

# Sistemas Hardware-Software

Aula 20 – Redirecionamento de arquivos

2020 – Engenharia

Igor Montagner , Maciel Calebe, Fábio Ayres

# Avaliação docente

- 10 minutos
- Acessar <https://insper.avaliar.org/>
- Código: 51436
- Chave: 946670

# Aulas passadas

- Entrada e saída
- Permissões e sistema de arquivos
- Criação e gerenciamento de processos
- Sinais

# E/S padrão

Todo processo criado por um shell Linux já vem com três arquivos abertos, e associados com o terminal:

0: standard input (stdin)

1: standard output (stdout)

2: standard error (stderr)

# Abrindo arquivos

```
int open(const char *pathname, int flags,  
         mode_t mode);
```

- Retorna um inteiro chamado *file descriptor*.
- flags indicam opções de abertura de arquivo
  - O\_RDONLY, O\_WRONLY, O\_RDWR
  - O\_CREATE (cria se não existir)
  - O\_EXCL + O\_CREATE (se existir falha)
- mode indica as permissões de um arquivo criado usando open.

# Fechando um arquivo

Fechar um arquivo informa ao kernel que você já terminou de acessar o arquivo.

```
int fd;      /* file descriptor */
int retval; /* return value */

if ((retval = close(fd)) < 0) {
    perror("close");
    exit(1);
}
```

Cuidado: não feche um arquivo já fechado!

# Lendo/escrevendo em um arquivo

```
ssize_t read(int fd, void *buf, size_t count);
```

```
ssize_t write(int fd, const void *buf, size_t count);
```

- Cada chamadas lê/escreve no máximo count bytes apontados por buf no arquivo fd.
- Ambas retornam o número de bytes lidos/escritos e -1 se houver erro.
- Se read retornar 0 acabou o arquivo.

# Hoje

- Redirecionamento de arquivos
- Comunicação (simples) entre processos



# Tabela de descritores de arquivos

## Descriptor table

[one table per process]

stdin	fd 0	
stdout	fd 1	
stderr	fd 2	
	fd 3	
	fd 4	

## Open file table

[shared by all processes]

### File A (disk)

File pos
refcnt=1
...

### File B (disk)

File pos
refcnt=1
...

## v-node table

[shared by all processes]

File access
File size
File type
...

File access
File size
File type
...

Criados via open

# Tabela de descritores de arquivos

## Descriptor table

[one table per process]

stdin	fd 0	
stdout	fd 1	
stderr	fd 2	
	fd 3	
	fd 4	

## Open file table

[shared by all processes]

### File A (disk)

File pos
refcnt=1
...

### File A (disk)

File pos
refcnt=1
...

