Fundamentos de Sistemas de Operação

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

Programas, Processos e o SO:

Continuação

Resumo da aula anterior…

- *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…*
  - *O SO oferece um conjunto de funções para manipular processos*
    - *Criar:* **`fork()`**
    - *Terminar:* **`kill(pid,…)`** *e* **`_exit(valor)`**
    - *Obter PIDs:* **`getpid()`** *e* **`getppid()`**
    - *Esperar:* **`wait()`**
    - *Executar novo programa no processo:* **`execl()`**, **`execlp()`**, **`execv()`**, **`execve()`**, **`execvp()`**

`fork()`: *mais exemplos* (1)

- Esqueleto de código com um `fork()` e `if`

```
int main( int argc, char *argv[] )
{
  int pid;
    ...
        pid= fork();

        if (pid > 0) {
          // Código a executar pelo pai
        } elseif (pid == 0) {
         // Código a executar pelo filho
        } else
         // Código para tratar o erro do fork;
        }

    ...
}
```

`fork()`: *mais exemplos* (2)

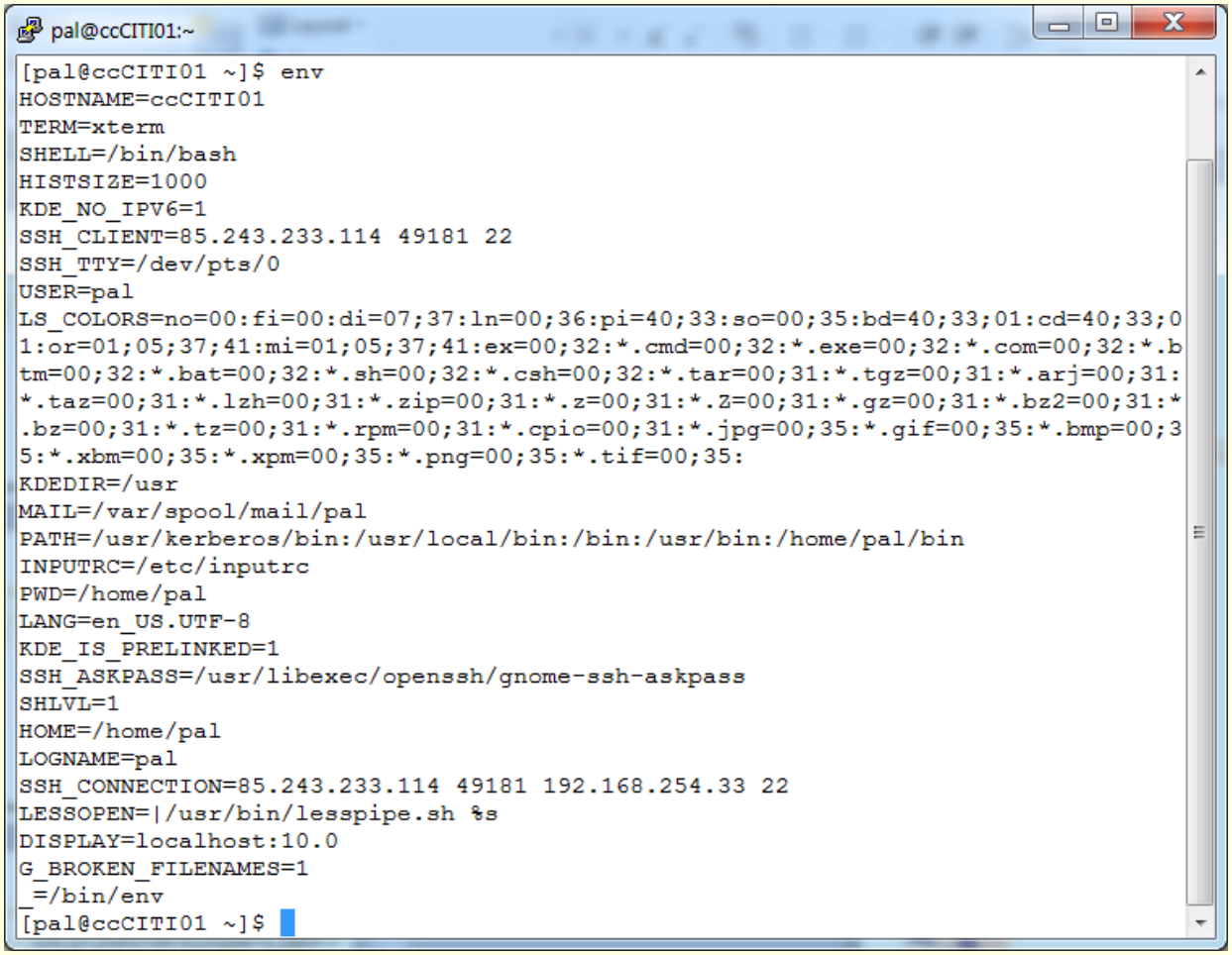
- Esqueleto de código com um `fork()` e `switch`

```
int main( int argc, char *argv[] )
{
  int pid;
    ...
  pid= fork();

  switch (pid) {
    case  0: // Código a executar pelo filho
    case -1: // Código para tratar o erro do fork
    default: // Código a executar pelo pai
  }

    ...
}
```

