
 INSTITUTO DE INFORMÁTICA UFG	Universidade Federal de Goiás Instituto de Informática Sistemas de Informação	 UFG
Professor: Kleber Vieira Cardoso		
Laboratório: processos no Linux		

Embora as atividades a seguir possam ser realizadas no sistema hospedeiro (Ubuntu), é recomendado o uso da máquina virtual do Kali Linux, uma vez que o acesso privilegiado ao SO pode facilitar a solução de problemas.

1. Por padrão, o terminal (*shell* ou linha de comando), ao receber um comando do usuário, cria um processo filho e aguarda que ele se encerre para então aceitar outro comando. É possível visualizar esse comportamento facilmente através de uma aplicação que utiliza a interface gráfica. Por exemplo, execute o comando `gcalctool`. No entanto, é possível alterar esse comportamento. Descreva uma maneira como isso pode ser feito e explique o que está acontecendo em termos de chamadas ao sistema nos dois cenários (com espera e sem espera do processo filho).

2. Após iniciar o processo do programa `gcalctool` sem espera pelo processo pai (*shell*), aproveite que a linha de comando está disponível e descubra o PID (*process identifier*) do processo `gcalctool`. Qual o PPID (*parent PID*) desse processo? A qual processo pertence esse identificador?

3. Inicie novamente o processo do programa `gcalctool`, porém utilize a interface gráfica dessa vez, conforme ilustrado na figura abaixo. Utilizando o terminal, descubra qual o PID desse novo processo `gcalctool` e de quem ele é filho, ou seja, seu PPID. É diferente do PPID anterior? Por que?



4. Utilizando o terminal (*shell*), identifique o estado em que estão os processos `gcalctool`. A descrição dos estados pode ser vista na documentação do programa `ps` (basta digitar: `man ps`). Comente o que você entendeu sobre esse estado.

5. Também no terminal (*shell*), descubra há quanto tempo os processos `gcalctool` foram iniciados. Por fim, também no *shell*, encerre todos os processos `gcalctool`.

6. Analise o código do programa a seguir e, sem o compilar e executar, anote as ocorrências previstas de cada um dos caracteres indicados na Tabela 1. Em seguida, compile, execute o programa e anote as ocorrências observadas de cada caractere. Entendeu como a chamada de sistema `fork()` funciona?

```
#include <stdio.h>
#include <unistd.h>

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

    printf("A\n");
    if(pid==0)
    {
        printf("B\n");
    }

    printf("C\n");

    if(pid>0)
    {
        printf("D\n");
    }

    printf("E\n");

    return 0;
}
```

Caractere	Ocorrências previstas	Ocorrências observadas
A		
B		
C		
D		
E		

Tabela 1

7. Implemente um programa que crie N processos, todos filhos do mesmo pai. N é um valor fornecido pelo usuário. Todos os processos devem imprimir seu PID e PPID.

8. Implemente um programa que crie uma cadeia de processos N processos (além do pai). A cadeia deve obedecer a seguinte sequência: 1º processo cria o 2º processo, 2º processo cria o 3º processo, ..., Nº – 1 processo cria o Nº processo. Cada processo deve imprimir seu PID e o PID do seu pai (ou PPID). Garanta que a informação exibida na tela ocorrerá na ordem inversa da criação dos processos, ou seja, inicialmente aparece as informações do Nº processo (PID e PPID), depois do Nº – 1, ..., depois do 2º e por fim do 1º.

9. Implemente um programa que cria dois processos filhos. O primeiro grava em um arquivo números de 1 a 10. O segundo grava em um arquivo letras de A a Z. O processo pai, após cada filho terminar sua execução, lista o conteúdo dos arquivos criados na tela.

10. Implemente um programa que permita o usuário entrar com um comando e seus parâmetros, e execute o comando. Faça uso das chamadas `fork()` e `exec()`.