

UNIVERSIDADE DE SÃO PAULO  
Escola de Engenharia de São Carlos  
Departamento de Engenharia Elétrica e de Computação

**SEL0337/SEL0630**  
**PROJETOS EM SISTEMAS EMBARCADOS**

**Capítulo 1**

**Sistemas Operacionais e Kernel**

Prof. Pedro de Oliveira C. Junior

[pedro.oliveiracjr@usp.br](mailto:pedro.oliveiracjr@usp.br)

# OBJETIVOS

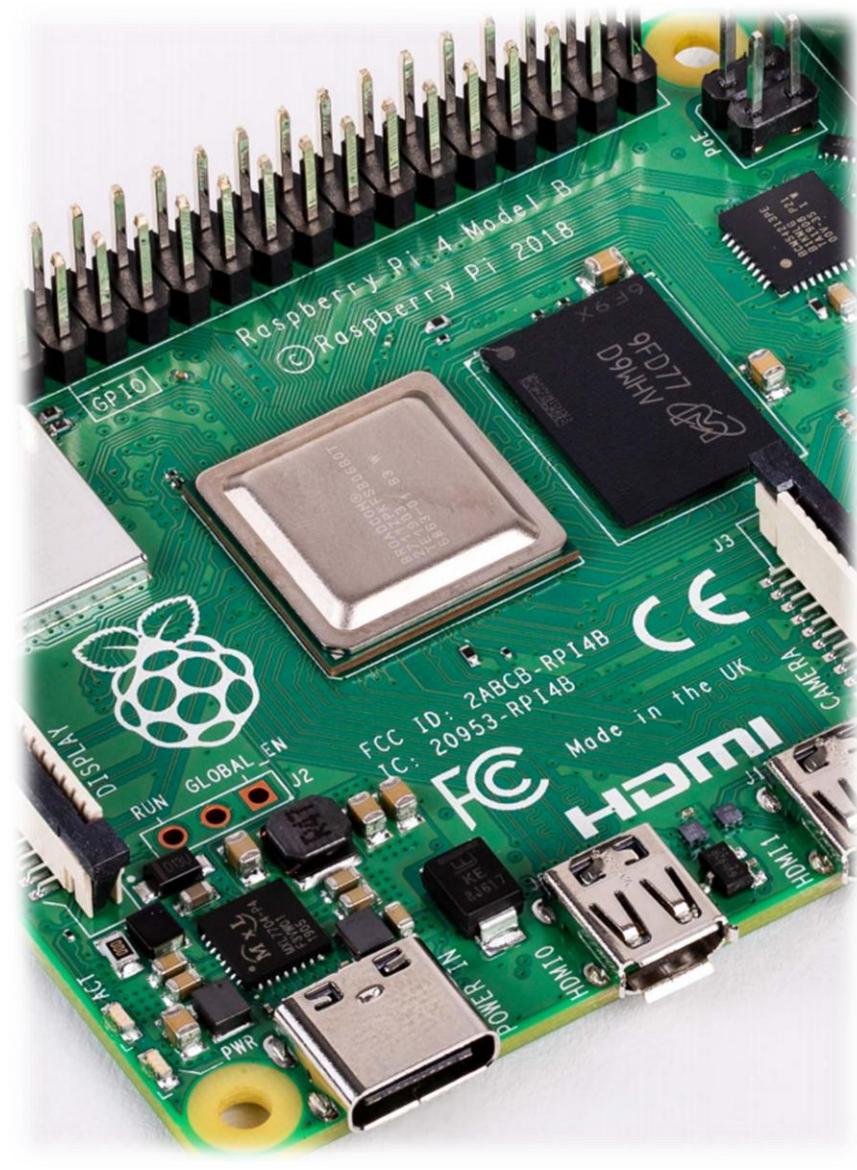


Revisão de Sistemas Operacionais e visão geral dos recursos do kernel;

Apresentar características fundamentais de Sistemas

Operacionais: tipos, aplicações, S.O. embarcado, módulos do kernel, sistemas de arquivos, e arquiteturas;

Introdução à alguns comandos (Unix) de verificação de recursos do S.O. usando Debian e Raspberry Pi O.S.

