



Processos ao nível do SO

José C. Cunha, DI/FCT/UNL

Concorrência em sistemas de operação

- Periféricos operando simultaneamente
- Multiprogramação
- Eventos em instantes imprevisíveis

SO monolíticos

- Programas e periféricos pedem serviços ao SO
- Todas as acções do SO com interrupções desligadas
- SO fica mais simples, mas...
- Processos e periféricos ficam a aguardar muito tempo
- SO grandes ficam muito complexos

SO baseados em processos

- Nem todas as acções do SO com interrupções desligadas
- Maior descentralização das acções do SO
- Processos:
 - entidades assíncronas interrompíveis
 - partilham o CPU e passam por estados executando, bloqueado, pronto
- Processo para acções de 'device driver'
- Processo para execução de programa

Noção de processo associada a device driver para cada periférico

Responsável pela sua operação física

Pela detecção do fim de operações e erros

Pelo tratamento de interrupções

Pelo serviço dos pedidos dos programas

Vantagens:

- maior clareza nos módulos do SO
- estruturas de dados descentralizadas e sob controlo de processos dedicados
- maior eficiência no atendimento dos pedidos dos programas e dos periféricos na gestão da concorrência

O Núcleo do SO

- Distribui o tempo de CPU pelos processos
- Atende interrupções e assinala-as aos respectivos processos responsáveis, sob a forma de sinais
- Oferece funções para:
 - Processos fazerem pedidos ao SO
 - Processos comunicarem entre si e sincronizarem as suas acções concorrentes

Processos no SO

Ambiente de execução de um programa:

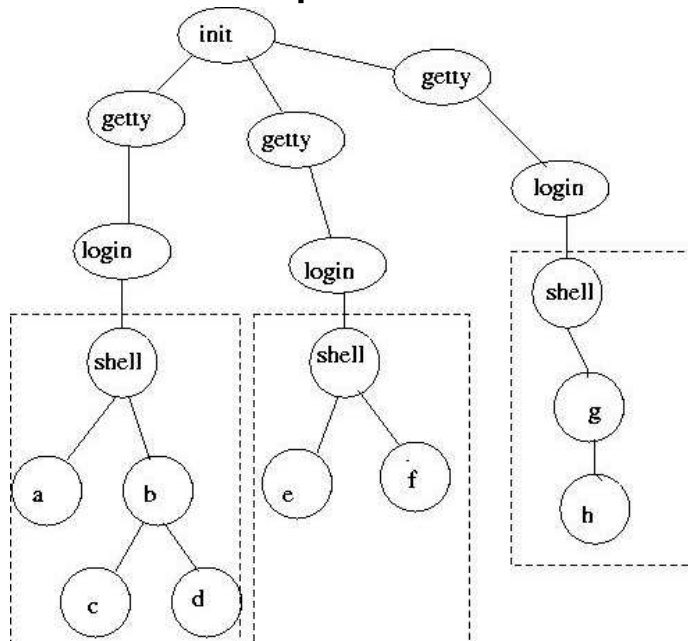
- Programa executável
 - Zonas de dados e pilha
 - Salvaguarda de estado do CPU (PC, SP outros registadores)
 - Informação sobre canais abertos para ficheiros
 - Outra informação de controlo do processo
- Guardada em Descritores internos do SO
(Process id PID -- aponta para descritor)

Exemplo de uso de processos

A nível do Shell no Unix:

- ler e interpretar comandos do terminal
- criar um processo por cada comando
- destruir o processo no fim da execução

