INE5412 Sistemas Operacionais I

L. F. Friedrich

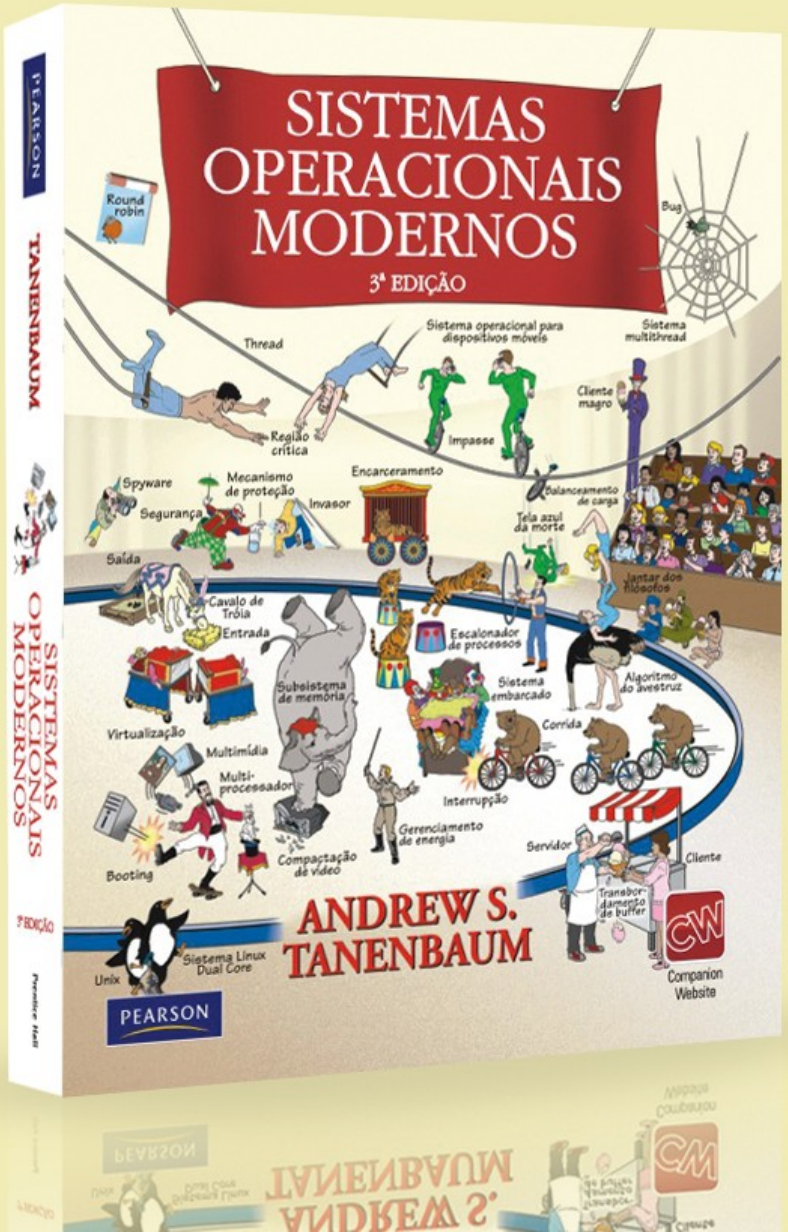
Introdução

Sistemas operacionais modernos

Terceira edição

ANDREW S. TANENBAUM

Capítulo 1
Introdução



O que é um sistema operacional?

Um computador moderno consiste em:

- Um ou mais processadores.
- Memória principal.
- Discos.
- Impressoras.
- Diversos dispositivos de entrada e saída

Para gerenciar todos esses componentes é necessária uma camada de software – o **sistema operacional.**

O que é um sistema operacional?

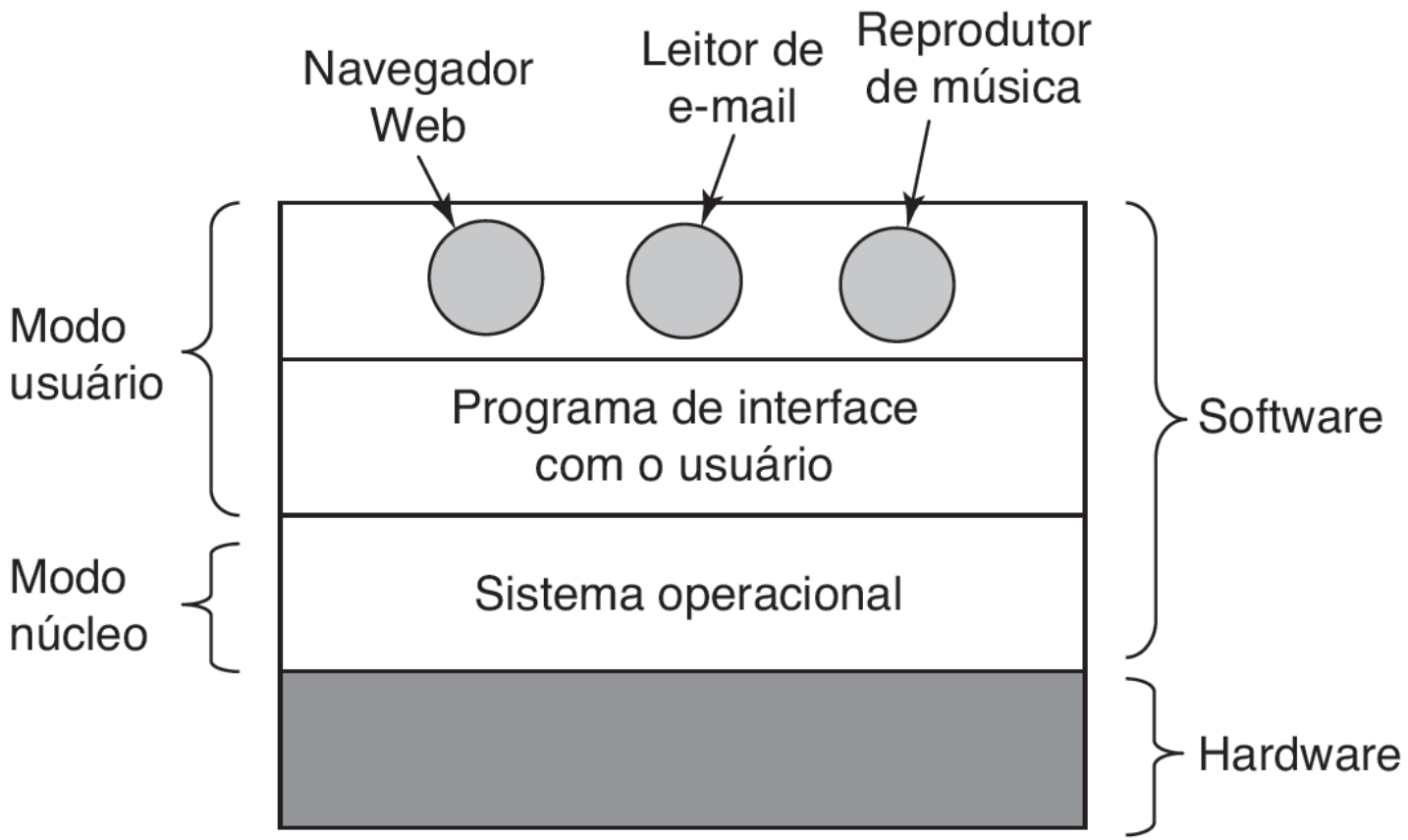


Figura 1.1 Onde o sistema operacional se encaixa.

O sistema operacional como uma máquina estendida

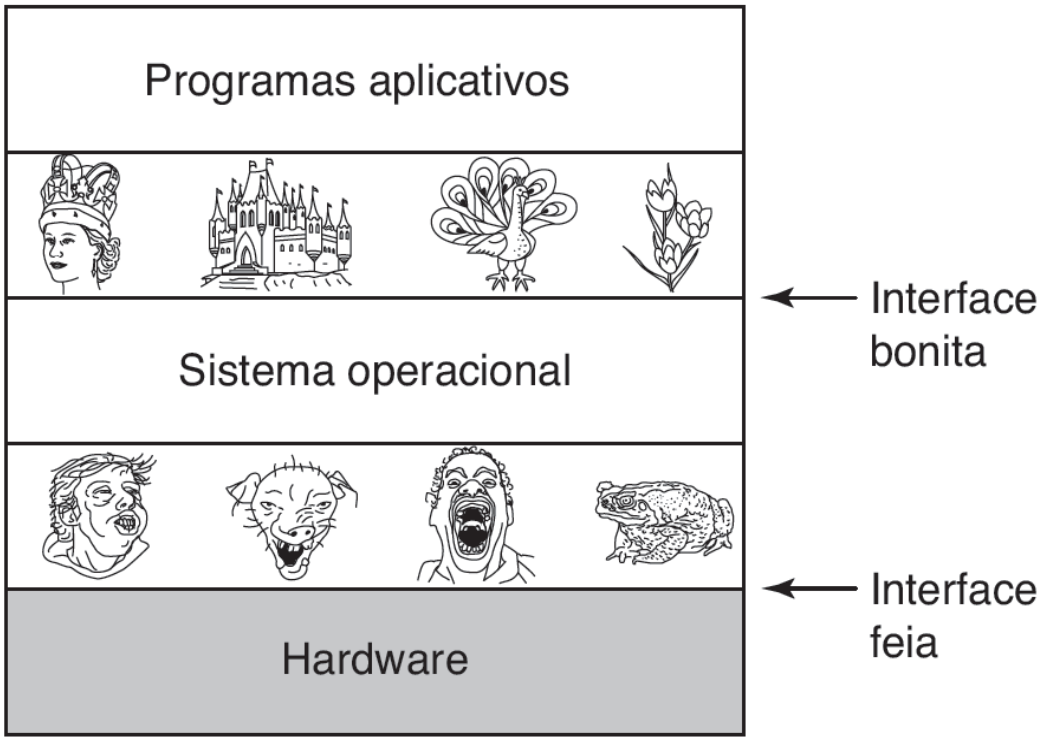


Figura 1.2 Sistemas operacionais transformam hardware feio em abstrações bonitas.

