Universidade Federal de Uberlândia
FEELT – Faculdade de Engenharia Elétrica

Resumo dos tópicos do Vídeo

Prof. Éder Alves de Moura

Sistemas Embarcados 2

Aluno: Pedro Jacob Favoreto - 11721EAU003

Data: 10/01/2022

1. About Linus and Linux Creation 1:00
O primeiro tópico inicia falando que a primeira versão do linux foi desenvolvida em 1991. Ele iniciou como um emulador terminal de simples escrita. Atualmente o Linux é um sistema operacional full-fledged. Ele consegue rodar tanto em pequenas máquinas quanto em grandes máquinas. O linux é parecido com um sistema Unix, mas não é um sistema UNIX.

2. How Linux Kernel was same and different to other kernels when it was created 4:25
O segundo tópico inicia falando que o Kernel podem ser divididos em dois principais tipos de design, que são eles: o núcleo monolítico e o micro Kernel. O Kernel monolítico é o design mais simples dos dois e todos os kernels foram projetados desta maneira até a década de 1980. Ele é implementado inteiramente como um único processo kernel monolítico são implementados inteiramente como um único processo em execução em um único espaço de endereço. Estes Kernels normalmente existem no disco como binários únicos, então todos os serviços do kernel existem e executam no grande espaço de endereço do kernel. A comunicação dentro do Kernel é trivial, porque tudo funciona em modo kernel no mesmo espaço de endereço. O kernel pode invocar funções diretamente como um aplicativo de espaço do usuário pode. O outro é o Micro Kernel, o qual não é implementado como um único grande processo em vez da funcionalidade do kernel é dividido em processos separados e esses são geralmente chamados de servidores. Todos os servidores são separados em diferentes espaços. Nele a comunicação é feita por uma mensagem passando um mecanismo de comunicação entre processos é construído no sistema e nos vários servidores comunicar e invocar serviços um do outro enviando mensagem pelo IPC (Inter Process Communication). Desta forma, Linux é um Kernel monolítico executado em um espaço de endereço único inteiramente no modo kernel. Então o Linux assume algumas coisas boas dos micro kernels.
Linux boost como um design modular, então o kernel linux funciona com invocação de função direta, não apresentando mensagem de passagem à medida que a finura e outros órgãos se desenvolvem para contribuir para o Linux Kernel. Eles decidem a melhor forma de avançar o linux sem negligenciar suas regras unix. Consequentemente, o linux não é baseado em nenhuma variante unix específica. A empresa pode escolher a melhor solução para qualquer problema, então há uma diferença notável entre o kenel do linux e o sistema unix clássico.

5. How coding inside the kernel is different then coding in user space 12:40

O espaço kernel é aquela área de memória virtual onde os processos do kernel serão executados. O espaço do usuário é aquela área de memória virtual onde os processos do usuário serão executados. Esta divisão é necessária para a memória para proteger de acessos. Então o espaço do usuário é a memória na qual o usuário utiliza quando os processos estão em execução.
Neste tópico foi diferenciado como codificar dentro do kernel é diferente da codificação no espaço do usuário.
Um kernel linux tem vários atributos em comparação com um espaço de usuário, embora essas diferenças não façam desenvolver o código do kernel mais difícil do que desenvolver o código do espaço do usuário, mas eles certamente tornam isso diferente.
O kernel não possui acesso nem à biblioteca c nem ao cabeçalho c padrão. O kernel é codificado em GNU C, que é muito parecido com qualquer kernel unix. O kernel pode matar

qualquer processo a qualquer momento se estiver causando algum problema para evitar o erro durante a execução de outros processos.

6. How processes are tracked and managed in kernel 18:26

Esse tópico começa falando que um processo é basicamente um programa que no estado de execução são mais do que apenas o código do programa em execução. Eles também incluem um conjunto de recursos como arquivos abertos e pendentes e sinalizam qualquer programa que está em estado de execução. O estado de execução é chamado de processo. O próprio programa não é um processo. Um processo é um programa ativo e está relacionado a alguns recursos. Então, quando um programa está em execução e solicita algum recurso em termo de sua memória principal ou em termos do seu disco rígido, chamamos de processo dois. Dois processos podem existir ao mesmo tempo e compartilhar vários recursos como: arquivo aberto ou um espaço de endereço no kernel linux. Um processo ocorre por meio de um sistema fork, que criam um novo processo duplicando um processo já existente. O processo chamado fork é o pai e o novo processo é a criança. O pai retorna a execução e a criança começa a executar no mesmo lugar. No linux, todo processo tem um pid que é chamado de processo único. O pid é um valor numérico que é atribuído a cada processo.

