Sistemas Operacionais

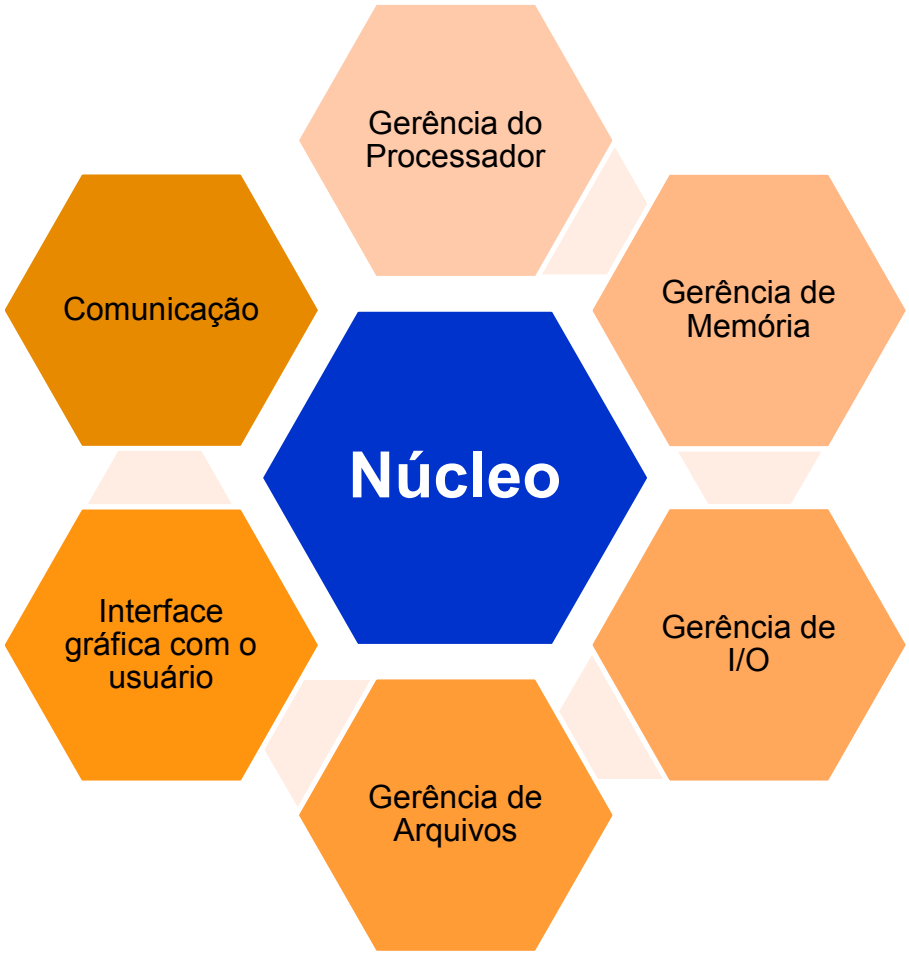
Estrutura

Introdução

- Análise de um Sistema Operacional
  1. Serviços fornecidos
  2. Interface fornecida a usuários e programadores
  3. Componentes e interconexões

- Nem todos os Sistemas possuem a mesma estrutura, mas existem alguns aspectos comuns aos quais dão suporte.

Introdução


Gerência do Processador
Comunicação
Gerência de Memória
Núcleo
Interface gráfica com o usuário
Gerência de I/O
Gerência de Arquivos

Introdução (2)

- Mecanismo de Interrupções
- Proteção
- Chamadas
- Estrutura

Mecanismo de Interrupções

- Um controlador de periférico que precisa enviar dados ao processador pode:
  - Aguardar até que o processador esteja livre e o mesmo o consulte para verificar se há alguma informação para o processador

  ou

  - Gerar uma interrupção (requisição de interrupção – *IRQ – Interrupt ReQuest*) por meio do barramento de controle
- O mecanismo de interrupções torna eficiente a interação do processador com os periféricos e permite E/S assíncrona.

Mecanismo de Interrupções

- Sinalizar ocorrência de eventos
- Tratador de interrupção
- Interrupção pode ser ocasionada por hardware, software ou por uma exceção
  - Exceções: overflow, instrução ilegal, divisão por zero...
- Vetor de Interrupções
- Execução de interrupção
  - Transfere o controle para o tratador (salva o contexto de execução)
  - Desvia controle para o tratador
  - Retorna execução (restaura contexto de execução)
- Chamada de Sistema
  - Processo usuário solicita serviços ao sistema operacional
    - Interrupções de software (traps)

Proteção

- O sistema operacional deve garantir a correta utilização dos recursos computacionais e propiciar a execução das aplicações no computador.

- Para isto, é necessário mecanismos de proteção para que as aplicações não consigam acessar o hardware diretamente e sim, que isto seja mediado pelo sistema operacional.