O sistema operacional como um gerenciador de recursos

- Permite que múltiplos programas sejam executados ao mesmo tempo
- Gerencia e proteje a memória, os dispositivos de entrada e saída e outros recursos
- Inclui a multiplexação (partilha) de recursos de duas maneiras diferentes:
  - No tempo
  - No espaço

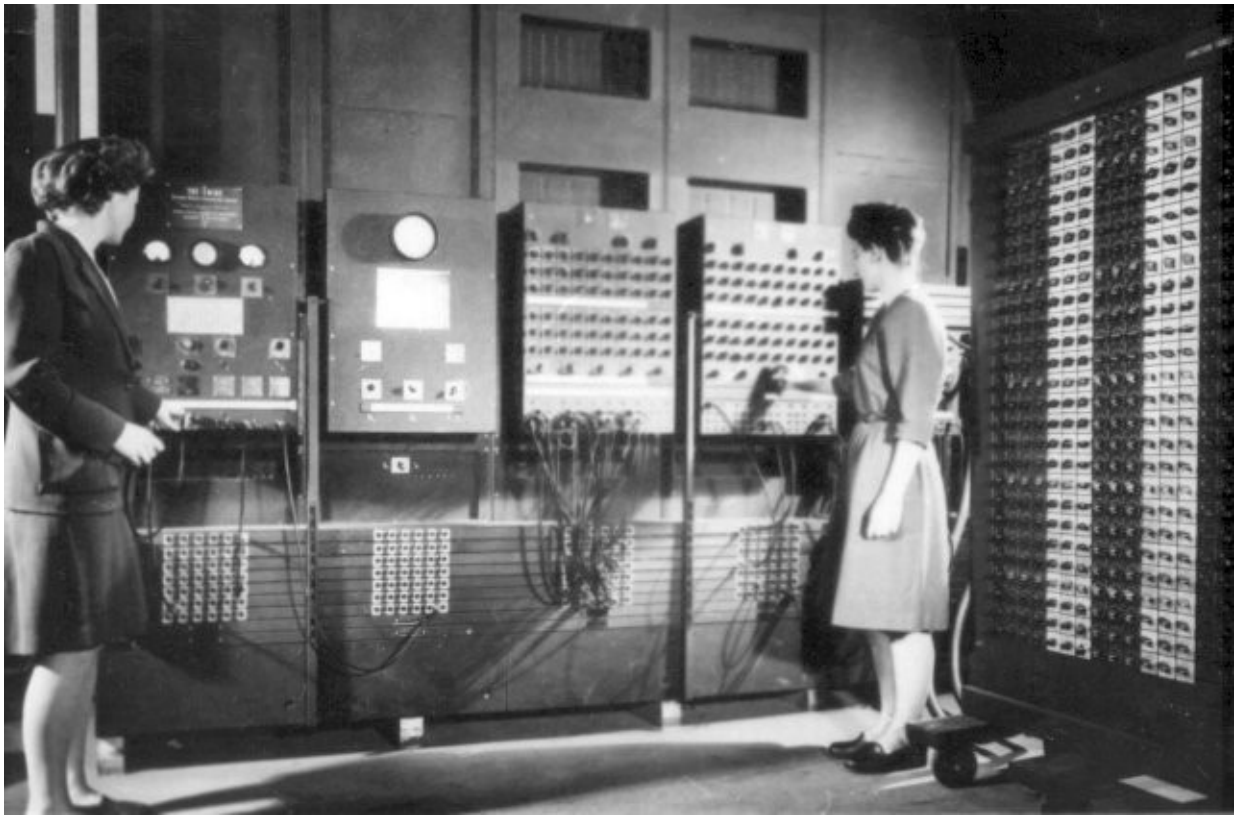
História dos Sistemas Operacionais

- O desenvolvimento de SO acontece junto com o desenvolvimento do computador.
- Avanços dos computadores requerem avanços correspondentes de sistemas operacionais.
- 4 gerações

1ª. Geração

- 1945-1955, Válvulas
  - Programação era feita em linguagem de máquina absoluta.
  - Sem linguagem de programação ou SOs.

1ª. Geração



Source: http://en.wikipedia.org/wiki/ENIAC

2ª. Geração

- 1955-1965, Transistores e Sistemas em lotes (Batch Systems)
  - The IBM mainframe's world.
  - Programas escritos em cartões perfurados.
  - Programas executavam em lotes (batch).
    - Junta vários jobs;
    - Executa os jobs um a um.
  - Principalmente para computações científicas e comerciais.

Transistores e sistemas em lotes (batch)

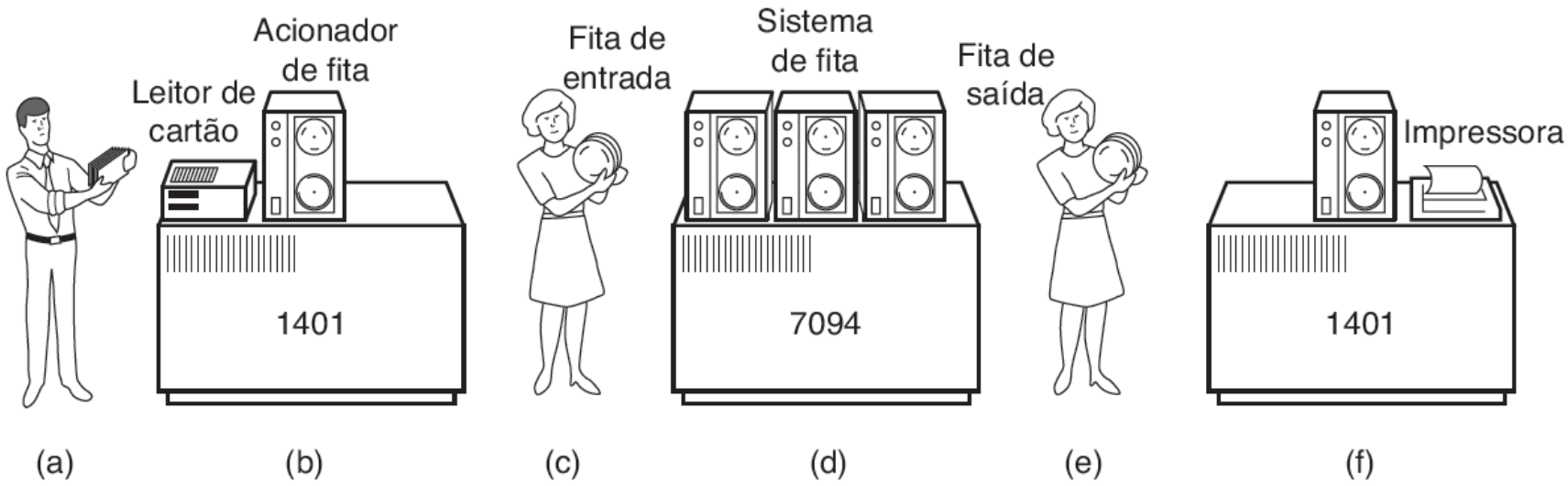


Figura 1-3 Um sistema em lotes antigo.

(a) Os programadores levam os cartões para o 1401.
(b) O 1401 grava os lotes de tarefas nas fitas.

2ª. Geração

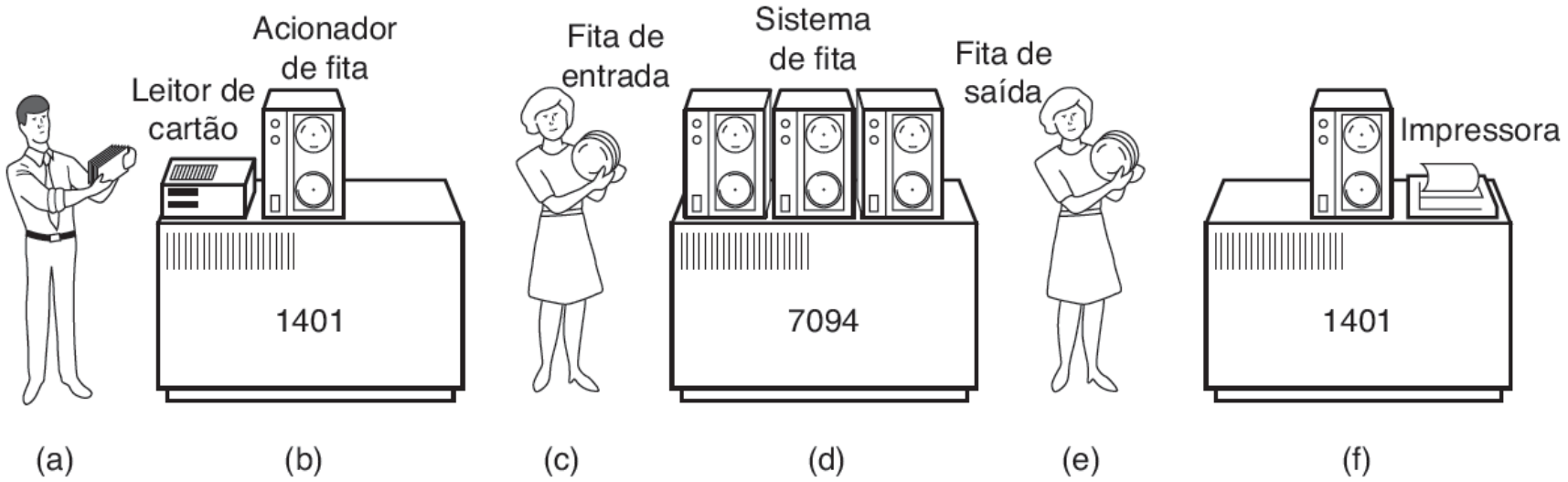


Figura 1-4.

(c) O operador leva a fita de entrada para o 7094.

(d) 7094 executa o processamento.

(e) O operador leva a fita de saída para o 1401.

(f) 1401 imprime as saídas.

2ª. Geração

- Fortran Monitor System – um primeiro SO.

