



Universidade Federal de São João del Rei

Departamento de Ciência da Computação

Sistemas Operacionais

Professor: Rafael Sachetto

Relatório Trabalho Prático 1

Relatório do primeiro Trabalho Prático para a disciplina de Sistemas Operacionais do Bacharelado em Ciência da Computação da Universidade Federal de São João del Rei.

Filipe Mateus

Gustavo Detomi

Marcos Martins

Maio
2022

1 Introdução

Para este Trabalho Prático foi implementado um interpretador de comandos que foi chamado *shellso*, e conta com diversas funcionalidades comuns aos interpretadores de comandos: disparar processos, encadear a comunicação entre eles usando pipes, executar processos em segundo plano, listar processos disparados pelo interpretador, retornar processos do segundo plano de volta para o interpretador, entre outras.

2 Resumo do projeto

Algumas estruturas importantes:

```
struct shell {
    bool running;
    bool verbose;
    prompt_function prompt;
    aliases* aliases;
    background_jobs* jobs;
    shell_builtin_commands* builtin_commands;
}
```

A estrutura *shell* representa a base do nosso interpretador de comandos, e deve manter registro dos jobs, dos comandos próprios e dos *aliases*.

```
struct command {
    int argc;
    char** argv;
    command* next;
    command_chain_type chain_type;
    char* stdin_file_redirection;
    char* stdout_file_redirection;
    char* stderr_file_redirection;
}
```

A estrutura *shell* representa os comandos em si, possui os campos necessários para armazenar informações do comando, do encadeamento de comandos e redirecionamento de entrada e saída.

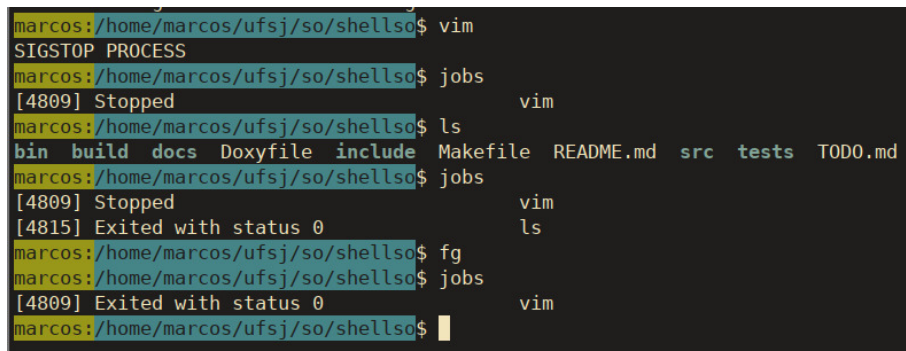
```
struct builtin_command {
    char* command;
    builtin_command_function function;
}
```

A estrutura *builtin_command* é usada para representar os comandos implementados internamente no *shellso*.

```
struct _shell_builtin_commands {
    builtin_command** commands;
    int size;
    int count;
} shell_builtin_commands;
```

É uma tabela *Hash* contendo todos os comandos que foram implementados diretamente no *shell*.

Alguns exemplos de *builtin_commands* são *jobs* e *fg*. Esses comandos são necessários para controlar a execução de processos filhos do interpretador. O comando *jobs* percorre os *background_jobs* do shell mostrando informações relevantes sobre eles, se estão executando, se foram terminados com sucesso além do *pid* de cada um. Enquanto isso o *buitin_command fg* traz de volta para o shell aqueles processos que estavam em segundo plano. Abaixo temos um exemplo do uso deles:



```
marcos:/home/marcos/ufsj/so/shellso$ vim
SIGSTOP PROCESS
marcos:/home/marcos/ufsj/so/shellso$ jobs
[4809] Stopped                  vim
marcos:/home/marcos/ufsj/so/shellso$ ls
bin  build  docs  Doxyfile  include  Makefile  README.md  src  tests  TODO.md
marcos:/home/marcos/ufsj/so/shellso$ jobs
[4809] Stopped                  vim
[4815] Exited with status 0      ls
marcos:/home/marcos/ufsj/so/shellso$ fg
marcos:/home/marcos/ufsj/so/shellso$ jobs
[4809] Exited with status 0      vim
marcos:/home/marcos/ufsj/so/shellso$
```