- Há diferentes modos de execução (privilégios), que constituem o Modo Dual de operação:
  - Modo supervisor, privilegiado, sistema, monitor ou kernel
  - Modo usuário

Modo Dual de Operação

- Arquitetura de processadores oferecem mecanismos para diferenciar dois modos de operação
  - Modo Supervisor (privilegiado)
    - Modo de execução sistema operacional (instruções privilegiadas)
    - Execução de todas as instruções do processador
  - Modo Usuário
    - Execução dos processos usuários
    - Somente um subconjunto de instruções do processador, registradores e portas de E/S estão disponíveis.
    - Caso o código em execução tentar executar uma instrução não disponível, será gerada uma exceção.
- Chaveamento de modos
  - Interrupção (modo usuário → modo supervisor)
  - Instrução (modo supervisor → modo usuário)

Proteção

- Proteção de memória
- Proteção de CPU

CHAMADAS AO SISTEMA

Chamadas ao Sistema

- Como uma aplicação poderá utilizar as rotinas oferecidas pelo núcleo do Sistema Operacional para acesso ao hardware?

- As Chamadas ao Sistema (*System Calls*) fornecem a interface entre um processo e o sistema operacional
  - Interface de programação para os serviços fornecidos pelo SO

- *System Calls*
  - Instruções em *assembly*
  - Podem ser escritas em uma linguagem de mais alto nível, como C ou C++
  - Acessada pelos programas de alto nível por meio da API – *Application Program Interface*
  - Programa de alto nível → pode gerar outras chamadas
  - *In-line*

