Comunicação entre Processos

José C. Cunha, DI/FCT/UNL

- Ficheiros
- Pipelines
- Mensagens
- Memória partilhada
- Sinais assíncronos

- Sincronização: AGUARDAR/ASSINALAR

-- um processo aguarda que outro atinja um certo ponto na sua execução:

-- para enviar dados ou mensagens

-- um processo assinala a ocorrência de algum evento a um outro processo

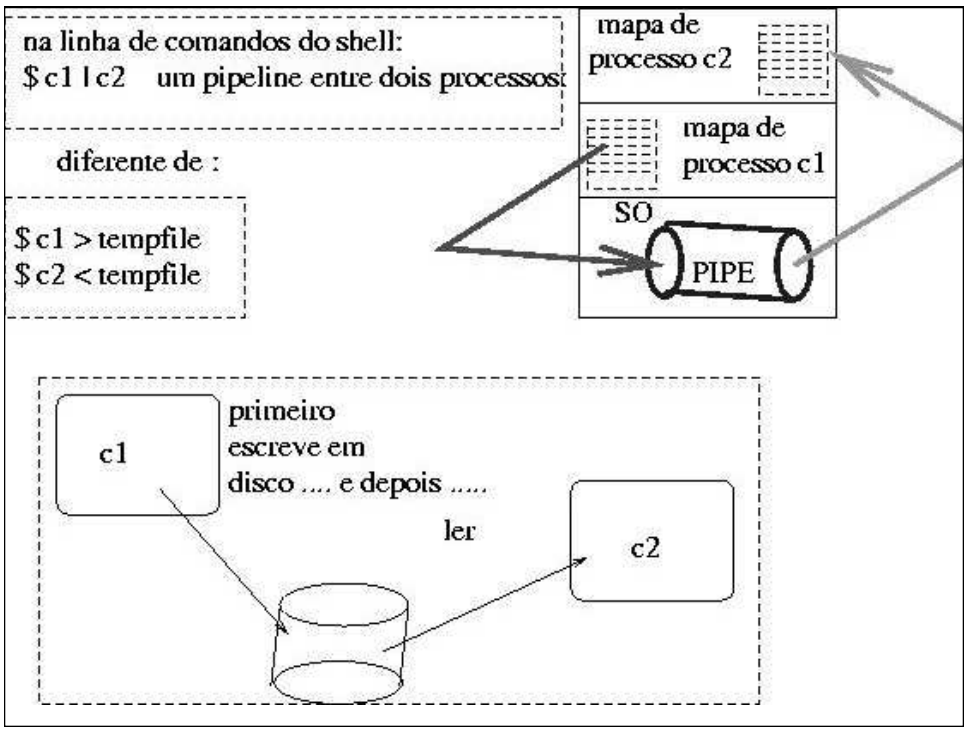
- Comunicação: TRANSFERIR INFORMAÇÃO

-- múltiplos processos concorrentes comunicam através de buffers em memória

-- processos comunicam por mensagens

Pipes no Unix

- Mecanismo de comunicação gerido pelo SO
- Realizados por buffers em memória do SO
- Fluxos de bytes consecutivos: Streams
- Só acessíveis a processos pais e filhos
- Não têm um 'pathname' associado