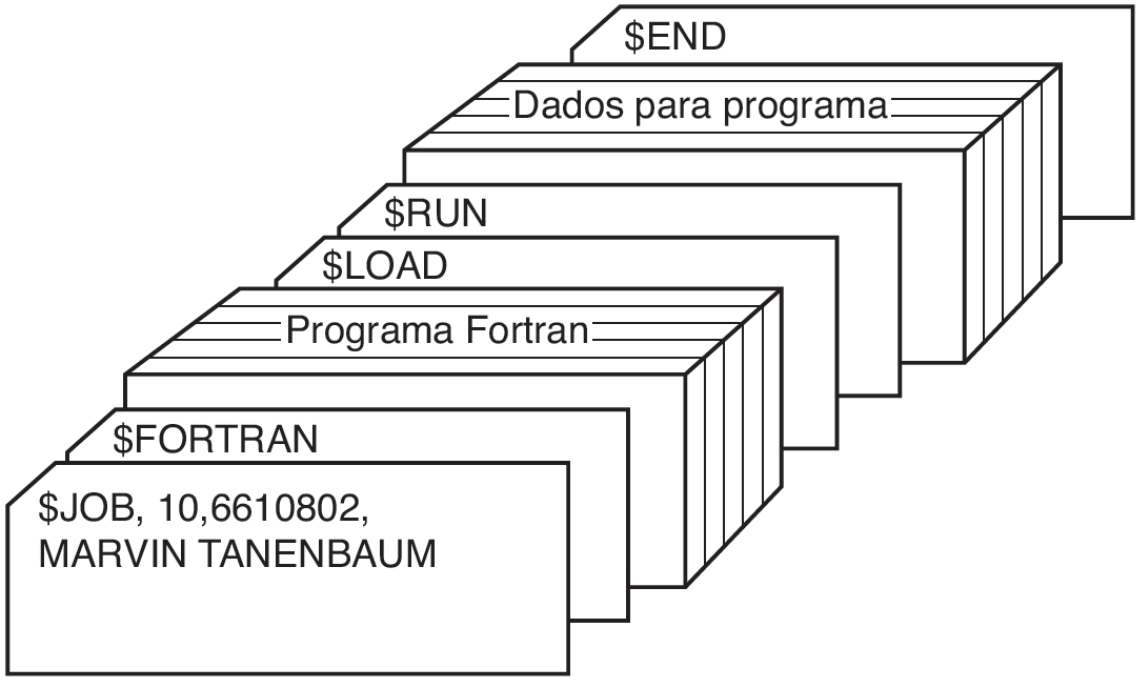


Figura 1.4 Estrutura de uma tarefa típica FMS.

3ª. Geração

- 1966-1980, CIs e Multiprogramação
  - Dividir a memória em várias partições, cada uma com um job diferente.
  - Quando um job esta esperando E/S, a UCP troca para trabalhar em outro job.
  - Mantém a UCP tão ocupada quanto possível , ~100%.

3ª. Geração

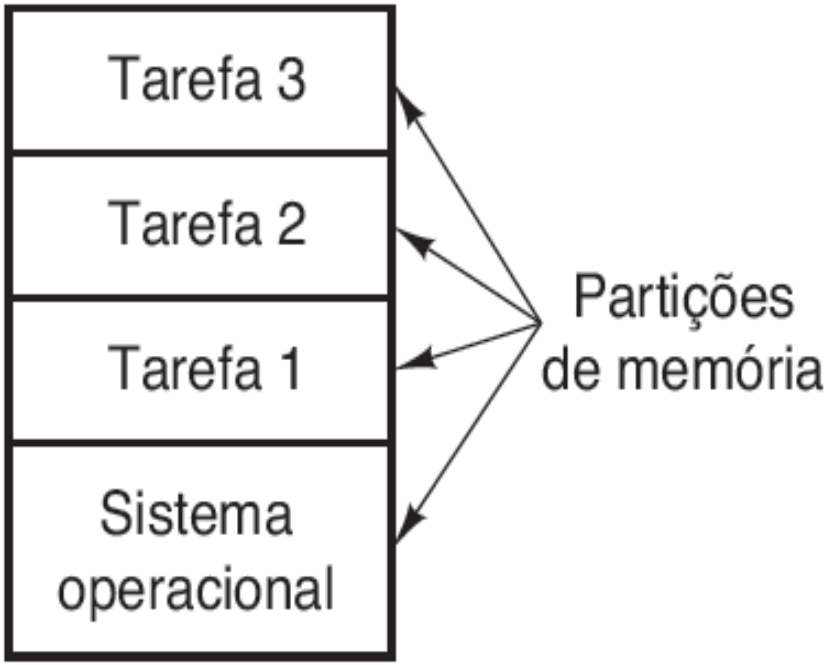


Figura 1.5 Um sistema de multiprogramação com três tarefas na memória.

3ª. Geração

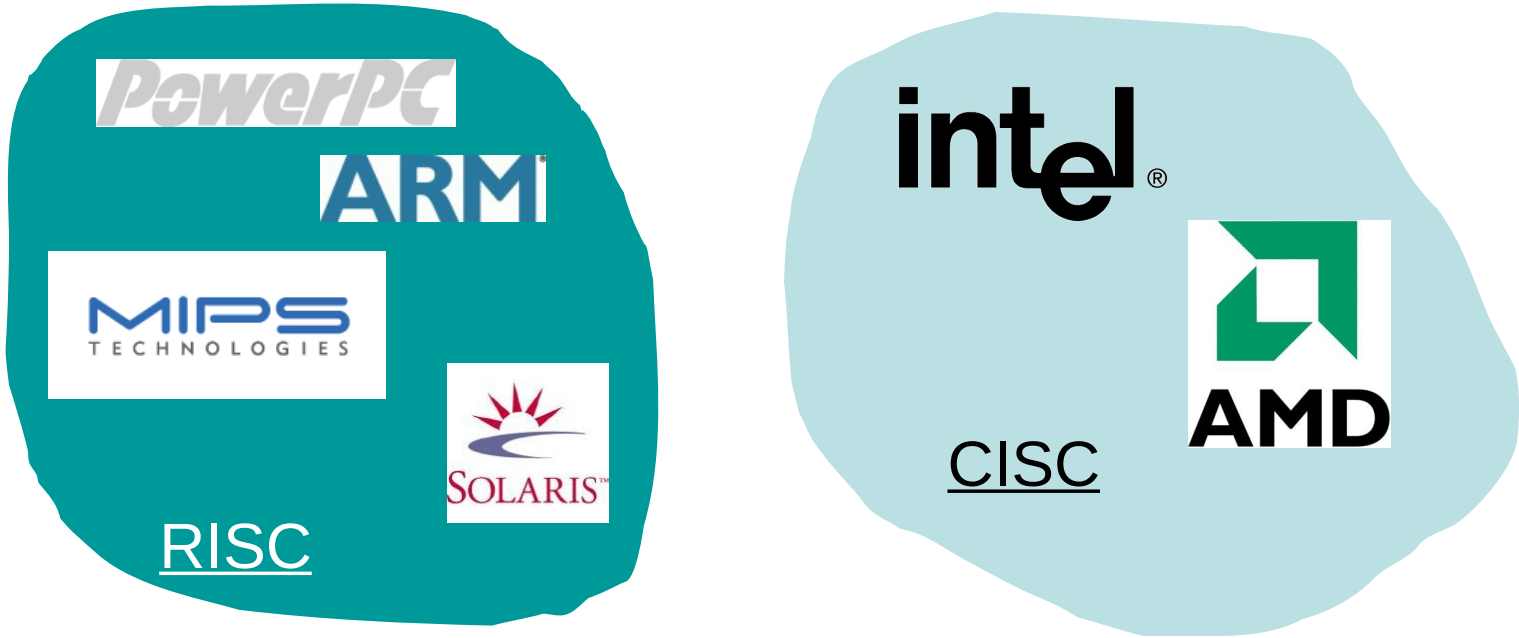
- O nascimento do sistema time-sharing.
  - Cada usuário tem um terminal online.
  - Cada um deles tem a ilusão de ter o computador todo o tempo.
  - Na verdade o computador esta servindo múltiplos usuários simultaneamente, usando o tempo ocioso de alguns para servir outros.
  - Permite jobs interativos.
- UNIX emerge.

4ª. Geração

- 1980-presente: Computadores
  - Idade do microprocessador.
  - Aliança IBM-Microsoft ;
  - Apple II;
  - etc.

Sistemas Operacionais Modernos

Tipos de Computadores



RISC – **R**educed **I**nstruction **S**et **C**omputer

CISC – **C**omplex **I**nstruction **S**et **C**omputer

CISC vs RISC

- Diferentes tamanhos de programas.
  - RISC é maior que CISC.
- Diferentes velocidades de processamento.
  - Uma instrução CISC instruction consome mais ciclos de relógio.

Reference: http://cse.stanford.edu/class/sophomore-college/projects-00/risc/risccisc/