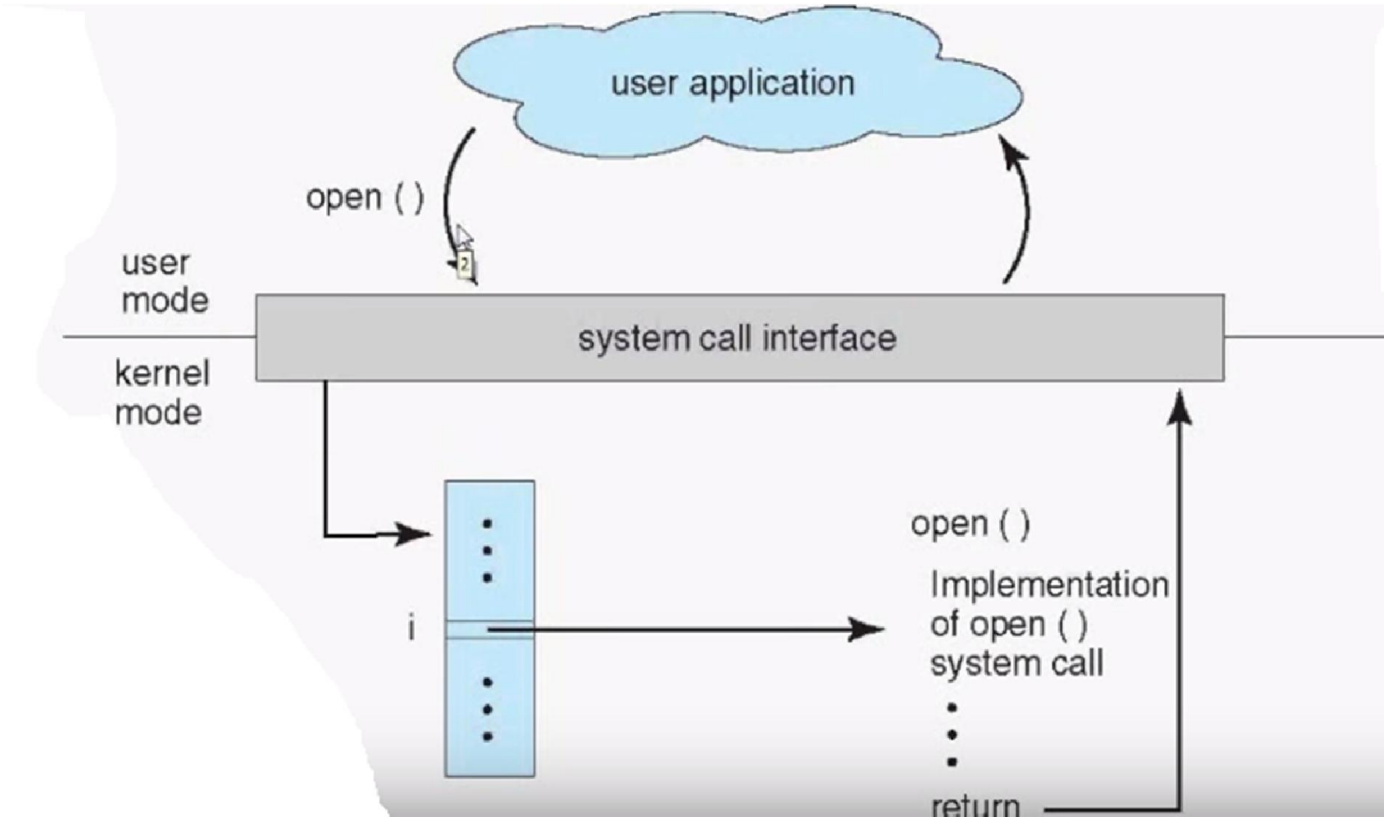
Chamadas ao Sistema

- É necessário o mecanismo de interrupções.
- API disponíveis aos compiladores pelos Sistemas Operacionais
- Exemplo API para manipulação de arquivos na Linguagem C
- Exemplo: Copiar 2 arquivos

Chamadas ao Sistema (2)

- Maioria das linguagens oferece interface mais simples
  - Detalhes da interface do sistema operacional são ocultos ao programador através do compilador e do pacote de suporte à execução.
  - Não é necessário conhecer como a *system call* é implementada.
- A API para as chamadas ao sistema invocam as chamadas ao Kernel do SO

Chamadas ao Sistema (3)



Fonte: SAURABH, S. Programming Interview: System Calls in operating system. Disponível em: https://www.youtube.com/watch?v=dmHaiRxqghU

Chamadas ao Sistema (3)

- 3 métodos para passar parâmetros às chamadas
  - Registradores (o mais simples)
  - Bloco ou tabela na memória e o endereço do bloco é passado ao registrador
  - Inseridos (*push*)na pilha do sistema pelo programa e lidos/removidos (*pop)* pelo SO
  - Não limitam a quantidade de parâmetros

Chamadas ao Sistema (3)

- 5 categorias de chamadas ao sistema
  - Controle de processo
  - Gerência de arquivos
  - Gerência de dispositivos
  - Manutenção de informações
  - Comunicação

Controle de processo

- Exemplos de chamadas
  - `end, abort, load, execute, create/terminate process, get/set process attributes, wait for time, wait/signal event, allocate/free memory`

- MS-DOS → Monotarefa
  - Interpretador de comandos chamado quando o computador é iniciado
  - Executa novo programa sem criar um processo
    - Carrega programa na memória, gravando sobre o si mesmo, para permitir maior quantidade possível de memória
    - Sistema define ponteiro de instruções para a primeira instrução
    - Programa executa até um erro acontecer ou parar
  - A pequena parte do interpretador, não sobreposta, recomeça a execução

Controle de processo (2)

- UNIX → Multitarefa
  - Interpretador de comandos é chamado quando um usuário efetua o logon no sistema (shell do usuário é executado)
    - Shell continua a execução enquanto outro programa é executado
  - Para iniciar um processo, o shell executa a chamada ao sistema *fork*
  - Programa é carregado na memória pela chamada *exec* (o programa é executado)

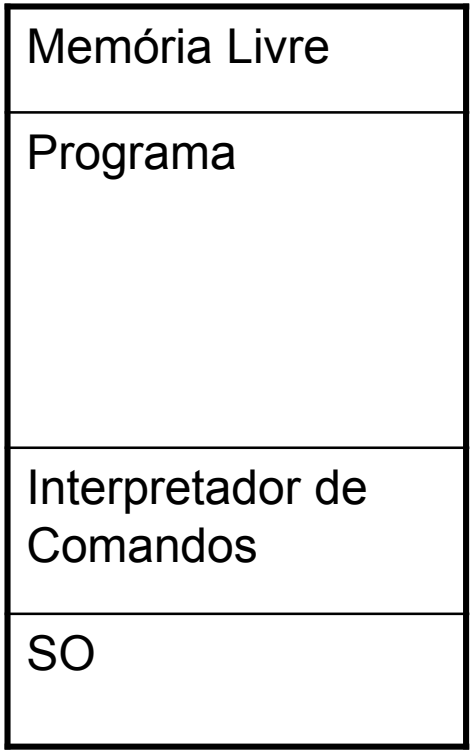
Controle de processo (3)

- Dependendo da forma em que o comando foi emitido, o shell espera o processo ser finalizado ou executa o processo em "segundo plano" (não pode receber entrada diretamente pelo teclado; o shell está livre para receber outras chamadas)

