

The background of the slide features a close-up of a white Apple logo on the left and a green Android robot on the right, both resting on a wooden surface. A green laser beam is visible in the upper center. A horizontal bar with a teal segment on the left and a green segment on the right spans the top of the slide.

# SISTEMAS OPERACIONAIS

Inter-process Communication (IPC)  
Tubos (Pipes) Filas (FIFOs, Named Pipes)

# Comunicação entre Processos (1)

2

- Os sistemas operacionais implementam mecanismos que asseguram a independência entre processos.
  - Processos executam em cápsulas autônomas
    - A execução de um processo não afeta os outros.
  - Hardware oferece proteção de memória.
    - Um processo não acessa o espaço de endereçamento do outro.



# Comunicação entre Processos (2)

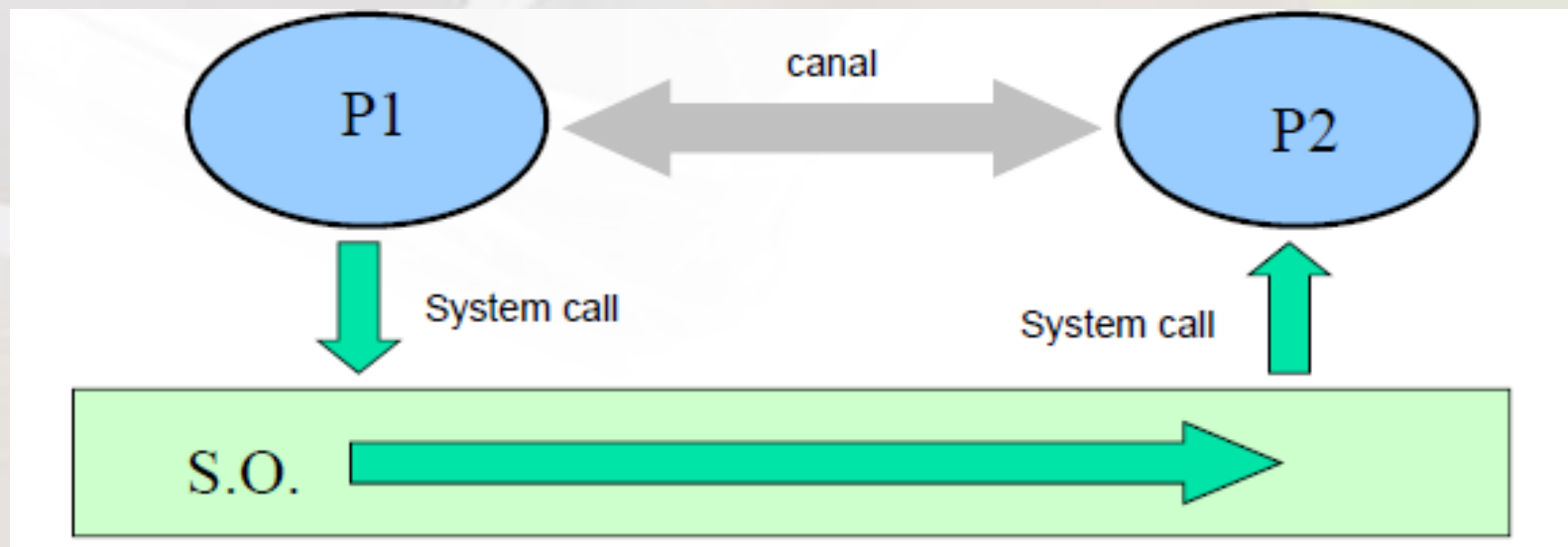
3

- Processos, entretanto, interagem e cooperam na execução de tarefas. Em muitos casos, processos precisam trocar informação de forma controlada para
  - dividir tarefas e aumentar a velocidade de computação;
  - aumentar da capacidade de processamento (rede);
  - atender a requisições simultâneas.
- Solução: S.O. fornece mecanismos que permitem aos processos comunicarem-se uns com os outros (IPC).
- IPC - Inter-Process Communication
  - conjunto de mecanismos de troca de informação entre múltiplas threads de um ou mais processos.
  - Necessidade de coordenar o uso de recursos (sincronização).

# Comunicação entre Processos (3)

4

- Ao fornecer mecanismos de IPC, o S.O implementa “canais” de comunicação (implícitos ou explícitos) entre processos.



# Comunicação entre Processos (4)

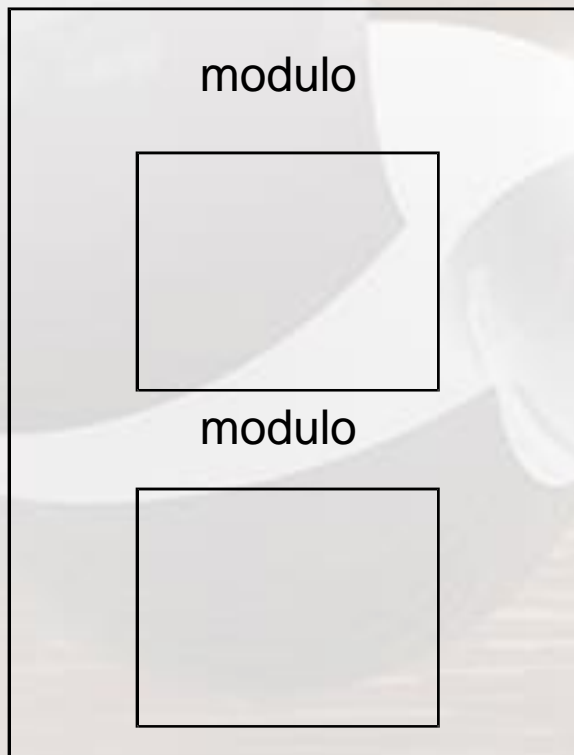
5

- Características desejáveis para IPC
  - Rápida
  - Simples de ser utilizada e implementada
  - Possuir um modelo de sincronização bem definido
  - Versátil
  - Funcione igualmente em ambientes distribuídos
  
- Sincronização é uma das maiores preocupações em IPC
  - Permitir que o sender indique quando um dado foi transmitido.
  - Permitir que um receiver saiba quando um dado está disponível.
  - Permitir que ambos saibam o momento em que podem realizar uma nova IPC.