O zoológico de sistemas operacionais

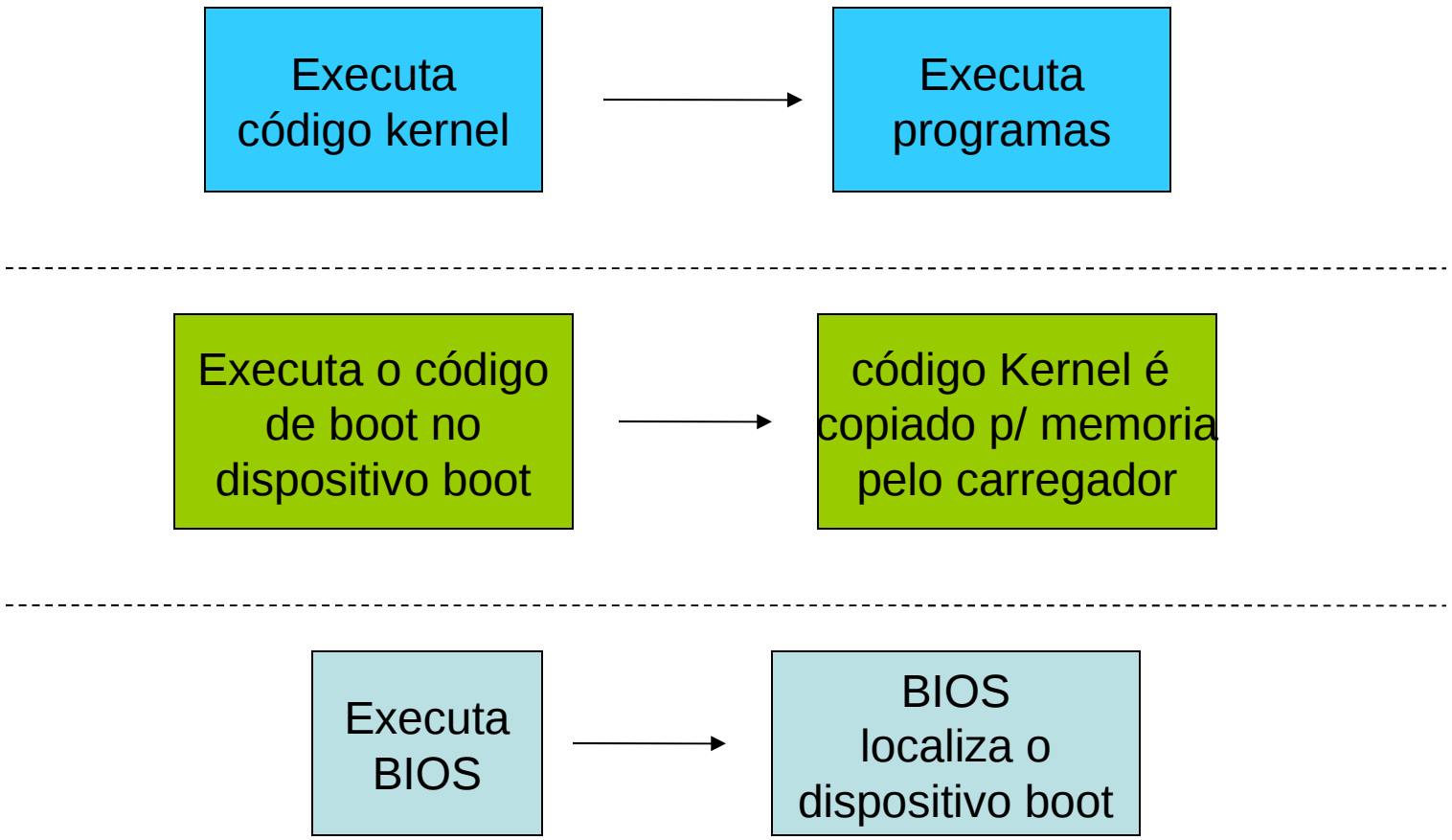
- Computadores de grande porte (computação tradicional).
- Servidores (WEB, base de dados).
- Computadores pessoais desktop (computação pessoal).
- computadores portáteis notebooks(computação pessoal móvel) .
- Computadores Handheld, PDAs(informações portável)
- Embarcados, tempo real(máquinas pequenas e não de PG-microondas, computadores usáveis)
- Sistemas de nós sensores (sensor node).
- Sistemas de tempo real.
- Smart card(telefones móveis, cartões de banco).

http://en.wikipedia.org/wiki/List_of_operating_systems

Sistemas de Tempo-Real

- Frequentemente usados como dispositivos de controle em aplicação dedicada:controle de experimentos científicos, imagens médicas, sistemas de controle industrial, etc.
- Restrições temporais bem definidas e fixas.
- Sistemas de tempo-real podem ser *hard* ou *soft* .

O que acontece quando ligamos o computador?



Um Sistema Operacional em ação

- CPU carrega programa *boot* da ROM (BIOS no PC)

- Programa *boot*
  - Examina/verifica configuração da máquina (no. de CPU's, quanta memória, no. e tipo de dispositivos de hw, etc..) conhecido como POST – Power On Self Test.
  - Monta uma estrutura de configuração descrevendo o hardware
  - Carrega o SO e passa a estrutura de configuração

- Inicialização do sistema operacional
  - Inicializa estruturas de dados do kernel
  - Inicializa o estado de todos os dispositivos de hw
  - Cria um numero de processos para iniciar operação (getty no UNIX, sistema de janelas no NT), incluíndo "deamons".

Um Sistema Operacional em ação

- Depois dos processos básicos iniciarem, o SO executa programas de usuário, se disponíveis, senão entra em espera (*idle loop*)

- No *idle loop*
  - SO executa um loop infinito (UNIX)
  - SO realiza funções de gerencia de sistema e *profiling*
  - SO pára o processador (*halt*) e entra no modo baixo consumo (*notebooks*)

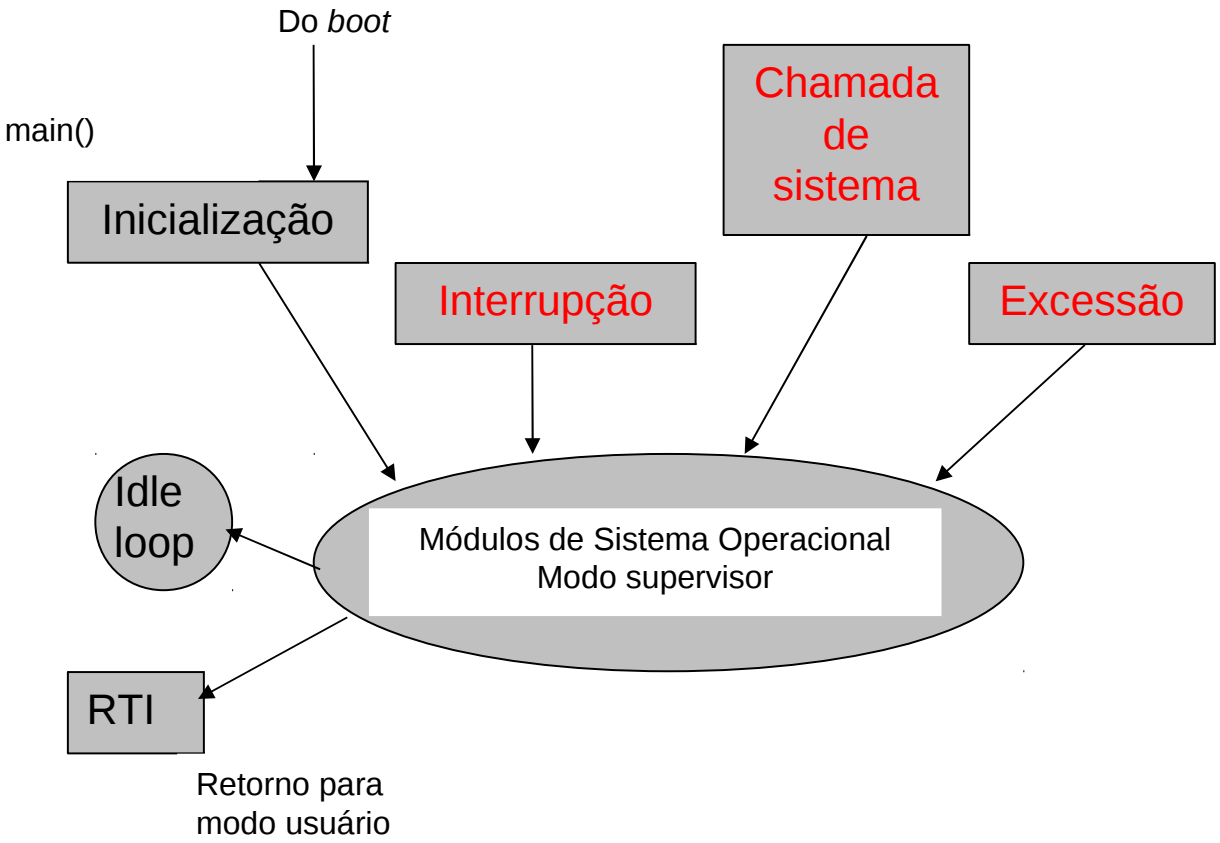
- SO acorda quando ocorre:
  - Interrupção dos dispositivos de hardware
  - Excessão de programas de usuário
  - Chamada de sistema de programas de usuário

- Dois modos de execução
  - Modo usuário: modo de execução restrito (aplicações)
  - Modo supervisor/kernel: acesso sem restrições (SO)

Modo Kernel vs Modo Usuário

- Modo Usuário
  - O tempo que o programa de usuário esta executando o seu código.
- Modo Kernel
  - O tempo que o programa de usuário invoca chamadas de sistema; ou
  - Espera E/S
    - Interrupção de hardware;

Fluxo de controle em um SO



Modo Kernel vs Modo Usuário

- Chamada de sistema
- A interface entre o SO (kernel) e os programas de usuário.
  - são funções oferecidas pelo kernel para programas de usuário.
    - da mesma forma que funções oferecidas por bibliotecas de usuário.
  - Ex., int read(int fd, char *buffer, int nbytes);
    - **read** – nome da chamada de sistema;
    - **fd** – descritor do arquivo;
    - **buffer** – nome do local de armazenamento ;
    - **nbytes** – numero de bytes a ler.