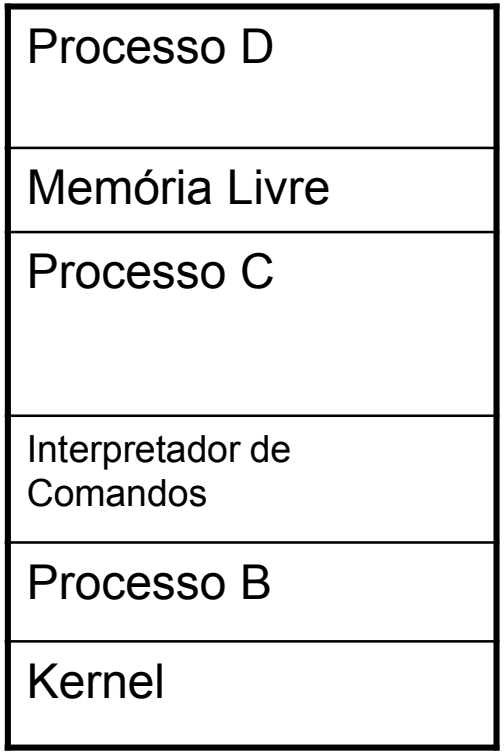
- Processo terminado
  - Executa chamada *exit* - passa ao programa que o criou código de status *zero* ou código de erro diferente de zero

Controle de processo (4)

- MS-DOS executando um programa



- UNIX executando vários programas



Gerência de arquivos

- Exemplos de chamadas
  - `create/delete file, open, close, read, write, reposition, get file attributes, set file attributes`
- Atributos de arquivos
  - Nome, tipo, códigos de proteção, tamanho, ...

Gerência de Dispositivos

- Exemplos de chamadas
  - `request/release device, read, write, reposition, get/set device attributes, logically attach devices`
- Programa em execução pode precisar de recursos adicionais para prosseguir
  - Memória, unidades de fita, acesso a arquivos...
- A similaridade entre dispositivos de I/O e arquivos é tão grande que muitos sistemas operacionais, como UNIX e MS-DOS, combinam os dois em uma estrutura de arquivo-dispositivo
  - Dispositivos de I/O são identificados por nomes de arquivos

Manutenção de Informações

- Refere-se às informações trocadas entre o programa de usuário e o sistema operacional
  - Chamadas ao sistema
    - Data e Hora atuais (`get/set date, time`)
    - Número de usuários atuais, versão do SO, memória livre...
    - Sistema operacional mantém informações sobre todos os processos (`get/set process/file/device attributes, por exemplo`)

Comunicação

- Exemplos de chamadas
  - `create, delete communication connection, send/receive messages, transfer status information, attach remote devices`
- Modelo de Trocas de Mensagens
  - Recurso de comunicação oferecido pelo SO
- Modelo de Memória Compartilhada
  - Os processos também são responsáveis por garantir a consistência da memória compartilhada

Exemplos de Chamadas ao Sistema

Gerência	Windows	Linux

Processos	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

Arquivos	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Dispositivos	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Manutenção de Informações	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Comunicação	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shmget()`
		`mmap()`

Estrutura do Sistema

- Sistema Operacional deve ser dividido em pequenos componentes e não ser um sistema monolítico
- Principais componentes:
  - **Núcleo**
  - **Drivers:** códigos específicos para acessar os dispositivos físicos. Ex.: discos rígidos IDE, SCSI, portas USB, placas de vídeo
  - **Códigos de inicialização:** reconhecer, testar e configurar os dispositivos instalados. Carregar o núcleo do SO e iniciar a execução.
  - **Programas utilitários:** funcionalidades complementares do SO

Kernel do SO

- É a parte do SO executada com interrupções desabilitadas e no modo privilegiado
- Nos trechos críticos não há interrupção
  - Por exemplo, no momento da inserção de um processo na fila de prontos
- O Kernel pode ser monolítico
  - Quanto todos os componentes do SO são executados com interrupções desabilitadas e no modo privilegiado
  - Toda e qualquer parte do SO é executada de maneira indivisível
    - Cada *trap* (chamada ao sistema) ou interrupção desabilita o sistema de interrupções
  - Desvantagens
    - Pode haver muita espera
    - Complexidade da estrutura de organização do kernel

Kernel do SO

- O Kernel é um pequeno monitor monolítico que recebe o controle quando ocorrem chamadas ao sistema ou interrupções
- Os serviços são implementados fora do kernel
  - Por processos específicos
  - Um processo para controlar o dispositivo = *driver* de dispositivo
- Mantém o descritor do processo e implementa funções para sincronização e comunicação
- Pode ser *microkernel* ou monitor monolítico

