Universidade Federal de Uberlândia
Disciplina: Sistemas Operacionais (SO)
Atividade 1: Ambiente Linux

Sistema Operacional LINUX

Essa atividade está dividida em duas partes, sendo que na Parte 1 os alunos devem explorar o ambiente do Linux. São apresentados comandos para que os estudantes possam se familiarizar com o interpretador de comandos e ao mesmo tempo observar sobre açoões e eventos para gerenciamento do sistema operacional (um gabarito de resposta será disponibilizado). Na Parte 2, será aplicado um questionário com questões objetivas sobre os tópicos abordados em nossa disciplina.

Parte 1: Ambiente Linux

Ao carregar o S.O. LINUX procure o terminal (use no teclado a opção **Ctrl+Alt+T**) que permitirá usar um terminal de comandos.

1. Terminal

1.1- Solicite a informação sobre quem está trabalhando na máquina (o Linux é um sistema multi usuário e multi tarefa): execute o comando **who**. Use o comando **man who** para verificar informações sobre esse comando.

- Esse é um manual on-line, que fornece uma explicação detalhada sobre cada comando. Se quiser sair do manual do comando digite a tecla **q .**

- Que informações são apresentadas com esse comando ao usuário?

1.2- Para listar os arquivos do diretório corrente deve executar o comando "**ls**". Se quiser ter uma listagem detalhada, execute "**ls -l**". Para listar um diretório qualquer, execute "**ls -l *diretório***" (use o "/" para separar sub-diretórios em um path, como em "/home/computacao").

- Execute o comando **ls -l** no diretório "/**home/<usuário>"**. Nesse comando, o parâmetro <usuário> deve ser inserido o nome do usuário que você acessou o Linux.
- Quais informações foram apresentadas com esse comando? Vamos usar o **man ls** para conhecer mais sobre esse comando?

1.3- O comando "**df -h"** permite mostrar as informações sobre os discos do sistema computacional. Execute esse comando e observe o resultado. Com uso do **man df** podemos definir as flags permitidas ao comando. O que representa o parâmetro **-h.**

1.4- Para mostrar o conteúdo de um arquivo, execute o comando "**cat <*nome_do_arquivo*>**".

a) Use o comando **cat** para mostrar o conteúdo de um arquivo do padrão ASCII. Use o programa **Text Editor** para construir um arquivo com uma mensagem e com o comando **cat** visualize essas informações?

b) Use o **cat** para mostrar o conteúdo do arquivo **cpuinfo** localizado no diretório /**proc**. O diretório **proc** contém as informações dos processos na memória RAM do S.O. Qual o modelo, nome, o velocidade da cpu em Mhz e o tamanho da cache?

c) Use o **cat** para mostrar o conteúdo do arquivo **filesystems** localizado no diretório /**proc**. Apresente três sistemas de arquivo disponíveis nessa distribuição?

1.5- Para saber em que diretório você está na árvore de diretório, execute o comando "**pwd**". Para mudar de diretório, execute "**cd <nome do *diretório*>**". Que comando devo usar para mudar o diretório em um nível?

1.6- Arquivos podem ser manipulados por uma série de comandos:

a) O comando **cp** permite copiar um arquivo e o comando **mv** permite mover ou renomear arquivos. Descreva um exemplo para utilização desses comandos numa operação de cópia e movimento de um arquivo? Use o manual on-line (**man**) para facilitar essa tarefa.

b) O comando **chmod** permite mudar atributos de arquivos (leitura, escrita ou execução). Use esse comando para alterar a permissão de um arquivo. Descreva um exemplo de uso desse comando?

1.7- Diretórios podem ser criados com o comando "**mkdir**", e removidos com "**rmdir**". Descreva os comandos utilizados para criar um diretório **SO** dentro de seu usuário em /**home/<usuário>.** Lembre-se que o comando **<usuário>** deve ser inserido o nome do usuário que você acessou o Linux ?

1.8- O shell permite que se redirecione a saída de um comando para um arquivo. Assim, o resultado da execução será gravada num arquivo, ao invés de ser mostrada na tela. A sintaxe é "**comando > <*arquivo*>**" (ex: "**ls *-l* > *ls.txt***").

1.9- Execute o comando **find** para ver os arquivos dentro de um diretório. Mostre um exemplo de uso desse comando no prompt de comandos.

1.10- Para verificar a versão do kernel, execute o comando: **"uname -a"**.

2. Processos em Linux

2.1- Use o comando **top** para mostrar os processos ativos em sua máquina. Lembre-se de usar o **man top** para consultar detalhes sobre esse comando. Qual a função desse comando?

2.2- Outro comando usado para mostrar os processos é o **ps.** Se você executar o comando **ps** sem nenhum argumento, ele exibirá os processos no shell. Use o comando **`ps -e --forest.`** Como são apresentados os processos com uso dessas flags?

2.3- O comando **bg** permite que um programa rodando em primeiro plano ou parado, rode em segundo plano (background), liberando o shell para outras atividades. Exitem 3 diferentes formas: Ctrl+Z, o comando do programa seguido do carácter & ou utilizando o comando **bg**. Como posso colocar um processo em segundo plano primeiro digite no terminal **`sleep 100`** e use o comando **Ctrl+Z** ? Crie um com sleep de 150 e 300. Use também o comando top para observar se esses programas estão presentes na fila.