int pipe(int fd[2]); chamada ao sistema

cria um
PIPE: canal unidireccional

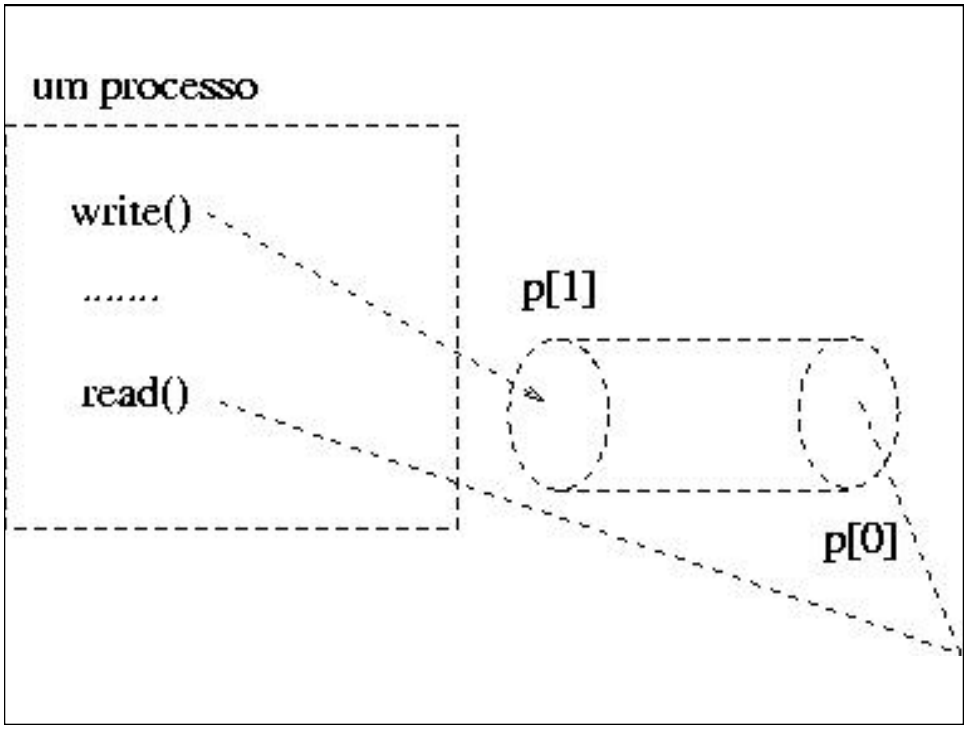
fluxo de bytes (stream)

ordem FIFO (first–in first–out)

devolve dois canais abertos para o pipe:

fd[0] para leitura

fd[1] para escrita



```
int p[2], j;
if (pipe(p) == -1) { …erro…}
write(p[1], buf1, buf1size);
write(p[1], buf2, buf2size);
for (j=0; j<2; j++)
  { read(p[0], buf, n);
    …..
  }
```

Ler de um pipe

- Read() bloqueia se o pipe está vazio
- Devolve os bytes que puder ler, mesmo se forem menos dos que os pedidos em read()

Escrever num pipe

- Tamanho do buffer usado pelo SO para o pipe: parâmetro do SO (min 512bytes)
- Write() de mais bytes dos que os disponíveis no momento no pipe:

-- bloqueia o escritor até haver espaço livre

- Write() de mais bytes do que o Tamanho do pipe no SO:

-- escreve os bytes que couberem

-- bloqueia o escritor até que possa escrever os outros

-- deixa de ser ATÓMICO ou INDIVISÍVEL!

Fechar canais abertos para pipe

close(fd[1]): fecha canal para escrita

--- se há outros canais abertos para escrita

limita-se a fechar a entrada na TabCanais do processo

--- se não há mais canais abertos para escrita: e se o pipe está vazio:

-- quaisquer read() posteriores devolvem 0

-- quaisquer read() bloqueados:

-- são reactivados e devolvem 0

Fechar canais abertos para pipe

Close(fd[0]): fecha canal aberto para leitura

-- se há outros canais abertos para leitura

-- se não:

-- quaisquer write() bloqueados:

--são assinalados com sinal SIGPIPE

(se o não previram: são terminados;