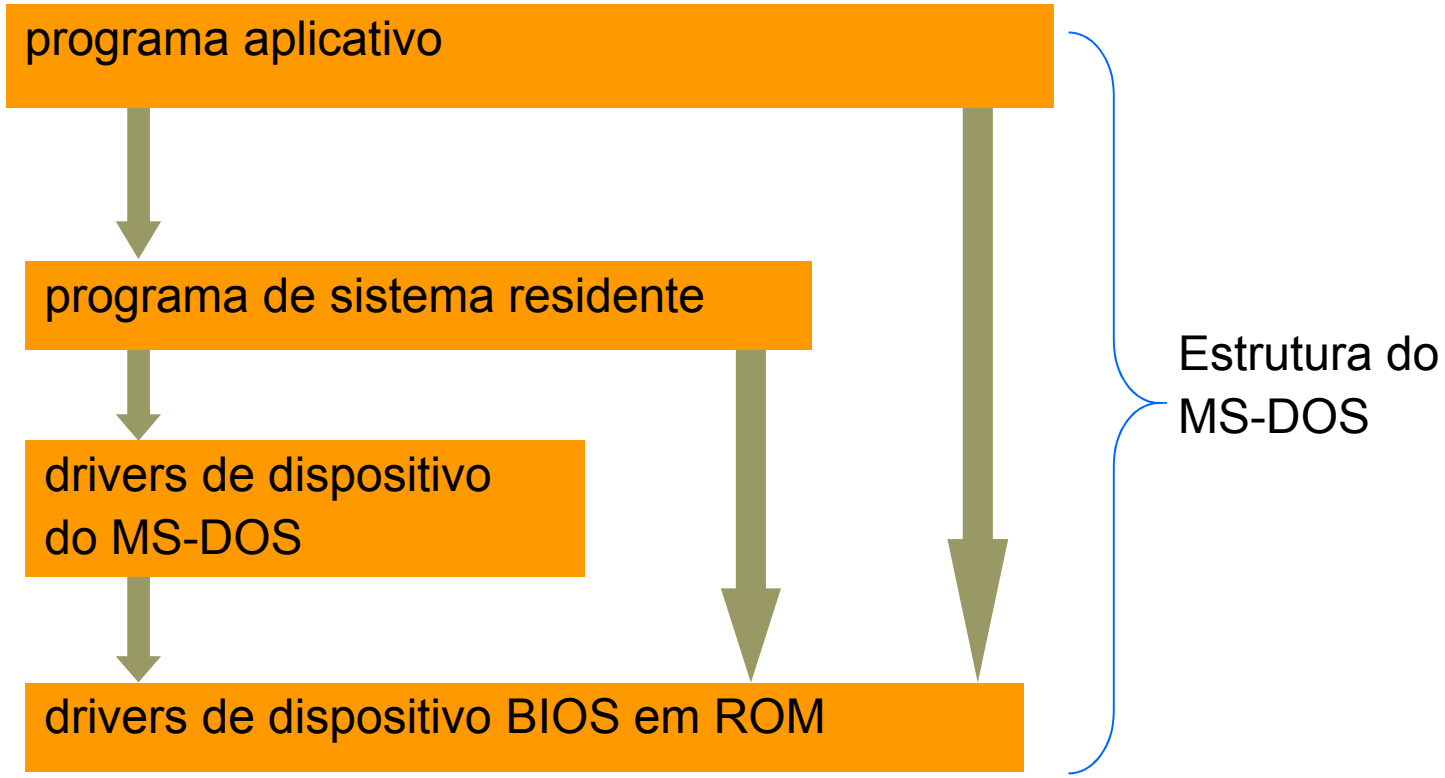
Estrutura do Sistema

- Como os componentes são combinados e interconectados
  - Estrutura simples
  - Camadas
  - Microkernels
  - Módulos

Estrutura Simples

- Sistemas sem estrutura bem definida
  - Iniciaram pequenos, simples e limitados → cresceram
  - Exemplo: MS-DOS
    - fornecer funcionalidade máxima no menor espaço possível (limitação hardware: Intel 8088 não fornece modo dual e proteção de hardware) → não foi dividido em módulos
    - Hardware base ficou acessível
    - Programas aplicativos podem acessar as rotinas básicas de I/O para escrever diretamente na tela e nas unidades de disco

Estrutura Simples (2)



Estrutura do Unix

- O UNIX originalmente foi limitado pela funcionalidade de hardware
- Dividido em duas partes
  - Kernel
    - Inclui Interfaces e drives de dispositivos (adicionados com a evolução)
    - No Unix, kernel é o que está abaixo da interface de chamadas ao sistema e acima do hardware
    - Fornece: sistema de arquivos, escalonamento de CPU, gerência de memória, funções do sistema operacional através de chamadas ao sistema
  - Programas de sistema

Estrutura do Unix (4)

usuários		
shells e comandos compiladores e interpretadores bibliotecas do sistema		
interface de chamada ao sistema para o kernel (API –*Application Programming Interface)*		
tratamento de sinal de terminal sistema de I/O de caracteres drivers de terminal	sistema de arquivos swapping sistema de I/O de bloco drivers de disco e fita	escalonamento de CPU substituição de página Paginação memória virtual
interface do kernel com hardware		
controladora de terminais Terminais	controladoras de dispositivo discos e fitas	controladoras de memória memória física

