Fundamentos de Sistemas de Operação

Unix Windows NT Netware MacOS DOS/VS Vax/VMS
Linux Solaris HIP/UX AIX Mach
Chorus

Programas, Processos e o SO:

Comunicação entre processos: I

(cont.)

Resumo da aula anterior…

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

- *O processo, conceito oferecido pelo SO,*
  - *"É um contentor" que tem um CPU, uma memória e periféricos idênticos aos da "máquina real". Cada programa corre nesta máquina virtual como se mais nada existisse…*
  - *Mas um processo tem de comunicar - ler e escrever em periféricos, em ficheiros e com outros processos - por exemplo, usando pipes.*
  - *Quando um processo invoca um* **`fork()`** *o processo-filho corre o mesmo programa e herda o EE e estado do pai: variáveis globais e de ambiente, canais abertos. Difere do pai no PID, PPID, e valor de retorno do* **`fork()`**.
  - *Quando um processo invoca um* **`exec()`**, *parte do seu espaço de endereçamento (stack, heap, código, variáveis globais) perde-se, sobreposto com o novo executável; mas mantêm-se os canais abertos e as variáveis de ambiente.*

Recapitulando: usando um pipe (1)


```
int main()
{ ...
  if ( pipe(fd) == -1 ) abort();          // cria pipe e abre fd's
  pid=fork();
  if (pid) {
    write(fd[1], "ola", 3);               // pai escreve
    ...
  } else {
    read(fd[0], buf, 3);                  // filho lê
    ...
  }
}
```

- *Pai e filho executam concorrentemente ☺…*
- *O SO sincroniza os processos de forma indirecta, através das operações* **`read()`** *e* **`write()`** *…*

Recapitulando: usando um pipe (2)


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

  if ( pipe(fd) == -1 ) abort();   // cria pipe e abre fd's para R e W
  pid=fork();
  if (pid) {
    close(fd[0]);                  // pai vai W, não precisa de R
    write(fd[1], "ola", 3);        // pai escreve
  } else {
      close(fd[1]);                // filho vai R, não precisa de W
      read(fd[0], buf, 3);         // filho lê
  }
}
```

- *Código praticamente completo! ☺…*

Para pensar: usando um pipe…