# IPC x Comunicação Interna

6

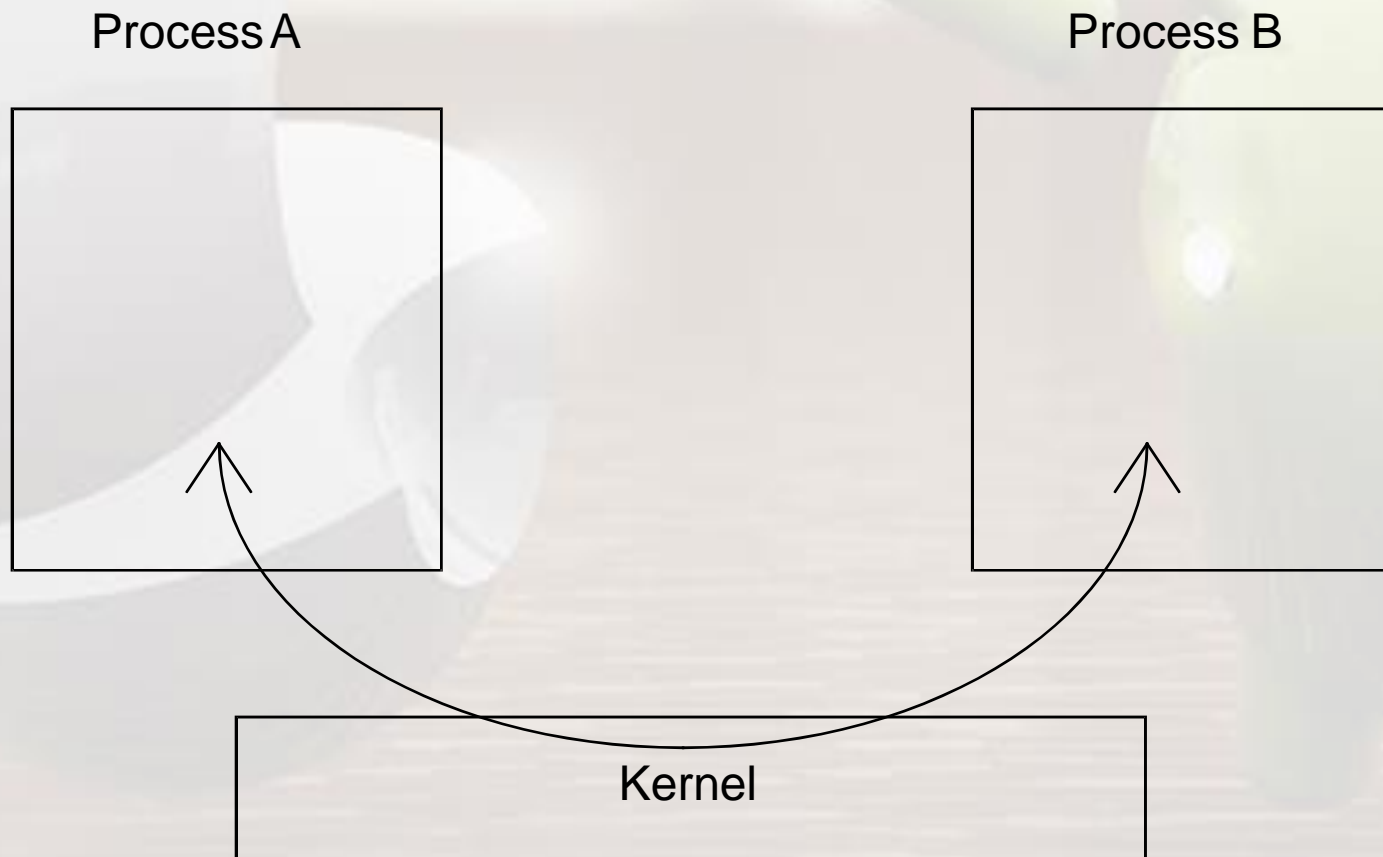
One Process



- Mecanismos de Comunicação Internos:
  - Variáveis globais
  - Chamadas de função
    - Parâmetros
    - resultados

# IPC – Um Computador

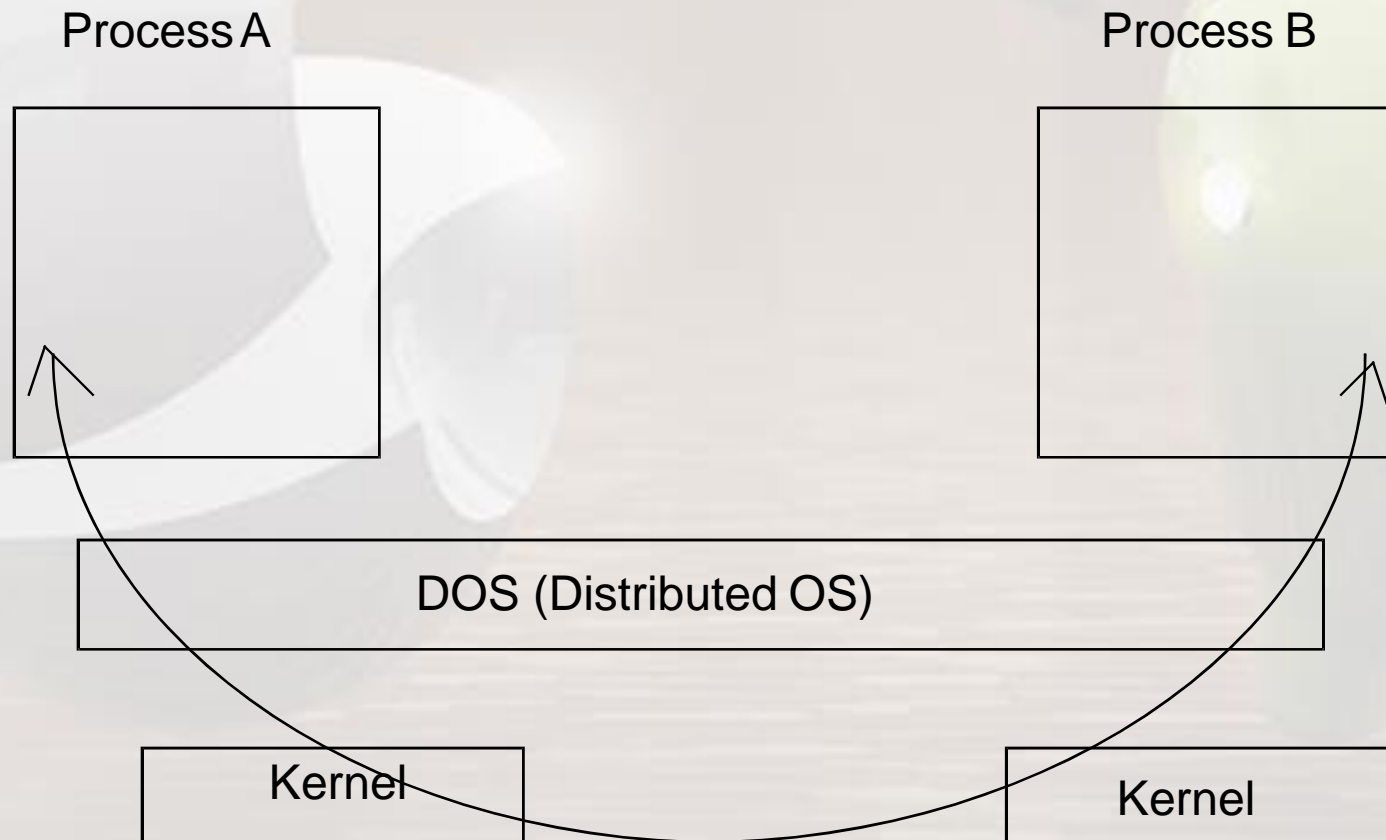
7





# IPC – Dois Computadores

8





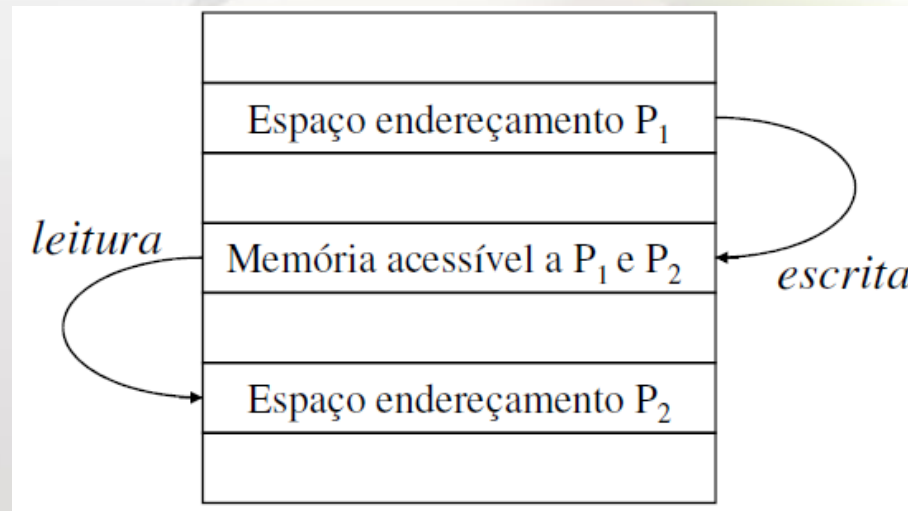
# Mecanismos de IPC

9

- Fundamentalmente, existem duas abordagens:
  - Suportar alguma forma de espaço de endereçamento compartilhado.
    - Shared memory (memória compartilhada)
  - Utilizar comunicação via núcleo do S.O., que ficaria então responsável por transportar os dados de um processo a outro. São exemplos:
    - Pipes e Sinais (ambiente centralizado)
    - Troca de Mensagens (ambiente distribuído)
    - RPC – Remote Procedure Call (ambiente distribuído)

# Comunicação via Memória Compartilhada

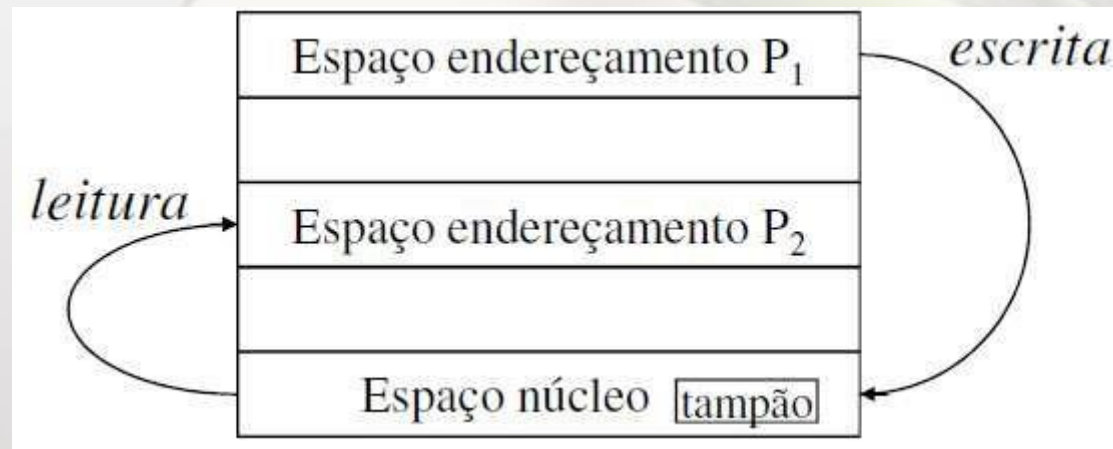
10



- Vantagens:
  - Mais eficiente (rápida), já que não exige a cópia de dados para alguma estrutura do núcleo.
- Inconveniente:
  - Problemas de sincronização.

