Elly@osh: um *Shell UNIX* com recursos básicos de histórico

Elyabe Alves, *Ciência da Computação*, *UFES*

Resumo— Este artigo apresenta a experiência realizada na atividade de laboratório para a disciplina de Sistemas Operacionais. Apresenta-se o ambiente o ambiente utilizado, o processo de criação e as funcionalidades, bem como as considerações levadas em conta na implementação de um simulador de um *shell.*

Index Terms— Sistema operacional; sheel; terminal de commando.

1 Introdução

ABSTRAÇÃO é um termo fortemente empregado na computação. O desenvolvimento de máquinas gradualmente mais rápidas fez com que fossem necessários meios de instruí-las empregando o formato de programas.

Dotado de um nível mais alto desta abstração, o *shell* é definido como uma interface de usuário normalmente modelada em linha de comando ou uma interface gráfica de usuário (GUI) cuja principal função é fornecer uma ponte aos serviços oferecidos pelo sistema operacional.

Dessa forma, a proposta deste artigo, é a criação de um programa em linguagem C que simule um interpretador de comandos de forma bem simplicista, adicionando, mais tarde, um recurso básico de histórico.

2 Ambiente emulado

2.1 Configurações de máquina

Toda o projeto foi desenvolvido em uma máquina com processador Intel Core I7 7th Gen, munida de uma GPU NVIDIA GEFORCE 940MX e 8 GB de memória RAM. Embora tenha, originalmente o sistema Windows 10, da Microsoft, como sistema padrão, o programa discutido foi codificado e executado no Ubuntu (versão 17), sistema amplamente conhecido por ser uma das mais usadas distribuições Linux. Dessa forma, toda a execução foi realizada em uma máquina virtual, com 4 GB de RAM.

2.2 Considerações de software

O *ellt@osh* foi compilado utilizando-se o compilado *GCC* via terminal, sem quaisquer parâmetros ou comandos adicionais em detrimento dos que são mostrados abaixo.

gcc elly_shell.c –o shell_elly

3 O projeto

Conforme proposto [1], o projeto consiste em criar um programa para servir como uma interface de shell que aceita comandos de usuário e, então, executa cada comando em um processo separado.

> Uma técnica para a implementação de uma interface de shell é que, primeiro, o processo-pai leia o que o usuário insere na linha de comando (nesse caso, cat prog.c) e, então, crie um processo-filho separado para executar o comando. A menos que especificado de outra forma, o processo-pai espera que o filho saia antes de continuar.
> Um processo-filho separado é criado com o uso da chamada de sistema *fork( )*, e o comando do usuário é executado com o uso de uma das chamadas de sistema da família *exec*( ) [1, p. 147].

Neste trabalho, utiliza-se a função

*int execvp(const char *file, char *const argv[])*

que executa um comando passado como parâmetro juntamente com sua lista de argumentos.

Ainda em concordância com a proposta dos autores, a função *main()* executa um loop contínuo enquanto *should_run* for igual a 1; apresenta o prompt *elly@osh*> e descreve os passos a serem executados depois que a entrada do usuário tenha sido lida; quando o usuário inserir exit no prompt, seu programa tornará *should_run* igual a 0 e terminará.

Trataremos este projeto é organizado em duas partes: (1) criação do processo-filho e execução do comando nesse processo, e (2) modificação do shell para permitir um recurso de histórico.

3.1 Criação e execução do processo-filho

Inicialmente, o programa foi concebido em 2 módulos: a

int main(void) ,

função principal, a partir da qual o *shell* será – de fato inicializado e aguardará o comando que será inserido pelo usuário conforme mostrado na Fig 1; e a função

*int commandInput(char **args) ,*

• *A. Elyabe é graduando em Ciência da Computação pela Universidade Federal do Espírito Santo, Ufes, São Mateus – ES, Brasil.*
E-mail: elyabe@outlook.com Github: https://github.com/Elyabe