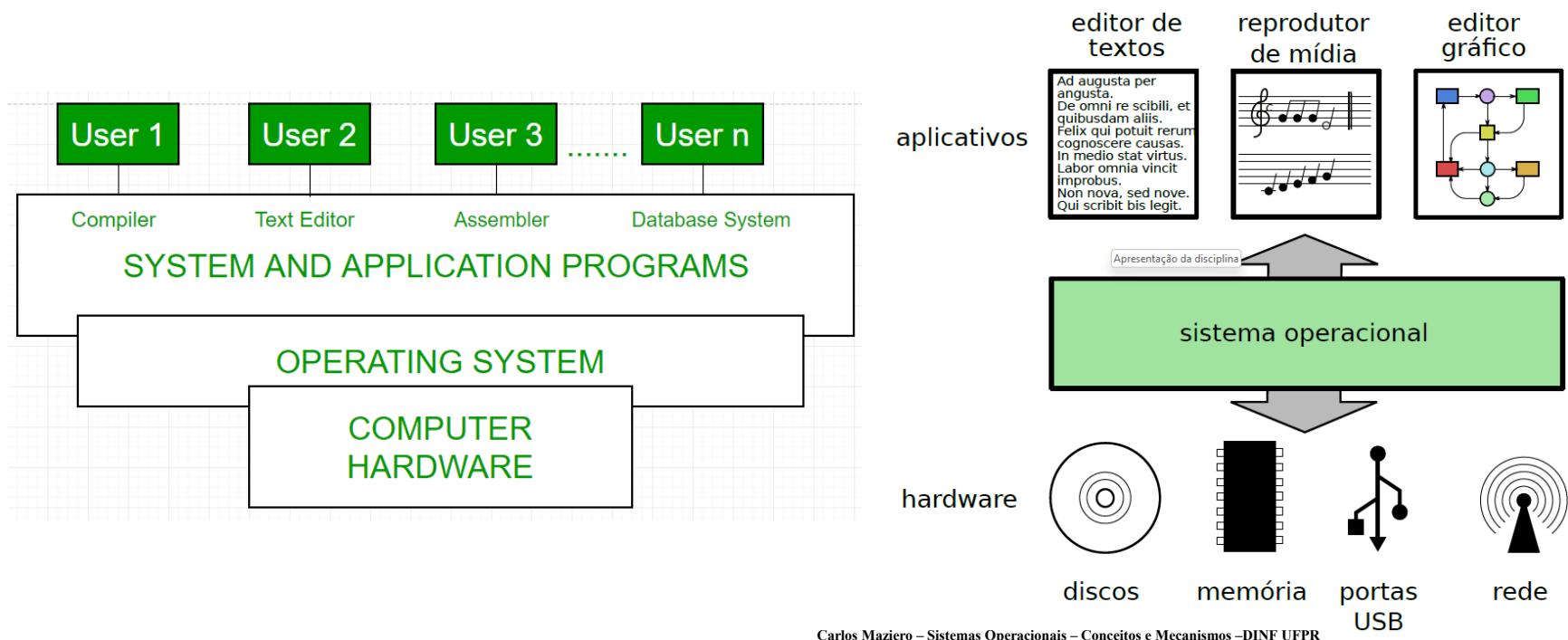


## A importância do Sistema Operacional

- S.O. (kernel)
- Detectar e preparar o hardware
- Gerenciar a memória
- Fornece as interfaces gráficas para o usuário
- Controlar os sistemas de arquivos
- Proporcionar o acesso e autenticação de usuário
- Oferecer os utilitários administrativos
- Virtualização
- Computação em tempo real

