

SUMÁRIO

[1. INTRODUÇÃO 3](#_Toc17400079)

[2. OBJETIVOS 3](#_Toc17400080)

[3. PROCEDIMENTO UTILIZADO 3](#_Toc17400081)

[3.1. Resumo 3](#_Toc17400082)

[3.2. Especificação do sistema utilizado 4](#_Toc17400083)

[3.3. Preparação do ambiente para configuração do kernel 4](#_Toc17400084)

[3.4. Configuração do kernel 5](#_Toc17400085)

[3.5. Compilação e instalação do Kernel 6](#_Toc17400086)

[4. RESULTADOS 8](#_Toc17400087)

[5. ANALISE DOS RESULTADOS 9](#_Toc17400088)

[6. CONCLUSÕES 10](#_Toc17400089)

1. INTRODUÇÃO

Não podemos negar que se há uma diversidade gigantesca de hardwares disponíveis no mercado, e as combinações que eles podem formar para criar um computador como um todo são maiores ainda. Agora imagine desenvolver um sistema para cada combinação possível de computador que existe ou já existiu. Isso seria uma tarefa inviável! Devido a isso hoje em dia os sistemas operacionais que utilizamos vem cheios programas que muitas vezes não utilizamos, no entanto, outros computadores podem fazer grande proveito deles, e os que utilizamos podem ser inúteis para eles. Mas afinal, o que são esses programas? Eles em geral são drivers e módulos de diversos tipos que fazem a comunicação com o hardware ou permitem que outros drivers e módulos funcionem (apenas módulos fazem isso). No entanto, apesar de ser altamente benéfico para a produção e distribuição ter todos esses programas embutidos em nossos sistemas operacionais, pagamos um preço por isso, a perda de desempenho em áreas como carregamentos, execuções e até mesmo na compilação do Kernel que poderiam ser mais otimizadas, pois há diversos programas sendo executados, carregados e compilados que, muitas vezes, não precisamos.

1. OBJETIVOS

Neste projeto temos como objetivo além de nos aprofundarmos mais em aprendermos comandos de administração do Linux, termos mais enfoque em aprender a configurar, compilar, instalar e testar um Kernel Linux compilado por nós mesmos, e até mesmo presenciar erros de Kernel. Além disso e não menos importante, temos o objetivo de desativar todos os sistemas desnecessários que vem por padrão, gerando assim um Kernel mais puro e otimizado para uma máquina específica. Com isso, visamos ter um sistema operacional mais eficiente para nossos computadores, diminuindo tempo de boot e compilação.

1. PROCEDIMENTO UTILIZADO
   1. Resumo

Para a realização do experimento foram utilizadas máquinas virtuais, este é o método mais seguro para a realização de testes, pelo kernel não ter restrição de execucao, qualquer que ocorre, o sistema interrompe totalmente a execução o que pode fazer com que alguns arquivos sejam corrompidos, quando executado em um ambiente controlado, esses problemas podem ser melhor controlados e previnir a inutilização da máquina que precisaria ser reformatada com uma imagem nova do sistema, se fosse executada de forma fisica.Por conta disso perdemos um pouco do desempenho, o que aumenta o tempo de compilação, pelo processador não se dedicar totalmente a máquina dividindo seu tempo para processar as requisições da máquina hospedeira e virtual.

* 1. Especificação do sistema utilizado

Para execução deste projeto foi um utilizado um computador com as seguintes especificações:

Processador: Intel I7 4790k 4.00GHz 8 CPUs

Memória RAM: 8Gb

Placa de vídeo: GTX 970

Sistema operacional: Windows 10

Foi o utilizado o software Virtual Box para virtualização do sistema operação Ubuntu 18.04, o qual foi utilizado durante os testes de compilação do Kernel. A configuração da máquina virtual usada foi:

Processador: Intel I7 4790k 4.00GHz 4 CPUs

Memória RAM: 5Gb

Sistema operacional: Ubuntu 18.04

* 1. Preparação do ambiente para configuração do kernel

Após a instalação da máquina virtual os seguintes comandos foram executados para preparar o ambiente para as alterações no Kernel.