Camadas

- Forma de modularização de um sistema
- Uma camada de sistema operacional
  - é uma implementação de um objeto abstrato
  - Consiste em estruturas de dados e rotinas que podem ser chamadas por camadas superiores
- As camadas são construídas de forma que somente utilizem os serviços das camadas de nível mais baixo
  - Simplifica a verificação e depuração do sistema
- Menos eficientes devido à hierarquia de camadas; Permite controle maior do hardware, em relação à abordagem simples

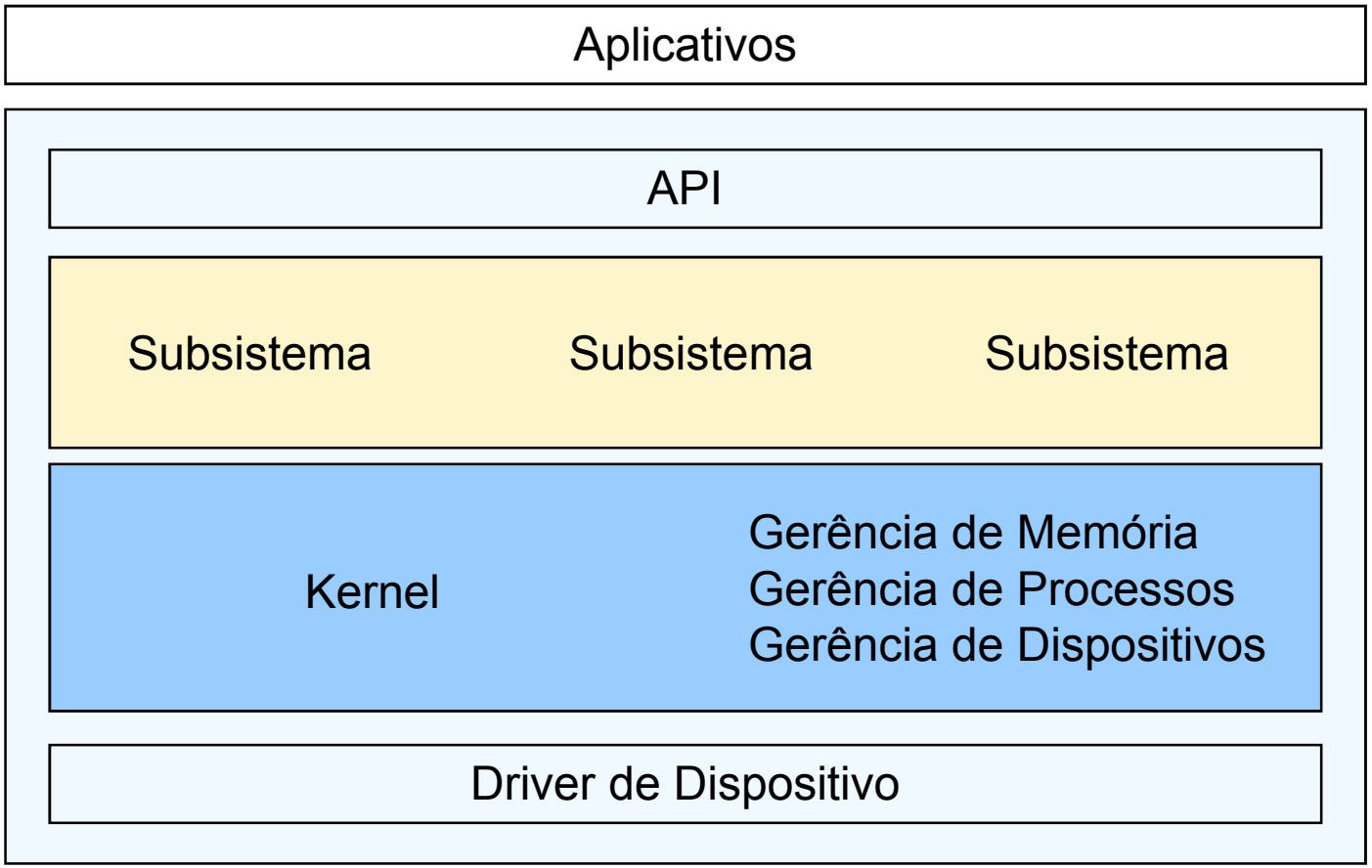
Camadas (2)

- A primeira camada pode ser depurada sem preocupação com o resto do sistema, porque utiliza somente o hardware básico para implementar suas funções
- E assim sucessivamente!
- Dificuldade: definição adequada das camadas
- Tendem a ser menos eficientes que outros tipos → em cada camada, os parâmetros podem ser modificados, os dados precisam ser transferidos... custos são acrescentados → maior tempo na chamada ao sistema