# Comunicação via Núcleo

11



- Vantagens:
  - Pode ser realizada em sistemas com várias CPUs.
  - Sincronização implícita.
- Inconveniente:
  - Mais complexa e demorada (uso de recursos adicionais do núcleo).

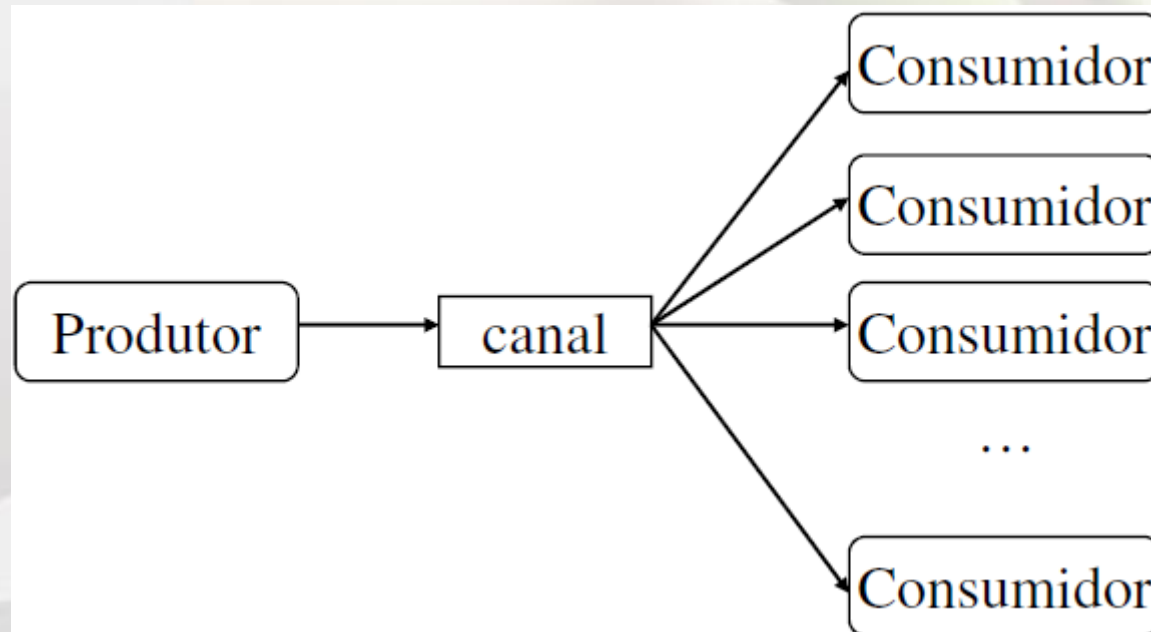
# Modelos de Comunicação

12

- Difusão (“broadcast”):
  - o emissor envia a mesma mensagem a todos os receptores.
- Produtor-consumidor:
  - comunicação uni-direcional.
- Cliente-servidor:
  - cliente controla totalmente o servidor.
- Caixa de correio (mailbox):
  - mensagens lidas por um processo receptor sem que o emissor (um entre vários) possa controlar quem recolhe a mensagem.
- Diálogo:
  - dois processos recebem um canal temporário para troca de mensagens durante uma sessão.

# Modelos de Comunicação: Difusão

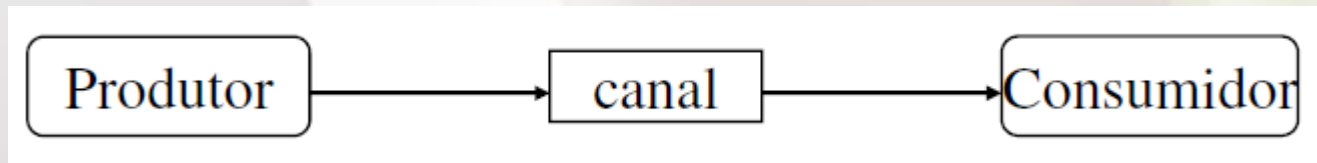
13



- ❑ O produtor envia mensagem a todos os consumidores, sem saber quem e quantos são.
- ❑ Comunicação "broadcast".

# Modelos de Comunicação: Produtor-Consumidor

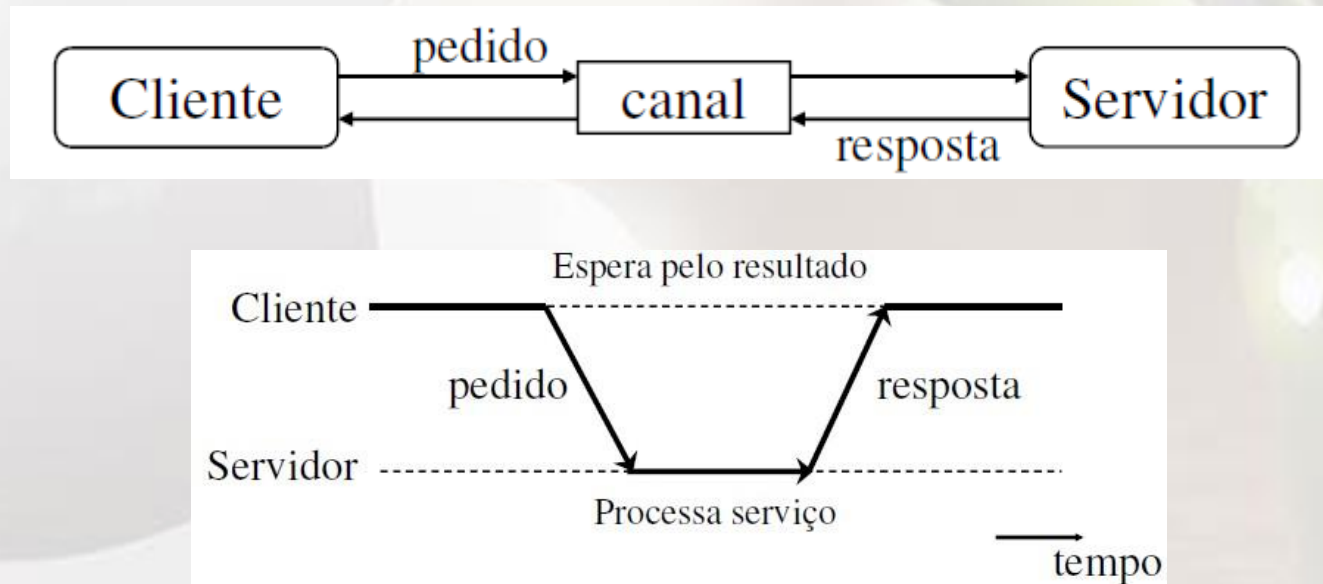
14



- ❑ Conexão unidirecional fixa, do produtor para o consumidor.
- ❑ Comunicação "Unicast".

