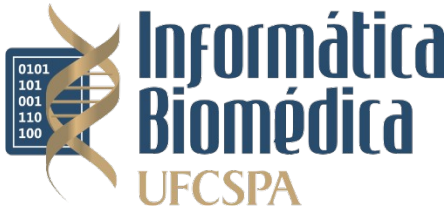


UFCSPA Informática Biomédica

Sistemas Operacionais

Serviços e Estrutura dos Sistemas Operacionais

Prof. João Gluz

Porto Alegre, RS, Brasil
2019

Serviços dos Sistemas Operacionais (SO)

- Um Sistemas Operacional (**SO**) fornece um ambiente para execução de programas e serviços para programas e usuários formado por:
- **Interface do usuário** - quase todos os sistemas operacionais têm uma interface de usuário (**UI**).
  - Varia entre: interface de comandos (**CLI**), interface gráfica (**GUI**) ou interface *batch*
- **Execução do programa** - o SO deve ser capaz de carregar um programa na memória, executar esse programa e finalizar sua execução de forma normal ou anormal (indicando erro)
- **Operações de E/S** - programas em execução podem requerer operações de Entrada ou Saída (E/S) envolvendo um arquivo ou um dispositivo de E/S

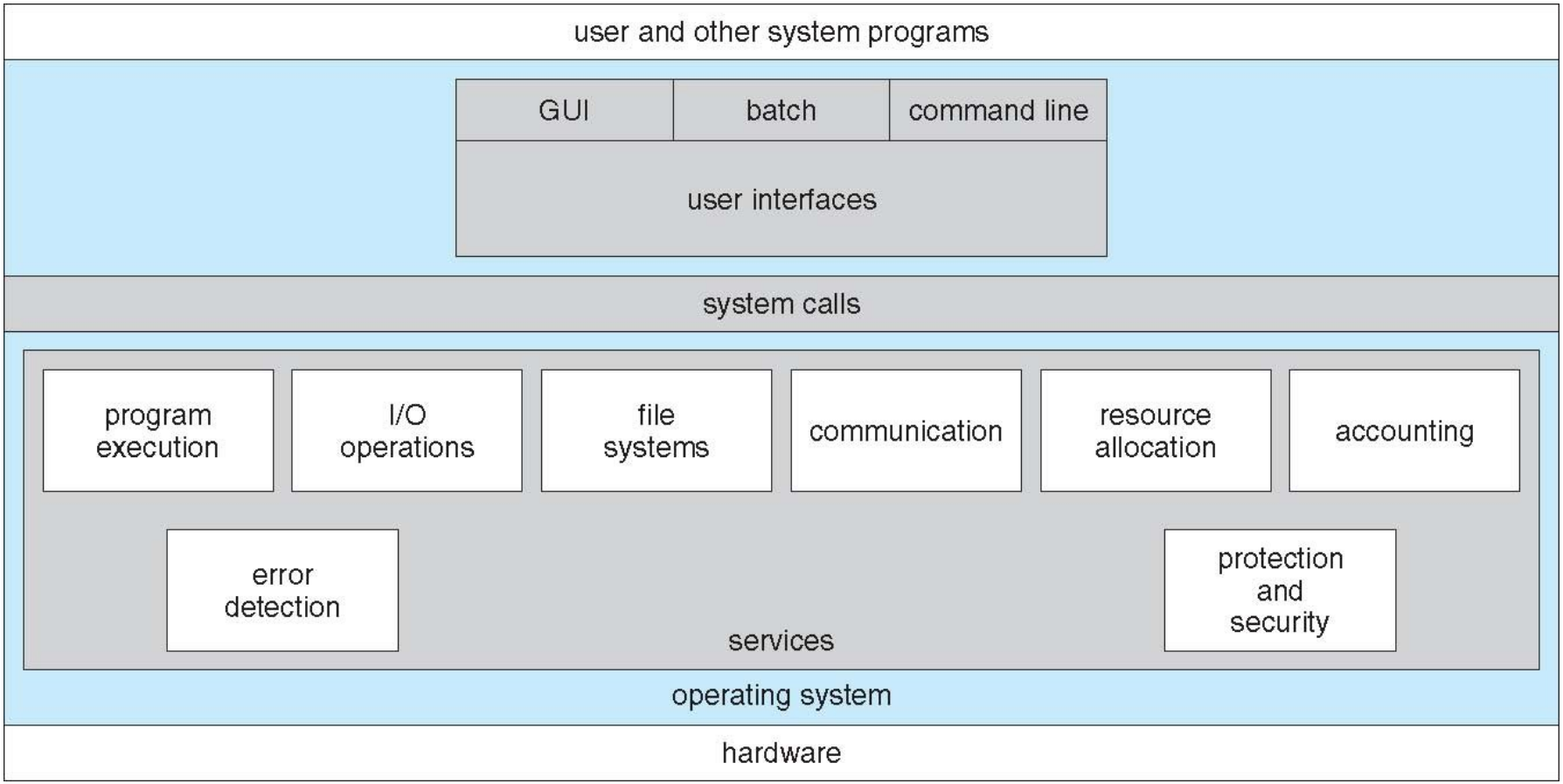
Serviços dos SO (Cont.)

- **Manipulação do sistema de arquivos** - o sistema de arquivos é um serviço importante dos SO porque os programas precisam ler e gravar arquivos e diretórios, criá-los e excluí-los, pesquisá-los, listar informações de arquivos além de gerenciar as permissões.
- **Comunicações** – programas em execução (processos) podem trocar informações no mesmo computador ou entre computadores em uma rede
  - As comunicações podem ser via memória compartilhada ou através de troca de mensagens
- **Detecção de erros** - o SO precisa estar constantemente ciente de possíveis erros que podem ocorrer no hardware da CPU e memória, em dispositivos de E/S ou nos programas de usuário
  - Para cada tipo de erro, o sistema operacional deve executar a ação apropriada para garantir uma computação correta e consistente

Serviços dos SO (Cont.)