7. Threads in Linux 24:06

Nesse tópico é mostrado como as Threads são gerenciadas no Linux, sistema operacional e kernel Linux. Uma thread é basicamente a menor unidade de processamento que pode ser executada em um sistema operacional. Uma thread existe dentro de um processo que é um fio único que pode conter vários tópicos. Então, quanto temos que fazer várias tarefas em um único processo é criado uma thread para cada processo e elas são executadas simultaneamente e conseguimos realizar o processo desejado. As threads basicamente permitem ao concorrente programar a programação simultânea. è o conceito de programação paralela. O linux tem uma implementação única de threads para o kernel do linux. Cada thread é considerado um processo separado no kernel do linux. O kernel do linux não fornece nenhuma semântica de programação especial e dados estruturais para representar os fios. As threads são escalonáveis e preemptivas. Os threads do kernel são criados quando outros threads do kernel são inicializados.

8. Process Scheduling and Scheduling Algorithms 27:09

Nesse tópico é mostrado como é o processo e o agendamento feito no kernel do linux e quais são os algoritmos de agendamento que é usado no linux kernel.
Os processos são basicamente os programas ativos que estão em estado de execução sempre que qualquer programa está em estado de processamento. O kernel do Linux tem um sistema separado para gerenciar todos esses processos. E para colocar esse subsistema como um programador de processos, o programador de processos ou Agendador divide os recursos finitos do processo tempo entre os processos executáveis em um sistema. Quando há múltiplos processos em execução em um sistema, é tarefa do planejador dividir o recurso em vários processos, dessa forma, não pode haver nenhum tipo de gargalo no desempenho do sistema. Então, o planejador tem a função de atribuir os recursos para os processos de acordo com a necessidade.Dessa forma, ao decidir qual

processo será executado, o planejador é responsável por organizar dando aos usuários a impressão de que múltiplos processos estão sendo executados simultaneamente. Já o Agendador divide os recursos em processos e é ele que decide o que é executado primeiro. Para isso, existem algumas políticas que dizem quais processos devem ser executados primeiro e quanto tempo deve ser dado a qualquer processo particular. Uma politica de agendamento determina a execução geral de um sistema.

9. What is a System Call, how to call them 37:07

Esse tópico começa falando que o kernel fornece um conjunto de interfaces por quais processos em execução pode interagir com o sistema e a chamada de sistema é uma dessas utilidades pelos quais os processos em execução no espaço do usuário interagem com o sistema. Essas interfaces dão aplicativos controlados, acesso a um hardware, um mecanismo para criar novos processos e se comunicar com as interfaces existentes que atuam como o mensageiro entre o aplicativo e o kernel. Os aplicativos emitem vários pedidos e o kernel os preenche. A chamada é a forma programática em que um programa de computador solicita um serviço do kernel do sistema operacional em que é executado. Em geral, as chamadas do sistema são basicamente uma maneira de interagir com o operador do sistema. No kernel do linux temos diversos tipos de programações de interfaces de aplicativos que permitem a comunicação entre o espaço do usuário e as chamadas do sistema. Elas são tipicamente acessadas pelas chamadas de função, definidas como bibliotecas C, No linux cada chamada de sistema é atribuído um número de syscall, este é um número único que é usado para fazer referência a uma chamada de sistema específica. Quando um processo de espaço do usuário executa uma chamada de sistema. O número de chamada do sistema identifica qual chamada do sistema foi executada quando uma determinada solicitação é gerada a partir do espaço do usuário e sempre que temos que chamar um sistema, nós especificamos o número de modo que seja identificado e qual chamada de sistema deve ser chamada naquele momento. Siscall é um número importante, então quando é atribuído não pode ser mudado, pelo contrário pode ocorrer problemas.

10. System Call implementation in the kernel 41:10

Neste tópico é falado como implementar uma chamada do sistema no kernel. A implementação real de chamada de sistema do linux não precisa ser preocupado com o comportamento do gerenciador de chamadas do sistema. A primeira etapa de implementação de um sistema de chamada está definido pelo seu propósito. A chamada do sistema deve ter uma interface limpa e simples com o menor número de argumentso possiveis. A semântica e o comportamento das chamadas de sistema não são importantes. Eles não devem mudar porque existem formulários que eles confiam com o propósito de a chamada do sistema permanecer constante. A chamada do sistema é executada no espaço do kernel e o usuário pode passar uma entrada invalida no kernel.

12. what is an interrupt and how they are handled in kernel 54:56

Esse tópico começa falando que uma interrupção é uma resposta do processador a um evento que precisa de atenção do software. Dessa forma, a interrupção gera um evento que perguntará ao software para dar a resposta dele. O kernel precisa se comunicar com os

dispositivos individuais da máquina como se tivéssemos vários dispositivos de hardware em nosso sistema de computador, então o kernel tem que gerenciar todos esses dispositivos de hardware. Os processadores podem ser pedidos de magnitude mais rápida que o hardware. Não é ideal para o kernel emitir um pedido e esperar por uma resposta do hardware significativamente mais lenta. Outro trabalho importante é lidar com o hardware somente após ele realmente completar o trabalho.