Nesse exemplo foi usado o *shellso* para executar o *vim*, em seguida sua execução foi pausada e isso pode ser verificado com o *jobs*, após executar um *ls* o *jobs* mostra o processo do *vim* pausado e o processo do *ls* terminado com sucesso. Após isso foi executado outro *buitin_command fg* para trazer o *vim* para o primeiro plano e terminar sua execução. Após isso o *jobs* pode ser utilizado uma última vez para verificar que o processo foi de fato terminado.

```
struct background_job {
    char* command;
    int pid;
    int status;
    background_job* next;
}
```

Essa estrutura representa os comandos que estão sendo executados em segundo plano pelo *shellso*. Um caso de uso interessante dessa estrutura foi no *handler* do sinal *SIGCHLD*. Nessa função foi preciso identificar quais processos de segundo plano foram finalizados, e quais continuam em execução, para isso é percorrida a *lista encadeada* de `textitbackground_job's` do *shell*, para cada um usamos a chamada de sistema `waitpid()` passando a flag `WNOHANG`, essa flag faz com que a chamada retorne imediatamente o estado de um determinado processo filho sem esperar até que ele seja terminado.

3 Decisões de projeto

O *shellso* foi projetado de forma a ser dividido em módulos bem separados com o objetivo de tornar seu desenvolvimento e eventual manutenção mais fácil. Cada elemento do interpretador foi implementado em arquivos separados dentro da pasta *shell*, algumas funções utilitárias para trabalhar com cadeias de caracteres foram implementadas na pasta *string*.

Essa decisão de dividir o programa em módulos primeiramente divide bem as responsabilidades, o que garante que a pessoa desenvolvedora possa trabalhar em um pedaço pequeno do problema simplificando o projeto e a implementação, além disso auxilia a evitar efeitos colaterais, já que se ao longo de todo o problema as estruturas apresentadas na seção anterior fossem visíveis e alteráveis seria bastante provável que ao longo do desenvolvimento e eventuais manutenções fossem provocados efeitos colaterais ou seja, alterações introduzidas sem cuidado causando modificações incompletas nas estruturas de dados, provocando erros difíceis de reproduzir e detectar.

Outra decisão que tomamos foi de implementar testes, para isso usamos a biblioteca Criterion, disponível em <https://criterion.readthedocs.io/en/master/intro.html>. Com essa simples biblioteca de testes foi possível escrever testes de unidades que facilitam o desenvolvimento tornando mais fácil visualizar quando um código novo resolve o problema a que se propõe sem causar efeitos colaterais em outras partes do código.

4 Bugs conhecidos ou problemas

Um erro conhecido desse trabalho é relacionado aos redirecionamentos de entrada e saída. A cada *pipe* utilizado um *file descriptor* é deixado aberto, não foi possível identificar onde seria necessário fechar esse descritor de arquivo. Abaixo temos um exemplo desse bug:

```

marcos:~/ufsjs/so/shellso$ make debug && bin/shellso
make: Nothing to be done for 'debug'.
marcos:/home/marcos/ufsjs/so/shellso$ # 4315 é o pid do shellso rodando
marcos:/home/marcos/ufsjs/so/shellso$ ls /proc/4315/fd
0 1 2
marcos:/home/marcos/ufsjs/so/shellso$ ls | grep b | grep bin
[4340] Background updated. Status: 0
bin
[4341] Background updated. Status: 0
marcos:/home/marcos/ufsjs/so/shellso$ ls /proc/4315/fd
0 1 2 3 4 5 6 7 8
marcos:/home/marcos/ufsjs/so/shellso$ ls | grep b | grep bin
[4347] Background updated. Status: 0
bin
[4348] Background updated. Status: 0
marcos:/home/marcos/ufsjs/so/shellso$ ls /proc/4315/fd
0 1 10 11 12 13 14 2 3 4 5 6 7 8 9
marcos:/home/marcos/ufsjs/so/shellso$

```

Um recurso interessante que não foi implementado foi a captura de alguns atalhos que poderiam facilitar o uso do interpretador, como o *ctrl+L* para limpar o terminal e das teclas direcionais para buscar comandos recém executados ou para movimentar o cursor do prompt para corrigir um comando sem reescrevê-lo do início.

Anexo

Repositório: <https://github.com/marcosfons/shellso>

Documentação online: <https://marcosfons.github.io/shellso/>