- **Alocação de recursos** - quando vários processos estão em execução simultânea, os recursos devem ser alocados para cada um deles
  - Muitos tipos de recursos são controlados e disponibilizados pelo SO: ciclos de CPU, memória principal, arquivos, dispositivos de E / S.
- **Contabilidade** - acompanha quais usuários usam quanto e que tipos de recursos de um sistema computacional
- **Proteção e segurança** - proprietários de informações armazenadas em um sistema computacional multiusuário ou em rede usualmente querem controlar o uso dessas informações, além disso processos concorrentes não devem interferir uns com os outros.
  - Proteção envolve garantir que todo o acesso aos recursos do SO seja controlado
  - Segurança do sistema requer a autenticação de usuários, estendendo-se à defesa de dispositivos de E/S contra tentativas de acesso inválidas

Visão Geral dos Serviços dos SO

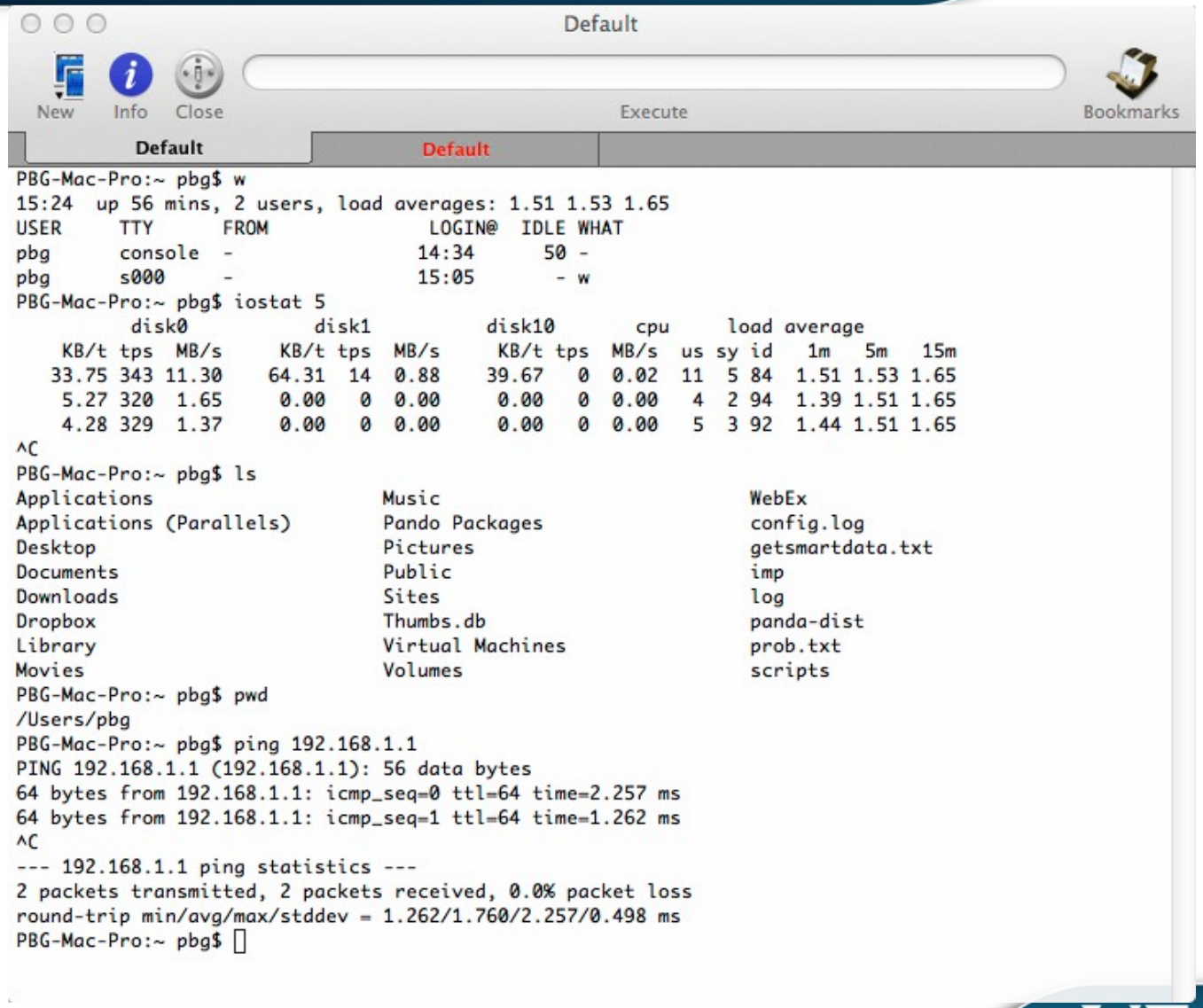


Interface de Comandos - CLI

A interface de comandos (**CLI** - *Command Line Interface*) ou o **interpretador de comandos** permite a entrada direta de comandos ao SO

- A CLI pode ser implementada pelo núcleo (kernel) do SO ou por programas de sistema (solução mais usual)
- As vezes várias opções de CLI estão disponíveis – vários **shells**
- Segue um ciclo simples de leitura e execução de comandos do usuário digitados na forma de texto
- Comandos pode ser implementados pelo interpretador (*built-in*) ou simplesmente nomes de programas disponíveis no sistema de arquivos que são automaticamente localizados, carregados e executados pelo interpretador