## v-node table

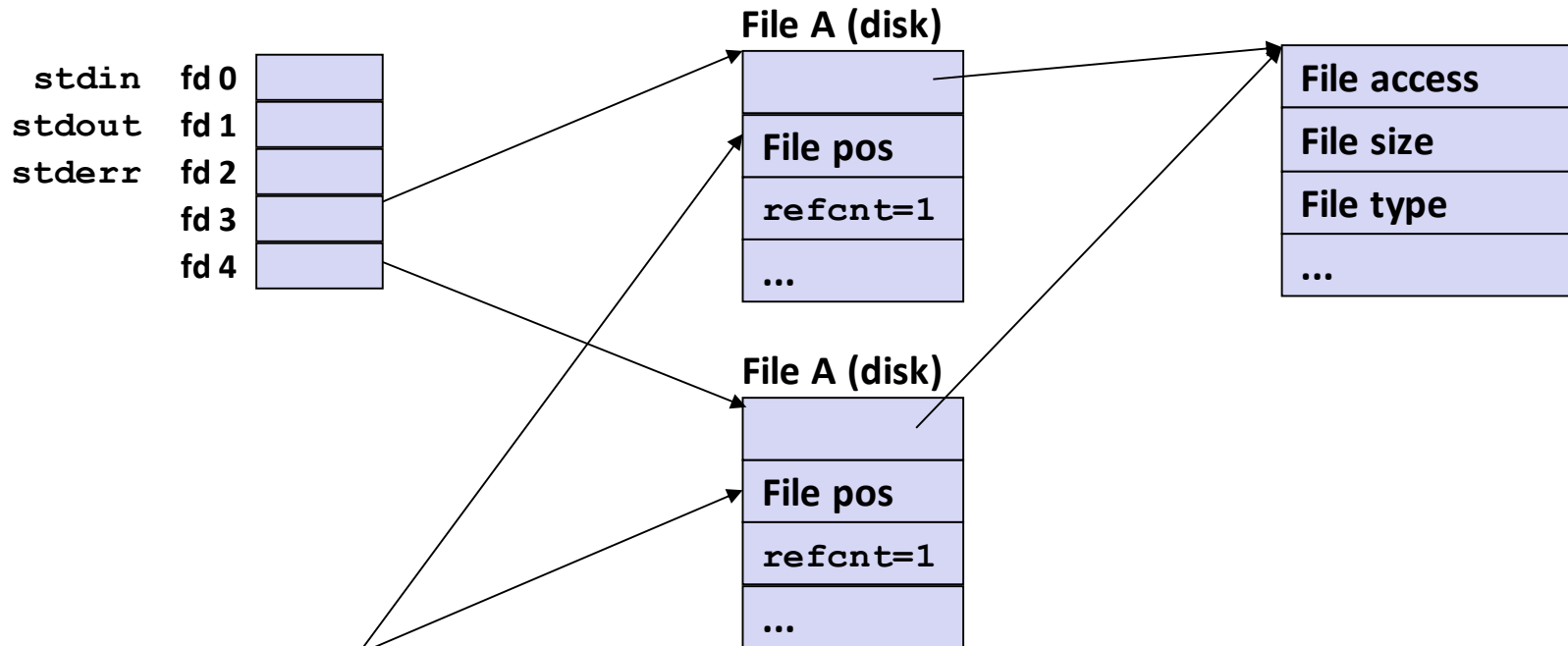
[shared by all processes]

File access
File size
File type
...

open chamado duas  
vezes com o mesmo  
arquivp

# Tabela de descritores de arquivos

**Descriptor table** [one table per process]      **Open file table** [shared by all processes]      **v-node table** [shared by all processes]



Cada um tem sua posição de leitura/escrita atual!

# Descritores de arquivos antes do fork

## Descriptor table

[one table per process]

stdin	fd 0	
stdout	fd 1	
stderr	fd 2	
	fd 3	
	fd 4	

Criados via `open`

## Open file table

[shared by all processes]

### File A (disk)

File pos
refcnt=1
...

### File B (disk)

File pos
refcnt=1
...