$sudo apt-get source linux-image-unsigned-$(uname -r) #Este comando irá baixar uma versão do Kernel igual à que está instalada atualmente na máquina usada, para isso é usado o comando $(uname -r) no final do nome do arquivo, ele irá completar o nome do arquivo que será baixado com a versão atual do Kernel.

$sudo apt install build-essential libncurses5-dev #Instalação dos pacotes necessários para exibição da interface do menu de configurações

$sudo passwd root #Este comando foi necessário para definir uma senha para o usuário root.

$su root #Após a definição da senha foi alterado para usuário root, por ser necessário permissões especiais para edição do Kernel.

$tar -xvf linux\_4.15.0.orig.tar.gz -C /usr/src/ #Os arquivos baixados anteriormente são extraídos dentro da pasta /usr/src

$cd /usr/src #Diretório é alterado para o local onde o arquivo foi extraído

$ln -s linux-4.15 linux #Este comando cria uma pasta com um link que aponta para o arquivo extraído, isso facilitará na digitação dos caminhos e caso a pasta linux seja deletada por acidente, não irá afetar a pasta original que está apontada. Feito isso, o ambiente está preparado para iniciar a edição do Kernel.

* 1. Configuração do kernel

$make menuconfig #Após a execução deste comando o menu abaixo será exibido

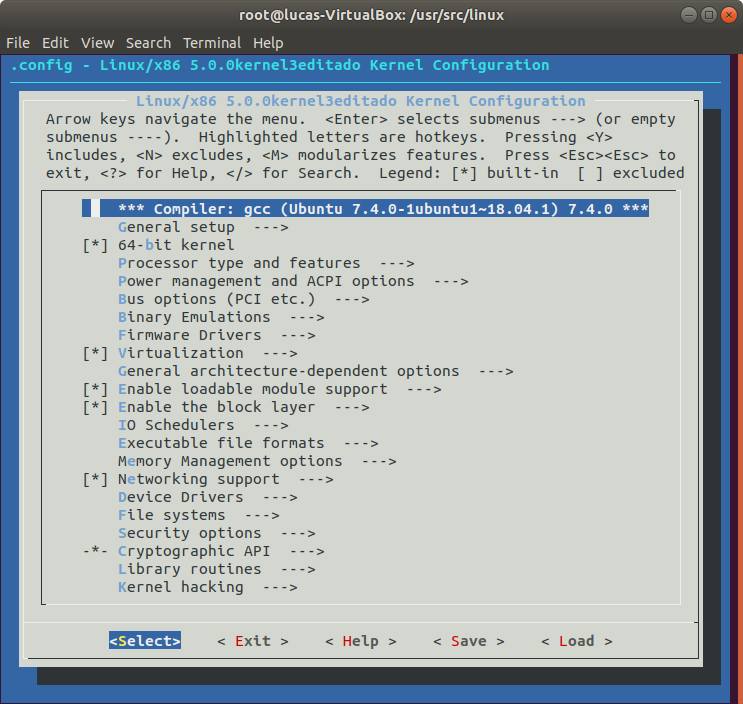


Figura 1- Menu de configurações do Kernel

O menu acima é utilizado para editar as configurações do Kernel, como módulos que serão instalados, dispositivos que podem ser habilitados ou não, os [\*] significam que o item selecionado será instalado completamente, [M] significa que será instalado como modulo e [ ] significa que o dispositivo não está setado, ou seja, não vai ser instalado. Os itens com ---> indicam que existe um sub-menu com mais opções que podem ser alteradas. Após as alterações serem feitas, deve-se salvar e sair para dar continuidade ao procedimento.

* 1. Compilação e instalação do Kernel

$nano Makefile # Este comando irá abrir o arquivo Makefile com editor nano, dentro do arquivo, será alterado o parâmetro EXTRAVERSION, que irá definir o nome da versão modificada do Kenrel.

