

INF 1316: Sistemas Operacionais

Markus Endler
endler@inf.puc-rio.br

Pedro Nogueira Barella
pedronoba25@gmail.com

<http://www.inf.puc-rio.br/~endler/courses/inf1316/>

Conceitos Fundamentais

O que é um Sistema Operacional?

É um software que:

- age como intermediário entre o usuário e o hardware
- fornece um ambiente onde o usuário pode criar e executar programas
- garante a utilização eficiente dos recursos de hardware (CPU, Memória RAM, Disco, SSD, Caches L1-L3, periféricos, etc.)
- protege o próprio sistema operacional e os programas de interferências mútuas.
- Cria uma **máquina virtual** que abstrai dos detalhes de acesso/controle aos recursos de hardware (evitando que o programador/usuário tenha que se preocupar com a heterogeneidade da arquitetura)

Bibliotecas e Sistema Operacional

Existem bibliotecas específicas do SO e específicas da linguagem e ambiente de programação

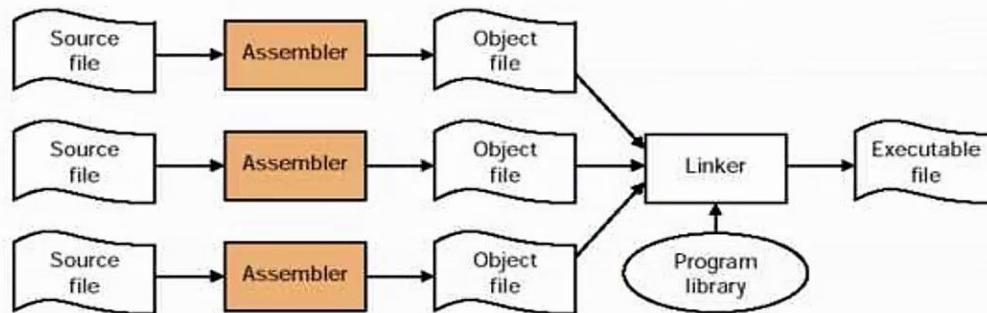
Exemplos: específicas da família de SO, como unistd.h em sistemas POSIX,
específicas do compilador,
específicos da linguagem (por exemplo, boost é específico para C++)

O que exatamente as bibliotecas contêm?

- tem de tudo: qq coisa que seja distribuída (em formato fonte ou binário) com os principais componentes (kernel, sistema de janelas, etc.) do SO no qual o seu Software será executado,
- header-only: são integradas aos arquivos fonte na compilação/linkedição, outras em formato binário.
- bibliotecas carregadas e tempo de execução (shared objects no POSIX, Dynamically Linked Libraries (DLLs) no Windows).

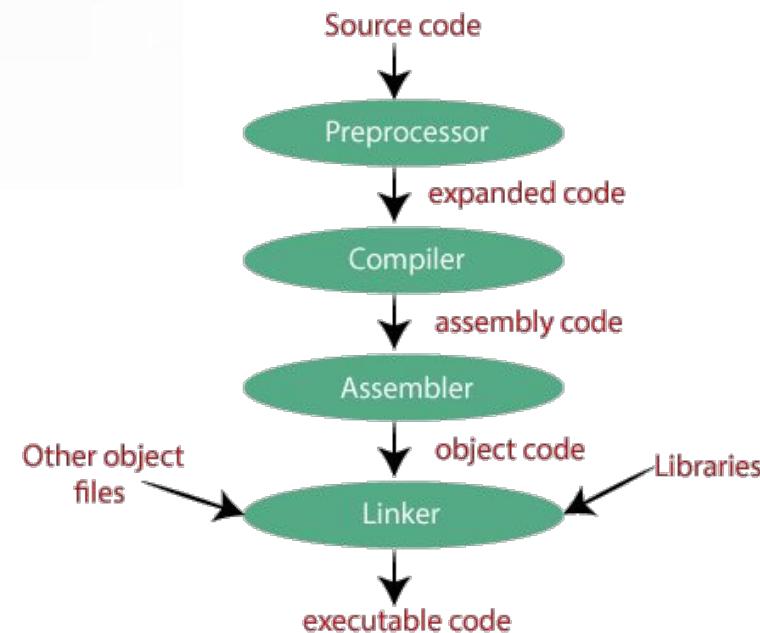
O processo de compilação e ligação - o caso da linguagem C

Compile, assemble, and link to executable
`gcc test.c` produces `test.exe`

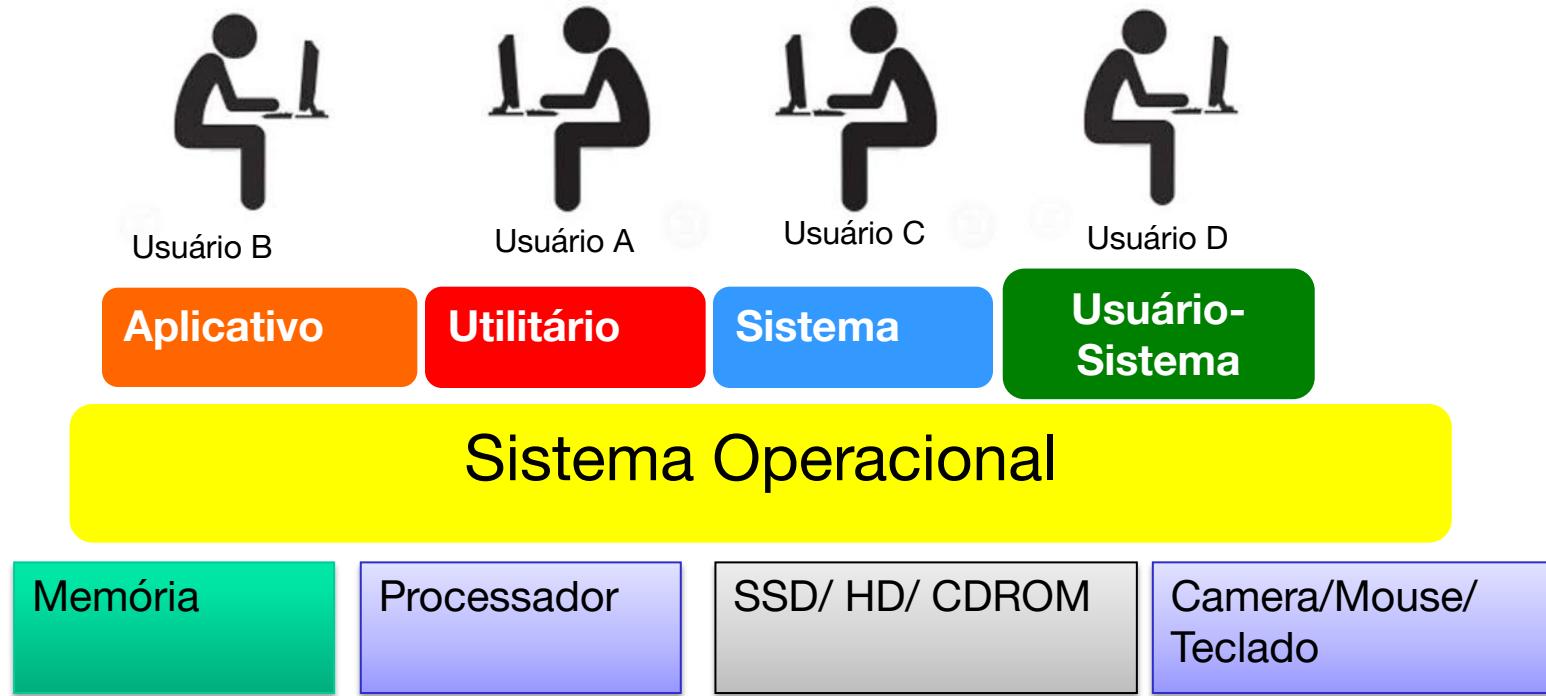


```
gcc test.c -o test.o
gcc test.c -l libxyz -o main
gcc -g test.c -o main
gdb
```

How to Debug C Program using gdb in 6 steps
<https://u.osu.edu/cstutorials/2018/09/28/how-to-debug-c-program-using-gdb-in-6-simple-steps/>



A função do Sistema Operacional



Exemplos:

Aplicativos: Planilha, processador de texto, jogo, ...

Utilitários: Compilador, linker, depurador, editor de texto, SGBD, browser Web, ...

Sistema de Informação: ERP, CRM, reserva de passagens aéreas, ...

Interface Usuário-sistema: interface por linha de comando (shell) ou Sistema de Janelas

Recursos de Hardware: CPU, memória RAM, discos, dispositivos de E/S, ...

Principais Responsabilidades do S.O.

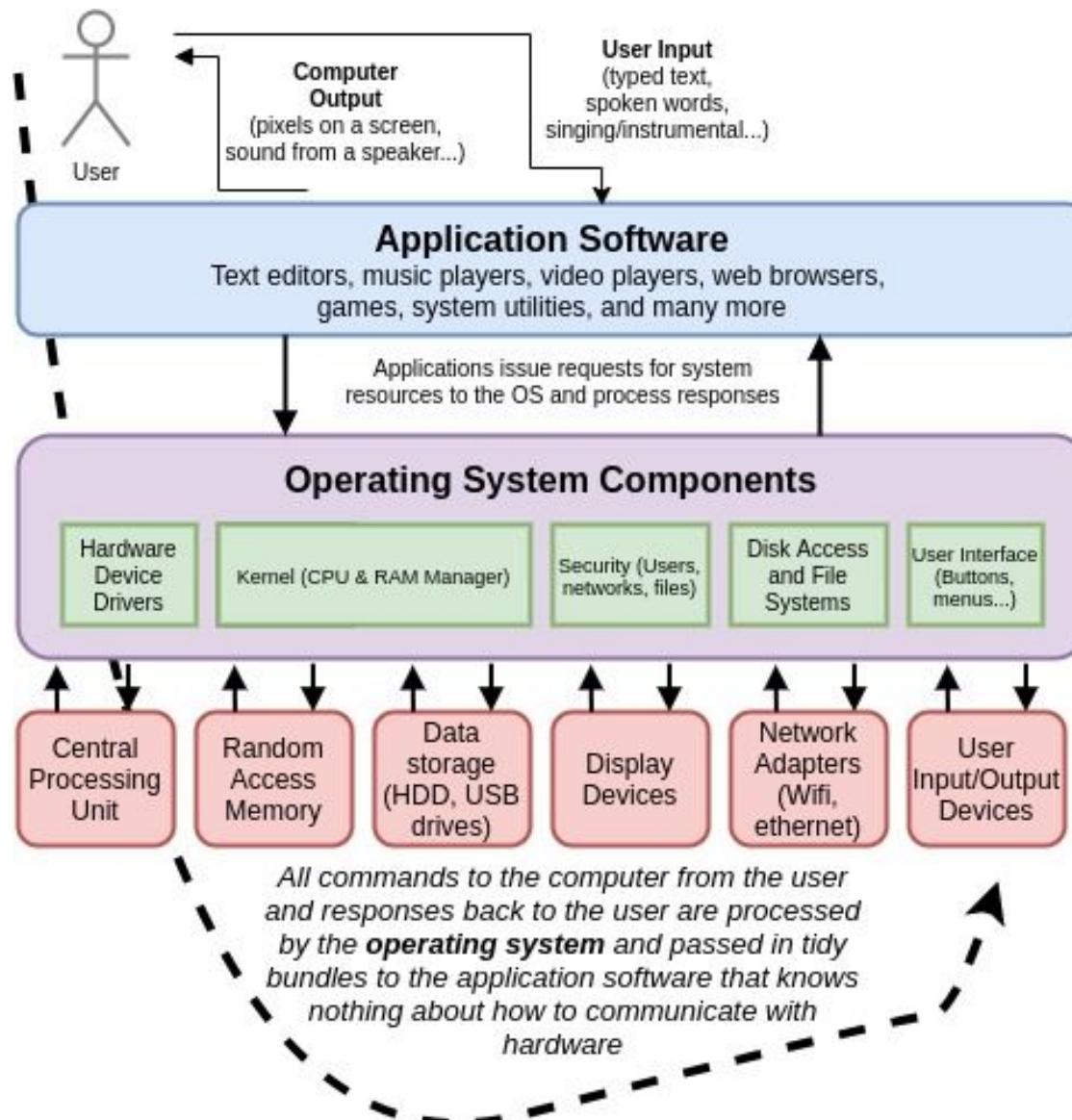
O Sistema Operacional fornece as seguintes funções para o usuário:

1. Autenticação e acesso ao sistema (segurança de acesso)
2. Gerenciar diretórios, arquivos
3. Executar programas (com ligação das bibliotecas)
4. Acessar e manipular arquivos e otimizar o seu mapeamento para o SSD ou HD
5. Detectar e notificar erros de execução
6. Usar dispositivos periféricos (impressora, scanner, driver de DVD, pen-drive, etc.)
7. Contabilizar a utilização do sistema
8. Garantia segurança e isolamento entre os programas

O que é um Sistema Operacional?

- É uma máquina estendida (visão *top-down*)
 - Implementa abstrações que escondem (do programador e do usuário final) os detalhes de como acessar e controlar os recursos de hardware;
 - Apresenta ao programador uma *máquina virtual*, que é mais fácil de usar (API simples baseada em conceitos independentes do hardware) e genérica (sem expor a diversidade dos recursos/dispositivos)
- É um gerenciador de recursos (visão *bottom-up*)
 - Garante o **compartilhamento** seguro de recursos pelas várias tarefas executando **concorrentemente** no sistema;
 - Visa **maximizar a utilização** dos recursos (CPU, Memória, Disco, etc.)

Papel do Sistema Operacional



system calls:
máquina estendida

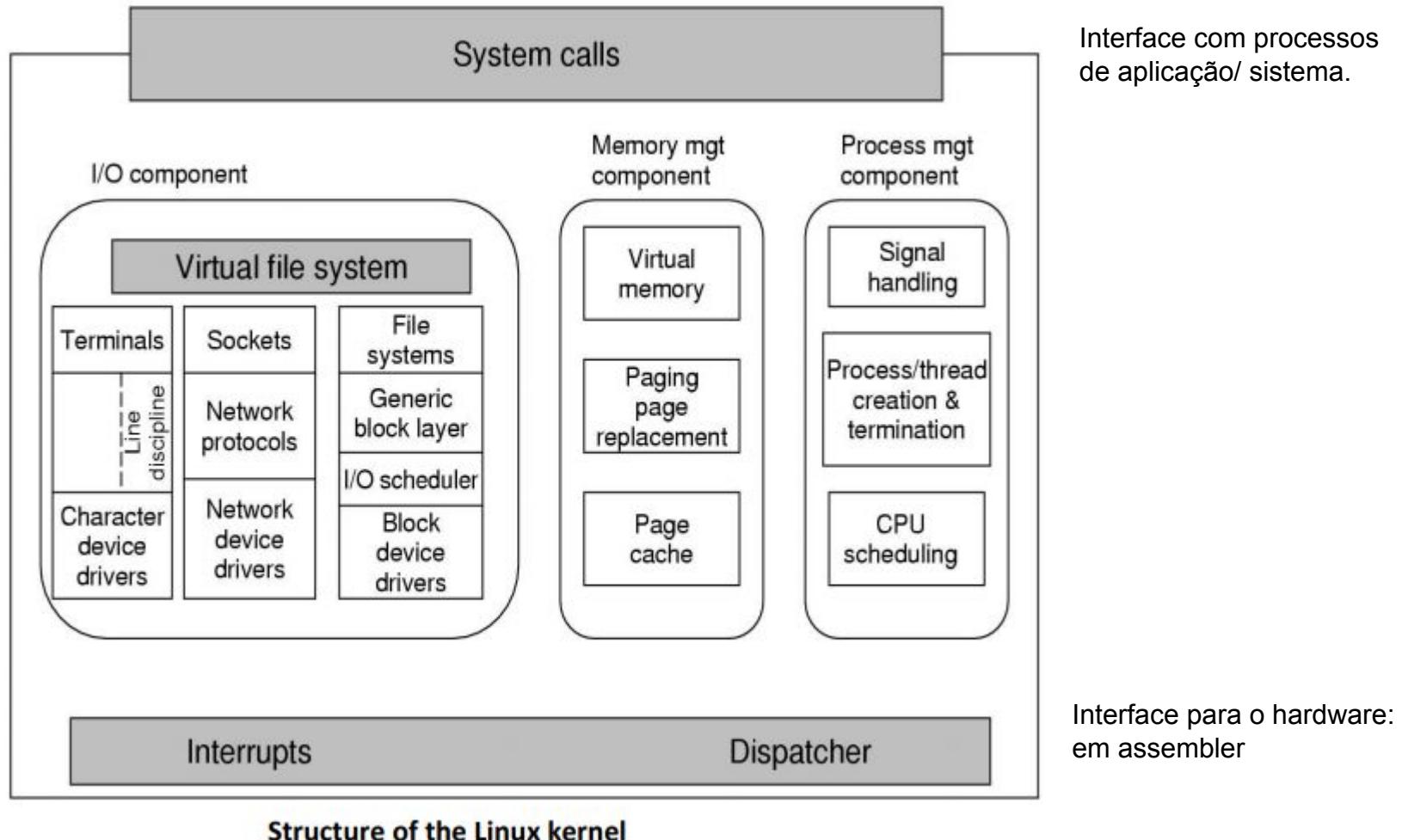
gerenciador
de recursos

Compartilhamento de Recursos

Recursos = {CPU (1+ core), memória principal (RAM), caches, SSD, HD, periféricos de E/S, interface de rede, etc} ...
são gerenciados pelo **núcleo (kernel)** do S.O.

- Compartilhar recursos significa que diferentes usuários ou processos usam recursos de **forma concorrente** (ao mesmo tempo, intercalando)
- Porque sempre há mais de um programa/processo executando concurrentemente.
- Os recursos são limitados, e precisam ser controlados e administrados de forma a evitar conflitos e interferências (**isolamento**)

Estrutura do kernel Linux

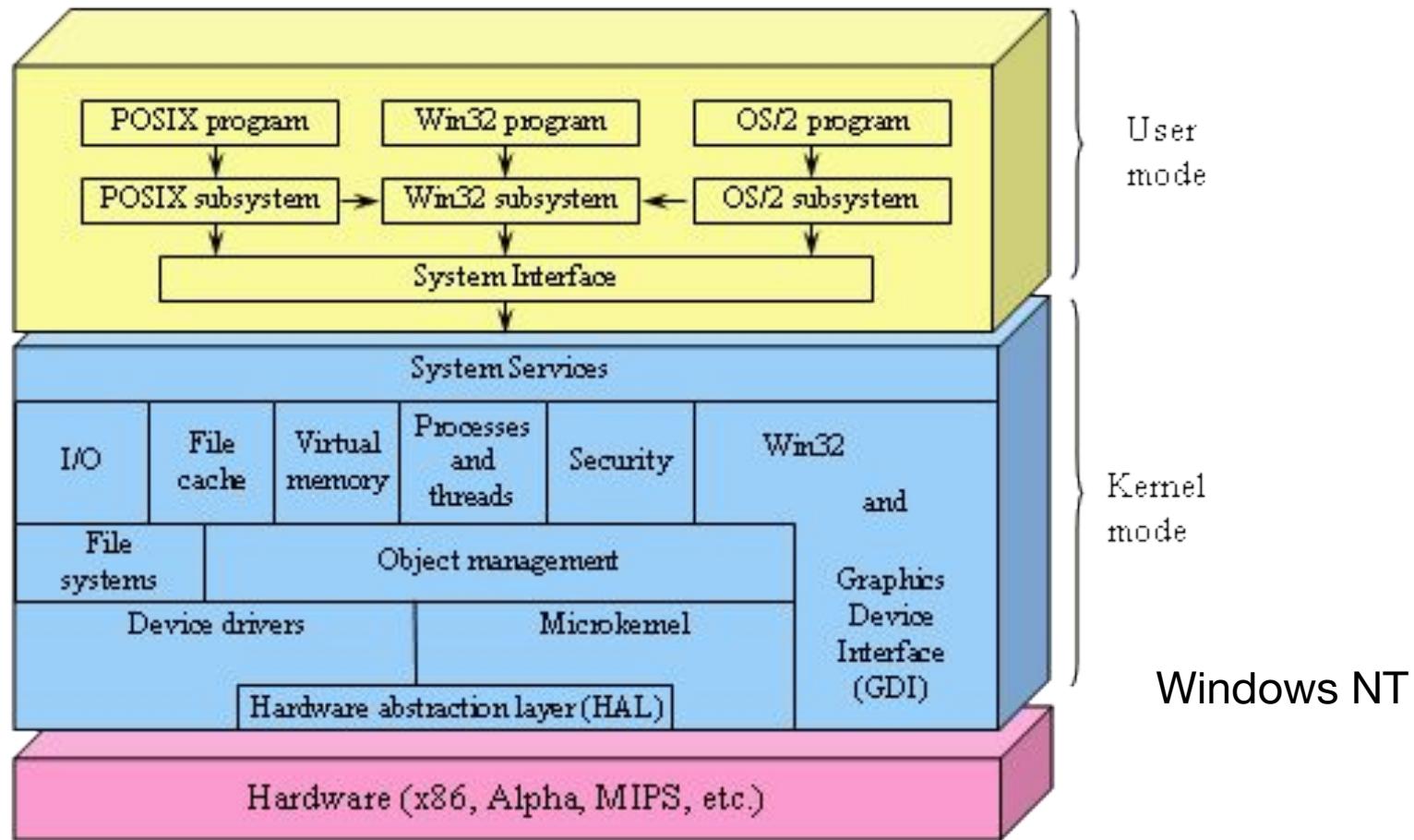


Fonte: https://examradar.com/linux-architecture-linux-kernel-structure/#google_vignette

S.O. visto como gerente de recursos

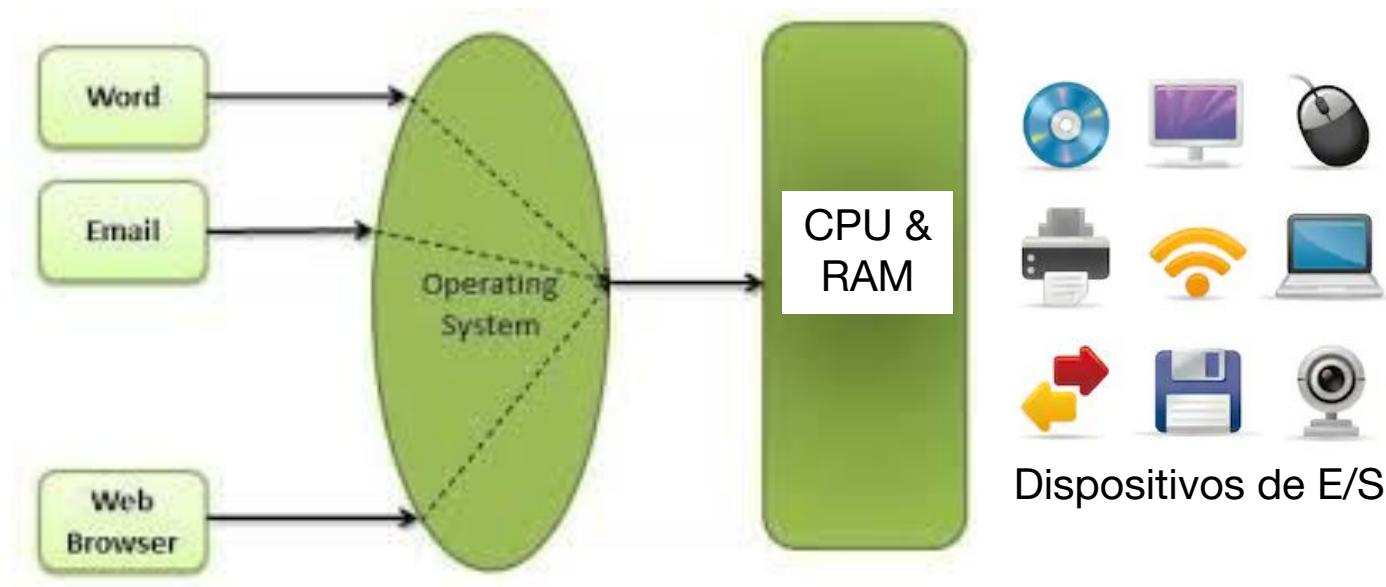
- Fornecer um esquema de alocação dos recursos (processador, memórias, canais de I/O, etc) entre os processos concorrentes,
- Estabelecer critérios de uso dos recursos e ordem de acesso aos mesmos,
- Impedir a violação de espaço de memória de processos concorrentes e tentativas de acesso simultâneo a um mesmo recurso → controle de acesso, segurança e locking
- Garantir a execução eficiente dos programas transferindo dados e código entre as várias memórias (caches, RAM, HD)

Sistema de várias componentes e camadas de software



Concorrência

- Nos **sistemas multi-programados (multi-tarefa)**, existem vários programas que executam concurrentemente (ao mesmo tempo), permitindo que:
 - a CPU e os dispositivos de Entrada/Saida (E/S) possam operar em paralelo;
 - utilização de CPU e RAM é maximizada, e sistema parece ser multi-core



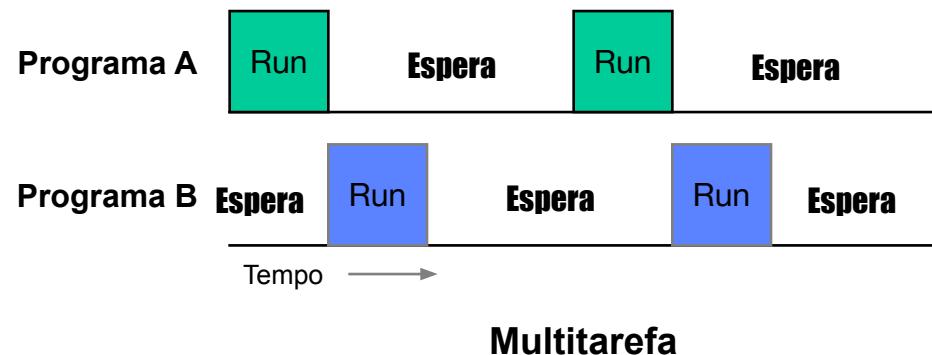
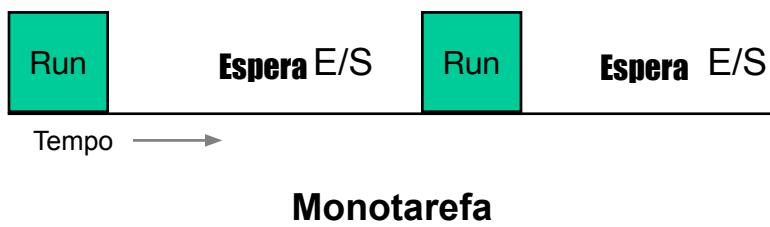
Concorrência

- Cada processo (=programa em execução) alterna **períodos de processamento** (uso efetivo da CPU) com **períodos de espera por entrada/saída (E/S)**.
- Seria um desperdício deixar a CPU esperando pelo término de cada E/S com um dispositivo => usa-se a CPU para outra tarefa
- Para tal, é importante que o SO possa iniciar uma E/S e prosseguir com a execução de outra tarefa.
- núcleo do SO **reage a interrupções** vindas dos dispositivos de E/S que notificam o término da operação
- Assim, em sistemas multiprogramados **aumenta-se a utilização da CPU**.

Classificação de Sistemas Operacionais

Monotarefa x Multitarefa

- **Sistema Mono-tarefa:** Executa apenas uma tarefa (programa) por vez. Ex: MS-DOS
- **Sistema Multi-tarefa:** Executa e gerencia várias tarefas concorrentes. Obs: a grande maioria dos SOs modernos são multitarefa



Obs: Os termos *Multitarefa* e *Multiprogramação* são sinônimos!

Sistemas Operacionais Classificação

Mono-usuário x Multi-usuário

- **Sistema Monousuário:** Permite o login de apenas um usuário – não permite que mais de um usuário esteja “logado” simultaneamente
 - Ex: Windows 98, iOS, Android, Arduino, etc.
- **Sistema Multiusuário:** Admite e gerencia vários usuários – permite que mais de um usuário esteja “logado” no sistema simultaneamente.
 - Ex: Linux, Windows 2000, VMS

⇒ Um Sistema multi-usuário requer **mecanismos de segurança** (controle de acesso aos arquivos, e proteção para áreas de memória das tarefas em execução)

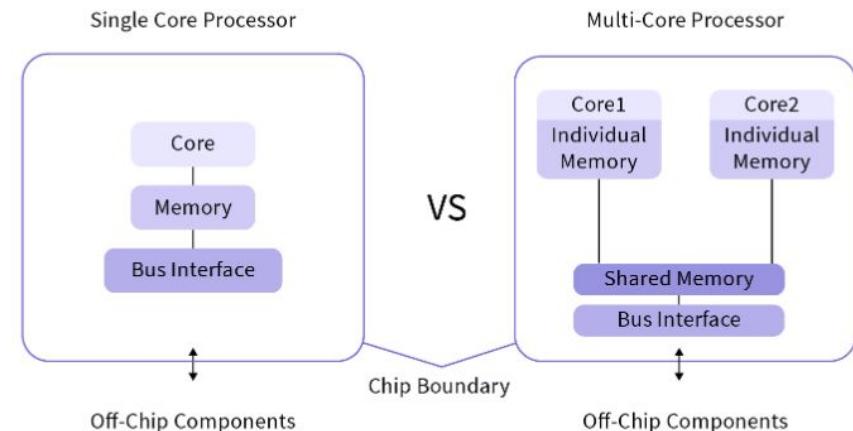
Classificação de Sistemas Operacionais

■ Sistema Monoprocessado

- Somente utiliza uma CPU
- Pode ser multi-tarefa ou mono-tarefa
- Ex: Windows 98

■ Sistema Multi-processado

- Reconhece mais de uma CPU (multi-core, arquiteturas paralelas ou em rede)
- Permite execução realmente paralela de tarefas
- Como a memória é comum, os cores precisam se sincronizar no acesso a memória
- Ex: Windows 2000/NT/XP, Linux, MACH



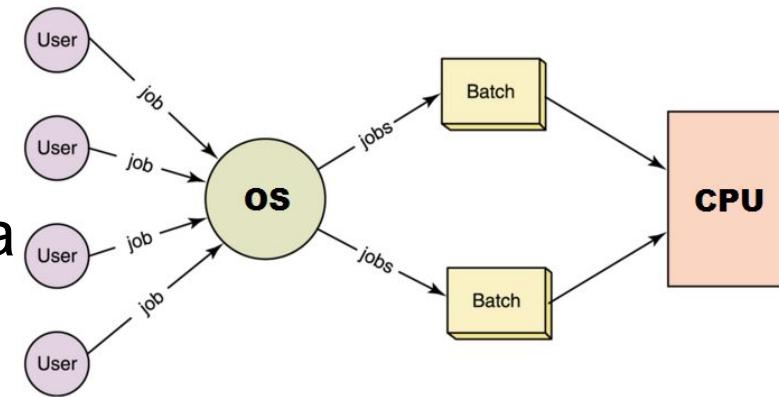
Sistemas Operacionais

Classificação

■ Sistemas Batch

Os programas são processados em lote, um de cada vez, não havendo interação com o usuário.

Ex: processamento numérico, simulações, aplicações científicas



■ Sistemas Time-Sharing

As tarefas compartilham o tempo de CPU.

Cada tarefa recebe fatias de tempo. Isso não é perceptível para os usuários, que obtém respostas quase imediatas.

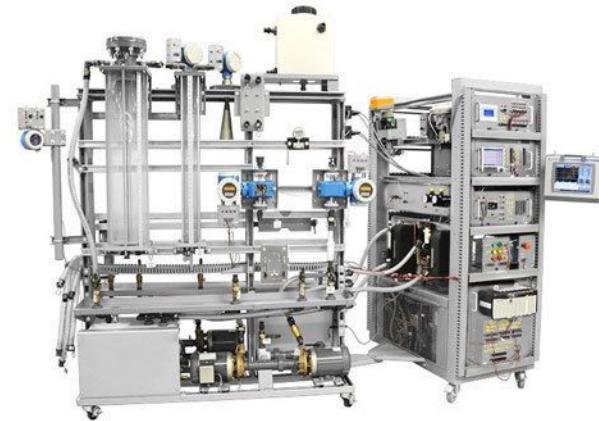


Sistemas Operacionais

Classificação

- **Sistemas de Tempo Real**

RTOS controlam processos do mundo físico (reações químicas, manufatura, veículos, etc.). A saída (output) só está correta se acontecer até um tempo pré-estabelecido.



- **Sistemas Embarcados**

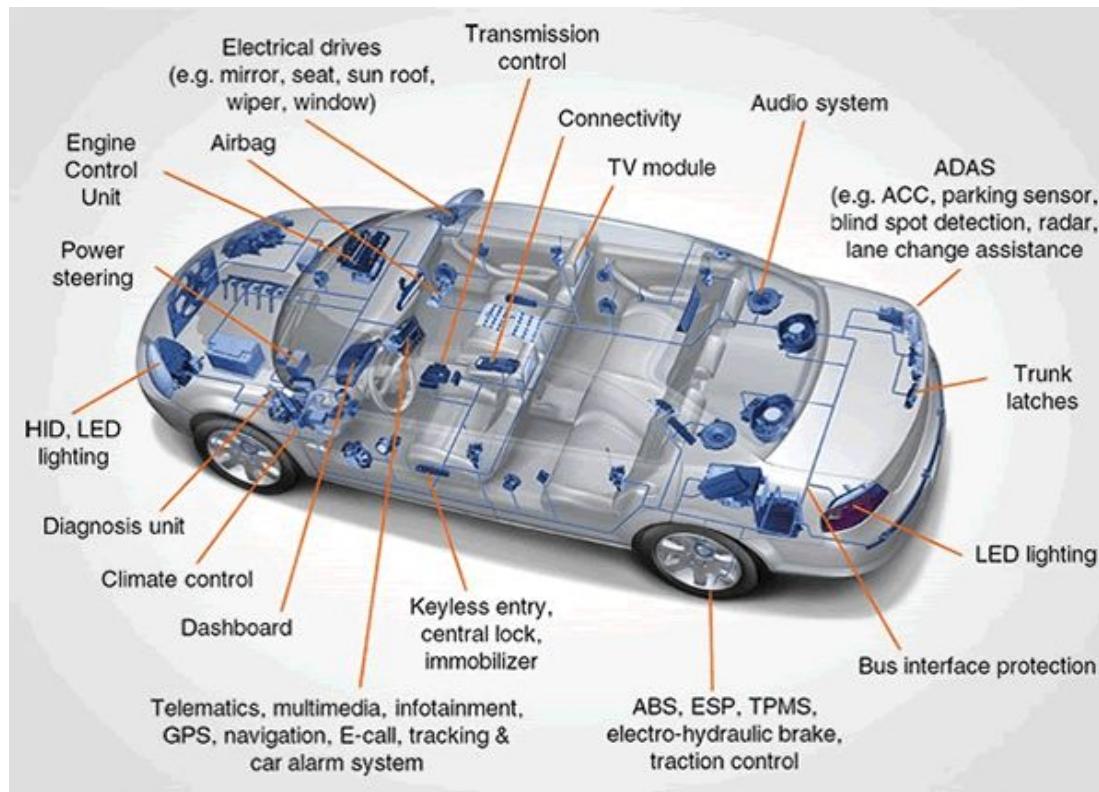
Sistemas embutidos em aparelhos dispositivos como micro-ondas, som, consoles de games, carros, etc.

