



Aula Prática N°8

IPC— Processos e Comunicação entre Processos

Processos e Sinais em Unix

Objectivo

Estudo de processos e sinais em Unix. Alteração da resposta por defeito dos processos a sinais. Utilização das *chamadas ao sistema* `fork`, `execl`, `wait`, `waitpid`, `sigaction` e `kill`.

Guião

1. Criação de processos

- a) Consulte no manual *on-line* a descrição das chamadas ao sistema `fork`, `getpid` e `getppid`.
- b) Leia atentamente o código fonte `fork1.c` e procure responder às questões seguintes sem executar o programa.
 - i) Quantas linhas são impressas no ecrã do monitor vídeo quando o programa é executado?
 - ii) Quem imprime o quê? Como pode verificá-lo quando o programa for executado?
 - iii) Construa um diagrama que identifique os processos que são lançados pela execução do programa, pondo em destaque as acções principais executadas por cada um deles.
- c) Crie o ficheiro executável `fork1` (*make fork1*), execute o programa e confirme as suas deduções. Quem é o *processo-pai* do programa em execução?

2. Distinção entre o processo-pai e o processo-filho

- a) Leia atentamente o código fonte `fork2.c`, crie o ficheiro executável `fork2` (*make fork2*), execute o programa e interprete os valores impressos.
- b) Explique como é que os processos envolvidos podem distinguir durante a execução quem é o *processo-pai* e quem é o *processo-filho*. De facto, o código apresentado já o faz. Como? E para que efeito?

Tarefa 1 – Altere o programa `fork2.c` de modo a que, após o `fork()`, o processo pai escreva PAI no ecrã e o processo filho escreva FILHO no ecrã.

3. Definição da acção a desenvolver pelo processo-filho

- a) Consulte no manual *on-line* a descrição da chamada ao sistema `execl`. Qual é a sua utilidade? Ela apresenta também a característica notável de poder ser invocada com um número variável de parâmetros. Como é que isso é conseguido e qual é o significado atribuído a cada um deles?
- b) Leia atentamente o código fonte `fork3.c` e procure responder às questões seguintes.
 - i) Os valores actuais dos dois primeiros parâmetros de invocação da função `execl` são o mesmo. Porquê?
 - ii) Qual é o comando de *shell* equivalente a esta invocação?
- c) Leia atentamente o código fonte `child.c`, crie o ficheiro executável `fork3` (*make fork3*) e o ficheiro executável `child` (*make child*), execute o programa e confirme as suas deduções. Procure também responder às questões seguintes.
 - i) A instrução `printf`, imediatamente a seguir à invocação de `execl` do código fonte `fork3.c`, nunca é executada. Porquê?
 - ii) As duas mensagens impressas pelo processo `child` são ligeiramente diferentes. Qual é a diferença? Porque é que o valor do PPID na segunda mudou? Para que processo mudou?
 - iii) Note também o posicionamento do *prompt*. Qual será a causa desta anomalia?

Tarefa 2 – Altere `fork3.c` de modo que o processo filho execute o comando `ls -l`.

4. Sincronização entre o processo-pai e o processo-filho

- a) Consulte no manual *on-line* a descrição das chamadas ao sistema `wait` e `waitpid`. Qual é a diferença entre elas?
- b) Leia atentamente o código fonte `fork4.c` e procure responder às questões seguintes.
 - i) O que é que é fundamentalmente modificado relativamente à versão `fork3`?
 - ii) Qual vai ser agora a ordem com que as instruções de impressão vão ser executadas?
 - iii) Onde é que vai ser agora posicionado o *prompt*?
- c) Crie o ficheiro executável `fork4` (*make fork4*), execute o programa e confirme as suas deduções.

Tarefa 3 – Usando as chamadas ao sistema `fork`, `execl` e `wait`, escreva um programa designado `myls` que execute o comando `ls -la` ladeado no topo e na base pela linha “=====”.

5. Sinais e a interrupção de processos

- a) Leia atentamente o código fonte `sig1.c`. O que é que o programa é suposto fazer? Note em particular o papel desempenhado pela instrução `fflush`. Qual é ele?
- b) Crie o ficheiro executável `sig1` (*make sig1*), execute o programa e confirme as suas deduções.
- c) Execute de novo o programa e prima durante a sua execução a combinação de teclas

que exprime `CTRL-C`. O que é que acontece?

- d) Execute de novo o programa e prima durante a sua execução a combinação de teclas que exprime `CTRL-Z`. O que é que acontece? Após premir `CTRL-Z` experimente utilizar os comandos `fg`. Prima novamente `CTRL-Z` e experimente o resultado do comando `bg`. Execute o comando `jobs`.

6. Alteração da rotina de serviço a um sinal

- a) Consulte no manual *on-line* a descrição da chamada ao sistema `sigaction`.
- b) Leia atentamente o código fonte `sig2.c` e procure responder às questões seguintes.
- i) Porque é que se premir agora durante a execução do programa a combinação de teclas que exprime `CTRL-C`, não se verificará a sua terminação?
 - ii) O que é que vai concretamente acontecer?
- c) Crie o ficheiro executável `sig2` (*make sig2*), execute o programa e confirme as suas deduções.
- d) Consulte no manual *on-line* a descrição do comando `kill`.
- e) Volte a executar o programa e memorize a identificação do processo correspondente (PID). Lance outro terminal e execute nele o comando `kill -SIGINT PID`, em que `PID` é o identificador do processo que memorizou. Compare com a alínea c).
- f) Execute o comando `kill -SIGSTOP PID` enquanto `sig2` está a executar. Como pode retomar a execução do programa?
- g) Execute agora um dos comandos `kill -SIGTERM PID`, `kill -SIGKILL PID` ou `kill PID` para terminar efectivamente o processo.

Tarefa 4 – Escreva um programa, designado de `sig3.c`, usando como base o programa `sig2.c`, que faça terminar o processo à quinta interrupção (quinta vez que a combinação de teclas que exprime `CTRL-C` é premida).

Sugestão: conte o número de vezes que a rotina de atendimento do sinal é invocada e, à quarta vez, reinstale a rotina de atendimento por defeito.