## v-node table

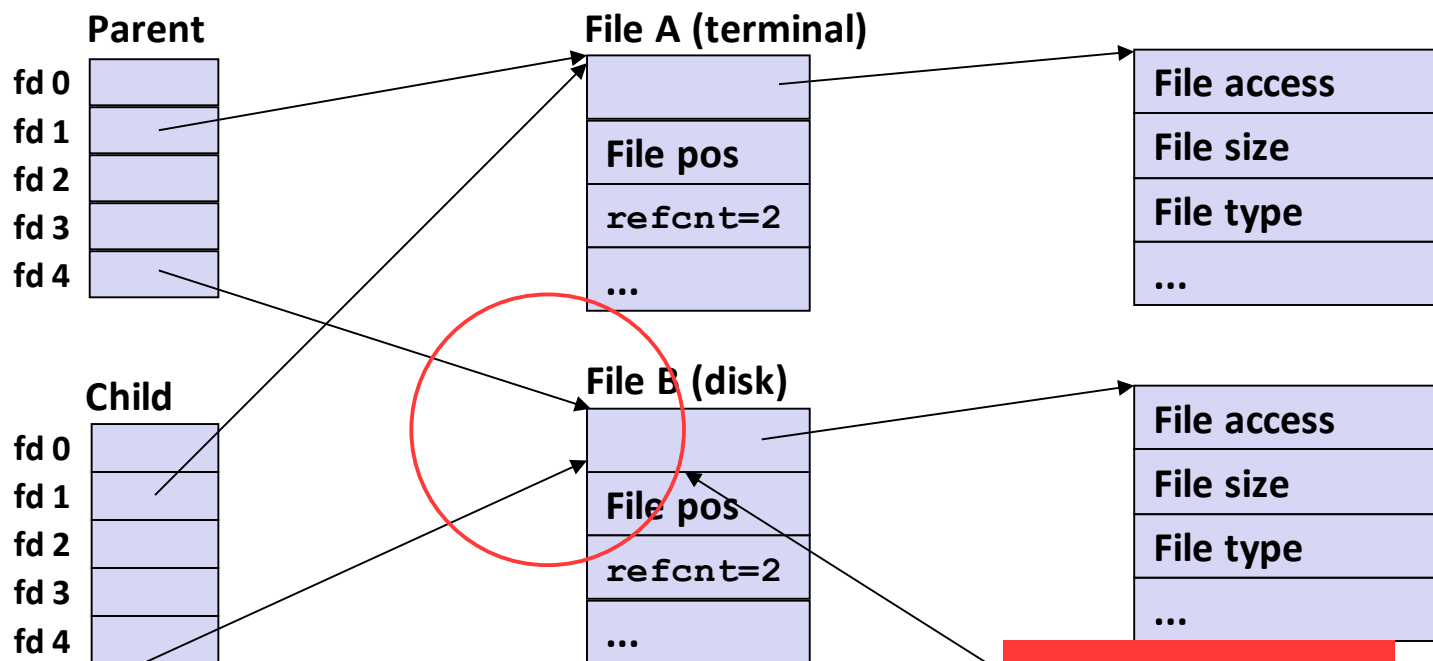
[shared by all processes]

File access
File size
File type
...

File access
File size
File type
...

# Descritores de arquivos depois do