Infraestrutura de Software
Implementação 1^a Unidade

Matheus Pereira do Rego Barros[1]

[1]Ciência da Computação – CESAR School
Avenida, Cais do Apolo, 77, Recife - PE, 50030-22

mprb@cesar.school

Resumo. *O objetivo deste arquivo é analisar e explicar, de forma clara e objetiva, a implementação do Shell, referente a implementação prática da 1^a unidade. Serão abordados ponto a ponto dos requisitos, bem como uma rápida análise do que foi feito ou não. Iremos dividir esse documento em 2 partes, cada uma falando do que foi implementado em cada modo do Shell.*

1. Interativo

1.1. Múltiplos Argumentos

(Implementado) O Shell produzido consegue receber tanto comandos simples como múltiplos, quando separados por ";" . Contudo, o limite máximo de caracteres aceitos em cada input do usuário reside em 200 caracteres e, para cada comando, um limite de 100 (ambos, contando com os espaços). Ademais, após extensos testes, poucas vezes foi verificado algum erro quando foi testado na quantidade máxima de caracteres.

1.2. Execução Baseada em Estilo

(Implementado) Ambas as execuções, tanto a sequencial como a paralela, foram implementadas no Shell em questão. Em ambos os casos, o shell fica indisponível até que todos os comandos sejam executados.

1.2.1. Estilo Sequencial

(Implementado) O estilo sequencial, representado por "`mprb seq>`" antes do input do usuário, foi implementado como padrão *default* ao iniciar o shell. Sua execução é feita de modo relativamente simples ao separar a string de input, por ";", em elementos de um único array. Assim, conforme o loop acontece, cada comando (da esquerda para direita), será executado de forma sequencial só passando para o próximo quando o comando em questão termina de sua execução. Ademais, quando estamos no estilo paralelo, podemos retornar ao sequencial ou inserir o comando "style sequential"

1.2.2. Estilo Paralelo

(Implementado) Por sua vez, o estilo paralelo de execução é determinado quando o usuário insere o comando "style parallel" e, a partir dai, todo comando inserido será executado de modo paralelo (ou até que o usuário digite o comando "style sequential"). Assim, independente de múltiplos comandos serem inseridos simultaneamente, todos serão

executados em paralelo, cada um em seu respectivo processo filho. Sendo juntadas (*join*) conforme terminam o respectivo processamento, voltando para o processo pai. Ademais, ressalto que foi necessário alteração no código em virtude do impedimento do uso de threads, mas como tudo estava bem modularizado, foi tranquilo realizar tal alteração. Sua representação no terminal é "`mprb par>`".

Confesso que o código do estilo paralelo ficou melhor modularizado pois, em sua implementação, consegui organizar melhor as ideias antes mesmo de começar escrever o código. O que não aconteceu na implementação do sequencial, que fui aprendendo conforme fui fazendo.

1.3. Pipe

(Implementado) Este shell tambem dá suporte a pipes, implementados de forma similar ao sequencial. Principalmente tento em conta que a saida de um comando será a entrada de outro. Assim, no lugar de simplemente esperar o outro processo terminar sua execução e continuar normalmente, aqui, a saída do processo filho é escrita num *pipe* que é lido por outro processo como entrada para a execução de seu comando em questão. Esse redirecionamento de saída (do STDOUT para `STDOUT_FILENO`) é realizado com o uso do dup2(). De mesmo modo, o redirecionamento da entrada de STDIN para `STDIN_FILENO`.

Ademais, alem da separação dos comandos (que gera a entrada e a saída), é preciso a separação do comando e seus respectivos argumentos. Assim, na chamada da função para execução são passados os 2 arrays que correspondem aos argumentos em questão `execvpSeqPipe(argv_pipe,argv_pipe2)`(sequencial). No paralelo, por sua vez, é só chamada a função de execução do comando paralelo (`execvpPar`) e o tratamento da string é realizado dentro da função.

1.4. Redirecionamento de Entrada e Saída com Arquivos

Este Shell implementa todas as formas de redirecionamento, produzidas de forma similar entre sí. O maior ponto de divergência surge no recebimento das entradas a partir de um arquivo, pois se assimila mais ao comportamento do *batch* do que com uma nova implementação.

1.4.1. Entrada de arquivo e redirecionamento de saída

(Implementado) Representado pelo uso do símbolo ">" após o comando e seguido pelo nome do arquivo que deseja salvar sua saída. Foi implementado usando o conceito similar ao pipe. Contudo, aqui, foi usado o dup2() para redirecionar o STDOUT da execução do comando para o arquivo que será utilizado para salvar a saída (*fileRed*). Se não existir um com o nome informado, será criado um novo para isso.

1.4.2. Redirecionamento de Entrada

(Implementado) Representado pelo uso do símbolo "<" após o programa e seguido pelo nome do arquivo que deseja receber as entradas. Foi direta sua implementação, tendo em vista que quando comecei sua produção, ja tinha o batch todo pronto. Então, só

foi necessário alterar alguns elementos para tornar mais genérica toda implmentação, assim, após o devido tratamento da string, só executei normalmente chamando a função de execução padrão do comando (sequencial), ou do padrão paralelo, funcionando como se tivesse chamado em *batch*.

Sua implmementação foi interessante, pois me fez repensar melhor a forma como meu código estava organizado e corrigir algumas falhas na implementação do Batch.

1.4.3. Redirecionamento de entrada e saída

(Implementado) Muito parecido com a implementação utilizada no redirecionamento de saída do símbolo ">". Contudo, aqui, um comando recebe as entradas de um arquivo (ou não) e insere sua saída no final de outro arquivo. Representada pelo uso do símbolo ">>", foi utilizada as mesmas lógicas dos redirecionamentos de saída e de entrada, só que de forma conjunta, onde o arquivo de entrada é o input do comando e a saída deste input é anexada ao final do outro arquivo referenciado. Isso foi possivel utilizando o dup2(), que permite manipular a entrada e a saída dos arquivos, conforme os parâmetros estipulados.

1.5. Histórico

(Implementado em parte) O Histórico, representado pelo uso do simbolo "! !" foi implementado. Funciona tanto com um único como com múltiplos comandos. Sua implementação funciona perfeitamente no sequencial, contudo, buga um pouco quando há ausencia de comandos ou no paralelo. Mas, em suma, funciona corretamente.

1.6. Background

(Não implementado) Em virtude da necessidade de mudar o style parallel de threads para o uso estrito de fork(), tive que priorizar tal alteração em virtude da implementação do background.

2. Batch

(Implementado) Em Resumo, a implementação do Batch foi realizada de modo bem similar ao interativo, respeitadas algumas especialidades do próprio modo de execução a partir da leitura e tratamento de arquivos. Creio que a única limitação reside na ausencia do histórico. Válido ressaltar, ainda, que após testes, para execução perfeita do batch, é preciso que tenha o "exit" no final ou, pelo menos, uma linha vazia no final (). Deste modo, não foi verificado nenhum erro na execução do batch. Só alguns dos testes geraram erros, mas nenhum que quebrasse o código.

3. Conclusão

Foi muito interessante a produção do Shell. Esta experiência de ter que pesquisar e conseguir implementar um código deste tamanho pela primeira vez foi muito massa! Realmente, eu não conseguiria fazer nem metade se tivesse no prazo original, mas creio que este prazo de 4 semanas foi o suficiente para produzir algo legal. No final, sinto orgulhoso do que fiz. Afinal, só quem se esforça por algo, sente os louros deste.