Universidade Federal de Uberlândia
FEELT - Faculdade de Engenharia Elétrica

SISTEMAS EMBARCADOS II

Resumo dos Tópicos do Vídeo

VILSON CAMILO BORGES DE MORAES NEVES 12011EAU020

Uberlândia/MG

03 de Março de 2024

Sumário

1 3
2 3
5 3
6 3
7 4
8 4
9 4
12 4
13 5
15 5
17 6
19 6
24 7

1

Linux teve sua criação por um estudante, Linus, que estava insatisfeito com os sistemas operacionais existentes (Windows). Linus evoluiu e melhorou, durante sua vida acadêmica, seu emulador de terminal até chegar em um unix completo (mas imaturo). Esta foi a primeira versão, em 1991, lançado na internet como um sistema operacional para computadores.

O Linux cresceu rapidamente, pois atraiu rapidamente muitos desenvolvedores. Tornou-se um projeto colaborativo desenvolvido e melhorado por muitas pessoas, o que o traz hoje como um completo sistema operacional, funcionando em simples e complexas máquinas.

2

Pode-se dividir o Kernel em dois grandes grupos: Monolitchic Kernel Design e Mocrokernel Design.

O primeiro é o modelo mais simples, utilizado como molde para os Kernels até 1980. É implementado como um único processo em execução em um único espaço de endereço, e tudo acontece neste espaço.

Já o segundo é dividido em vários processos separados: servidores, e são separados em endereços diferentes. A comunicação, diferentemente da utilizada no grupo anterior, é construída nos sistemas, invocando serviços por mensagens pelo IPC (inter processo de comunicação.

5

Essa sessão percorreu pelo tema “Espaço do usuário”, que é um sistema de memória na qual os processos do usuário estão em execução, abordando como a codificação dentro do Kernel é diferente da codificação do usuário

6

Para permitir que o kernel gerencie os processos, cada processo é representado por um descritor de processo que inclui informações sobre o estado atual do processo. Quando o kernel para a execução de um processo, ele salva o conteúdo atual de vários registros do processador no descritor do processo.

7

O Linux tem uma implementação única de threads. Para o kernel Linux, não existe o conceito de thread. O Linux implementa todos os threads como processos padrão. O kernel do Linux não fornece nenhuma semântica de agendamento especial ou estruturas de dados para representar threads. Em vez disso, um thread é apenas um processo que compartilha certos recursos com outros processos. Cada thread possui um task_struct exclusivo e aparece para o kernel como um processo normal (que apenas compartilha recursos, como um espaço de endereço, com outros processos).

8

O Linux usa um algoritmo CFS *(Completely Fair Scheduling, programação* completa), que é uma implementação de *filas justas ponderadas* (WFQ). Em um único sistema de CPU, o tempo do CFS corta o tempo da CPU entre os segmentos em execução. Há um intervalo de tempo fixo durante o qual cada segmento no sistema deve ser executado pelo menos uma vez. Este intervalo é dividido em *timeslices* que são alocadas em roscas de acordo com seus pesos.

9

Uma chamada de sistema é uma função primitiva no kernel do sistema que pode ser chamada a partir de aplicativos. É o nível mais baixo de abstração do sistema operacional e os únicos pontos de entrada de baixo nível disponíveis para os aplicativos. Pode ser tão primitivo quanto uma função enviar alguns bytes de dados para um dispositivo de saída atual.

12

Uma interrupção é um evento que altera o fluxo normal de execução de um programa e pode ser gerado por dispositivos de hardware ou até mesmo pela própria CPU. Quando na interrupção ocorre o fluxo atual de execução, é suspenso e interrompe as corridas do manipulador. Depois que o manipulador de interrupção executa o fluxo de execução anterior é retomado.

As interrupções podem ser agrupadas em duas categorias com base na fonte da interrupção. Eles também podem ser agrupados em duas outras categorias com base na

capacidade de adiar ou desativar temporariamente a interrupção: síncrono, assíncrono, mascarado e não mascarado.

13

Um IRQ é uma solicitação de interrupção de um dispositivo. Atualmente, eles podem entrar por cima de um pino, ou sobre um pacote. Vários dispositivos podem ser conectados ao mesmo pino, compartilhando assim um IRQ.

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 enumerado 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

As condições de corrida geralmente envolvem um ou mais processos acessando um recurso compartilhado (tal arquivo ou variável), onde esse acesso múltiplo não foi devidamente controlado.

Em geral, os processos não são executados atomicamente; 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

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 (muitas vezes chamado *de bater* ou *estalar*) 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. Este período é chamado de *carrapato* e é igual a *um sobre-a-taxa de carrapato* segundos. 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

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 configuráveis. A maioria dessas configurações estão disponíveis via sistema de arquivos e podem ser ajustadas usando.

24

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 suporte 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.