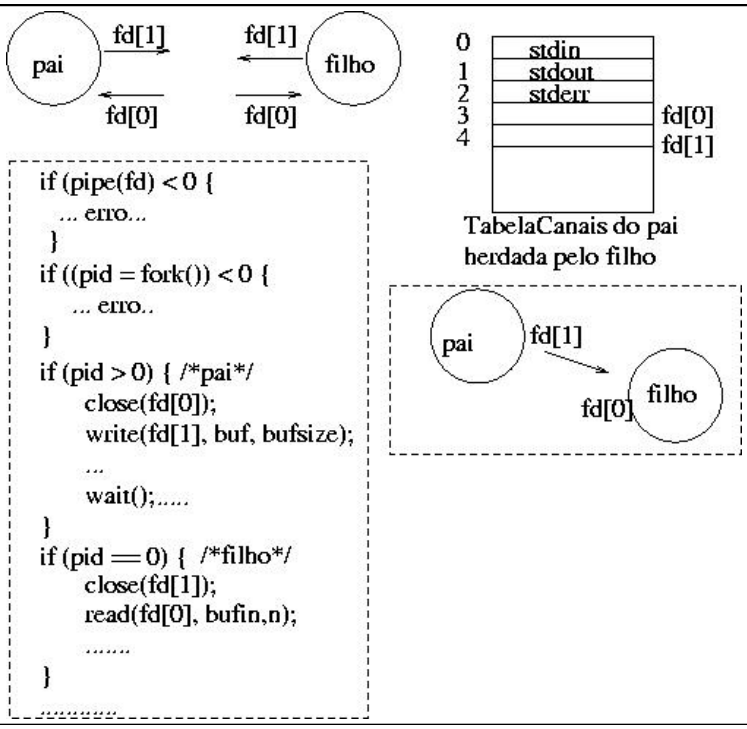
se o previram: podem tratar o sinal

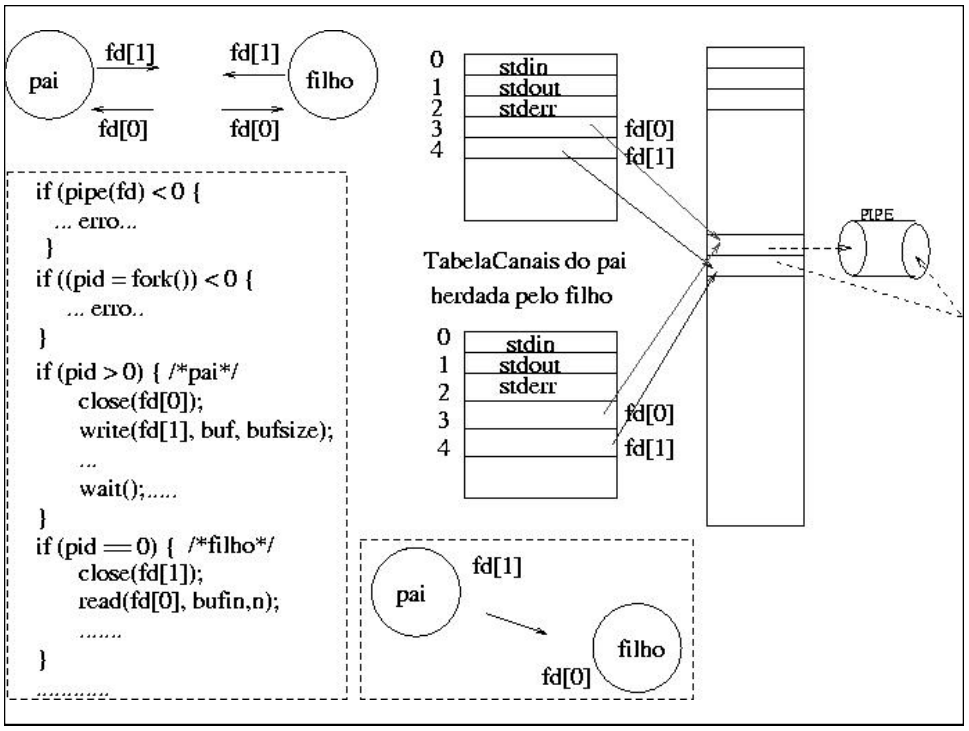
