EP1 - Sistemas Operacionais

Tiago Martins Nápoli - 9345384

September 10, 2018

A fim de facilitar a identificação dos comandos escolhidos pelo usuário, foi feito o seguinte vetor de strings:

```
const char *commands[4] = {
  "protegepracaramba",
  "liberageral",
  "rodeveja",
  "rode"
};
```

A shell requisita, em um laço sem fim, comandos que o usuário deseja executar (a shell só sai quando o usuário força sua interrupção). O comando digitado pelo usuário é identificado entre os comandos possíveis. Para isso foram feitas funções auxiliares `check_command`, que checa se o comando digitado pelo usuário, em `command.str`, foi determinado comando no vetor `commands`.

i) protegepracaramba: Simplesmente executa a chamada de sistema `chmod` com o argumento (path) dado pelo usuário e o valor correspondente à proteção total do arquivo. Em caso de erro imprime-o na tela.

```
if(check_command(command.str, 0)) {
    if(chmod(args.str, 0) < 0) {
        printf("%d %s\n", errno, strerror(errno));
    }
}
```

ii) liberageral: Semelhante ao anterior.

```
else if(check_command(command.str, 1)) {
    if(chmod(args.str, 511) < 0) {
        printf("%d %s\n", errno, strerror(errno));
    }
}
```

iii) rodeveja: Executa um `fork`, que duplica o processo atual, e daí é feita uma checagem se o processo é o pai ou o filho. Caso seja o filho `(pid == 0)`, é feito uma chamada de sistema ao `execve`, que substitui a execução do programa atual no processo filho pelo programa dado como argumento ao `execve`. Caso o processo seja o processo pai, há uma chamada ao `waitpid`, que esperará o processo filho criado terminar de executar (o pid devolvido pelo fork é o do processo filho).

```
else if(check_command(command.str, 2)) {
    pid = fork();
    if(pid == 0) {
        strcpy(argv[0], args.str);
        execve(args.str, argv, envp);
        printf("=> erro ao executar o programa %s\n", args.str);
        exit(127);
    } else {
```

```
            waitpid(pid, &status, 0);
            printf("=> programa %s retornou com codigo %d\n",
                    args.str, WEXITSTATUS(status));
        }
    }
```

iv) rode: Semelhante ao anterior, mas agora o processo pai não espera o processo filho terminar, somente continua sua execução normalmente.

```
    else if(check_command(command.str, 3)) {
        pid = fork();
        if(pid == 0) {
            execve(args.str, argv, envp);
            printf("=> erro ao executar o programa %s\n", args.str);
            exit(127);
        }
    }
```