```
int main()
{ ...
  pid=fork();
  if (pid) {
    close(fd[0]);                          // pai vai W, não precisa de R
                                           // É necessário?
    while (utilizadorInsereTexto())
       write(fd[1], "ola", 3);             // pai escreve
    << Como indicar que terminou? >>
  } else {
      close(fd[1]);                        // filho vai R, não precisa de W
                                           // É necessário?
      while (<< Como detectar que não há mais para ler? >>)
        read(fd[0], buf, 3);               // filho lê
  }
```

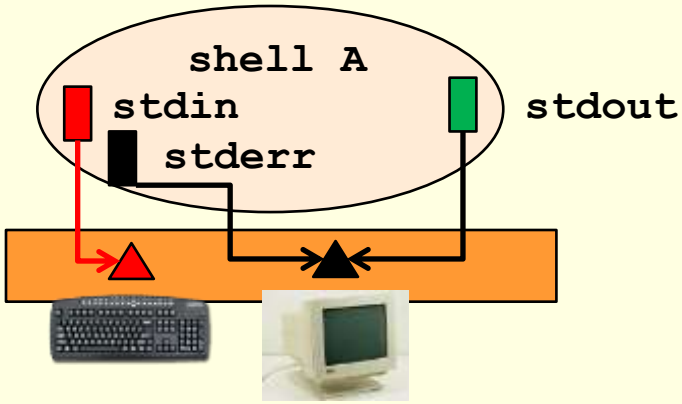
Comunicação usando um pipe (1)

- *Caso prático: pipes na shell*