Chamadas de sistema

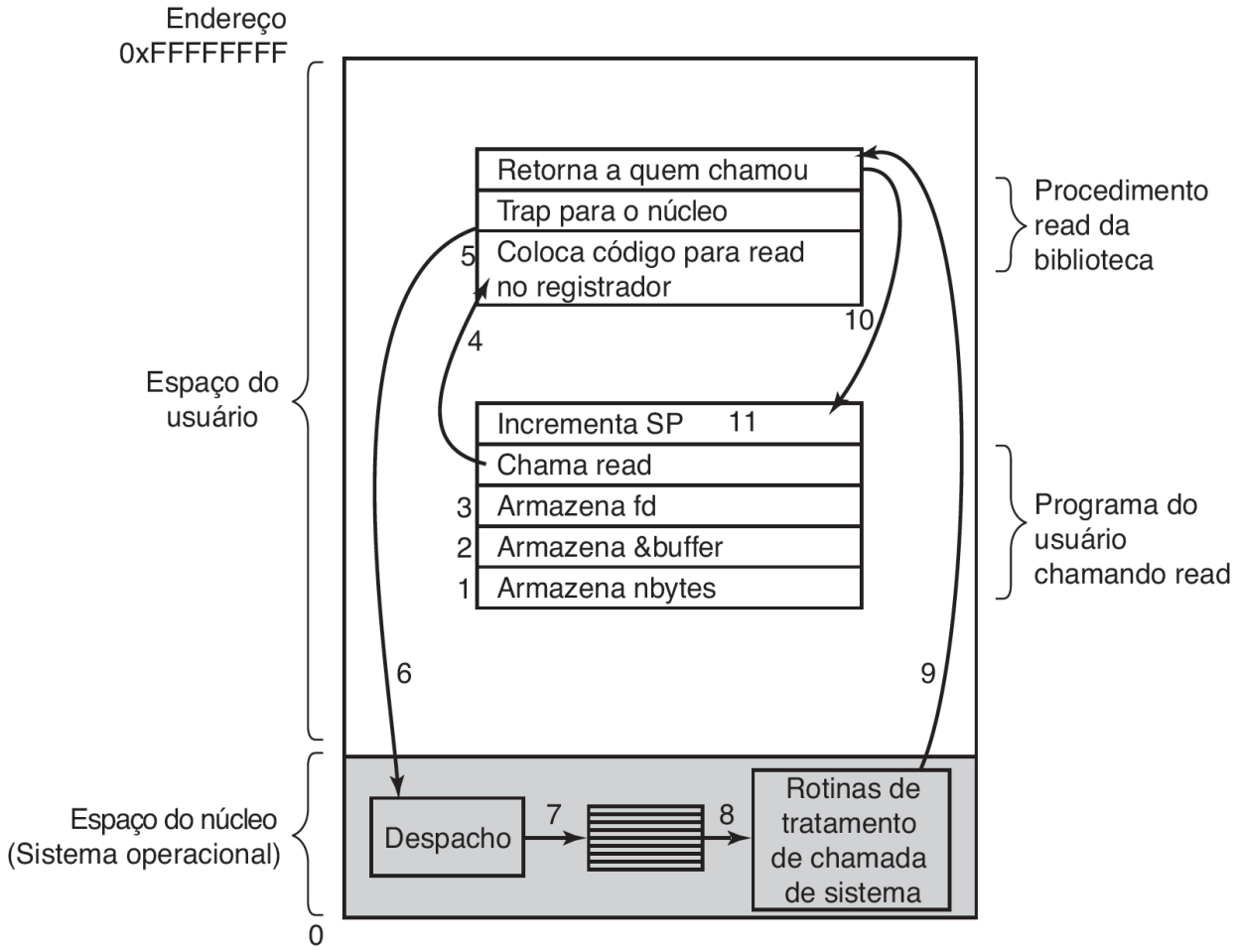


Figura 1.17 Os 11 passos na realização da chamada de sistema read (fd, buffer, nbytes).

Proteção para SO (hardware)

- Proteção de E/S
  - Instruções Privilegiadas
- Proteção de Memória
  - Não permite que a área de memória que contém o monitor seja alterada, programas são protegidos
  - Registrador Base/Limite
- Proteção de CPU
  - Previne que um job possa monopolizar o sistema
  - Timer/Clock

Principais componentes do SO

- Gerência de Processos
- Gerência de Recursos
  - CPU
  - Memoria
  - Dispositivo
- Gerência de Arquivos
- Interpretador de comandos

Processo

Um processo é um envelope que armazena todas as informações necessárias para executar um programa.

O SO é responsável pelas seguintes atividades com relação a gerência de processos(Ex. shell):

- Criação e deleção de Processos.
- Suspensão e retomada de processos.
- Fornecimento de mecanismos para:
  - sincronização de processo
  - comunicação de processo

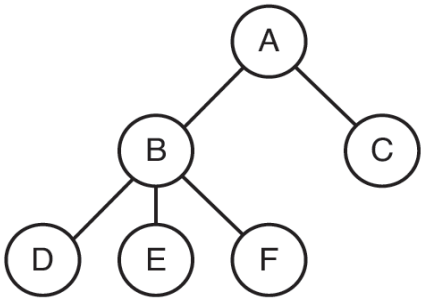


Figura 1.13 Uma árvore de processo. O processo *A* criou dois processos filhos, *B* e *C*. O processo *B* criou três processos filhos, *D*, *E* e *F*.

Chamadas de sistema para gerenciamento de processos

Gerenciamento de processos

Chamada	Descrição
pid = fork( )	Cria um processo filho idêntico ao pai
pid = waitpid(pid, &statloc, options)	Espera que um processo filho seja concluído
s = execve(name, argv, environp)	Substitui a imagem do núcleo de um processo
exit(status)	Conclui a execução do processo e devolve status

Tabela 1.1 Algumas das principais chamadas de sistema do POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é uma compensação no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

Memória

- O SO é responsável pelas seguintes atividades com relação a gerência de memória (Espaços de Endereçamento):
  - Manter informações de que partes da memória estão em uso e por quem.
  - Decidir que processos carregar quando espaços de memória estão disponíveis.
  - Alocar e liberar espaço de memória quando necessário.

Layout de memória

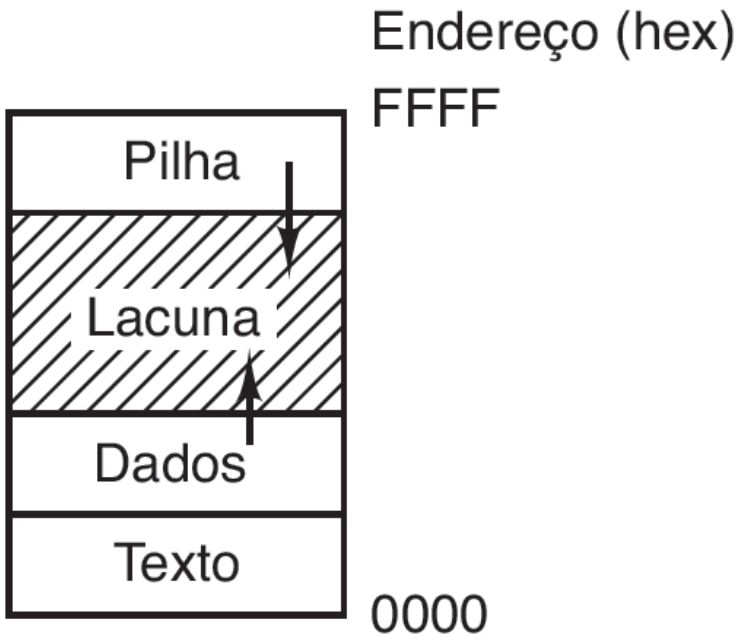


Figura 1.19 Os processos têm três segmentos: texto, dados e pilha.

Arquivos

- O SO é responsável pelas seguintes atividades com relação a gerência de arquivos:
  - Criação e deleção de arquivo.
  - Criação e deleção de diretorio.
  - Suporte de primitivas para manipular arquivos e diretórios.
  - Mapeamento de arquivos na memória secundária.

Arquivos

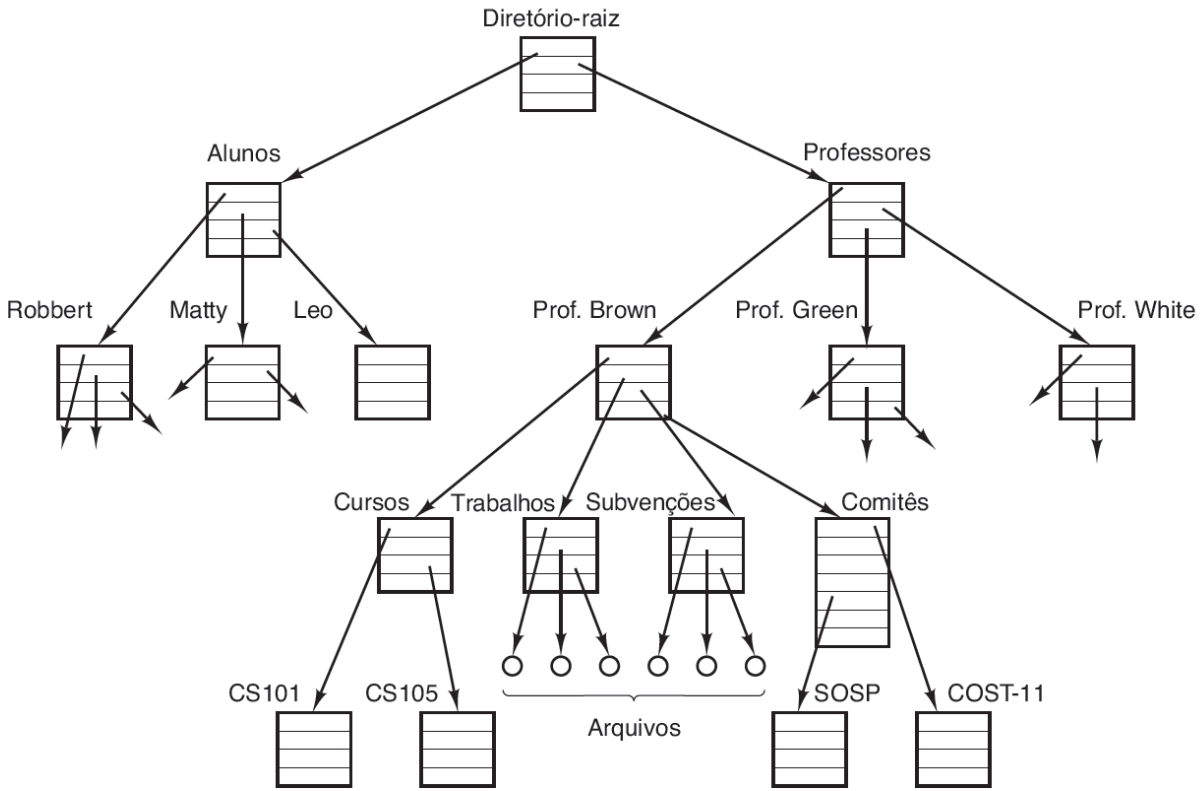


Figura 1.14 Sistema de arquivos para um departamento universitário.

Gerenciamento de arquivos

Chamada	Descrição
Fd = open(file, how, ...)	Abre um arquivo para leitura, escrita ou ambos
s = close(fd)	Fecha um arquivo aberto
n = read(fd, buffer, nbytes)	Lê dados a partir de um arquivo em um buffer
n = write(fd, buffer, nbytes)	Escreve dados a partir de um buffer em um arquivo
position = lseek(fd, offset, whence)	Move o ponteiro do arquivo
s = stat(name, &buf)	Obtém informações sobre um arquivo

Tabela 1.1 Algumas das principais chamadas de sistema do POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é uma compensação no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

Gerenciamento do sistema de diretório e arquivo

Chamada	Descrição
s = mkdir(name, mode)	Cria um novo diretório
s = rmdir(name)	Remove um diretório vazio
s = link(name1, name2)	Cria uma nova entrada, name2, apontando para name1
s = unlink(name)	Remove uma entrada de diretório
s = mount(special, name, flag)	Monta um sistema de arquivos
s = umount(special)	Desmonta um sistema de arquivos

Tabela 1.1 Algumas das principais chamadas de sistema do POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é uma compensação no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

E/S

- Existem vários tipos de dispositivos de Entrada/Saída: teclados, monitores e impressoras .
- O SO é responsável pelo gerenciamento desses dispositivos:
  - Possui um subsistems de E/S para gerenciar seus dispositivos
  - Alguns dos componentes de E/S são independentes de dispositivo, aplicam-se para todos os dispositivos
  - Outros são específicos de cada dispositivos (ou família), como drivers.