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

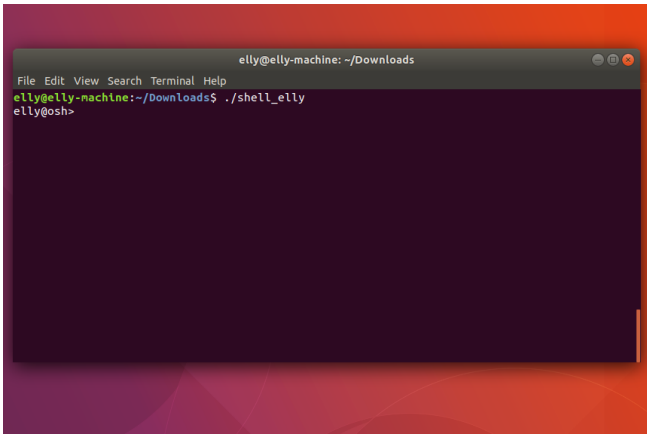

Fig. 1. O *shell* pronto para receber os comandos.

recebe o vetor de argumentos, composto comumente por um ou mais comandos, e retorna 1, caso exista comando a ser executado e 0 caso contrário. Isto é necessário para o caso no qual o usuário entre com o comando vazio. Havendo comandos, será este o módulo cuja tarefa é separar devidamente os comandos de seus argumentos para serem submetidos ao terminal em fase posterior.

3.2 Implementação do recurso de histórico

A fim de suportar a execução de um comando recentemente inserido pelo usuário, foi implementado um recurso de histórico. Uma das formas de prover tal ferramenta, é o uso de uma fila implementada em um vetor circular de tamanho fixo, na qual, não havendo mais espaço disponível, sobrescreve-se o menos recentemente inserido, conforme mostrado na Figura 7.

3.2.1 Técnicas de recuperação de um comando no histórico

Basicamente, existem três formas de obter acesso ao histórico. A primeira delas, é solicitar ao terminal, por meio do comando *history,* visualizar todos os últimos **N** comandos inseridos do mais ao menos recente associados à numeração do comando na atual execução. A Figura 2 apresenta

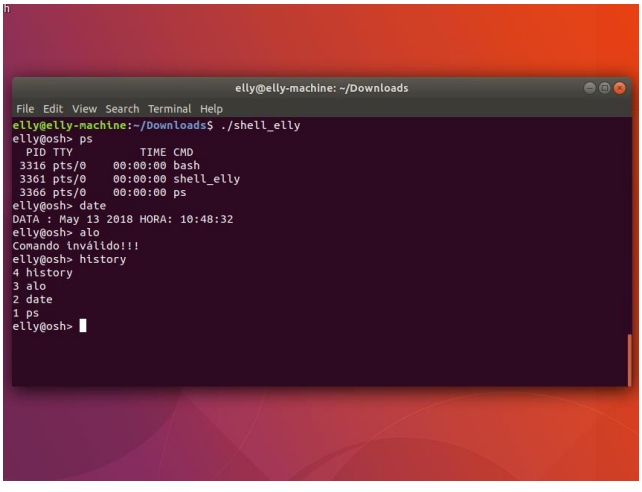

Fig. 2. Tela de comando após execução dos comandos *ps*, *date* e *history.*.

os resultados da execução do comando *history* para uma implementação com *N = 10*.

Com caráter mais prático, as duas maneiras restantes possibilitam ao usuário executar um dado comando salvo no histórico. Ao inserir *!!* no *elly@osh*, o terminal responderá executando o comando mais recente. Por outro lado, se inserido, o comando *!I* executa, se existir, o I-ésimo comando no histórico, sendo *I,* um número correspondente ao comando exibido à sua esquerda pelo comando *history*.