  - *O utilizador executa:* **`$ ls | wc -l`**
  - *O resultado, do ponto de vista do utilizador, é: o output do comando* **`ls`**, *(a lista com os nomes dos ficheiros existentes na directoria) é enviado para o pipe, que por sua vez serve de input do comando* **`wc -l`**, *que mostra ao utilizador o número de ficheiros existem…*
  - *Não percebeu? Experimente* ☺

    `$ cd /bin` *(ou outra directoria com muitos ficheiros)*

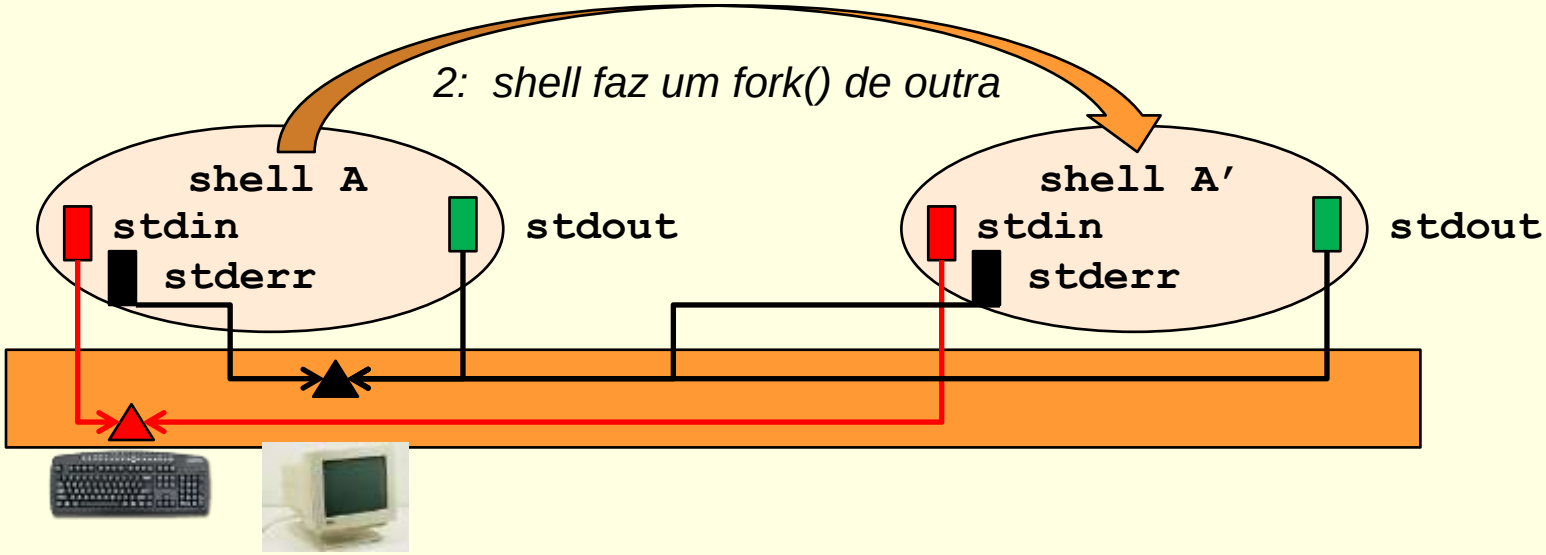
    `$ ls | wc -l`

Pipe na shell - como funciona? (1)

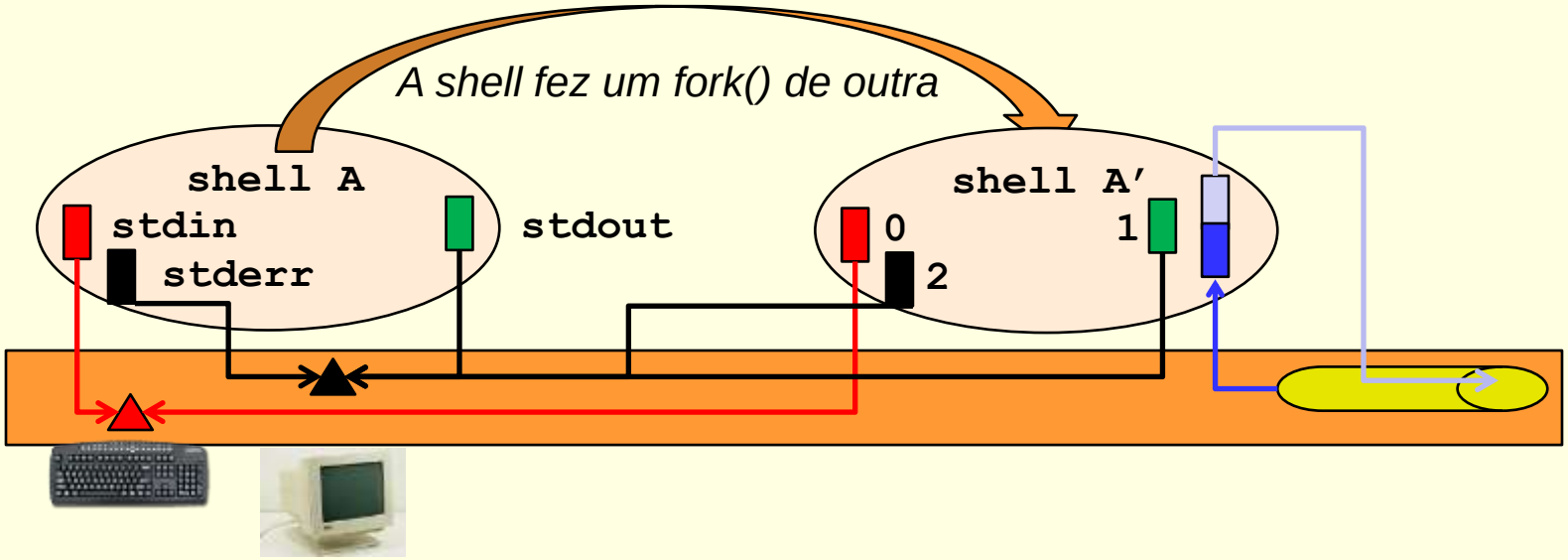


1: Estado inicial:
canais abertos por omissão,
ligados aos periféricos standard.
Utilizador escreve:

```
$ ls | wc -l
```

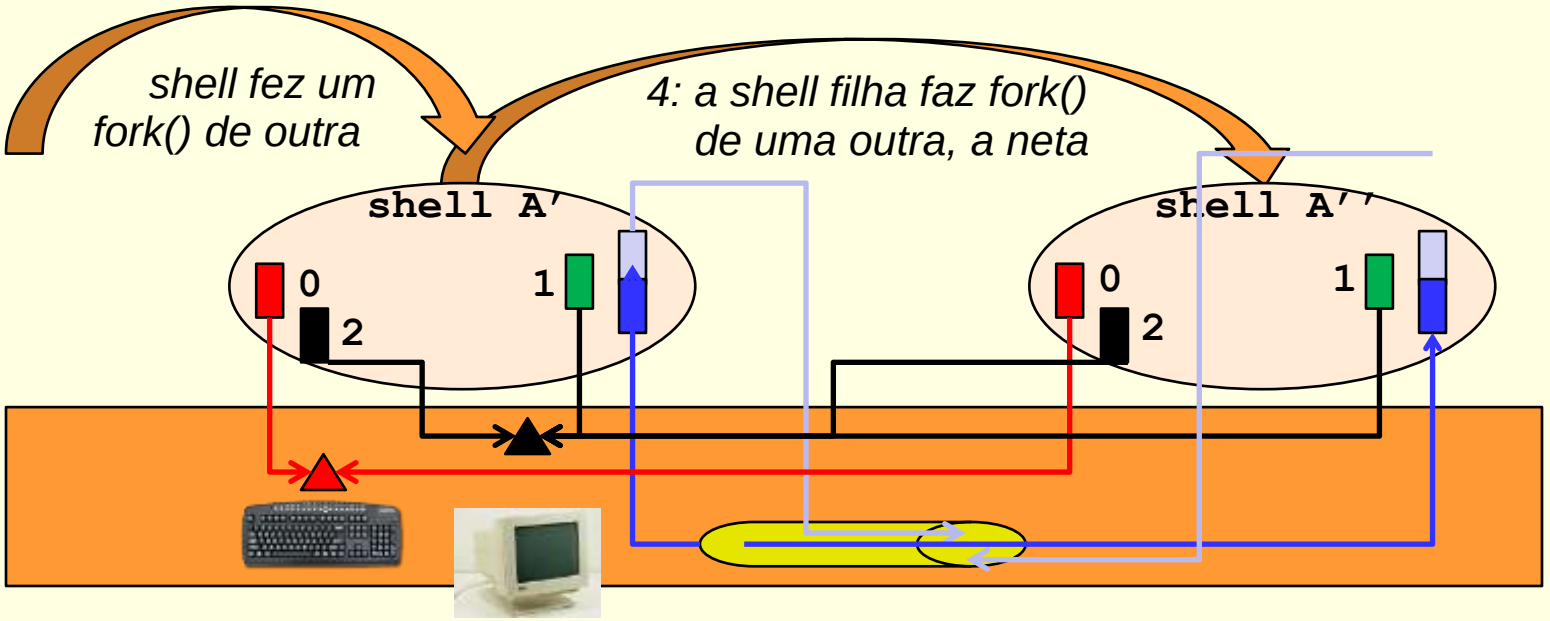


Pipe na shell - como funciona? (2)



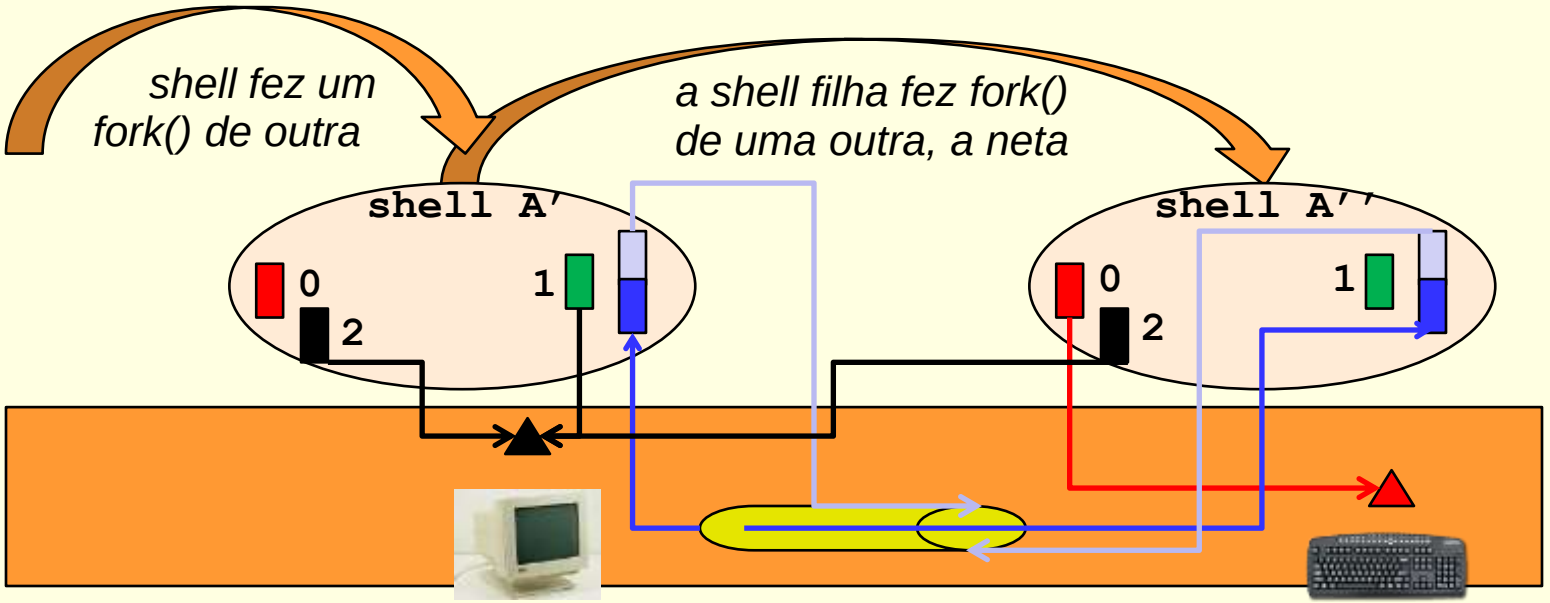
3: A nova shell (filha) cria um pipe; este usa dois novos descritores

Pipe na shell - como funciona? (3)



Nota: só há um pipe, que é acedido pelos dois processos usando os novos descritores

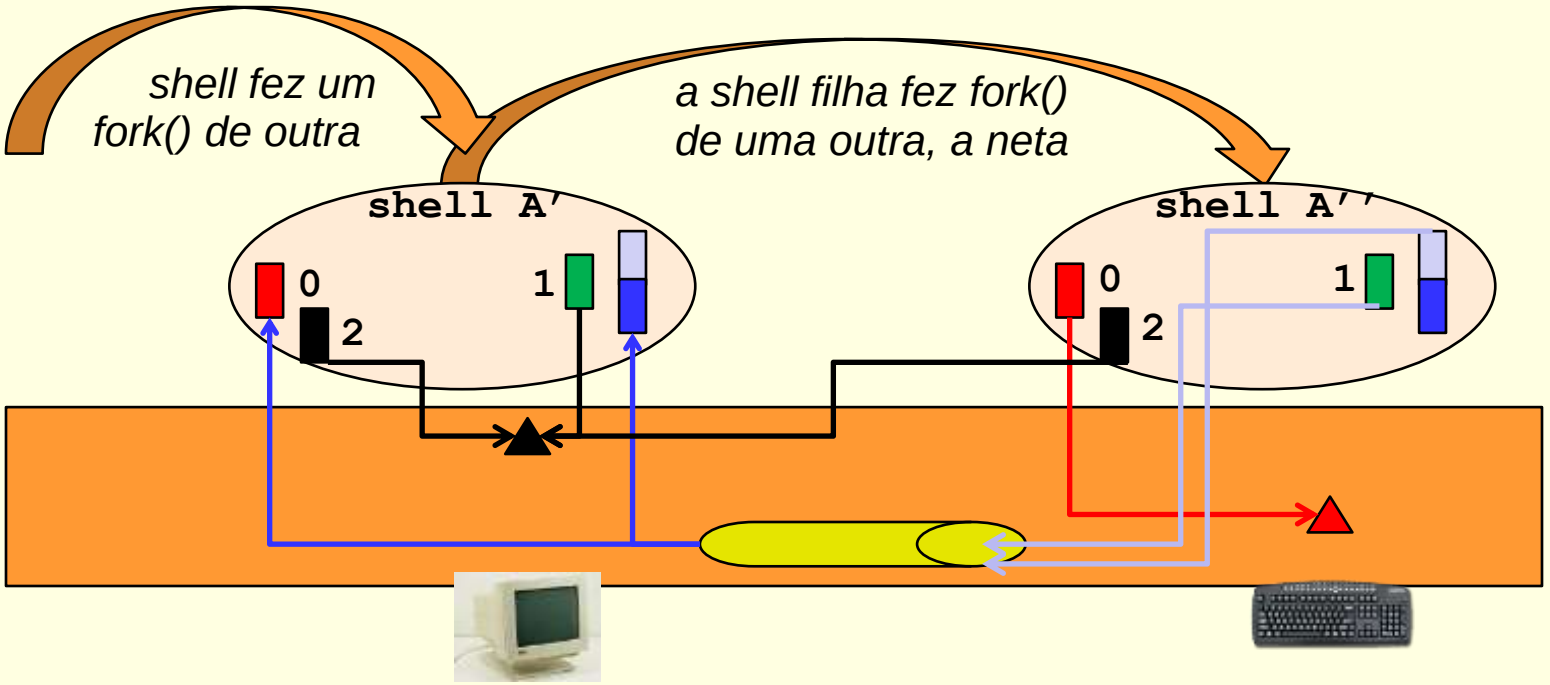
Pipe na shell - como funciona? (4)



5: A shell filha vai correr o programa wc. Precisa do descritor 0 (stdin) associado ao canal R do pipe. Por isso, começa por fechar o 0.

6: A shell neta vai correr o programa ls. Precisa do descritor 1 (stdout) associado ao canal W do pipe. Por isso começa por fechar o 1.

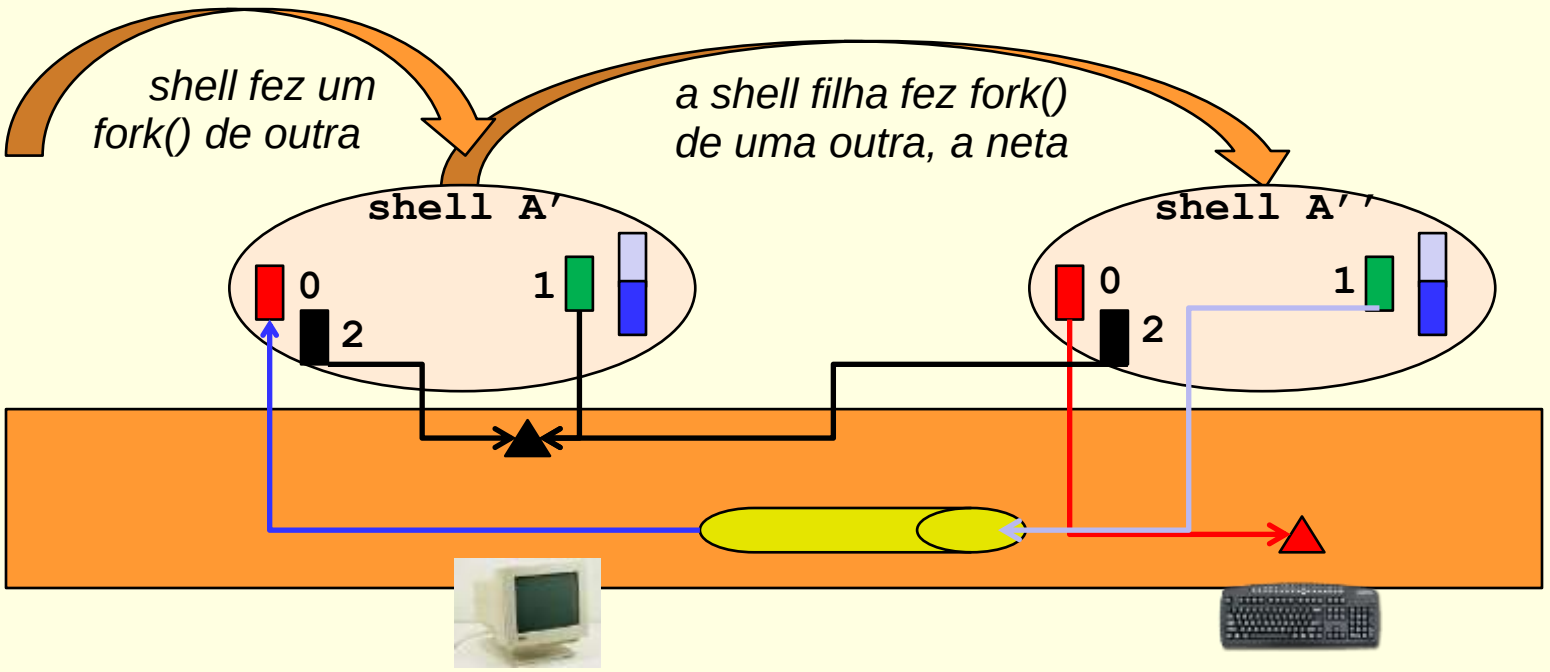
