1.Sistemas monolíticos

Essa estruturação de um SO é tal que não há nenhuma estrutura. O sistema operacional é escrito como um conjunto de rotinas, cada uma das quais podemos chamar qualquer uma das outras sempre que precisar. Quando essa técnica é utilizada, cada rotina do sistema tem uma interface bem definida em termos de parâmetros e de resultados e cada uma está livre para chamar qualquer uma das outras, se a última fornecer alguma computação útil de que a primeira precise.

Para sua construção, deve-se compilar todas as rotinas individualmente e então, ligá-las em um único arquivo objeto usando o *linker* do sistema.

Os serviços (chamadas de sistema) fornecidos pelo sistema operacional são solicitados colocando-se os parâmetros em lugares bem definidos, como em registradores ou na pilha e, então, executando-se uma instrução de interrupção especial, conhecida como chamada de núcleo ou chamada de supervisão.

Essa instrução troca o modo do usuário para o modo núcleo e transfere o controle para o sistema operacional.

Para executar a função de biblioteca *read*, que realmente faz a chamada de sistema *read*, o programa primeiro insere os parâmetros na pilha e então, os compiladores C e C++ colocam os parâmetros na pilha na ordem inversa. O primeiro e o terceiro parâmetro são chamados por valor, mas o segundo parâmetro é passado por referência, significando que é passado o endereço do buffer e não seu conteúdo. Em seguida, vem a chamada *real* para a função *read* da biblioteca que é, essencialmente, uma chamada normal de execução de qualquer rotina.

A função da biblioteca, possivelmente escrita em assembly, normalmente coloca o código numérico correspondente à chamada de sistema em um lugar esperado pelo sistema operacional, como em um registrador. Em seguida, ela executa uma instrução *TRAP* para trocar do modo usuário para o modo núcleo e inicar a execução de um endereço fixo dentro do núcleo. Então, o núcleo inicia examinando o código numérico da chamada de sistema para depois chamar a rotina de tratamento correta, normalmente feito através de uma tabela de ponteiros para rotinas de tratamento de chamada de sistema, indexada pelo número de chamada de sistema.

A rotina de tratamento de chamada de sistema nesse ponto é executada. Quando a rotina de tratamento de chamada de sistema terminar seu trabalho, o controle poderá ser retornado para a função de biblioteca no espaço de usuário, na instrução que segue a instrução *TRAP*.

Para concluir a tarefa, o programa do usuário precisa limpar a pilha, como faz após qualquer chamada de rotina. O código compilado incrementa o ponteiro da pilha exatamente o suficiente para remover os parâmetros colocados antes da chamada *read*. Agora, o programa está livre para fazer o que quiser em seguida.

Essa organização sugere uma estrutura básica para o sistema operacional

1. Um programa principal que ativa a função de serviço solicitada

2. Um conjunto de funções de serviço que executam as chamadas de sistema

3. Um conjunto de funções utilitárias que ajudam as funções de serviço

Nesse modelo, para cada chamada de sistema há uma função de serviço que cuida dela. As funções utilitárias fazem coisas que são necessárias para várias funções de serviço, como buscar dados de programas de usuários. Essa divisão ocorre em três camadas previamente citadas.

1.2 Sistemas em camadas

Outra forma de organizar o sistema operacional é como uma hierarquia de camadas, cada uma construída sobre a outra. O primeiro sistema feito dessa maneira foi o THE. Ele tinha seis camadas como mostrado na tabela abaixo.

Camada	Função
5	Operador
4	Programas de usuário
3	Gerenciamento de entrada/saída
2	Comunicação operador-processo
1	Gerenciamento de memória e tambor
0	Alocação do processador e multiprogramação

Acima da camada 0, o sistema possuía processos sequenciais, cada um dos quais podia ser programado sem se preocupar com o fato de que vários processos estavam sendo executados num único processador. Em outras palavras, a camada 0 proporcionava multiprogramação básica da CPU.

A camada 1 realizava o gerenciamento de memória. Ela alocava espaço para processos na memória principal e em um tambor com 512K de palavras, utilizado para conter partes das páginas para os quais não havia espaço na memória principal. Acima da camada 1, os processos não precisavam se preocupar com o fato de estarem na memória ou no tambor.

