ISEP – LEEC - Sistemas Computacionais 2016/2017

Ficha 3 - Programação multi-processo em Unix

Objetivos

O aluno deverá ser capaz de aplicar as funções básicas relacionadas com a criação e gestão de processos (*fork*(), *wait*(), *waitpid*(), *system*(), família *exec*())
O aluno deverá ser capaz de analisar as interações existentes entre processos executados numa mesma sessão de Unix, no que respeita à gestão de memória e à execução concorrente.

Exercícios

1) Considere o seguinte programa:

```
1:  int i = 0;
2:
3:  int main() {
4:
5:    pid_t r = fork();
6:    if(r == 0)  {
7:      sleep(10);
8:      printf("%d: %d %d\n", getpid(), i, r);
9:      return 0;
10:   }
11:
12:   i = i + 1;
13:   wait(NULL);
14:   printf("%d: %d %d\n", getpid(), i, r);
15:
16:   return 0;
17: }
```

   a) Indique a sucessão de mensagens impressas no ecrã durante a execução do programa.

   b) Por que motivo a impressão da linha 8 é feita antes da impressão da linha 14 apesar da pausa de 10 segundos da linha 7?

   c) Indique a sucessão de mensagens no caso da linha 9 ser omitida.

2) Execute e analise o seguinte programa:

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void simula_processamento();

int main() {
  int r, i, status;

  r = fork();
  printf("pid = %d, ppid = %d\n", getpid(), getppid());
  sleep(1);
  srandom(getpid());//

  if(r==0) {
    for(i=0; i<20; ++i) {
      simula_processamento();
      printf("%d ", i);
      fflush(stdout);
    }
    exit(1);
```

```
  }

  for(i=20; i<40; ++i) {
    simula_processamento();
    printf("%d ", i);
    fflush(stdout);
  }

  r = wait(&status); //aguarda que processo filho termine
  printf("\n");
  if(WIFEXITED(status))
    printf("Valor de retorno de (%d): %d\n", r, WEXITSTATUS(status));
  else
    printf("Filho (%d) terminou de forma anormal\n", r);

  return 0;
}


#define NELEM 64
void simula_processamento() {
  int i,j;
  double d1[NELEM];
  for(i=0;i<random();++i) {
    for(j=0;j<NELEM;++j)
      d1[j]=d1[j]*1.1;
  }
}
```

Observe que tanto o processo criado como o processo inicial são executados concorrentemente. De que forma se verifica esse facto?

3) Escreva um programa que execute a seguintes sequência de ações:
   - Alocar um bloco de 8 bytes da memória de dados (`char *ptr = malloc(8)`).
   - Guardar a *string* "pai" no bloco alocado (use a função *strcpy*()).
   - Criar um processo filho que deverá escrever a *string* "filho" no bloco apontado por *ptr*, fazer a impressão do conteúdo do mesmo no ecrã e terminar.
   - Aguardar a conclusão do processo filho (*waitpid*()) e imprimir a *string* apontada por *ptr*.

Qual o resultado esperado?

4) (Processos "orfão")
   Escreva um programa que crie um novo processo. Após a criação do novo processo, o processo inicial deverá aguardar 1 segundo (use a função *sleep*()) e terminar. O processo filho deverá começar por imprimir os seus PID (*getpid*()) e PPID (*getppid*()), esperar o tempo suficiente para que o processo inicial termine (use novamente a função *sleep*()) e voltar a imprimir os seus PID e PPID.

5) (Processos "zombie")
   a) Escreva um programa que crie um novo processo. O processo filho deverá imprimir os seus PID e PPID e terminar. O processo pai deverá aguardar alguns instantes (*sleep*(5)) e então executar o comando "ps -f*"* (use a função *system()*). Antes de terminar, o pai deverá imprimir um aviso de que vai terminar. Após a terminação do programa volte a executar o comando "ps -f*",* na *shell*. Analise os resultados.
   b) Repita o exercício usando uma das funções *exec*() (e.g., *execlp*) em vez da função *system*() para executar o coamndo "ps -f". Repare que a mensagem final deixa de aparecer no ecrã. Porquê?
      Nota: pode comparar as várias funções da família *exec*() escrevendo o seguinte comando na *shell*: *man 3 exec*