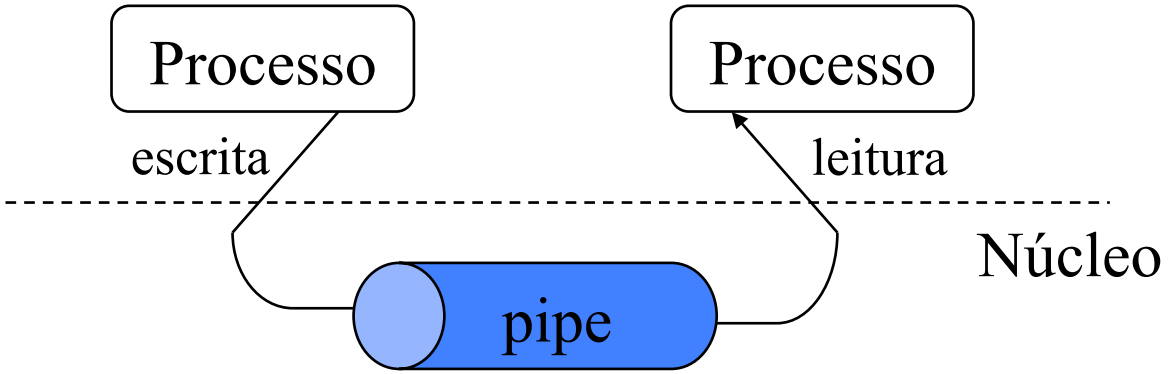
Programação de Sistemas

Tubos

Tubos - introdução

- No Unix, os tubos (“pipes”) constituem o mecanismo original de comunicação entre processos.
  - Mensagens limitadas a sequências de Bytes (caracteres).
  - Fluxo de informação unidireccional (i.e., um processo escreve num canal e o outro lê do canal).
  - Descritores semelhantes aos dos ficheiros.



Tubos – definição

- Um tubo é criado pela chamada de sistema

  POSIX:
  ```
  #include <unistd.h>
  int pipe(int fd[2]);
  ```
  - Descritor `fd[0]` aberto para leitura
  - Descritor `fd[1]` aberto para escrita.
  - Em caso de sucesso (erro) retorna 0 (-1, com causa de erro indicada na variável de ambiente `int errno;`)
  - Os tubos apenas podem ligar processos com antepassado comum.

Nota: Os descritores são índices de uma tabela contendo os detalhes de todos os ficheiros abertos.

- Frequentemente, os tubos constituem o canal de comunicação entre os processos pai-filho, sendo definidos antes do lançamento dos processos descendentes (por `fork`).

Tubos – fecho (1)

- Depois de usados, ambos os canais de comunicação devem ser fechados pela chamada do sistema

  POSIX:
  ```
  #include <unistd.h>
  int close(int);
  ```

  – Em caso de sucesso (erro) retorna 0 (-1, com causa de erro indicada na variável de ambiente `int errno;`)

Exemplo:

```
int fd[2];
if (pipe(fd)==0) {
   …
   close(fd[0]); close(fd[1]);
}
```

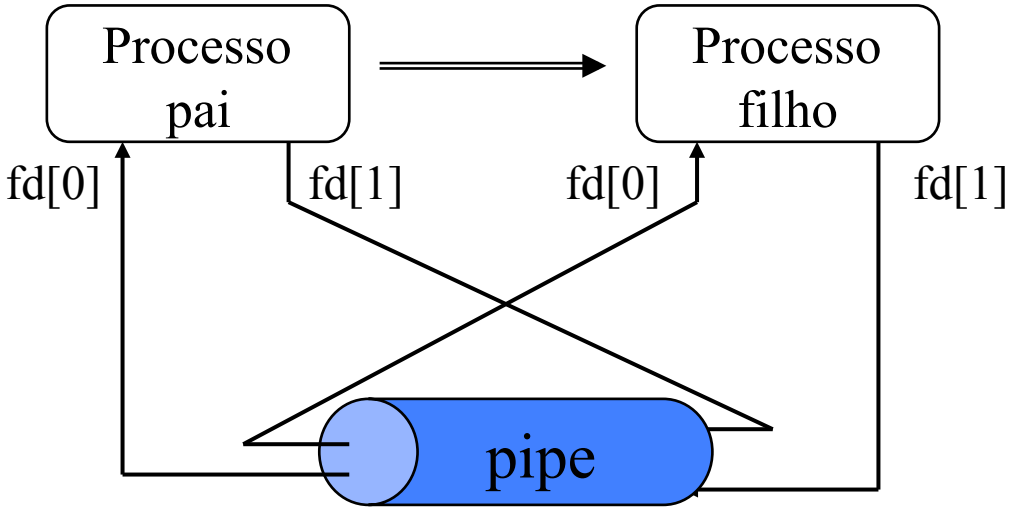
Tubos – fecho (2)

- Quando todos os descritores associados a um tubo, ou FIFO são fechados, todos os dados residentes no tubo ou no FIFO são perdidos.
- Quando todos os descritores associados a um ficheiro são fechados, o descritor do ficheiro é eliminado.