Um interpretador de comandos simples

```
#define TRUE 1

while (TRUE) {                                          /* repita para sempre     */
        type _prompt( );                                /* mostra prompt na tela  */
        read _command(command,     parameters);         /* lê entrada do terminal */

        if (fork( ) != 0) {                             /* cria processo filho    */
             /* Código do processo pai.     */
             waitpid( -1, &status, 0);                  /* aguarda o processo filho acabar */
        } else {
             /* Código do processo filho.     */
             execve(command, parameters, 0);            /* executa o comando      */
        }
}
```

Figura 1.18 Um interpretador de comandos simplificado. Neste livro, presume-se *TRUE* como 1.

Windows Win32 API

UNIX	Win32	Descrição
fork	CreateProcess	Cria um novo processo
waitpid	WaitForSingleObject	Pode esperar que um processo saia
execve	(nenhuma)	CreateProcess = fork + execve
exit	ExitProcess	Conclui a execução
open	CreateFile	Cria um arquivo ou abre um arquivo existente
close	CloseHandle	Fecha um arquivo
read	ReadFile	Lê dados a partir de um arquivo
write	WriteFile	Escreve dados em um arquivo
lseek	SetFilePointer	Move o ponteiro do arquivo
stat	GetFileAttributesEx	Obtém vários atributos do arquivo

Estrutura de sistemas operacionais

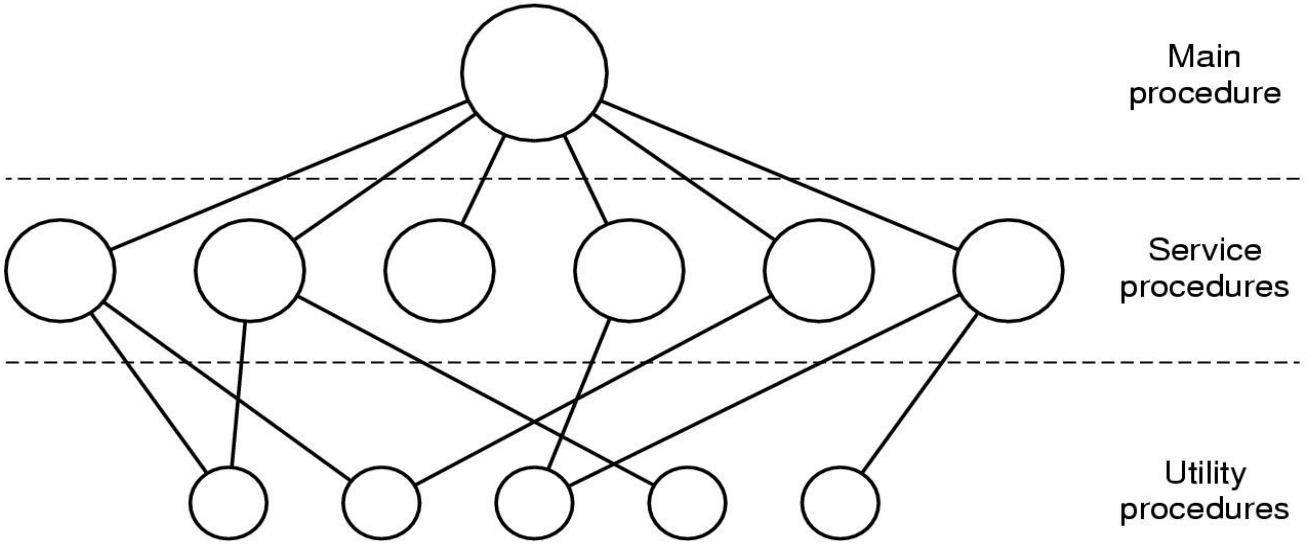
Enquanto a interface do sistema nos dá uma visão externa do sistema,a estrutura nos permite uma visão interna.

- Vamos examinar algumas estruturas :
  - Sistemas monolíticos, sistemas em camada, micronúcleo, sistemas cliente-servidor, máquinas virtuais.

Estrutura de sistemas operacionais

Sistemas monolíticos – estrutura básica:

- Um programa principal executa a rotina requerida.
- Um conjunto de rotinas de serviço que executa as chamadas de sistema.
- Um conjunto de rotinas de utilidade que ajudam as rotinas de serviço.



Estrutura de SO - Camadas

- Estrutura em níveis ou camadas – organiza o SO como uma hierarquia de camadas
- Primeiro sistema construído assim – THE – seis camadas

Camada	Função
5	O operador
4	Programas de usuário
3	Gerenciamento de entrada/saída
2	Comunicação operador–processo
1	Memória e gerenciamento de tambor
0	Alocação do processador e multiprogramação

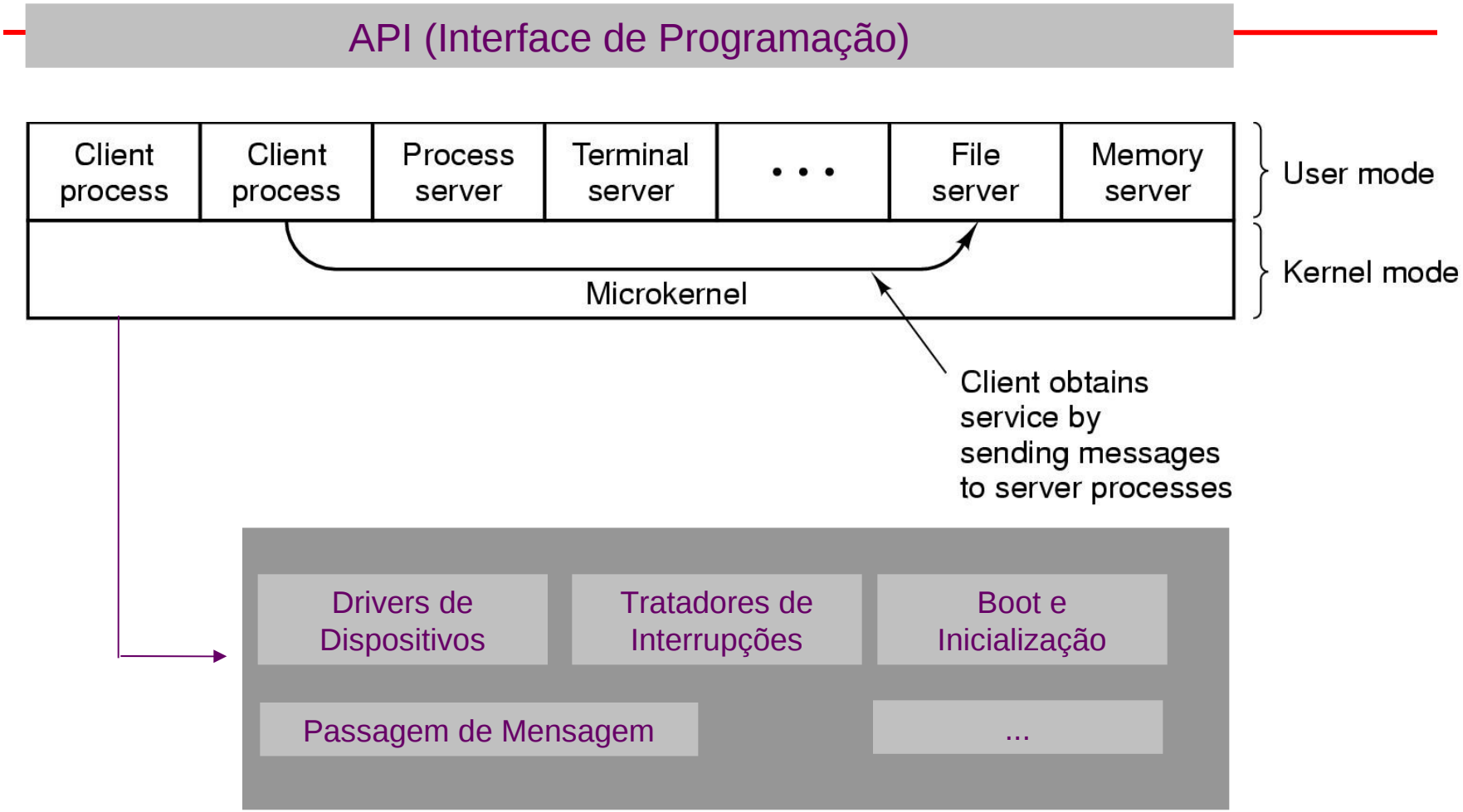
Tabela 1.3 Estrutura do sistema operacional THE.

- Camada 0 responsável pela multiprogramação – acima processos
- Camadas 1,2 e 3 tarefas de gerencia e facilidade de utilização
- Camada 4 programas – sem preocupação com gerencia

Arquitetura Microkernel

- A principio todas as camadas entram no núcleo, mas existem argumentos fortes (bugs) para se colocar o mínimo possível, ou seja, mover funcionalidade do núcleo para espaço do usuário.
- Idéia principal é alta confiabilidade, dividindo o sistema em módulos pequenos, bem definidos, sendo apenas um módulo (micronúcleo) executado em modo núcleo.
- Aplicações TR, industriais, aviônicas e militares. Alguns sistemas: Integrity, K42, L4, PikeOS, QNX, Symbian
- Benefícios:
  - facilidade de extender um microkernel
  - facilidade de portar o SO para novas arquiteturas
  - mais confiável (menos código executando em modo kernel)
  - mais seguro

Estrutura de SO – Microkernel



Micro-kernel (Mach, Exokernel)

Modelo cliente-servidor

- Variação da idéia de microkernel, distingue duas classes de processos: **servidores** e **clientes**. A camada inferior é o micronúcleo.
- Generalização desta idéia é executar clientes e servidores em computadores diferentes, conectados por uma rede local por exemplo.