A camada 2 manipulava a comunicação entre cada processo e o console do operador. Acima dessa camada, cada processo tinha efetivamente seu próprio console de operador.

A camada 3 gerenciava os dispositivos de E/S e armazenava em buffers os fluxos de informação. Acima da camada 3, cada processo podia lidar com os dispositivos de E/S abstratos com interfaces amigáveis, em vez de dispositivos reais cheios de peculiaridades.

A camada 4 era onde ficavam os programas de usuário. Eles não tinham de preocupar-se com gerenciamento de processos, de memória, de console ou de E/S. O processo do operador do sistema localizava-se na camada 5.

Uma generalização maior do conceito de camadas estava presente no sistema MULTICS. Em vez de camadas, ele era organizado como uma série de anéis concêntricos, com os anéis internos sendo mais privilegiados do que os externos. Quando uma função em

um anel externo queria chamar uma função em um anel interno, ela tinha que fazer o equivalente de uma chamada de sistema, isto é, uma instrução *TRAP*, cujos parâmetros eram cuidadosamente verificados, antes de permitir que a chamada prosseguisse. Embora no MULTICS, o sistema operacional inteiro fizesse parte do espaço de endereçamento de cada processo de usuário, o hardware tornava possível designar individualmente funções como protegidas contra leitura, escrita ou execução.

1.3 Máquinas virtuais

O centro do sistema, conhecido como monitor de máquina virtual, era executado no hardware básico e fazia multiprogramação, oferecendo não uma, mas várias máquinas virtuais à camada superior seguinte. Entretanto, ao contrário de todos os outros sistemas operacionais, essas máquinas virtuais não eram máquinas estendidas, com arquivos e com outros recursos interessantes. Em vez disso, elas eram cópias exatas do hardware básico, incluindo os modos núcleo e usuário, E/S, interrupções e tudo mais que uma máquina real tem.

Como cada máquina virtual é idêntica ao hardware verdadeiro, cada uma pode executar qualquer sistema operacional que fosse executado diretamente no hardware básico. Diferentes máquinas virtuais podem executar diferentes sistemas operacionais.

Quando um programa CMS (Conversational Monitor System) executa uma chamada de sistema, essa chamada é capturada pelo sistema operacional em sua própria máquina virtual, exatamente como se estivesse sendo executada em uma máquina real. Então, o CMS envia as instruções normais de E/S de hardware para ler seu disco virtual ou o que for necessário para executar a chamada. Essas instruções de E/S são capturadas pelo VM/370, que então as executa como parte de sua simulação do hardware real. Fazendo uma separação completa das funções de multiprogramação e fornecendo uma máquina estendida, cada uma das partes se torna muito mais simples, mais flexível e mais fácil de manter.

A Intel fornece um modo virtual do 8086 no Pentium. Nesse modo, a máquina age como um 8086 (idêntica a um 8088, do ponto de vista de software também), incluindo o endereçamento de 16 bits com um limite de 1 MB.

Este modo é utilizado pelo Windows e por outros sistemas operacionais para executar programas mais antigos do MS-DOS. Esses programas são iniciados no modo 8086 virtual. Contanto que executem instruções normais, eles funcionam no hardware básico. Entretanto, quando um programa tenta interromper o sistema operacional para fazer uma chamada de sistema, ou tenta fazer E/S protegida diretamente, ocorre uma interrupção no monitor da máquina virtual.

Duas variantes desse projeto são possíveis. Na primeira, o próprio MS-DOS é carregado no espaço de endereçamento do 8086 virtual, de modo que o monitor da máquina virtual apenas reflete a interrupção para o MS-DOS, exatamente como aconteceria em um 8086 real. Quando, posteriormente, o próprio MS-DOS tentar fazer a E/S, essa operação será capturada e executada pelo monitor da máquina virtual.

Na outra variante, o monitor da máquina virtual apenas captura a primeira interrupção e faz a E/S sozinho, pois conhece todas as chamadas de sistema do MS-DOS e, portanto, o

que cada interrupção deve fazer. Esta variante é menos pura do que a primeira, já que simula corretamente apenas o MS-DOS e não outros sistemas operacionais, como a primeira.

Uma desvantagem em executar o MS-DOS no modo 8086 virtual é que o MS-DOS desperdiça muito tempo habilitando e desabilitando interrupções, o que implica em custo considerável para simular um processo.