Árvore de processos no Unix



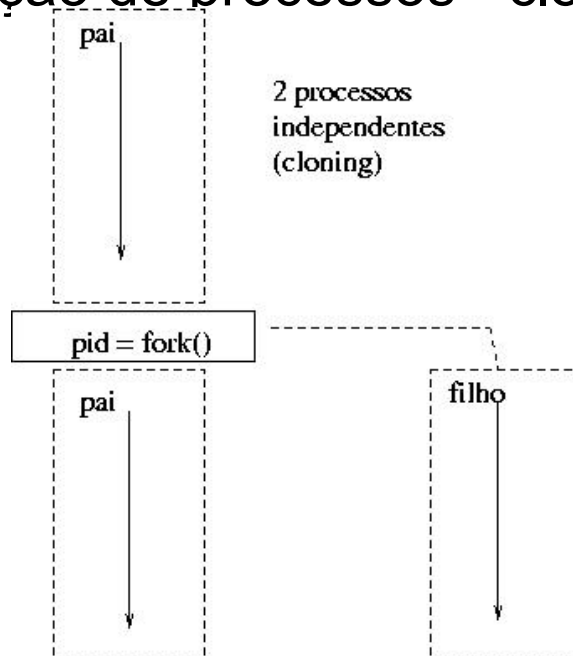
Classes de chamadas ao SO

- Acesso a ficheiros e directorias
- Pedir e libertar memória
- Pedir a execução de novos programas
- Controlar a execução de programas
- Comunicação e sincronização de processos concorrentes
- Obter informação de estado do sistema

Chamadas Unix para Processos

- Fork: cria um duplicado exacto do processo original incluindo informação sobre canais abertos
- Exec: nova imagem executável para o processo, a partir de um ficheiro executável -- um novo mapa de memória
- Wait: permite a um processo aguardar pela terminação de um filho
- Exit: termina o processo corrente
- Getpid: indica o identificador do processo