Nota: quando o descritor do ficheiro é eliminado, todos os dados residentes na memória que ainda não tenham sido enviados para disco, são descarregados ("flushed").

Tubos – fecho (3)

- Quando um processo faz um `fork()` depois de ter aberto um tubo, ambos os processos ficam com extremidades de leitura e escrita.

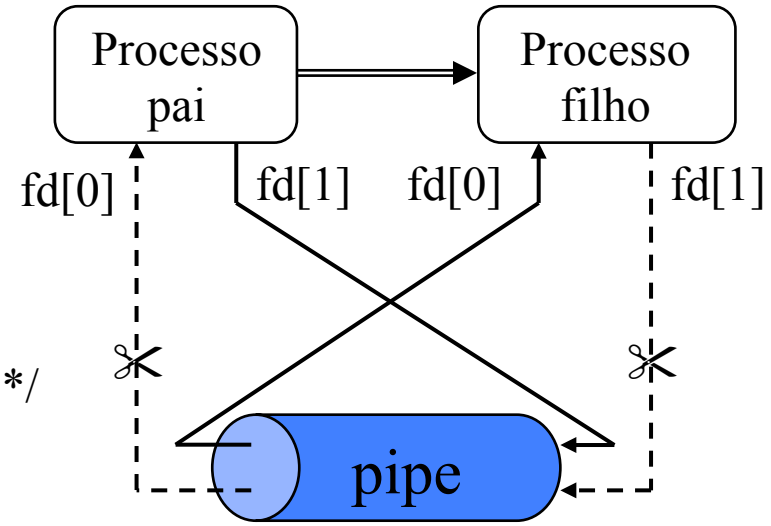


- Cada um dos processos **deve** fechar a extremidade desaproveitada.

Tubos – fecho (4)

- Por exemplo, se o fluxo de informação for pai a escrever e filho a ler,

```
int fd[2];
pid_t pid;

if( pipe(fd)<0 ) exit(1);
if( pid=fork()<0 ) exit(1);

if ( pid==0 ) { /* processo filho */
   close( fd[1] );
   …
}
if ( pid>0 ) { /* processo pai */
   close( fd[0] );
   …
}
```



Tubos – comunicação (1)

- Comunicação feita pelas seguintes chamadas de sistema:

  POSIX:
  ```
  #include <unistd.h>
  ssize_t read(int,char *,int);
  ssize_t write(int,char *,int);
  ```

  – O 1º parâmetro é o descritor de ficheiro.
  – O 2º parâmetro é o endereço dos dados de utilizador.
  – O 3º parâmetro é o número de Bytes a comunicar.
  – A função devolve o número de Bytes efectivamente transferidos.

  Nota1: O número de Bytes que podem ser temporariamente armazenados por um tubo é indicado por _POSIX_PIPE_BUF (512B, definido em `<limits.h>`)

  Nota2: Envio directo de dados como int, e.g. write(fd, val, sizeof(int)), gera problemas se as dimensões de dados nas máquinas do processos intervenientes forem distintas!

Tubos – comunicação (2)

- Para uma comunicação bidireccional usar dois tubos.
- Regras aplicadas aos processos escritores:
  - Escrita para descritor fechado resulta na geração de sinal `SIGPIPE` (acção por omissão de terminar o processo).
  - Escrita de dimensão inferior a _POSIX_PIPE_BUF é atómica (i.e., os dados não são entrelaçados). No caso do pedido de escrita ser superior a _POSIX_PIPE_BUF, os dados podem ser entrelaçados com pedidos de escrita vindos de outros processos.
- Regras aplicadas aos processos leitores:
  - Leitura para descritor fechado retorna valor 0.
  - Processo que pretenda ler de um tubo vazia fica bloqueado até que um processo escreva os dados.

Tubos – comunicação (3)

Exemplo: Envio de dados do processo filho para o pai.

