Sistemas Operativos **Ano lectivo 2025/2026**

Guião desenvolvido por Artur Carneiro Pereira

Aula Prática Nº 8

Objetivos

Estudo de processos e sinais em Unix. Alteração da resposta por omissão 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 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 lançados pelo programa, pondo em destaque as acções principais que cada um executa.
   c) Crie o ficheiro executável `fork1` (*make fork1*), execute-o e confirme as suas deduções. Qual é o *processo-pai* do programa?
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 e interprete os valores impressos.
   b) Explique como é que os processos envolvidos podem distinguir qual deles é o *processo-pai* e qual é o *processo-filho*. De facto, no código apresentado já o fazem. Como? E para que efeito?

Tarefa 1 – Altere o programa `fork2.c` de modo 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 a característica notável de poder ser invocada com um número variável de parâmetros. Como é isso 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 dois primeiros parâmetros de invocação da função `execl` são iguais. Porquê?
      ii. Qual é o comando de *shell* equivalente a esta invocação?
   c) Leia atentamente o código-fonte `child.c` e crie os ficheiros executáveis `fork3` (*make fork3*) e `child` (*make child*). Execute o programa `fork3` e procure responder às questões seguintes.
      i. A instrução `printf` imediatamente a seguir a `execl` em `fork3.c` nunca é executada. Porquê?
      ii. As duas mensagens impressas pelo processo `child` são ligeiramente diferentes. Qual é a diferença? O valor do PPID na segunda mudou. Porquê? Para que processo?
      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 fundamentalmente modificado relativamente a `fork3.c`?
      ii. Em que ordem vão agora ser executadas as instruções de impressão?
      iii. Onde vai agora ser posicionado o *prompt*?

   c) Crie o ficheiro executável `fork4` (*make fork4*), execute-o 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` fazendo aparecer no topo e na base da listagem linhas como a seguinte: "`==========`".

5. Sinais e a interrupção de processos.

   a) Leia atentamente o código-fonte `sig1.c`. O que é suposto que este programa faça? Note, em particular, a instrução `fflush`. Que papel desempenha?

   b) Crie o ficheiro executável `sig1` (*make sig1*), execute-o e confirme as suas deduções.

   c) Execute de novo o programa e prima durante a sua execução a combinação de teclas que exprime `CRTL-C`. O que acontece?

   d) Execute de novo o programa e prima durante a sua execução a combinação de teclas que exprime `CRTL-Z`. O que acontece? Após premir `CTRL-Z` experimente utilizar os comandos. Descubra o efeito do comando `fg`. Prima novamente `CTRL-Z` e teste o efeito 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`. Crie o ficheiro executável `sig2` (*make sig2*) e execute-o. Agora, se durante a execução do programa premir a combinação de teclas que exprime CRTL-C, não se verifica a sua terminação. Porquê? O que vai, concretamente, acontecer?

   c) Crie o ficheiro executável `sig2` (*make sig2*), execute-o 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 `sig3.c` , adaptado de `sig2.c`, que faça terminar o processo à quinta interrupção (i.e. à quinta vez que a combinação de teclas que exprime `CRTL-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 omissão.