Criação de processos - clonagem

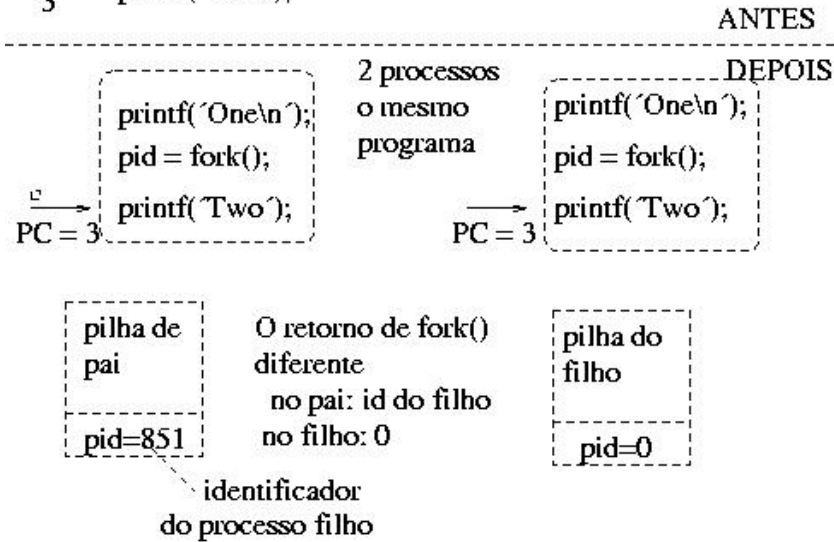


Exemplo de fork

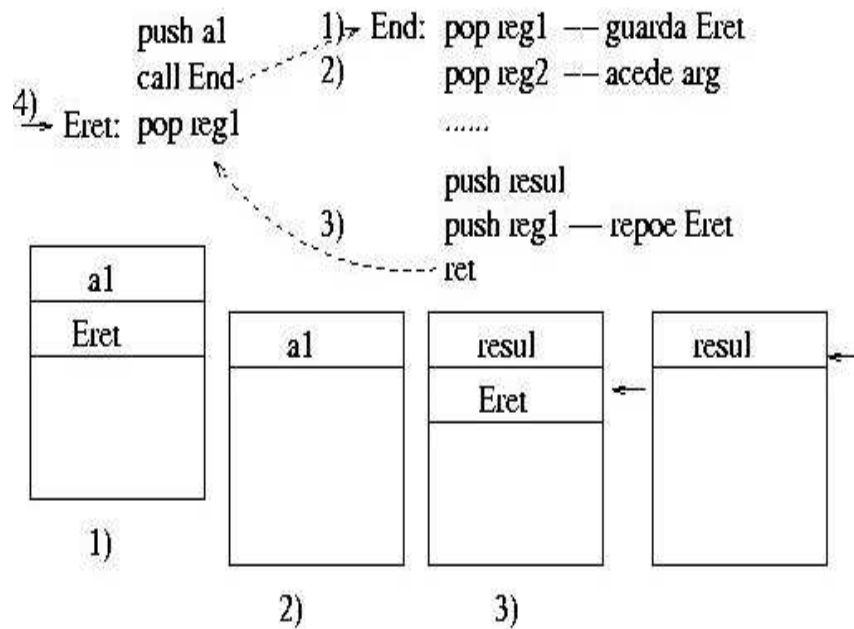
```

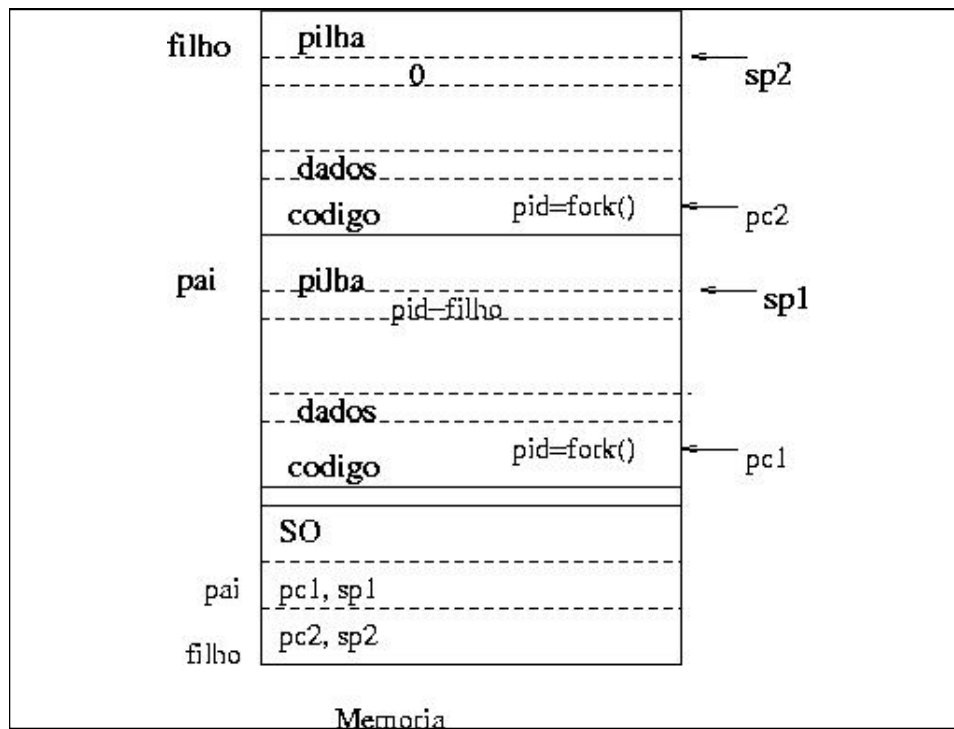
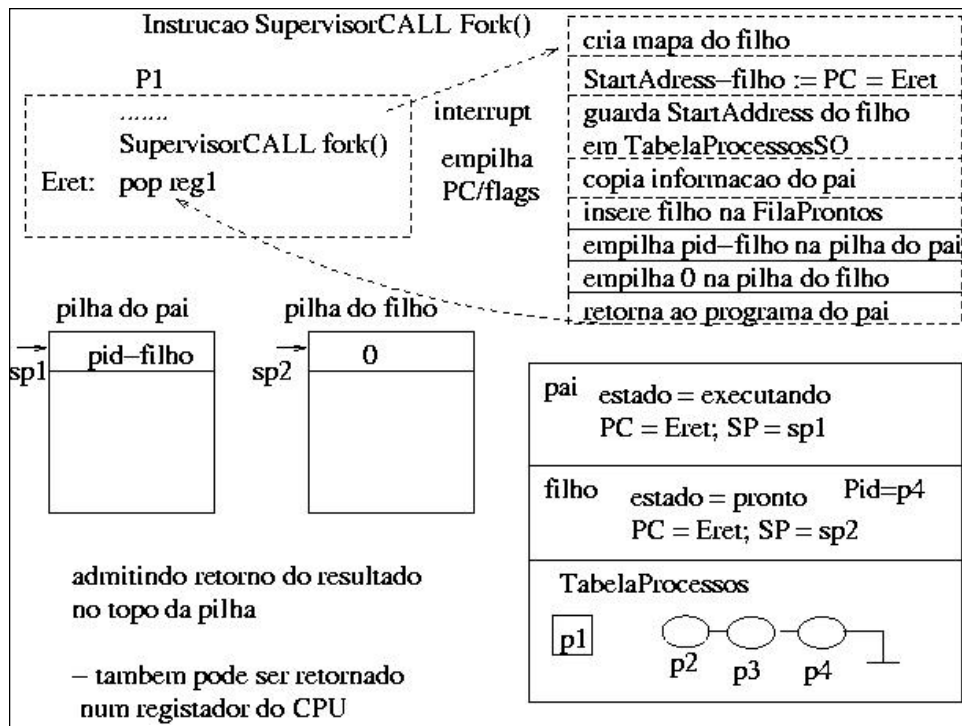
1  printf('One\n');
2  pid = fork();
3  printf('Two\n');
    
```

