```

8/28/2003

José Alves Marques

102

Exemplo: Memória Partilhada

```
/* consumidor*/  
  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/shm.h>  
  
#define CHAVEMEM 10  
  
int IdRegPart;  
int *Apint;  
int i;
```

```
main() {  
    IdRegPart = shmget (CHAVEMEM, 1024, 0777);  
    if (IdRegPart < 0)  
        perror("shmget:");  
  
    Apint=(int*)shmat (IdRegPart, (char *)0, 0);  
    if (Apint == (int *) -1)  
        perror("shmat:");  
  
    printf(" mensagem na regioao de memoria partilhada \n");  
    for (i = 0; i<256; i++)  
        printf ("%d ", *Apint++);  
  
    printf (" \n liberta a regioao partilhada \n");  
    shmctl (IdRegPart, 0, IPC_RMID,0);  
}
```

Semáforos (Sistema V)

IPC no SV - Semáforos

- um semáforo consiste num conjunto de contadores que só podem ter valores positivos
- criação de um semáforo:


```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget (key_t key, int nsems,
            int semflg)
```
- “nsems” especifica o número de contadores, que são inicializados a zero na criação
- operações sobre semáforos:


```
int semop (int semid,
           struct sembuf *sops,
           int nsops)
```

são executadas as nsops operações definidas em sops e devolvido o valor do último contador acedido
uma operação é definida, de acordo com a estrutura sembuf, por:

```
short sem_num;    /* numero */
short sem_op;     /* operação */
short sem_flg;    /* flags */
```

segundo o valor de sem_op temos:

- ⇒ sem_op > 0 o valor de sem_op é adicionado ao do contador
- ⇒ sem_op < 0 o valor de sem_op é adicionado ao do contador; o processo pode ficar bloqueado
- ⇒ sem_op = 0 o processo é suspenso até o valor do contador atingir zero

8/28/2003

José Alves Marques

105

Semáforos - Controlo

- sintaxe:


```
int semctl (int semid,
           int semnum,
           int cmd,
           union semun
           arg)

union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
};
```

⇒ comandos possíveis:

- ♦ IPC_STAT preenche *arg.buf* com estado actual
- ♦ IPC_SET inicializa parâmetros a partir de *buf.arg*
- ♦ IPC_RMID elimina o semáforo em causa
- ♦ GETALL copia os valores dos contadores para *arg.array*
- ♦ SETALL inicializa os valores a partir de *arg.array*

⇒ comandos possíveis com *semnum* especificado:

- ♦ GETVAL devolve o valor do contador
- ♦ SETVAL inicializa o valor do contador
- ♦ GETPID devolve o pid da última operação
- ♦ GETNCNT devolve o número de processos aguardando um valor não zero do contador
- ♦ GETZCNT devolve número de processos aguardando um valor zero do contador

8/28/2003

José Alves Marques

106

Operações com semáforos

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define SEMMUTEX 5
#define SEMALOC 6
int mutex, semaloc;

void IniSem(int nmax) {
    union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
    } init;

    if ((mutex = semget (SEMMUTEX, 1, 0777|IPC_CREAT)) < 0) erro ("semget SEMMUTEX");
    if ((semaloc = semget (SEMALOC, 1, 0777|IPC_CREAT)) < 0) erro("semget SEMALOC");
    init.val = 1;
    if (semctl (mutex, 0, SETVAL, init) < 0) erro("semctl mutex");
    init.val = nmax;
    if (semctl (semaloc, 0, SETVAL, init) < 0) erro("semctl SemAloc");
}
```

8/28/2003

José Alves Marques

107

Operações com semáforos(II)

```
void EsperarMutex (int semid, int semnum)
{
    struct sembuf s;
    s.sem_num = semnum;
    s.sem_op = -1;
    s.sem_flg = SEM_UNDO;
    if (semop (semid, &s, 1) < 0)
        erro ("semop EsperarMutex");
}

void AssinalarMutex (int semid, int
semnum) {
    struct sembuf s;
    s.sem_num = semnum;
    s.sem_op = 1;
    s.sem_flg = SEM_UNDO;
    if (semop (semid, &s, 1) < 0)
        erro ("semop AssinalarMutex");
}
```

```
void Esperar (int semid, int semnum, int
uni) {
    struct sembuf s;
    s.sem_num = semnum;
    s.sem_op = -uni;
    s.sem_flg = 0;
    if (semop (semid, &s, 1) < 0)
        erro ("semop Esperar");
}

void Assinalar (int semid, int semnum,
int uni) {
    struct sembuf s;
    s.sem_num = semnum;
    s.sem_op = uni;
    s.sem_flg = 0;
    if (semop (semid, &s, 1) < 0)
        erro ("semop Assinalar");
}
```

Comunicação em Windows

Named pipes e mailslots
Windows Sockets (Winsock)
Drivers de protocolo
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 feita 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 unidireccional
 - 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

