Aluno: Matheus Felipe Alves Durães
Curso: Ciência da computação
Disciplina: Sistemas operacionais
Tema: Relatório do desenvolvimento do trabalho opcional

O trabalho consiste em tentar recriar quatro funções nativas do sistema linux, sendo estas: cat, grep, zip e unzip. No trabalho foram criadas como: wcat, wgrep, zip e wunzip respectivamente as funções originais.

Agora uma breve descrição sobre o que cada uma faz:

- wcat: Recebe como parâmetro de entrada nenhum ou mais arquivos, imprimindo seu conteúdo no terminal.

  Ao receber um arquivo esta função imprime o seu conteúdo no terminal e encerra, caso haja mais de um arquivo wcat irá ler todos os arquivos passados por parâmetro antes de encerrar, porém caso algum dos arquivos não consiga ser aberto, irá ser impresso qual arquivo não pode ser aberto e o programa encerrará com status de erro.

  Por fim caso não receba nenhum parâmetro o programa irá ler os caracteres digitados no teclado e os escreverá no terminal após o pressionamento da tecla “Enter”, neste modo de operação o programa só encerra quando enviamos uma linha vazia.

- wgrep: Recebe como parâmetro de entrada um termo de busca obrigatório e nenhum ou mais arquivos, imprimindo a linha de texto no terminal caso essa possua uma sub-palavra que corresponda ao termo de busca.

  Caso wgrep não receba um termo de busca o programa imprimirá uma mensagem de erro, dizendo o que ele espera que seja passado como parâmetro e saíra com status de erro.

  No entanto se ele receber um termo de busca, porém nenhum arquivo, assim como wcat ele também irá ler os caracteres do teclado e irá imprimi-los no terminal se houver dentro do texto digitado uma ou mais sub palavras que correspondam ao termo de busca, inclusive diferenciando letras maiúsculas e minúsculas, neste modo de operação o programa só encerra quando enviamos uma linha vazia.

  Por fim se wgrep receber o termo de busca e um ou mais arquivos, ele irá abrir os arquivos e imprimir as linhas que contêm uma sub-palavra correspondente ao termo de busca, caso o programa não consiga abrir um arquivo, ele informará qual arquivo não pôde ser aberto e encerrará com status de erro.

- wzip: Recebe como parâmetro de entrada um ou mais arquivos para compactação em um arquivo padrão chamado de “file.z” (o mesmo nome utilizado no PDF de instruções do trabalho), vale ressaltar que tudo será

compactado em uma linha, ou seja caso o arquivo a ser compactado tenha separações de linha, ele as perderá na compactação.

Ao receber os arquivos ele irá abrir um por vez, e compactá-los usando o padrão run-length encoding (RLE), conforme a compactação ocorrer será impresso no terminal quais informações exatamente ele está escrevendo no arquivo binário, porém para melhor entendimento no terminal não será impresso em binário, caso o programa não consiga abrir um arquivo, ele informará qual arquivo não pôde ser aberto e encerrará com status de erro.

- wunzip: Recebe como parâmetro de entrada um ou mais arquivos compactados e os descompacta para um arquivo padrão chamado “Unzip.txt”, além de imprimir as informações no terminal.

  Ao receber os arquivos em binário ele os lerá um por vez, descompactando-os, escrevendo o seu conteúdo tanto no terminal quanto no arquivo “Unzip.txt”, vale ressaltar que os arquivos compactados através do wzip são juntos todos em uma linha, portanto na descompactação eles também sairão com todas as informações em uma linha somente.

  Caso o programa não consiga abrir um arquivo, ele informará qual arquivo não pôde ser aberto e encerrará com status de erro.

Certo após descrevermos cada um dos programas recriados, vamos falar a respeito das decisões de implementação, frustrações e bugs.

Observação: Enquanto escrevia este PDF, por algum motivo que eu desconheço o bug simplesmente deixou de ocorrer, estava tentando tirar uma print dele quando percebi que ele simplesmente não está mais ocorrendo, porém como forma de documentação deixarei aqui o relato de como ele era.

- Bug: Vou começar falando sobre ele, pois este bug sozinho impactou o trabalho como um todo, já que não achei métodos bons o bastante para contorná-lo ou resolvê-lo.

  Este bug é basicamente iniciado quando funções como readline() e fgets() encontram um “%s” , eu realmente não consegui entender exatamente o por quê disso ocorrer, mas a partir do momento que eles encontram esse termo, o programa reage de duas formas:

  1. Ele dá falha de segmentação e encerra
  2. Caso haja um printf(“\n”) separando o printf(buffer) o programa continua funcionando, porém não como o esperado, ele basicamente escreve no terminal até aparecer o “%s” depois para e em seguida retoma reescrevendo a linha por inteiro, dessa vez sem nenhum problema.

