Insper

17 - Chamadas de sistema: `exec`

Sistemas Hardware-Software - 2020/1

Igor Montagner

Parte 1 - argumentos de um programa

Leia com atenção o código antes de responder os próximos exercícios.

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

int main(int argc, char *argv[]) {
    for (int i = 0; i < argc; i++) {
        printf("arg: %s\n", argv[i]);
    }

    return 0;
}
```

Exercício: Compile o código acima e rode-o. Qual sua saída?

Exercício: Como visto em aula, um programa pode receber argumentos. Execute o programa acima com `exemplo-args ag1 arg2`. Qual sua saída?

Exercício: Qual o significado de `argv[0]`?

Agora vamos pensar antes de rodar o programa. **Responda abaixo antes de rodar o programa**.

Exercício: Qual seria o valor das variáveis abaixo para a invocação `exemplo-args teste var bla foo` ?

```
argc =
argv =
```

Exercício: Crie um programa `soma` que tem o seguinte comportamento:

1. Se o programa for chamado com menos de 2 argumentos mostrar mensagem de erro e sair.
2. Se o programa for chamado com mais de 2 ou mais argumentos, mostrar no terminal a soma deles.
3. Se um argumento não for um número considerá-lo como 0.

Dica: execute `man atof` ;)

Parte 2 - carregando novos programas com `exec`

A chamada `execvp` é usada para carregar programas na memória e executá-los. O novo programa é carregado no contexto do processo atual, substituindo-o por completo. Veja um exemplo de uso correto do `execvp` abaixo.

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

int main(int argc, char *argv[]) {
    char prog[] = "ls";
    // a lista de argumentos sempre começa com o nome do
    // programa e termina com NULL
    char *args[] = {"ls", "-l", "-a", NULL};

    execvp(prog, args);
    printf("Fim do exec!\n");

    return 0;
}
```

Pergunta: por que o programa acima não dá o `printf` abaixo do `execvp` terminar?

Os argumentos passados no `execvp` são passados para o `main` do programa executado via argumentos do `main`. Ao fazer a chamada

```c
char prog[] = "prog1";
char *args[] = {"prog1", "arg1", "arg2", NULL};

execvp(prog, args);
```

O main de `prog1` será chamado com `argc=3` e `argv={"prog1", "arg1", "arg2"}`. O primeiro argumento é sempre o nome do programa chamado. Como já vimos antes, os argumentos são sempre strings.

Importante: Faça os dois programas abaixo do zero. Adaptar exemplos anteriores, apesar de prático, evita que repitamos os comandos e isto atrapalha a memorização dos comandos usados.

Exercício: Crie um programa `eh_par` que recebe um inteiro como argumento de linha de comando e cujo `main` retorne 1 se o número for par, 0 caso contrário e -1 se ele for negativo.

Dica:

- pesquise para função `atol` para fazer a conversão do argumento de linha de comando para `long`.
- você pode testar seu programa no terminal: basta rodar `eh_par 10` para checar se o número 10 é par.
- para ver o valor de saída do último programa rodado execute `echo $?`

Vamos agora juntar `fork`, `wait` e `exec` em um único exercício!

Exercício: Crie um programa que recebe números via `scanf`, executa `eh_par` em um processo filho e usa seu valor de retorno para decidir se o número é par ou não. Seu programa deverá parar de receber números quando `eh_par` retornar -1.

Dica: você pode usar `sprintf` para converter o inteiro lido para string. Se não souber como usar