fork

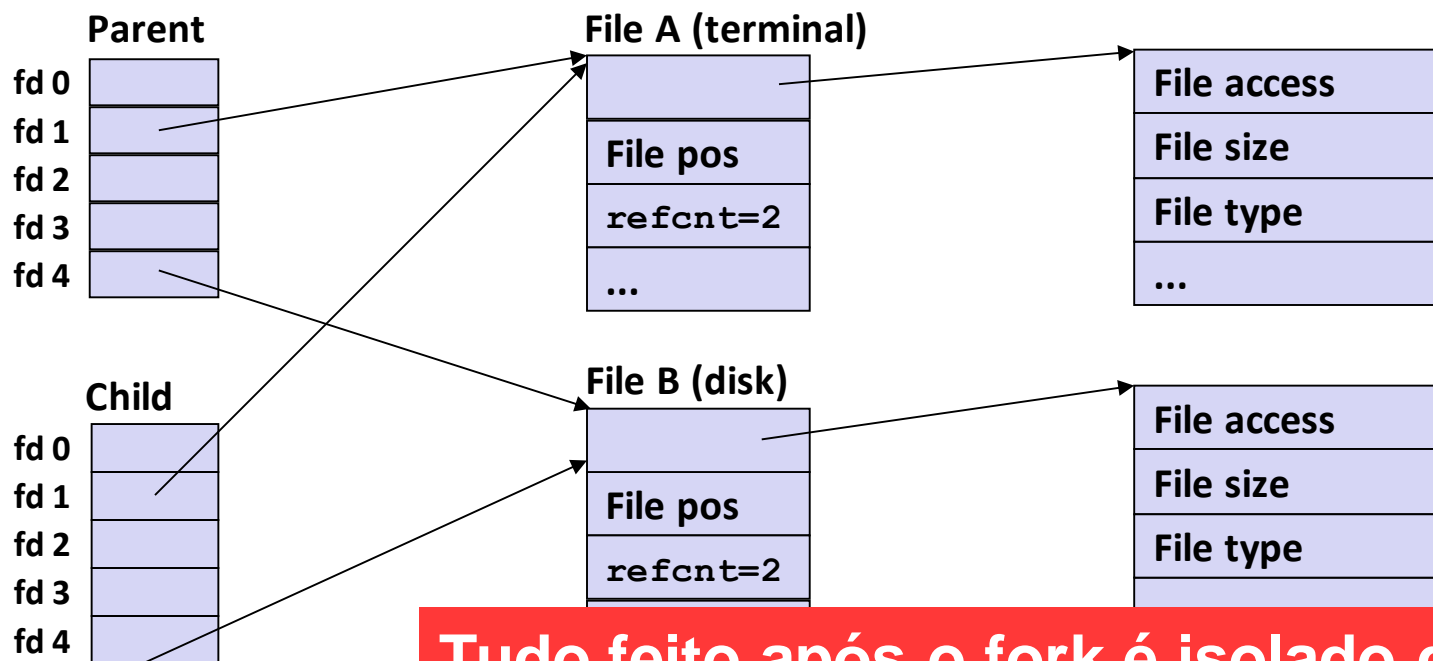
**Descriptor table** [one table per process]      **Open file table** [shared by all processes]      **v-node table** [shared by all processes]