# Modelos de Comunicação: Cliente-Servidor

15

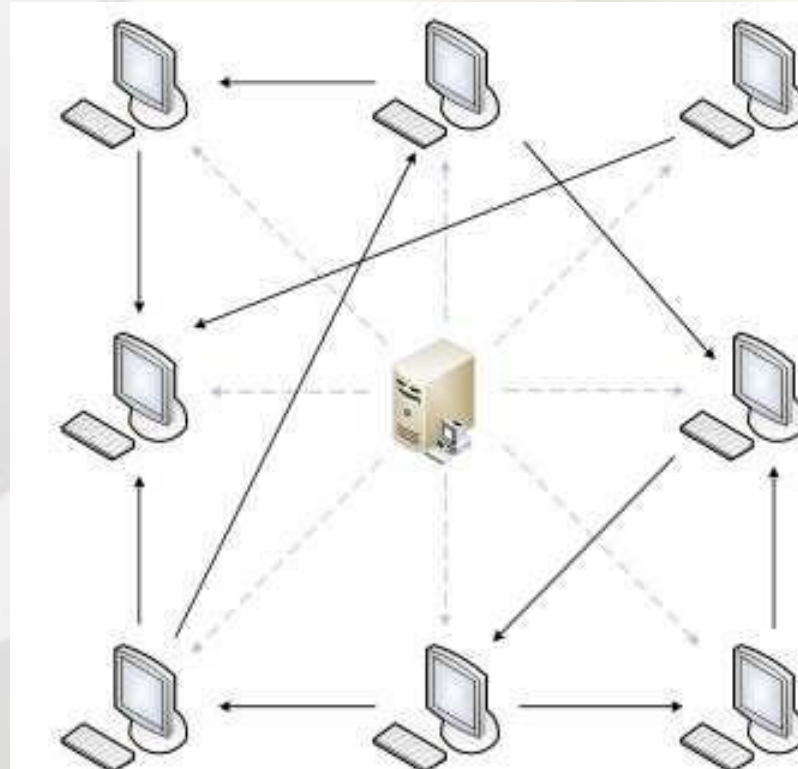


- Conexão bi-direcional fixa, entre o cliente (computador, programa ou processo que requer a informação) e o servidor (computador, programa ou processo que disponibiliza determinado serviço ao cliente).



# Modelo de Comunicação: Peer-to-Peer (P2P)

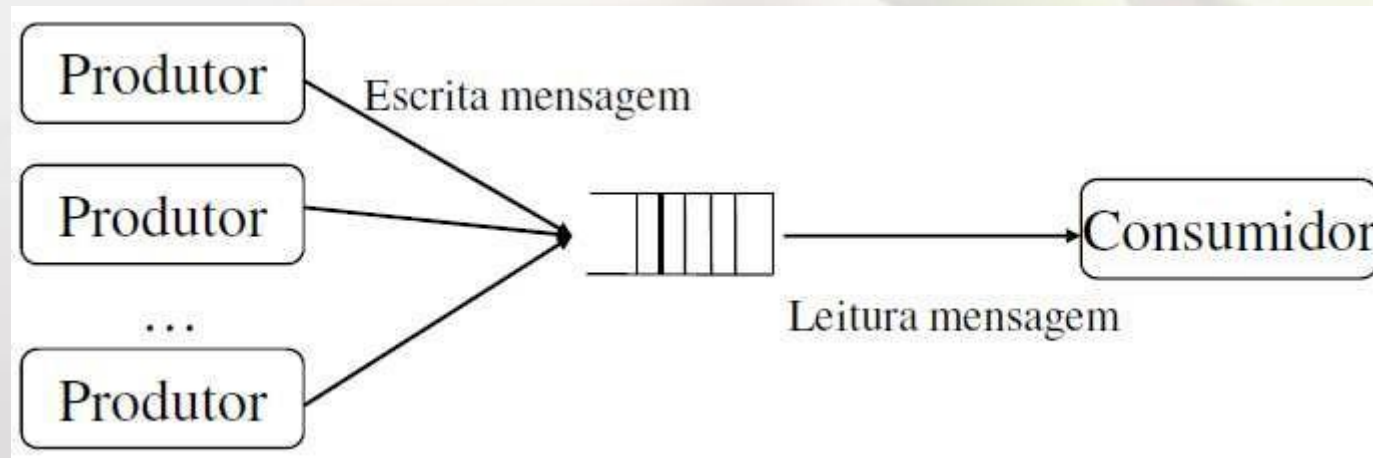
16



- No modelo P2P cada nó realiza, simultaneamente, tanto funções de servidor quanto de cliente.

# Modelo de Comunicação: Mailbox

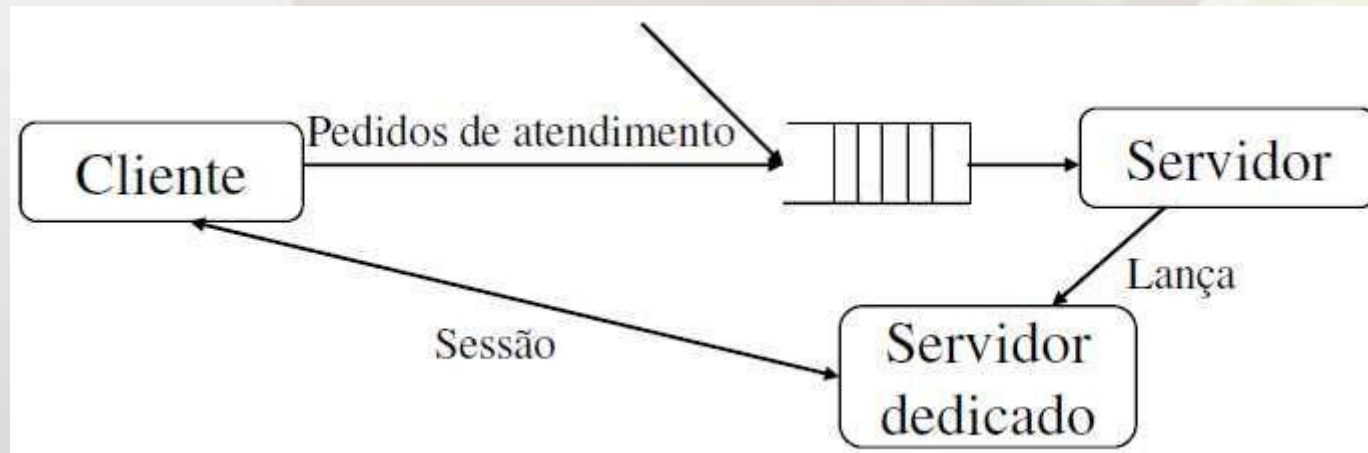
17



- A ligação entre produtor e consumidor é indireta, via caixa do correio (mailbox).
  - O consumidor não escolhe o produtor que escreveu a mensagem.
  - Tal como no modelo produtor-consumidor, a escrita não é bloqueante (admitindo uma caixa de correio de capacidade ilimitada) e a leitura é bloqueante quando a caixa se encontra vazia.

# Modelo de Comunicação: Diálogo

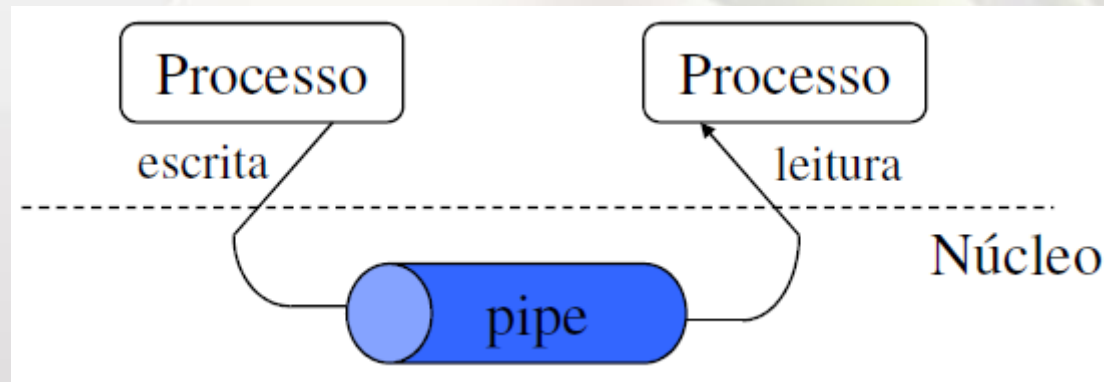
18



- ❑ Criado um servidor dedicado para cada cliente, com ligação por canal dedicado.
- ❑ O canal é desligado quando a sessão termina.

# Tubos (Pipes) <sup>(1)</sup>

19



- ❑ No UNIX, os *pipes* constituem o mecanismo original de comunicação unidirecional entre processos.
- ❑ São um mecanismo de I/O com duas extremidades, correspondendo, na verdade, a filas de caracteres tipo FIFO.
- ❑ As extremidades são implementadas via descritores de arquivos (vide adiante).

