Processos em Sistemas Operacionais

Processo é um conceito utilizado em sistemas operacionais para identificar um elemento que concorre à execução.
Um mesmo programa pode estar sendo executado por vários usuários, ao mesmo tempo, mas para cada usuário existe um processo.
Cada processo trabalha sobre uma área de memória privativa

Criação de Processos

A chamada ao sistema `fork` cria um novo processo (processo filho).
A chamada `fork` devolve um identificador do processo (PID). Ao processo Pai este identificador retornado é diferente de zero e ao novo processo criado (processo Filho) é zero. Em caso de erro, a função devolve -1 ao processo Pai e o processo Filho não é criado.

```
Protótipo da função fork:
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

Término de Processos

A chamada ao sistema `exit` permite o término de um processo a qualquer altura de sua execução (fecha todos os arquivos abertos).

```
Protótipo da função exit:
#include <stdlib.h>

void exit(int status);
```

Exercício 1

Compile e execute o código a seguir, nomeando o fonte como `processo.c`.
Quais os valores de PID para cada processo?
Identifique a finalidade de cada chamada ao sistema utilizada.

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

int main()
{
       pid_t procID;

       procID = fork();

       if (procID < 0)
       {
             printf("Erro na criação do novo processo\n");
             return -1;
       }
       else if (procID == 0)
            {
             printf("Processo filho - para o FILHO o fork devolveu %d\n", procID);
             printf("Processo filho - PID = %d\n", getpid());
             return 1;
            }
            else
            {
             printf("Processo Pai - para o PAI o fork devolveu %d\n", procID);
             printf("Processo Pai - PID = %d\n", getpid());
             return 1;
            }
}
```

Espera por Processo

A chamada ao sistema `wait` causa bloqueio do processo pai até que um processo filho termine. Esta função devolve o identificador do processo filho.

```
Protótipo da função wait:
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

Exercício 2

Compile e execute o código a seguir, nomeando o fonte como `processo2.c`.
Como acontece a execução deste código?

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

int f1(int x)
{
       printf("x = %d", x);

}

int main()
{
       pid_t procID;
       procID = fork();

       if (procID < 0)
       {
             printf("Erro na criacao do novo processo\n");
             return -1;
       }
       else if (procID == 0)
            {
             printf("Processo filho - PID = %d\n", getpid());
             f1(100);
             printf("Filho executou a funcao f1 do Pai...\n");
             return 1;
            }
            else
            {
              wait(NULL);
              printf("Processo Pai - PID = %d\n", getpid());
             f1(50);
              printf("Pai executou a funcao f1...\n");
              return 1;
            }
}
```

Chamada de Programas

O código do processo filho pode executar um outro programa. Para isso, usa-se a função execl (ou outra semelhante).

```
Exemplo: execl("./arq", "0", "0");
```

Exercício 3

Compile e execute o código a seguir, nomeando o fonte como `processo3.c`
Obs.: na função execl, o primeiro argumento indica o caminho e o nome do arquivo a ser executado; os demais são argumentos que podem ser passados ao programa chamado.
Identifique as chamadas ao sistema utilizadas

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

int main()
{
       pid_t procID;
       procID = fork();

       if (procID < 0)
       {
             printf("Erro na criacao do novo processo\n");
             return -1;
       }
       else if (procID == 0)
            {
             printf("Processo filho - PID = %d\n", getpid());
             //execl("./procTwo", (char *)NULL);
             //execl("/bin/ls", "ls", "-l", (char *)NULL);
            // execl("/bin/ps", "ps", "-aux", (char *)NULL);
             exit(0);
            }
            else
            {
              printf("Processo Pai - PID = %d\n", getpid());
              return 0;
            }
}
```

Exercício 4

Quantos processos são criados no exemplo abaixo? Justifique.

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

int main() {

  fork();
  fork();

  printf("PID = %d\n", getpid());

  return 1;
}
```

Exercício 5

Execute o código abaixo e verifique o valor da variável `s` em cada um dos processos. Os processos ocupam o mesmo espaço de endereçamento? Justifique.

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

int s = 0;

int main() {

  if (fork() == 0) {
    s = 10;
    printf("Processo filho, s = %d\n", s);
  } else {
    wait(NULL);
    printf("Filho terminou a execução!\n");
    printf("Processo pai, s = %d\n", s);
  }

  return 0;
}
```