Interpretador BASH - Bourne “Again” Shell




```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                 14:34      50 -
pbg      s000     -                 15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10       cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

UFCSPA

Interface Gráfica - GUI

- Metáfora de interface gráfica **desktop** usualmente inclui:
  - Mouse, teclado e monitor gráfico (colorido)
  - **Icones** representam arquivos, programas, ações, etc
  - Click nos botões do mouse sobre objetos na interface causam varias ações (provêm informações ou opções, executam funções, abre arquivos e diretórios, etc.)
  - Inventado no Xerox PARC
- Maioria dos SO incluem interfaces CLI e GUI
  - O Microsoft Windows tem uma GUI com CLI "command" shell
  - O Apple Mac OS X tem uma interface GUI sobre um SO UNIX que possui vários tipos de shell CLI disponíveis
  - Unix e Linux têm várias interfaces CLI (sh, bash) com várias interfaces GUI opcionais (CDE, KDE, GNOME)

Mac OS X GUI


Grab File Edit Capture Window Help
15:06 EDT Mon 2 Jul
fig-dir
TIFF
Address Book
Apple Computer Inc.
Dictionary and Thesaurus
operating system
op•er•at•ing sys•tem
PCalc
Trash

Novas Interfaces Touchscreen

- Dispositivos touchscreen requerem novas interfaces
  - O Mouse não é possível ou não desejado
  - Ações e seleções baseadas em gestos
  - Uso de teclado virtual para entrada de texto
- Evolução das interfaces GUI
- Uso de comandos de voz

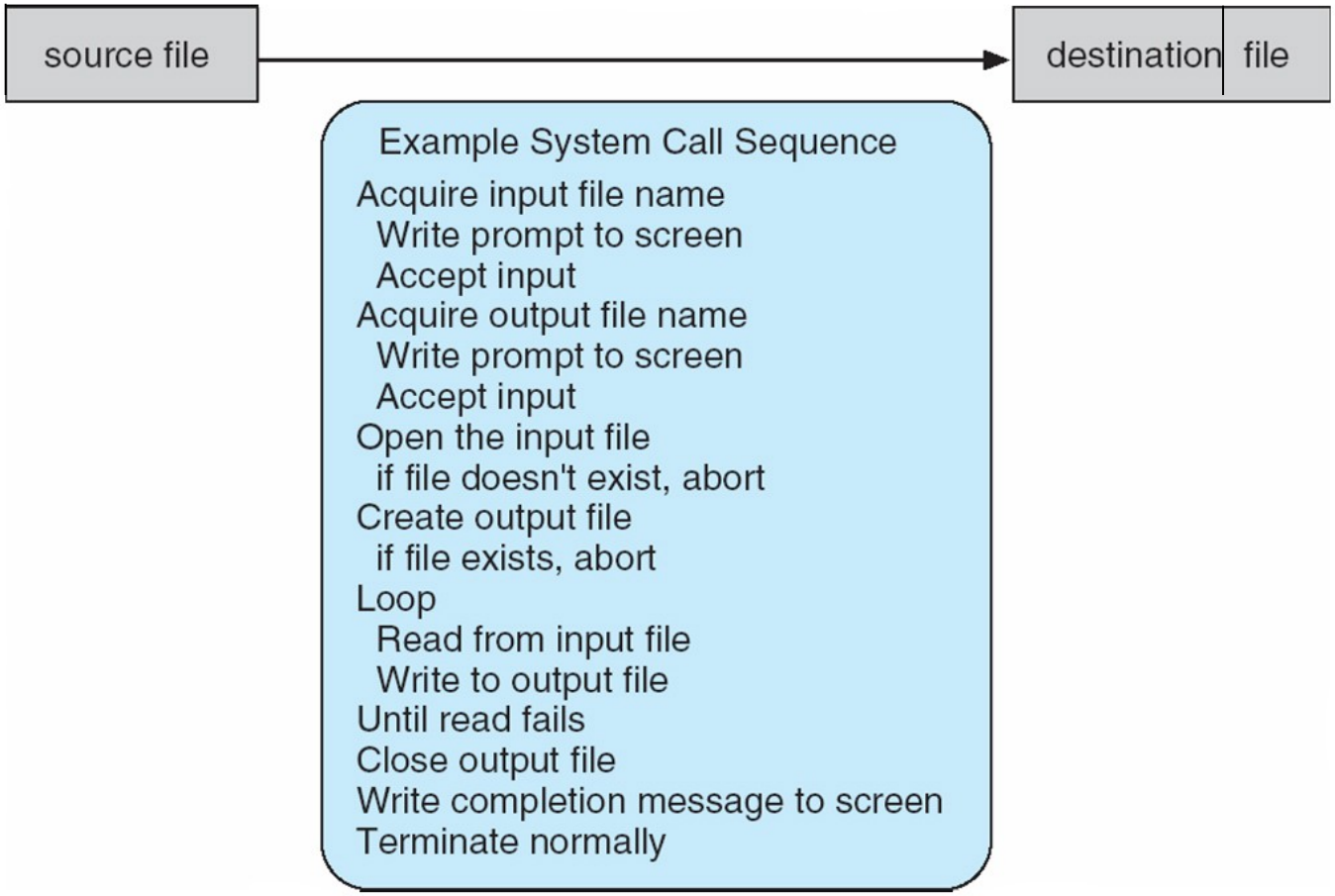
UFCSPA

Chamadas de Sistema (*System Calls*)

- As **chamadas de sistemas** (***system calls*** ou ***syscalls***) são a interface de programação para os serviços providos pelo SO
- São tipicamente escritas em linguagems de alto nível (C / C++)
- São acessadas pelos programas através de uma **Application Programming Interface (API)** ao invés de chamadas diretas de interrupção
