* Linha 27 - 52 && Linha 111 - 165 → Isa
* Linha 53 - 110 → Day

Execução

* teste 1, 3, 5 → Isa: Explicar melhor as aplicações “ls”, “sort” e os parametros “<”, “>”, “-la”

ls -la

sort < teste.txt > novo\_teste.txt

sort < teste.txt | grep “x” > novo\_teste.txt

sort < teste.txt | grep “x” >> novo\_teste.txt

sort < teste.txt | grep na >> novo\_teste.txt

* teste 2, 4 → Day: Explicar melhor as aplicações “grep”, “more” e “>>”, “|”

ls -la | grep “teste”

ls -la | grep teste

ls -la | grep teste | more

more teste.txt

sort < teste.txt >> novo\_teste.txt

Explicação do video

**Linhas 27 - 52**

INTRO

Olá, este é o vídeo de apresentação do trabalho 1 da disciplina de Sistemas Operacionais, feito por mim, Isadora Muniz e pela aluna Daiana Santos.

A proposta do trabalho é de implementar um interpretador de comandos em terminal para Linux.

Foi desenvolvido um programa em linguagem c, em que serão explicados os passos para a sua implementação.

Nosso programa é dividido basicamente em duas partes, do qual a primeira é o armazenamento do comando descrito no terminal pelo usuário, e a segunda o processamento e execução do comando inserido.

Das linhas 27 à 31 foram declaradas as variáveis a serem usadas ao longo do programa.

Cada variável será explicada ao longo do seu uso.

Na linha 33 é realizada a leitura da linha de comando a ser executada, e na linha 34 é atribuída a variável token, declarada como um vetor de char, os tokens da linha de comando, no caso apenas os comando, desconsiderando espaço.

Entre as linhas 36 a 39 é criado o pipe, que é uma função que cria um canal de dados unidirecional que pode ser usado para comunicação entre processos, da qual está disponível através da biblioteca unistd.h

Na linha 41 é feita uma comparação checando a existência de um token.Em caso positivo é dado seguimento pras linhas 42 a 65, caso contrário, é detectada a não existência do comando.

Quando é encontrado um caso que tenha um token, é iniciado um laço, que dura enquanto o token é diferente de null, ou seja, enquanto há token a ser lido.Na linha sequente é criada a matriz de tipo char em que armazenará os tokens encontrados.

Antes do seu armazenamento é realizada uma verificação quanto ao token ser ou não um símbolo de pipe, que no caso é essa barra vertical. Para isso, foi criada uma variável de tipo int, que serve como uma flag, para informar ao programa se foi detectado ou não um pipe.

A variável verificaPipe é inicializada em 0, e ao programa executar a linha 46 e considerar a verificação como verdadeira, é armazenado o token na matriz de comando ( a matriz\_cmd), assim como seu índice no vetor do tipo int de índices( o Indices\_cmd). Após isso é trocado o valor de verificaPipe para 1.

Linhas 53 -110

ola, meu nome é Daiana, continuando o que a isadora explicou... no caso do else, o token a ser avaliado é um parametro ou o operador pipe, então se for o pipe preenche a matriz com NULL na posição cnt e VerificaPipe para 0, já que após o pipe, em seguida é sempre uma aplicação.

Depois de tratar o token, na linha 62 utiliza a função strtok para pegar o próximo token após o espaço e adicionar mais um ao contador de posição da matriz.

Na linha 66 só entra no else caso não tenha identificado nenhum token.

Na linha 74 começa a execução do comando a partir da matriz feita anteriormente, começando por um laço que vai de 0 até o numero maximo de aplicações encontradas no comando. Na linha 78 fork cria um processo filho, retornando 0 se tiver executando o processo filho para poder na linha 82 executar o laço while para verificar quais são os parâmetros da aplicação, utilizando a função strcmp() para comparar. No primeiro if, compara se é o parâmetro "<", qual abre o arquivo de entrada apenas para leitura (O\_RONDLY - read only) com o nome armazenado na matriz, na posição n+1, o segundo parâmetro ">" que cria um arquivo de saída (O\_CREAT), que pode ser feito a leitura ou escrita nele (O\_RDWR) e truncar o conteúdo no arquivo (O\_TRUNC), ou seja, se tiver algo escrito no arquivo ele vai sobrescrever o que está lá, já o parâmetro ">>" também é um arquivo de saída, mas diferente do ">" ele não sobrescreve o arquivo, ele acrescenta o novo conteúdo no final do arquivo. no fim do laço, na linha 110 é acrescentado mais um no n para caso tenha a aplicação tenha mais algum parâmetro.

**Linhas 111 - 165**

Seguindo as comparações feitas no programa, linha 112 é seguida de uma condição que após identificar os símbolos de entrada e saída de arquivo, seguida de um pipe, fecha o pipe na posição i corrente no laço. A mesma verificação ocorre quando i é a posição anterior ao último comando.

Após as verificações feitas, na linha 125 é executada a função execvp(), que executa o comando contido na Matriz\_cmd[] na posição do índice armazenado no vetor Indices\_cmd na posição de i. As linhas 126 e 127 fecham os arquivo de entrada e saída abertos anteriormente.

Da linha 130 a 138 é feita é caso a verificação do pid filho seja falsa, ou seja, é executado o processo pai, em que é fechado o pipe e executada a função waitpid, disponível através das bibliotecas sys/types.h e sys/wait.h, que suspende a execução do processo de chamada até que um filho especificado pelo argumento *pid* mude de estado.

As linhas subsequentes, das linhas 141 a 165 é realizada a impressão dos comandos armazenados na matriz de comandos, em que também são impressas mensagens no decorrer da sua execução.