Possui apenas as funcionalidades para gerenciar os recursos do equipamento



Quais são as características de um Real-Time OS?

Características de um sistema embarcado?



Cada sistema operacional com seu objetivo...

- Aumentar a praticidade/facilidade de uso;
- Obter o máximo de desempenho (processamento de alto desempenho)
- Minimizar o tempo de resposta para todos usuários/programas
- Operar em sistemas com poucos recursos (sistemas embarcados, aparelhos hospitalares, etc);
- Garantir tempos de resposta previsíveis (controle e automação, sistemas de tempo real);
- Ser tolerante à falhas através de dados e funções replicadas (data centers); ...

Classifique os seguinte SOs

1. Android/ iOS
2. Linux
3. WatchOS/ WearOS
4. HarmonyOS
5. RTOS
6. QNX
7. TinyOS
8. Android Automotive OS
9. Hipervisor
10. CloudOS (Cloud Management Platforms)

Vários tipos de Sistemas

=> Vários tipos de SO



Mica Mote



Personal Comp.



Tablets



Mainframe Server



Process Control Board



Game Console



MP3 Player



Cell / Smart Phone

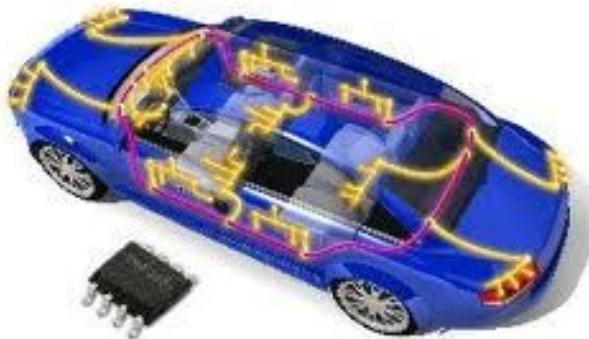


iPod Touch/
iPhone



Cluster/ Server farm para Cloud

Tipos de Sistema: Sistemas Operacionais em Rede



Automotive Vehicle Network



Network Routers



Wireless Routers



Redes de Sensores sem fio



Robôs Autonômicos (em depósito)

Tipos de Sistemas: Sistemas Emergentes



Car Computer



Wearable Computer



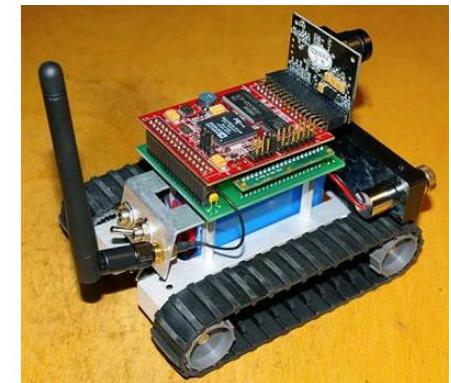
Wearable Comp.



Data Centers



Table Computing



Embedded Computing

Um S.O. para cada tipo de hardware

Exemplos:

- PCs/Notebooks: Linux, variantes do UNIX, Windows XP/NT/7, Vista, Mac OS X, ...
- Computação em nuvem: Amazon EC, Xen, ...
- Smartphones/Tablets: iOS, PalmOS, Android, Windows Phone, BlackBerry, AlyunOS...
- Embarcados: Android, Inferno, Maemo, etc.
- Sistemas de Tempo Real: QNX, RT-UNIX, RTOS...
- Redes de Sensores (MicaMotes): TinyOS, Contiki, ...
- Clusters/Sist. distribuídos: Mach, Plan9, Amoeba, Beowulf, Chorus ...
- WebOS: ChromeOS, JoliOS, DesktopTwo, EyeOS, ...

Funções comuns

Apesar das diferenças, todos os SOs provêm as seguintes funções:

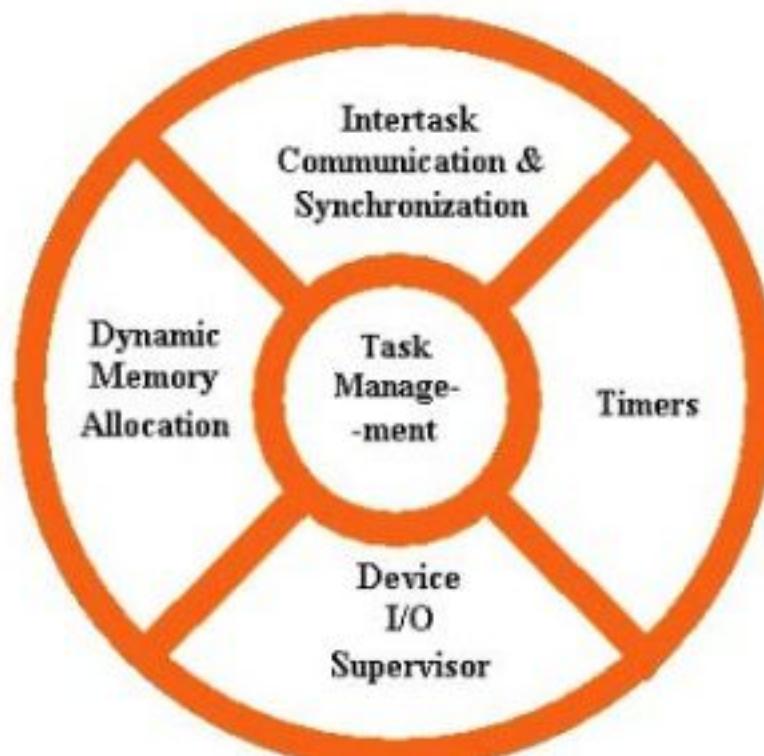
- Gerência e escalonamento de tarefas (processos e threads)
- Comunicação/sincronização entre processos/threads
- Sistema de Arquivos
- Gerenciamento de memória RAM
- Subsistema de Entrada/Saida
- Drivers e gerenciadores de dispositivos periféricos: teclado, monitor, mouse, rede, HDs, CD/DVD, etc.

Funções essenciais

Apesar das diferenças, todos os SOs provêm as seguintes funções básicas:

1. tratamento de **interrupções de hardware**;
2. criação e **gerenciamento de processos/tarefas**;
3. **escalonamento** e controle dos processos;
4. **sincronização e comunicação** entre processos;
5. gerência de **memória principal (RAM)**;
6. gerência do **sistema de arquivos**;
7. Operações de **entrada e saída** (levando em conta taxas de E/S dados diferentes)
8. contabilização e **segurança** do sistema.
9. verificação de **erros e tolerância a falhas**

Funções essenciais do kernel de S.O. de tempo real e convencionais



1. Tarefas são escalonadas de acordo com suas prioridade
2. Tarefa em execução tem todos pre-alocados todos os recursos que precisa

Elementos Típicos de um S.O.

Programas utilitários:

- Auxiliam o usuário/desenvolvedor em tarefa específica
- Exemplos: compiladores, loader, linker, editores, instalação de programas (**install**), depuradores, interpretadores de comando (**shell**), gerenciador da interface gráfica, clientes de email, etc.

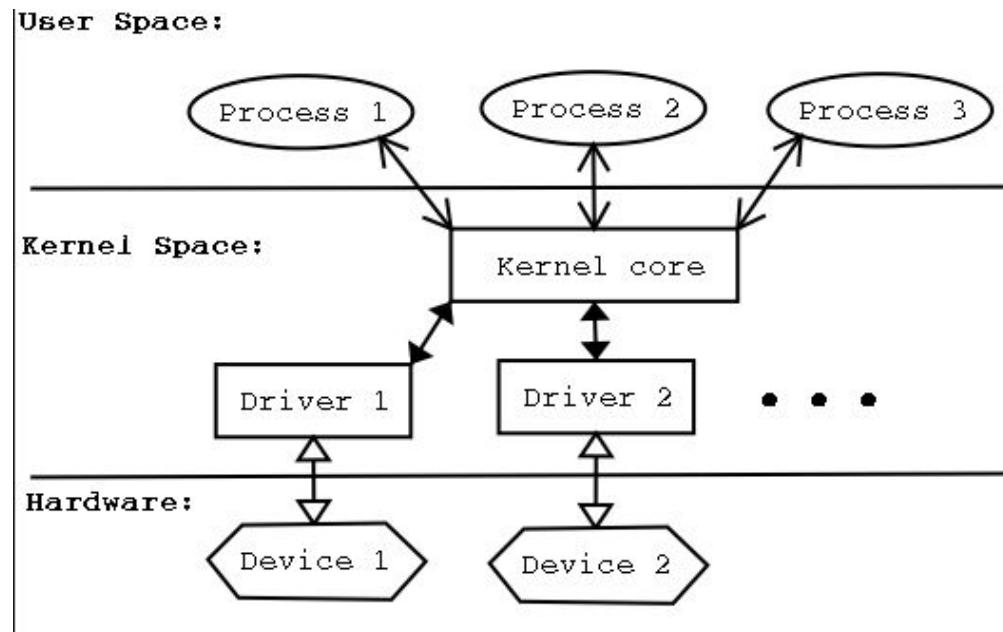
Programas/Servidores de sistema:

- São ativados *automaticamente* após o boot, e depois periodicamente
- São indispensáveis para a operação do sistema
- Para espera por um novo login (**getty**), conectam os módulos /libs de um programa (**ld**); esperam por conexões http (**Apache httpd**), fazem o controle de processos (**kill**), colhem estatísticas sobre a uso de recursos. etc.

Componentes Típicos de um S.O.

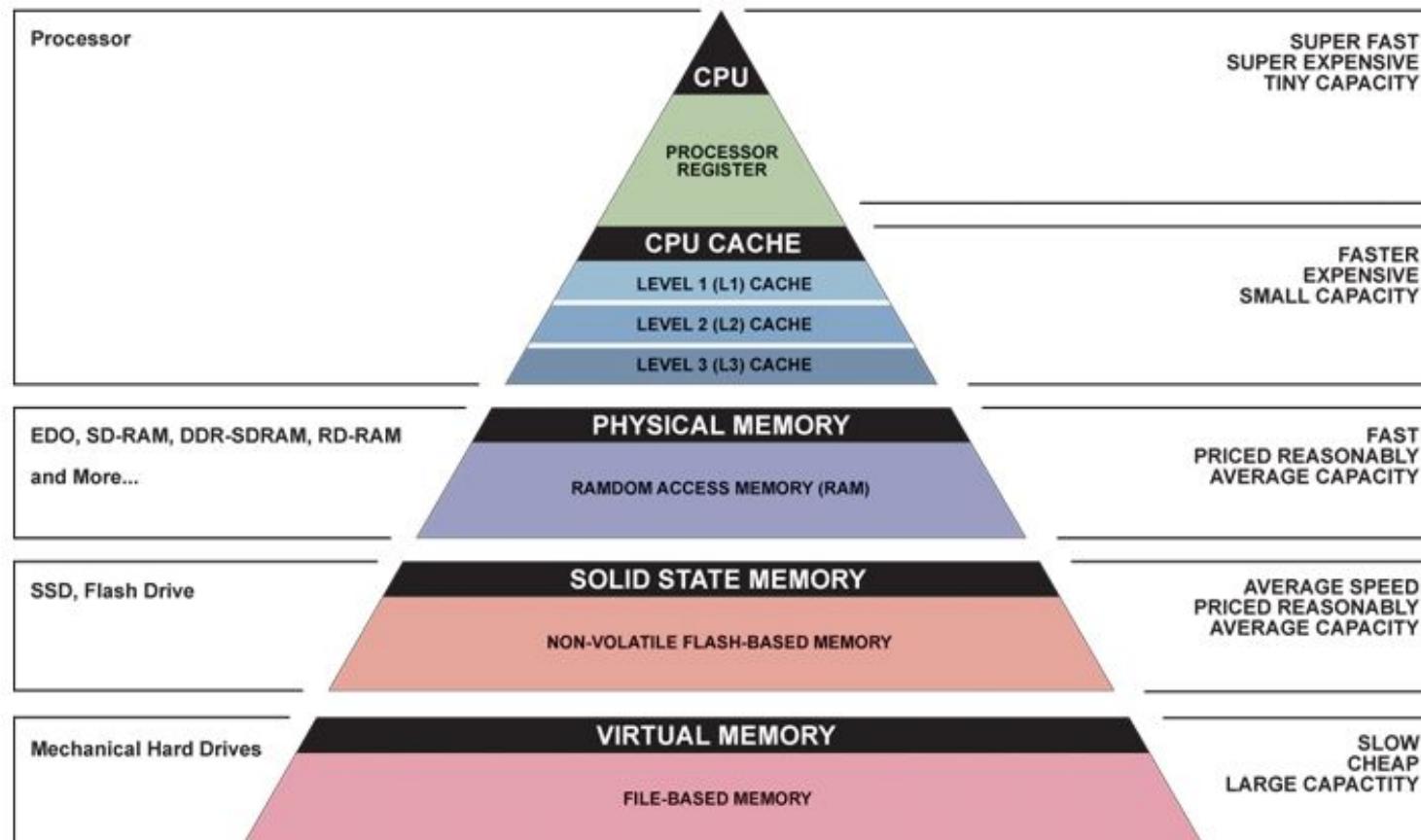
- Núcleo (kernel) e drivers:

- Executam as funções mais básicas para alocação e gerenciamento seguro dos recursos
- Acessam diretamente o hardware e todas as instruções da máquina (modo kernel)
- Drivers: fazem o controle específico de uma (classe de) dispositivos de Entrada e Saída



Hierarquia de Memória

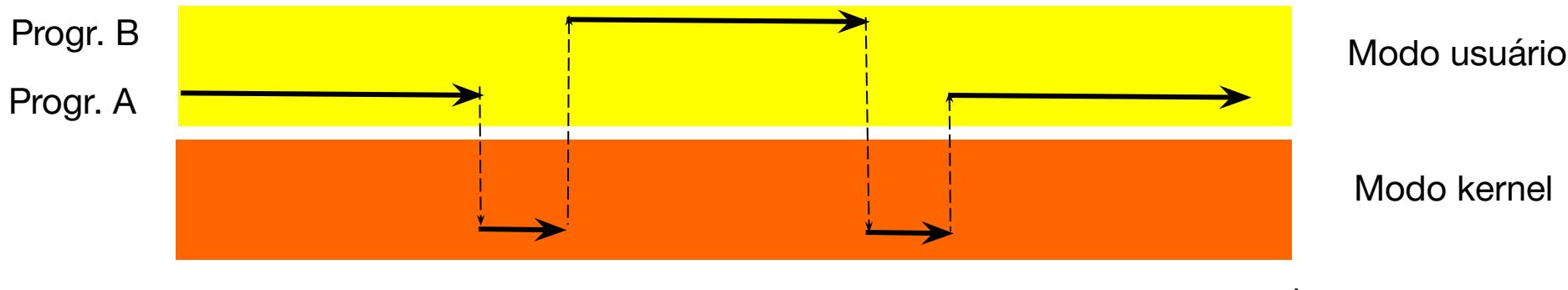
O SO precisa gerenciar a transferência *transparente* de dados de um nível para outro



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Níveis de Privilégio: Modos Usuário X Supervisor (kernel)

- Arquiteturas modernas permitem a operação do processador em dois (ou mais) *níveis/modos de privilégio* da CPU:
 - Modo supervisor/kernel: tem acesso a todas as instruções de máquina e regiões da memória;
 - Modo usuário: apenas um conjunto restrito de instruções de máquina e acesso controlado à memória (apenas a certos endereços da memória)
- Exemplo: No **modo supervisor** é possível acessar as estruturas de dados do núcleo, executar instruções de mudança de mapeamento de memória, manipular qualquer processo, desabilitar certas interrupções, etc.



Modo de Usuário e de Supervisor

Os níveis de proteção são definidos pela arquitetura,

- INTR_PRIVILEGE: para troca de contexto e tratamento de interrupções – *modo kernel*
- TASK_PRIVILEGE: para escrita/leitura de portas de Entrada/Saida (E/S) – *modo intermediário*
- USER_PRIVILEGE: para servidores e processos do sistema (sem acesso direto a portas de E/S) – *modo user*

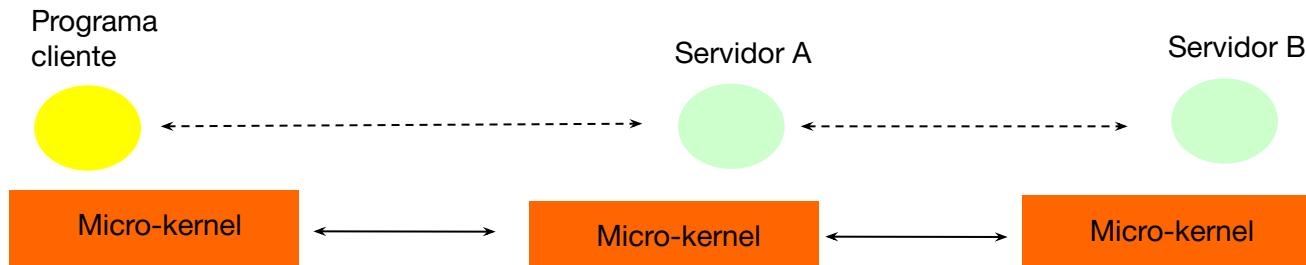
Três eventos que causam a mudança p/ o modo supervisor (kernel)

1. **Interrupções de hardware** de um dispositivo de E/S (disco, clock, mouse, etc.):
 - São assíncronos, e independentes do processo em execução
 - Por exemplo: timer informa que fatia de tempo terminou, ou dispositivo de E/S avisa que completou operação de escrita
2. **Ocorrência de uma exceção** em uma instrução
 - Por exemplo: divisão por zero, ou acesso a um endereço inválido
 - Núcleo geralmente termina o processo!
3. Programa em execução faz uma **chamada de sistema (*system call*)** solicitando algum serviço do Sistema Operacional.
 - Isso causa um **sinal TRAP** para o kernel que faz com que este leia o código da chamada de sistema a partir do registrador e execute a chamada de sistema.

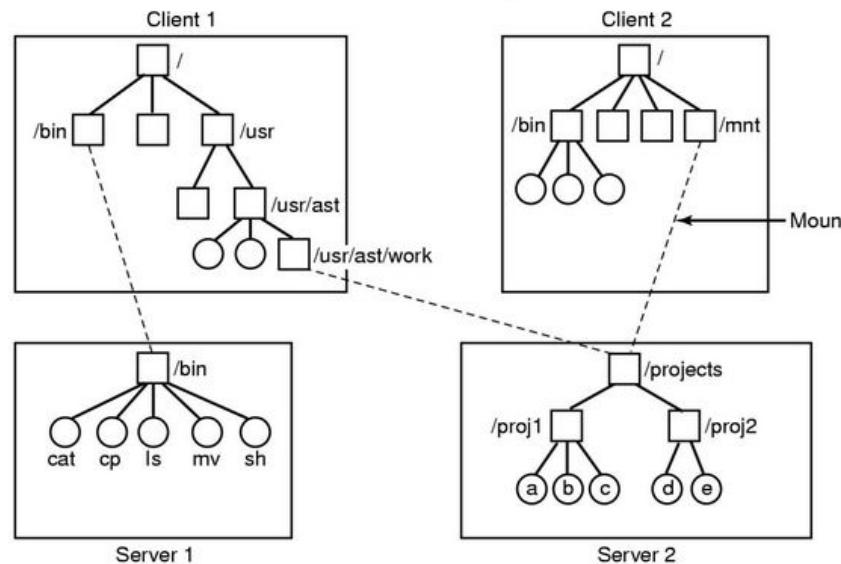
Nos três casos, núcleo faz a troca de contexto e passa controle da CPU a outra tarefa (processo).

Sistemas Operacionais em Rede

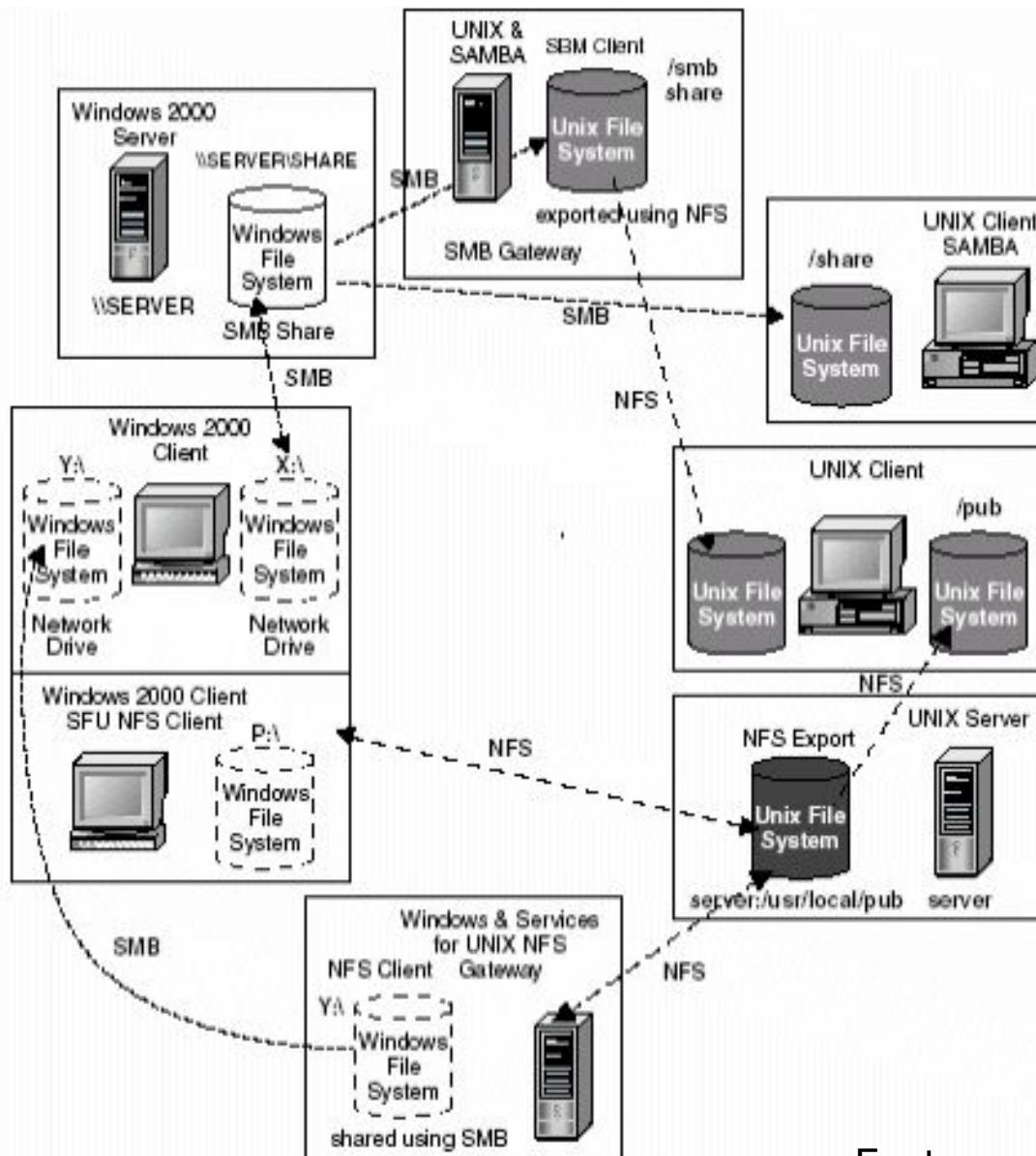
- O mico-kernel e novos protocolos de rede criaram a possibilidade de sistemas em que os recursos fossem acessíveis através de redes locais (LANs)
- Exemplo: compartilhamento de servidores, impressoras, etc.



- Exemplo 2: Sistemas de Arquivos compartilhados em rede



Sistemas de Arquivo em rede, usando NSF e Samba

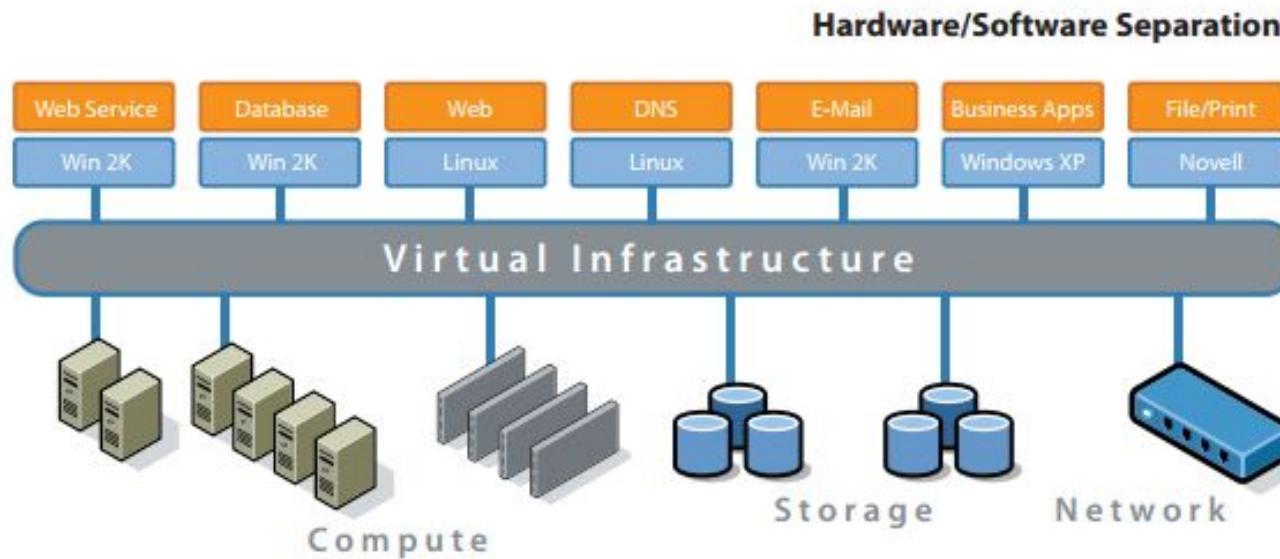


Servidores de arquivos “exportam” partes da árvore de diretórios local, que são referenciados em outros sistemas de arquivos como se fossem parte deste, no mesmo disco.

NSF e Samba têm uma arquitetura Cliente-servidor

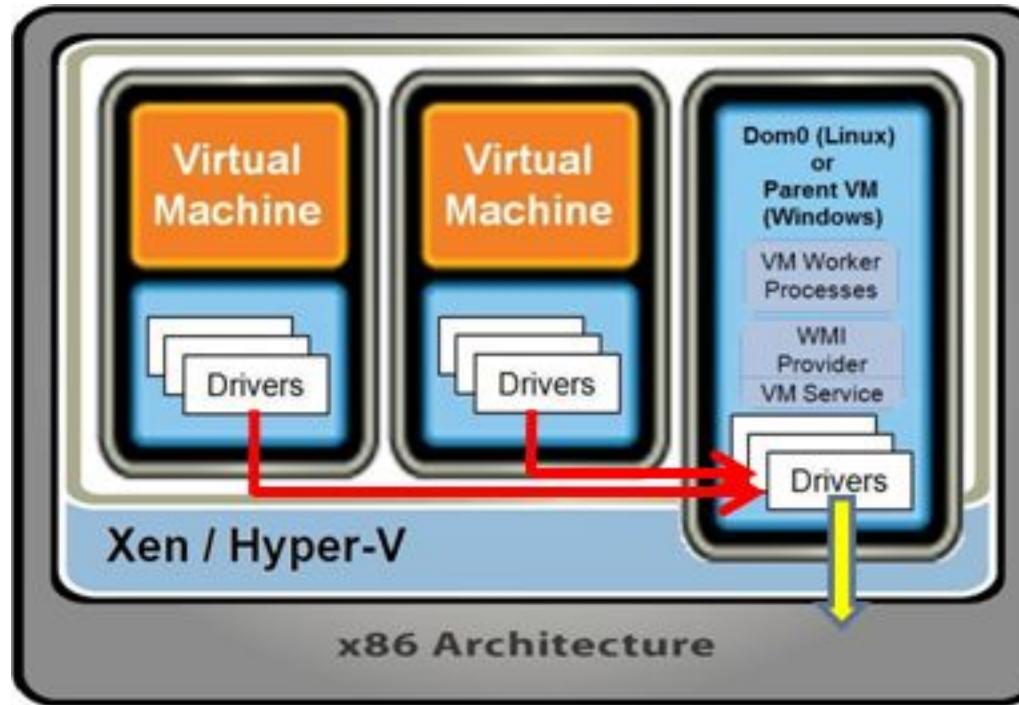
Máquinas Virtuais

Se usados em um cluster, data centro ou rede local, podem também virtualizar conjuntos de processadores, discos, etc.



Máquinas Virtuais

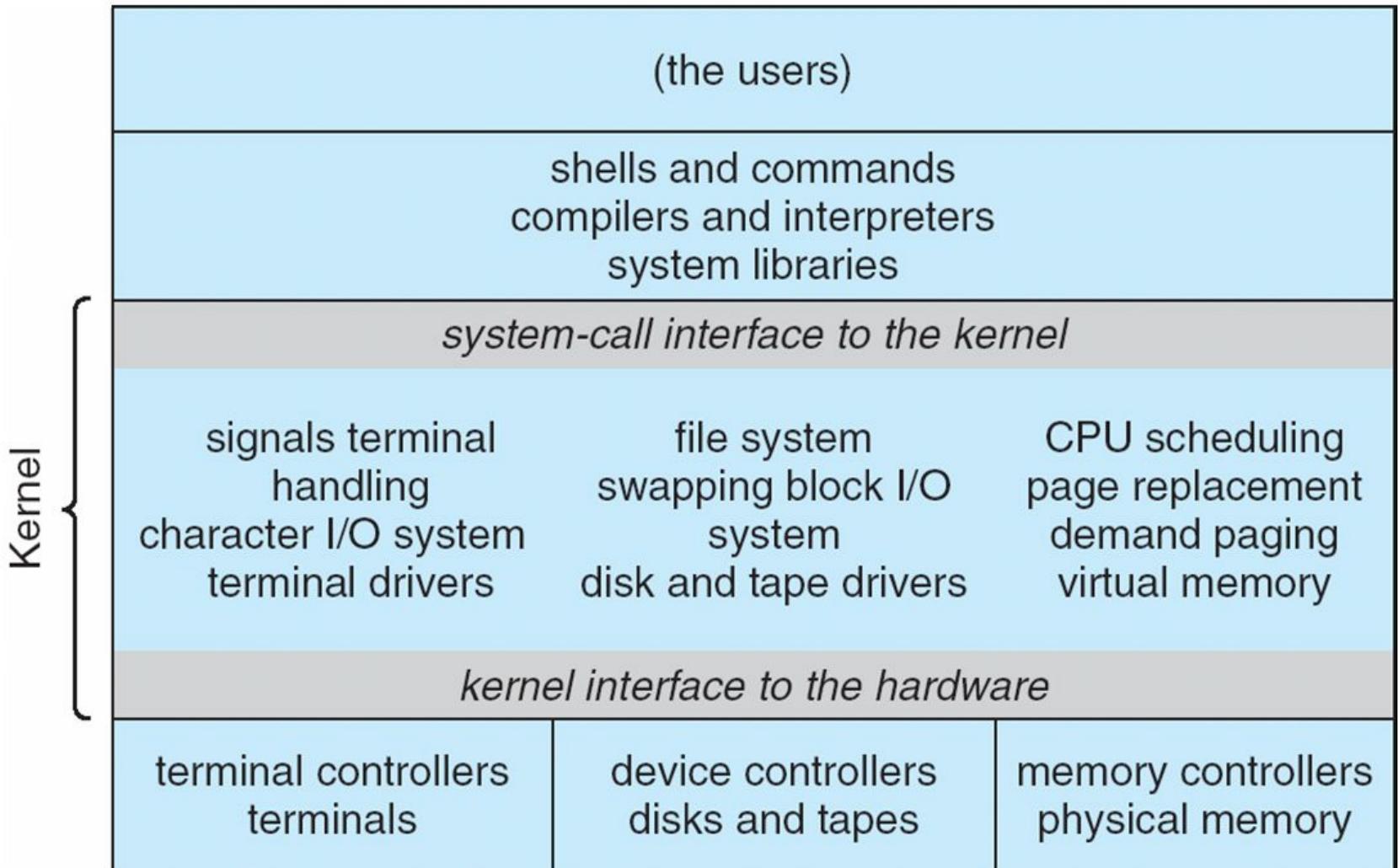
Cada MV tem os seus drivers, cujas operações de I/O são mapeadas – através do hypervisor - para comandos dos drivers do sistema hospedeiro (Em VM Tipo 2) .