2.4- Para verificar se o processo está em segundo plano digite: "**jobs -l**". O comando **fg** permite que um programa rodando ou parado em segundo plano rode em primeiro plano. Nesse caso deve usar **fg %<número>** para modficar um processo para primeiro plano. Utilize novamente o comando "**jobs -l**" para verificar se a tarefa foi realizada com sucesso.

3. Makefile para linguagem C

O ***makefile*** é um arquivo para configuração de compilação utilizado pelo programa **Make**, cuja ideia é simplificar e agilizar a compilação de programas. Esse arquivo, normalmente, deve estar localizado no mesmo diretório em que estão os arquivos do projeto. Um arquivo **makefile** consiste de um conjunto de **dependências** e **regras**.

Uma dependência tem um alvo, ou seja, um arquivo a ser criado, e um conjunto de arquivos fontes, os quais são os dependentes. As regras descrevem como criar o arquivo alvo e suas dependências. Basicamente, o alvo é um simples arquivo executável.

Antes de executar os comandos de uma regra, o programa **make** certifica de que todas as dependências foram satisfeitas. Uma dependência pode ser outra regra ou algum arquivo necessário para execução dos comandos.

Após as dependências, tem-se a lista de comandos. Os comandos são edentados com o **TABs** e não com espaços. Cada comando pode ser qualquer chamada de um programa de linha de comando.

A regra pode ter qualquer nome, ou seja, o nome da regra que compila o arquivo "programa.c" pode ser "programa.exe". É ideal usar o nome do arquivo quando existe um arquivo de saída dos comandos da regra, pois é através do nome da regra que o **makefile** sabe se precisa recompilar o arquivo.

Acesse os arquivos no **Moodle** (exemplo) **hellomake.c**, **hellofunc.c** e **hellomake.h** e observe que há uma relação entre os arquivos.

Para compilar esses arquivos **sem** o **makefile** use o comando:

```
$ gcc -o hello hellomake.c hellofunc.c -I.
```

O **hello** irá criar o arquivo executável do projeto. O comando "**-I.**" é incluido para que o **gcc** análise o diretório atual (.) para inclusão do arquivo com a biblioteca **hellomake.h**.

Uma sentença do **makefile** pode ser comentada usando o caractere "**#**". Todo o texto do comentado será ignorado na execução do make: **# Makefile.**

Crie um simples arquivo denominado **makefile1** e insirá o seguinte código:

```
hellomake: hellomake.c hellofunc.c hellomake.h
     gcc -o hellomake hellomake.c hellofunc.c -I.
```

Então, para compilar deve usar o comando "**make -f makefile.1".** Após a compilação para execução basta digitar .**/hellomake**. Quando a lista de arquivos aparece na primeira linha depois do **":",** o programa **Make** sabe que a regra **hellomake** precisa ser executada se algum desses arquivos forem modificados.

Para ser um pouco mais eficiente, tente o seguinte:

```
# o compilador em uso
```

```
# o comando CFLAGS é usado como opções passadas ao compilador.
CC=gcc
CFLAGS=-I.

hellomake: hellomake.o hellofunc.o
     $(CC) -o hellomake hellomake.o hellofunc.o $(CFLAGS)
```

Nesse exemplo, há as constantes **CC** e **CFLAGS** que se comunicam para compilar os arquivos hellomake.c e hellofunc.c. Em particular, a macro **CC** é para o compilador C e **CFLAGS** é a lista de sinais para o comando de compilação. Ao colocar os arquivos de objeto **- hellomake.o** e **hellofunc.o** - na lista de dependências e na regra, **Make** sabe que deve primeiro compilar as versões .c, individualmente, e, em seguida, construir o executável **hellomake**.

Se houver necessidade de fazer uma alteração no **hellomake.h**, por exemplo, o **Make** não recompilaria os arquivos "***.c**". Então, o arquivo makefile deve ser descrito da seguinte forma:

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

#
%.o: %.c $(DEPS)
     $(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
     $(CC) -o hellomake hellomake.o hellofunc.o $(CFLAGS)
```

O uso da macro **DEPS** informa que na compilação o **arquivo .h** é necessário para compilação dos arquivos "**.c**". Também há uma regra que se aplica a todos os arquivos que terminam no sufixo "**.o"**. A regra informa que o **arquivo ".o"** **depende** da **versão do arquivo .c** e o **arquivo .h** incluído na macro **DEPS**.

Ainda é informado que para **gerar o arquivo ".o"**, deve-se compilar o arquivo "**.c**" usando o compilador definido na macro **CC**. A **opção -c** representa gerar o arquivo de objeto, os comandos "**-o $ @"** representam colocar a saída da compilação no arquivo nomeado no lado esquerdo do "**:"** ("hellomake"), os símbolos **"$ <"** é o primeiro item na lista de dependências (**DEPS**) e o **CFLAGS** é o macro definido com as opções das **FLAGS**.

3.1- Utilizando esses modelos descritos construa os arquivos **makefile1**, **makefile2** e **makefile3**. Após o uso do makefile para compilação execute o código do programa.

3.2- Acesse a pasta denominada "**exercício 1" no moodle** e faça as atualizações necessárias para que o arquivo **makefile1** funcione com os arquivos **pilha.c** e **loop-malloc.c** disponíveis no arquivo.