Pipe na shell - como funciona? (5)



7: A shell filha associa o descritor 0 ao canal R do pipe. Como não precisa do canal W do pipe, fecha-o.

8: A shell neta associa o descritor 1 ao canal W do pipe. Como não precisa do canal R do pipe, fecha-o.

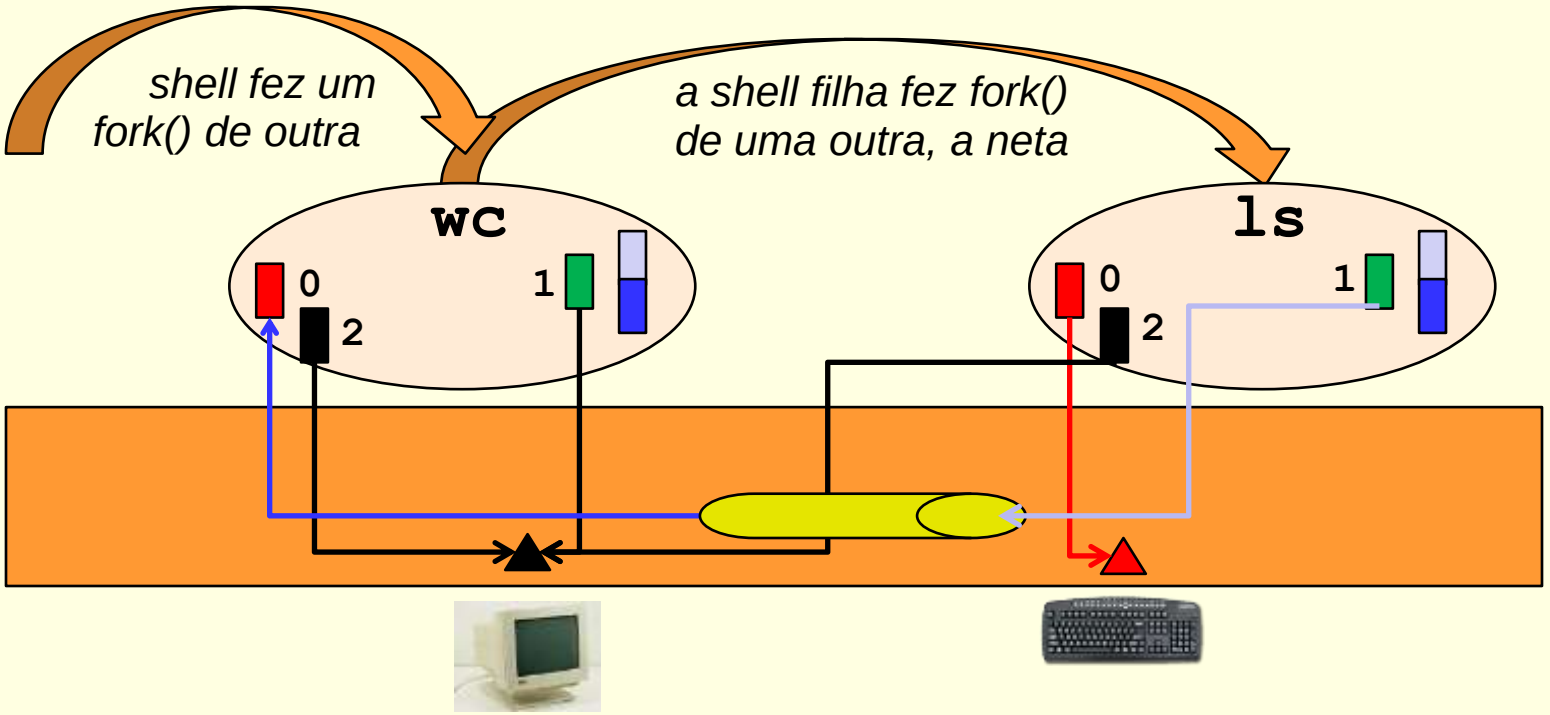
Pipe na shell - como funciona? (6)



9: A shell filha já tem o descritor 0 associado ao canal R do pipe, por isso o descritor inicial do pipe para R não é necessário e deve ser fechado.

10: A shell neta já tem o descritor 1 associado ao canal W do pipe, por isso o descritor inicial do pipe para W não é necessário e deve ser fechado.

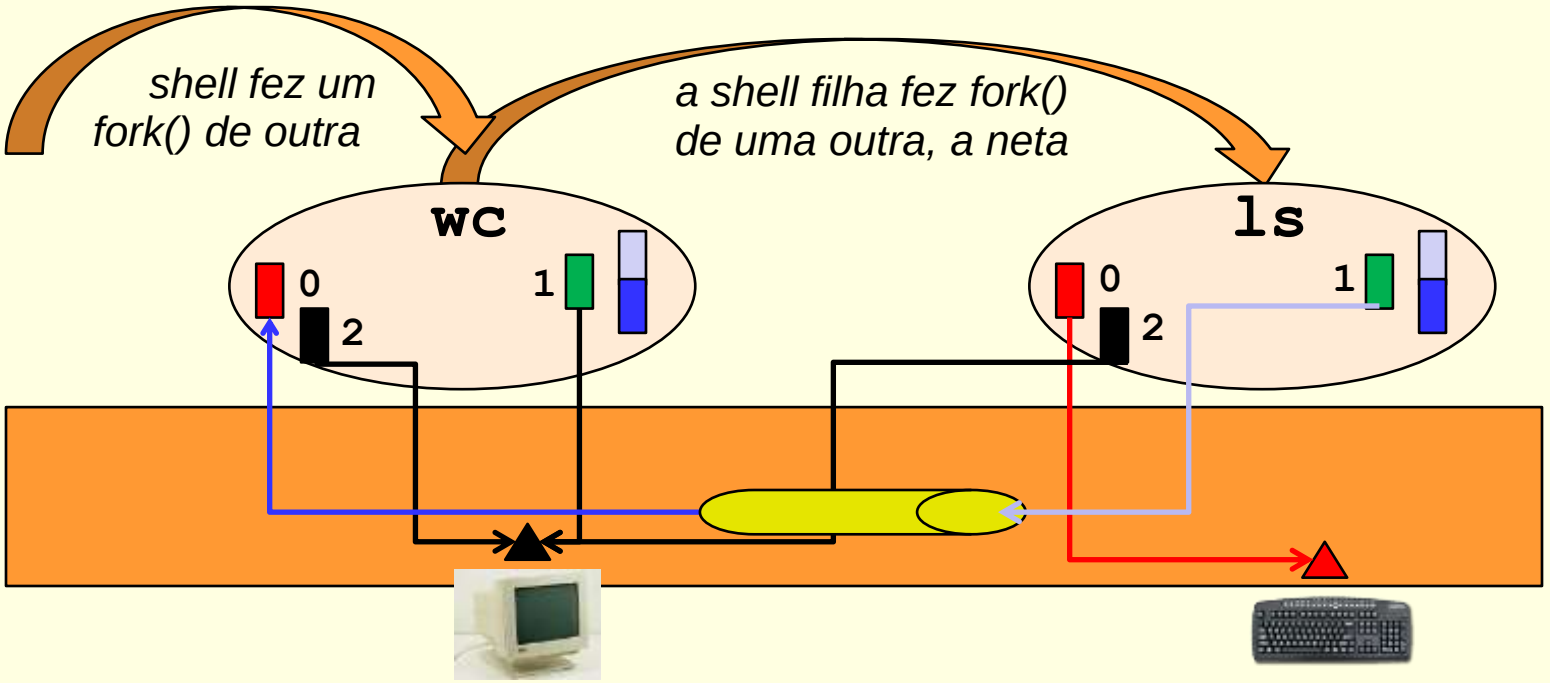
Pipe na shell - como funciona? (7)



11: A shell filha faz o exec do programa wc.