8/28/2003

José Alves Marques

111

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 assincronamente
 - ConnectNamedPipe só se completa quando um cliente faz um pedido de estabelecimento de ligação (análogo ao *accept* nos sockets)

8/28/2003

José Alves Marques

112

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

8/28/2003

José Alves Marques

113

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,  
      
);
```

8/28/2003

114

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 interações 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.

8/28/2003

José Alves Marques

115

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

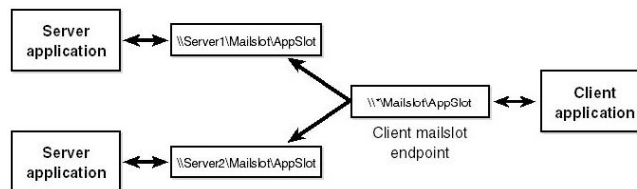
8/28/2003

José Alves Marques

116

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 mailslot 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*



8/28/2003

José Alves Marques

117

Mailslots (3)

- Dimensão das mensagens:
 - O limite teórico para a dimensão das mensagens é de 64 *Kbytes*.
 - em termos operacionais este limite é de 424 *bytes*,
 - limite da dimensão de um datagrama em Windows
 - Para enviar mensagens de maior dimensão, é necessário recorrer a um protocolo ponto-a-ponto com ligação
 - Ex: SMB (*Server Message Blocks*)
 - No caso de difusão, a dimensão das mensagens é estritamente limitada a 424 *bytes*.

8/28/2003

José Alves Marques

118

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.

8/28/2003

José Alves Marques

119

Windows Sockets - Winsock

- **É a implementação dos Sockets BSD pela Microsoft:**
 - Tem algumas funcionalidades extra.
- **Suporta:**
 - 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

8/28/2003

José Alves Marques

122

Windows Sockets - Winsock (2)

- **Suporta:**
 - Aceitação de uma ligação avaliada dinamicamente, delegada numa função *callback* que é invocada sempre que ocorre um pedido de ligação.
 - Transmissão de dados associada ao estabelecimento e fecho de uma ligação
 - Reduz a latência na interação cliente-servidor. Mensagens trocadas no estabelecimento da ligação (normalmente opacas às aplicações) também carregam dados.
 - 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*

8/28/2003

José Alves Marques

123

Winsock

- **API independente do protocolo usado:**
 - Pode ser especificado um endereço para qualquer um dos NetBEUI, TCP/IP, IPX
- **Passos iniciais (comuns aos clientes e servidores):**
 - Inicializar a Winsock API
 - Criar um *socket* que representa um endpoint.
 - Fazer *bind* do socket a um endereço no computador local
- **Passos seguintes divergem conforme se trate:**
 - Do servidor ou do cliente
 - Canal com ou sem ligação
- **Para um canal sem ligação:**
 - Enviar/receber dados com *send/recv* indicando/obtendo o endereço do receptor/emissor

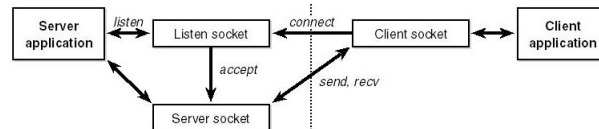
8/28/2003

José Alves Marques

124

Winsock com Ligação

- Passos do lado do servidor:
 - *listen* indica o número máximo de pedidos de ligação simultâneos
 - *accept* bloqueia o servidor à espera de pedidos de ligação do cliente
 - se houver algum pedido pendente, o servidor é desbloqueado de imediato
 - caso contrário, o servidor só será desbloqueado quando houver um pedido de estabelecimento de ligação
 - quando uma ligação é estabelecida o *accept* retorna um novo socket que é o "endpoint" da ligação
- Operações de envio e recepção de mensagens:
 - *send* e *recv*
- Passos do lado do cliente:
 - Pedido de estabelecimento de ligação com função *connect* indicando o endereço do servidor
 - especifica a remote address.



8/28/2003

José Alves Marques

125

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
);
  
```

8/28/2003

José Alves Marques

126

Windows Sockets

Envio paralelo de múltiplos *buffers* (*Scather 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);
```

8/28/2003

José Alves Marques

127

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);
```

8/28/2003

José Alves Marques

128

Extensibilidade da Winsock

- Pode ser adicionado um transport service provider que:
 - Suporte outros protocolos de comunicação
 - Suporte um namespace service provider que aumente as funcionalidades das operações de nomeação
- Namespace service providers:
 - Possibilitam que os servidores se registem de formas diferentes
 - Exemplo:
 - Permitir que o servidor registe o seu endereço na Active Directory
 - Permitir que os clientes procurem o servidor que pretendem usando o Active Directory
 - Suportam esta funcionalidade através da implementação das funções da Winsock relacionadas com a nomeação (ex.: gethostbyaddr, getservbyname, getservbyport)
- Os “service providers” usam a Winsock service provider interface (SPI):
 - Quando um “transport service provider” é registado na Winsock, esta usa as funções oferecidas por este:
 - Connect, accept, etc.
 - Não há restrições quanto à implementação das funções em causa (que implica interação com o transport driver no núcleo)