- As três APIs mais comuns são: **Win32 API** para o Windows, **POSIX API** para sistemas baseados em POSIX-based systems (virtualmente todas as versões de UNIX, Linux e Mac OS X) e a API Java para a máquina virtual Java (**JVM**)

Exemplo de System Calls

- Sequência de system calls para copiar os conteúdos de um arquivo para outro



Exemplo de API padrão

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

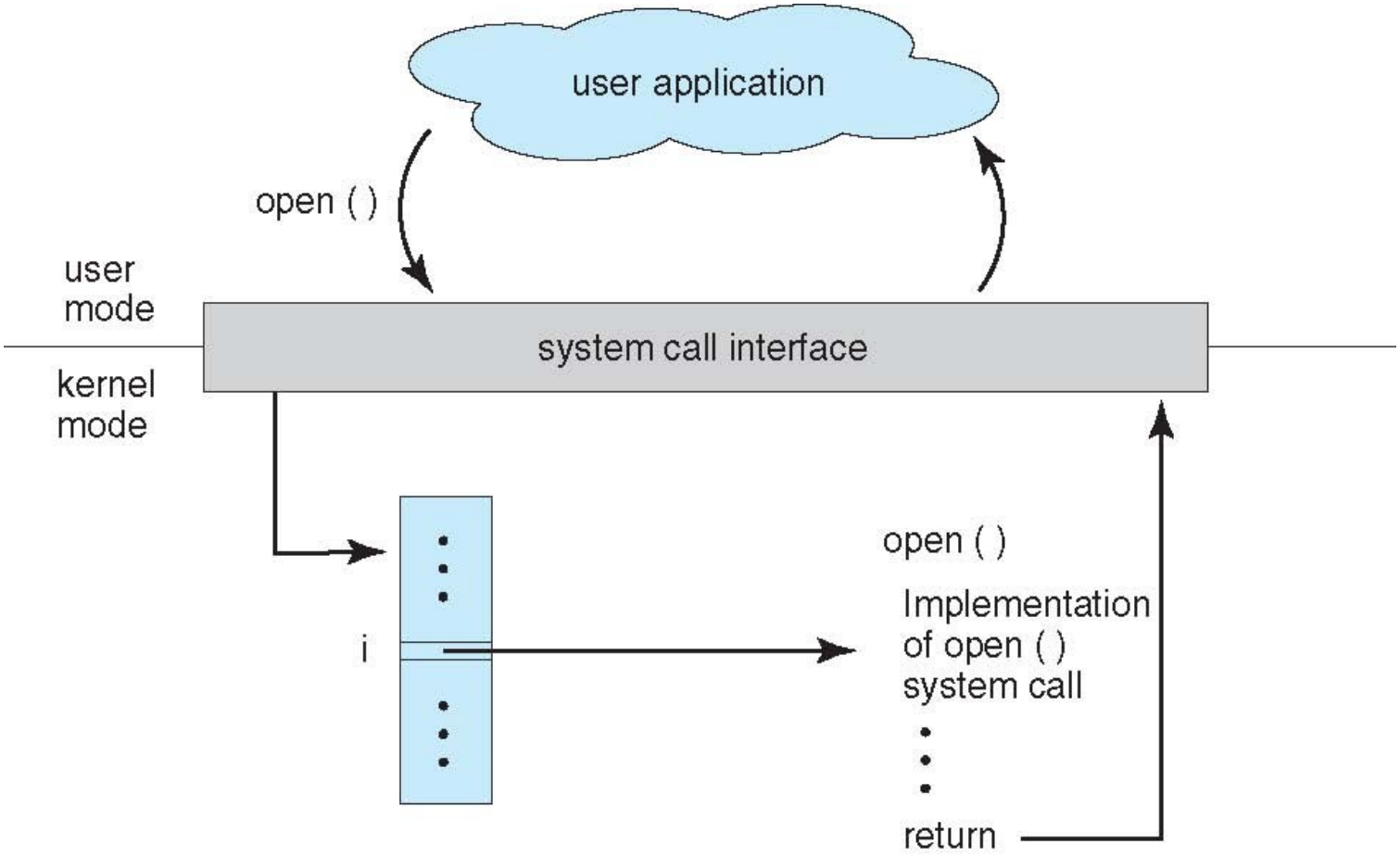
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

UFCSPA

Implementação de System Calls

- Uma chamada de sistema é implementada através de uma interrupção de software, também chamada de ***trap*** identificada por um número (endereço) específico
- A invocação de uma *trap* passa o modo de execução do programa de modo usuário para modo privilegiado (modo *kernel*)
- Detalhes das chamadas de sistema são escondidos pela API de chamadas de sistema, implementada por uma biblioteca (*library*) de sistema ligada ("linkada") ao programa de usuário durante a compilação
- Uma rotina nesta biblioteca que invoca a chamada de sistema e retorna o estado da chamada com dados adicionais produzidos pela chamada
- O programa chamador não necessita saber como a chamada é realmente implementada, apenas necessita estar conforme a API de sistema e entender os resultados da chamada

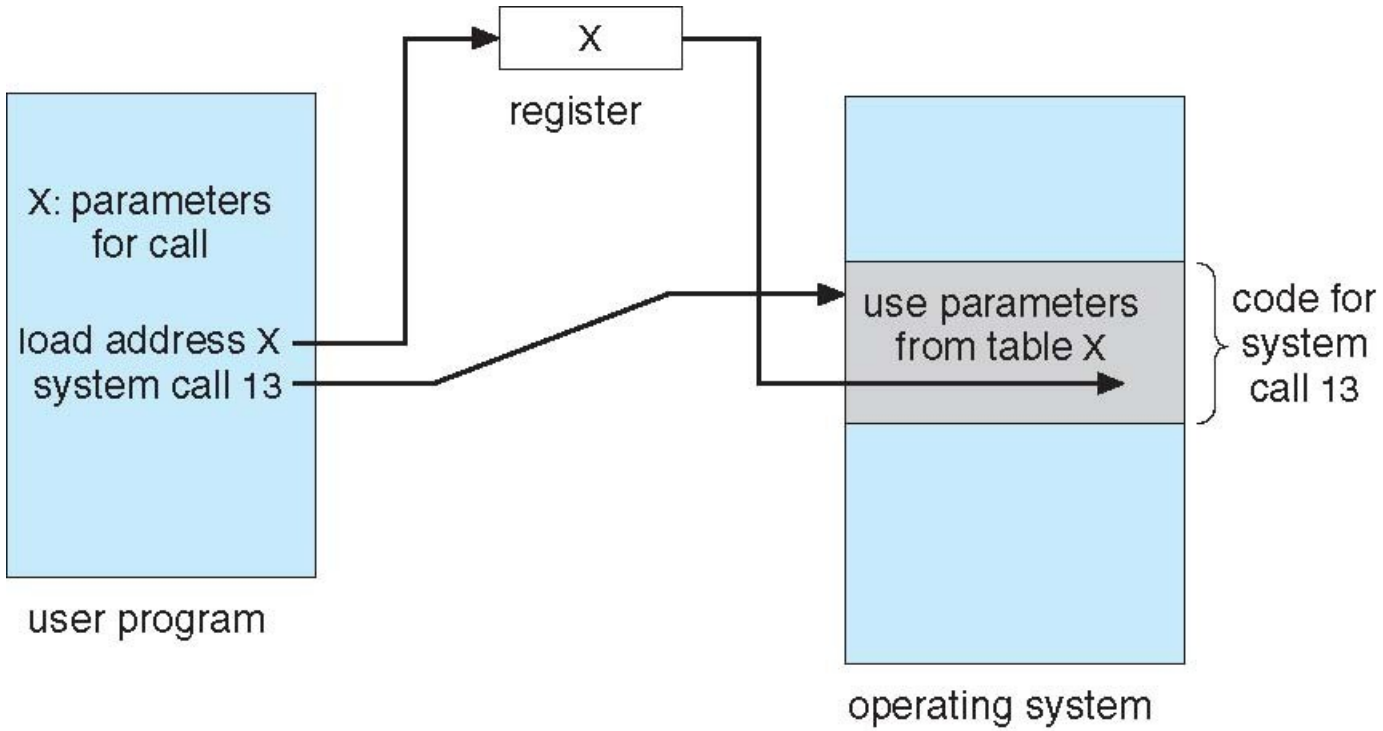
API – System Call – OS



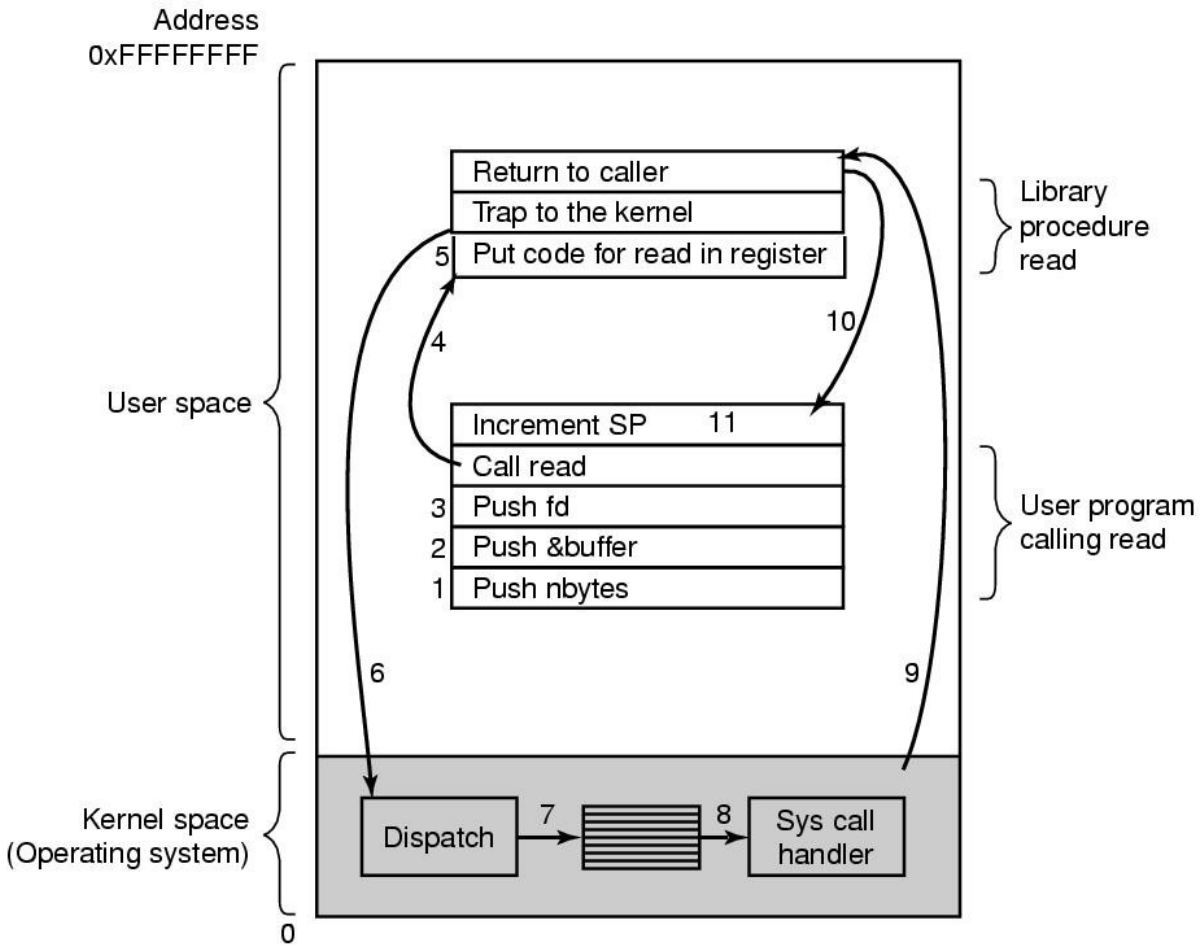
Parâmetros de System Calls

- Usualmente mais informações são necessárias do que apenas a identificação da chamada de sistema – as chamadas de sistema usualmente requerem **parâmetros** (de entrada ou de saída)
- Três métodos podem usados para passar parâmetros ao SO:
  - Mais simples: passar os parâmetros por registrador