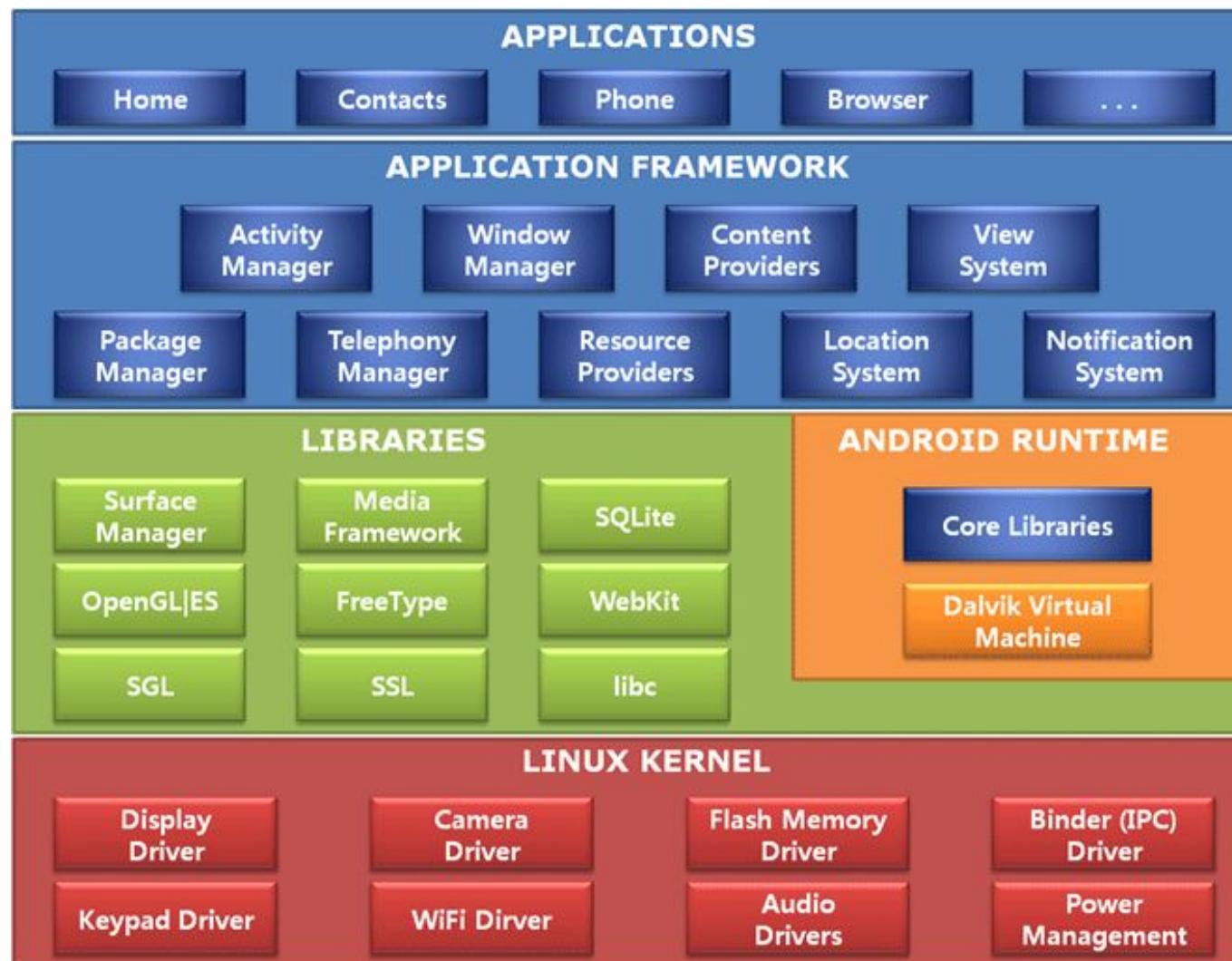
Fonte: blogs.vmware.com/virtualreality

Arquiteturas de Sistemas Operacionais

Estrutura Tradicional de UNIX



Sistema de várias componentes e camadas de software



Android

Características comuns (da estruturação do software)

Software em camadas:

Onde cada camada:

- esconde algumas particularidades (e a heterogeneidade) da camada superior
- através de uma API que apresenta as operações essenciais em um nível de abstração mais alto. uma abstração.

Módulos de Software:

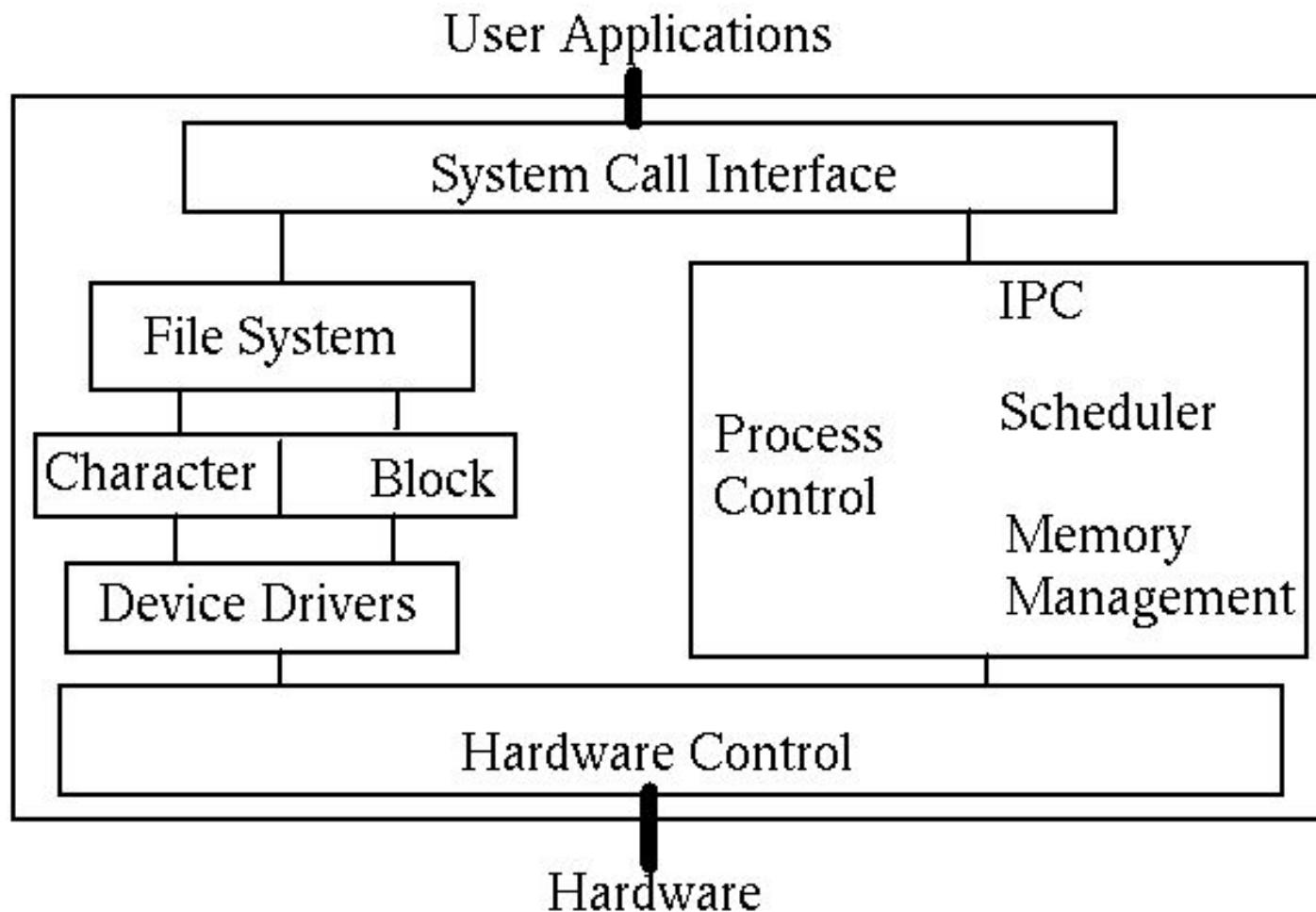
- Cada software responsável por uma função e/ou recurso bem específico.
- A depender do hardware disponível ou do uso do SO, apenas alguns módulos são carregados (ex. Drivers)

Componentes fundamentais

- Os módulos que controlam diretamente o hardware devem ser “enxutos”,
- Se o módulo controla um recurso muito veloz (p.ex. Escalonamento de CPU), a sua otimização é fundamental (p.ex. Melhor em assembly).
- Virtualização e Multiplexação: módulos permitem compartilhar recursos de forma segura.

Estrutura Tradicional de UNIX

(muito simplificado)

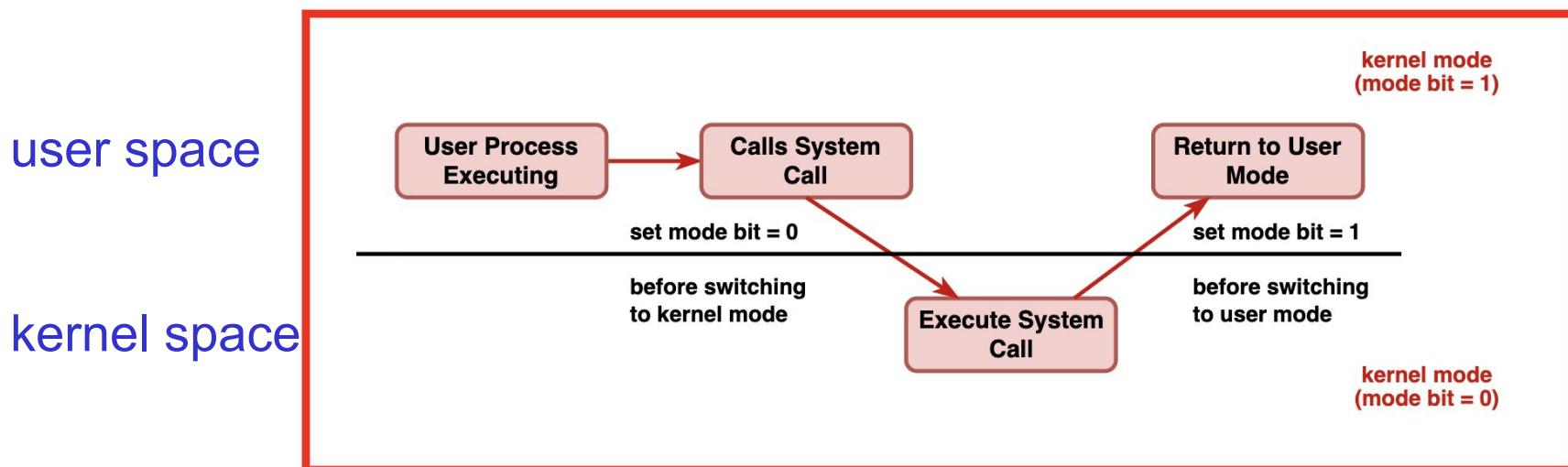


Chamada de sistema (system call)

É como processos solicitam um serviço ao núcleo (kernel). Passa-se o controle para o núcleo, que só mais tarde devolve o controle para o processo que fez a system call.

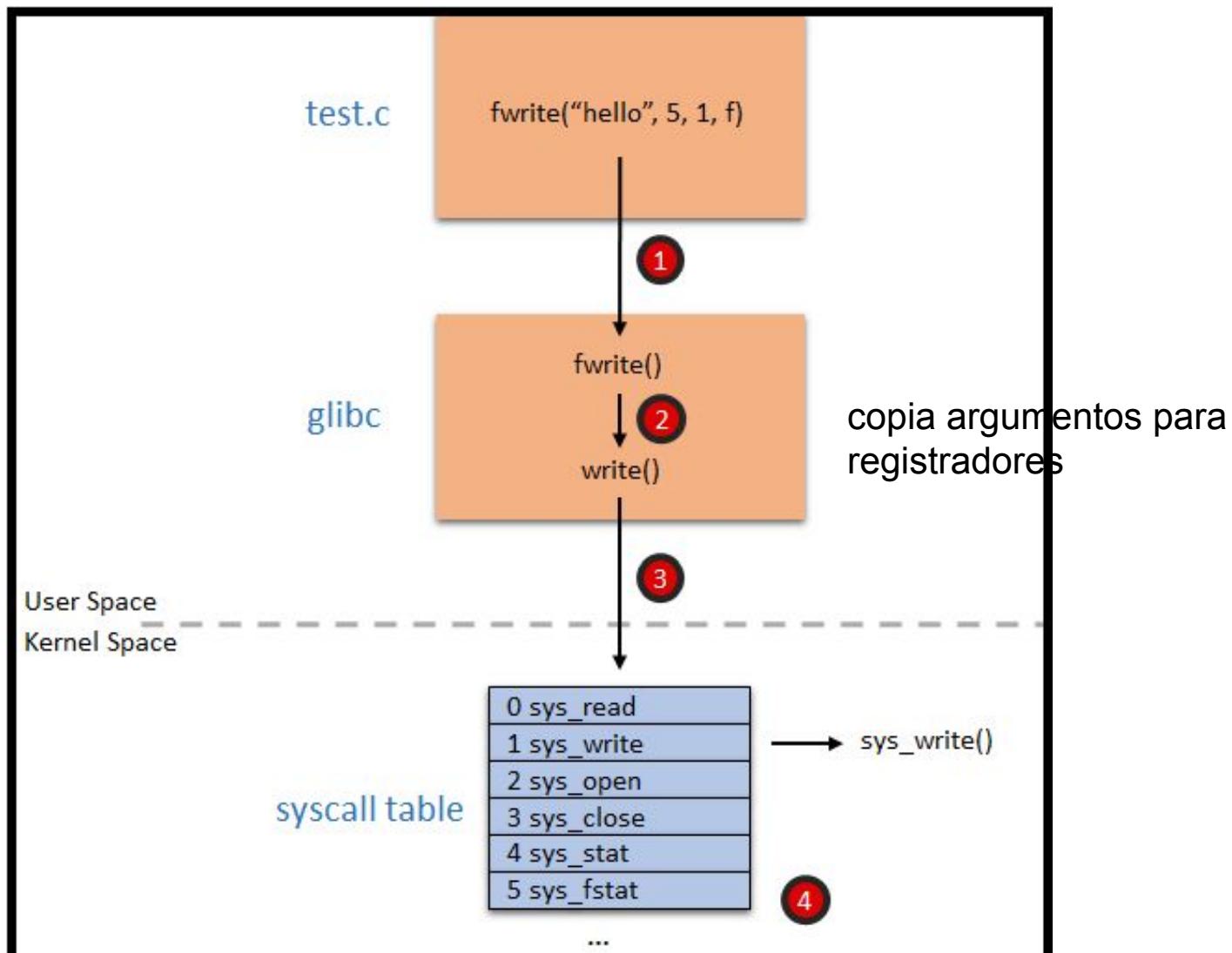
Cada arquitetura tem os seus próprios requisitos sobre como os argumentos da **chamada de sistema** são passados para o kernel.

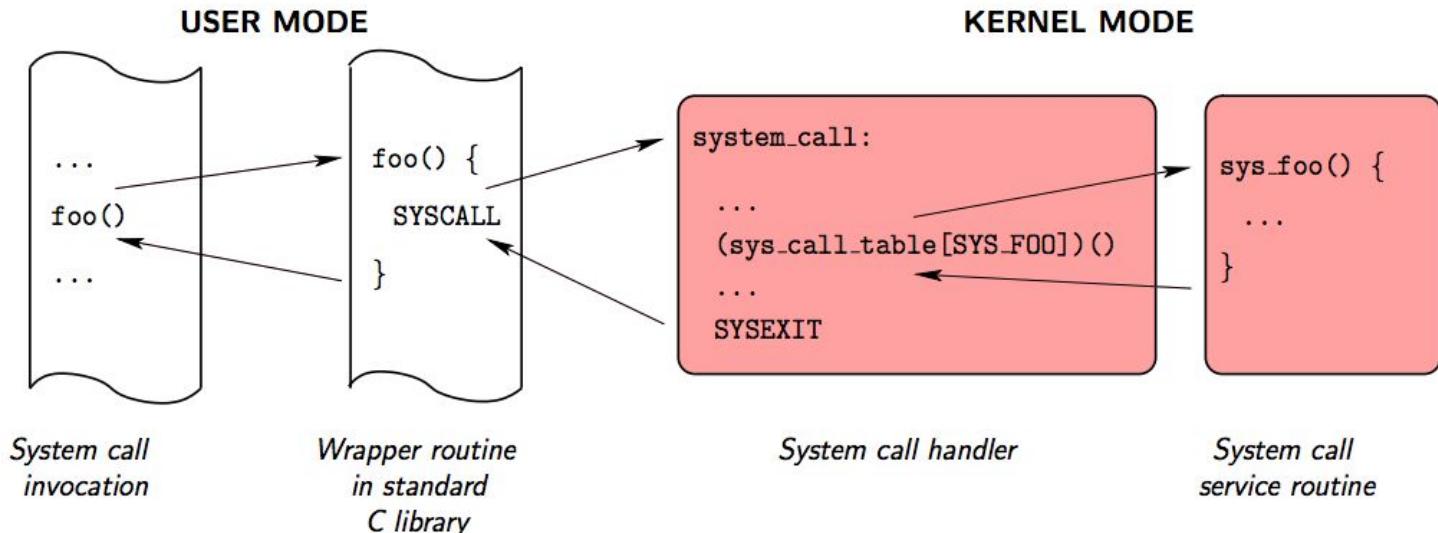
A biblioteca **glibc** realiza a cópia de argumentos da chamada para os registradores da CPU de uma forma adequada e executa um TRAP.



Chamada de sistema

glibc prepara a chamada da função sys_ do núcleo





Acções da biblioteca glibc (wrapper)

1. Mover parâmetros do user stack para registradores da CPU
2. Entrar em modo kernel e desvia para system call handler
instrução específica do processador (TRAP, SYSCALL,)
3. Processa o valor de retorno e seta errno

Acões do System call Handler

1. Salva contexto da CPU para a pilha do modo kernel
2. Chama função de serviço no kernel (“system call service routine”)
Em Linux: vetor com ponteiros de função indexado pelo número da chamada de sistema
3. Recarrega o contexto da CPU
4. Volta para o modo usuário com SYSEXIT
instrução específica do processador (rti, sysexit, ...)

Chamadas de Sistema

Existem esencialmente 5 tipos diferentes de chamadas de sistema no sistema operacional:

1. Controle de processos

- /> Para abortar o processo à força
- /> Executar um processo depois de o carregar na memória principal.
- /> Terminar o processo atual antes de iniciar um novo processo.
- /> Atribuir memória a um processo e, em seguida, libertar a memória se o processo for terminado.
- /> identificar PID do processo e seu pai

Description	Syscall
Process creation	fork
Changing executable file	exec (execvp)
End process	exit
Wait for a child process(can block)	wait/waitpid
PID of the calling process	getpid
Father's PID of the calling process	getppid

Chamadas de Sistema

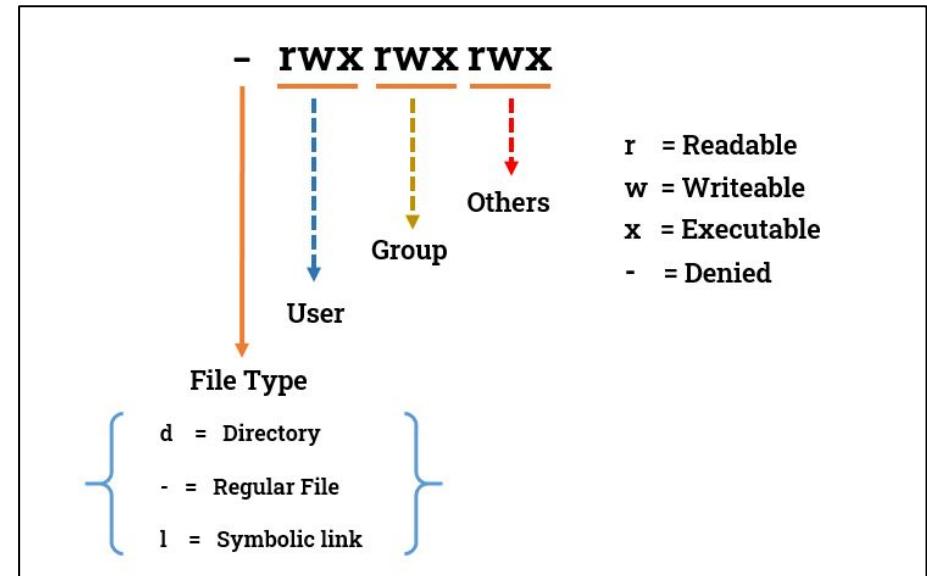
2. Gestão de Arquivos

Os seguintes serviços são fornecidos pela chamada de sistema para gestão de arquivos:

- /> Criar e remover arquivos
- /> Abrir o arquivo (prepará-lo para o acesso)
- /> e depois fechá-lo.
- /> Ler um arquivo a partir de determinada posição no arquivo.
- /> Escrever bytes em um arquivo e
- /> Consultar ou alterar atributos de um arquivo

(exemplo:

controle de acesso rwx.rwx.rwx)



Chamadas de Sistema

3. Gestão de dispositivos

Os dispositivos podem ser necessários enquanto um processo está em execução, como o acesso ao sistema de arquivos, dispositivos de E/S, memória principal, etc.

- pode-se requisitar um dispositivo e libertá-lo quando a tarefa estiver concluída. (**request/release device**)
- chamadas para acessar um dispositivo alocado (**read/ write/ reposition**)
- Para associar ou disassociar do sistema (**attach/ detach device**)
(ex. câmera, smartphone, pendrive conectado na entrada USB
 - Em POSIX, dispositivos são tratados como “arquivos especiais” (no diretório /dev)
 - Acesso aos dispositivos é mediado pelo S.O. como um acesso ao um arquivo convencional
 - chamada ioctl() permite executar várias funções de controle sobre um dispositivo

Chamadas de Sistema

4. Manutenção de informações

A chamada de sistema para manutenção da informação transfere dados do programa do usuário para o sistema operacional. Exemplo:

- Acessar ou setar a hora ou data do sistema.
 - Obter informações relacionadas ao sistema. Configurar os dados do sistema.
 - Obter as características de um processo específico do sistema operacional.
 - Definir características/prioridades de um processo específico do sistema operacional.
-
- get (or set) timer or date
 - get (or set) system date
 - get process, file or device attributes
 - set process, file or device attributes

Chamadas de Sistema

5. Comunicação

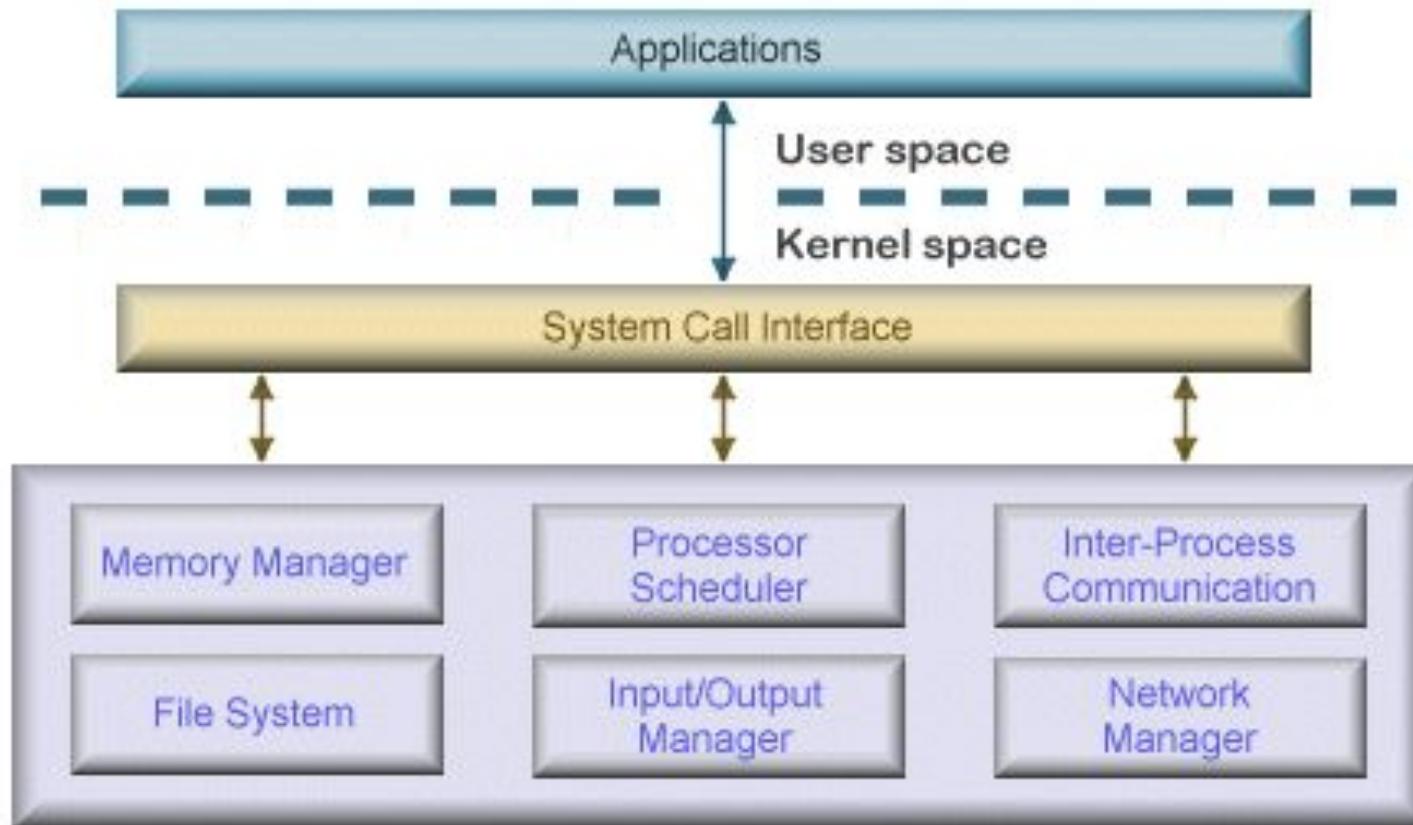
Esta chamada de sistema facilita a ligação do sistema à interface de rede. Os serviços que estas chamadas de sistema oferecem são:

- Abrir uma nova conexão para enviar os dados.
- Depois de terminada a transmissão, desfazer a conexão.
- Enviar uma mensagem numa determinada conexão.
- receber pacotes de dados a partir de uma ligação específica
- Identificar e se ligar a um dispositivo remoto específico à rede.
Remover um computador ou dispositivo remoto específico da rede.
 - criar, usar e fechar uma pipe()
 - definir uma shared memory
shmget()
 - definir um memory map mmap()

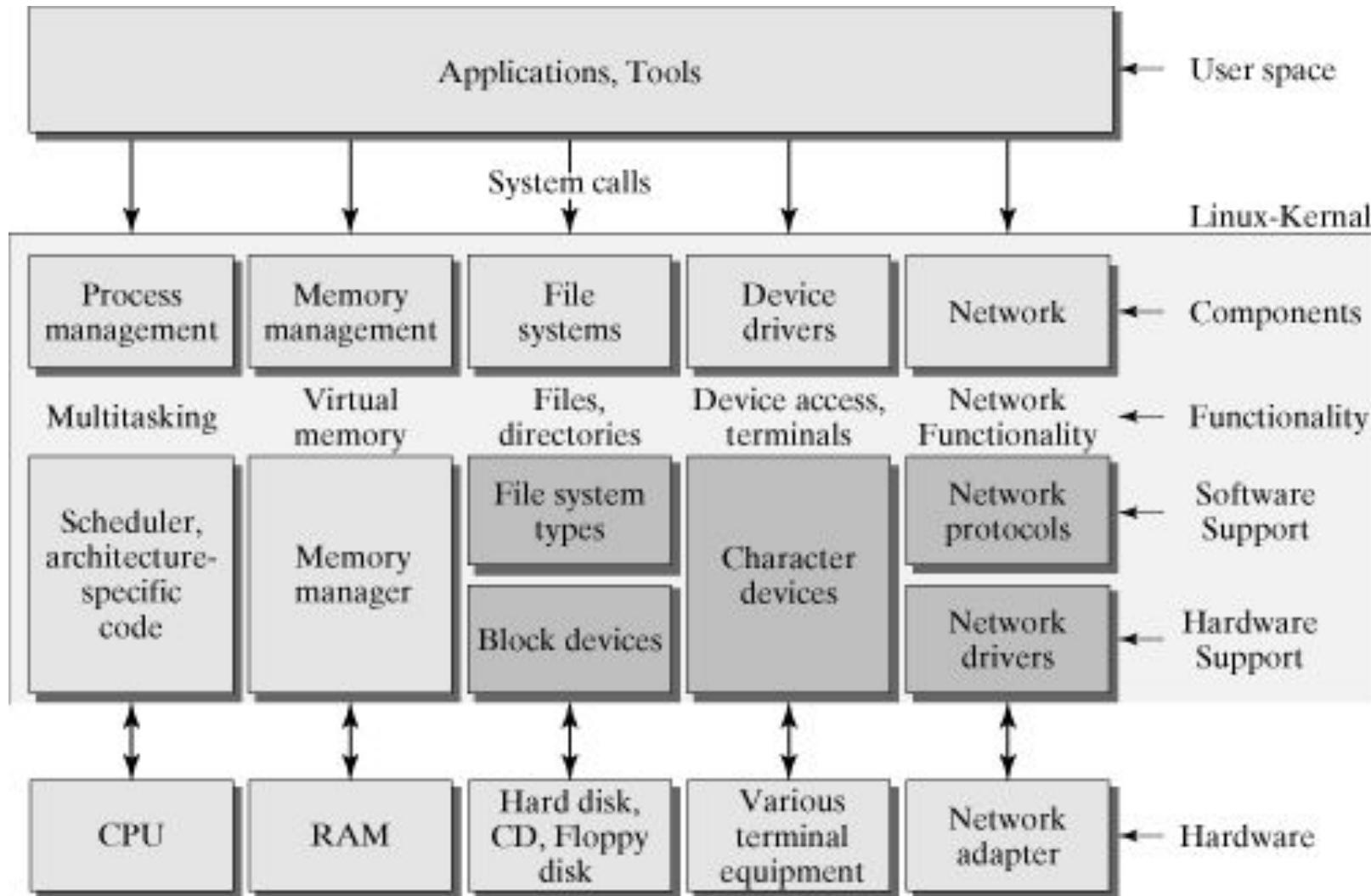
Chamadas de Sistema em Windows e Unix (as mais comuns)

Types of System Calls	Windows	Linux
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

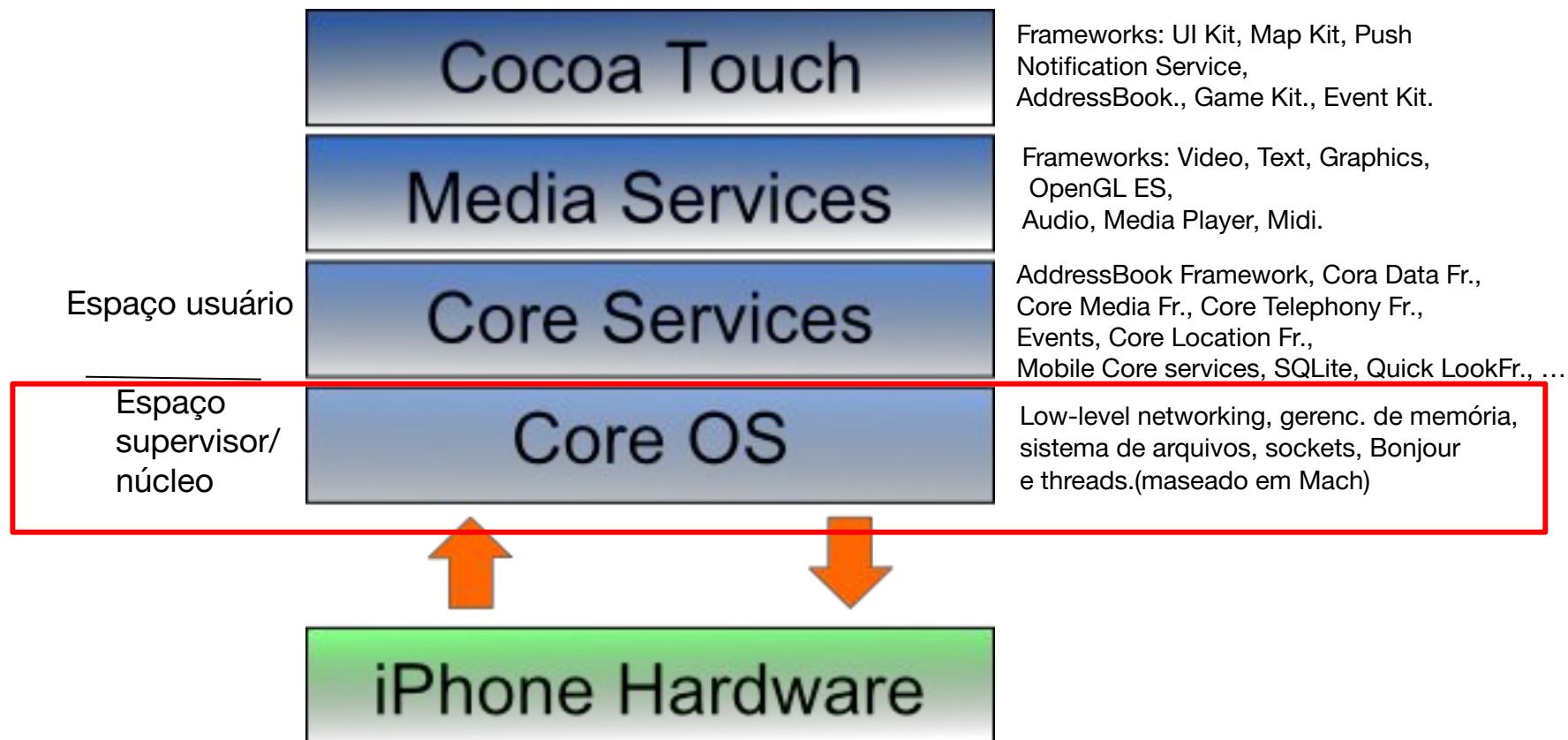
Arquitetura monolítica



Estrutura (monolítica) do Linux



Estrutura do iOS (Apple)