`fork()`: *erro? Como?*

- *Como com (quase) todas as chamadas ao sistema, a execução de um* **`fork()`** *pode falhar… quando?*
  - *Quando o número máximo de processos definido por utilizador (considerando o utilizador que está a lançar o novo processo) é atingido, ou*
  - *Quando o número máximo de processos definido para o sistema é atingido*
  - *Quando há escassez de memória livre…*

Variáveis de ambiente (1)



Variáveis de ambiente (2)

- *As variáveis de ambiente,*
  - *São usualmente criadas, destruídas, inicializadas e modificadas usando uma shell, e os comandos*
    - *Ver (a lista de) todas as variáveis de ambiente:* **`set`**
    - *Ver apenas as variáveis de ambiente que foram exportadas:* **`env`**
    - *Exportar uma variável de ambiente:* **`export`**
    - *Destruir uma variável de ambiente:* **`unset`**
  - *Algumas são "criadas e inicializadas" durante o processo de login do utilizador:* `HOME, LOGNAME, PATH, PS1,...`
  - *Só as variáveis exportadas são passadas a um processo filho na altura que este é lançado.*

Variáveis de ambiente (3)

- *Exemplos:*
  - *Criar uma nova variável com a bash e atribuir-lhe um valor*

```
$ NOVA=123
```

  - *Ver o valor de uma única variável*

```
$ echo $NOVA
123
```

  - *Nota: usar maiúsculas para os nomes das variáveis é uma convenção; tudo funciona na mesma se usarmos minúsculas.*

Variáveis de ambiente (4)

- *Algumas variáveis são especialmente importantes:*
  - **`PATH:`** *Lista de directorias a percorrer para procurar um programa (ou script) que se tenta executar*

```
$ echo $PATH
/bin:/usr/bin:/home/pal/bin
```

Se dermos a ordem para executar o programa prog

```
$ prog
```

O programa vai ser procurado na directoria **`/bin`***; se não existir aí, vai ser procurado na directoria* **`/usr/bin`***; e assim sucessivamente. Se chegarmos à última directoria da lista sem se ter encontrado o programa, temos um erro...*

Fundamentos de Sistemas de Operação

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

Programas, Processos e o SO:

Canais de Comunicação

Canais de Comunicação (1)

- *Um processo necessita de comunicar; necessita de:*
  - *Transferir informação de/para periféricos*
    - *Exemplos: teclado, écrã*
  - *Transferir informação de/para ficheiros*
  - *Transferir informação de/para outros processos*
- *Nos SOs modernos o conceito de "canal de comunicação" suporta os 3 casos*

Canais de Comunicação (2)