  - Armazenar os parâmetros em um bloco ou tabela na memória e passar o endereço do bloco em um registrador (Linux e Solaris)
  - Os parâmetros pode ser empilhados (**pushed**) na pilha de sistema (**stack**) pelo programa e desempilhados (**popped**) pelo SO

Passagem de Parâmetros por Tabela



Exemplo de System Call



read(fd, buffer, nbytes).

UFCSPA

Tipos de System Calls

- Controle de processos:
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump de memória, caso erros ocorram
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** (semáforos) para controle de acesso à recursos compartilhados entre vários processos

Types of System Calls

- Gerência de arquivos:
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Gerência de dispositivos:
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (Cont.)

- Informações de sistema/arquivos:
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Comunicações:
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

Types of System Calls (Cont.)

- Proteção:
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

System Calls do Windows e Unix

	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() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Programas de Sistema

- Os programas do sistema fornecem um ambiente conveniente para desenvolvimento e execução de programas. Eles podem ser divididos em:
  - Manipulação de arquivos
  - Informações de status, às vezes, armazenadas como informações de arquivo
  - Suporte à Linguagens de Programação
  - Carregamento e execução de programas
  - Comunicações
  - Serviços do SO rodando em *background*
- A visão do sistema operacional para a maioria dos usuáriosl é definida por programas do sistema e não pelas chamadas reais do sistema

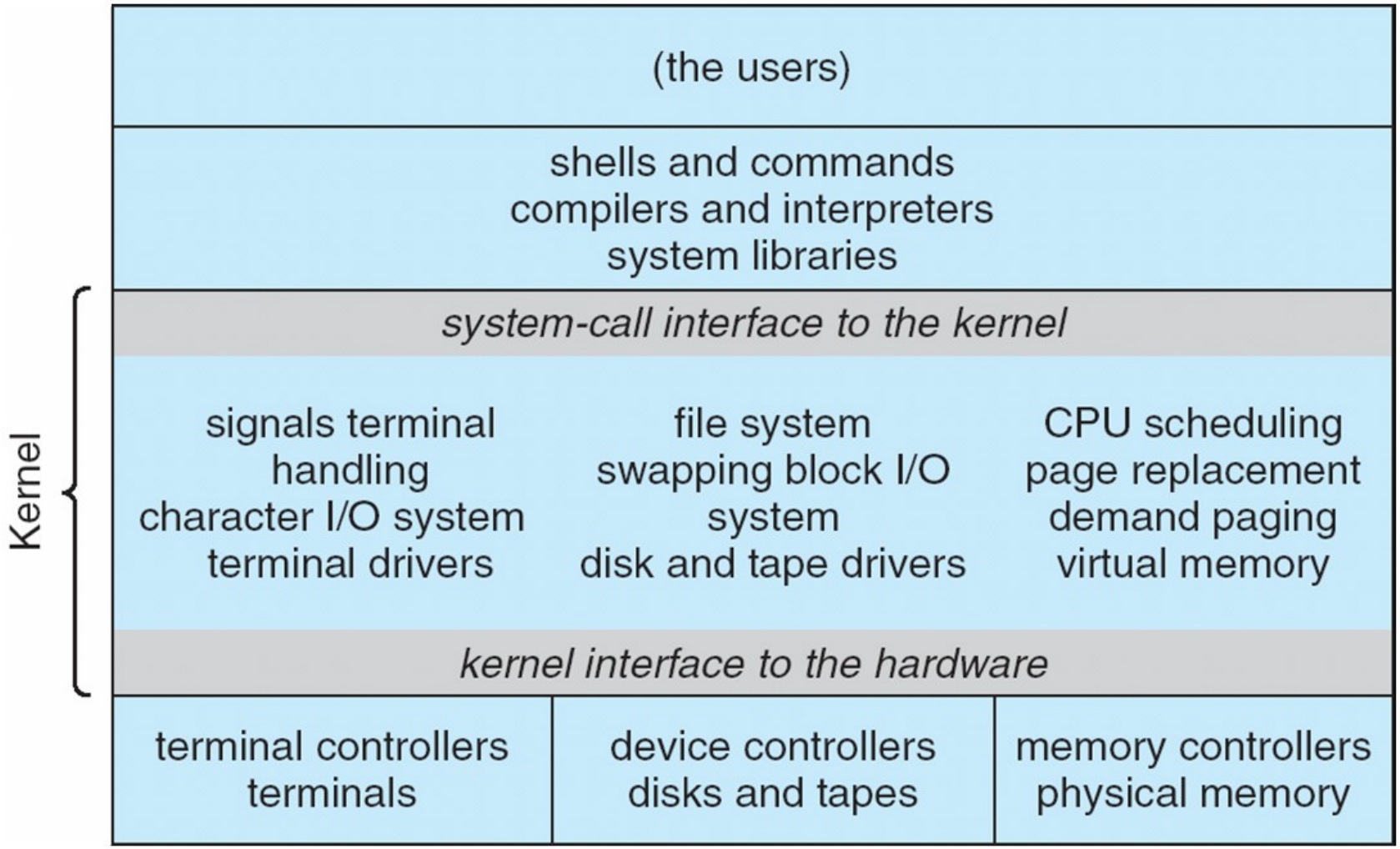
Programas de Sistema (Cont.)

- **Serviços do SO rodando em background**