PC = 2 1 processo
1 programa

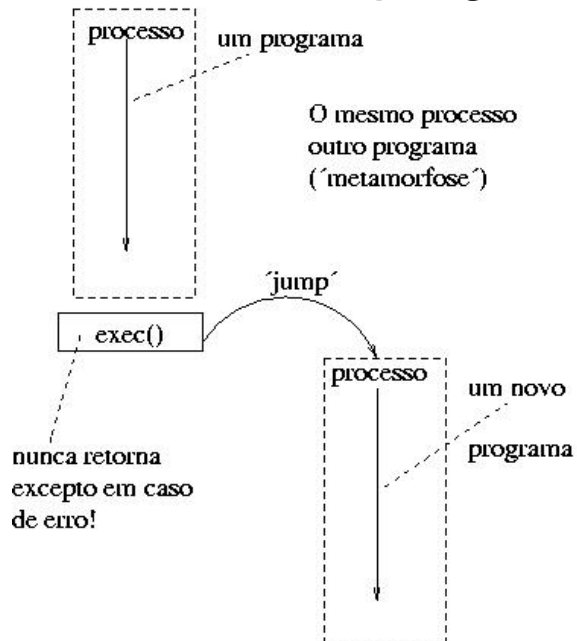


Revisao da instrucao CALL Subrotina

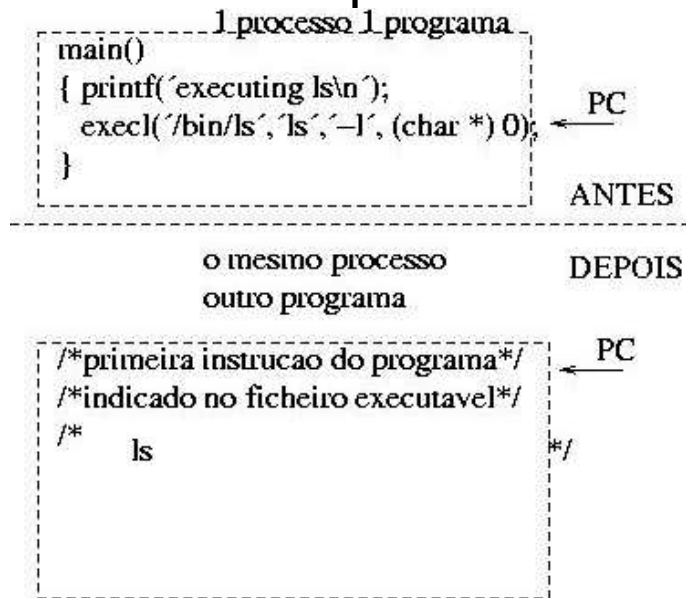




Execução de novo programa



Exemplo de exec



```

pid = fork ();
if (pid == 0)
    -- filho
else if (pid > 0)
    -- pai
else /* pid < 0 */
    -- erro