# Tubos (Pipes) (2)

20

- Um *pipe* tradicional caracteriza-se por ser:
  - Anônimo (não tem nome).
  - Temporário: dura somente o tempo de execução do processo que o criou.
- Vários processos podem fazer leitura e escrita sobre um mesmo *pipe*, mas nenhum mecanismo permite diferenciar as informações na saída do *pipe*.
- A capacidade do *pipe* é limitada
  - Se uma escrita é feita e existe espaço no *pipe*, o dado é colocado no *pipe* e a chamada retorna imediatamente.
  - Se a escrita sobre um *pipe* continua mesmo depois dele estar cheio, ocorre uma situação de bloqueio (que permanece até que algum outro processo leia e, conseqüentemente, abra espaço no *pipe*).

# Tubos (Pipes) (3)

21

- É impossível fazer qualquer movimentação no interior de um pipe.
- Com a finalidade de estabelecer um diálogo entre dois processos usando pipes, é necessário a abertura de um pipe em cada direção.



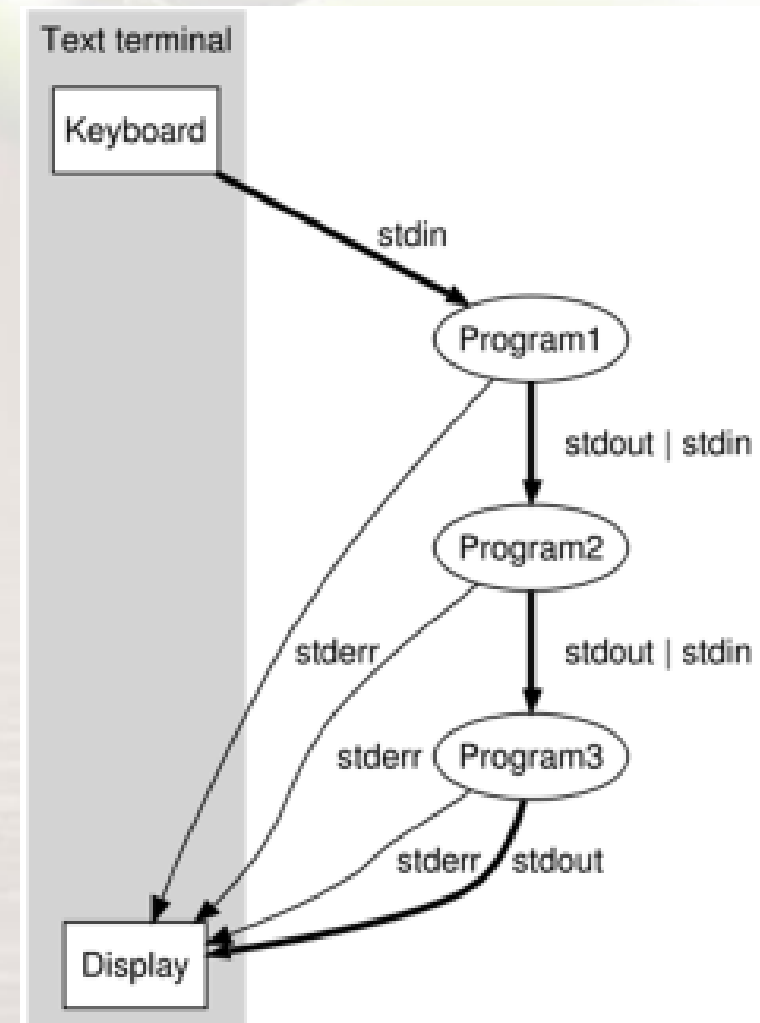
# Uso de Pipes

22

- `who | sort | lpr`
  - ▣ + output of *who* is input to *sort*
  - ▣ + output of *sort* is input to *lpr*

```
curl "http://en.wikipedia.org/wiki/Pipeline_(Unix)" | \  
sed 's/[^a-zA-Z ]/ /g' | \  
tr 'A-Z' 'a-z\n' | \  
grep '[a-z]' | \  
sort -u | \  
comm -23 - /usr/dict/words
```

(1) **curl** obtains the HTML contents of a web page. (2) **sed** removes all characters which are not spaces or letters from the web page's content, replacing them with spaces. (3) **tr** changes all of the uppercase letters into lowercase and converts the spaces in the lines of text to newlines (each 'word' is now on a separate line). (4) **grep** removes lines of whitespace. (5) **sort** sorts the list of 'words' into alphabetical order, and removes duplicates. (6) Finally, **comm** finds which of the words in the list are not in the given dictionary file (in this case, /usr/dict/words).





# Criação de Pipes (1)

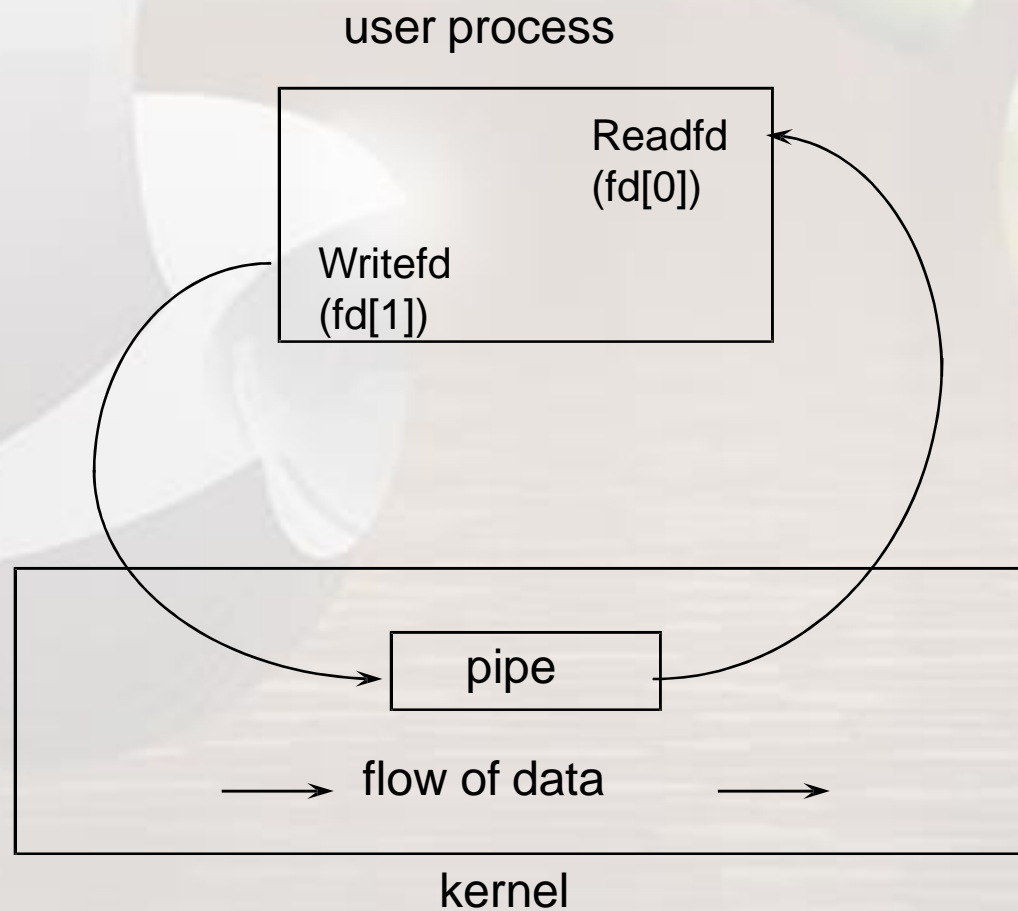
23

- Pipes constituem um canal de comunicação entre processos pai-filho.
  - Os pipes são definidos antes da criação dos processos descendentes.
  - Os pipes podem ligar apenas processos com antepassado comum.
- Um pipe é criado pela chamada de sistema:
  - POSIX:

```
#include <unistd.h>
int pipe(int fd[2])
```
- São retornados dois descritores:
  - Descritor `fd[0]` - aberto para leitura
  - Descritor `fd[1]` - aberto para escrita.