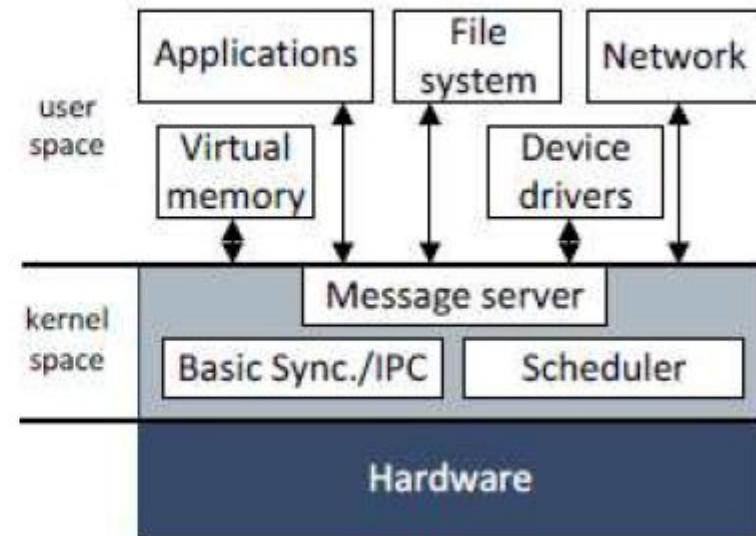
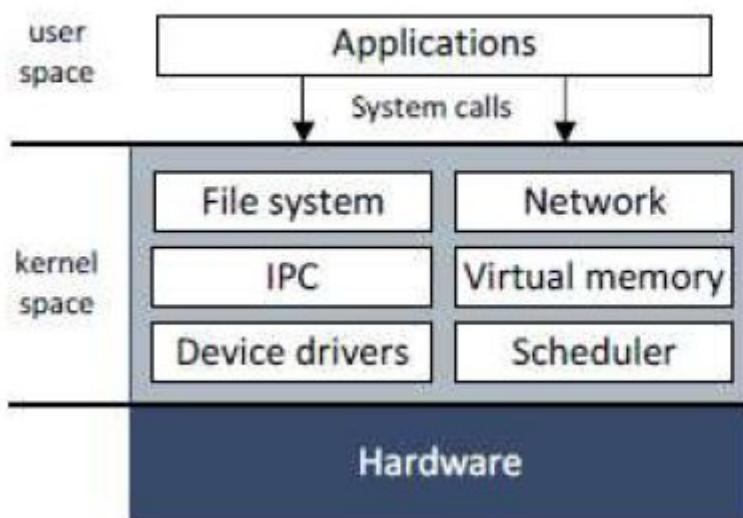


Fonte:

http://www.techotopia.com/index.php/The_iOS_4_Architecture_and_SDK_Frameworks

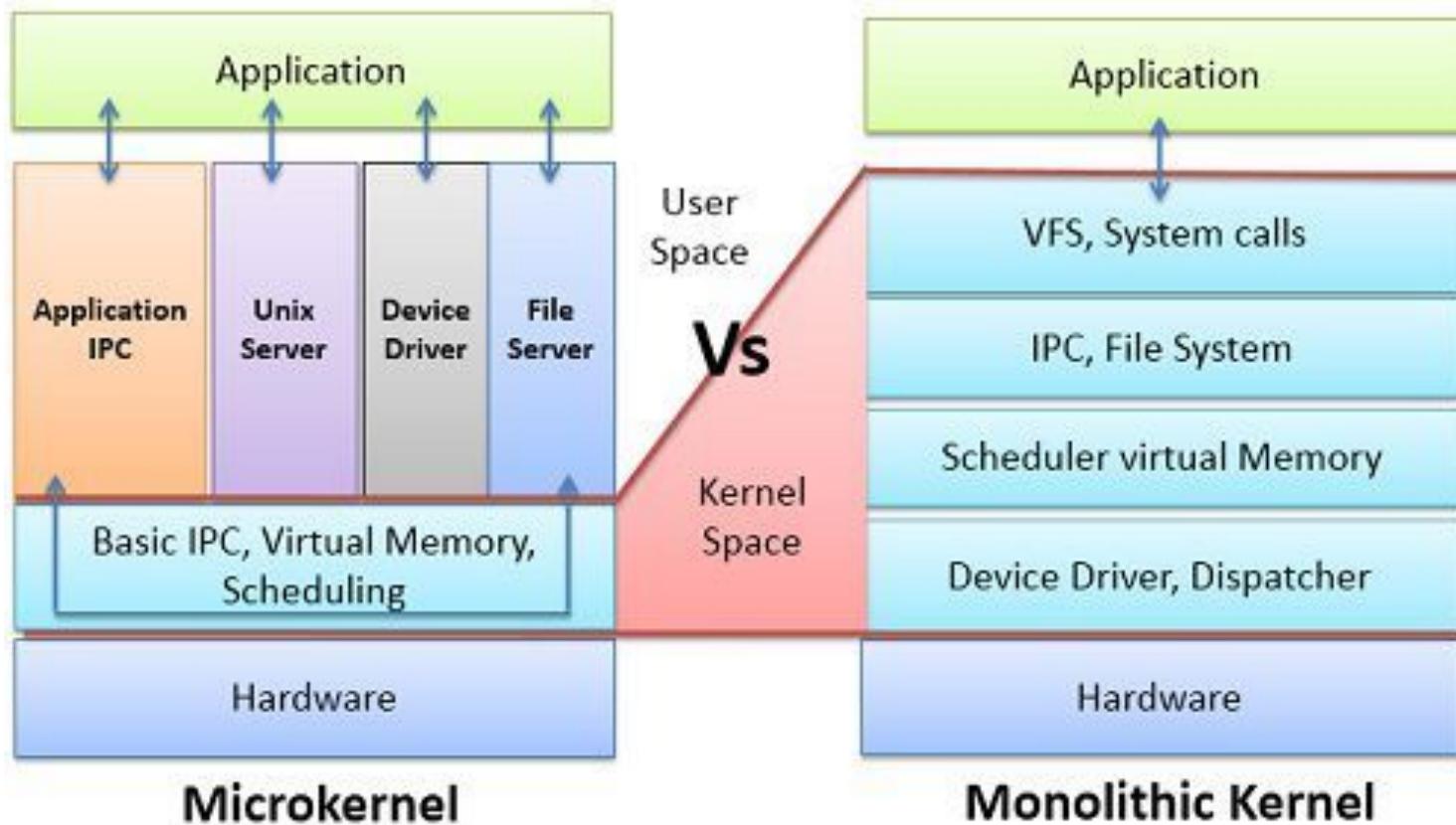
Padrão arquitetural *microkernel*

- O padrão arquitetural micro-kernel (micro-núcleo) é para sistemas de software que precisam ter a habilidade de se adaptar a variações nos requisitos do sistema.
- Separa um núcleo funcional mínimo e todos os demais serviços executam como processos em user space.
- O microkernel também é um “ponto de conexão”/“barramento” para conectar todos os serviços e coordenar a colaboração entre elas.

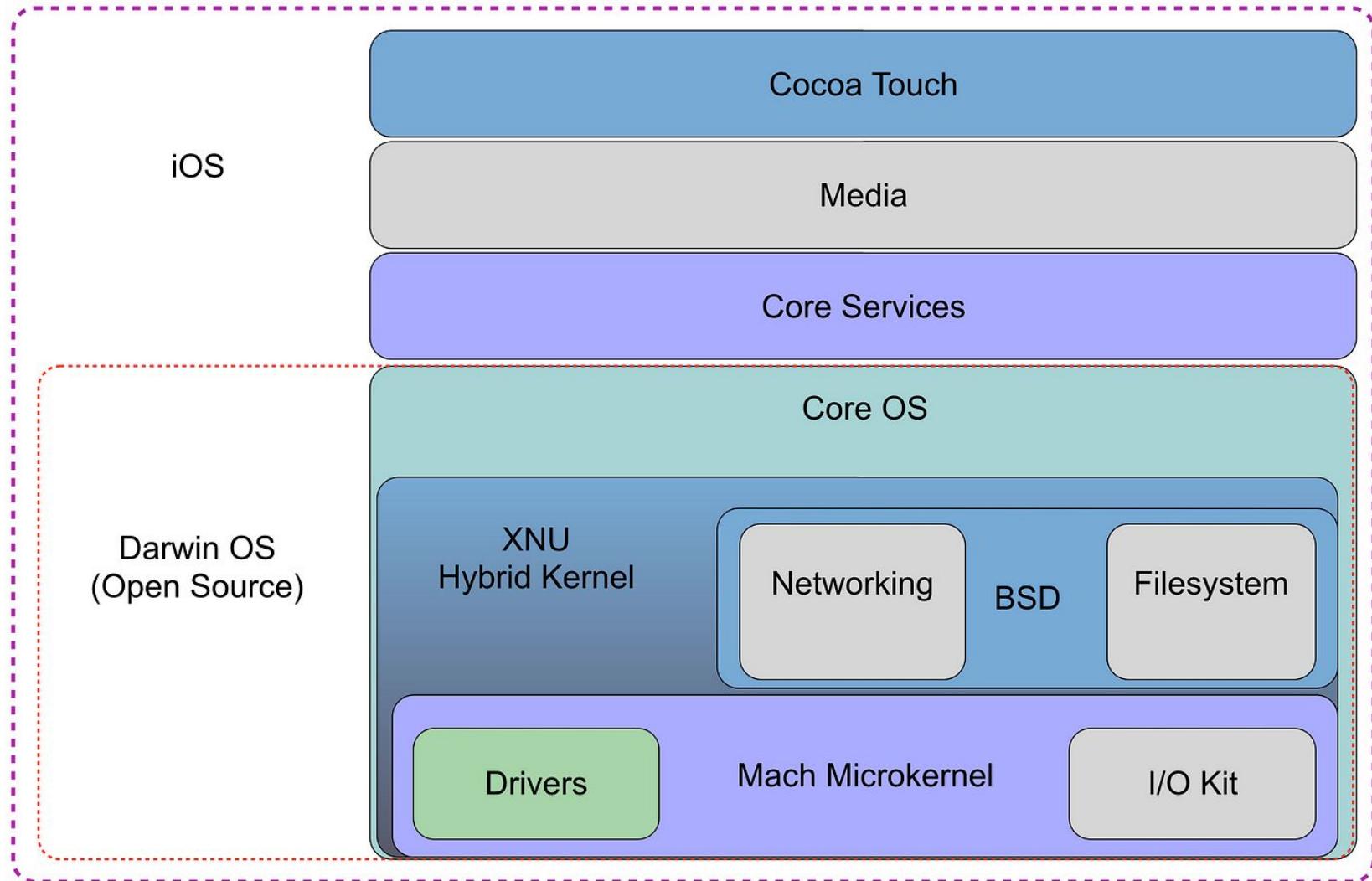


Exemplo: MACH Microkernel (CMU, Rashid, Tevanian, 1985-94)

Mach – SO baseado em microkernel



iOS e MacOS - baseados no micro-kernel MACH



Darwin OS = híbrido de MACH e FreeBSD

Padrão arquitetural *microkernel*

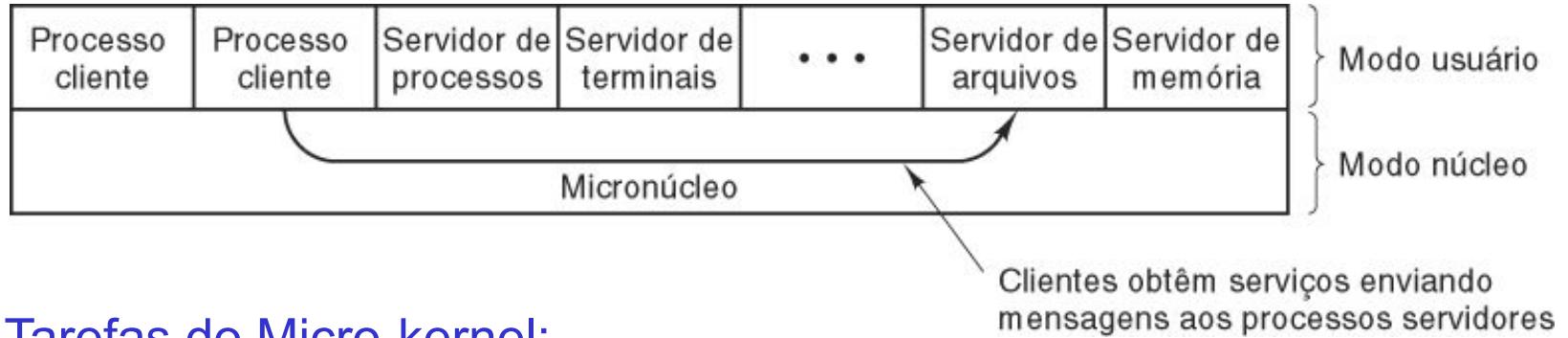
Benefícios:

- Boa **portabilidade**, pois somente o microkernel precisa ser modificado ao portar o sistema para um novo ambiente.
- Alta **flexibilidade e extensibilidade**, pois as modificações ou extensões podem ser feitas modificando ou estendendo os servidores internos.
- A separação dos mecanismos de baixo nível (fornecidos pelo microkernel e pelos servidores internos) e das políticas de alto nível (fornecidas pelos servidores externos) melhora a capacidade de manutenção e de alteração do sistema.

Problemas:

- O sistema de microkernel exige muito mais comunicação entre processos
- O projeto e a implementação do sistema baseado em microkernel são muito mais complexos do que os de um sistema monolítico
-

Estrutura de micro-núcleo (*microkernel*)



Tarefas do Micro-kernel:

- Troca de contexto
- Tratamento de interrupções
- Gerenciamento básico de memória
- E/S básica, e.g. envio de dados e de controle a dispositivos
- Comunicação entre processos (como E/S)

Vantagens:

- Boa parte dos drivers executa em modo usuário (→ maior robustez)
- Menor trabalho de portar o S.O. para outra arquitetura

Sistemas Operacionais com arquitetura microkernel

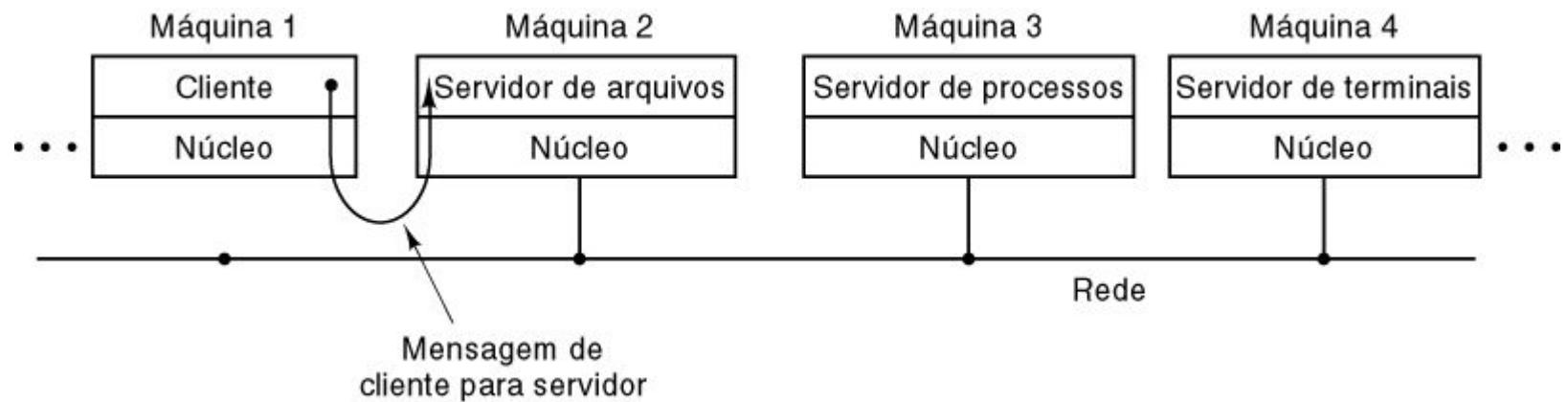
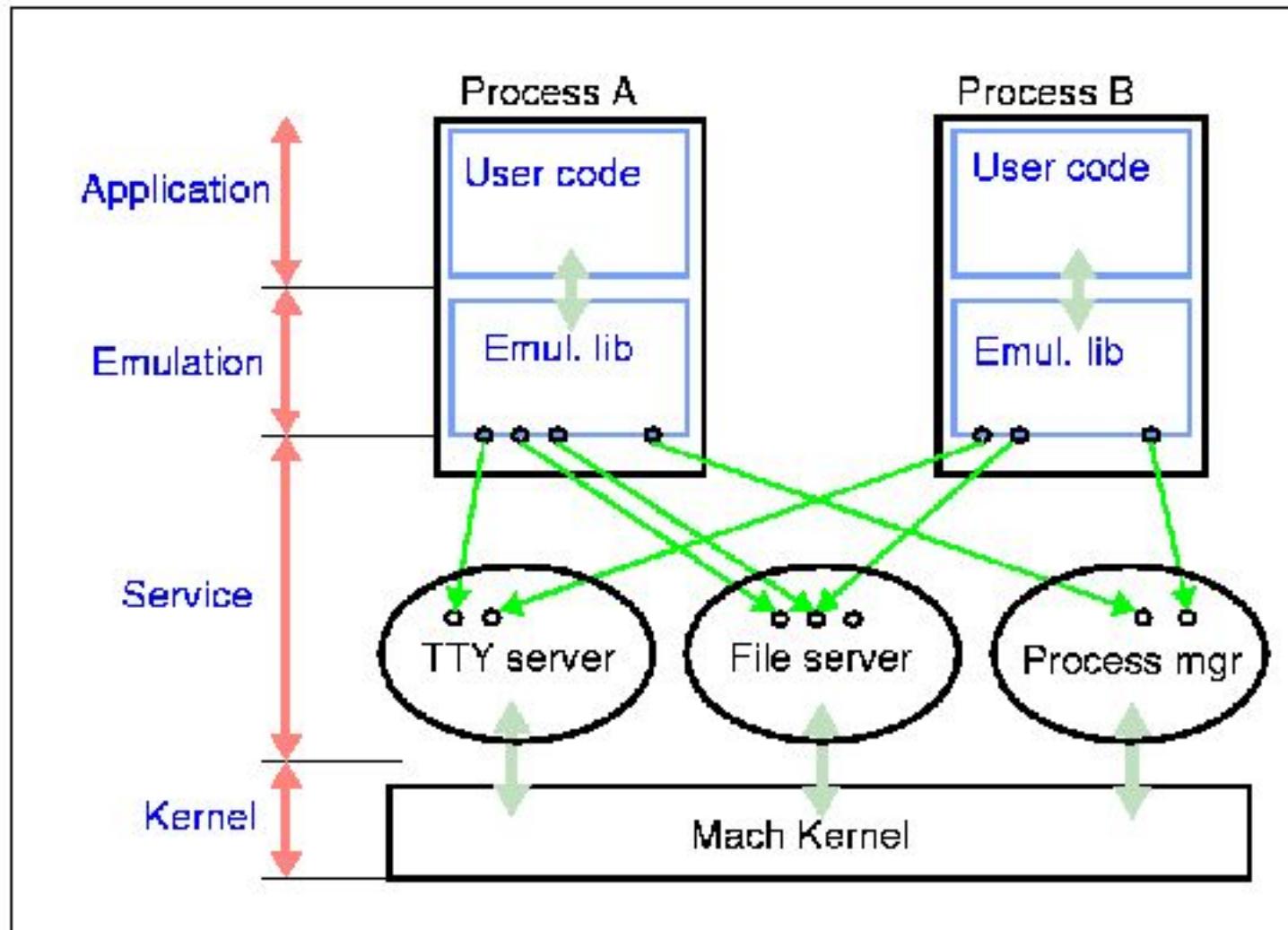


Fig.: O modelo cliente-servidor em um sistema distribuído

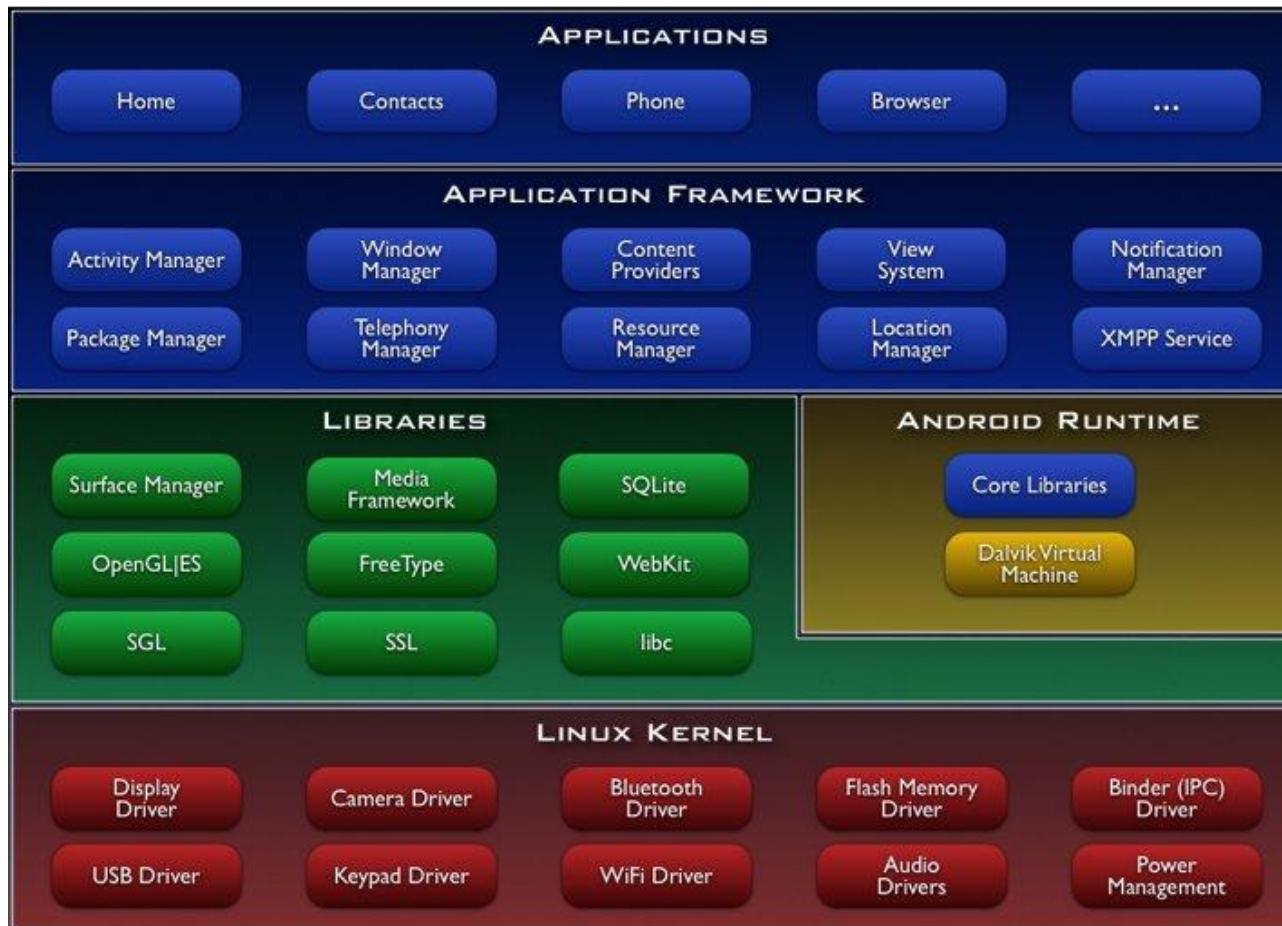
Micro-kernel também facilita a criação de sistemas operacionais em rede, onde determinados servidores podem ser acessados remotamente.

Mach – SO baseado em microkernel



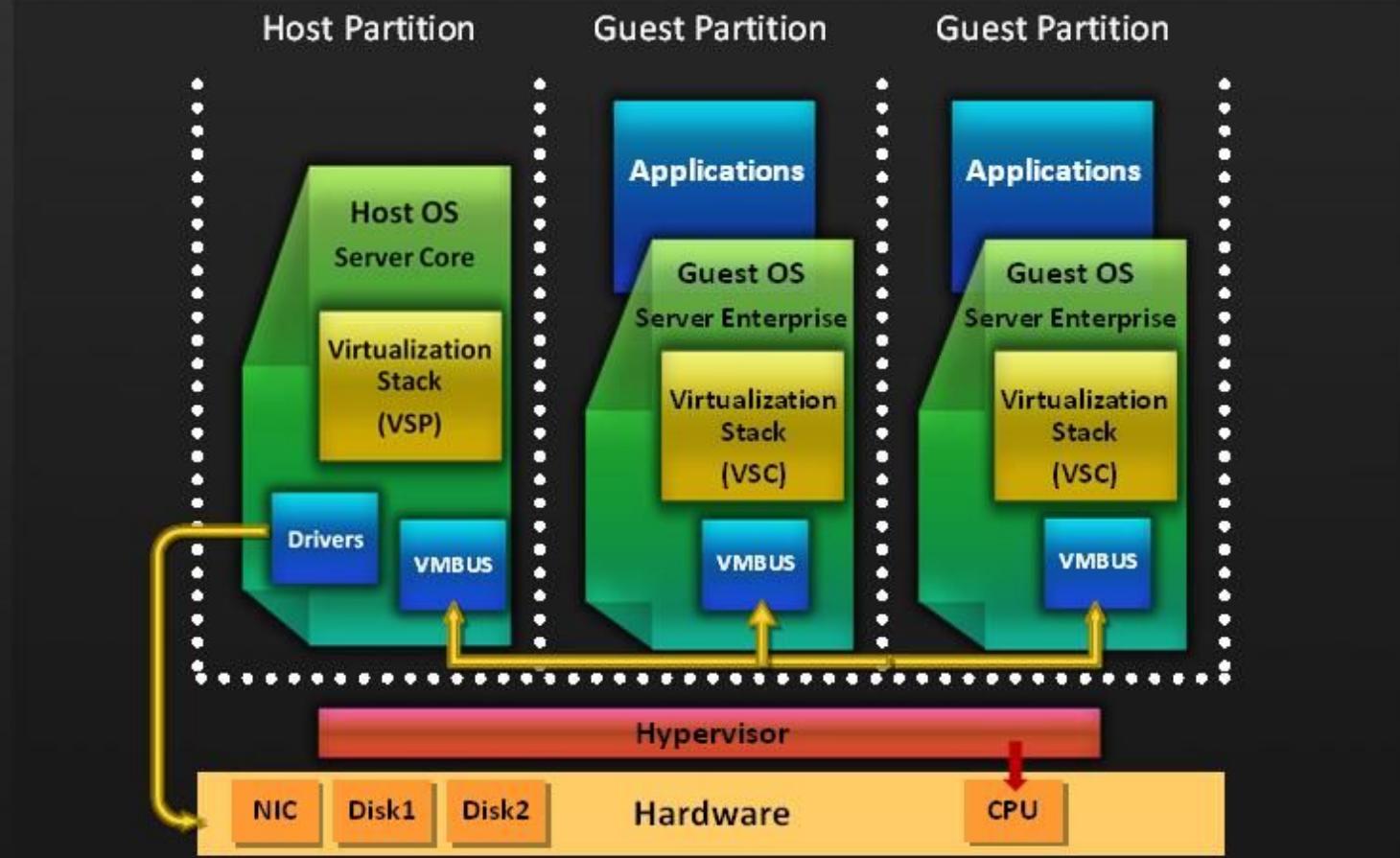
Mach – SO baseado em microkernel

Estrutura do Android

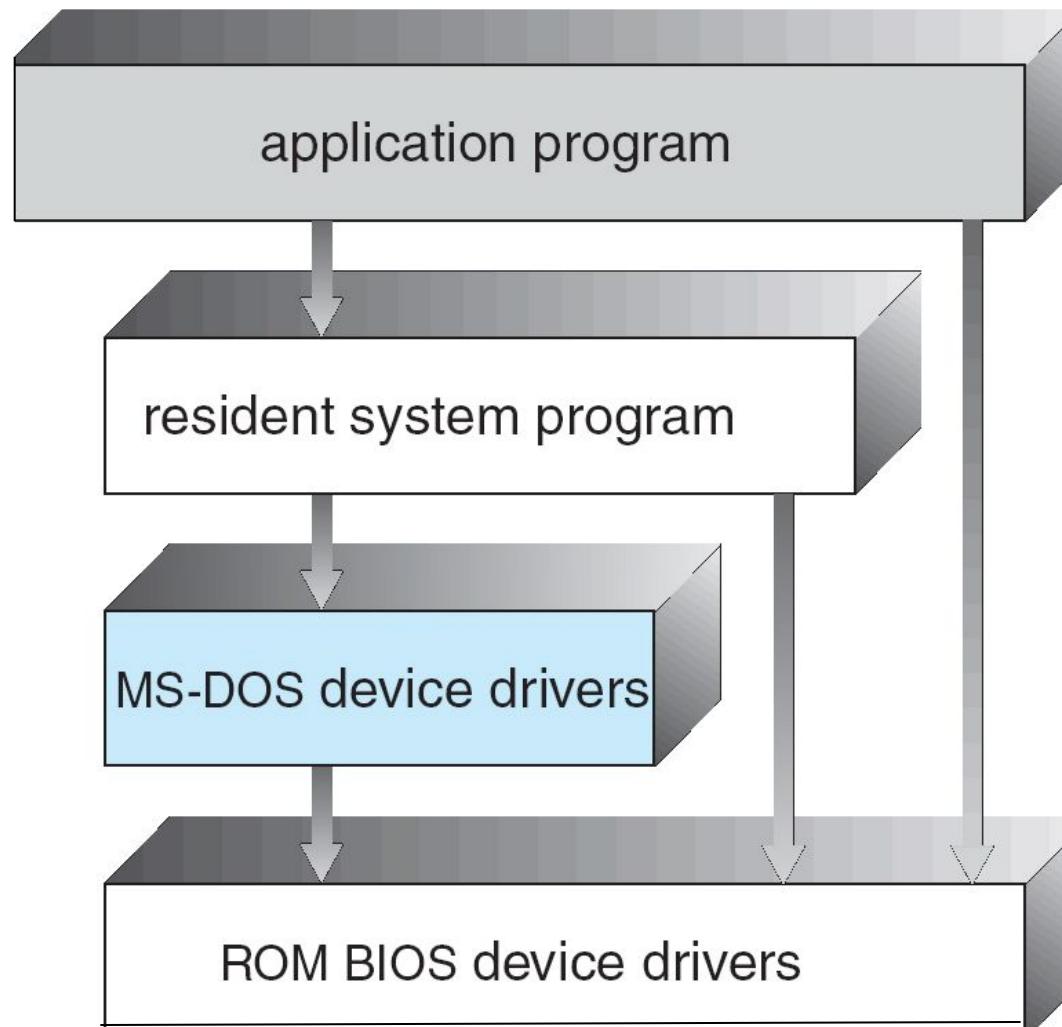


Estrutura do Azure (Miccosoft)

High-Level Architecture

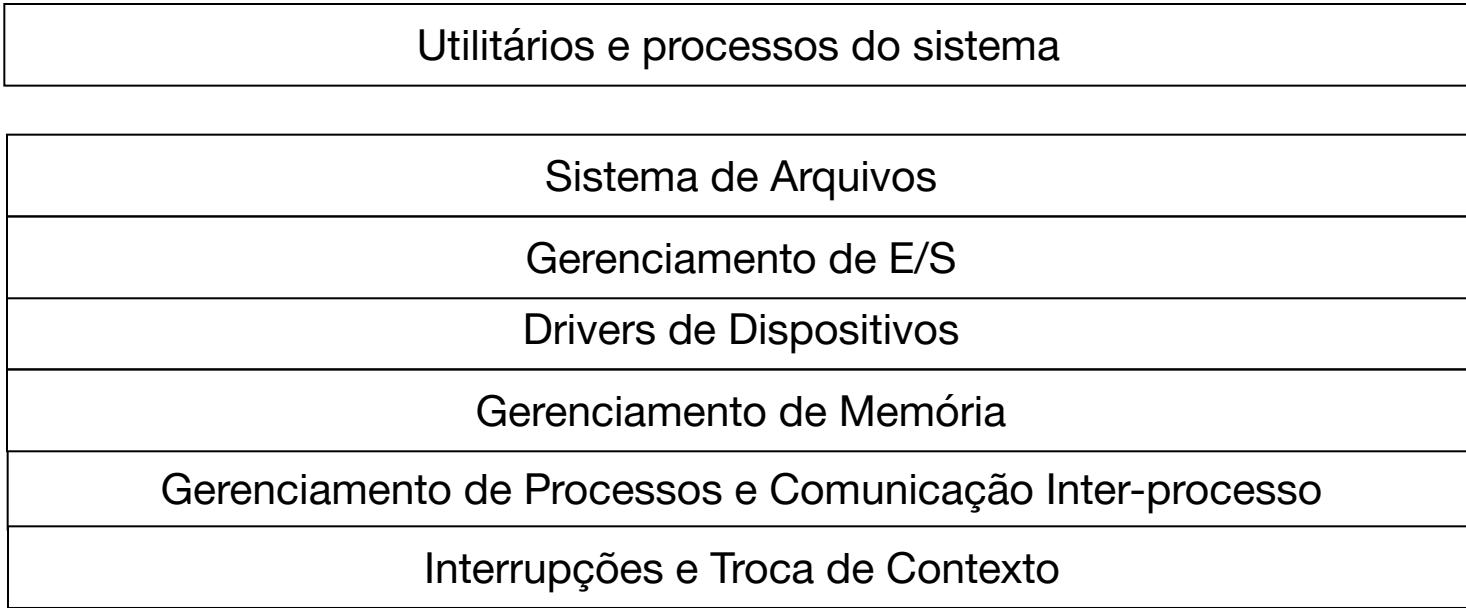


MS-DOS Layer Structure



Organização do núcleo

núcleo



Cada camada define uma interface bem-definida para a camada de cima;

Chamadas só ocorrem entre camadas adjacentes;
Boa organização, mas problemas de desempenho.