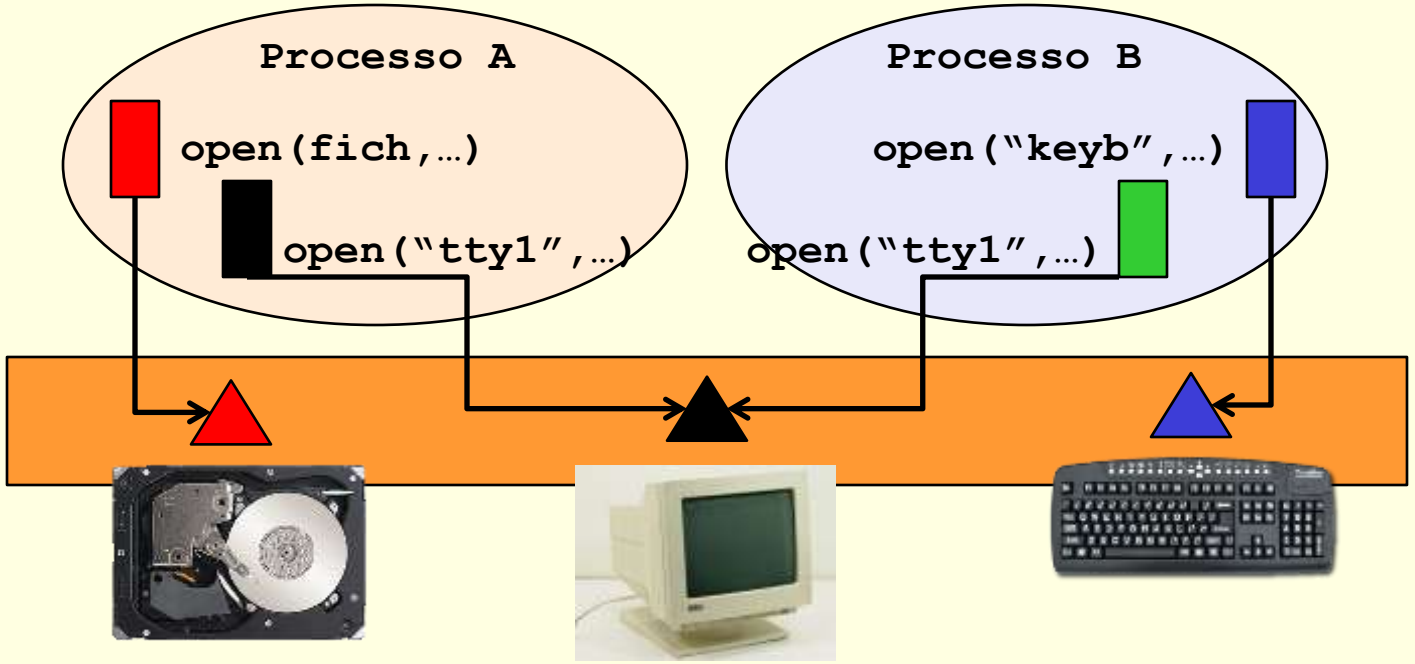
- *Um canal de comunicação é uma abstracção oferecida pelo SO que permite a um processo comunicar com o "exterior".*
- *Para usar um canal,*
  - *Abre-se o canal, com uma operação* **`open()`**, *associando-o ao dispositivo (periférico), ficheiro, ou mecanismo de comunicação entre processos (IPC) que se quer usar*
  - *Transfere-se informação de/para o canal usando as primitivas* **`read()`** *e* **`write()`** *– afinal, as mesmas que já conhecemos para aceder a ficheiros…*
  - *Sinaliza-se o desejo de não querer transferir mais informação fechando o canal com uma operação* **`close()`**

Canais de Comunicação em Unix (1)

- *No Unix um canal de comunicação é uma abstracção que "existe" no interior do núcleo; o acesso de um processo a essa abstracção faz-se usando um **descritor**.*
  - *Quando, com* **`open()`**, *se abre um canal - associando-o ao dispositivo (periférico), ficheiro, ou mecanismo de comunicação entre processos (IPC) que se quer usar – se a operação for bem sucessida o* **`open()`** *retorna um descritor para o canal*
  - *Transfere-se informação de/para o canal usando as primitivas* **`read()`** *e* **`write()`** *– afinal, as mesmas que já conhecemos para aceder a ficheiros…*
  - *Sinaliza-se o desejo de não querer transferir mais informação fechando o canal com uma operação* **`close()`**

Canais de Comunicação em Unix (1)