Camadas (3)

- OS/2
  - Acrescentou camadas, operações multitarefas e em modo dual
  - Não permite ao usuário acesso aos recursos de baixo nível
  - Sistema operacional tem mais controle sobre o hardware e programas em execução
- MULTICS
- Windows NT (parcialmente)
  - Implementa camada inferior de abstração do hardware (*HAL – Hardware Abstraction Layer*)

Camadas (4)


Aplicativos
API
Subsistema
Subsistema
Subsistema
Kernel
Gerência de Memória
Gerência de Processos
Gerência de Dispositivos
Driver de Dispositivo

Microkernel ou Micronúcleo

- Expansão do Unix, o kernel tornou-se grande e difícil de gerenciar

- 1980 – Sistema Operacional Mach – Carnegie Mellon University
  - Abordagem Microkernel
    - Implementa somente os componentes essenciais do SO no Kernel; os demais são implementados como programas de sistema e de usuário
    - Retirou-se do kernel o código de “alto nível”
    - Os componentes do SO comunicam-se por meio de trocas de mensagens
    - Kernel menor
    - Facilidade de manutenção

Microkernel ou Micronúcleo

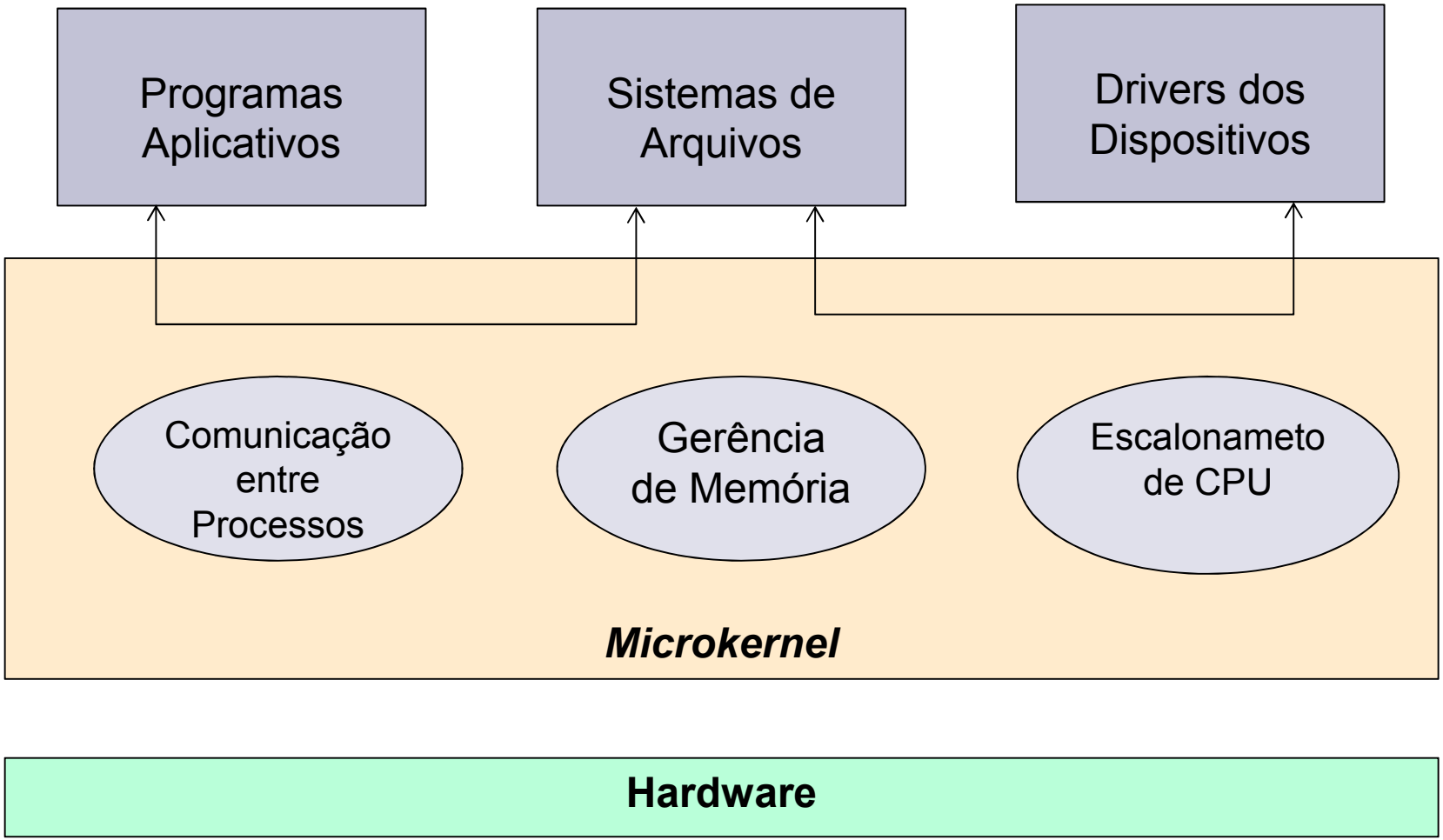
- Fornecem
  - Gerência mínima de memória e processos
  - Um recurso de comunicação entre programa do usuário e serviços que estão em execução também no espaço de usuário