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

#define READ 0
#define WRITE 1
#define STDOUT 1

int main() {
   int n, fd[2];
   pid_t pid;

   if ( pipe(fd)<0 ) { fprintf(stderr,"Erro no tubo\n");_exit(1); }
   if ( (pid=fork())<0 ) { fprintf(stderr,"Erro no fork\n");_exit(1); }
```

Tubos – comunicação (4)

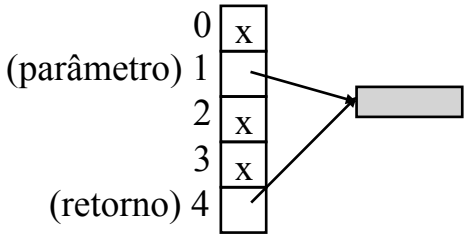
```
   if ( pid>0 ) { /* processo ascendente */
#define MAX 128
      char line[MAX];
      close( fd[WRITE] );
      n = read(fd[READ],line,MAX);
      write(STDOUT, &line[0], n);
      close( fd[READ] );
      kill( pid,SIGKILL );    /* elimina processo descendente */
      _exit(0); }

   if ( pid==0 ) { /* processo descendente */
#define LEN 8
      char msg[LEN]={'B','o','m',' ','d','i','a','\n'};
      close( fd[READ] );
      write( fd[WRITE], &msg[0], LEN);
      close( fd[WRITE] );
      pause(); }
}
```

APUE 3.12

Redirecção E/S por tubos (1)

- A redirecção E/S é feita com auxílio da chamada de sistema de duplicação do descritor de ficheiro

  POSIX:
  ```
  #include <unistd.h>
  int dup(int);
  ```
  - O parâmetro é o descritor de ficheiro duplicado.
  - O valor retornado identifica um novo descritor para o ficheiro (Nota: o ficheiro não é duplicado, apenas o descritor!).
    O valor devolvido é o primeiro índice vago da tabela de descritores.



Redirecção E/S por tubos (2)

- Os dois descritores partilham a mesma posição do ficheiro.
  Se fosse efectuado `open` duas vezes sobre o mesmo ficheiro, cada descritor é posicionado em locais distintos!

- A função dup2 duplica descritor de ficheiro, fechando previamente o primeiro descritor

  POSIX:
  ```
  #include <unistd.h>
  int dup2(int,int);
  ```

  – O 2º parâmetro é o descritor fechado.

  `dup2(newD,d);` é equivalente a `close(d);dup(newD);`

Redirecção E/S por tubos (3)

- O programa redirecciona E/S nas seguintes etapas:
  a) Abrir ficheiro, com modo de acesso apropriado
  b) Fechar descritor de E/S (0-entrada, 1-saída).
  c) Executar `dup`, com parâmetro do valor recolhido no passo a).
  d) Fechar o descritor recolhido no passo a).

Exemplo: redireccionar escrita de instruções printf

```
int newD=open("fich.txt",
    O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);
/* newD recebe um descritor livre, por exemplo 4 */
close(1);       /* liberta descritor 1 */
dup(newD);      /* procura primeiro descritor livre, que é 1 */
close(newD);
/* descritor 4 libertado, mas fich.txt continua acedido pelo descritor 1 */
```

Redirecção E/S por tubos (4)

- Os tubos são implicitamente criados quandos dois comandos são sequenciados por `|`
- A saída do comando anterior é reencaminhada para a entrada do comando seguinte.
  Ex: `who | sort | lpr`
  Imprime utilizadores em sessão, por ordem alfabética.
  - No Unix, os comandos são lançados em simultâneo com tubos ligando o comando anterior ao comando seguinte.
  - No sistema operativo MSDOS, de tarefa única, cada comando na ordem pos executado isoladamente com a saída enviada para o ficheiro pos.tmp (armazenado na directoria %TEMP%, se não existir na directoria corrente) e o comando seguinte a ler a entrada do ficheiro.

3º EXERCÍCIO TEORICO-PRATICO

- Neste exercício dois processos escrevem no mesmo tubo.
  1. O processo *Double* lança dois processos filho, de identificadores A e B. Cada processo filho substitui a sua imagem.
     Logo, crie os programas Double.c e Son.c
  2. Cada imagem envia para o processo pai a saudação "ID: Bom dia!", em que ID é o identificador.
     Nota: Como o tubo é único, ele tem de ser criado antes dos processos filho serem lançados.
  3. O processo filho termina só depois de enviar a mensagem.
  4. O processo *Double* lê e imprime as mensagens no terminal. Por fim, fica à espera que os processos filho terminem.