Pai e filho  
compartilham o  
descritor!

# Descritores de arquivos depois do fork

**Descriptor table** [one table per process]      **Open file table** [shared by all processes]      **v-node table** [shared by all processes]



**Tudo feito após o fork é isolado de cada processo!**

# Descritores de arquivos e fork

- São mantidos após o fork e exec
- Acessos podem ter problemas de concorrência
- Podemos manipular os descritores de processos filhos

# Redirecionamento de I/O

`dup2 (oldfd, newfd)`

Copia o valor da posição `oldfd` para a posição `newfd` da tabela de descritores

Descriptor table

*before* `dup2 (4, 1)`

fd 0	
fd 1	a
fd 2	
fd 3	
fd 4	b



Descriptor table

*after* `dup2 (4, 1)`

fd 0	
fd 1	b
fd 2	
fd 3	
fd 4	b



# Redirecionamento de I/O

```
int d = dup(oldfd)
```

Cria um novo fd que aponta para o mesmo arquivo que oldfs

Descriptor table  
*before* dup (1)

fd 0	
fd 1	a
fd 2	



Descriptor table  
*after* dup (1)

fd 0	
fd 1	a
fd 2	
fd 3	a

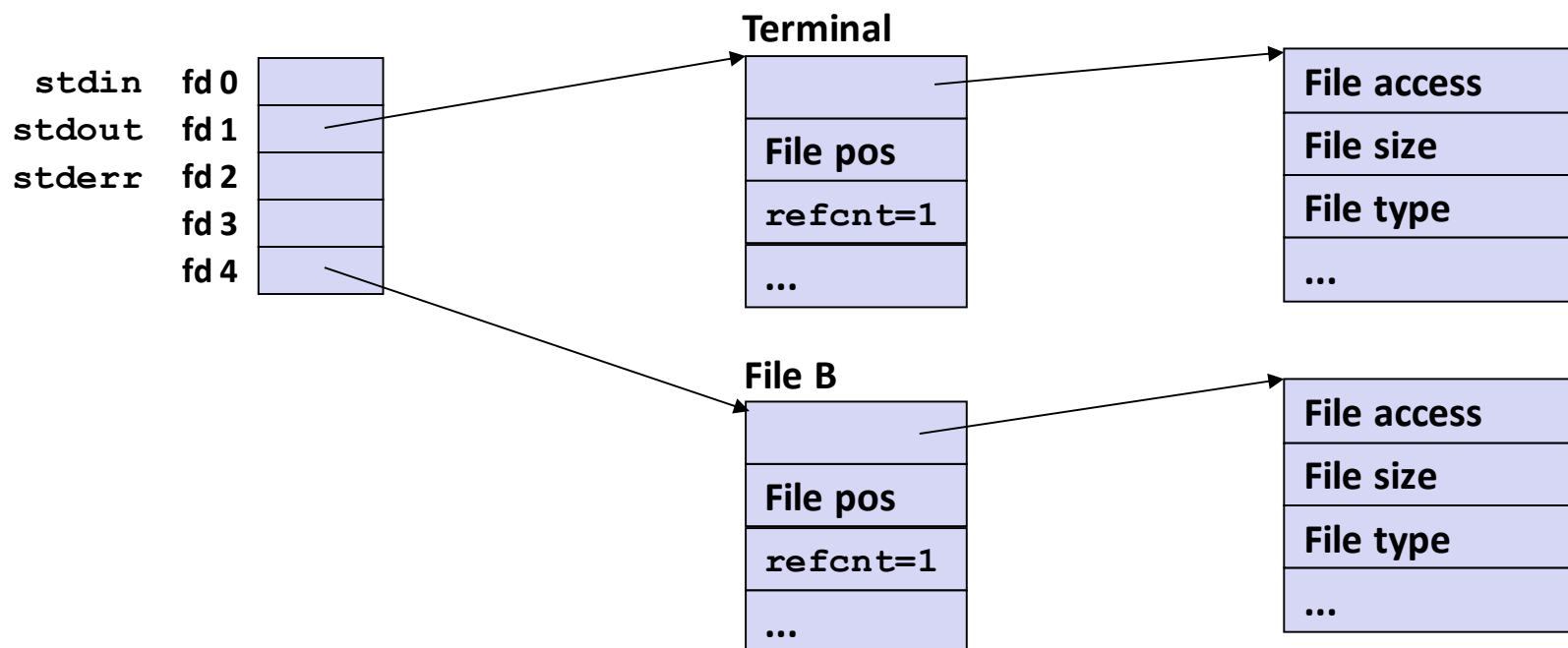
# Redirecionamento de saída

O que acontece quando fazemos `./programa > arquivo?`

# Redirecionamento de saída

1. Abrir o arquivo para o qual stdout será redirecionado

**Descriptor table** [one table per process]      **Open file table** [shared by all processes]      **v-node table** [shared by all processes]



