Universidade Federal de Uberlândia

Disciplina: Sistemas Operacionais (SO)

Prática: Processos

Processos em Ambiente LINUX

Nessa atividade, os estudantes devem explorar os conceitos sobre processos (tarefa) em ambiente do Linux. São apresentados comandos para que se familiarizarem com os comandos para gerenciamento de processos. Também serão explorados a chamada de sistema fork e vfork durante a análise dos códigos em linguagem C.

Processos em Linux

1. Processos

1.1 - A relação de processos existentes no sistema operacional Linux pode ser obtida com o comando "**ps**". Para obter uma relação detalhada, com várias estatísticas, usa-se o comando "**ps -*aux***". Com uso do comando "**top**" também observa-se os processos. Utilizando o comando **top,** determine em qual coluna é possível verificar a prioridade de um processo?

1.2 - O comando **kill** tem como objetivo enviar sinais para os processos. Há vários sinais diferentes que o **kill** pode enviar para um processo. Veja a lista de todos os sinais conhecidos através do comando "**kill -l".**

A função de cada sinal pode ser obtida consultando com a man page: "**man 7 signal**". O uso do comando **kill -9 <PID> f**inaliza um processo no sistema. Acesse com o comando **top** o número de processo e use o comando **kill** para finalizar um processo.

1.3 - A forma de analisar os detalhes de um processo pode ser explorado na pasta /proc. Neste caso, acesse o diretório **cat /proc/<PID>/?** para informações sobre um processo no S.O.

2. Programas para Trabalhar com Processo

2.1 - Um simples editor de textos permite edições rápidas (vi, vim, gedit , etc). Usando o gedit, execute-o digitando: **gedit <nome do arquivo>.c** - esse comando criará um novo arquivo para você trabalhar.

Os programas C e C++ são compilados com os compiladores **gcc** (GNU C Compiler) e **g++** (GNU C++ Compiler). A sintaxe para compilação em Linguagem C é : "**gcc -w -o nome nome do programa.c**".

a) Compile e execute o código do arquivo **fork1.c.** Utilize o comando **strace -c** e observe as chamadas de sistema que ocorreram durante a execução desse programa. Quantas chamadas aconteceram? Análise o código e qual a função das chamadas de sistema getpid() e getppid()?

b) Compile e execute o código do arquivo **fork2.c**. Faça a análise das chamadas de sistema que ocorrem durante a execução desse código. O programa foi finalizado com a chamada **"return 0"**? Isso não ocorreu devido a qual instrução do código? Use a chamada "**sleep(1)"** e verifique se há alguma mudança no código. Todos os processos foram finalizados após essa mudança? Execute pelo menos três vezes seu código para resposta a essa questão.

c) Compile e execute o código **fork3.c**. Qual o valor da variável "**value"** para o processo pai e para o processo filho? Implemente mensagens no código que mostrem esses valores. Justifique por que esses valores são diferentes. O que acontece no mível do sistema operacional entre os processos e a variável global?

d) Compile e execute o código **fork4.c**. Os processos, pai e filho, são executados, separadamente, e cada um chama a função **adjustX()** com parâmetros diferentes em cada processo. Vamos examinar o código-fonte para entender as instruções que são executadas. Qual a saída para os processos (pai e filho)? Enquanto o programa executa, use o comando "**ps xl" (em outro terminal)** para acompanhar os processos criados por esse programa. Qual o PID do processo pai e do processo filho? (Dica, verifique a coluna PID). Use o comando "**kill -9 PID**" para finalizar processos. Use o comando "**kill -9 PID**" para finalizar o processo pai. O que aconteceu com o processo filho? Rode o programa novamente, identifique e finalize o processo filho, primeiro e, em seguida, o pai. O que aconteceu? Faz diferença finalizar o pai ou o filho antes? Há algum processo orfão durante essas execuções?

e) Compile e execute o código **fork5.c**. Quantos processos são criados assumindo que nenhuma chamada de sistema irá falhar?

```
fork();
fork();
fork();
```

Usando essas mesmas chamadas com o fork, o que ocorre quando você implementa a modificação descrita abaixo no código do programa fork5.c. Explique por meio de um fluxograma as etapas de criação dos processos.

```
...
int pid;
pid = fork();
if (pid == 0){
     fork();
     fork();
  }
...
```

f) Construa um programa em C com base no código e explique qual será a saída na LINHA A.

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

/*variavel global */
int value = 5;

int main(){
 pid t pid;
 pid = fork();
```

```
 if (pid == 0) { /* processo filho */
   value += 15;
   return 0;
 }
 else if (pid > 0) { /* processo pai */
  wait(NULL);

 printf("PAI: valor = %d",value); /* LINHA A */
 return 0;
}
```

g) O comando **fork** duplica a maior parte dos recursos do processo pai. Descreva quais são as principais diferenças desse comando em relação ao comando **vfork** do Linux. Compile e execute o código **vfork1.c**. O que acontece com a variável global nesse código. Justifique por que isso ocorre com essa chamada.