e no retorno: o write devolve –1

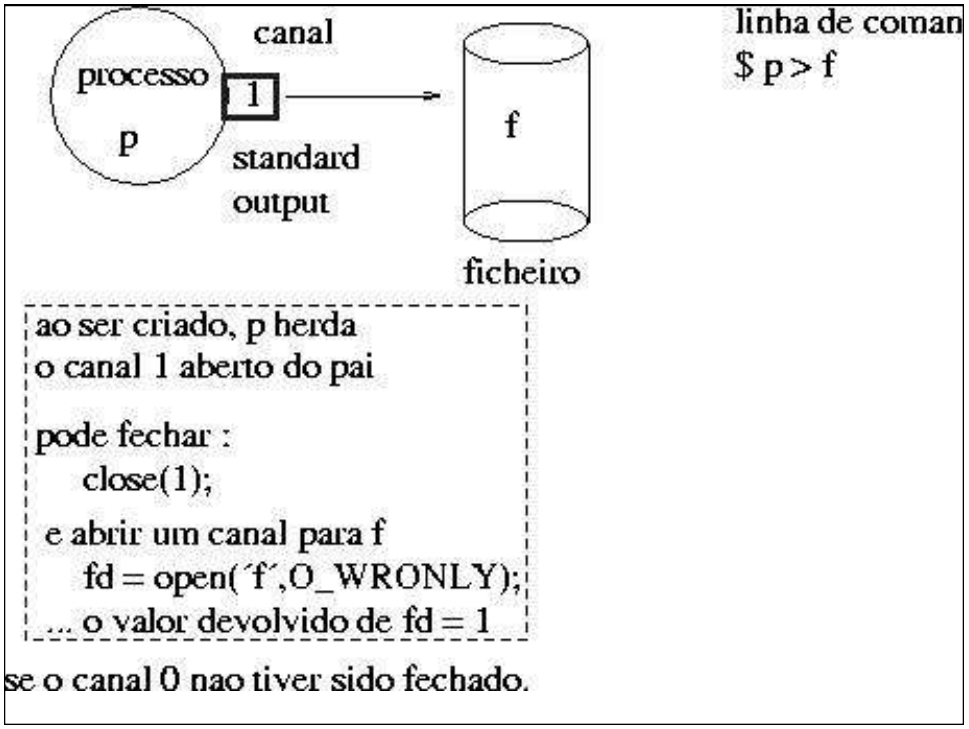
e errno == EPIPE)

-- quaisquer write posteriores:

-- também recebem o sinal

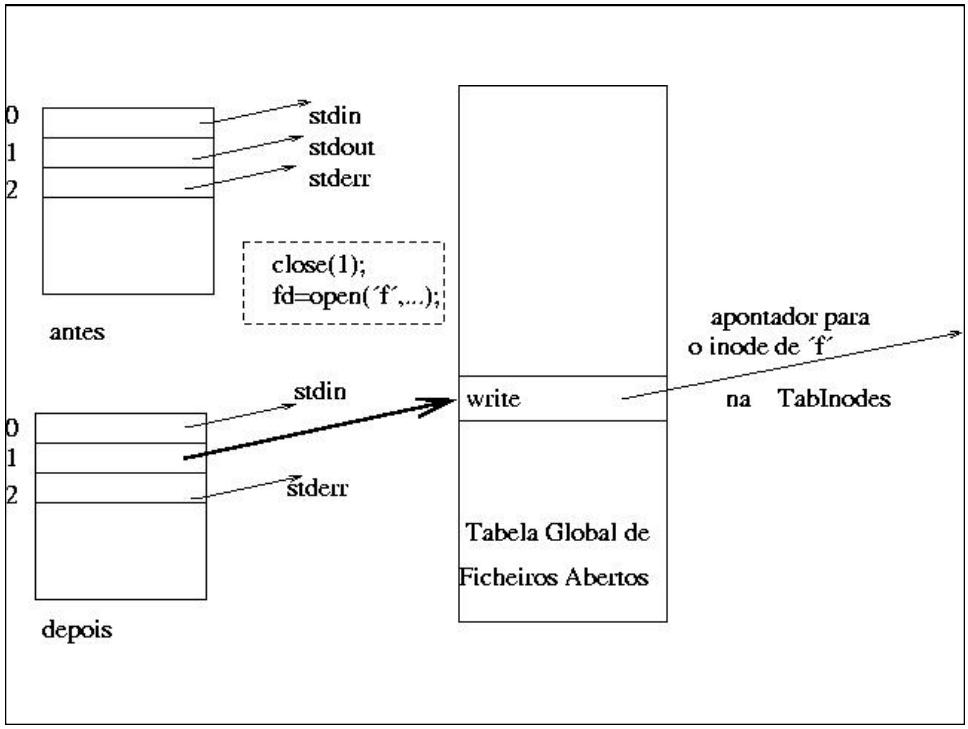


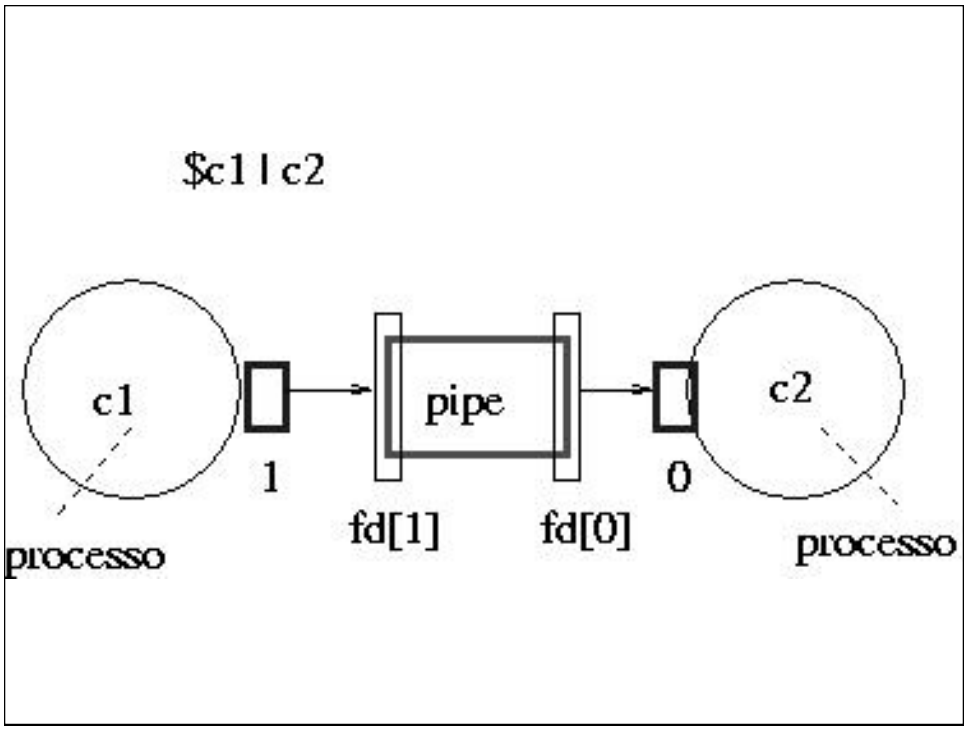

pai
filho
fd[1]
fd[1]
fd[0]
fd[0]
if (pipe(fd) < 0 {
... erro...
}
if ((pid = fork()) < 0 {
... erro..
}
if (pid > 0) { /*pai*/
close(fd[0]);
write(fd[1], buf, bufsize);
...
wait();.....
}
if (pid == 0) { /*filho*/
close(fd[1]);
read(fd[0], bufin,n);
.....
}
0
1
2
3
4
stdin
stdout
stderr
fd[0]
fd[1]
PIPE
TabelaCanais do pai
herdada pelo filho
0
1
2
3
4
stdin
stdout
stderr
fd[0]
fd[1]
pai
fd[1]
filho
fd[0]


canal
processo
1
p
standard
output
f
ficheiro
linha de coman
$ p > f
ao ser criado, p herda
o canal 1 aberto do pai
pode fechar :
close(1);
e abrir um canal para f
fd = open(´f´,O_WRONLY);
... o valor devolvido de fd = 1
se o canal 0 nao tiver sido fechado.