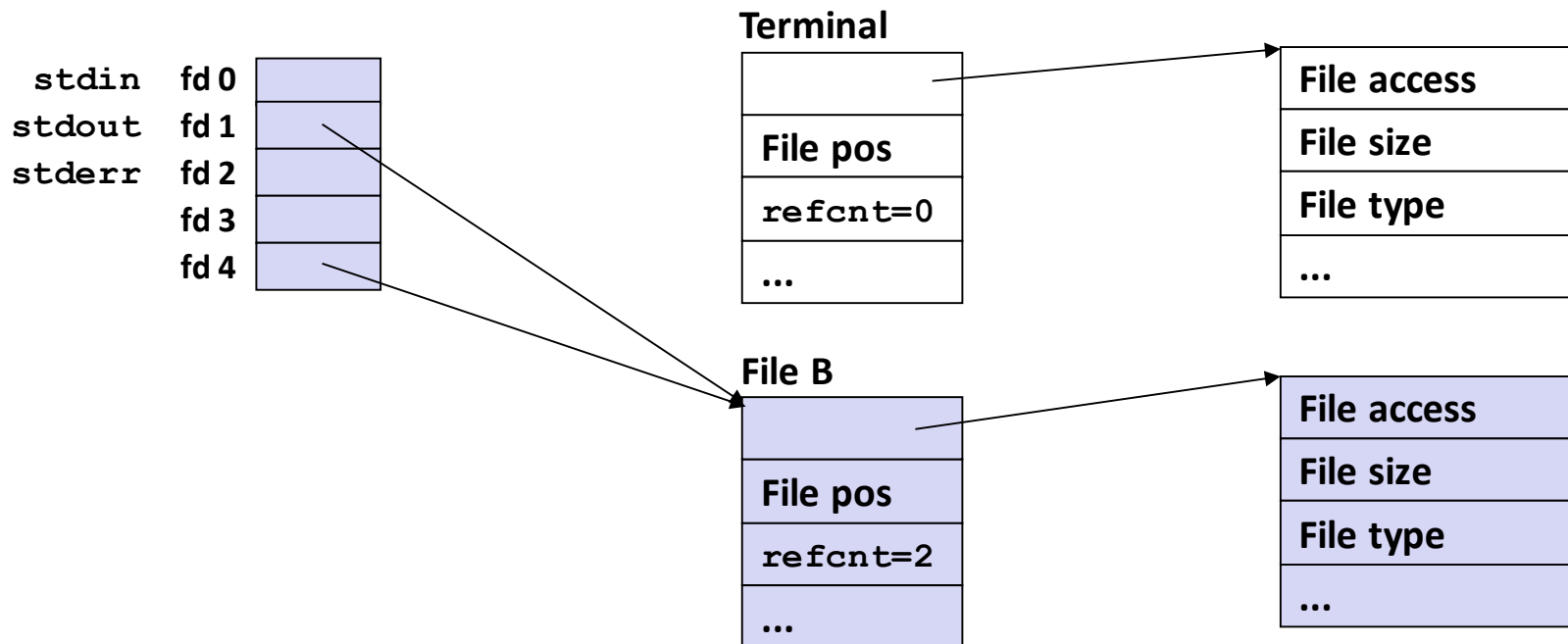
# Redirecionamento de I/O

2. Chamar `dup2(4, 1)`

**Descriptor table**  
[one table per process]

**Open file table**  
[shared by all processes]

**v-node table**  
[shared by all processes]



# Redirecionamento de I/O - usos

- Salvar saída de um comando para arquivos
- Automatizar a digitação de comandos ao redirecionar a entrada de um programa
- Permitir a comunicação entre dois programas a partir da entrada/saída padrão

# Redirecionamento de I/O – Atividade

- Parte 1 do roteiro de hoje – 30 minutos

# Comunicação entre Processos (IPC)

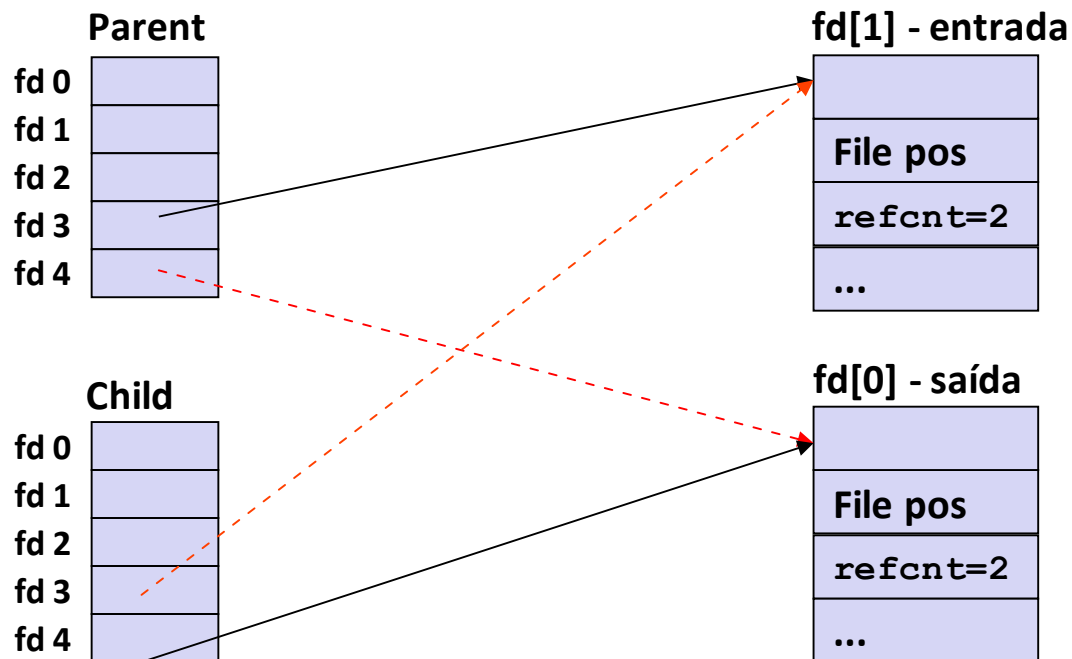
Chamada **pipe** permite comunicação unidirecional entre processos

```
#include <unistd.h>  
int pipe(int fd[2]);
```