  - Executados (ativados) durante o boot (carga do SO)
  - Fornecem recursos como verificação de disco, agendamento de processos, registro de erros, impressão, etc.
  - Normalmente executam no espaço do usuário e não no espaço do kernel
  - Conhecidos como **serviços**, **subsistemas** ou **daemons**

Implementação e Estrutura dos SO

- Muita variação na implementação
  - Primeiros OS em linguagem assembly
  - Posteriormente foram usadas linguagem de programação de sistemas como Algol, PL/1
  - Atualmente C, C++
- Atualmente é usado um mix de linguagens
  - Níveis mais baixo em assembly
  - Código principal do SO em C
  - Programas de sistema em C, C++ e linguagens de script como PERL, Python e shell
- **Emulação** permite que SOs rodem em hardware não nativo

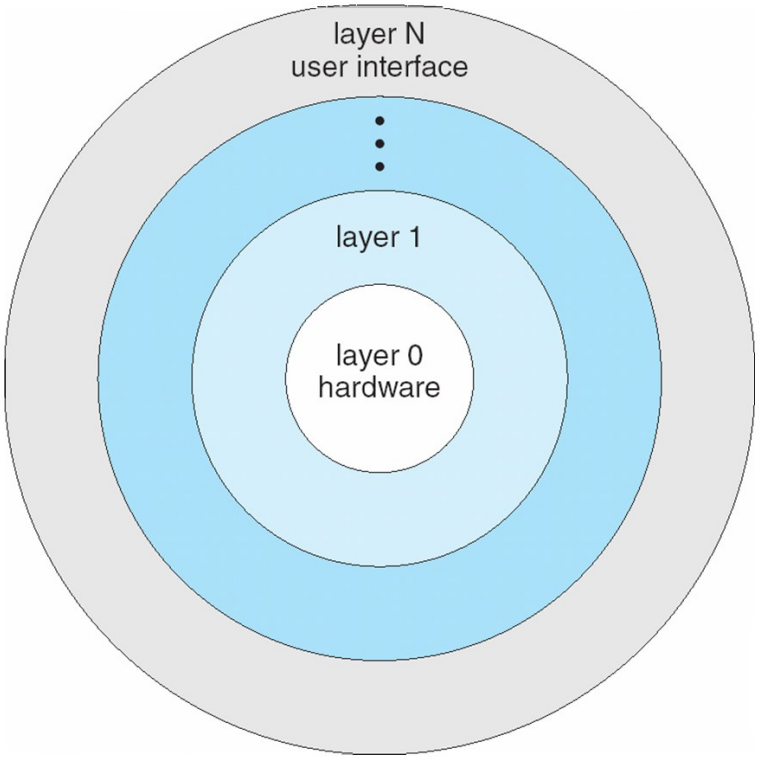
Estrutura Tradicional do UNIX



Kernel monolítico que não segue totalmente a estrutura em camadas

Estrutura em Camadas

- O sistema operacional é dividido em várias camadas (níveis), cada uma construída em cima das camadas inferiores. A camada inferior (camada 0) é o hardware; a mais alta (camada N) é a interface do usuário.
- Espera-se que as camadas sejam projetadas de forma que cada uma use funções (operações) e serviços apenas de camadas de nível inferior.



Estrutura de Microkernel

- Move o máximo possível do kernel para o espaço (contexto) do usuário
- A comunicação ocorre entre os programas do usuário e o SO usando passagem de mensagens
- Benefícios:
  - Mais fácil de estender um microkernel
  - Mais fácil de portar o sistema operacional para novas arquiteturas
  - Mais confiável (menos código está sendo executado no modo kernel)
  - Mais seguro
- Problemas:
  - Sobrecarga de desempenho do espaço do usuário para a comunicação de espaço do kernel

Estrutura Modular do Kernel