Estrutura de Sistemas Operacionais: máquinas virtuais

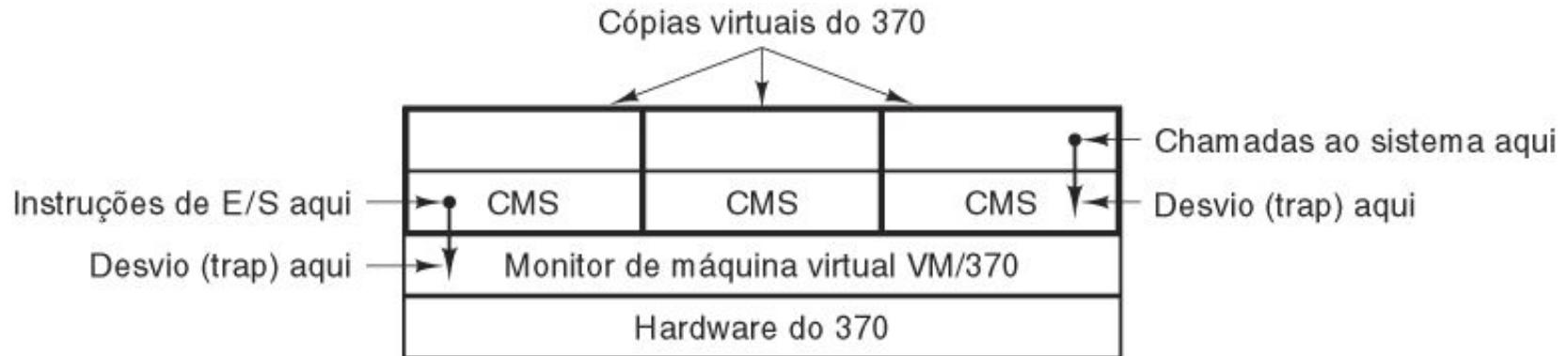


Fig.: Estrutura do VM/370 com o CMS

Idéia central: separação entre máquina estendida e multiprogramação

Princípio básico: Monitor realiza a multiplexação do hardware:

- mapeia endereços lógicos para endereços reais (partições de disco, regiões de memória)
- instruções de HW geradas pelos system calls são capturadas pelo monitor que as executa como simulação do hardware virtual

Virtualização

Execução de vários sistemas operacionais independentemente em um único computador.

Cada SO executa em uma máquina virtual (VM) que acessa recursos através de um monitor.

O Monitor (ou hypervisor):

- implementa - em software - exatamente as instruções da máquina hospedeira, e traduz essas para as instruções da máquina física.
- permite compartilhar os recursos (CPU, memória, disco, etc.) e multiplexar o uso dos demais dispositivos, com total isolamento.

Vantagens:

- Melhor utilização dos recursos de HW (usado em *data centers*)
- Evita que um sistema operacional hospedado possa *bloquear ou terminar* uma máquina

Desvantagens:

- Torna mais lenta a execução de cada sistema hospedado

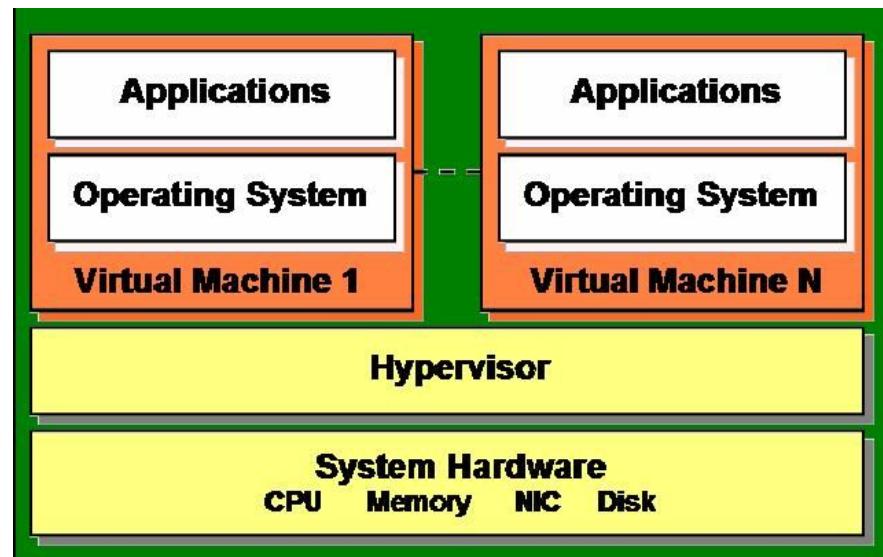
Exemplos: VMware, Bochs, Xen, Virtual Box, Virtual PC,...

Máquinas Virtuais

Máquinas virtuais emulam o hardware, e virtualizam os recursos físicos de uma máquina física.

O monitor (hypervisor) isola as máquinas virtuais e implementa a camada de virtualização.

Todas as instruções de máquina são interceptadas pelo monitor, e o hypervisor multiplexa a utilização dos recursos físicos.



Dois tipos de máquinas virtuais

Type 1 hypervisor

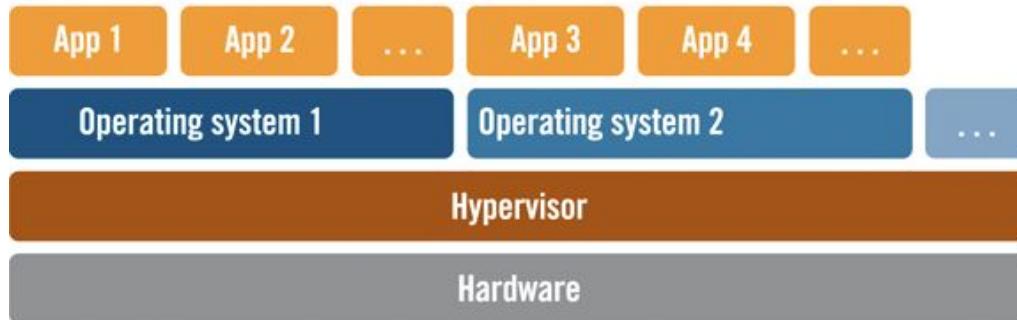


Figure 2. A Type 1 or bare-metal hypervisor sits directly on the host hardware.

Exemplos:

Hyper-V (Microsoft)
VMware ESX Server
...

Usado em servidores
de data centers

Exemplos:

VirtualBox, VMware
Server, VMware
Workstation and
Microsoft Virtual PC

Usado em PCs e
máquinas pouco
compartilhadas

Type 2 hypervisor (Guest OS Virtualization)

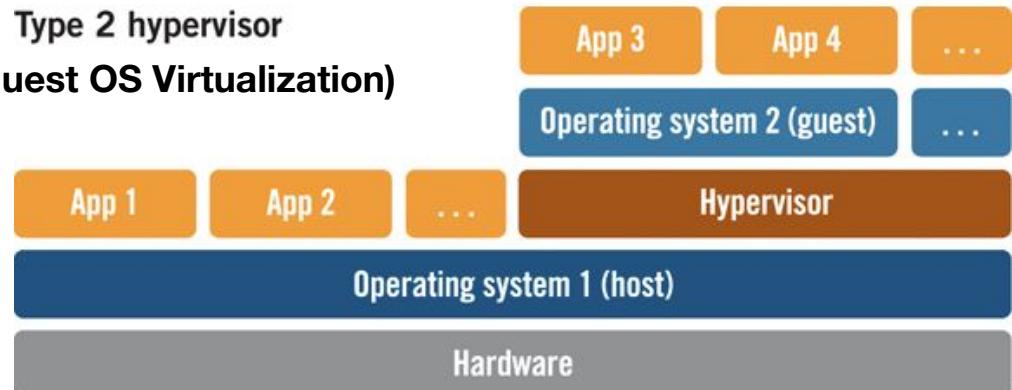
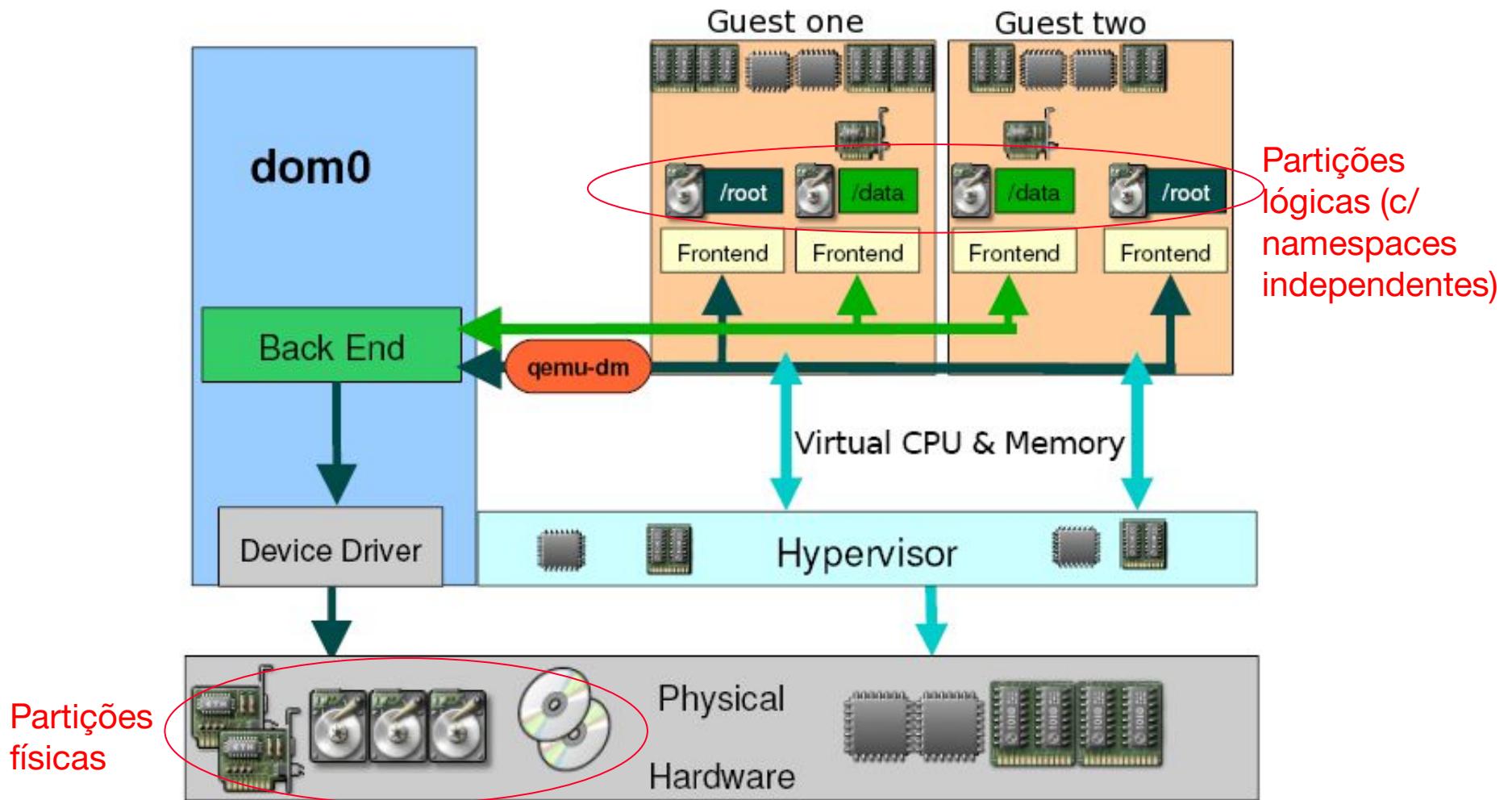


Figure 1. A Type 2 hypervisor runs as an application on a host operating system.

Exemplo de Virtualização de E/S



Cada Guest executa um Qemu daemon. Este que trata todas as requisições de rede e disco dos Guests

Conceitos básicos de sistemas tipo UNIX

Conceitos básicos (UNIX)

- **Usuário** invoca comandos (i.e. executa programas) através de interpretador de comandos em linha de comando (*shell*).
- Cada comando da shell é um programa, que pode ser executado em 1º ou 2º plano.
- Exemplo: `ps`, `ls`, `sed`, `sort`, ...
- **Shell** tem o terminal como entrada e saída padrão (STDIN e STDOUT)
- A shell e os comandos herdam o UserID e um GroupID do usuário que executou o programa login
- Programas acessam, criam e modificam arquivos, e têm associados a eles atributos e permissões.

Processos

Um processo é um **programa em execução**.

A partir do boot do sistema, cada processo é criado por um outro processo pai, e ambos prosseguem suas **execuções independentes**.

Um processo consiste de:

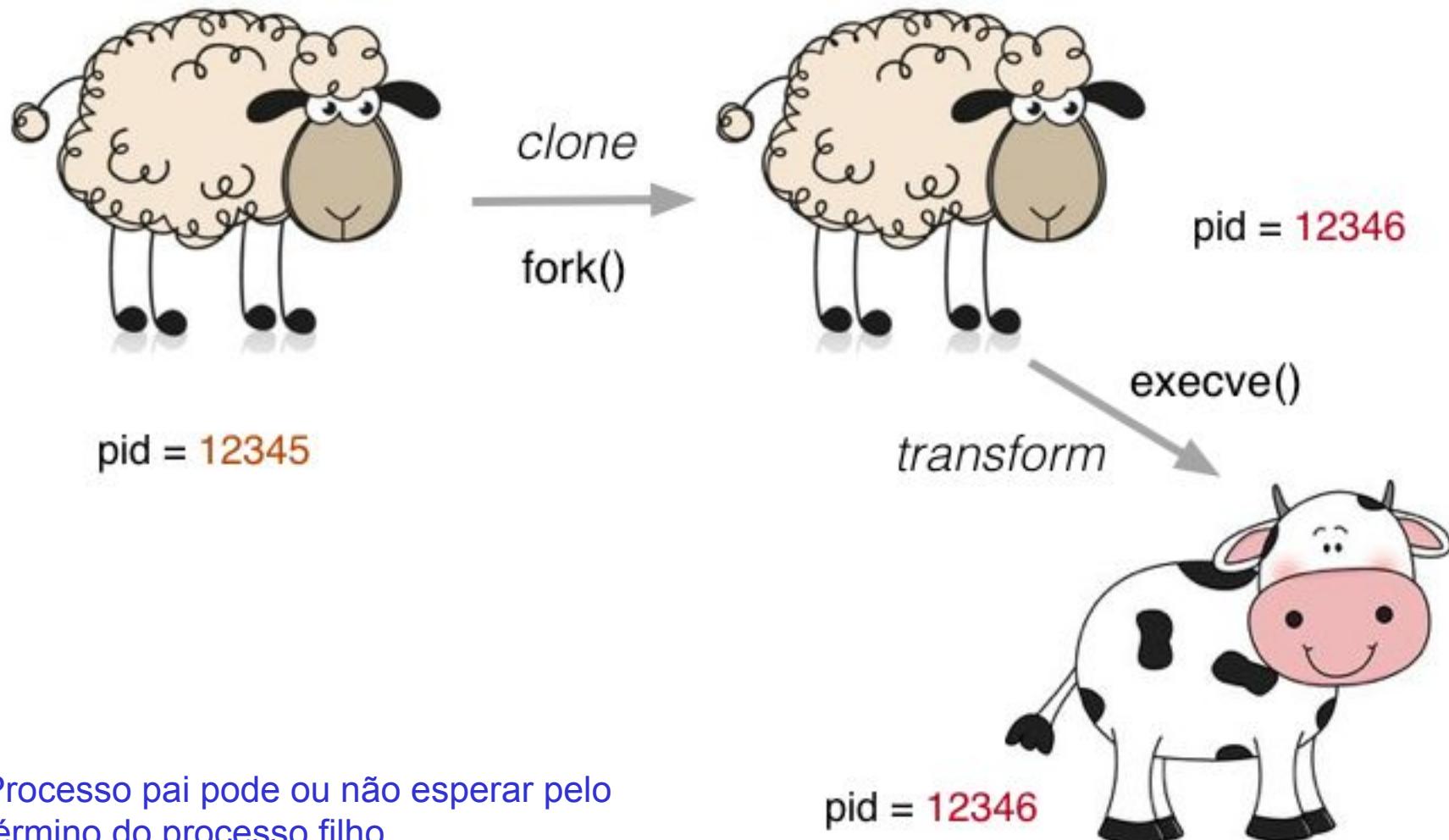
- imagem (código e estado de variáveis, constantes, etc.)
- pilha dos registros de ativação, e dados alocados dinamicamente (*heap*)
- estado da CPU (*contexto*)

O Unix mantém muitas informações sobre cada processo.

- Exemplo: arquivos abertos, tempo de criação, permissões, etc.

Na inicialização do sistema, alguns processos do sistema são criados (mas executam em modo usuário): `init`, `login`, `shell`, `inetd`

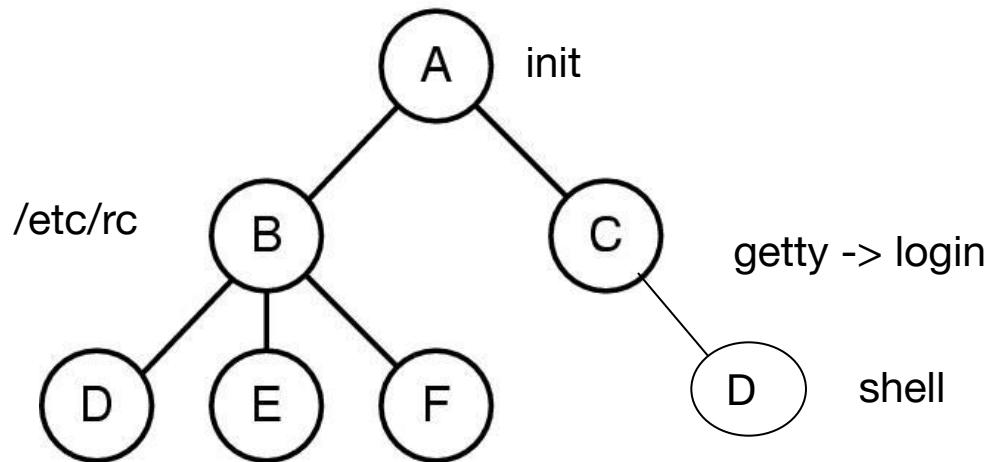
Criação de processos



Processo pai pode ou não esperar pelo término do processo filho.
Se não for esperar, diz-se que processo filho executa em 2º. Plano. (background)

Processos

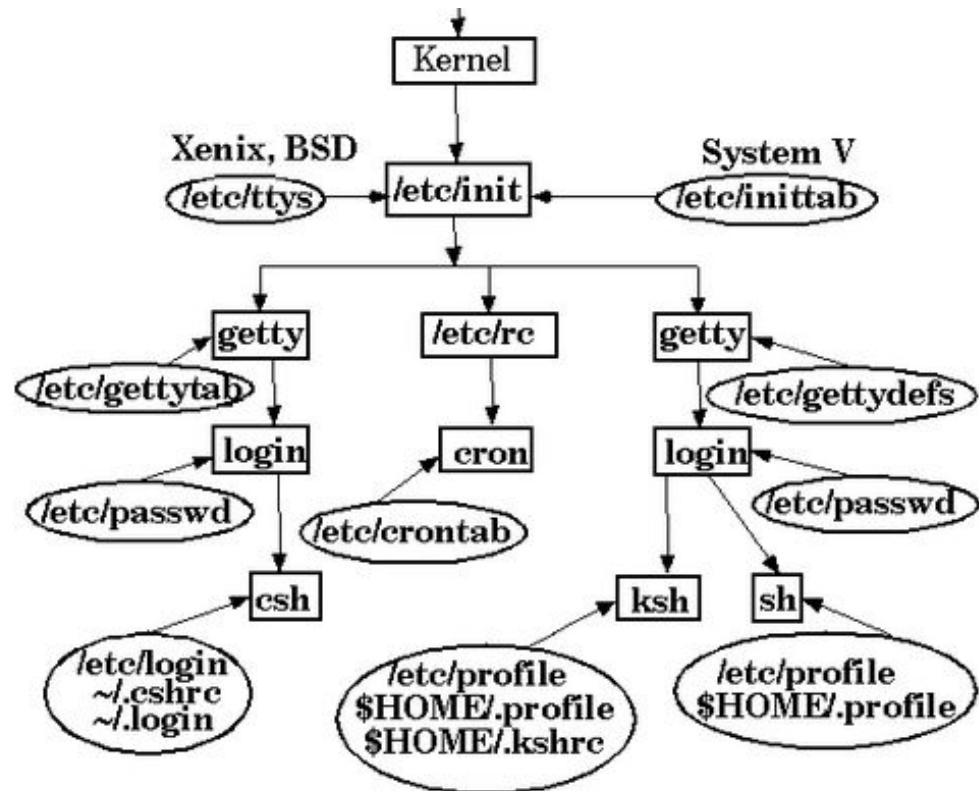
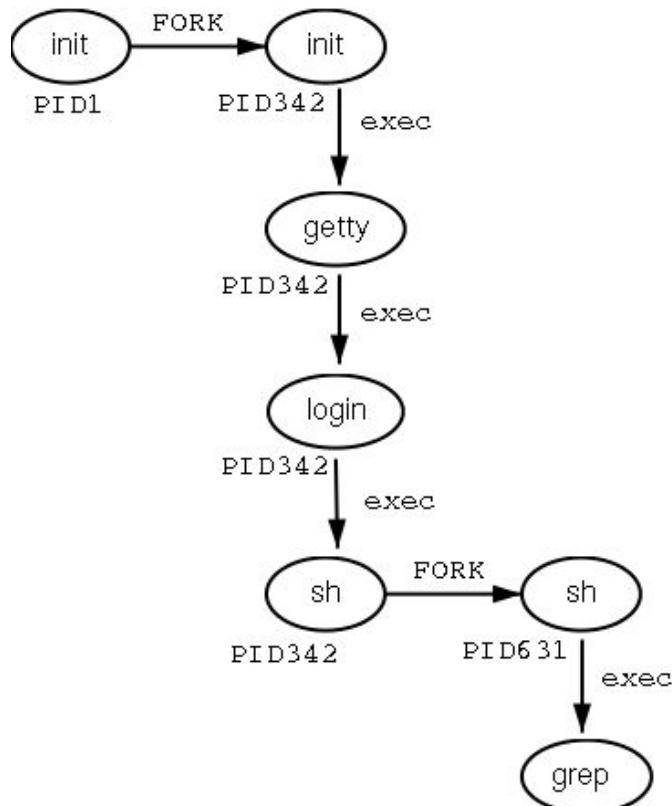
- Estabelece-se uma hierarquia entre processos
- Processos criados por um mesmo usuário mantém um parentesco, e podem se controlar mutuamente (suspending, terminar, etc.)



- Exemplo:
 - A = init, B = /etc/rc (script que inicia diversos daemons)
 - C = getty → login → shell → comandos do usuário
- Obs: A maioria dos S.O. executa vários processos concorrentes

Processos criados logo após o boot UNIX/Linux

Parentesco padrão de processos na inicialização



Parentescos de processos nas versões BSD e System V

“Bootstrap” ou boot



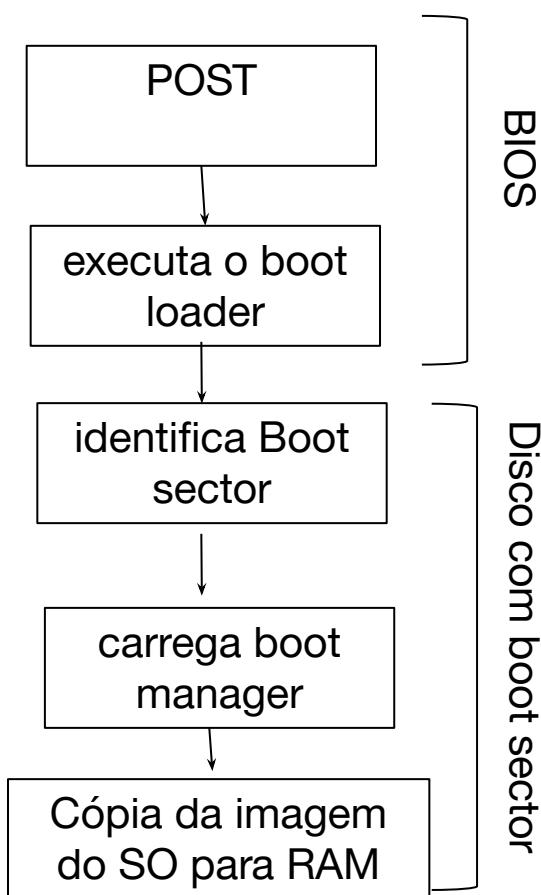
Barão de Münchhausen (autor de aventuras e conhecido mentiroso do século XIX)

Atribui-se a ele a criação do termo "bootstrap", como forma de explicar como ele escapou de um pântano (puxando-se pelos cadaços da própria bota).

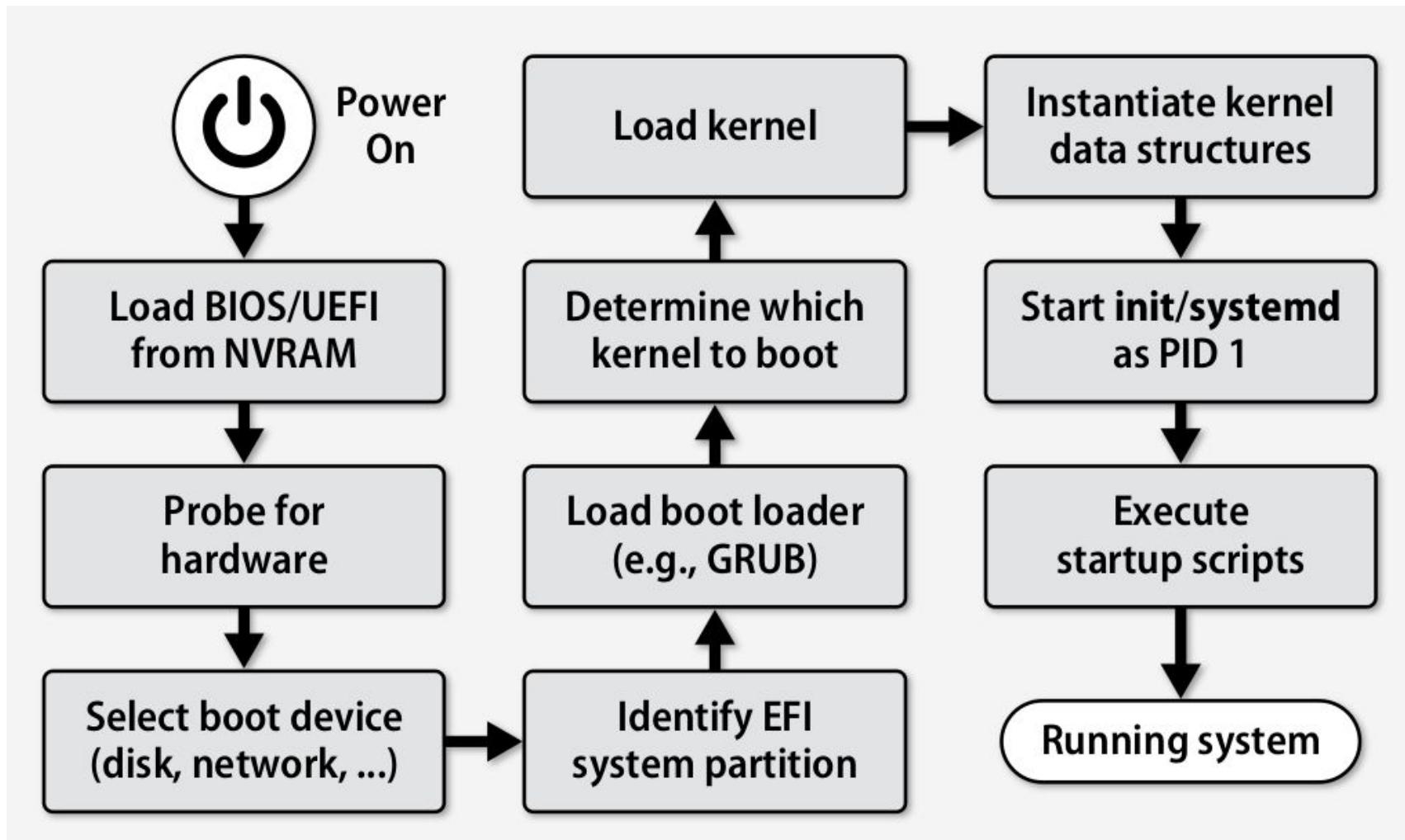
Carregamento do núcleo do SO

- *Sistema de ativação/ boot* carrega o sistema operacional para a memória principal
- A imagem precisa estar em uma partição do disco.
- Ao ligar o computador, o POST (Power-On Self Test) verifica se há algum problema no hardware
- Se tudo ok, carrega-se um boot manager
- Em seguida, o *boot manager* verifica qual é o disco/periférico indicado com um setor de boot (este contém um programa para carregar a imagem do sistema operacional)

Componentes do Sistema de Boot



Passo a passo do processo do boot do Unix

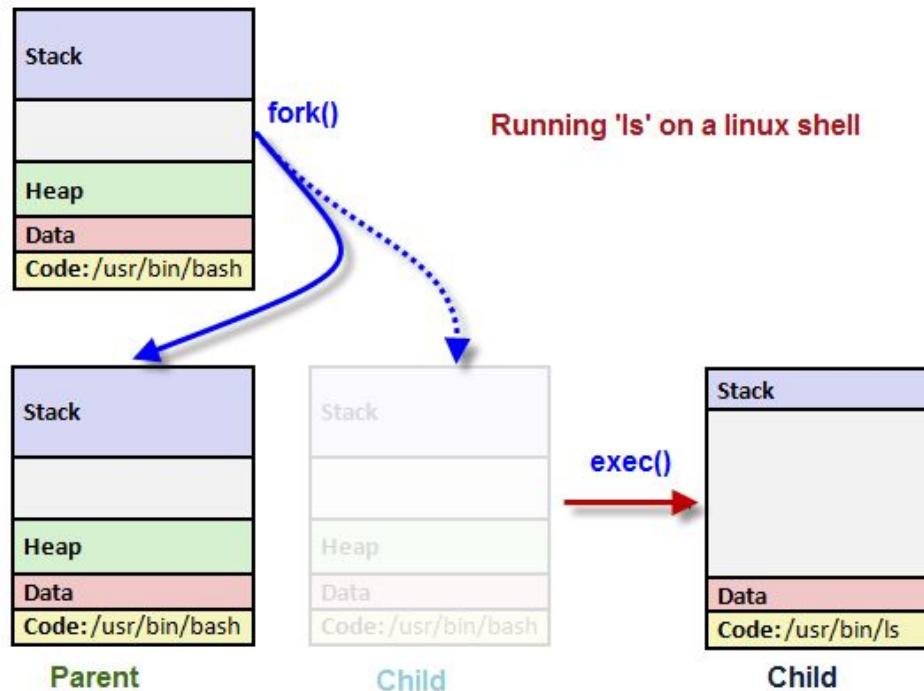


GRUB = Grand Unified Bootloader é um bootloader para Linux e outras versões do UNIX. É carregado em memória e entra em ação depois que BIOS encerrou todos os testes de hardware.

Chamadas de Sistema: Gerenciamento de Processos

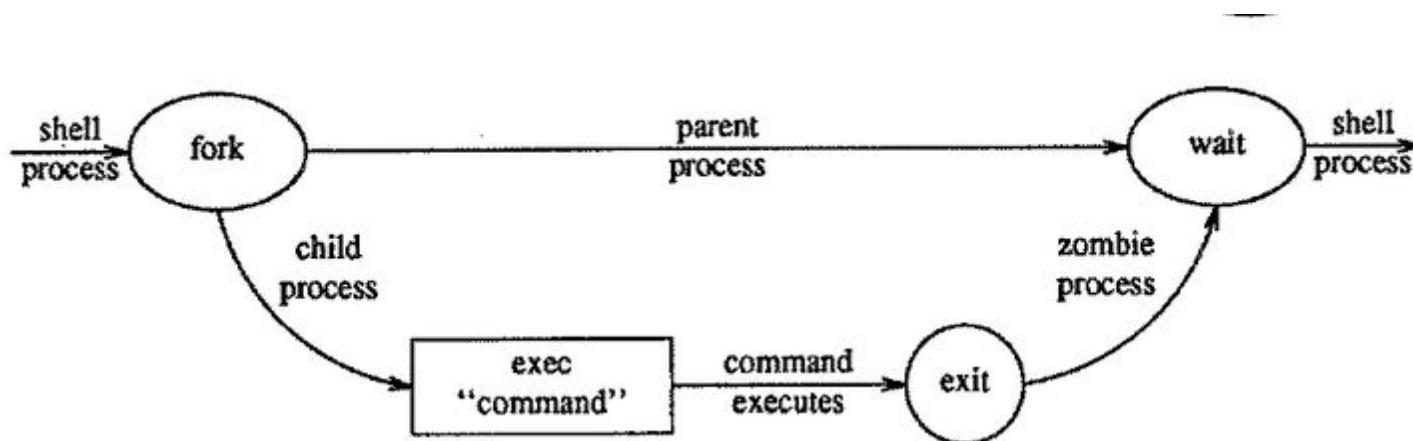
Gerenciamento de processos

Chamada	Descrição
pid = fork()	Crie um processo filho idêntico ao processo pai
pid = waitpid(pid, &statloc, options)	Aguarde um processo filho terminar
s = execve(name, argv, environp)	Substitua o espaço de endereçamento do processo
exit(status)	Termine a execução do processo e retorne o estado



Fork/Exec

- **pid = fork()** cria uma cópia do processo pai, que inicia execução na instrução seguinte ao fork (processo pai e filho terão PIDs diferentes)
- O **exec(command, parameters)** carrega e inicia execução de um novo programa executável (“command”) no lugar do atual
- **Pid = wait(pid, &status)** faz o processo pai esperar pelo término do processo filho.