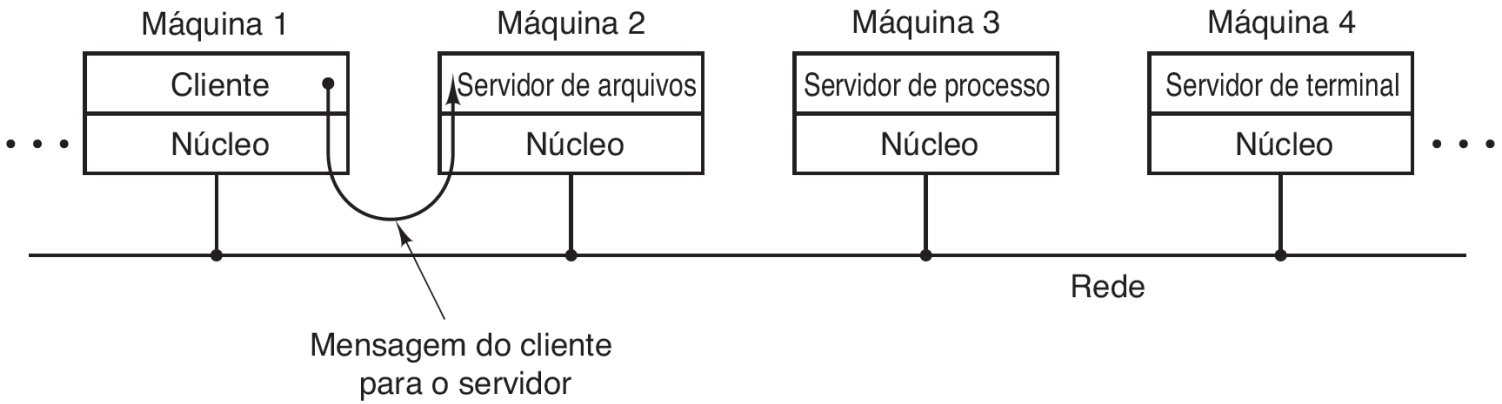


Figura 1.24 O modelo cliente–servidor em uma rede.

Máquinas virtuais

- A idéia de *máquina virtual* da IBM busca fornecer multiprogramação e uma máquina estendida:
  - CMS – conversational monitor system
  - VM/370 – Virtual Machine
- A máquina virtual é o coração do sistema e é executado diretamente sobre o hardware e implementa a multiprogramação.
- Ao contrário dos demais sistemas operacionais não são máquinas estendidas, são cópias exatas do hardware. Cada uma delas pode executar qualquer sistema operacional.
  - Chamadas de sistema são encaminhadas para CMS
  - E/S são encaminhadas para a VM

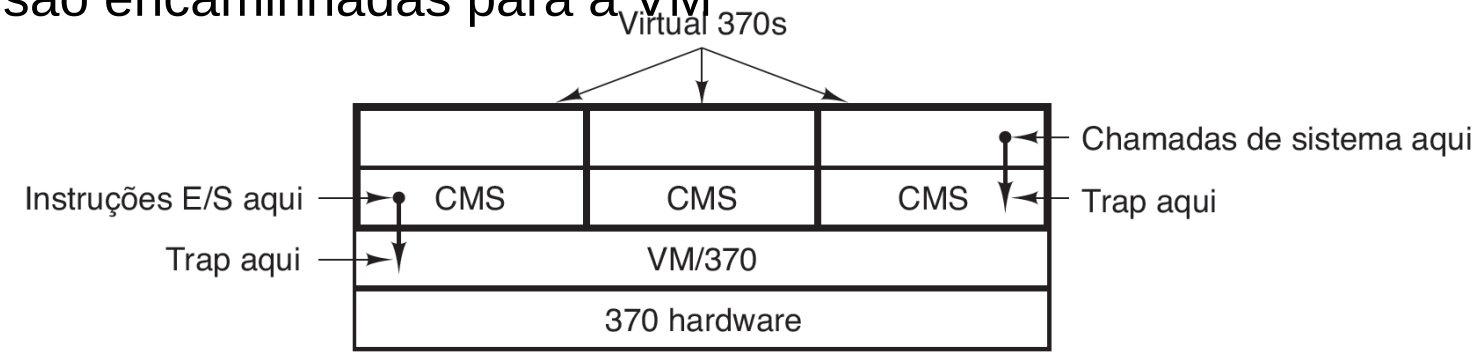


Figura 1.25 Estrutura do VM/370 com CMS.

Máquinas virtuais redescobertas

- Virtualização foi ignorada até pouco tempo. Novas necessidades e tecnologias, novo interesse.
- Industria de servidores, industria de hospedagem, usuários finais.
- O problema: implementação. Para executar o software de máquina virtual em um computador, sua CPU deve ser virtualizável (Popek e Goldberg, 1974).
- Projetos e pesquisa acadêmica, 1990, retomada do interesse (Vmware) .
- Instruções de controle so sistema são substituídas por chamadas ao hipervisor (virtualização) ou removidas (paravirtualização)

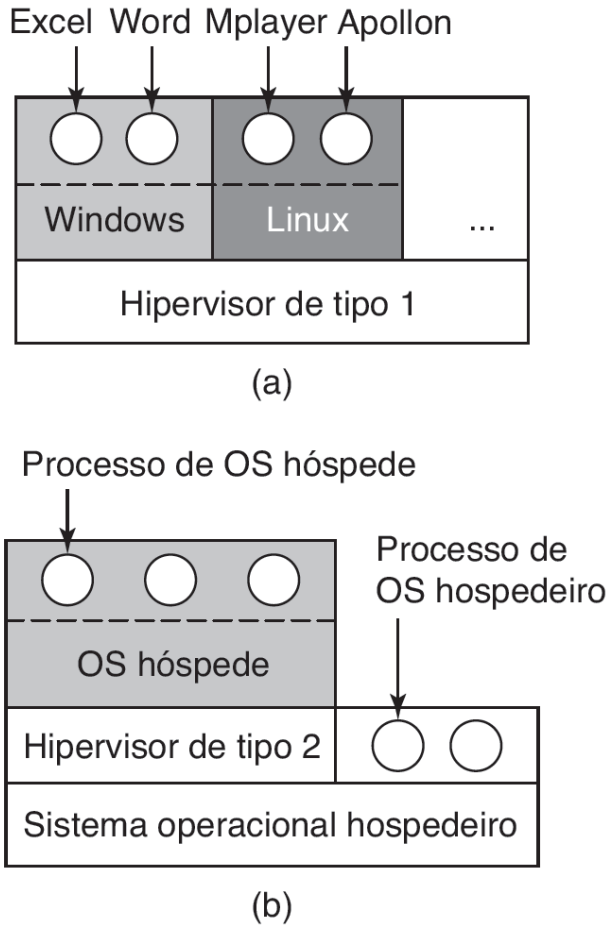


Figura 1.26 (a) Hipervisor de tipo 1. (b) Hipervisor de tipo 2.

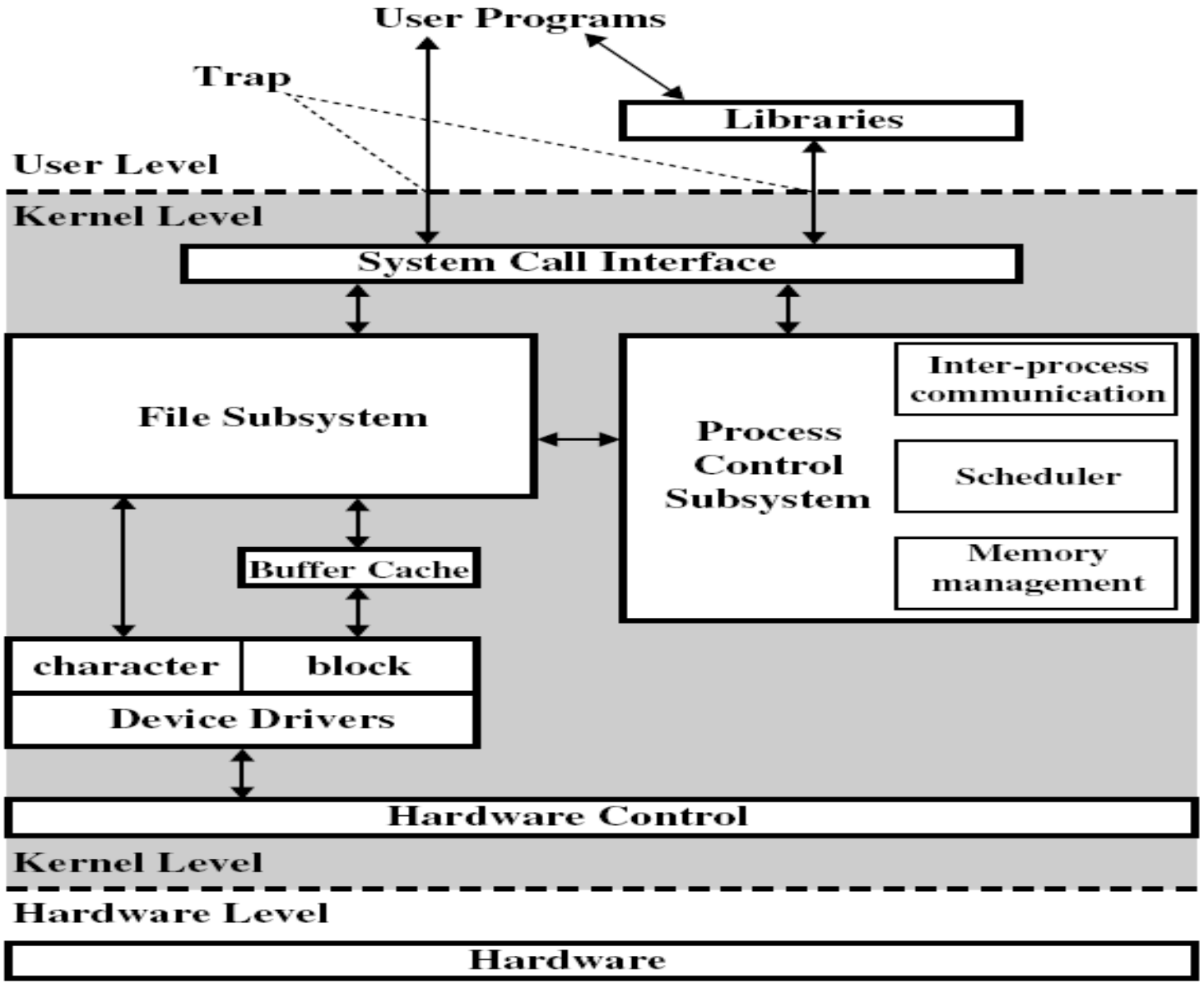


Figure 2.16 Traditional UNIX Kernel [BACH86]