- *No Unix um canal de comunicação é uma abstracção que "existe" no interior do núcleo; num processo o acesso a essa abstracção faz-se usando um **descritor**.*



Canais de Comunicação em Unix (2)

- *Num processo, o descritor que "liga" a um dado canal é um número inteiro; (o valor d)esse número pode ser diferente noutro processo e no entanto os dois identificarem o mesmo canal - vejam-se os descritores negro (processo A) e verde (processo B) no slide anterior.*

- *Note-se que, no exemplo acima, as mensagens escritas tanto pelo processo A como pelo B se "misturam" no mesmo écrã… o que ilustra o que anteriormente referimos quando falamos, no* **`fork()`** *da "clonagem" de canais…*

Operações fundamentais de E/S: `open()`

- **`int open(char *alvo, int opções)`**

Onde alvo é o identificador, no sistema de ficheiros do computador, da entidade que queremos usar na comunicação… Por exemplo:

- *Se queremos aceder a um ficheiro,* **`alvo`** *pode ser* `"/home/pal/texto.txt"`
- *Se queremos aceder a um periférico (teclado),* **`alvo`** *pode ser* `"/dev/keyboard"`

*As opções servem para especificar muitas e variadas "coisas"… Por exemplo, podem ser usadas para especificar se apenas queremos usar o canal para ler (***`O_RDONLY`***), para ler e escrever (***`O_RDWR`***), ou só para escrever (***`O_WRONLY`***)*

Operações fundamentais de E/S: `open()`

- *Exemplo 1: Queremos abrir o ficheiro* `"/home/pal/texto.txt"` *para leitura/escrita*
  - **`fd= open(`**`"/home/pal/texto.txt"`**`, O_RDWR)`**
- *Exemplo 2: Queremos abrir o ficheiro* `"texto.txt"`*, localizado na directoria de trabalho do processo, somente para leitura*
  - **`fd= open(`**`"texto.txt"`**`, O_RDONLY)`**

  ou,
  - **`fd= open(`**`"./texto.txt"`**`, O_RDONLY)`**

Operações fundamentais de E/S: `open()`

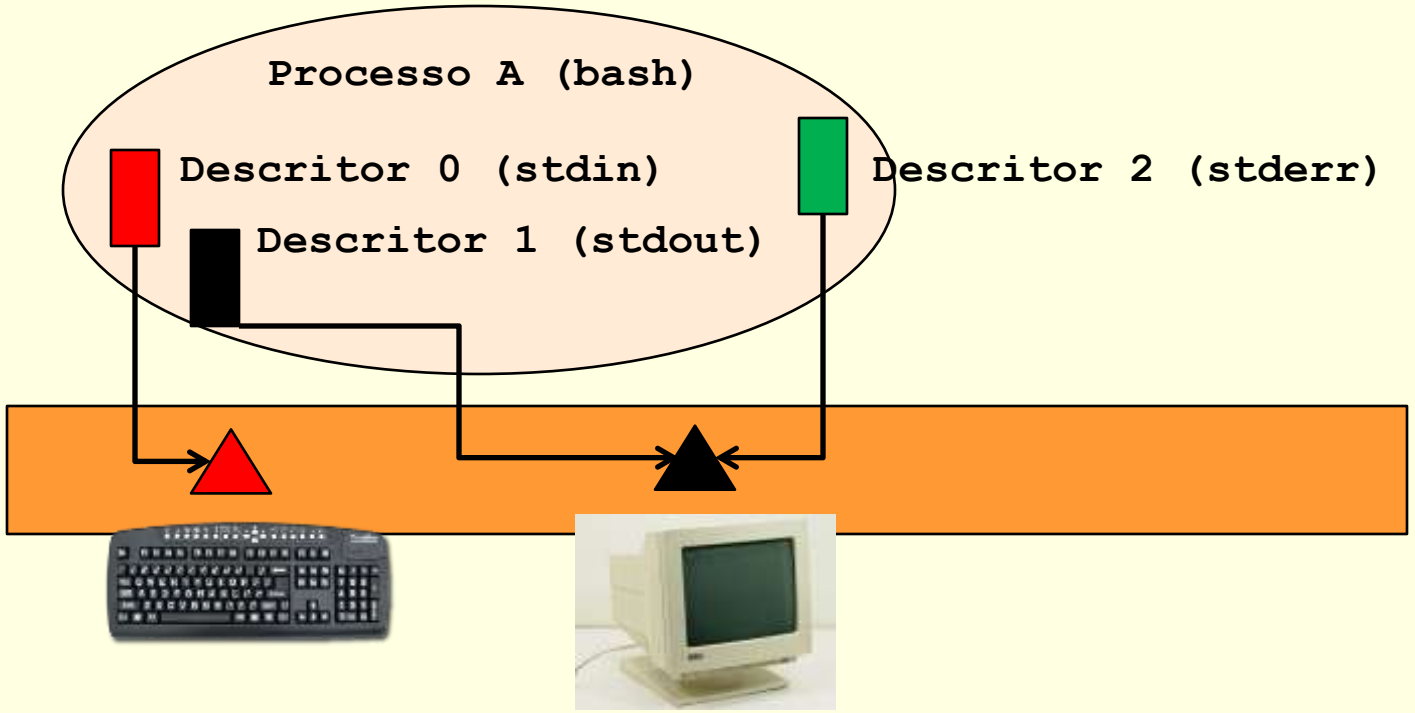
- *O* **`open`** *é uma chamada de sistema complexa porque permite fazer muitas "coisas" diferentes… Por exemplo,*
  - *permite criar um ficheiro:*

