Matheus Vinicius da Costa - 28/03/2024

---

1 - Execute o comando ps aux e identifique três programas do sistema (daemons) e três programas do usuário, explicando os valores cada uma das colunas para um de cada tipo (sistema e usuário).

R:
> Daemons : rtkit-daemon, chroot helper, accounts-daemon,.

> Sys/User : gnome-shell, firefox-esr, pipewire.

Colunas:
• USER: Indica qual usuário iniciou o processo.
• PID: Identificador de processo.
• CPU%: Tempo de CPU.
• MEM%: Memória sendo usada pelo processador.
• RES: A memória não swap que uma tarefa usou.
• PRI: Prioridade do processo
• NI: Prioridade, nível amigável.
• VIRT: Memória Virtual.
• SHR: Memória compartilhada. Shared
• S ou STAT: Status do processo.
• Time: Tempo do processo desde sua init.
• Command: Nome do processo/comando.

---

2 - Há processos zombies executando em seu sistema operacional? Posso eliminá-los do sistema usando o comando kill -SIGKILL pid_zombie? Justifique.

R:
Não há processos zombies no momento, portanto o comando "kill -SIGKILL pid_zombie" não pode eliminar tais processos porque eles não existem ou já estão mortos.
O comando proprio é enviar o sinal SIGCHLD para o processo pai com > kill -s SIGCHLD [ppid]

ppid = pid do processo pai

---

3 - Quais os processos com maior utilização de CPU? Quais os processos

com maior utilização de memória? Qual o processo do usuário está a mais tempo em execução?

R:
• CPU: Firefox
• Memória: Firefox
• Tempo: Firefox

> Colocar filtro em cpu,mem e time
ps -eo pid,cmd,%mem,%cpu,time –sort=-%cpu
ps -eo pid,cmd,%mem,%cpu,time –sort=-%mem
ps -eo pid,cmd,%mem,%cpu,time –sort=-time

---

4 - Como eu faço para suspender um processo no Linux? Como eu faço para retomar a execução novamente?

R:
• Suspender: kill -STOP [pid]
• Continuar: kill -CONT [pid]

---

5 - O que aconteceria se um processo criasse recursivamente processos filhos indefinidamente? Implemente um programa em Linux que faça isso e apresente o resultado. (Sugestão: testar na máquina virtual).

R:
Criar um programa que gera processos filhos indefinidamente resultaria em uma rápida exaustão dos recursos do sistema, já que cada processo filho continuaria a gerar mais filhos independentemente. Isso consumiria gradualmente toda a memória disponível e os recursos do sistema, levando eventualmente à terminação do programa devido à falta de recursos.

Code:

```
void infinite_fork() {
    pid_t pid = fork();
    if (pid > 0) {
        printf("Processo pai: PID %d\n", pid);
        wait(NULL);
    } else if (pid == 0) {
            printf("Processo filho: PID %d\n", pid);
            infinite_fork();
```

```
        } else {
            perror("Erro ao criar processo filho");
            exit(1);
        }
}
```

---

1 - Faça um programa que crie uma hierarquia de processos com N níveis (1 + 2 + 4 + 8 + . . . + 2N-1) processos. Visualize a hierarquia usando um comando do sistema (pstree).

```
void cria_hier(int level, int max_level) {
    if (level == max_level) {
        printf("Processo final da hierarquia: PID %d\n", getpid());
        return;
    }

    printf("Processo nível %d: PID %d\n", level, getpid());

    for (int i = 0; i < 2; ++i) {
        pid_t pid = fork();
        if (pid == 0) {
            create_hierarchy(level + 1, max_level);
            break;
        } else if (pid < 0) {
            perror("Erro ao criar processo filho");
            _exit(1);
        }
        wait(NULL); // Espera o filho terminar
    }
}
```

---

2 - Faça um programa que receba um comando Linux como parâmetro e execute como um filho do processo. O processo pai deve aguardar o término da execução do comando.

```
int main(int argc, char *argv[]) {
```

```
    if (argc < 2) {
        fprintf(stderr, "Uso: %s <comando>\n", argv[0]);
        exit(1);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("Erro ao criar processo filho");
        exit(1);
    } else if (pid == 0) {
        execvp(argv[1], &argv[1]);
        exit(1);
    } else {
        wait(NULL);
    }

    return 0;
}
```

---

4 - Faça uma interface de shell simples que fornece um prompt ao usuário
para executar comandos do shell do sistema. Se o comando for executado
em segundo plano (&), a interface deve possibilitar a execução de outros comandos.
Caso contrário, a interface deve esperar o retorno do comando e, em
seguida, exibir o prompt novamente.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <wordexp.h>
#include <sys/wait.h>

#define MAX_LINE_LENGTH 100

int main(){
    char* params[MAX_LINE_LENGTH/2 + 1];
    char cmd[MAX_LINE_LENGTH];
    int saida= 0;
    int espera = 1;
    int i;
    int status;
```

```
    wordexp_t aux;

    do{
        printf("shell> ");
        fflush(stdout);
        fgets(cmd, MAX_LINE_LENGTH, stdin);

        if(strcmp(cmd, "exit") == 0){
            saida = 1;
        } else{
            if(cmd[strlen(cmd)-2] == '&'){
                cmd[strlen(cmd)-2] = '\0';
                espera = 0;
            }else{
                cmd[strlen(cmd)-1] = '\0';
            }

            wordexp(cmd, &aux, 0);
            for(i = 0; i < aux.we_wordc; i++){
                params[i] = aux.we_wordv[i];
            }
            params[i] = NULL;

            pid_t pid = fork();
            if(pid == 0){
                execvp(params[0], params);
                exit(0);
            }

            if(espera == 1){
                waitpid(pid, &status, 0);
            } else{
                espera = 1;
            }

        }

    } while(!saida);

    return 0;
}
```