12: A shell neta faz o exec do programa ls.

Pipe na shell - como funciona? (8)



FIM

Os dados produzidos por ls, em vez de saírem no ecrã, vão para o pipe, e daí vão para o programa wc, que depois escreve o resultado final no ecrã.

Associar um descritor a um canal aberto (1)

- *Todas as operações descritas nos slides anteriores…*
- *Podem ser realizadas com primitivas que já conhecemos, nomeadamente,* **`fork ()`**, **`exec()`**, **`pipe()`** *e* **`close()`**…
- *Excepto no slide 7, onde dizemos: "o descritor X precisa de ser associado ao canal (aberto) Y"*
- *Como fazer isso?*

Associar um descritor a um canal aberto (2)

- **`int dup(int oldfd)`**
  - *Cria um novo descritor e associa-o ao mesmo "alvo" identificado pelo descritor* **`oldfd`**.
  - *O número de canal (ou descritor) atribuído é o menor índice livre na tabela de canais.*
- *Sempre que um descritor é associado a um "alvo" - periférico, ficheiro, pipe, etc.- um contador de referências é incrementado "no alvo".*
- *Sempre que um descritor é fechado, o contador de referências é decrementado. Só quando o contador chega a zero é que o "alvo" é efectivamente descartado.*

Para pensar…

- *Agora! Usando os slides 7 a 11,*
  - *Programe uma shell que suporte a execução de dois programas interligados por um pipe…*
  - *[Nota: vai ter de fazer isto num trabalho prático]*
- *Mais tarde…*
  - *Tente fazer o mesmo usando outro algoritmo: em vez da shell raiz lançar outra (filha) que por sua vez lança uma terceira (neta)…*
  - *Faça com que a shell raiz lance duas "sub-shells" (irmãs) e que, mesmo assim, se consiga comunicar através de um pipe entre elas*

Usando um pipe... experimente!


```
/* Faltam os #includes */

int main()
{ int p[2];

  if (fork()) {
    printf("Pai -pseudo-shell- em WAIT\n");  wait(NULL);
  } else {
    printf("A'\n");
    pipe(p);

    if (fork()) {
      close(0); dup(p[0]);
      close(p[0]); close(p[1]);

      execlp("wc", "wc", "-l", NULL);
    } else {
```

Usando um pipe… experimente!


```
  /* } else {                                          Linha repetida! */
       printf("A''\n");
       close(1); dup(p[1]);
       close(p[0]); close(p[1]);

       execlp("ls", "ls", NULL);
     }
   }
}
```