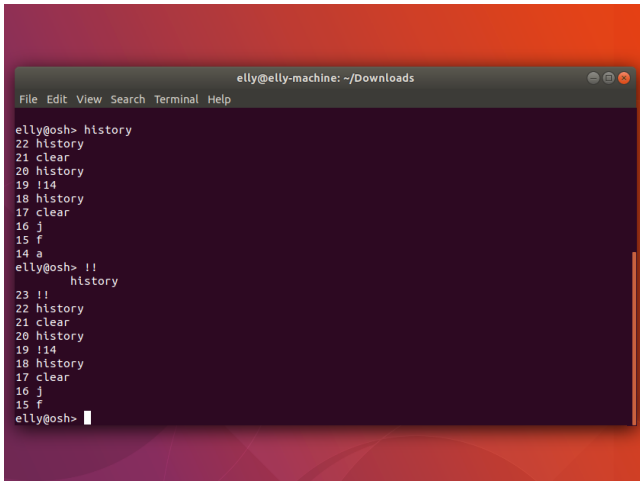

Fig. 3. Utilizando o histórico para executar o último comando.

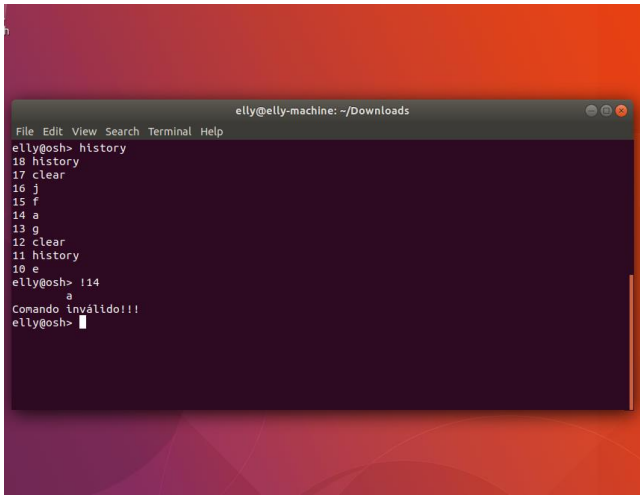

Fig. 4. Utilizando o histórico para executar o I-ésimo comando. Neste exemplo, *I = 14.*

3.2.2 Aprimoramento do recurso

Imagine que o usuário insira o comando *!!* imediatamente após a inicialização. Então, não há comandos no histórico, e portanto, não há o que executar. Outro cenário possível é o que o usuário solicita a execução do i-ésimo comando submetendo a entrada *!I* para algum *I* não armazenad no histórico, ou ainda, que último comando tenha sido *!!* e o usuário tente utilizá-lo novamente provocando assim, um ciclo infinitamente repetitivo. As figuras (Fig.) 5 e 6, exibem estes casos e o que o interpretador de comandos deve retornar ao usuário.

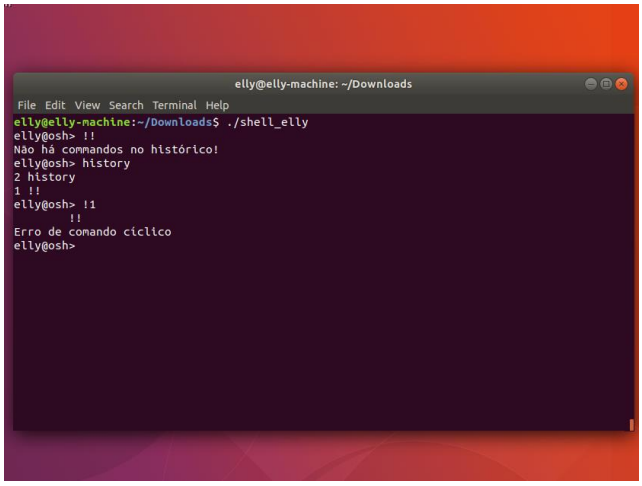

Fig. 5. Nas primeiras linhas, o usuário tenta acessar comando inexistente no histórico. Nas linhas seguintes, um erro de comando cíclico.