Criando processos com fork/waitpid/execv

```
int main(void) {
    int mypid, pid, status;
    pid = fork();
    if (pid!=0) { //Pai
        mypid = getpid();
        waitpid(-1, &status, 0);
        if ( !WIFEXITED(status) )
            printf("O processo filho nao terminou
corretamente!\n");
        printf("Parent PID: %d\n",mypid);
    }
    else { //Filho
        mypid = getpid();
        printf("Child PID%d\n",mypid);
        exit(3);
    }
    return 0;
}
```

Criando processos com fork/waitpid/execv

```
int main(void) {
    int mypid, pid, status;
    pid = fork();
    if (pid!=0) { //Pai
        mypid = getpid();
        waitpid(-1, &status, 0);
        if ( !WIFEXITED(status) )
            printf("O processo filho nao terminou
corretamente!\n");
        printf("Parent PID: %d\n",mypid);
    }
    else { //Filho
        char * argv[2] = {"meuprograma",NULL};
        execv ("meuprograma", argv);
        exit(3);
    }
    return 0;
}
```

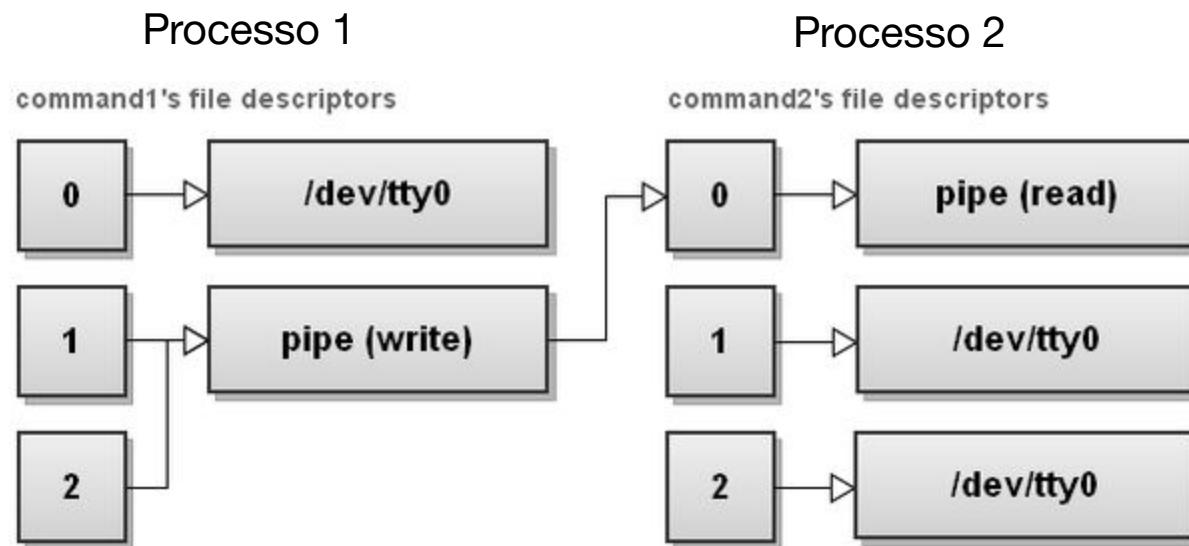
Processos: Redirecionamento de E/S

Todos os processos em primeiro plano possuem:

- **STDOUT** (standard output), default é a tela
- **STDIN** (standard input), default é o teclado
- **STDERR** (standard error), default é também a tela
- Shell permite redirecionamento de STDIN/STDOUT.
Exemplos:
 - `who > file,`
 - `date >> arquivo`
 - `sort < infile > outfile`
- Um pipe é uma conexão entre STDOUT e STDIN de dois processos filho
 - Ex: `who | sort` – saída de `who` será exibida ordenada na tela

Pipes

- Um pipe é uma conexão entre STDOUT e STDIN de dois processos filho
 - Ex: `who | sort` - saída de `who` será exibida ordenada na tela



Saida do utilitário ps

```
dhcpdip30:~ endler$ ps -elf
UID PID PPID   F CPU PRI NI   SZ RSS WCHAN S      ADDR TTY      TIME CMD          STIME
 0  1  0 4006 0 48 0 2472652 9144 -  Us      0 ??    1:10.13 /sbin/launchd 1Jan01
 0 40  1 4004 0 4 0 2451476 1880 -  Ss      0 ??    0:36.60 /usr/sbin/syslog 1Jan01
 0 41  1 4004 0 63 0 2478388 7064 -  Ss      0 ??    0:07.94 /usr/libexec/Use 1Jan01
 0 43  1 4004 0 31 0 2743636 1360 -  Ss      0 ??    0:00.90 /usr/local/bin/w 1Jan01
...
501 2121  1 4004 0 31 0 2446896 480 -  S      0 ??    0:00.04 /usr/libexec/USB 13Aug19
501 2152  1 4004 0 4 0 2521500 5824 -  S      0 ??    0:04.29 /System/Library/ 13Aug19
501 2971  1 4004 0 4 0 2470236 724 -  Ss      0 ??    0:00.04 /System/Library/ 15Aug19
 0 2977 2937 4104 0 31 0 2437980 152 -  S      0 ??    0:00.28 /Library/Dropbox 15Aug19
 0 2978 2977 4 0 31 0 2446172 856 -  S      0 ??    0:01.07 /Library/Dropbox 15Aug19
501 2979 2978 104 0 31 0 2437980 72 -  S      0 ??    0:00.70 /Library/Dropbox 15Aug19
501 2980  1 4004 0 4 0 2482748 1192 -  Ss      0 ??    0:00.26 /Applications/Dr 15Aug19
501 3291 227 4004 0 31 0 3205688 116184 -  S      0 ??    1:17.91 /Applications/Go Mon09AM
501 3304  1 84004 0 4 0 2514468 22796 -  S      0 ??    0:04.08 /System/Library/ Mon09AM
501 3305  1 84004 0 4 0 2486272 17740 -  S      0 ??    0:02.15 /System/Library/ Mon09AM
501 3527  1 4004 0 4 0 2446956 5312 -  Ss      0 ??    0:00.11 /Applications/Dr Tue09AM
 0 3545  1 4004 0 31 0 2470472 2808 -  Ss      0 ??    0:00.04 /System/Library/ Tue09AM
501 3563 227 4004 0 31 0 3111688 123700 -  S      0 ??    0:09.02 /Applications/Go Tue09AM
501 3660 227 4004 0 31 0 3053748 66928 -  S      0 ??    0:03.12 /Applications/Go Tue10AM
 89 3715  1 84004 0 4 0 2452332 8004 -  S      0 ??    0:00.04 /System/Library/ Tue12PM
 0 3719  1 4004 0 63 0 2483392 15196 -  Ss      0 ??    0:00.21 /usr/sbin/ocspd Tue12PM
501 3728  1 84004 0 4 0 2486872 17904 -  S      0 ??    0:00.28 /System/Library/ Tue12PM
501 3735  1 4004 0 4 0 2470312 5772 -  S      0 ??    0:00.03 /System/Library/ 10:02AM
501 3865 227 4004 0 31 0 3097612 153928 -  U      0 ??    0:04.97 /Applications/Go 10:09AM
501 3866 227 4004 0 31 0 2847780 30140 -  S      0 ??    0:00.26 /Applications/Go 10:15AM
501 3867  1 4084 0 47 0 2557780 25168 -  S      0 ??    0:00.99 /Applications/Ut 10:15AM
 0 3869 3867 4106 0 31 0 2446444 2680 -  Ss      0 ttys000 0:02.37 login -pf endler 10:15AM
501 3870 3869 4006 0 31 0 2442588 1268 -  S      0 ttys000 0:00.03 -bash          10:15AM
 0 3875 3870 4106 0 31 0 2432948 796 -  R+      0 ttys000 0:00.00 ps -elf        10:16AM
dhcpdip30:~ endler$
```

Conceitos fundamentais

- Processo
- Parentesco entre processos
- stdin, stout, sterr
- Sinais
- Arquivo
- Diretório
- Usuário
- Grupo
- Permissões

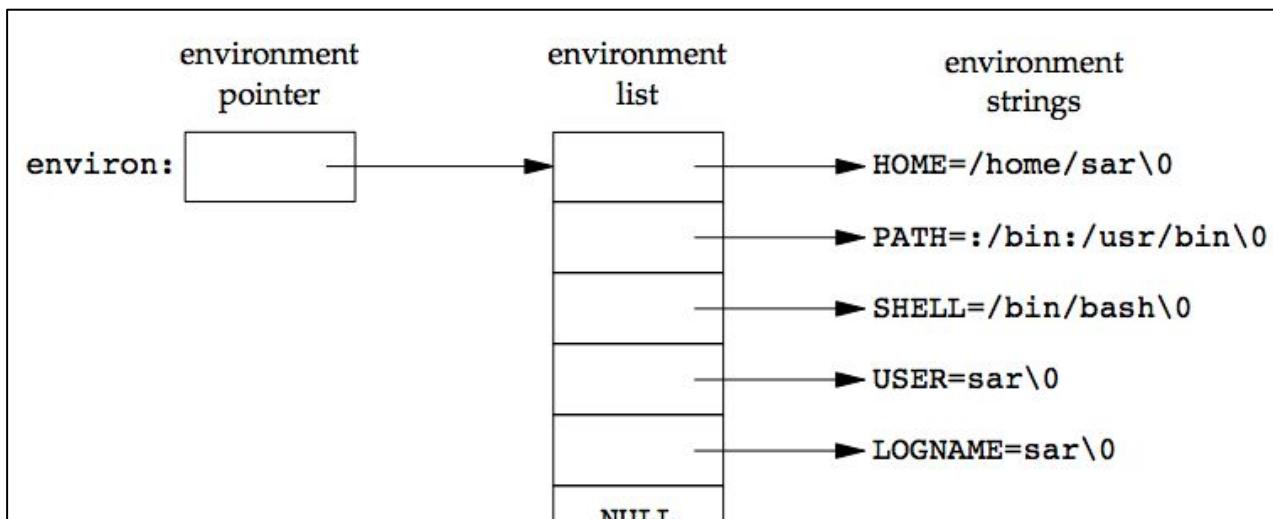
Variáveis de Ambiente

Cada processo tem associado a ele várias variáveis de ambiente, que definem o seu ambiente de execução no sistema (na variável `environ`).

A variável PATH, é usada pela shell para percorrer diretórios a procura do programa executável

Outras variáveis:

- HOME caminho até o diretório *home* do usuário
- PAGER programa para exibir man pages
- PWD directorio corrente
- GROUP em qual grupo o usuário se encontra
- USER login name
-



Variáveis de Ambiente

Através da shell, é possível criar novas variáveis de ambiente e/ou modifica-las. Programas podem consultar e modificar as variáveis através de `getenv()`, `putenv()`, `setenv()`.

```
#include <stdio.h>
#include <stdlib.h>

int main () {
    printf ("PATH : %s\n", getenv ("PATH")) ;
    printf ("HOME : %s\n", getenv ("HOME")) ;
    printf ("ROOT : %s\n", getenv ("ROOT")) ;

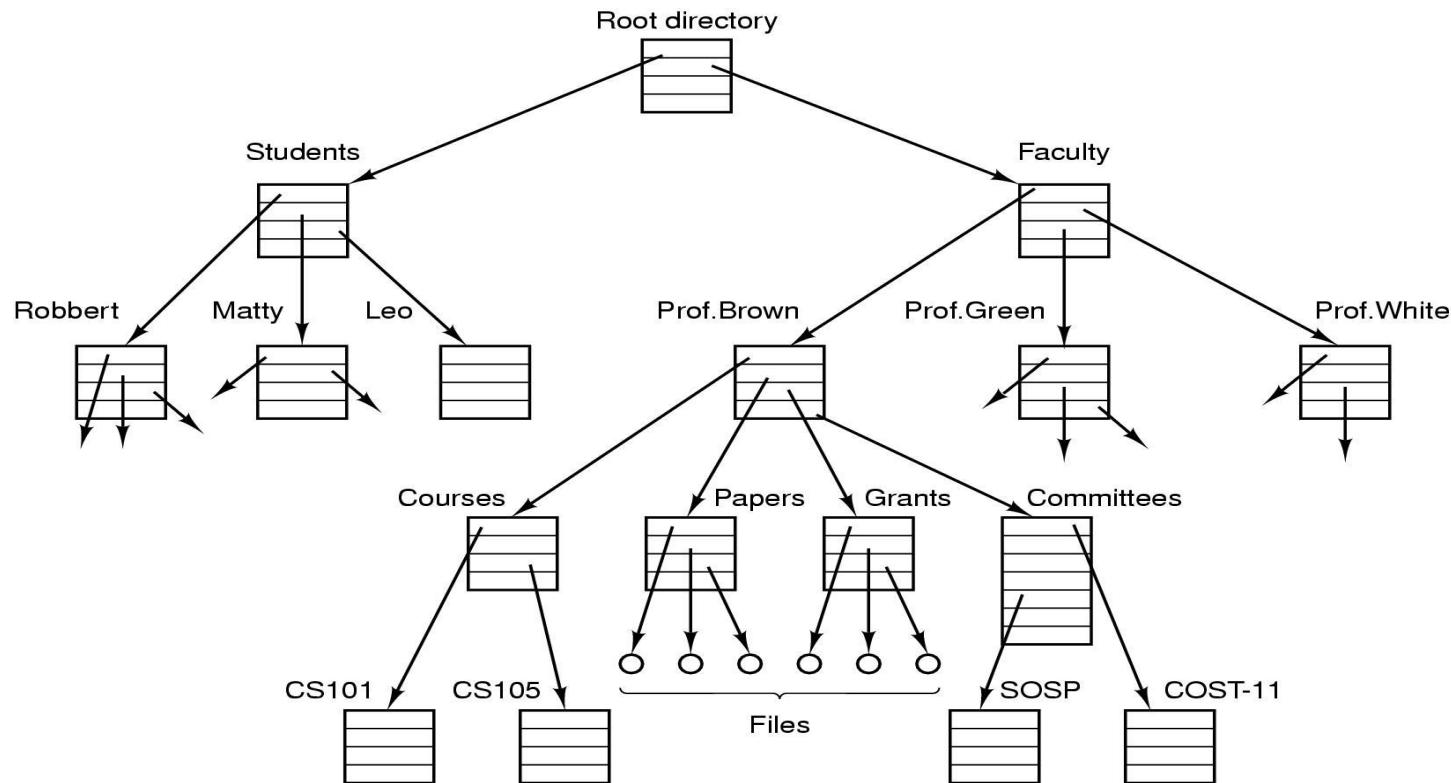
    return (0);
}
```

Sistema de Arquivos

Arquivo = conceito abstrato para *repositório durável* de informação binária, que é *independente do meio*

Além do conteúdo, arquivo possui uma série de atributos (meta-dados)

Pastas aninhadas permitem uma hierarquia de arquivos.



Chamadas de Sistema: Acesso a Arquivos

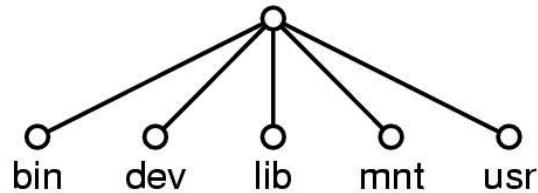
Gerenciamento de arquivos

Chamada	Descrição
fd = open(file, how, ...)	Abra um arquivo para leitura, escrita ou ambas
s = close(fd)	Feche um arquivo aberto
n = read(fd, buffer, nbytes)	Leia dados de um arquivo para um buffer
n = write(fd, buffer, nbytes)	Escreva dados de um buffer para um arquivo
position = lseek(fd, offset, whence)	Mova o ponteiro de posição do arquivo
s = stat(name, &buf)	Obtenha a informação de estado do arquivo

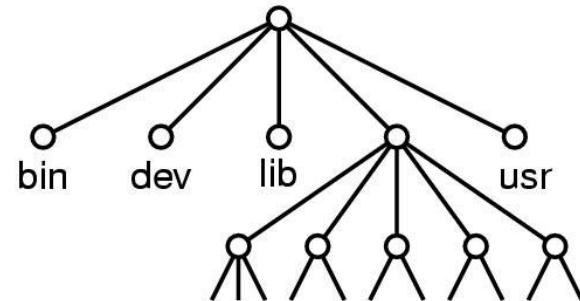
Chamadas de Sistema: Gerenciamento de Diretórios

Gerenciamento do sistema de diretório e arquivo

Chamada	Descrição
s = mkdir(name, mode)	Crie um novo diretório
s = rmdir(name)	Remova um diretório vazio
s = link(name1, name2)	Crie uma nova entrada, name2, apontando para name1
s = unlink(name)	Remova uma entrada de diretório
s = mount(special, name, flag)	Monte um sistema de arquivo
s = umount(special)	Desmonte um sistema de arquivo



(a)



(b)

Sete tipos de arquivos padrão do Unix

O POSIX define os seguintes tipos:

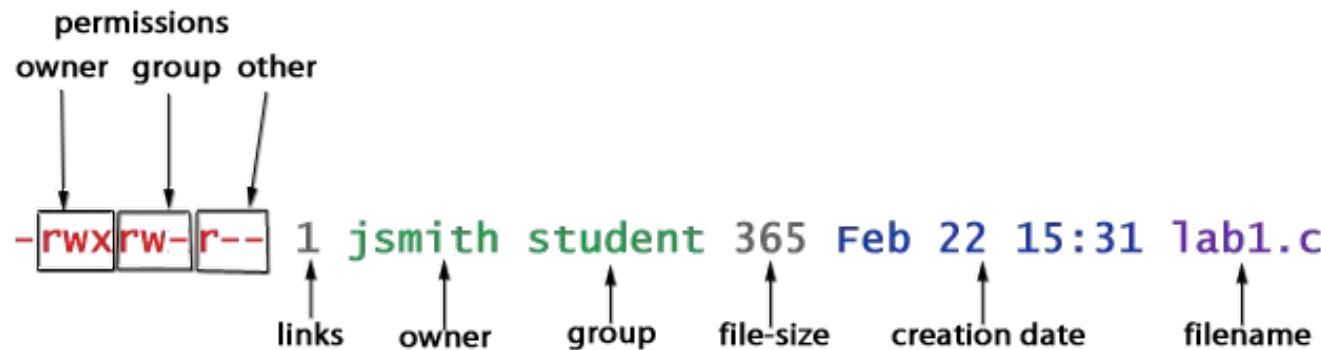
- convencional (regular),
- diretório,
- link simbólico,
- especial FIFO (named pipe),
- especial de bloco,
- especial de caractere e
- socket

Todos são acessados pelas mesmas chamadas:

`open()`, `read()`, `write()`, `close()`, `ioctl()`

Atributos de Arquivos/Diretórios

Todo arquivo, diretório ou dispositivo têm atributos visíveis ao usuário:



ou mantidos pelo sistema:

- número de blocos em disco
- ponteiro para o primeiro bloco no disco
- última data de modificação/acesso,
- número de referências ao arquivo
- etc

Sistema de Arquivos: alguns utilitários

- **pwd** – mostra o diretório corrente
- **cd <dir>** troque o diretório corrente
- **ls <dir>** - lista os arquivos do diretório corrente
- **ls -a/l** - lista tb os arquivos escondidos dot-files (-a), lista todos os detalhes (-l)
- **cp <old file> <new file>** - cópia
- **mv <old file> <new file>** - move (renomeia)
- **rm <file>** -remove um arquivo
- **mkdir <new dir name>** -cria um diretório
- **rmdir <dir>** -remove um diretório vazio
- **rm -r <dir>** remoção recursiva
- **cp -r <dir>** - cópia recursiva

Filosofia UNIX é ser um sistema extensível: quase todos os comandos são programas utilitários, chamados pela shell

Através de redirecionamento de E/S é possível compor comandos mais complexos

Usuários e Grupos

Grupos de usuários facilitam a definição de permissões (read, write, execute)

- Cada processo herda o userID e grpID do seu processo pai
- após o login, todos os processos tem user definido
- Somente administrador (root) pode cadastrar novos usuários e grupos
- Lista de usuários contida em /etc/passwd e todos os grupos em /etc/groups (com conteúdo cifrado)
-
- Cada usuário pode pertencer a um ou mais grupos
`useradd, groupadd, groupmod, grouprem`

Ex: root pertence aos grupos `root, bin, daemon, sys, ...`

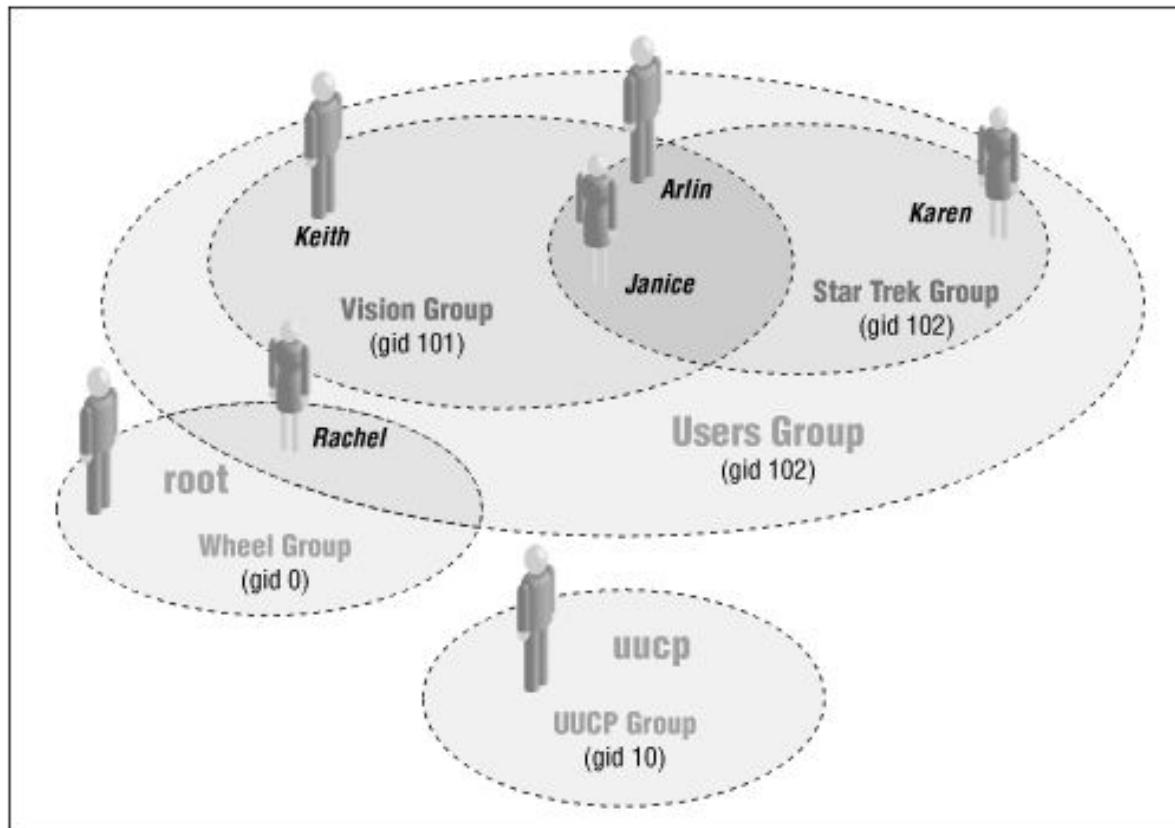
- comando groups para descobrir em quais grupos um user está.

```
# groups root
root : root bin daemon sys adm disk wheel
```

Usuários e Grupos

Usuário pode pertencer a vários grupos,

- mas a cada momento só atua como membro de um grupo:



- Usuário pode trocar usando **newgrp**
groups user lista todos os grupos aos quais user pertence

Os “Usuários” do Sistema

Além dos usuários humanos, o UNIX também usa nomes de usuário especiais para uma variedade de funções do sistema.

- **root**: o superusuário, que executa funções de contabilidade e de baixo nível do sistema.
- **daemon** ou **sys**: que lida com alguns aspectos da rede. Esse nome de usuário também está associado a outros utilitários, como os spoolers de impressão, em algumas versões do UNIX.
- **agent**: que lida com aspectos do correio eletrônico.
- **guest**: que é usado para visitantes do site acessarem o sistema.
- **ftp**: usado para acesso FTP anônimo.
- **uucp**: que gerencia o sistema UUCP.
- **news**: que é usado para notícias da Usenet.
- **lp**: que é usado para o sistema de impressão

Permissões de Arquivos e Diretórios

- Cada arquivo/diretório tem atributos de controle de acesso:
 - Bits rwx rwx rwx (relativos ao dono, seu grupo e outros), onde “1/0” significam com/sem permissão
- **chmod g+r <filename>** - acrescentando direto de leitura aos membros do grupo
- **chmod 744 <filename>** - direitos plenos apenas para dono, demais apenas leitura
- **chown <user> <filename>** - mudando o dono do arquivo
- **chgrp <group> <filename>** - alterando a qual grupo o arquivo pertence

Links simbólicos

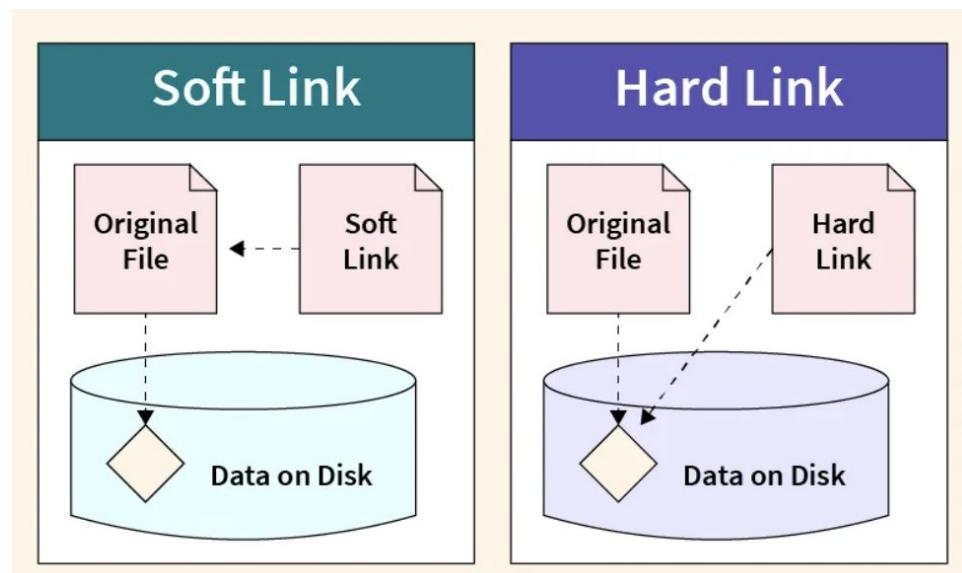
É uma forma de criar um atalho para um arquivo dentro do diretório corrente. Exemplo:

- `ln -s <arquivo-original> <novo nome>`

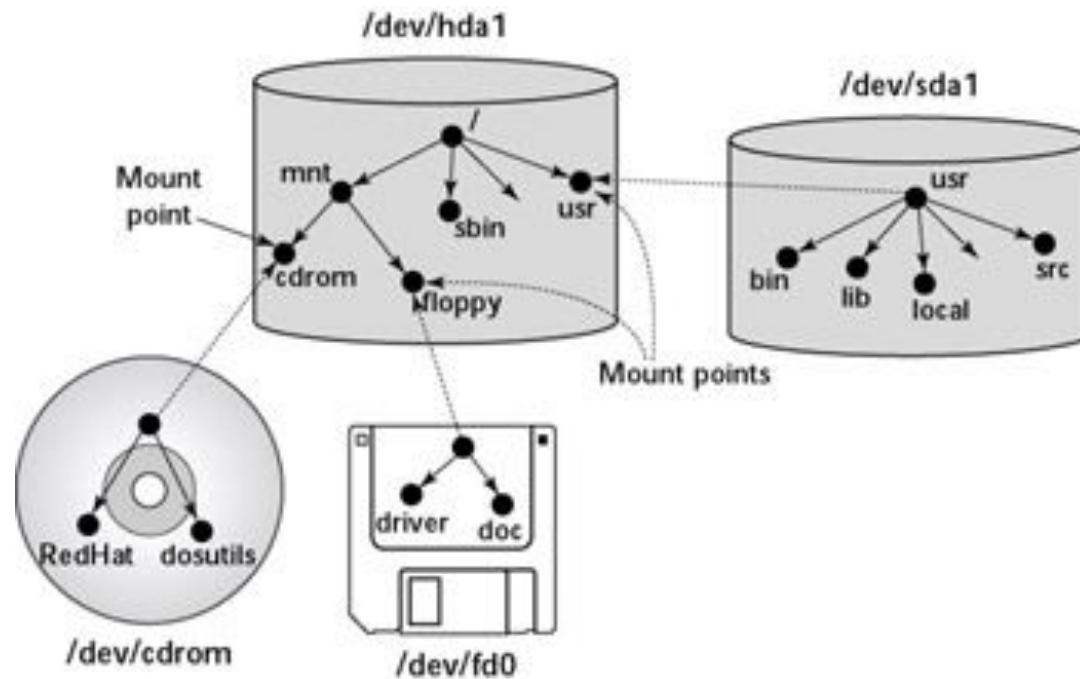
```
>ls -l .forward*
-rw-r--r-- 1 darin  csua   .forward
lrwxr-xr-x 1 darin  csua   .forward.link@ -> .forward
```

O primeiro “1” nos bits de permissão indica que é um link simbólico

Existe Link hard e
link soft



Conexão entre Sist. de Arquivos (montagem)

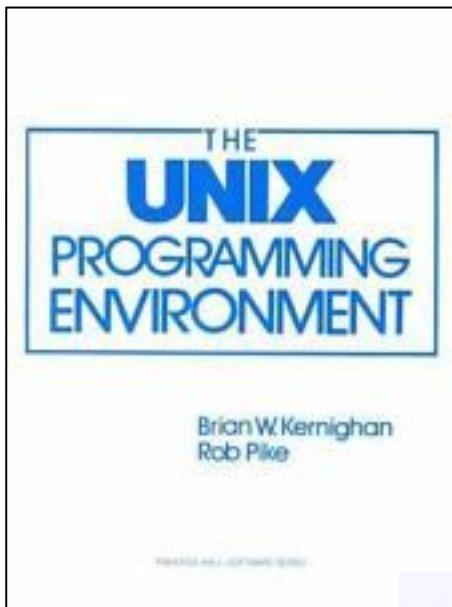


- `mount /dev/hda1/usr /dev/sda1`
- `mount /dev/hda1/cdrom /dev/cdrom`
- `mount /dev/hda1/floppy /dev/fd0` – incorpora a árvore do floppy ao sistema de arquivos principal (**no mounting point**)
- `umount b` - desfaz a conexão

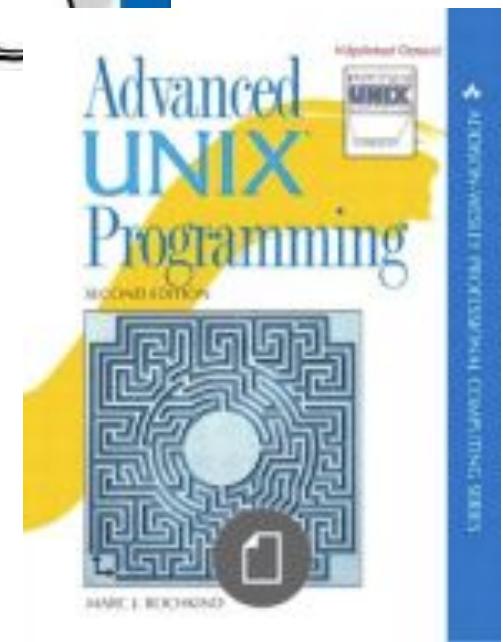
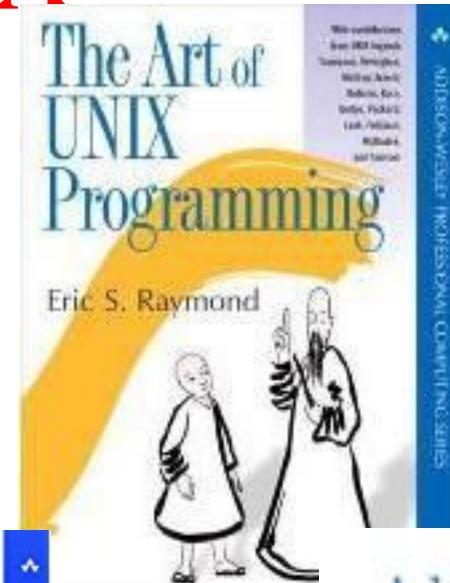
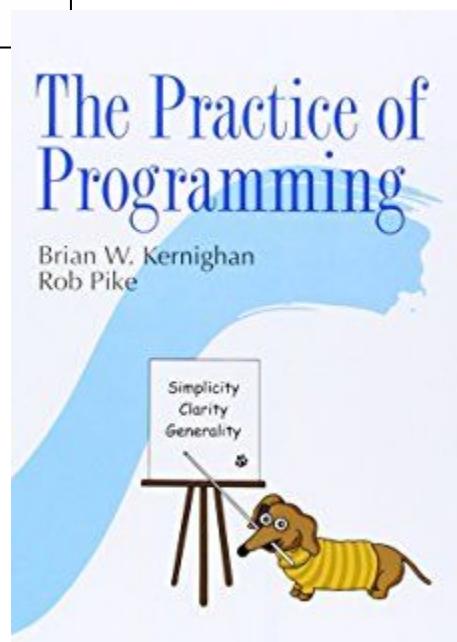
Outros comandos úteis

- **grep** *<expr> <arq>* - busca as linhas que contém *<expr>* no arquivo
- **diff** *<arq1> <arq2>* - mostra as diferenças entre o conteúdo dos dois arquivos
- **vi** ou **pine** - editores de texto simples
- **sed** , **awk** – filtros/ processadores de texto
- **perl** - linguagem para scripts
- **make** – automatizar a geração de programas a partir de várias componentes interdependentes (dependências em Makefile), e verificando se alguma foi modificada
- **man** *<termo>* – manual on-line
- **ftp** *<user@rem-host>*- transferência de arquivos
- **rsh** *<user@rem-host>* - login remoto