Tudo o que for escrito (usando *write*) em *fd[1]* fica disponível para leitura em *fd[0]* (usando *read*)

# Redirecionamento de I/O - II

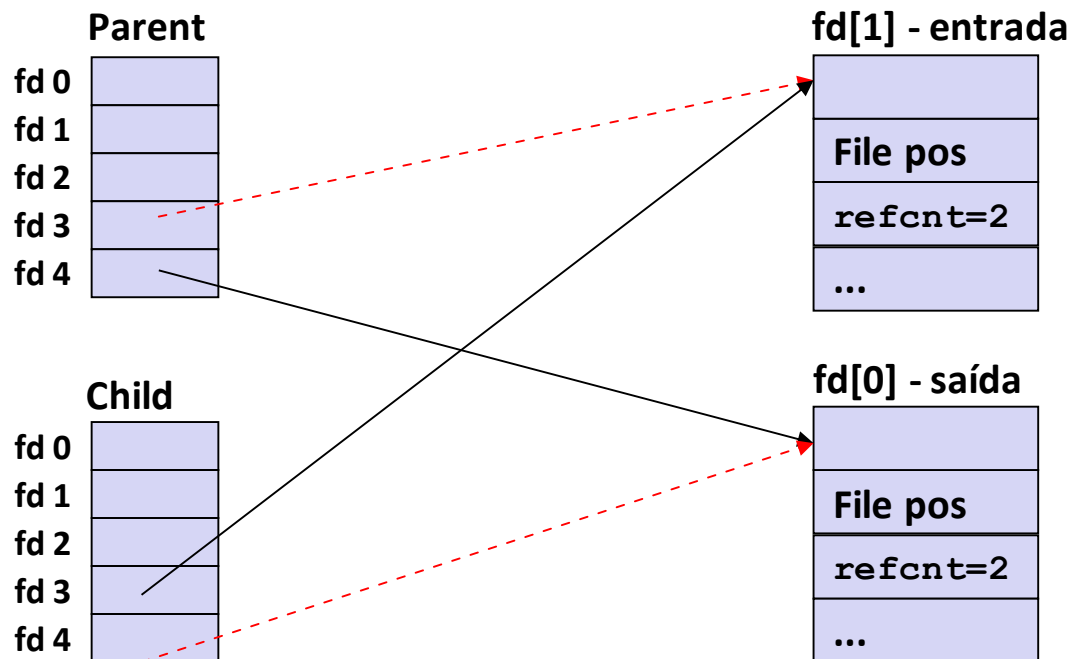
Comunicação unidirecional entre processos: Parent → Child





# Redirecionamento de I/O - II

Comunicação unidirecional entre processos: Child → Parent



# Limitações de pipes

```
#include <unistd.h>  
int pipe(int fd[2]);
```

- Comunicação unidirecional
- Só vale para processos na própria máquina
- Precisa ocorrer antes do fork

# Outras técnicas

- UNIX domain sockets (arquivo)
- Sockets (via rede) locais
- Memória compartilhada

Mais complexas, não veremos nesta disciplina

# Pipes – Atividade

- Parte 2 do roteiro de hoje – 30 minutos

# Insper

[www.insper.edu.br](http://www.insper.edu.br)