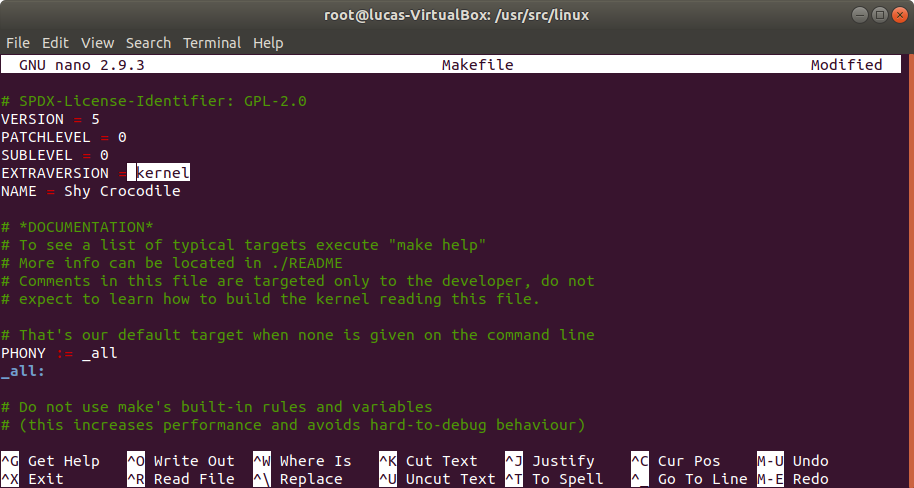


Figura 2- Arquivo Makefile aberto com o nano editor de textos

$make bzImage

Este comando irá gerar um arquivo de imagem do Kernel. Pode demorar, dependendo da quantidade de módulos que serão instalados.

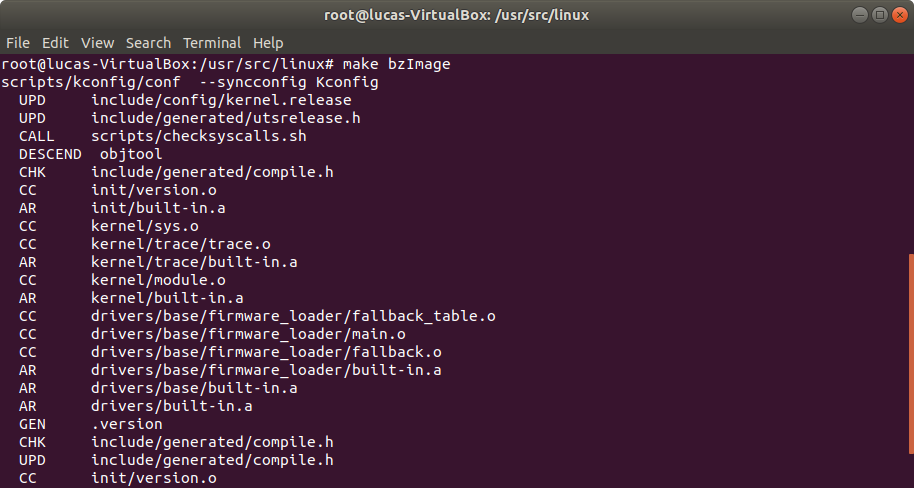


Figura 3- Processo de criação da imagem do Kernel

$make modules

Após este comando os módulos serão compilados, está etapa do procedimento pode demorar bastante, dependendo da quantidade de módulos que serão instalados.