- Muitos sistemas operacionais modernos implementam módulos carregáveis de kernel
  - Usa abordagem orientada a objetos
  - Cada componente principal é separado
  - Cada componente fala com os outros por meio de interfaces conhecidas
  - Cada componente é carregado conforme necessário dentro do kernel
- No geral, semelhante a camadas, mas com mais flexibilidade
  - Linux, Solaris, etc

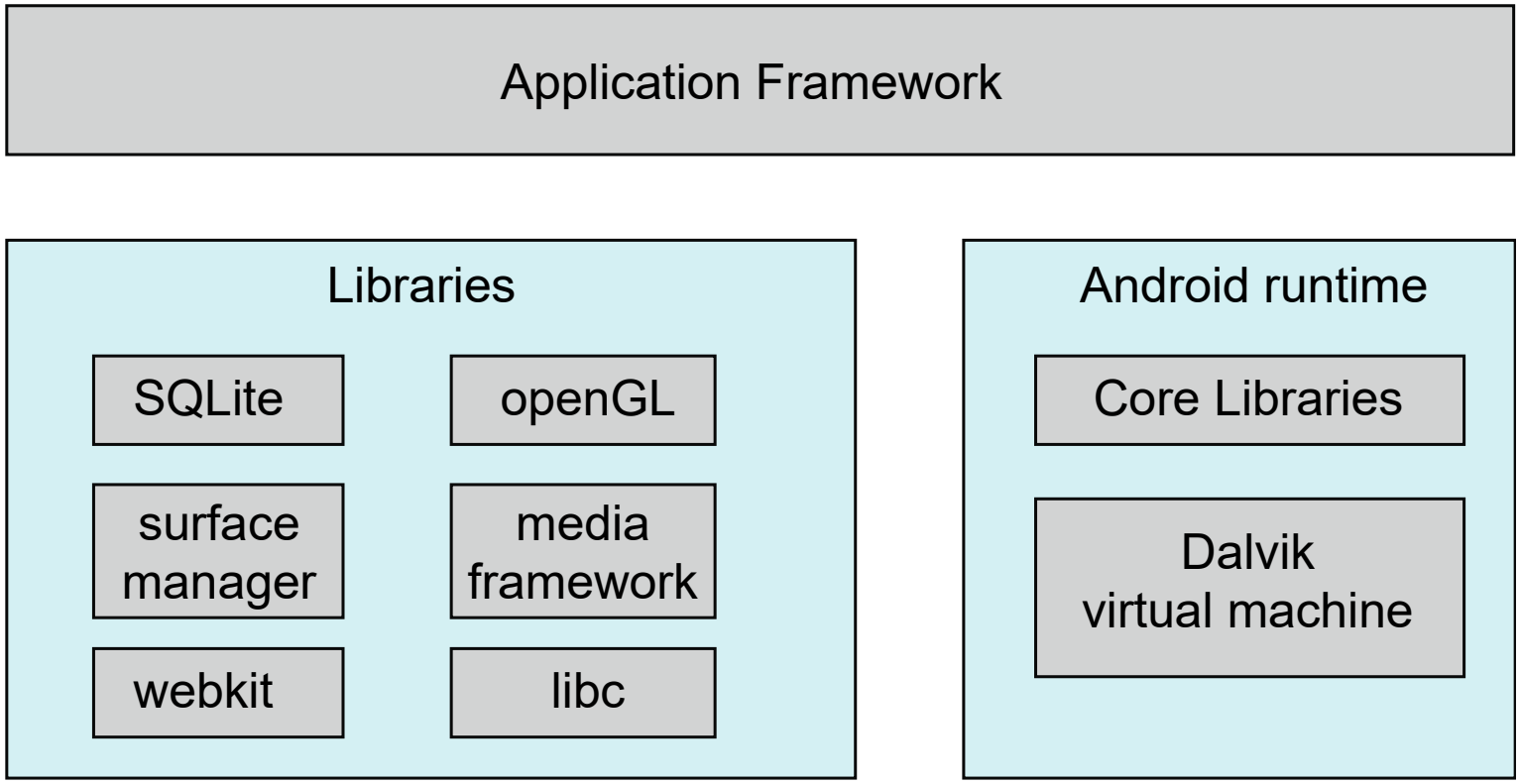
Estruturas Híbridas

- A maioria dos sistemas operacionais modernos não seguem um único tipo de estrutura
- Estruturas híbridas combinam várias abordagens para tratar do desempenho, da segurança e das necessidades de usabilidade
- Os kerneis do Linux e Solaris são monolíticos, mas permitem modularidade para carregamento dinâmico de funcionalidade
- O Windows é principalmente monolítico, mas suporta microkernel para diferentes personalidades do subsistema

Android

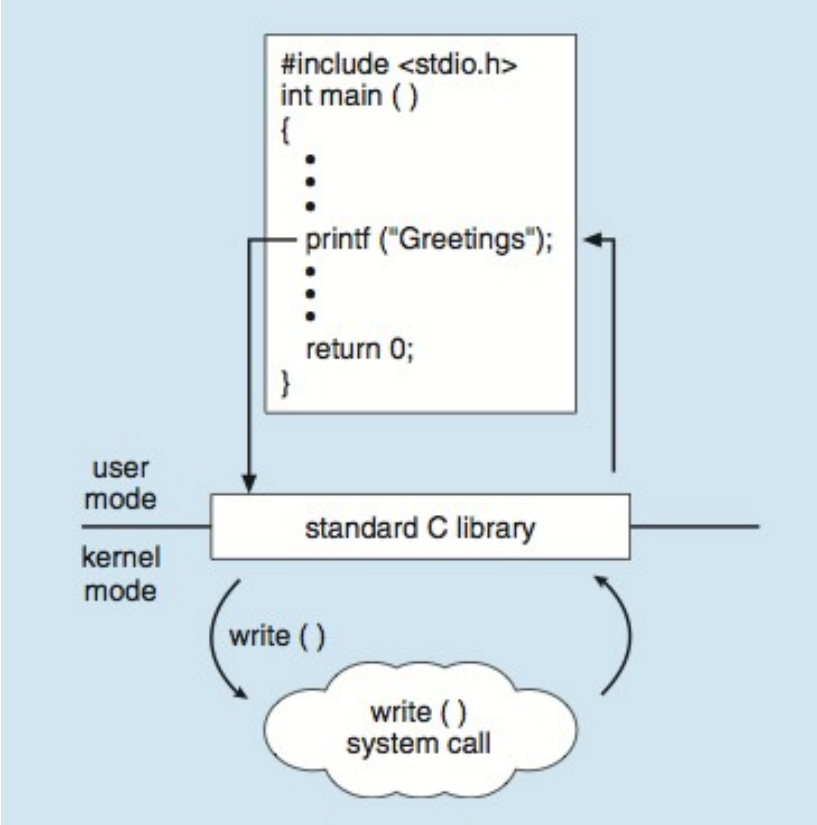
- Desenvolvido pela Open Handset Alliance (principalmente Google) - Código aberto
- Baseado no kernel do Linux, mas modificado
- Fornece gerenciamento de processo, memória e driver de dispositivo - adiciona gerenciamento de energia
- Ambiente de tempo de execução inclui o conjunto principal de bibliotecas e a máquina virtual Dalvik
- Aplicativos desenvolvidos em Java mais a API do Android
- Arquivos de classe Java compilados para bytecode Java, em seguida, convertidos em executáveis da máquina virtual Dalvik
- Bibliotecas incluem frameworks para navegador web (webkit), banco de dados (SQLite), multimídia, libc menor

Estrutura do Android



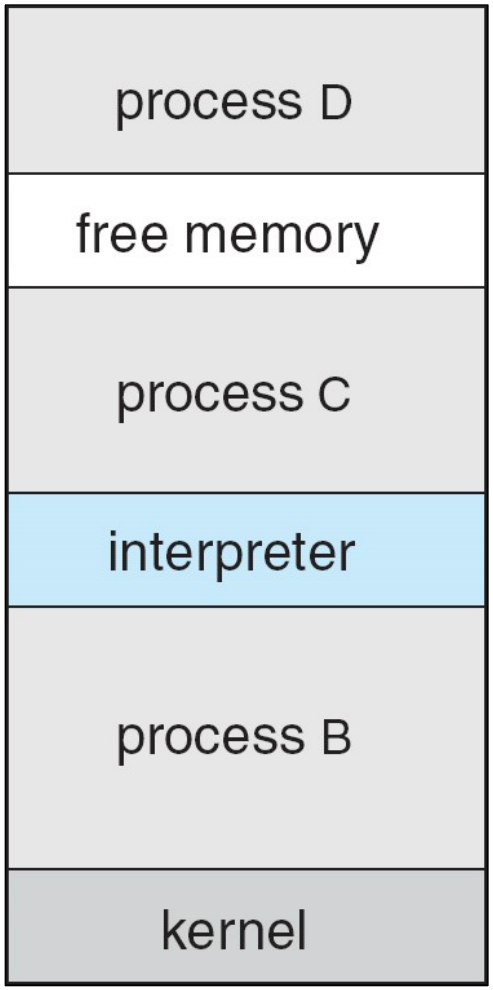
Exemplo de programa C

- Exemplo de programa C chamando a rotina ***printf***() que invoca a chamada de sistema ***write***()



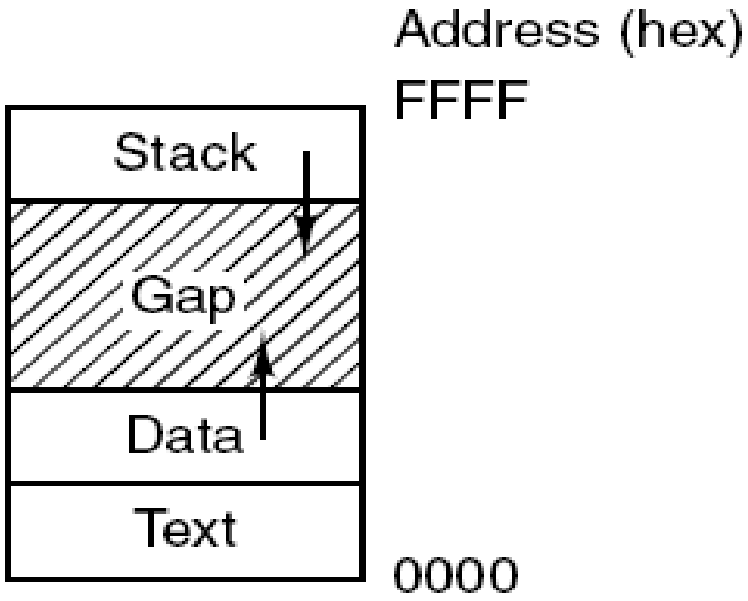
Execução de Programa no UNIX

- Multitarefa
- Execução de um programa:
- O shell executa uma chamada de sistema ***fork***() para criar um processo
  - Então executa uma chamada ***exec***() to carregar o programa no processo
  - Após o shell espera pelo processo terminar (ou continua tratando de comandos de usuário, opção &)
- Processos terminam com:
  - código = 0 – sem error
  - código > 0 – código do erro



Exemplo de processo na memória

Processos usualmente têm três segmentos:
text (código), data (dados) e stack (pilha)



Questões sobre processos

- Problemas:
  - Como proteger os programas uns dos outros e o núcleo de todos eles?
  - Como tratar a realocação?
- O primeiro é mais fácil de entender;
- Segundo: o compilador e linkeditor não sabem previamente a posição onde o programa ficará na memória física;

Questões sobre processos (2)

- Eles pressupõem que o programa estará no endereço 0;
- Mas se o programa e dados forem colocados no endereço 50000 e for requisitada a posição 10000, vai falhar;

Questões sobre processos (3)

- Alterar todos os endereços do programa compilado;
- Por exemplo, se há uma instrução que acessa a posição 10000 da memória, mas o programa foi colocado no endereço 50000, essa instrução seria alterada para acessar a posição 60000;
- Problema: fazer isso para todo o programa é complexo

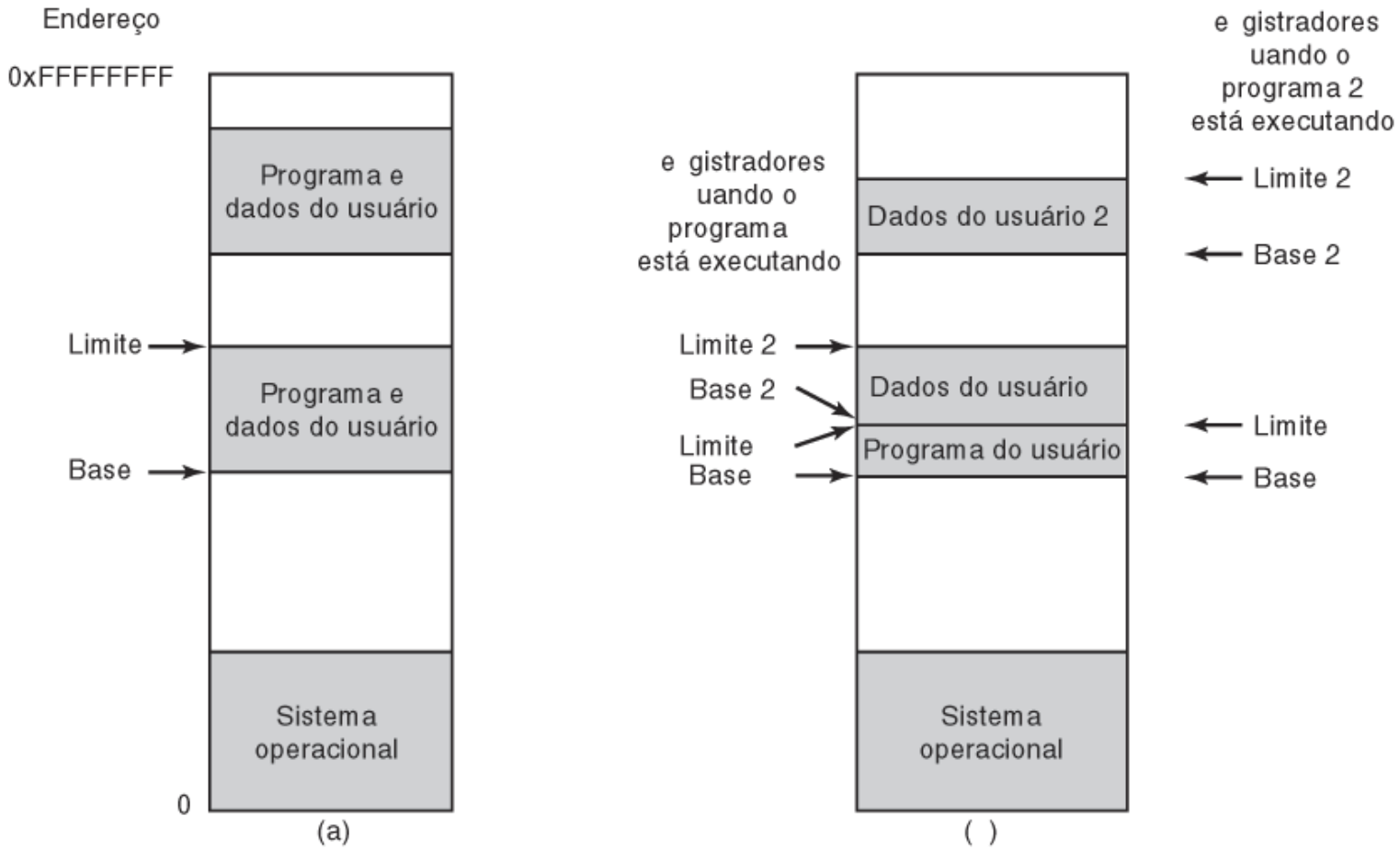
Carga e proteção de processos

- Solução: usar registrador-base e registrador-limite;
- Isso resolve também o problema de segurança: um programa não pode endereçar abaixo do registrador-base nem acima do registrador-limite;

Carga e proteção de processos (1)

- A verificação e mapeamento: conversão do endereço virtual (gerado pelo compilador) para o endereço físico (endereço de memória);
- Isso é realizado pela MMU (Unidade de Gerência de Memória);

Carga e proteção de processos (2)



Um par base-limite e dois pares base-limite