8/28/2003

José Alves Marques

129

WinSock Comunicação Multiponto

- Moldura para definição de sistemas de comunicação multiponto
 - comunicação em grupo, difusão, teleconferência, etc.
- A definição é feita a dois níveis, designados planos:
 - **plano-controlo**: determina o modo como uma sessão de comunicação multi-ponto é estabelecida
 - **plano-dados**: especifica como é realizada a transmissão de dados entre os participantes no âmbito de uma sessão
- Cada um dos planos pode ter ou não coordenação
 - Existência de um **nó-raiz** que controla a comunicação
 - Restantes nós são designados **nós-folha**

8/28/2003

José Alves Marques

131



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

WinSock

Comunicação Multiponto (2)

- **Coordenação no plano-controlo**
 - Nó-raiz desencadeia o estabelecimento da sessão
 - Permanece disponível durante toda a duração da sessão,
 - Centraliza a associação de novos participantes à sessão.
 - Ausência de coordenação assume sessão pré-existente
- **Coordenação no plano-dados**
 - Nó-raiz coordena os fluxos de informação
 - Nó-raiz que centraliza a comunicação
 - Não há transferências de dados directamente entre nós-folha.
 - Estes apenas enviam e recebem dados para, e do nó-raiz.
 - Na Ausência de coordenação
 - todos os nós-folha podem trocar dados
 - Podem receber dados de outros processos não inscritos na sessão

8/28/2003
José Alves Marques
132



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

WinSock

Comunicação Multiponto - Exemplos

- **Multicast IP**
 - Plano-controlo: sem coordenação
 - entrada na sessão: escuta de um endereço multicast-IP comum
 - Plano-dados: sem coordenação
 - todos os nós podem enviar para esse endereço e todos recebem os dados
- **T.120 (Teleconferência)**
 - Plano-controlo: com coordenação
 - nó-raiz (top-conference-provider) controla a entrada na sessão
 - Plano-dados: sem coordenação
 - contempla comunicação directa entre nós-folha (conference clients)
- **Redes ATM**
 - Plano-controlo: com coordenação
 - nó-raiz controla a entrada na sessão
 - Plano-dados: com coordenação
 - Comunicação exclusivamente unidireccional do nó-raiz para os nós-folha

8/28/2003
José Alves Marques
133

Mecanismos de Comunicação entre Janelas no Windows

Window Messages

Átomos

Mensagens de Cópia de Dados

Clipboard

8/28/2003

José Alves Marques

134

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

8/28/2003

José Alves Marques

135

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*

8/28/2003

José Alves Marques

136

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

8/28/2003

José Alves Marques

137

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_LBUTTONDOWN` 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

8/28/2003

José Alves Marques

138

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 interbloqueio se ocorrer envio directo entre duas janelas, nos dois sentidos

8/28/2003

José Alves Marques

139



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

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.

8/28/2003
José Alves Marques
141



INSTITUTO
SUPERIOR
TÉCNICO

Departamento de Engenharia Informática

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.

8/28/2003
José Alves Marques
142

Comunicação entre Janelas no Windows

Window Messages (7)

- Rotinas de Tratamento (window procedures)
 - Cada classe de janelas tem uma rotina de tratamento
 - A mesma para todas as janelas da mesma classe
 - A rotina é invocada quando a mensagem é entregue à janela
 - A rotina é responsável pelo processamento da mensagem
 - Caso não o faça, delega esse tratamento ao Windows
 - rotina de tratamento por omissão: **DefWindowProc**
 - Nunca são descartadas mensagens
 - A rotina recebe como parâmetros cada campo da mensagem
 - janela-destino (**hWnd**), classe da mensagem (**msg**) e campos adicionais

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
```

8/28/2003

José Alves Marques

143

Comunicação entre Janelas no Windows

Window Messages (9)

- Herança entre Classes de Janelas - Delegação
 - Redefinição da rotina tratadora de uma janela
 - Realizado usando a função **SetWindowLong** com parâmetro **GWL_WNDPROC**
 - Devolve como retorno um apontador para a rotina tratadora antiga
 - Na primeira vez, este diz respeito à rotina tratadora da superclasse
 - Se este apontador for salvaguardado, a rotina pode ser invocada mais tarde.
 - Na prática, esta forma de delegação ocorre quase sempre
 - a rotina de tratamento da subclasse apenas se encarrega de classes mensagens ou parâmetros muito específicos.

8/28/2003

José Alves Marques

145

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

8/28/2003

José Alves Marques

150

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

8/28/2003

José Alves Marques

151