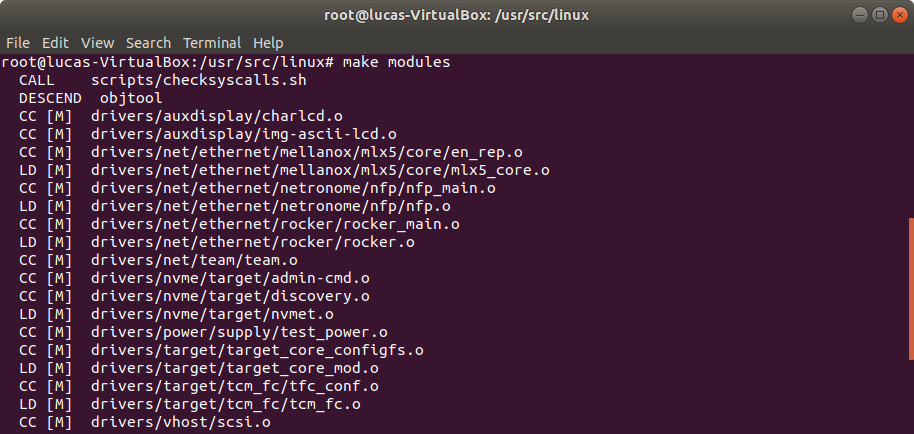


Figura 4- Compilação dos módulos

$make modules\_install | tee modulosInstalados.txt #Instala os módulos e salva um log dos módulos que foram instalados no arquivo modulosInstalados.txt.

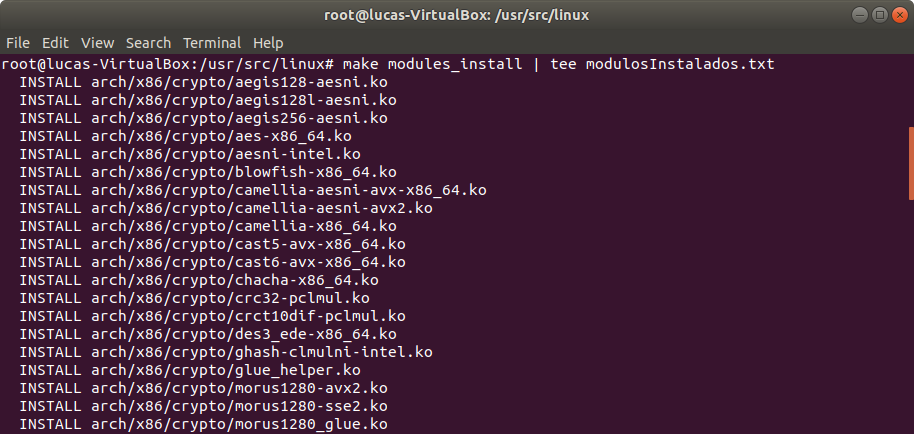


Figura 5- Instalação dos módulos

$make install #Instala o Kernel.

$mkinitramfs -o initrd.img-5.0.0kernel3editado 5.0.0kernel3editado # Criação da imagem de inicialização.

$update-grub #Atualiza o GRUB para exibição do novo Kernel editado, assim o usuário pode seleciona-lo para dar boot.

$uname –r #Após a máquina virtual ser reiniciada este comando foi executado para confirmar a instalação do novo Kernel.

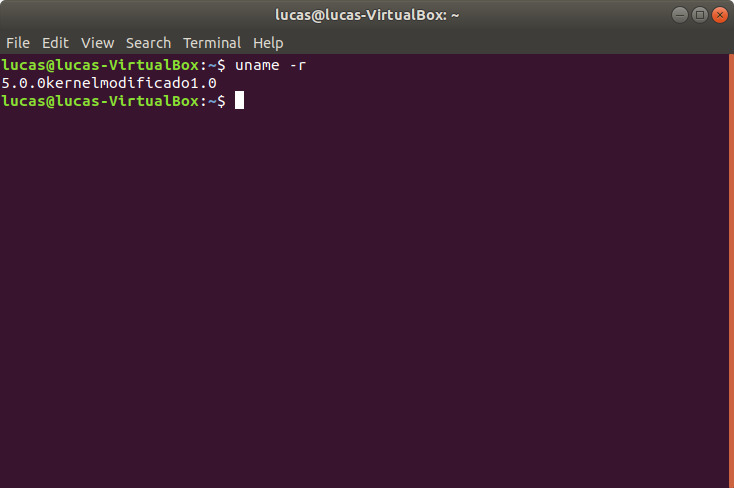


Figura 6- Versão de Kernel instalado

1. RESULTADOS

Como foi dito o kernel vem compilado com as funcionalidades mais genéricas possíveis, o que faz com que o tempo de compilação exceda 2 horas facilmente, e faz com que o tamanho do kernel também cresça, para se ter uma ideia foi feita uma análise do arquivo de configuração do kernel e criado uma tabela comparando a quantidade de funcionalidades habilitadas no kernel genérico e no kernel modificado:

Tabela 1- Comparação entre as versões do Kernel

|  |  |  |  |
| --- | --- | --- | --- |
| TIPO DE COMPILAÇÃO | KERNEL GENERICO | KERNEL MODIFICADO | DIFERENÇA |
| IN-KERNEL | 2482 | 1353 | 1129 |
| MÓDULOS | 5132 | 45 | 5090 |

Na compilação do kernel, podemos escolher a forma que as funcionalidades e drivers serão compiladas

Compilações *in-kernel*, são funcionalidades e drivers que são compilados junto com o kernel.

Módulos são programas que podem ser anexados ao kernel durante sua execução quando algum hardware o requisita

Analisando a tabela vimos que a quantidade de funcionalidades habilitadas no kernel genérico é superiormente alta comparada a modificada. Percebemos também que as funcionalidades compiladas *in-kernel,* no kernel genérico, só não é maior para não correr o risco de se ter um alto uso de memória após o carregamento do kernel, fazendo com que a quantidade de funcionalidades compiladas como módulos cresça para que o kernel consiga lidar com qualquer tipo de hardware.

A máquina virtual virtualiza também os drivers usados pelo sistema, com o comando *lspci -vv* descobrimos os nomes dos modelos correspondentes usados, conforme exemplo abaixo:

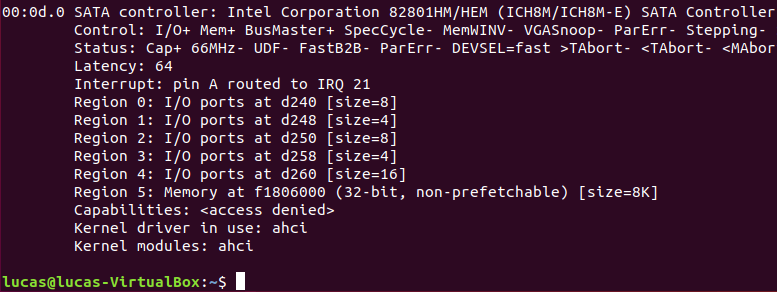


Figura 7- Detalhe de controlador com seu respectivo modulo

Da mesma forma, conseguimos também, com a ajuda do comando *Ismod* saber quais módulos estavam sendo usados pela máquina.

Juntando essas informações, procuramos e desativamos as funcionalidades e dispositivos que não estavam sendo usadas pela máquina virtual.

1. ANALISE DOS RESULTADOS

Com essas modificações o tempo de compilação caiu drasticamente, que passou de aproximadamente 3 horas no kernel genérico, para 30 minutos no kernel modificado.

A exclusão de drivers e módulos vistos como desnecessários, com base nos componentes que estão em uso no computador, para a compilação do kernel permite a redução do número de drivers a serem carregados tanto na compilação, quanto posteriormente na instalação do kernel customizado.

Além da compilação, o processo desde a aquisição do kernel do Linux, o acesso do mesmo através do terminal até chegar no menu de configuração do kernel foram essenciais para o nosso próprio aprendizado, além de serem fundamentais para as próximas atividades que venham a utilizar manipulações no kernel.

1. CONCLUSÕES
2. REFERENCIAS

Slides da Aula 01 disponível no AVA.

Material de apoio “Atividade 1: Compilação de um kernel Linux” disponível no AVA.

<https://www.diolinux.com.br/2017/07/como-compilar-um-kernel-linux-passo-a-passo.html> (Acesso: 13/08/2019 19:27)