

SUMÁRIO

[1.](#_gjdgxs) INTRODUÇÃO 2

[2.](#_30j0zll) OBJETIVOS 2

[3.](#_1fob9te) PROCEDIMENTO UTILIZADO 2

[4.](#_3znysh7) RESULTADOS 2

[5.](#_2et92p0) ANALISE DOS RESULTADOS 2

[6.](#_tyjcwt) CONCLUSÕES 2

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

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. 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:

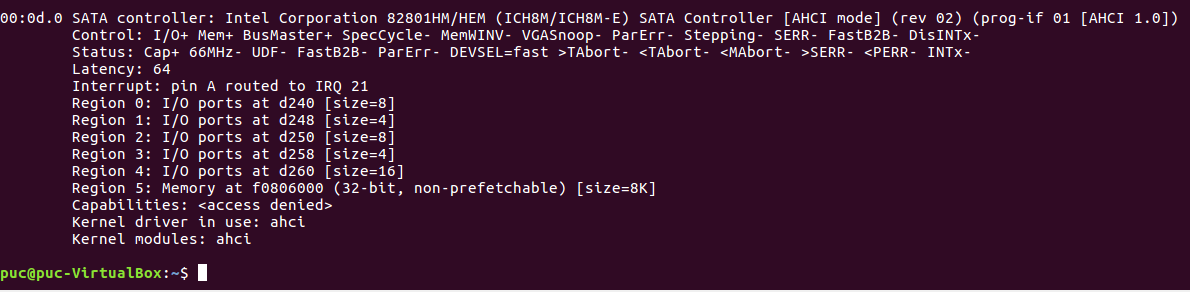
|  |  |  |  |
| --- | --- | --- | --- |
| TIPO DE COMPILAÇÃO | KERNEL GENERICO | KERNEL MODIFICADO | DIFERENÇA |
| IN-KERNEL | 2482 | 1353 | 1129 |
| MÓDULOS | 5135 | 45 | 5090 |

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:

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. ANÁLISE DOS RESULTADOS

Com essas modificações o tempo de compilação caiu drasticamente, que passou de aproximadamente 3 horas no kernel genérico, para 1 hora 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)