Diversas implementações de máquina virtual são vendidas comercialmente. O VMWare e o Virtual PC da Microsoft são comercializados para tais instalações. Essesprogramas utilizam arquivos grandes no sistema *host* para simular os discos de seus sistemas *guest*, aqueles executados pela máquina virtual. Para obter eficiência, eles analisam os arquivos binários do programa de sistema *guest* e permitem que código seguro seja executado diretamente no hardware do hospedeiro, capturando instruções que fazem chamadas de sistema operacional.

Outra área onde máquinas virtuais são usadas é na execução de programas Java. Quando a Sun Microsystems inventou a linguagem de programação Java, inventou uma máquina virtual chamada JVM. O compilador produz código para JVM, o qual então é normalmente executado por um interpretador JVM, em software. A vantagem dessa estratégia é que o código da JVM pode ser enviado pela Internet para qualquer computador que tenha um interpretador JVM e executado no destino. Se o compilador tivesse produzido programas binários em SPARC ou Pentium, eles não poderiam ser enviados e executados em qualquer lugar tão facilmente.

1.4 Exonúcleos

Na camada inferior, executando em modo núcleo, existe um programa chamado exonúcleo. Sua tarefa é alocar recursos para as máquinas virtuais e, então, verificar tentativas de utilizá-lo para garantir que nenhuma máquina use recursos pertencentes à outra pessoa. Cada máquina virtual em nível de usuário pode executar seu próprio sistema operacional.

A vantagem deste esquema é que ele economiza uma camada de mapeamento. Em outros projetos, cada máquina virtual “enxerga” um disco próprio, com blocos que vão do 0 até algum valor máximo, de modo que o monitor de máquina virtual precisa manter tabelas para fazer um novo mapeamento dos endereços de disco. Com o exonúcleo, esse novo mapeamento não é necessário. O exonúcleo apenas precisa monitorar qual recurso foi designado para qual máquina virtual. Esse método tem ainda a vantagem de separar multiprogramação do código do sistema operacional do usuário, mas com menor sobrecarga.

1.5 Modelo cliente-servidor

Uma tendência nos sistemas operacionais modernos é levar ainda mais longe essa ideia de mover código para camadas mais altas e remover o máximo possível do sistema operacional, deixando um *kernel* mínimo, o *microkernel*. A estratégia normal é implementar a maior parte das funções do sistema operacional em processos de usuário. Para solicitar um serviço, como ler um bloco de arquivo, um processo de usuário (processo cliente) envia uma requisição para um processo servidor, o qual então realiza o trabalho e devolve a resposta.

Tudo que o *kernel* faz é gerenciar a comunicação entre clientes e servidores. Dividir o sistema operacional em partes, cada uma gerenciando apenas uma faceta do sistema, como serviços de arquivo, serviços de processo, serviços de terminal ou serviços de memória, torna cada parte pequena e gerenciável. Além disso, como todos os servidores são executados como processos em modo usuário e não modo *kernel*, eles não têm acesso direto ao hardware. Como consequência, se ocorrer um erro no servidor de arquivos, o serviço de arquivos pode falhar, mas isso normalmente não derrubará a máquina inteira.

Algumas funções do sistema operacional são difíceis, senão impossíveis de serem feitos a partir de programas em espaço de usuário. Há duas maneiras de lidar com esse problema. Uma delas é fazer com que alguns processos servidores críticos, como E/S, sejam executados realmente em modo núcleo, com acesso completo a todo hardware, mas ainda se comuniquem com outros processos, utilizando o mecanismo de mensagens.

A outra maneira é construir um mínimo do mecanismo no *kernel*, deixando as decisões políticas para os servidores no espaço de usuário. Por exemplo, o *kernel* poderia reconhecer que uma mensagem enviada para um certo endereço especial significa pegar o conteúdo dessa mensagem e carregá-lo nos registradores do dispositivo de E/S de algum disco, para iniciar uma leitura de disco. Nesse exemplo, o núcleo nem mesmo inspecionaria os bytes presentes na mensagem para ver se seriam válidos ou significativos, ele apenas os copiaria cegamente nos registradores de dispositivo do disco. A divisão entre mecanismo e política é um conceito importante, ela ocorre repetidamente nos sistemas operacionais em diversos contextos.