## O que é um Sistema Operacional?

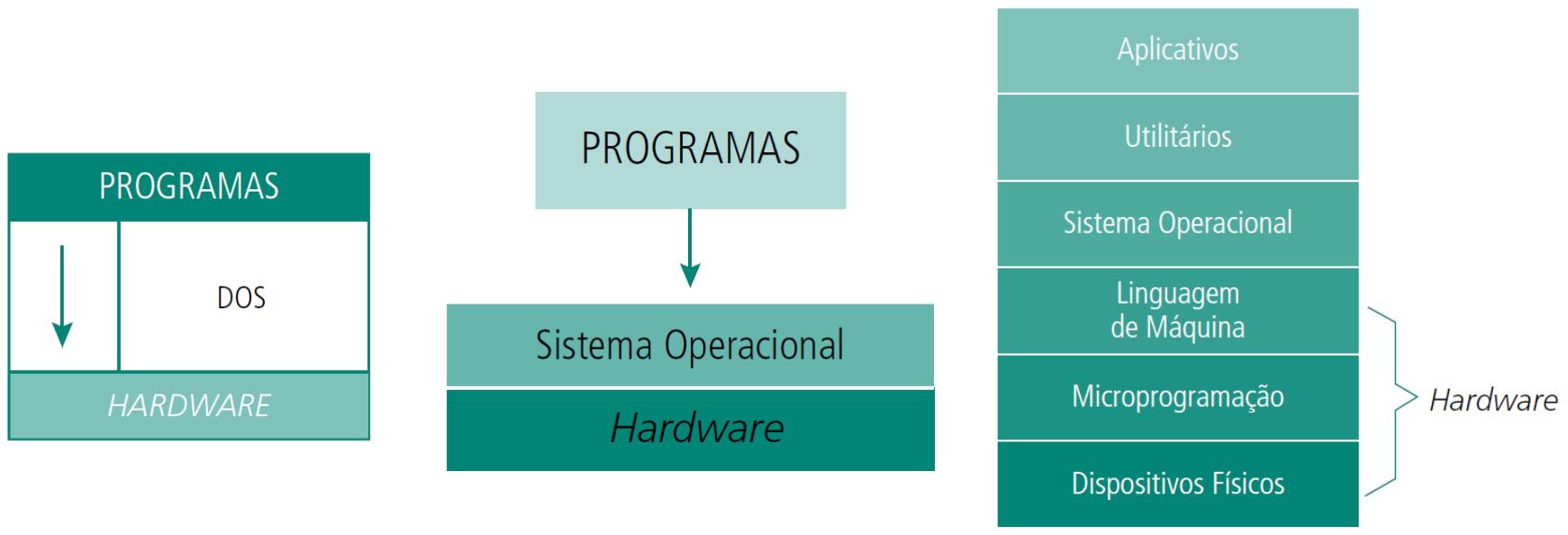
- ✓ Atua como intermediário entre usuário e hardware de um computador
- ✓ Fornece uma **interface de alto nível** para aplicações com facilidade de acesso!
- ✓ Gerencia recursos do sistema computacional (compartilhamento otimizado, processador, memória, organização de tarefas etc.) para maximização de desempenho de uso



## O que é um Sistema Operacional?

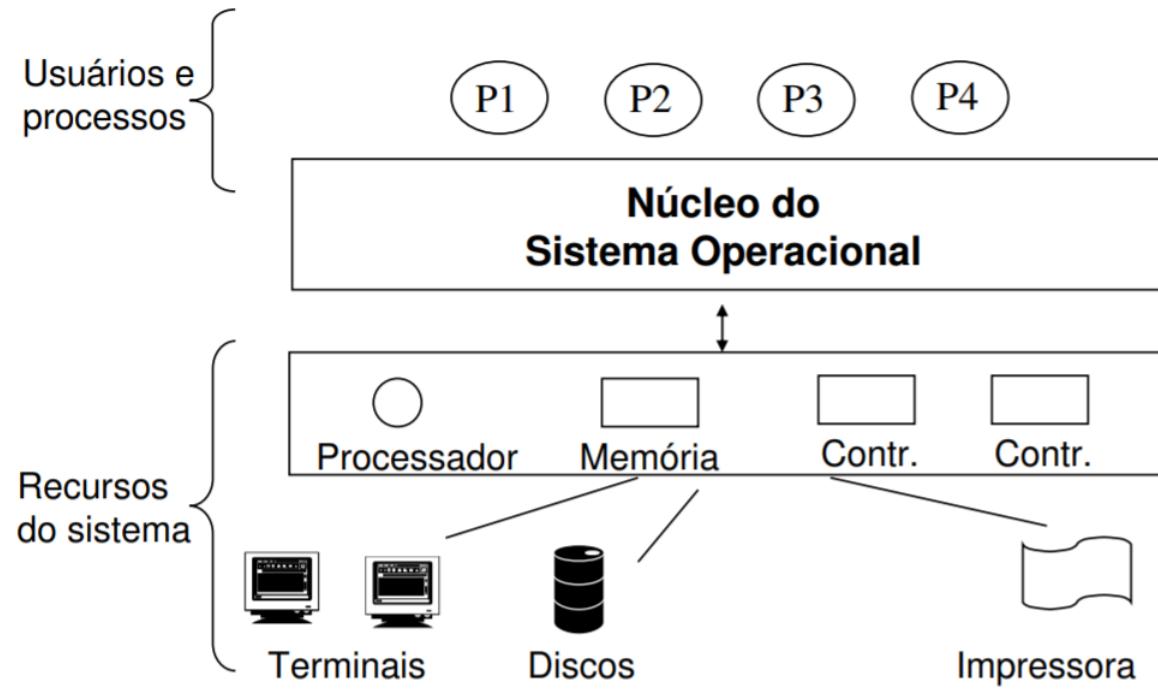
### ➤ Principais funcionalidades

- ✓ Execução de programas, operações de E/S, manipulação de sistema de arquivos, comunicação, detecção de erros, proteção.
- ✓ Interface com o hardware para programas desenvolvidos pelo usuário (**ao contrário do antigo sistema DOS que permitia acesso direto ao hardware sem tal interface!**)