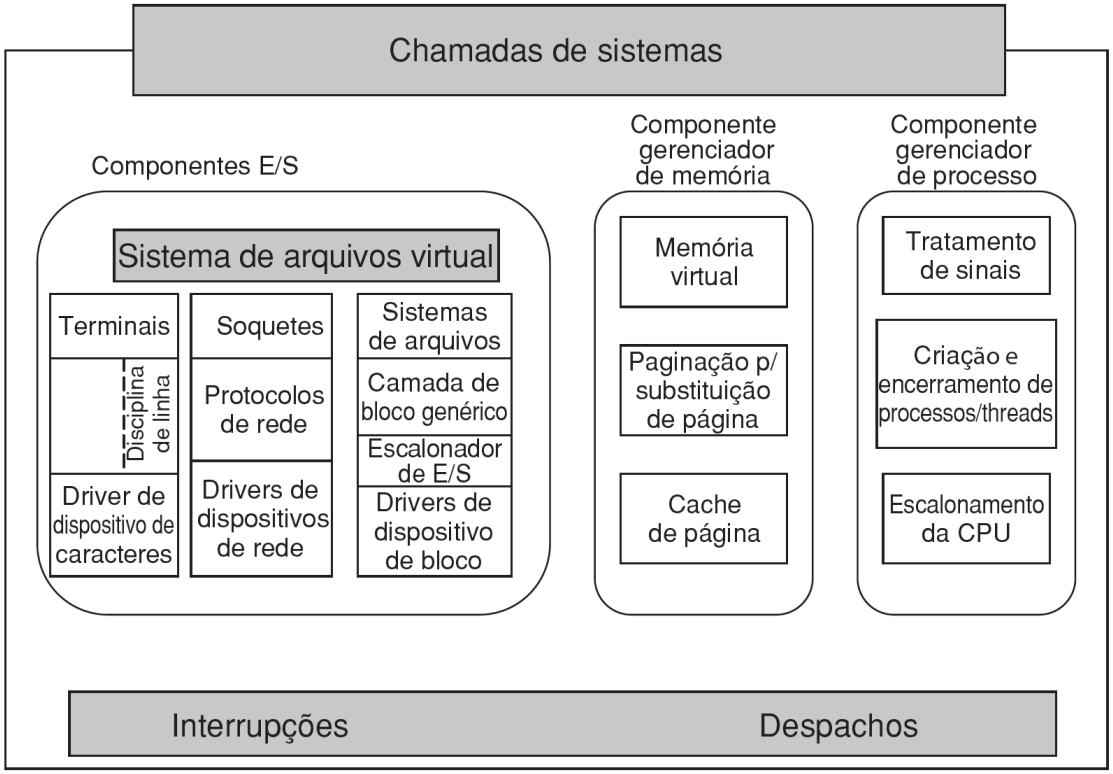


Figura 10.2 Estrutura do núcleo do Linux.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Estrutura do sistema operacional

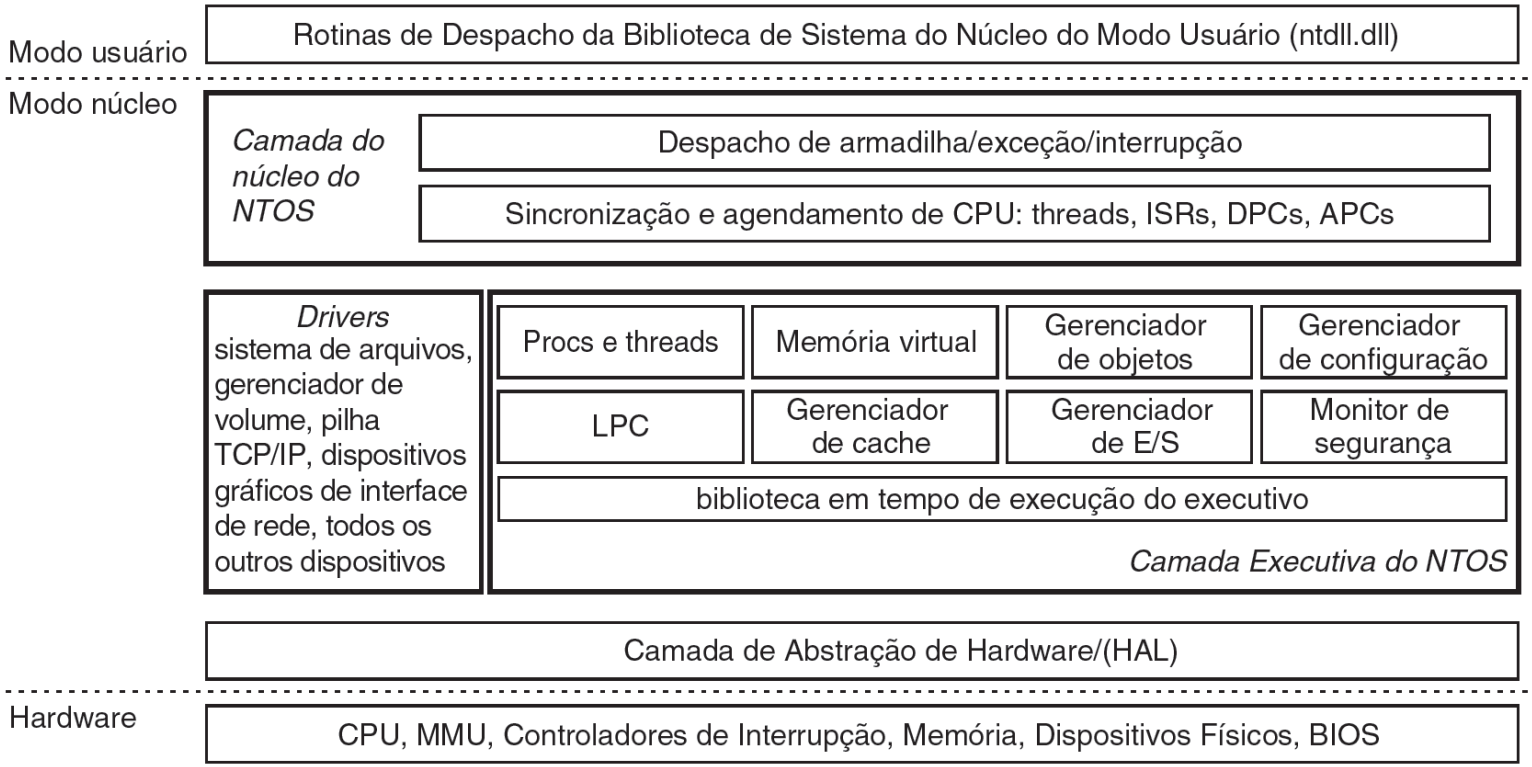


Figura 11.4 Organização do modo núcleo do Windows.

Microkernel

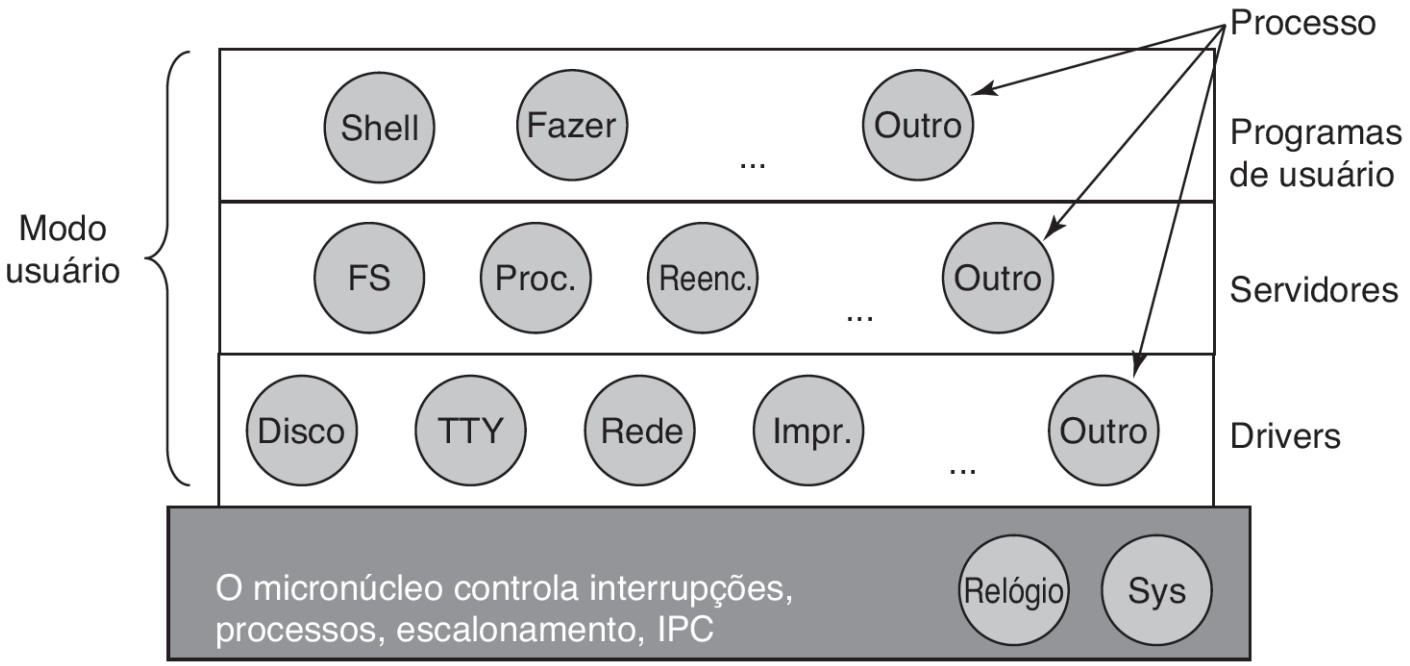


Figura 1.23 Estrutura do sistema MINIX 3.