Programação usando scripts UNIX



The Unix Programming environment,
Kernighan & Pike



Princípios básicos de funcionamento

Boot do Sistema

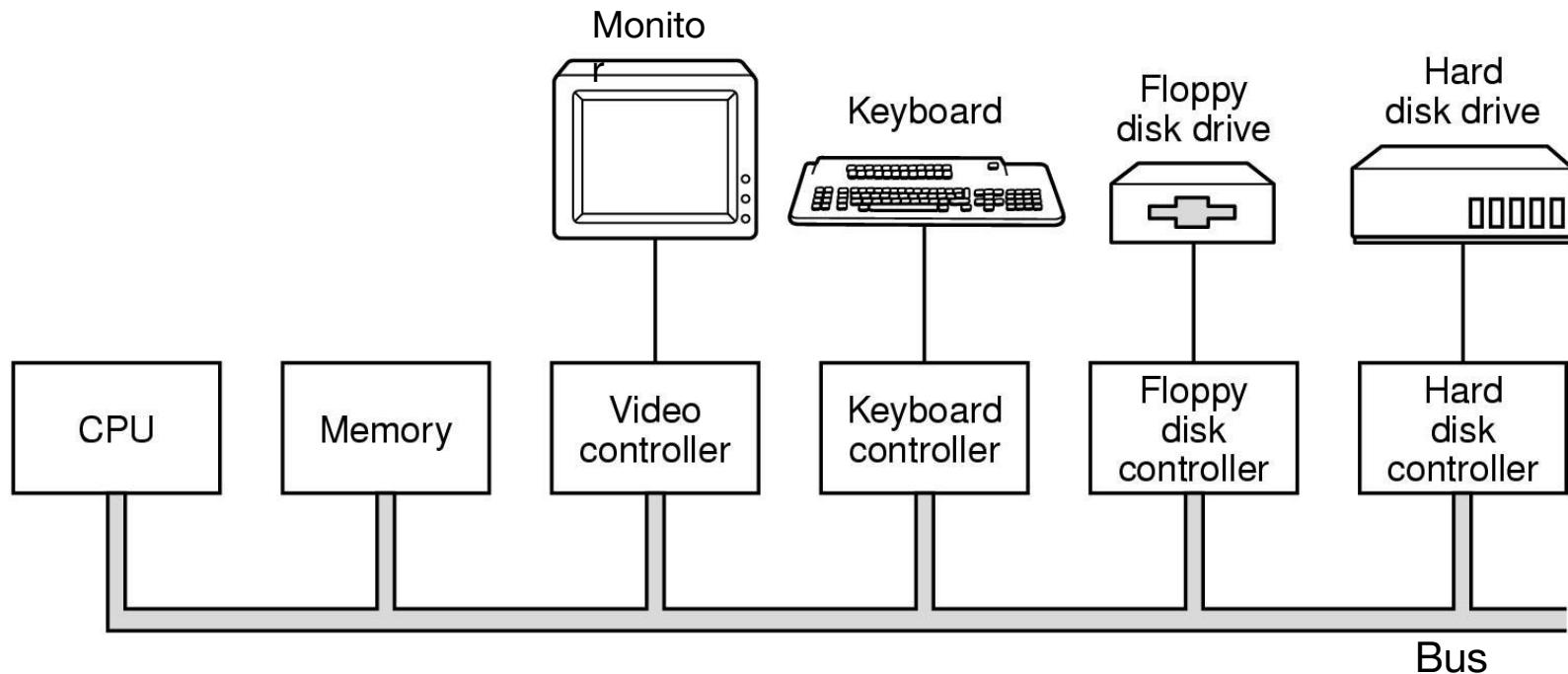
Interrupções de hardware

Execução de Chamadas de Sistema

Alocação de memória

Chamadas de Sistema típicas

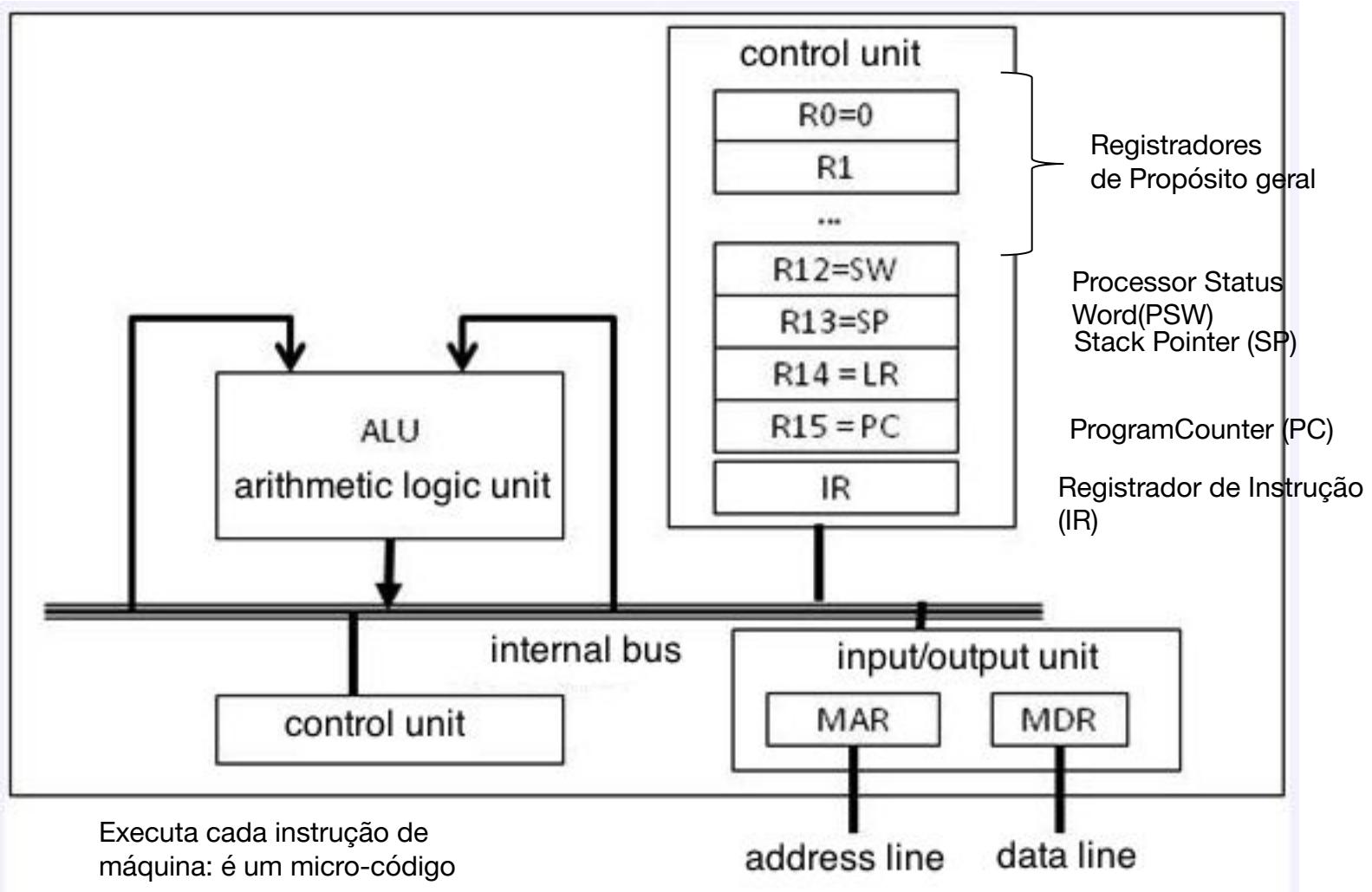
Visão simplificada do Hardware



Através do barramento trafegam:

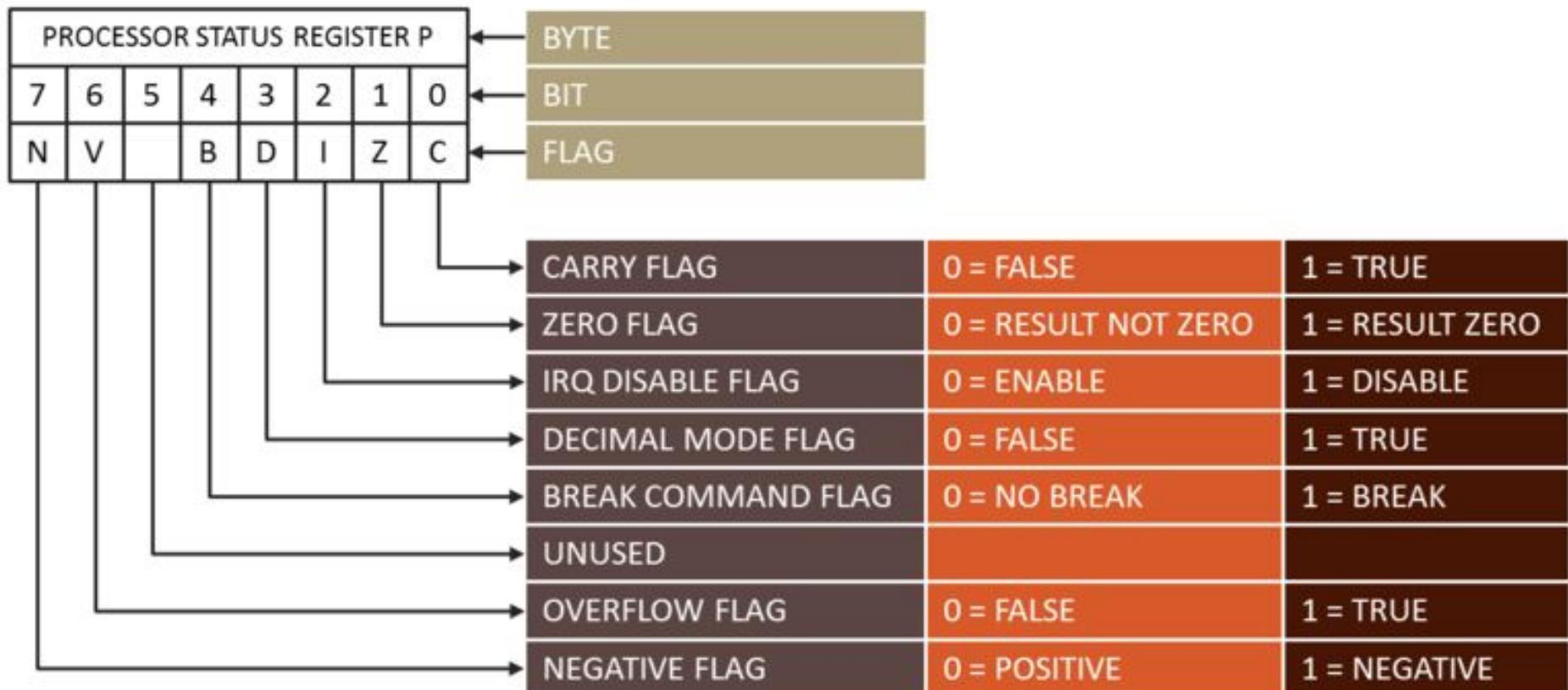
- Dados
- Endereços (Memória e portas de E/S)
- Instruções de máquina (p. CPU controladores de E/S)

CPU: Uma breve revisão

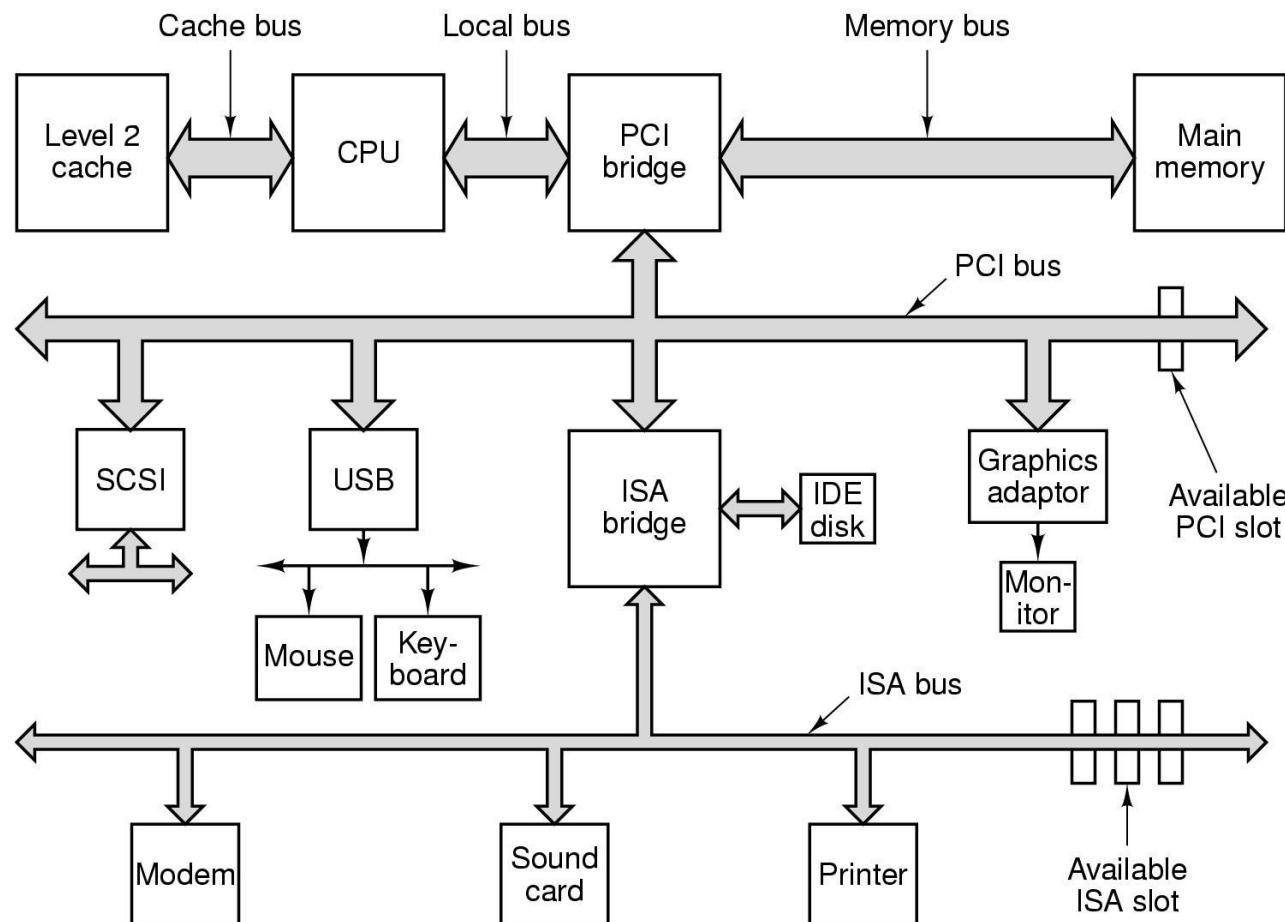


Processor Status Word

- Indica também o estado da CPU



Arquitetura do Hardware e Interrupções



Barramentos com diferentes taxas de transmissão

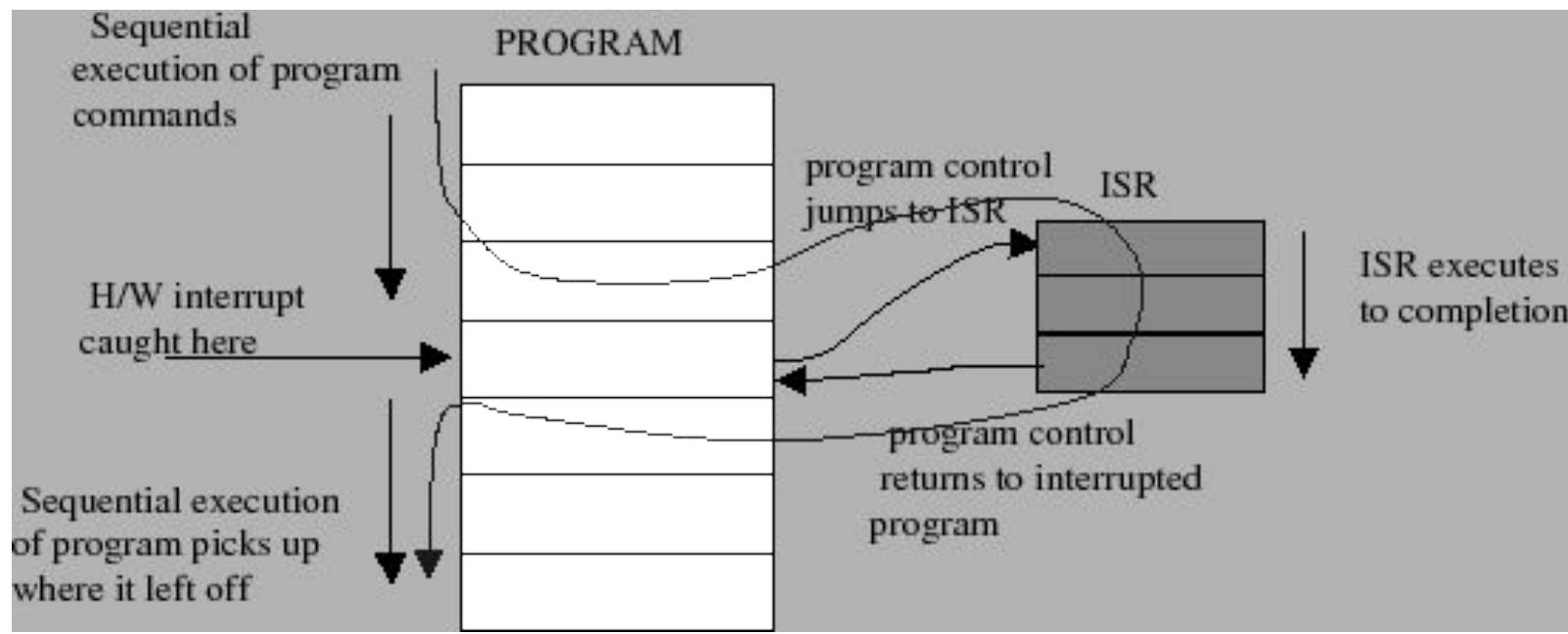
Toda interação periférico-CPU através de interrupções de HW

Interrupções de Hardware

Interrupções são a forma de um dispositivo de HW avisar o núcleo que alguma ação precisa ser feita. **Ocorrem a qualquer momento!**

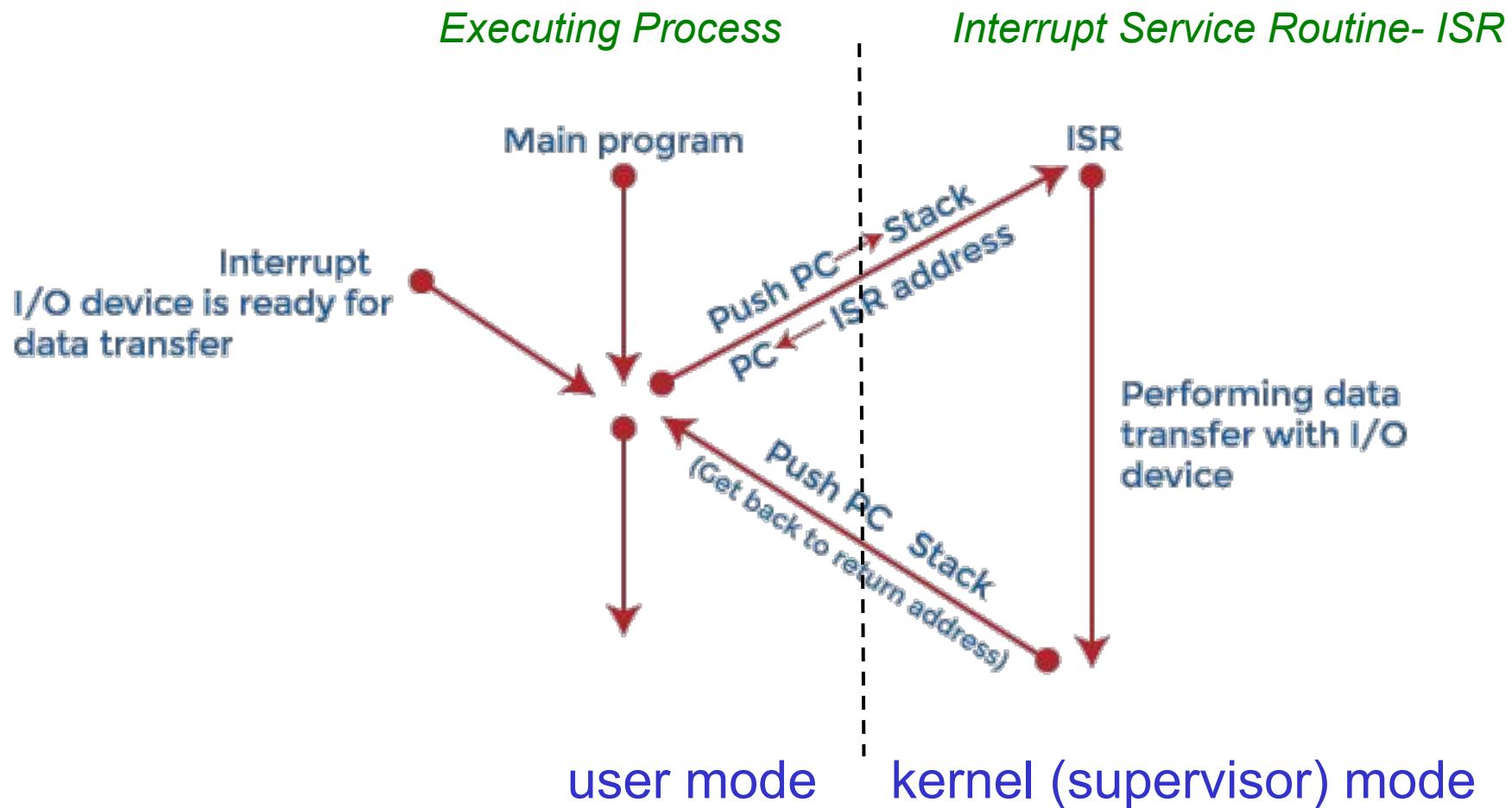
O processo corrente é interrompido e um **tratador de interrupção** (*Interrupt Service Routine- ISR*) é executado. Todos tratadores executam em modo kernel.

Existem tb interrupções de software (TRAP), interrupções de relógio (para time-sharing), e da MMU (para paginação)



Interrupções de Hardware

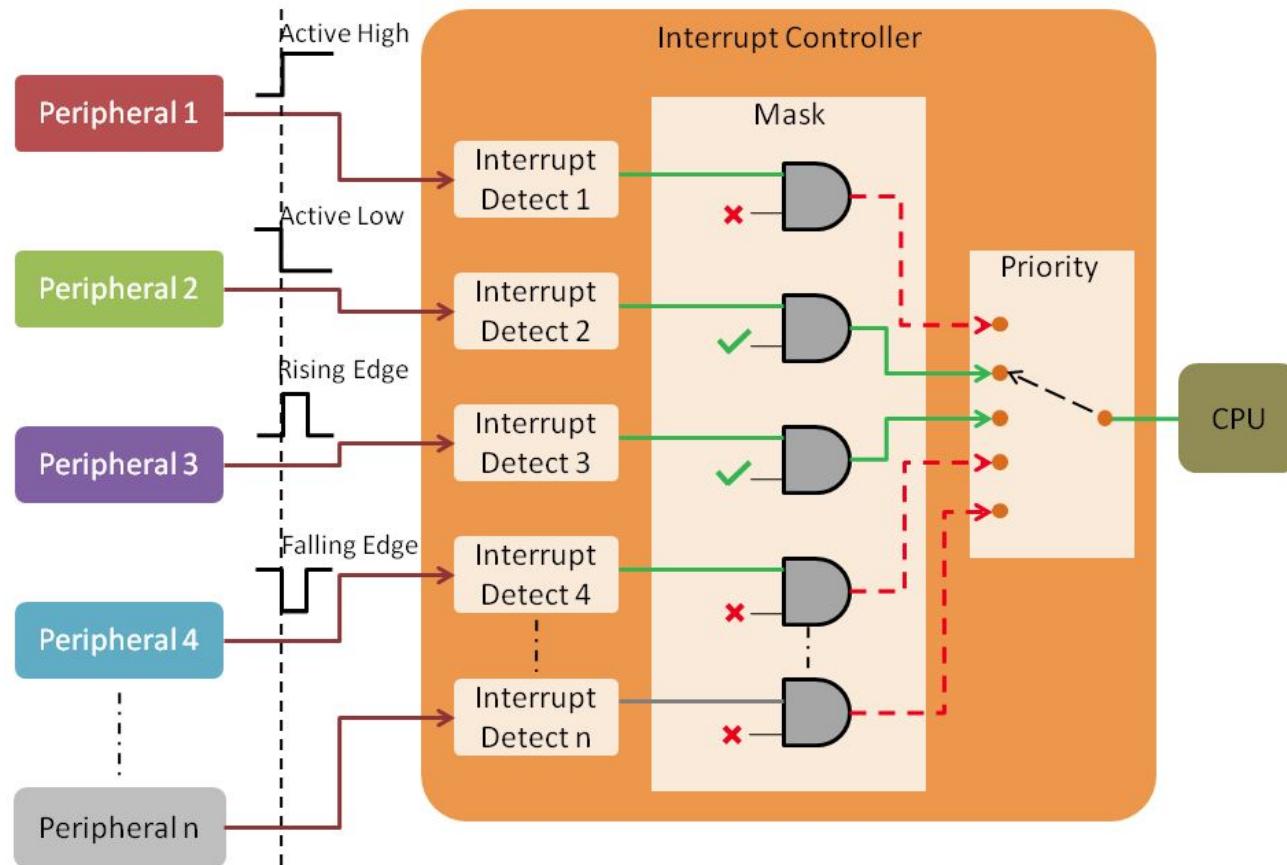
Troca de controle para tratamento da interrupção.



Controlador de Interrupções

Controlador da Interrupção é que recebe o sinais (elétricos) dos dispositivos, detecta as interrupções e repassa para a CPU as interrupções não mascaradas.

Interrupções têm prioridade e podem ser mascaradas (desativadas ativadas).



Níveis de Interrupção

O tratador de interrupção **executa em modo kernel**.

Geralmente, existem vários níveis de interrupção de hardware. Os mais comuns são:

- RESET – processador entra em processo de inicialização
- NMI – não mascarável (que não pode ser postergada)
- **IRQ nr – interrupções mascaráveis**
- ABORT – redireciona execução se uma exceção de hardware é detectada (violação de acesso à memória)

Algumas interrupções IRQ são mais prioritárias do que outras.

Por exemplo: clock interrupt é mais prioritária do que network interrupt.

Non-Maskable Interrupts (NMI)



High Priority Interrupts



Medium Priority Interrupts



Low Priority Interrupts

Interrupções não mascaráveis (Non Maskable Interrupts - NMI)

NMIs são interrupções com a prioridade mais alta que não podem ser desativadas ou mascaradas pela CPU.

São usadas para sinalizar eventos críticos que exigem atenção imediata, como:

- Falhas de hardware: Erros irrecuperáveis, como corrupção de memória ou superaquecimento.
- Temporizadores Watchdog: Para alertar o sistema se ele estiver preso em um loop ou apresentando um mau funcionamento.
- Depuração e diagnóstico: Para forçar um ponto de interrupção durante a depuração de um programa.

Exemplos:

- Watchdog: se um sistema não reiniciar o timer do watchdog em um tempo predefinido, ele gera uma NMI.
- Se o sistema detectar um erro irreparável na RAM de código de correção de erros (ECC), uma NMI poderá ser acionada.
- As NMIs podem ser acionadas por vários problemas de hardware, ex superaquecimento

Em contraste, interrupções mascaráveis (IRQ) podem ser desativadas ou ignoradas pela CPU

Requisição de interrupção (IRQ)

Uma requisição de interrupção (IRQ) é uma requisição para serviço, **enviada em nível de hardware**.

- Interrupções podem ser enviadas por linhas de hardware dedicadas ou através de um bus de hardware como um pacote de informações
- O Kernel recupera o número do IRQ e usando o vetor/lista de Rotinas de Serviço de Interrupção (ISRs) registradas, inicia a ISR correspondente
- O ISR ignora interrupções redundantes do mesmo IRQ, depois enfileira um manuseador deferido para terminar o processamento de interrupções
-

O arquivo `/proc/interrupts` relaciona o número de interrupções por CPU por dispositivo de E/S.

Os IRQs possuem uma propriedade de "afinidade" associada, `smp_affinity`, que define os núcleos específicos de uma CPU multi-core que podem executar o ISR para aquele IRQ

Int	Dispositivo	Int	Dispositivo
0	Cronômetro do sistema	9	Porta de comunicação COM3
1	Teclado	10	Porta de comunicação COM2
2	Controlador de interrupção	11	Ponte PCI (*)
4	Porta de comunicação COM1	12	Mouse porta PS/2 (*)
5	Placa de som (*)	13	Coprocessador numérico
6	Controlador de disco flexível	14	Controlador IDE/ESDI
7	Porta de Impressora LPT1	15	Controlador IDE/ESDI
8	CMOS/Relógio do sistema		(*) Opções não padronizadas

Dump de /proc/interrupts

Para um quad-core

	CPU0	CPU1	CPU2	CPU3		
20:	29825	9674	8921	8102	GIC	arch_timer
25:	0	0	0	0	GIC	MSM_L1
33:	2258	0	0	0	GIC	bw_hwmon
34:	0	0	0	0	GIC	MSM_L2
35:	0	0	0	0	GIC	apps_wdog_bark
39:	2722	1754	1635	1389	GIC	arch_mem_timer
61:	80	0	0	0	GIC	mxhci_hsic_pwr_evt
64:	5573	0	0	0	GIC	xhci-hcd:usbl
65:	4519	0	0	0	GIC	ksgl-3d0
74:	0	0	0	0	GIC	msm_iommu_nonsecure_irq
75:	0	0	0	0	GIC	msm_iommu_secure_irq, msm_iommu_secure_irq
76:	548	0	0	0	GIC	msm_vide
... (Omit some lines)		<u>IRQ number</u>	<u>PIC name</u>			<u>Device name</u>
436:	0	0	0	0	msmgpio	bluetooth_hostwake
437:	22	0	0	0	qpn-p-int	smb135x_chg_stat_irq
438:	1	0	0	0	msmgpio	max170xx_battery
439:	129	0	0	0	msmgpio	atmel_mxt_ts
440:	48	0	0	0	msmgpio	bcm2079x
... (Omit some lines)				<u>The amount of interrupts occurred</u>		

Níveis de Interrupção

Unix define até 32 níveis de prioridade de interrupção (ipl).

Por isso, **uma interrupção pode preemptar (suspender) o tratamento de outra**, e kernel precisa manter uma pilha de tratamentos pendentes. A cada momento, o kernel está executando em um ipl.

No entanto, como tratadores podem precisar acessar concorrentemente estruturas de dados do núcleo (p.e. a tabela de processos), o kernel também pode desabilitar todas as interrupções para evitar ser interrompido durante o salvamento e restauração do estado de execução dos processos.

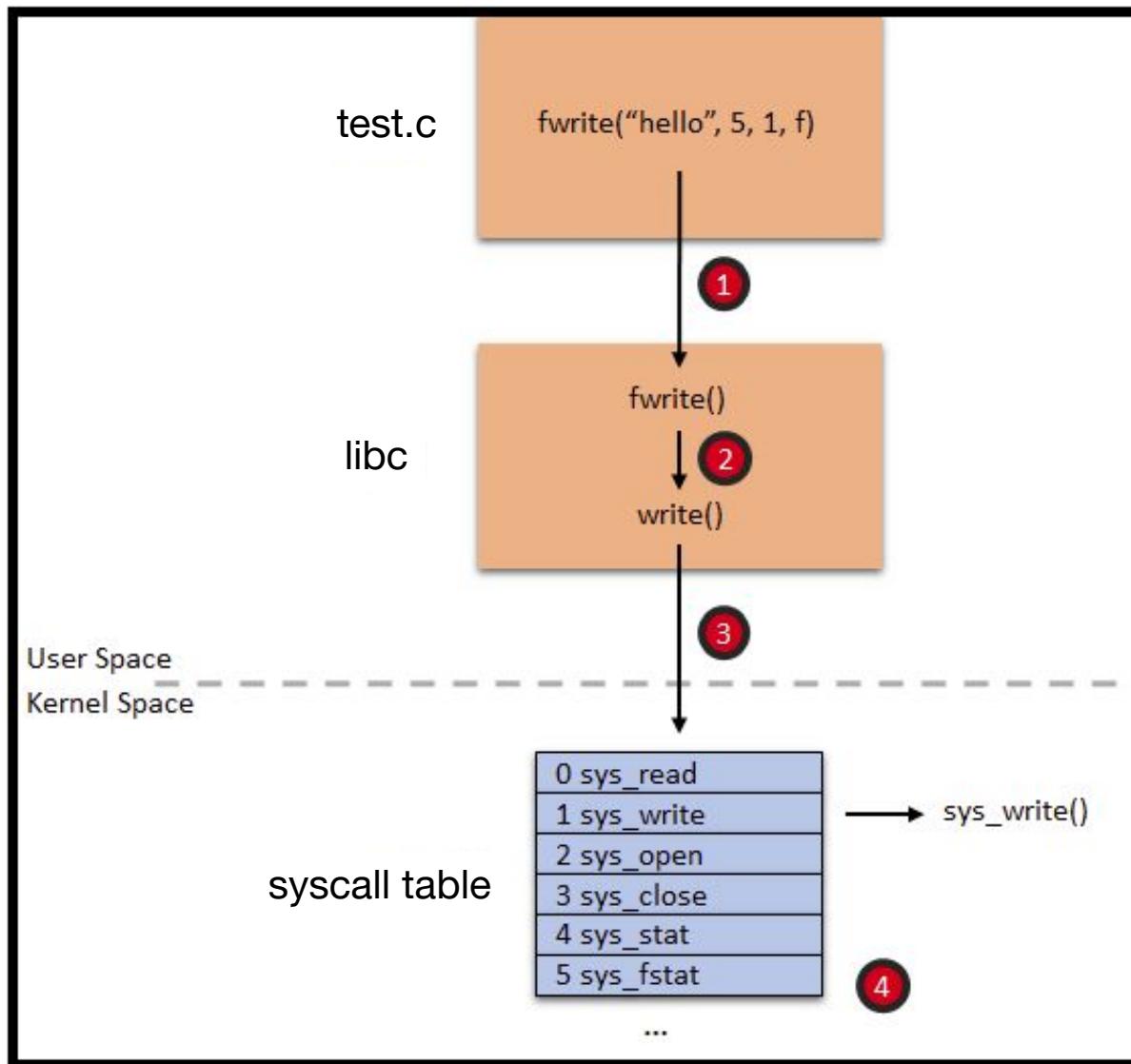
Chamadas de Sistema – System Calls

É o conjunto de procedimentos disponibilizadas pelo núcleo para acesso a serviços do núcleo e a recursos da máquina: gerenciar processos, sinais, arquivos e diretórios, I/O, etc.