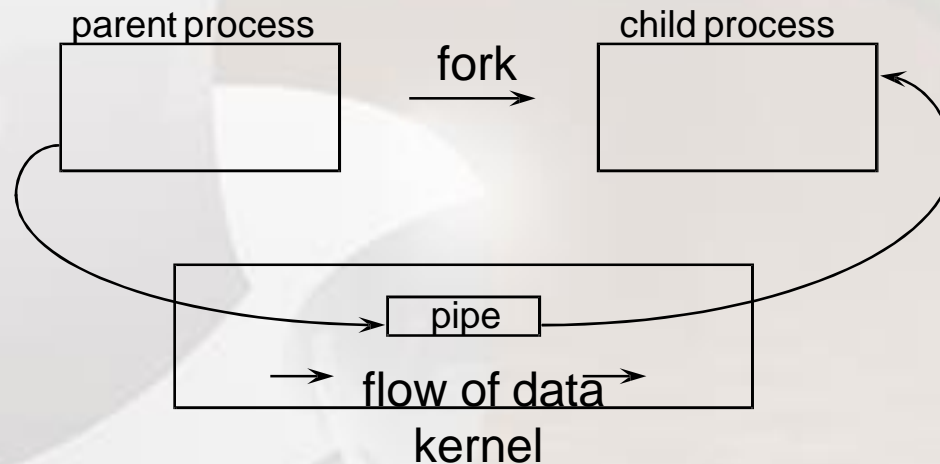
# Criação de Pipes (2)

24



# Criação de Pipes (3)

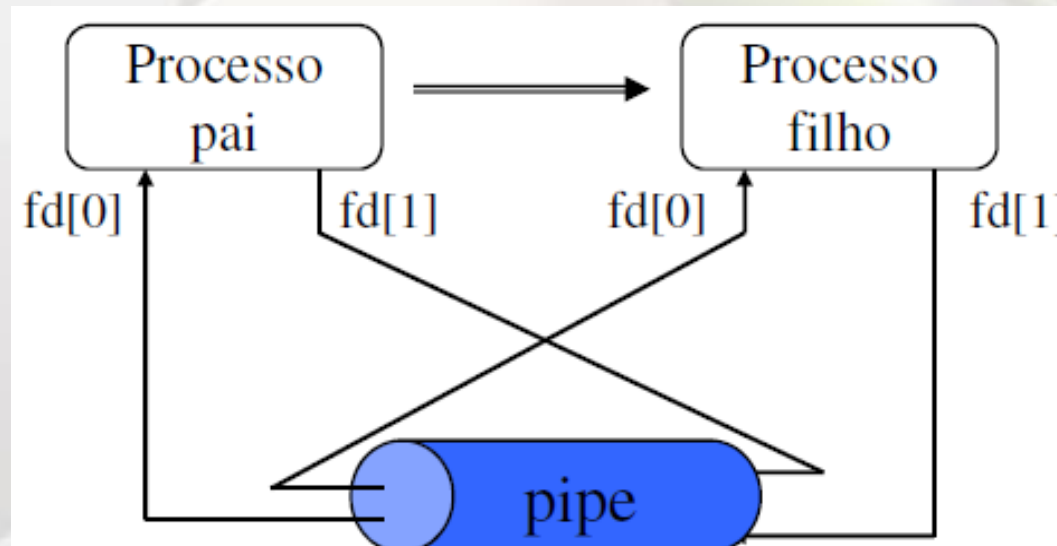
25



- Um *pipe* criado em um único processo é quase sem utilidade. Normalmente, depois do *pipe*, o processo chama `fork()`, criando um canal e comunicação entre pai e filho.

# Criação de Pipes (4)

26



- Quando um processo faz um `fork()` depois de criado o *pipe*, o processo filho recebe os mesmos descritores de leitura e escrita do pai. Cada um dos processos deve fechar a extremidade não aproveitada do *pipe*.

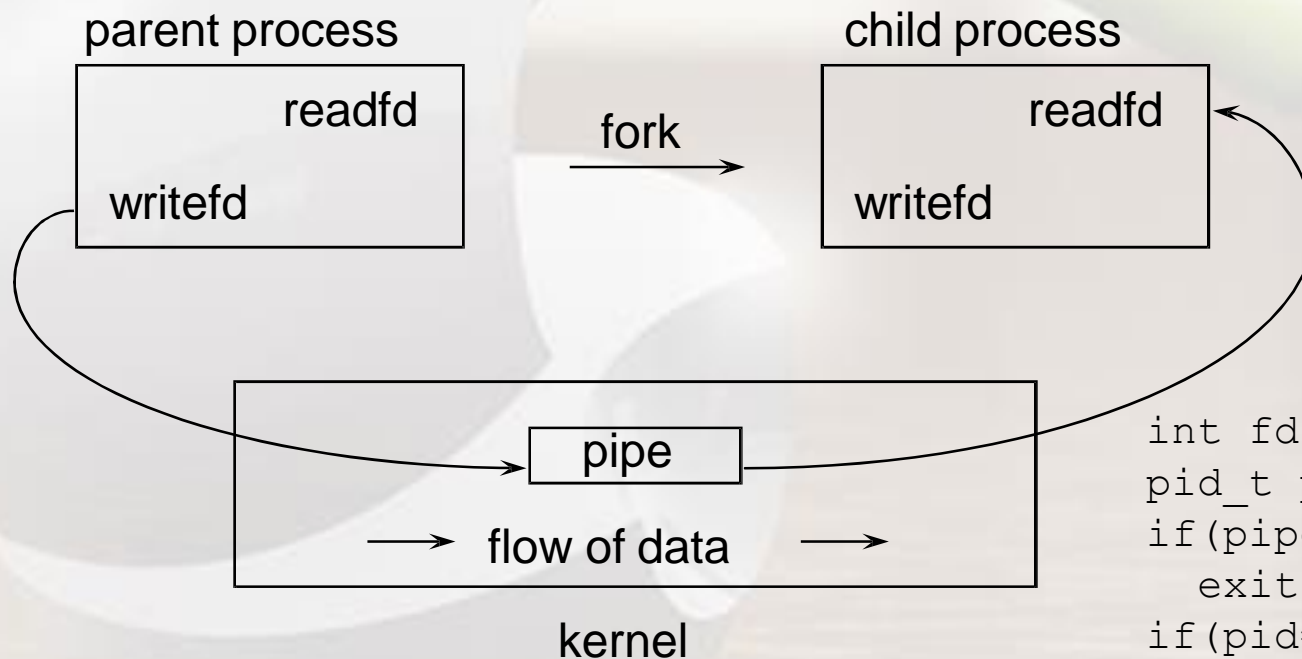
# Fechamento de Pipes

- Depois de usados, ambos os descritores devem ser fechados pela chamada do sistema:
  - POSIX:  
`#include <unistd.h>`  
`int close (int);`
- Quando todos os descritores associados a um pipe são fechados, todos os dados residentes no pipe são perdidos.
- Em caso de sucesso retorna 0 , Em caso de erro retorna -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]);
}
```

# Comunicação Pai-Filho Unidirecional

28

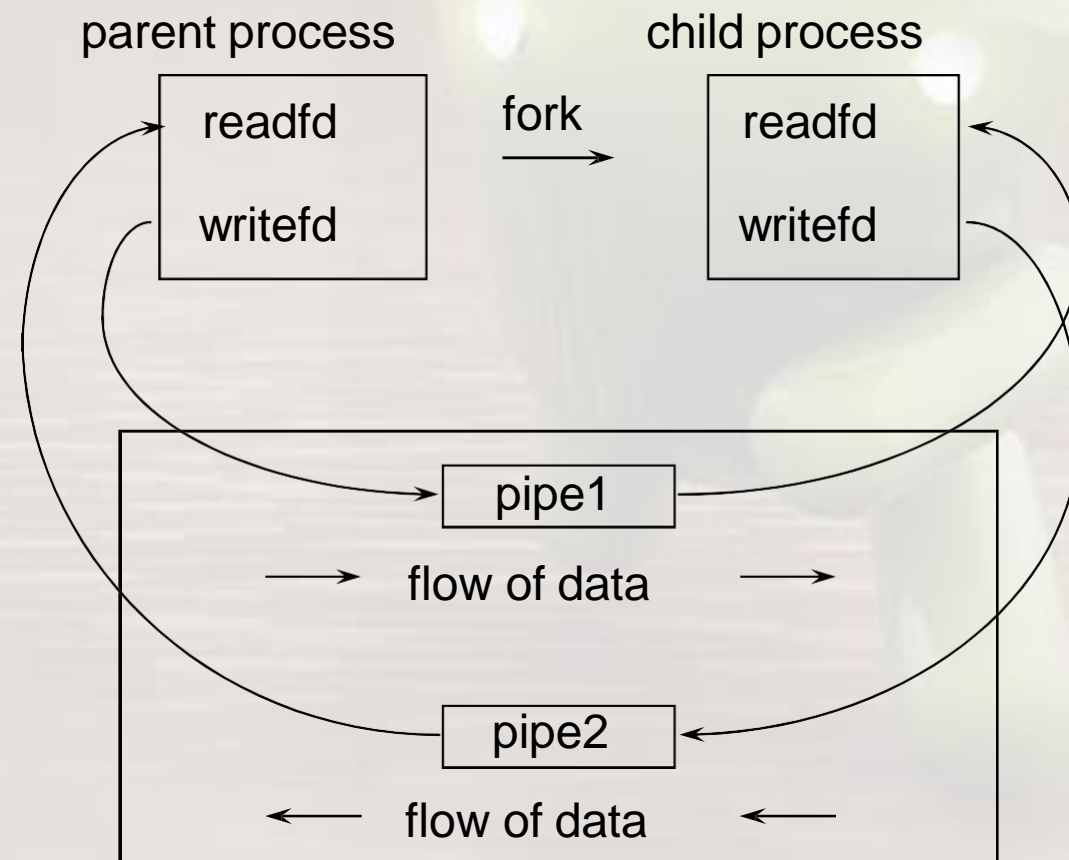


- ❑ Processo pai cria o pipe.
- ❑ Processo pai faz o fork().
- ❑ Os descritores são herdados pelo processo filho.
- ❑ Pai fecha fd[0]
- ❑ Filho fecha fd[1]