Semântica do open()

- procurar a primeira entrada livre na TabCanais do Processo e utilizá-la para guardar o 'file descriptor' do canal a abrir





Redirigir canais para ‘pipes’

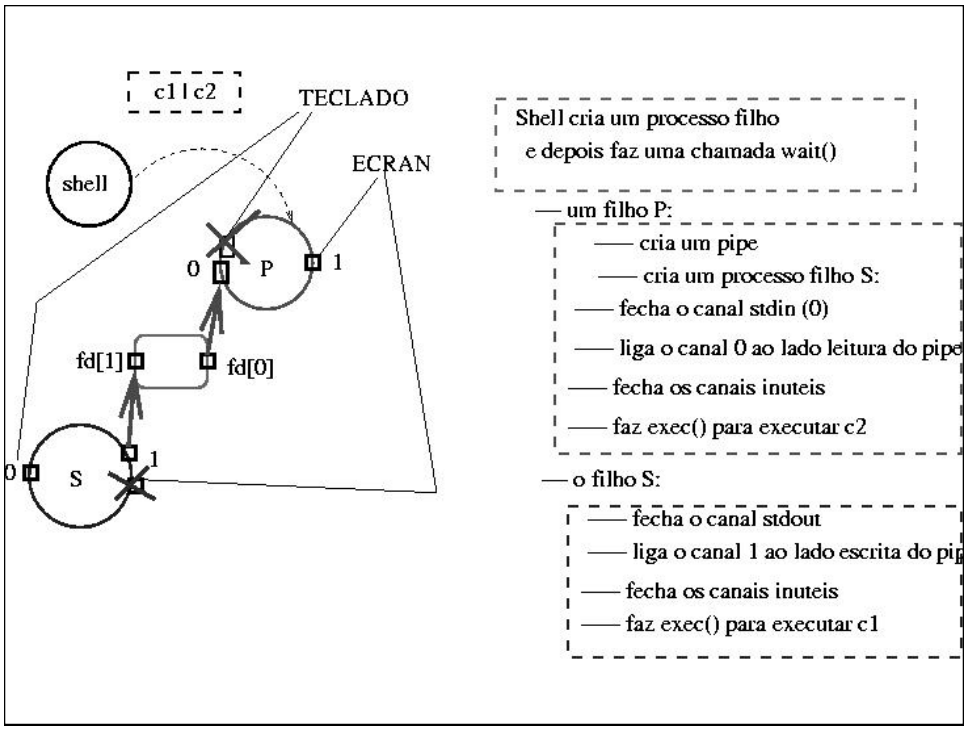
- Objectivo:

-- quando C1 fizer write(1,…), deve escrever no lado de escrita do ‘pipe’

-- quando C2 fizer read(0,…), deve ler do lado de leitura do ‘pipe’, os bytes escritos por C1

NOTAR QUE os filhos HERDAM os canais do pai

Portanto se o pai criar um pipe, os seus filhos herdam os canais abertos para o pipe



Ligar um canal a um lado de pipe

?

Duplicar um 'file descriptor'

int dup(int oldfd);

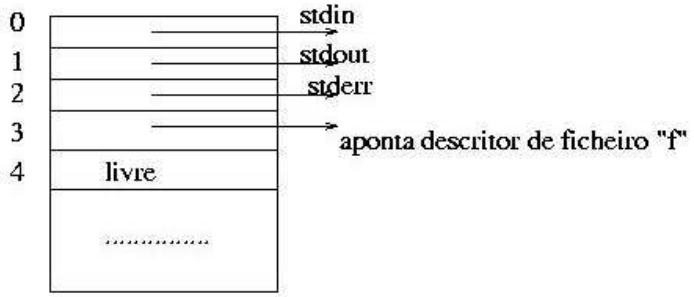
-- devolve um novo fd que é uma cópia do indicado por oldfd

-- a semântica é a de procurar a primeira entrada livre na TabCanais do processo

int dup2(int oldfd, int newfd);

-- newfd é uma cópia de oldfd

(fechando newfd, se estiver aberto)



TabCanais do processo

A

```
newdf = dup(3);      /*copia entrada 3 para ?*/
                     /*newfd == ?*/
```

B

```
close(1);        /*liberta entrada 1*/
newdf = dup(3);  /*copia entrada 3 para ?*/
                 /*newfd == ?*/
```

```
main()
{ char *c1[4] = {“nome1”, “a11”, “a12”, NULL};
  char *c2[3] = {“nome2”, “a21”,NULL};
  int ret:

 ret = join(c1,c2);
}
```