- wcat: Bom por conta do bug citado acima tive que refazer a função de leitura do wcat, substituindo ela de fgets() para fgetc() que ao invés de pegar blocos de texto agora pega char a char, felizmente nos meus testes não impactou no desempenho, mas creio eu que esta seja uma forma muito mais lenta, fora essa mudança não houve nenhuma outra alteração ou frustração com o wcat.

- wgrep: Esse aqui eu optei por implementar de um jeito mais confiável do que a minha ideia original que era usar o conhecimento de estrutura de dados para fazer uma busca de sub-palavra, porém como seria mais demorado sua implementação e essa semana não tive muito tempo, optei por pesquisar funções em bibliotecas que fizessem esse trabalho por mim, em minhas buscas achei a biblioteca <string.h> que continha a função strstr(), para a minha sorte faz exatamente a busca de uma subpalavra dentro de outra string.
  No wgrep foi onde identifiquei o bug pela primeira vez, e onde tentei consertá-lo sem sucesso, até começar a escrever esse trabalho.

- wzip: Com certeza um dos que achei mais divertido e curioso de programar, já havíamos feito um trabalho que era um programa compactador e descompactador, porém utilizando árvore Huffman que por sinal deu muito trabalho, no entanto o wzip foi o mais tranquilo de todos para eu programar.
  A parte curiosa veio quando descobri quando fazemos o malloc, se colocarmos por exemplo uma variável inteira no sizeof() ele alocará um espaço maior do que se colocassemos sizeof(int), para mim até então as duas abordagens resultavam na mesma saída, porém a partir de testes para saber se o wzip funcionava corretamente percebi que existe diferença dependendo se colocamos dentro do sizeof() um tipo de variável ou uma variável.
  Por sorte não ocorreu nenhum bug, fora o que descobri de curiosidade no wzip.

- wunzip: Praticamente uma extensão do anterior, sem muitos problemas para programar, sem nenhum bug para tratar, após completar o wzip quase natural a programação do wunzip.

Testes: Coloquei alguns arquivos de teste junto aos arquivos .c , para a execução dos testes basta executar o makefile na função de “executarLinux” ou “executarWindows”, quando um desses for chamado ele automaticamente irá compilar os programas com as instruções do gcc já especificadas no trabalho. Consegui testar os programas em: Windows, Linux e no replit.io (que possui um terminal shell e um terminal bash, por sinal foi o mais trabalhoso de retirar os errors)

Instruções do Makefile: (Caso necessário adicionar “./” antes do “make”)

- Para compilação: make gcc

  A execução dos testes também irá compilar o código

- Para execução dos testes em Windows: make executarWindows
- Para execução dos testes em Linux: make executarLinux

Considerações finais:

Gostei muito de fazer esse trabalho, até tentei usar o Ecma como sugerido, mas após começar a ler sua documentação preferi voltar ao habitual VsCode, porém além de aprender mais sobre o sizeof() e uma nova forma de compactação, utilizei muito dos laboratórios da FACOM para desenvolver esse trabalho, o que julgo ser uma experiência única, já que nunca havia utilizado eles fora do horário das aulas, estar com outros colegas desenvolvendo, conversando, discutindo soluções e falando a respeito de pseudocódigos foi muito divertido e proveitoso, se trabalhar em um time de desenvolvimento for meramente parecido com isso acho que estou feliz com isso.

Infelizmente não consegui achar nem na documentação oficial nem por meio de teste como eu faria para encerrar o wcat e o wgrep quando usamos a entrada padrão, ou seja o teclado, a não ser pelo uso de Ctrl + C, na documentação não fala nada a respeito, e mesmo em fóruns como Stack Overflow só achei a exata mesma resposta: “Fecha o terminal e abre de novo ou dá Ctrl + C para encerrar o programa”, com isso já que nos requisitos do trabalho não é especificado como faríamos para encerrar o programa nessa situação, fiz com que quando eles recebam uma linha vazia, ou seja apenas o “Enter” e nada mais escrito, eles encerram o while em que estão presos e por fim o programa finaliza.

Em relação ao wzip e wunzip, apesar de ter sido mais fácil desenvolve-los tenho uma crítica ao PDF que os descrevem, em relação ao nível de detalhes do wcat e wgrep, o wzip e wunzip estavam com poucos detalhes e ainda abertos a interpretação, cheguei a discutir com uma colega se no trabalho era necessário ou o uso de “>” como parametro, pois o texto do PDF deixava ambíguo, pois não apresentava outra exemplo ou forma de implementação, portanto eu assumi um padrão próprio com o que entendi e creio eu que o descrevi ao falar sobre o wzip e wunzip.

Por fim ressalto o quanto este trabalho foi proveitoso, por nos forçar voltar a programar de verdade, pelas curiosidades aprendidas e finalmente aprender como produzir um makefile, além é claro do tempo no laboratório da faculdade que nunca tinha utilizado fora de aula.

Muito obrigado professor e até a próxima aula. ^u^