```
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] );
    ...
}
```

# Comunicação Pai-Filho Bi-Direcional

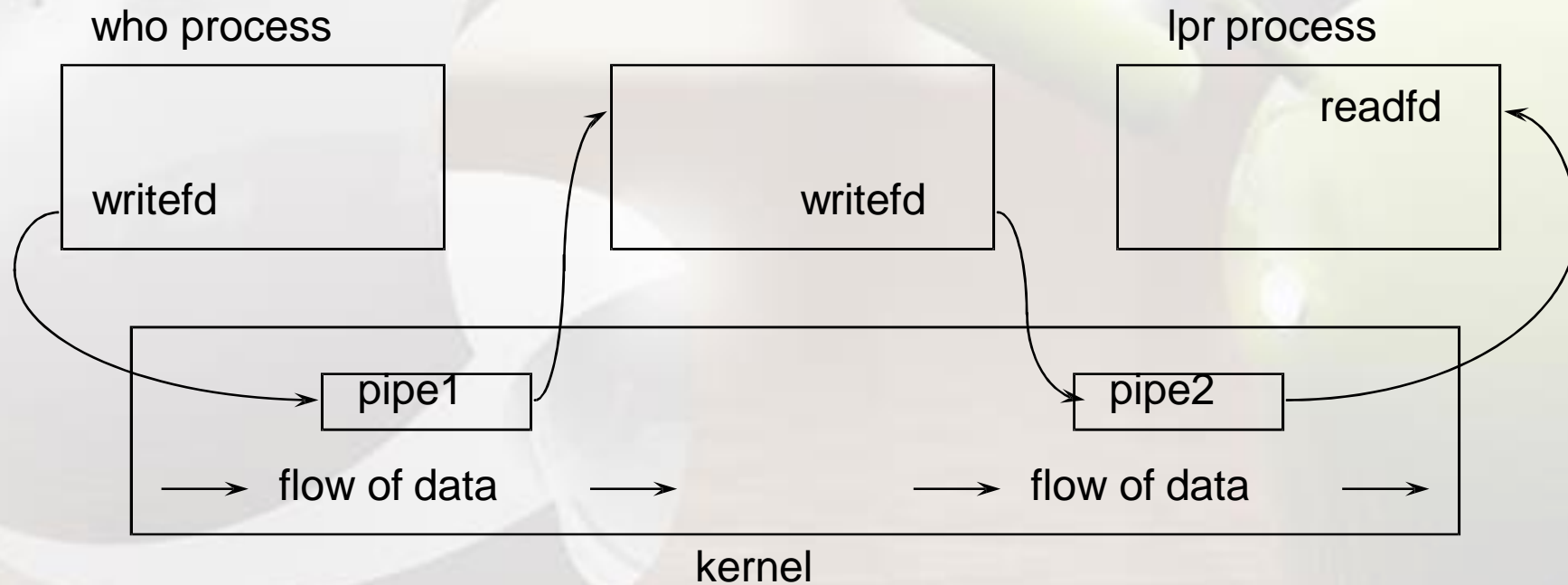
- Ex: pai envia filename para o filho. Filho abre e lê o arquivo, e retorna o conteúdo para o pai.
  - Pai cria pipe1 e pipe2.
  - Pai fecha descritor de leitura de pipe1.
  - Pai fecha descritor de escrita de pipe2.
  - Filho fecha descritor de escrita de pipe1.
  - Filho fecha descritor de leitura de pipe2.





# who | sort | lpr

30



- ❑ Processo *who* escreve no pipe1.
- ❑ Processo *sort* lê do pipe1 e grava no pipe2.
- ❑ Processo *lpr* lê do pipe2.

# Escrita e Leitura em Pipes (1)

31

- A comunicação de dados em um *pipe* (leitura e escrita) é feita pelas seguintes chamadas de sistema:
  - POSIX:

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

    - 1º parâmetro: descritor de arquivo.
    - 2º parâmetro: endereço dos dados.
    - 3º parâmetro: número de bytes a comunicar.
  - A função retorna o número de bytes efetivamente comunicados.
    - Nota1: O número de bytes que podem ser temporariamente armazenados por um *pipe* é indicado por `_POSIX_PIPE_BUF` (512B, definido em `<limits.h>`).
    - Nota2: Envio direto 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!

# Escrita e Leitura em Pipes (2)

32

- Regras aplicadas aos processos escritores:
  - Escrita para descritor fechado resulta na geração do sinal `SIGPIPE`
  - Escrita de dimensão inferior a `_POSIX_PIPE_BUF` é atômica (i.e., os dados não são entrelaçados).
  - No caso do pedido de escrita ser superior a `PIPE_BUF`, os dados podem ser entrelaçados com pedidos de escrita vindos de outros processos.
    - O número de bytes que podem ser temporariamente armazenados por um *pipe* é indicado por `_POSIX_PIPE_BUF` (512B, definido em `<limits.h>`).
- Regras aplicadas aos processos leitores:
  - Leitura para descritor fechado retorna valor 0.
  - Processo que pretende ler de um *pipe* vazio fica bloqueado até que um processo escreva os dados.

# Exemplo 1

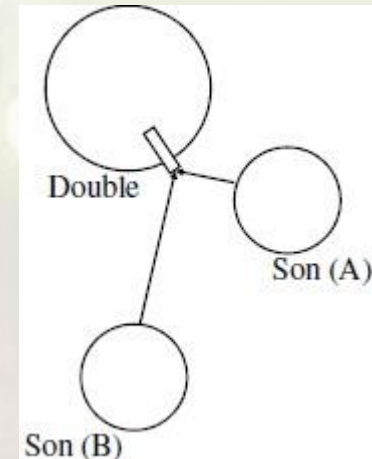
33

- Processo filho envia dados para o processo pai.
  - ▣ exemplo1.c

# Exemplo 2

34

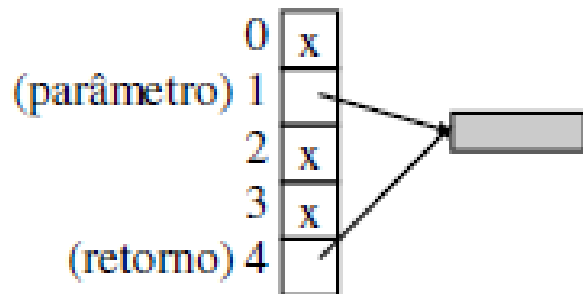
- Dois processos filhos enviam mensagens para o processo pai.
  - ▣ defs.h
  - ▣ son.c
  - ▣ double.c



# Redirecção E/S por pipes (1)

35

- 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 arquivo duplicado.
  - O valor retornado identifica um novo descritor para o arquivo
    - Nota: o arquivo não é duplicado, apenas o descritor!. O valor devolvido é o primeiro índice vago da tabela de descritores.



# Redirecção E/S por pipes (2)

- Os dois descritores partilham a mesma posição do ficheiro.
  - Se fosse efetuado open duas vezes sobre o mesmo arquivo, cada descritor é posicionado em locais distintos!
- A função dup2 duplica descritor de ficheiro, fechando previamente o primeiro descritor
  - POSIX: 