13. What is an IRQ? 54:56

É chamado de interrupção de ativação para sinalizar para o processador qual é o dispositivo de hardware que gerencia os problemas do teclado e sinal elétrico para o processador para alertar o sistema operacional para novos processos chaves disponíveis. Um número IRQ é um identificador de kernel usado para falar sobre uma fonte de interrupção de hardware. Normalmente, este é um índice no array global irq_desc, mas com exceção do que o linux/interrupt.h implementa os detalhes são específicos da arquitetura. Um número irq é uma enumeração das possíveis fontes de interrupção em uma máquina. Normalmente o que é mencionado é o número de pinos de entrada em todo o controlador de interrupção no sistema. No caso do ISA o que é enumerado são os 16 pinos de entrada nos dois controladores de interrupção i8259. As arquiteturas podem atribuir significado adicional aos números do IRQ e são encorajadas no caso em que há qualquer configuração manual do hardware envolvido. Os IRQs ISA são um exemplo clássico de atribuir esse tipo de significado adicional.

15. About critical regions and race conditions, how to protect? 1:06:52

As condições de corrida geralmente envolvem um ou mais processos acessando um recurso compartilhado, onde esse acesso múltiplo não foi devidamente controlado.
Em geral, os processos não são executados automaticamente; outro processo pode interrompê-lo entre essencialmente duas instruções. Se o processo de um programa seguro não estiver preparado para essas interrupções, outro processo poderá interferir no processo do programa seguro. Qualquer par de operações em um programa seguro ainda deve funcionar corretamente se as quantidades arbitrárias do código de outro processo forem executadas entre elas.

17. Understanding Kernel Notion of Time 1:17:22

O hardware fornece um temporizador de sistema que o kernel usa para medir a passagem do tempo. Este temporizador do sistema funciona a partir de uma fonte de tempo eletrônica, como um relógio digital ou a frequência do processador. O temporizador do sistema dispara em uma frequência pré-programada, chamada de
taxa de carrapato. Quando o temporizador do sistema se apaga, ele emite uma interrupção que o kernel manuseia através de um manipulador especial de interrupção. Como o kernel conhece a taxa de carrapato pré-programada, ele sabe o tempo entre duas interrupções consecutivas do temporizador. É assim que o kernel mantém o controle do tempo de parede e do tempo de atividade do sistema. Tempo de parede A hora real do dia é de maior importância para aplicativos de espaço do usuário. O kernel mantém o controle simplesmente porque o kernel controla a interrupção do temporizador. Uma família de chamadas do sistema

fornece a data e a hora do dia para o espaço do usuário. O tempo de atividade do sistema o tempo relativo desde a inicialização do sistema é útil tanto para o espaço do kernel quanto para o espaço do usuário. Muitos códigos devem estar cientes da passagem do tempo. A diferença entre duas leituras de tempo de atividade agora e, em seguida, é uma simples medida desta relatividade. A interrupção do temporizador é muito importante para a gestão do sistema operacional. Um grande número de funções de kernel vivem e morrem com o passar do tempo.

19. Kernel Memory Management Theory 1:26:11

O subsistema de gerenciamento de memória Linux é responsável, como o nome indica, pelo gerenciamento da memória no sistema. Isso inclui a implementação de memória virtual e paginação de demanda, alocação de memória tanto para estruturas internas do kernel quanto para programas espaciais do usuário, mapeamento de arquivos em processos de espaço de endereço e muitas outras coisas legais. O gerenciamento de memória Linux é um sistema complexo com muitas configurações. A maioria dessas configurações estão disponíveis via sistema de arquivos e podem ser ajustadas.

24. Filesystem Abstraction Layer 1:46:45

A interface genérica para qualquer tipo de sistema de arquivos só é viável porque o próprio kernel implementa uma camada de abstração em torno de sua interface de sistema de arquivos de baixo nível. Esta camada de abstração permite que o Linux suporte diferentes sistemas de arquivos, mesmo que eles diferem muito em recursos ou comportamento suportados. Isso é possível porque o VFS fornece um modelo de arquivo comum capaz de representar os recursos e comportamentos gerais de qualquer sistema de arquivos concebíveis. A camada de abstração funciona definindo as interfaces conceituais básicas e estruturas de dados que todos os sistemas de arquivos suportam. Os sistemas de arquivos moldam sua visão de conceitos para corresponder às expectativas do VFS. O código real do sistema de arquivos oculta os detalhes da implementação. Para a camada VFS e o resto do Kernel, no entanto, cada sistema de arquivos parece o mesmo. Todos eles suportam noções como arquivos e diretórios, e todos eles suportam operações como criação e exclusão de arquivos. O resultado é uma camada geral de abstração que permite que o Kernel suporta muitos tipos de sistemas de arquivos de forma fácil e limpa. Os sistemas de arquivos são programados para fornecer as interfaces abstratas e estruturas de dados que o VFS espera; por sua vez, o Kernel funciona facilmente com qualquer sistema de arquivos e a interface de espaço de usuário exportada funciona perfeitamente em qualquer sistema de arquivos.