```

Exemplo de fork e exec

```

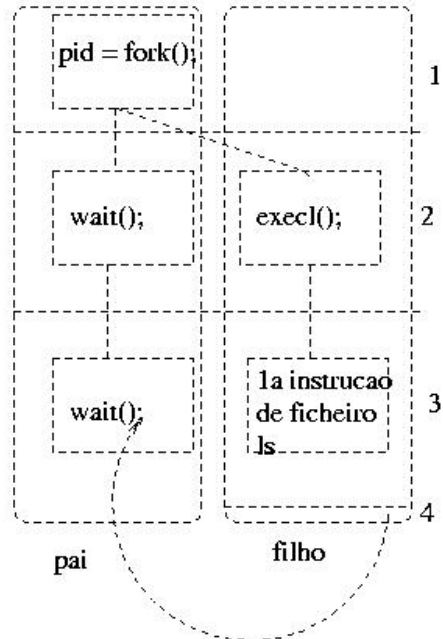
main()
{ int pid;

  pid = fork();

  if (pid>0) { /* pai*/
    wait( (int *) 0 );
    printf("ls completed\n");
    exit(0);
  }

  if (pid == 0) { /*filho*/
    execl("/bin/ls", "ls", "-l", (char *)0);
    .....
    ..... /*se pid <0*/
  }
}
o mesmo programa

```



Shell

```
while (true) {  
    read_command(command, params);  
    if ((p = fork()) != 0)  
        wait(&status);  
    else  
        execve(command, params, 0);  
}
```

Um Shell simplificado

```
while (true) {  
    read_command(command, params);  
    if (fork () != 0)  -- cria  
        wait (&status);  -- o pai espera  
    else  -- o filho executa o comando  
        execve(command,params,0);  
}  
cp file1 file2    na linha de comandos  
main(argc, argv, envp) no programa
```

FORK	EXEC
mapa do pai duplicado para o filho copia privada	mapa do invocador esmagado e criado um novo mapa para um novo programa executavel
retorna um valor de pid no pai diferente do valor retornado no filho	nunca retorna excepto em caso de erro
o filho herda uma copia dos canais	mantem os canais abertos (por omissao)
pid do pai difere do pid do filho 2 processos com igual programa	pid mantem-se pois o processo nao mudou o programa e ´ que mudou



Controlo de Processos (cont.)

- Família de funções exec()
- Funções wait() e waitpid()

pid_t fork(void);

exec():

uma família de funções que chamam
uma chamada ao SO

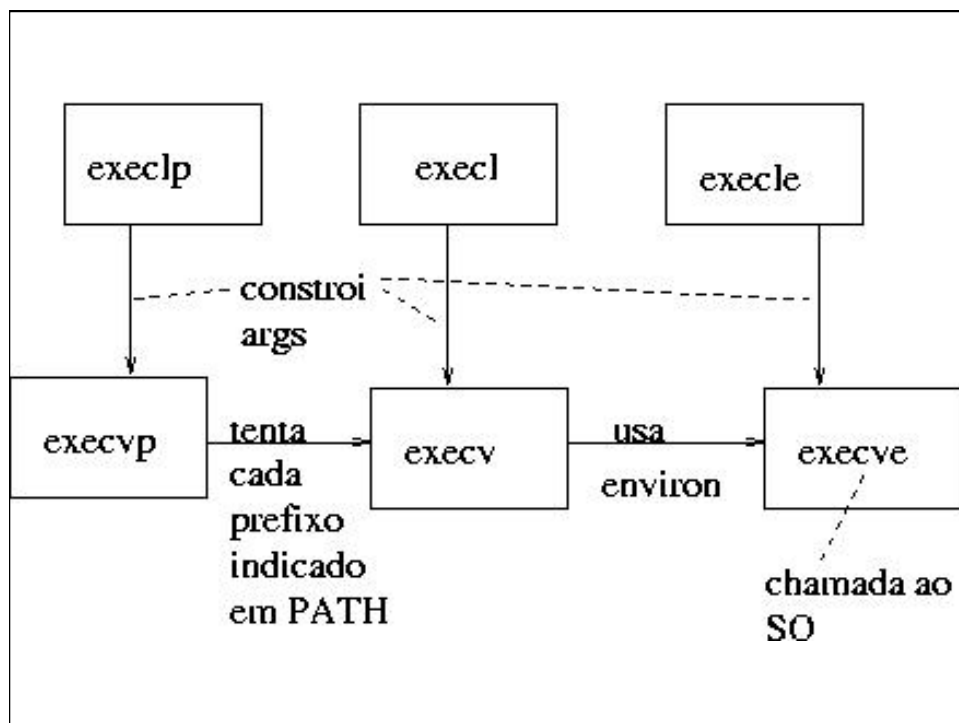
execve()