As funções da System Calls fazem parte de uma biblioteca (em UNIX, libc).

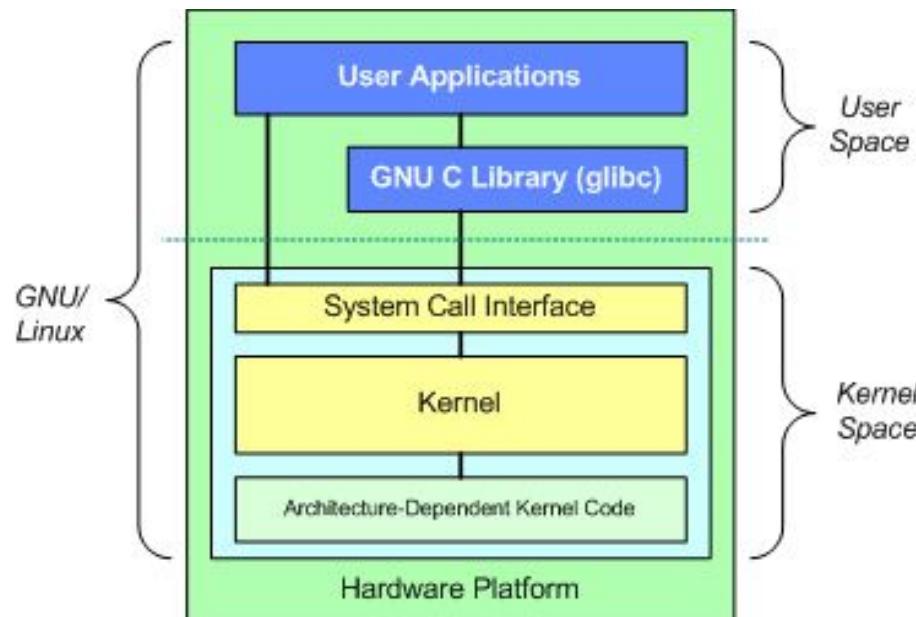
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Chamadas de Sistema



Chamadas de Sistema – System Calls

Cada função dessa da biblioteca executa a instrução TRAP para passar para o modo kernel.

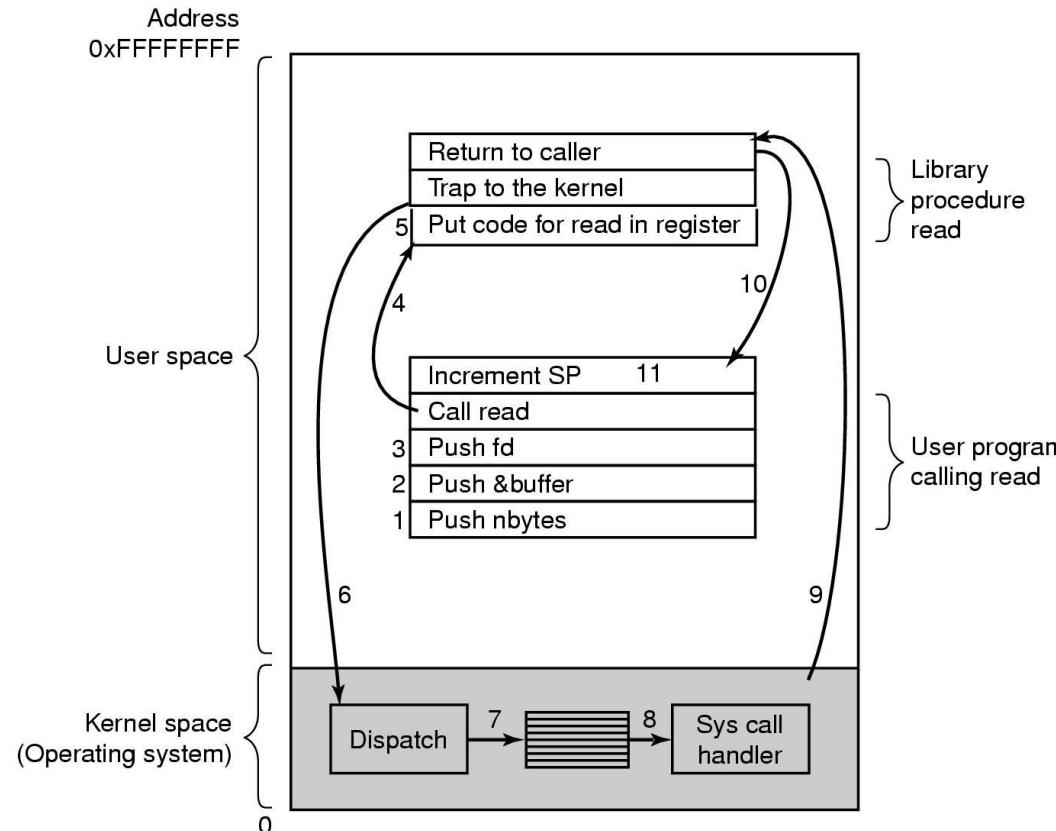


`libc` escreve o número da chamada em um registrador da CPU, que está associado ao tratador da chamada.

O processo retoma o controle exatamente na instrução seguinte ao TRAP.

Etapas de uma Chamada a Sistema (interrupção por software)

Exemplo: `read (fd, buffer, nbytes)`



Fluxo de controle: programa de aplicação → biblioteca libc → núcleo → biblioteca libc → programa de aplicação.

Processamento de cada instrução

A Execução de cada instrução segue os seguintes passos:

1. Obtenção da instrução da memória (Fetch)
2. Decodificação da instrução (Decode)
3. Execução:
 1. Obtenção dos argumentos
 2. Processamento pela Unidade Aritmético-Lógica (ALU)
 3. Armazenamento do resultado (em registrador ou Memória)
 4. Atualização do PC

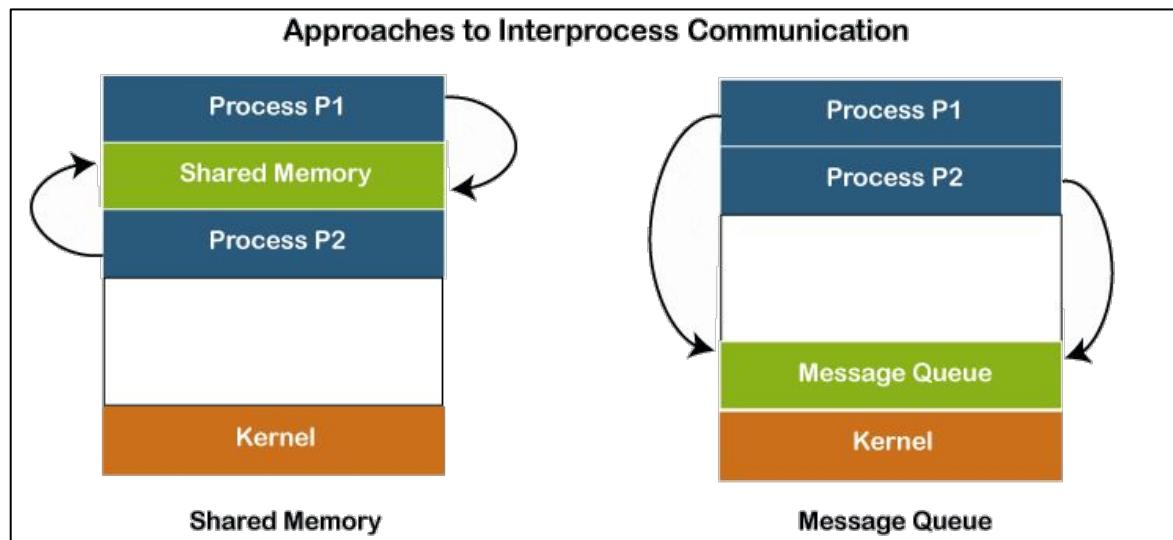
Obs: Passos 1, 2 e 3 são executados em paralelo (por uma pipeline)

Uma interrupção faz com que a instrução corrente seja cancelada e talvez re-iniciada

A Memória Principal (RAM)

Onde reside o núcleo do S.O (kernel), os drivers e os processos

- cada processo está em seu “espaço de endereçamento”, totalmente isolado dos outros espaços de endereçamento
- mas segmentos de memória podem ser compartilhados
- ou então, comunicação é feita através do núcleo (kernel)

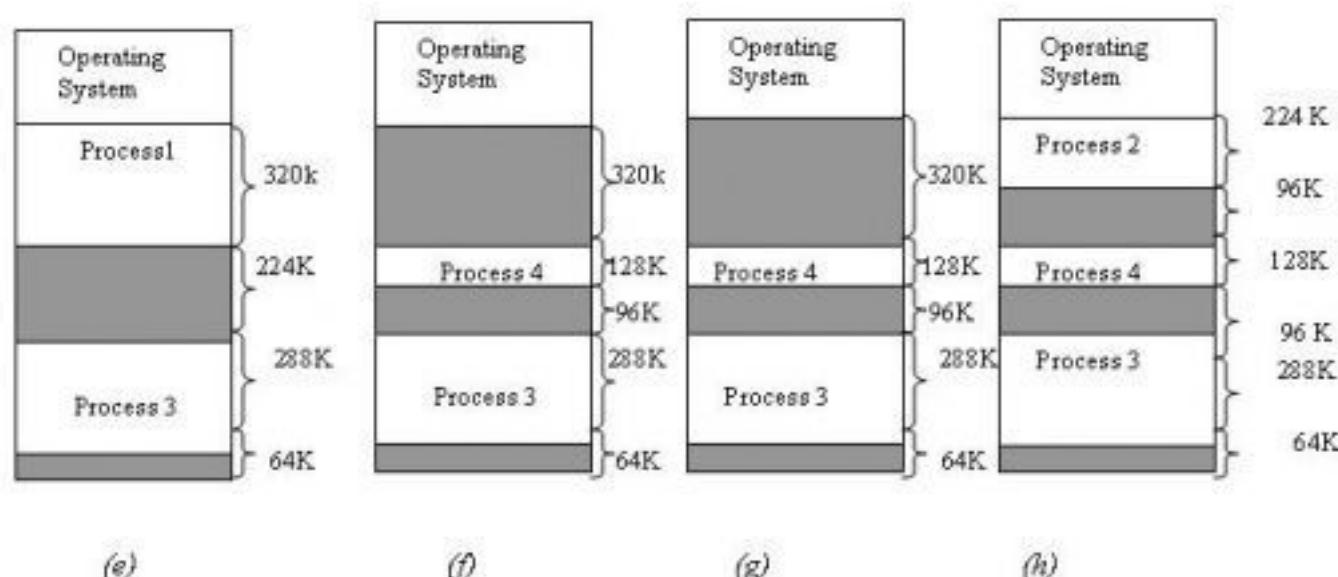


Tempos de acesso
RAM: 50-70 ns
L1 cache: 0.5 ns
L2 cache: 1.8 ns
L3 cache: 4.2 ns
SSD: 25-100 µs (=25000 ns)
HD: 5 to 10 milliseconds.

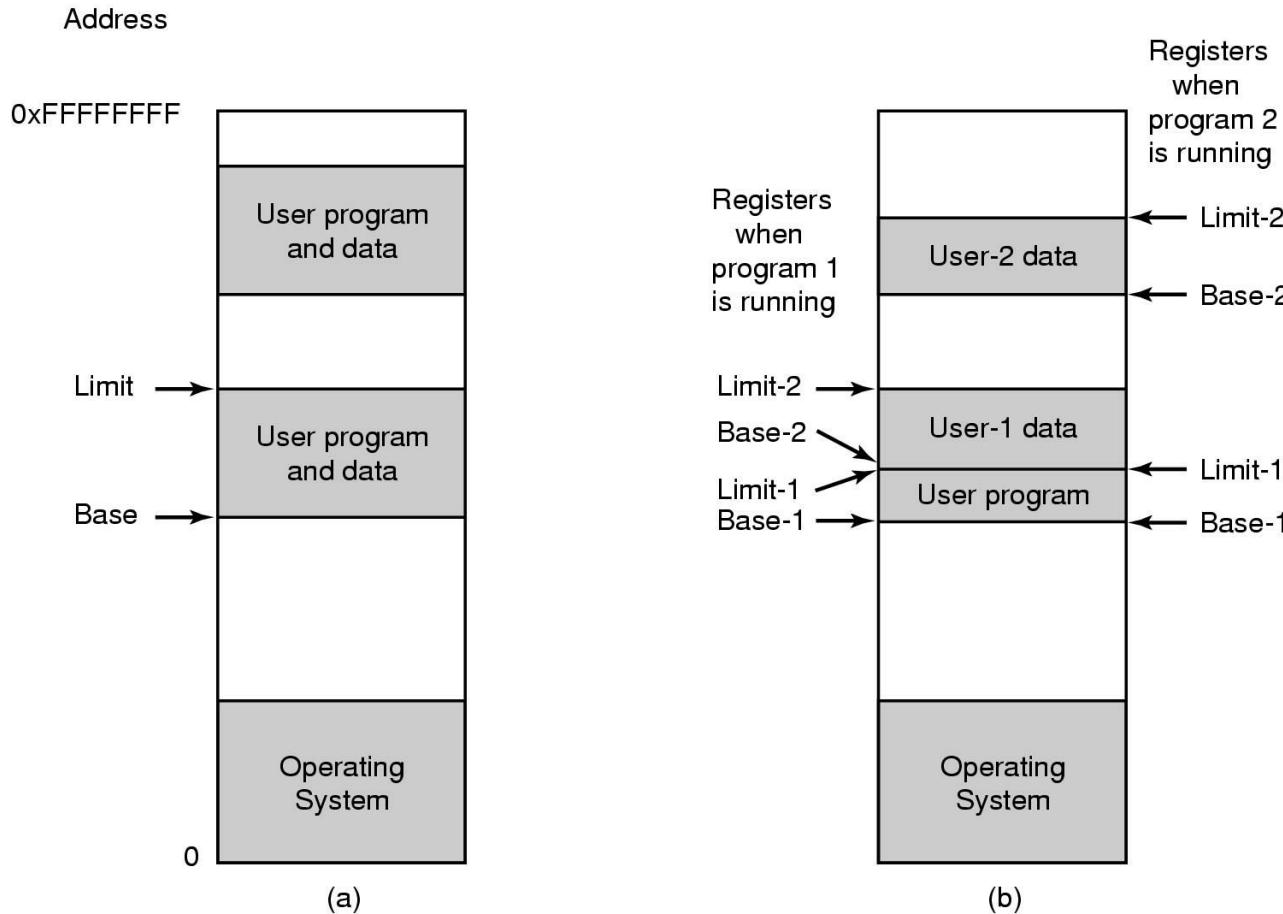
Uso de memória principal

- Para executar, um processo precisa estar (pelo menos parcialmente) na memória física (RAM)
- Mas a soma da memória dos processos “em execução” (aptos a executar) no sistema geralmente ultrapassa o tamanho da memória física
- Por isso, o escalonador decide quais processos irá carregar ou remover da memória
- Isso é feito pelo Gerente de Memória

Obs: o núcleo permanece permanentemente na memória (endereços iniciais)



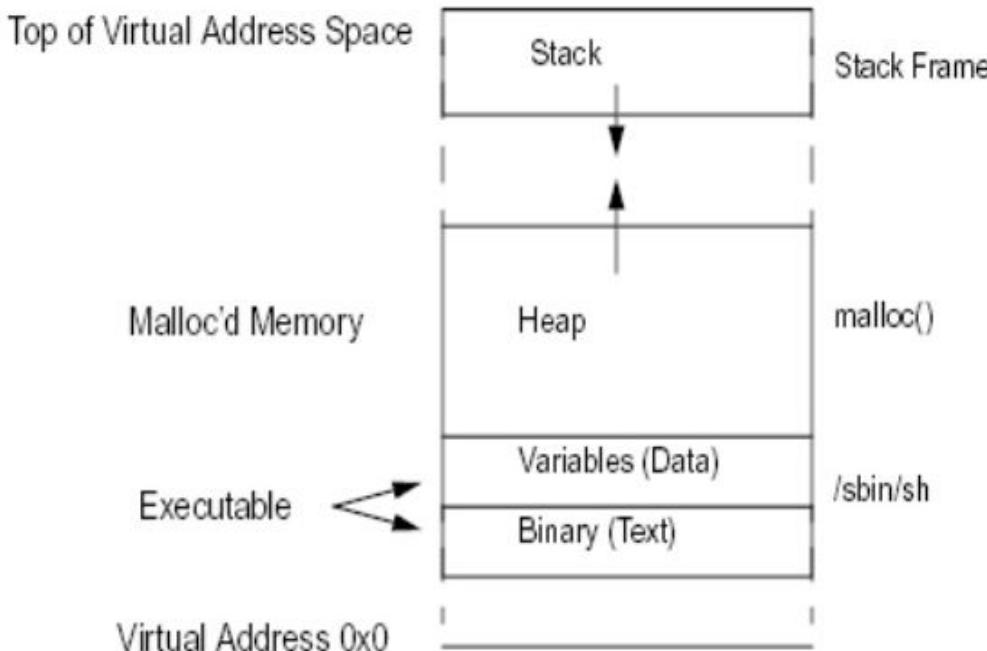
Ocupação da Memória



Cada processo é composto de segmentos que **ocupam uma região própria (e isolada) na memória**. e endereços de memória são traduzidos pela **Memory Management Unit (MMU)** controlada pelo kernel (end. lógico -> endereço físico).

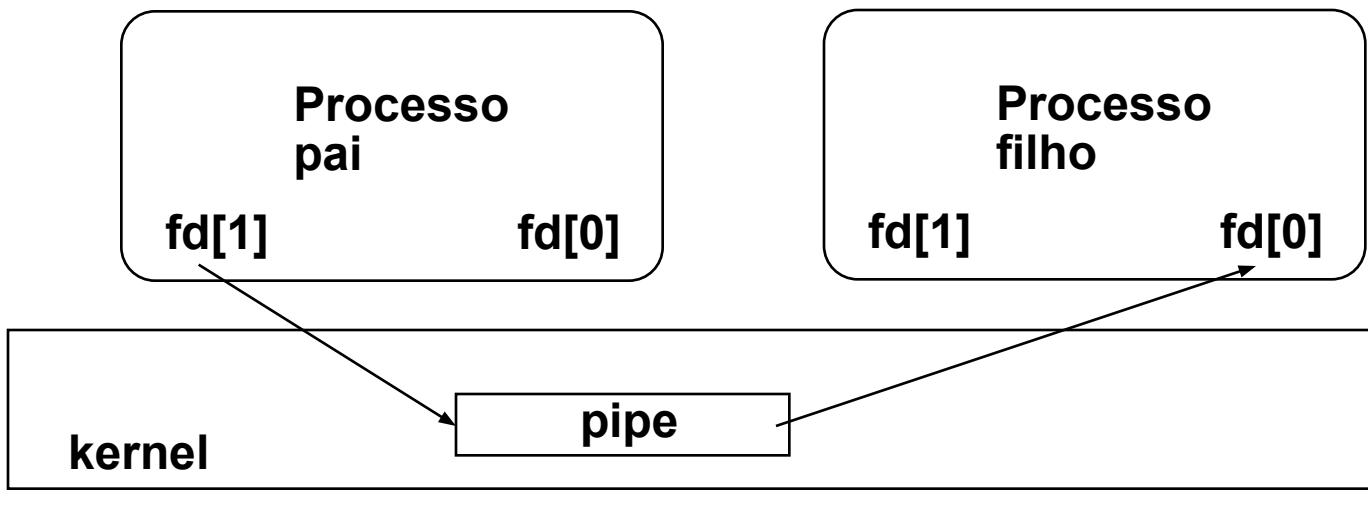
Segmentos podem estar em região contígua (a) ou não (b)

Um processo na memória



- Processos possuem 3 segmentos:
 - Text+variables (executável), heap, e pilha
 - segmento Data: strings, constantes, argv[], variáveis globais
- Segmentos heap e pilha crescem em sentidos opostos
- Total de memória a ser alocada é definida na link-edição ou no carregamento do processo
- Através da chamada de sistema brk(newDataLimitAddr) processo pode requisitar mais espaço na memória.

Pipe de pai para filho



```
int n, fd[2];
...
pipe(fd);
if ((pid = fork()) > 0) {
    close(fd[0]);
    write(fd[1], "hello\n", 6);
} else {
    close(fd[1]);
    n = read(fd[0], line, MAXLINE);
    write(STDOUT, line, n);
}
```

Exemplo Fork/Exec: Esboço simplificado de uma shell

```
while (TRUE) {                                /* repeat forever */  
    type_prompt( );                          /* display prompt */  
    read_command (command, parameters)      /* input from terminal */  
  
    if (fork() != 0) {                      /* fork off child process */  
        /* Parent code */  
        waitpid( -1, &status, 0);            /* wait for child to exit */  
    } else {  
        /* Child code */  
        execve (command, parameters, 0);    /* execute command */  
    }  
}
```

Obs: **fork()** retorna 0 para o processo filho (criado), e o Pid do filho para o processo pai,

Chamadas de Sistema: Outras tarefas

Diversas

Chamada	Descrição
s = chdir(dirname)	Altere o diretório de trabalho
s = chmod(name, mode)	Altere os bits de proteção do arquivo
s = kill(pid, signal)	Envie um sinal a um processo
seconds = time(&seconds)	Obtenha o tempo decorrido desde 1º de janeiro de 1970

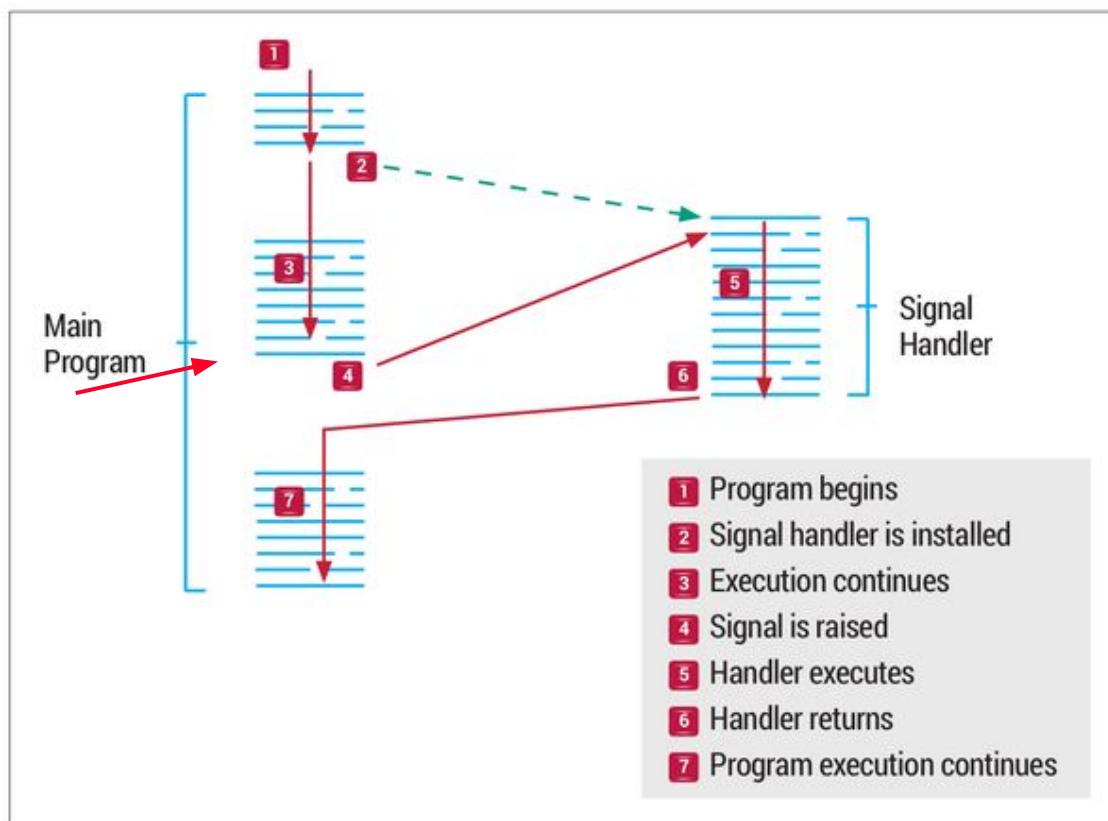
Table 2.1: Unix file system application programming interface

Procedure	Brief description
OPEN (<i>name, flags, mode</i>)	Open file <i>name</i> . If the file doesn't exist and <i>flags</i> is set, create file with permissions <i>mode</i> . Set the file cursor to 0. Returns a file descriptor.
READ (<i>fd, buf, n</i>)	Read <i>n</i> bytes from the file at the current cursor and increase the cursor by the number of bytes read.
WRITE (<i>fd, buf, n</i>)	Write <i>n</i> bytes at the current cursor and increase the cursor by the bytes written.
SEEK (<i>fd, offset, whence</i>)	Set the cursor to <i>offset</i> bytes from beginning, end, or current position.
CLOSE (<i>fd</i>)	Delete file descriptor. If this is the last reference to the file, delete the file.
FSYNC (<i>fd</i>)	Make all changes to the file durable
STAT (<i>name</i>)	Read metadata of file.
CHMOD, CHOWN, etc.	Various procedures to set specific metadata.
RENAME (<i>from_name, to_name</i>)	Change name from <i>from_name</i> to <i>to_name</i>
LINK (<i>name, link_name</i>)	Create a hard link <i>link_name</i> to the file <i>name</i> .
UNLINK (<i>name</i>)	Remove <i>name</i> from its directory. If <i>name</i> is the last name for a file, remove file.
SYMLINK (<i>name, link_name</i>)	Create a symbolic name <i>link_name</i> for the file <i>name</i> .
MKDIR (<i>name</i>)	Create a new directory named <i>name</i> .
CHDIR (<i>name</i>)	Change current working directory to <i>name</i> .
CHROOT (<i>name</i>)	Change the default root directory to <i>name</i> .
MOUNT (<i>name, device</i>)	Graft the file system on <i>device</i> onto the name space at <i>name</i> .
UMOUNT (<i>name</i>)	Unmount the file system at <i>name</i> .

Sinais

Um sinal interrompe um processo instantaneamente, na instrução corrente, para informá-lo da ocorrência de uma exceção.

O controle da execução é desviado para um procedimento (tratador do sinal, *signal handler*), e depois controle retorna a instrução interrompida do processo.



Sinais

Sinais podem ser enviados:

- Entre processos de um mesmo grupo (com mesmo userID, real ou efetivo) ou pelo processo de root
- Pelo núcleo do S.O. para um processo (devido a ocorrência de um erro: divisão por zero, segmentation fault, etc.) ou como consequência de um comando do usuário CTRL-C, CTRL-\,

Em Unix existem aproximadamente 64 sinais: e.g.

- keyboard interrupt,
- erro em um processo,
- start/stop processo,
- kill,
- eventos da rede,
- sinal de timer,
- etc.

Sinais

Exemplo de alguns sinais:

- SIGINT interrupt, gerado pelo núcleo, (p.ex. ^C)
- SIGQUIT encerra e faz um core dump do processo corrente
- SIGILL instrução ilegal (depende do hardware)
- SIGFPE floating point exception
- SIGKILL/SIGTERM/SIGHUP gerado pela system call kill()
- SIGALRM alarm clock, timeout
- SIGUSR1/SIGUSR2 sinal definido pelo desenvolvedor
- ...

Exemplo:

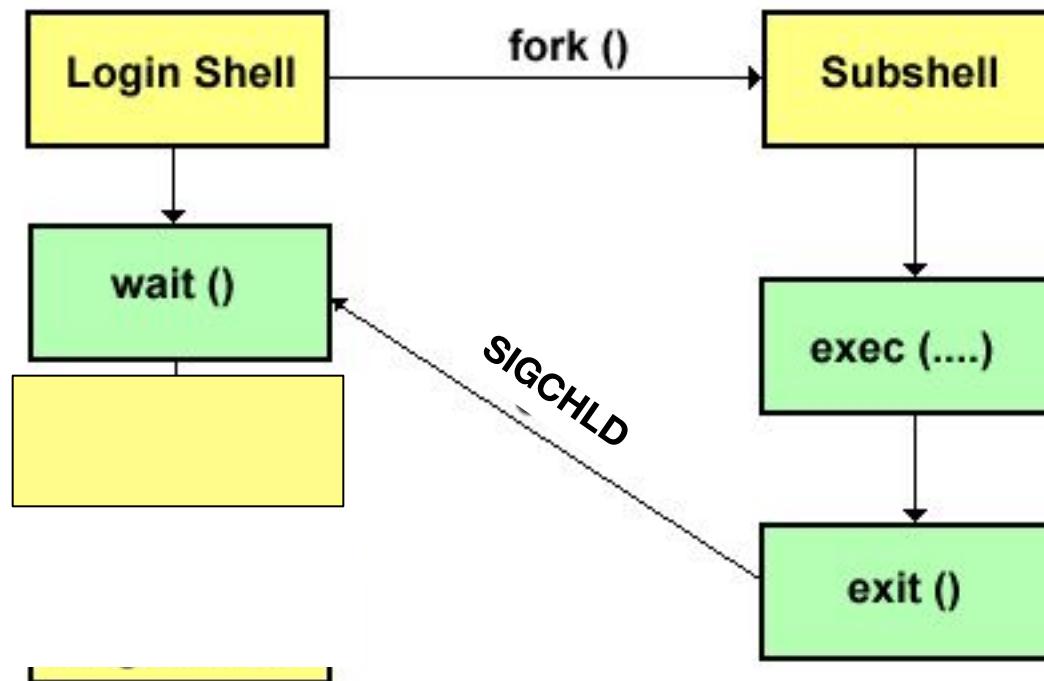
```
signal(SIGUSR1, SIG_IGN); /* SIG_IGN indica que sinal SIGUSR1 deve ser  
ignorado*/
```

Tratamento de sinal SIGINT pelo procedimento myintr

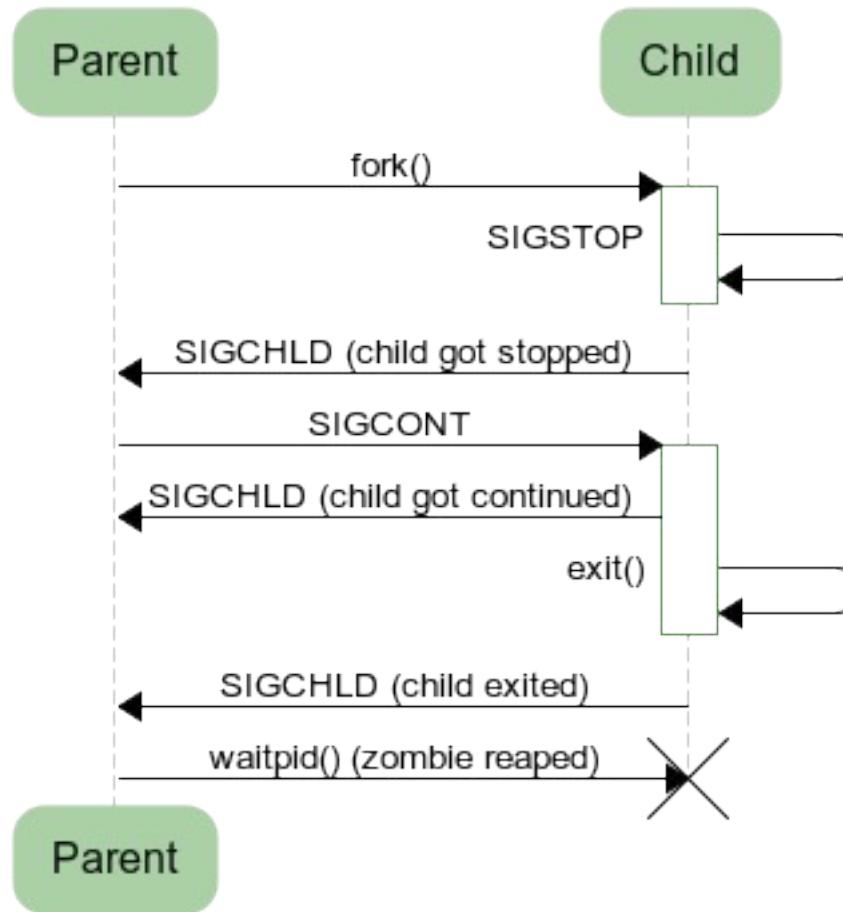
```
#include <signal.h>  
extern void myintr();  
...  
if (signal(SIGINT, SIG_IGN) != SIG_IGN) signal(SIGINT, myintr);  
...
```

Exemplo: Sinal SIGCHLD

de processo filho para o processo pai



Sinais SIGCONT/SIGSTOP



Sinais

Signais podem ser:

- **ignorados**,
- **bloqueados** ou
- **tratados** por um procedimento provido pelo desenvolvedor.

Um processo pode bloquear um sinal através de **sigblock(mask)**

Diversos sinais podem ser especificados na máscara usando

sigmask(NOME_SINAL)

```
mask = sigmask(SIGQUIT) | sigmask(SIGINT)  
sigblock(mask)
```

- Existem sinais (como o **SIGKILL**) que não podem ser bloqueados ou ignorados e invariavelmente causam o término do processo alvo.
- Atenção: Sinais podem ser perdidos: se vários sinais são enviados para processo antes de serem tratados, o último sobre-escreve os anteriores

Recapitulando...

Responsabilidades do núcleo (Kernel)

Núcleo é responsável por:

- Tratar as interrupções de hardware
- Ativar o processo *init*
- Fazer a troca de contexto (da CPU) entre processos
- Manter a informação sobre todos os processos em execução e os recursos usados por cada um
- Garantir a independência de cada processo
- Entregar os sinais para os processos destinatários
- Implementar um tratador para cada chamada de sistema
- Implementar os mecanismos para comunicação entre processos (pipe, FIFO, MessageQueue)
- Coletar estatísticas sobre a execução dos processos e
- Manter informações sobre a ocupação e disponibilidade dos recursos