```
#include <unistd.h>
```

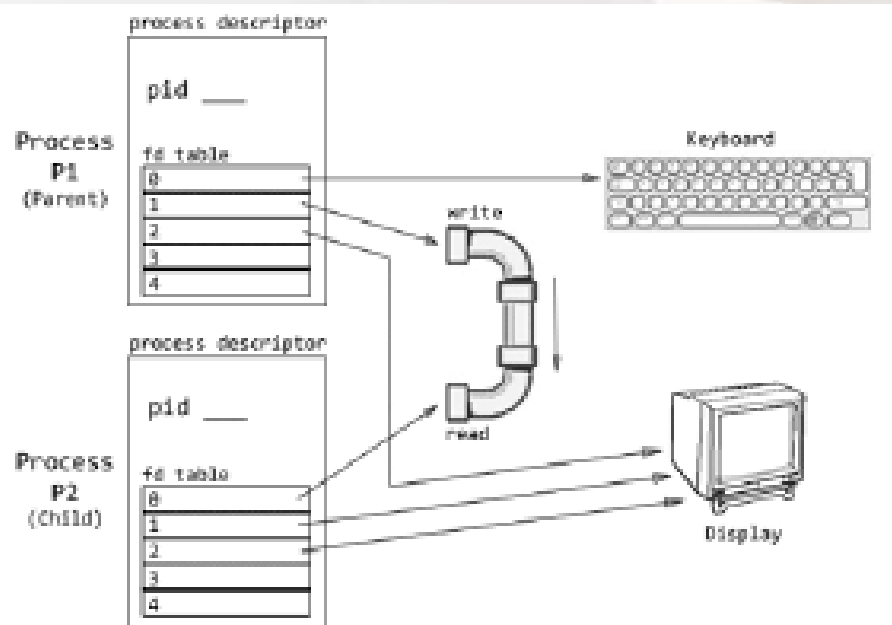
```
int dup2(int, int);
```

    - O 2º parâmetro é o descritor fechado.
    - `dup2(newD, d);` é equivalente a `close(d); dup(newD);`



# Exemplo 3

## □ exemploSaidaPadrao.c



- Usando-se a técnica de IPC pipes, pode-se implementar o comando `"ls | wc"`. Resumidamente:
  - (i) cria-se um pipe;
  - (ii) executa-se um `fork`;
  - (iii) o processo pai chama `exec` para executar `"ls"`;
  - (iv) o processo filho chama `exec` para executar `"wc"`.
- O problema é que normalmente o comando `"ls"` escreve na saída padrão 1 e `"wc"` lê da entrada padrão 0.
  - Como então associar a saída padrão com a saída de um pipe e a entrada padrão com a entrada de um pipe? Isso pode ser conseguido através da chamada de sistema `int dup2(int newfd, int oldfd)`.
- Essa chamada cria uma cópia de um descritor de arquivo existente (`oldfd`) e fornece um novo descritor (`newfd`) tendo exatamente as mesmas características que aquele passado como argumento na chamada. A chamada `dup2` fecha antes `newfd` se ele já estiver aberto.

# Exemplo 4

38

- exemplo4.c
  - ▣ O que faz esse programa?

# Fila (FIFO, Named Pipe)

39

- Trata-se de uma extensão do conceito de pipe.
  - Pipes só podem ser usados por processos que tenham um ancestral comum.
  - Filas (FIFOs – First In First Out), também designados de “tubos nomeados” (*“named pipes”*), permitem a comunicação entre **processos não relacionados**.
- As Filas:
  - são referenciadas por um identificador dentro do sistema de arquivos
  - persistem além da vida do processo
  - são mantidas no sistema de arquivos até serem apagadas (ou seja, precisam ser eliminadas quando não tiverem mais uso).
- Normalmente são implementadas através de arquivos especiais (tipo: pipe).
  - Um processo abre a Fila para escrita, outro para leitura.

# Criação de Filas (1)

40

- Uma fila é criada pela chamada de sistema:
  - POSIX: `#include <sys/stat.h>`  
`int mkfifo(char *, mode_t);`
    - 1º parâmetro: nome do arquivo.
    - 2º parâmetro: identifica as permissões de acesso, iguais a qualquer arquivo, determinados por OU de grupos de bits.
- As permissões de acesso também podem ser indicados por 3 dígitos octais, cada um representando os valores binários de rwx (*Read, Write, eXecute*).
  - Exemplo: modo 644 indica permissões de acesso:
    - Dono: 6 = 110 (leitura e escrita)
    - Grupo e Outros: 4 = 100 (leitura)

# Criação de Filas (2)

- Uma fila também pode ser criada, via shell, por meio do comando:  
`#mkfifo [-m modo] fichID`
- Exemplo 1:  
`mkfifo -m 644 tubo`  
`ls -l tubo`  
`prw-r--r-- 1 rgc docentes 0 2008-10-11 15:56 tubo`
  - OBS: p indica que "tubo" é um arquivo do tipo named pipe
- Exemplo 2:  
`mkfifo teste`  
`cat < teste /* o pipe fica esperando até obter algum dado */`
  - Em outra tela execute:  
`ls > teste /* a saída do comando ls será redirecionada para o pipe nomeado "teste" */`

# Eliminação de Filas

42

- Uma fila é eliminada pela seguinte chamada ao sistema:
  - POSIX: 

```
#include <unistd.h>
```

```
int unlink(char *);
```

    - 1º parâmetro: nome do arquivo.
- Uma fila também é eliminada via shell, usando o comando:

```
rm fichID
```

# Abertura de Filas (1)

43

- Antes de ser usada, a fila tem de ser aberta pela chamada de sistema:
  - POSIX: 

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(char *,int);
```
- 1º parâmetro: nome do arquivo.
- 2º parâmetro : formado por bits que indicam:
  - Modos de acesso: `O_RDONLY` (leitura apenas) ou `O_WRONLY` (escrita apenas)
  - Opções de abertura: `O_CREAT` (criado se não existir)
  - `O_NONBLOCK` (operação de E/S não são bloqueadas)
- O valor de retorno é o descritor da fila (positivo) ou erro (-1).

# Abertura de Filas (2)

44

- Regras aplicadas na abertura de filas:
  - Se um processo tentar abrir uma fila em modo de leitura, e nesse instante não houver um processo que tenha aberto a fila em modo de acesso de escrita, o processo fica bloqueado, exceto se:
    - a opção `O_NONBLOCK` tiver sido indicada no `open()` (nesse caso, é devolvido o valor -1 e `errno` fica com valor `ENXIO`).
  - Se um processo tentar abrir uma fila em modo de escrita, e nesse instante não houver um processo que tenha aberto a fila em modo de acesso de leitura, o processo fica bloqueado, exceto se:
    - a opção `O_NONBLOCK` tiver sido indicada no `open()` (nesse caso, é devolvido o valor -1 e `errno` fica com valor `ENXIO`).



# Leitura e Escrita em Filas (1)

45

- A comunicação em uma fila é feita pelas mesmas chamadas de sistema dos pipes:
  - POSIX: `#include <unistd.h>`  
`ssize_t read(int, char *,int);` `ssize_t`  
`write(int, char *,int);`
- Regras aplicadas aos processos escritores:
  - Escrita para uma fila que ainda não foi aberta para leitura gera o sinal `SIGPIPE` (ação por omissão de terminar o processo. Se ignorado `read` retorna -1 com `errno` igual a `EPIPE`).
  - Após o último processo escritor tiver encerrado a fila, os processos leitores recebem `EOF`.

# Exemplo

46

- Dois processos writer enviam mensagens para o processo reader através de uma fila.
  - reader.c
  - writer.c
- O identificador da fila e o comprimento da memória tampão (buffer) é definida no arquivo à parte.

```
#define LEN 100  
#define FNAME "testFIFO"
```

- `./reader & ./writer & ./writer &`
- `ls -l`
  - Observe que a fila não havia sido eliminada pelos programas (arquivo `testFIFO` tem tipo `p`, de named pipe).

# Referências

- Slides adaptados de Roberta Lima Gomes (UFES)
- Bibliografia
  - Kay A. Robbins, Steven Robbins, *UNIX systems programming: communication, concurrency, and threads*. Prentice Hall Professional, 2003 - 893 pages
    - Capítulo 6.2