execl – lista de argumentos

execv – vector de apontadores para args

execp – usam variável PATH

exece – usam vector para vars ambiente



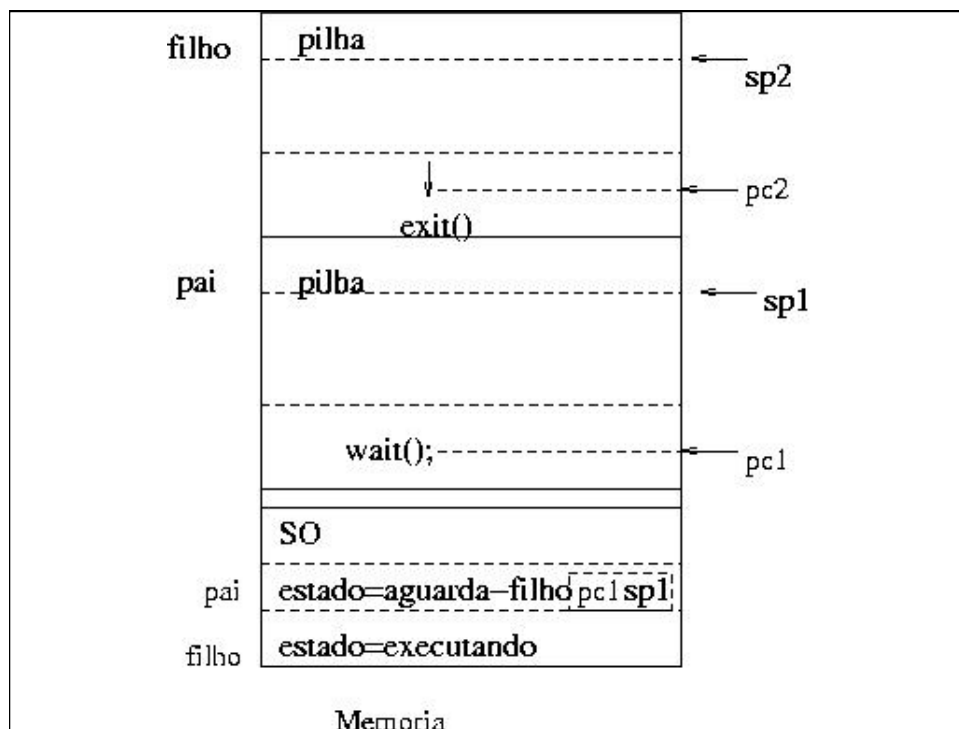
wait e waitpid

`pid_t wait(int *status)`

- aguarda pela terminação de um dos filhos

`pid_t waitpid(pid_t pid, int *status, int options)`

- se `pid = -1` `wait()`
- se `pid > 0` aguarda pelo filho indicado 'pid'
- options: `WNOHANG` devolve 0 de imediato se o filho não tiver ainda terminado.
- status: pode ser obtido usando macros



O pai e o filho são executados
concorrentemente pelo sistema de
multiprogramação.

Cenários possíveis:

1. Pai chega primeiro à chamada de wait()
2. Filho chega primeiro a exit() → ZOMBIE
quando pai fizer wait(), pai prossegue e filho
é eliminado.
3. Pai não faz wait() e termina antes de filho
→ filho fica ÓRFÃO: quando filho termina, é
adoptado por INIT

Atributos de cada processo

Process id	getpid	
Parent process id	getppid	
Login user name	getlogin	
Real user id	getuid	
Current directory	getcwd	chdir
(herdada do pai via fork)		
File mode creation mask	umask	
.....mais outros atributos		

Variáveis do ambiente

Exemplos:

- HOME – home directory
- LOGNAME – login name
- PATH – lista de prefixos de caminhos para pesquisar ficheiros executáveis

.....

São um conjunto de strings nome=valor

São passadas de pais para filhos e mantidas no exec.

Definir variáveis do ambiente

- automaticamente definidas no login (HOME, USER)
- ou – explicitamente pelo utilizador
- ou – definidas num ficheiro de inicializações

Environment list é passada ao programa através de uma variável global: char **environ

- um apontador para um vector de apontadores para strings