    `open(``"texto.txt"`**`, O_RDWR|O_CREAT|0600)`**

Já sabemos que a opção **`O_RDWR`** *especifica que o canal deve ser aberto para ler e escrever, mas combinamos essa opção com* **`O_CREAT`**, *que indica que o ficheiro deve ser criado se já não existir, e ainda com* **`0600`** *que especifica que, no caso de ser criado, deve ficar com um conjunto de permissões que nos permitem apenas a nós, "seus criadores" aceder ao ficheiro – e mais ninguém. [As questões de controle de acessos/permissões serão apresentadas mais tarde ☺]*

Canais standard em Unix

- *No Unix, quando um utilizador efectua o login e "cai" numa shell, por omissão há 3 canais abertos:*

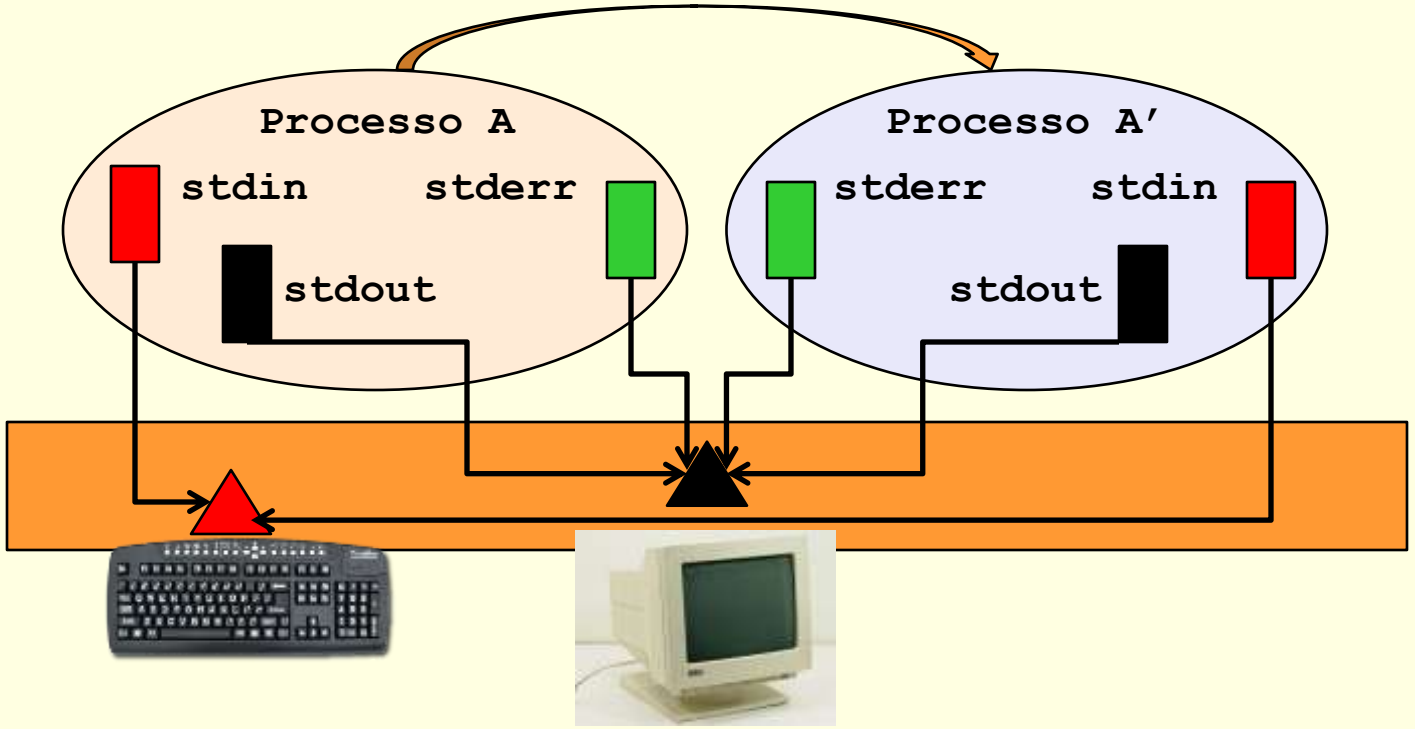


Canais standard em Unix e o `open()`

- *Na prática, isto quer dizer que:*
  - *A shell, e qualquer programa dela lançado pode, por omissão, ler do descritor 0 (e.g., teclado) e escrever nos descritores 1 (ecrã de saída) e 2 (ecrã de erros).*
  - *A execução de um novo* **`open()`**, *se efectuada com sucesso, retornará um descritor com o número 3.*
- *Em geral,*
  - *O número de descritor retornando por um open() é o menor inteiro que não está associado a um canal (aberto). Por exemplo,*
    - *Se num dado instante estão "em uso" os descritores 0, 1, 2 e 3, fecharmos o 2 e efectuarmos um novo* **`open()`**, *este retornará 2.*

Canais standard em Unix e o `fork()`

- *Num* **`fork()`** *os descritores do processo filho referenciam os mesmos canais - e dispositivos - que os do pai.*



Operações fundamentais de E/S: `close()`

- **`int close(int descritor)`**