- Proporcionam
  - Facilidade de expansão do sistema → novos serviços são adicionados ao espaço do usuário
  - Modificações facilitadas no kernel
  - Maior segurança e confiabilidade em caso de falhas

Microkernel ou Micronúcleo

- Possibilitar a comunicação entre o programa cliente e os diversos serviços disponibilizados
  - Trocas de mensagens
  - O programa cliente e o serviço não interagem diretamente e sim via microkernel
- Sistemas baseados no *kernel* do *Mach*
  - UNIX Digital
  - Apple MacOS X

Microkernel ou Micronúcleo



Módulos

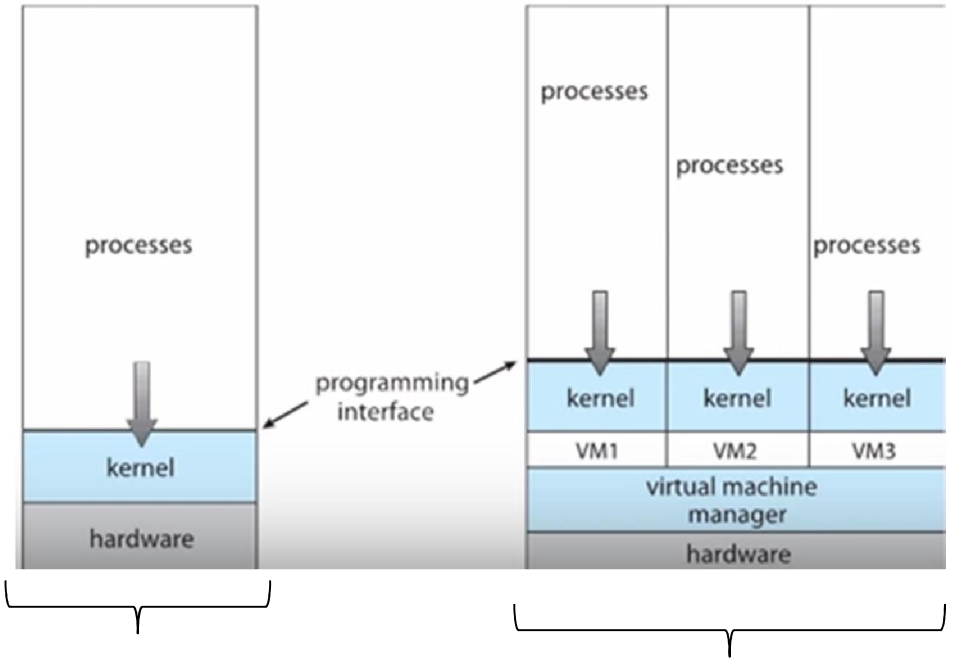
- POO → Kernel modular
- Há um kernel básico e módulos carregáveis dinamicamente
- A interface entre os módulos é claramente definida
- Solaris, Linux, Mac OS X
- Flexibilidade

Máquina Virtual

- VM da IBM
- O SO possibilita que um processo “possua” seu próprio processador com sua própria memória
  - Técnicas de escalonamento e memória virtual
- Cada processo recebe uma cópia do computador básico
- O computador físico compartilha seus recursos para que isto ocorra
  - Escalonamento de CPU pode criar a aparência que cada usuário possui seu próprio processador
- Atualmente, usada para portabilidade dos sistemas

Máquina Virtual

- Vantagens:
  - Aos usuários e desenvolvedores
  - Isomamento dos recursos da Máquina Virtual
  - Testes
- Desvantagens:
  - Difícil de implementar



Máquina não Virtual

Máquina Virtual

Considerações Finais

- Abordagens
  - Camadas
  - Microkernel
  - Módulos
  - Máquina Virtual

  não são mutuamente exclusivas.

Bibliografia

- SILBERSCHATZ, Abraham; GALVIN, Peter; GAGNE, Greg. *Sistemas operacionais:* com Java. Rio de Janeiro: Campus, 2008.

- DEITEL, Harvey M; DEITEL, Paul J; CHOFFNES, David R. *Sistemas operacionais*. São Paulo: Pearson Pretince Hall, 2005.