PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

INF 1316
Relatório de Atividade - Lab 2

2210872 Felipe Antelo Machado de Oliveira
2112171 Pedro Henrique de Oliveira Valentim Gomes

Objetivo:

O objetivo do trabalho se trata em desenvolver um programa com (algumas das) funcionalidades bases para o sistema operacional Unix. Especificamente, o grupo deve buscar, através do uso de processos com fork( ) e exec( ), implementar novos comandos “echo” e “cat”, com os respectivos nomes de programas “meuecho” e “meucat”. Além disso, uma shell para simulação deve ser feita, capaz de realizar comandos básicos nativos do sistema, tais como o ls, Ao todo, o que o exercício busca é testar o conhecimento da nossa dupla sobre como funciona a leitura e impressão de sistemas operacionais e suas shells, mostrando como resultado os 3 programas base que realizam as suas respectivas operações desejadas.

Estrutura do Programa:

Dividido em 3 códigos principais (meuecho.c, meucat.c e meushell.c), o programa busca utilizar de métodos ensinados em aula para, através de loops, processos de leitura e prints, realizar funções já existentes no sistema operacional do Unix. Abaixo, podemos ver com mais detalhes como cada um dos 3 códigos montados pela nossa funcionam.

- **meuecho.c:** Busca simular o “echo” do unix, isso é, busca realizar o ato de imprimir argumentos passados no terminal. Isso é feito através do recebimento de argumentos da linha de comando, imprimindo-os, separando-os por “espaços” e introduzindo novas linhas (\n).
- **meucat.c:** Busca simular o “cat” do unix, isso é, leitura e exibição de conteúdo presente em um arquivo. O processo é realizado atráves da verificação de nome do arquivo que, se existir, é seguido de aberturas com “fopen” que, caso a abertura seja um sucesso, é seguida por “fgetc” armazenando carcater por caracter e, finalmente, com putchar, imprindo o todo. Caso o “fopen” seja um fracasso, uma mensagem de erro é exibida (nesse caso, escolhemos “Erro ao abrir o arquivo”).
- **meushell.c:** Cria uma shell simples que executa comandos básicos como o “meuecho”, o “meucat” e “ls”. Esse processo é feito através da execução de processos como fork( ) e execvp( ) sempre que há a interpretação de algum

argumento e programa.

Solução:

Novamente, na solução precisaremos analisar cuidadosamente cada um dos 3 módulos. Sendo assim, temos para:

- **meuecho.c:** O echo exibe os argumentos com base em uma verificação de número de argumentos que, percorre cada argumento (“argv[i]”) e o imprime na tela, adicionando um espaço entre os argumentos (a não ser o último) e adicionando uma quebra de linha ao fim.
- **meucat.c:** O cat utiliza uma sequência de “fopen”s com “fgetc”s combinados de “putchar” para exibir todo o conteúdo dos arquivos até o seu final. Ele possui incluso um tratamento simples de erro ao abrir arquivos.
- **meushell.c:** O shell, sendo discutivelmente a parte mais confusa da tarefa, trabalha definindo tamanhos adequados para MAX_INPUT e MAX_ARGS, buscando fazer sequências de “pass” no input dado, percorrendo a string, identificando os argumentos como palavras-chave com base nas interrupções entre uma palavra e outra (representado por ”\0” ou “\n”) e os copiando para um array de args[ ] e finalmente marcando o fim desse array como NULL para identificação na hora do uso de execvp. E, então, ela executa o comando em questão, criando processos filho e tentando executar o comando em questão com o execvp que recebe o array de args e o argumento em questão. Naturalmente, após o filho terminar seu processo, chega a vez do pai. A shell chega ao fim quando recebe um comando de exit( ) padrão e realiza loops até esse momento. Na prática, portanto, o usuário vê um prompt padrão de shell (utilizamos “meushell>”), escreve uma entrada que é lida pelo código em argumentos, divide esses argumentos em pedaços menores para interpretar (e separa com espaços e uma quebra de linha), cria os processos filho necessários (utilizando o fork), executa o comando no filho (através do execvp), após a conclusão do processo filho retorna ao processo pai e finalmente aguarda a próxima ordem do usuário.

Observação e conclusões:

Ao todo, a tarefa foi um sucesso, uma vez que a dupla se mostrou capaz de criar os 3 códigos diferentes. Algumas dificuldades foram encontradas no caminho em questão da utilização adequada dos comandos e funções apresentadas recentemente, além de problemas nos prints dos argumentos no terminal. Entretanto, ao pouco o grupo foi corrigindo essas dificuldades, chegando ao resultado final que não possui problemas aparentes.

O exercício busca testar os conhecimentos da dupla sobre os comandos centrais de sistemas operacionais Unix, tais como fork( ), exec( ) e wait( ). A implementação da shell também indica com os comandos são interpretados e executados a partir de uma interface padrão. Certos conhecimentos básicos sobre como tratar elementos de argumentos também foram necessários para esse trabalho, assim como a utilização do tipo característico pid_t (o qual poderia ser utilizado no último trabalho, mas não era necessário). Portanto, é mais um exercício

que testa conhecimentos básicos de interpretação, execução e exibição de comandos para um sistema operacional Unix.