Se o descritor "está aberto" (ou seja, está associado a um canal), então esta operação dissocia-os, deixando o descritor de poder ser usado para aceder a esse canal – e então qualquer tentativa subsequente de usar o descritor redunda em erro…

Se o descritor não "está aberto" , a operação falha com erro.

Operações fundamentais de E/S: `read()`

- **`int read(int descr, void *buf, size_t sz)`**

Se a execução da função tiver sucesso, a variável **`buf`** *é preenchida com uma quantidade não superior a* **`sz`** *de bytes lidos do ficheiro…*

- *Se o* **`read()`** *retorna zero, então não há mais dados para ler [se o canal está associado a um ficheiro, estamos no fim-de-ficheiro]*
- *Se o* **`read()`** *retorna um valor* **`nb`** *positivo mas inferior a* **`sz`**, *então foram lidos* **`nb`** *bytes, mas não a totalidade, porque não havia dados suficientes para ler.*
- *Se o* **`read()`** *retorna o valor* **`sz`**, *então foi lida a quantidade exacta de bytes que pretendíamos ler.*

Operações fundamentais de E/S: `write()`

- **`int write(int descr, void *buf, size_t sz)`**

Se a execução da função tiver sucesso, o conteúdo da variável **`buf`** *(numa quantidade não superior a* **`sz`** *de bytes) é transferido para o canal [escrito no ficheiro, se o canal está associado a um ficheiro]*

- *Se o* **`write()`** *retorna zero, então nada foi escrito*
- *Se o* **`write()`** *retorna o valor* **`sz`***, então foi escrita a quantidade exacta de bytes que pretendíamos.*
- *Se o* **`write()`** *retorna -1, houve um erro; consultar o código do erro, e o manual para conhecer a sua causa*