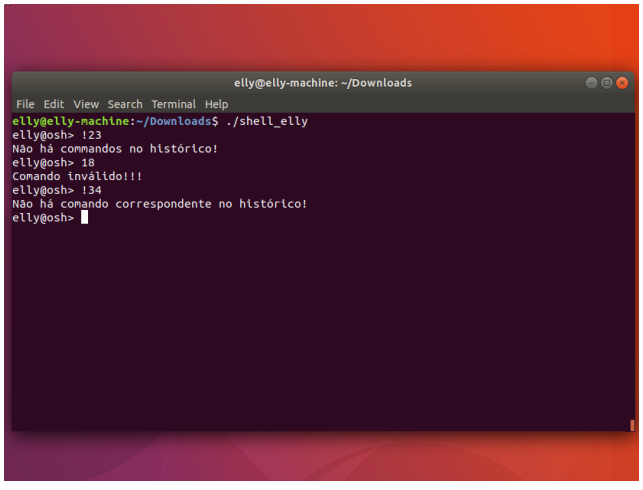

Fig. 6. Tentativa de execução de comando não armazenado ou esquecido pelo *shell.*

6 Conclusão

Embora esta seja uma primeira versão, o *shell* tem se mostrado como uma tarefa interessante. No entanto, ajustes de programação ainda devem ser feitos a fim de abranger o maior número possíveis de casos e exceções que possam a vir ocorrer. Relembramos, entretanto, que o objetivo principal do trabalho é simular, não em sua totalidade, o funcionamento de um interpretador de comandos. Desse modo, trabalhos futuros poderão incorporar novas funcionalidades que o torne ainda mais utilizável. Uma das melhorias, pode ser resultado, por exemplo, do suporte à comandos de outras famílias de processos que não sejam apenas aqueles da função *exevp*. Ou, ainda, a contrução de um programa similar para sistemas Windows.

Referências

[1] A. Silberschatz, P. B. Galvin e G. Gagne, Fundamentos de Sistemas Operacionais, LTC, 2012.

[2] L. H. G. Valim, *Mysh,* São Mateus, 2018.

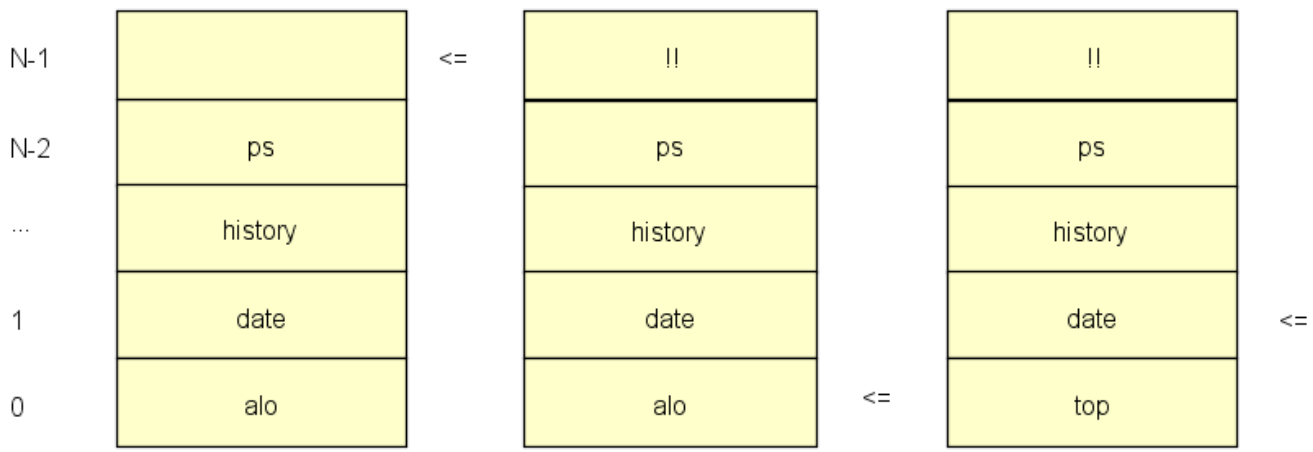

Fig. 7. Esquema do vetor circular aplicado ao recurso de histórico. Note que após o preenchimento da capacidade total do vetor, o comando *alo* é substituído por *top*, o novo comando inserido pelo usuário.