## Estrutura geral de um Sistema Operacional

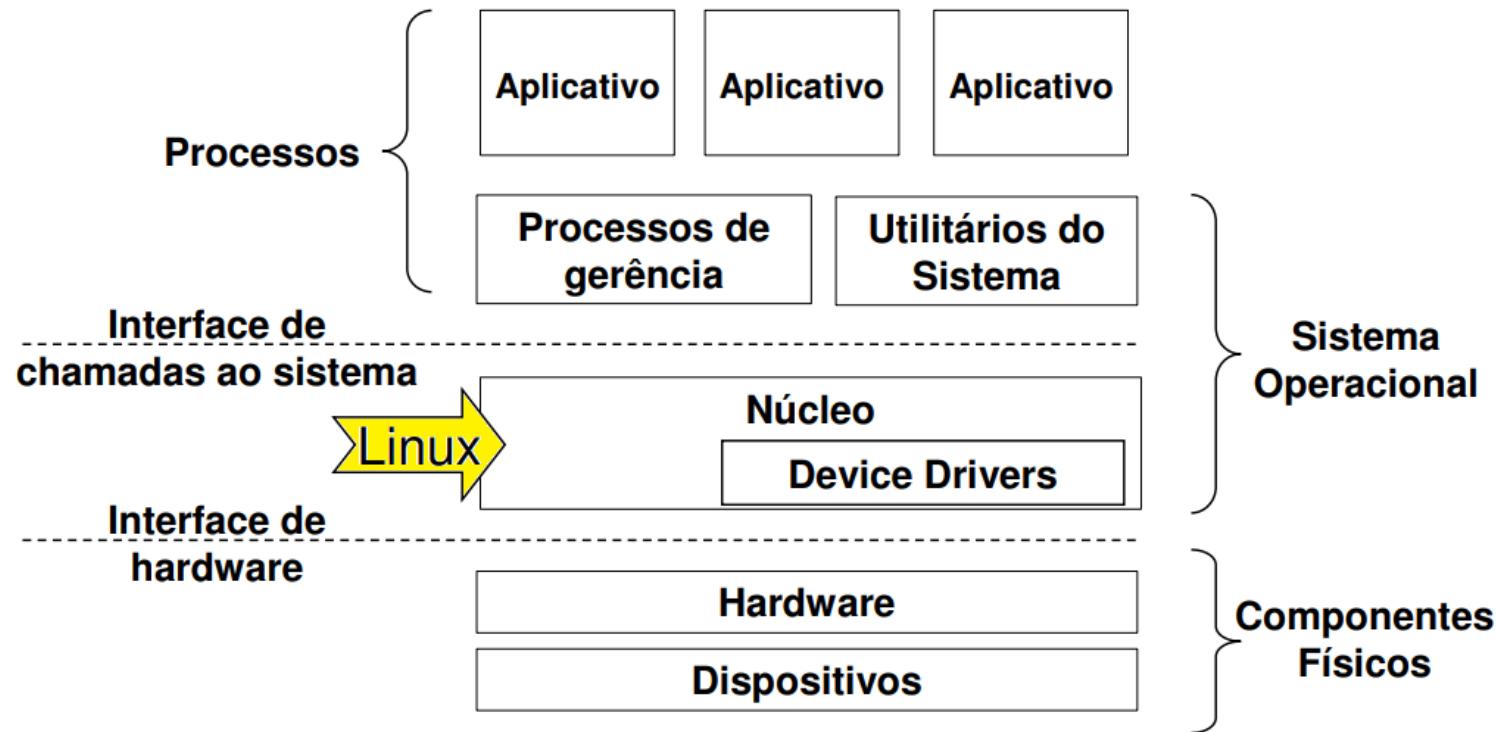
- Possui um kernel (núcleo) que permanece carregado em memória; Processos de gerência; Arquivos de configuração; Utilitários do sistema (programas básicos necessários para operação)



© 1997-2017 Volnys Bernal

## Linux

- Linux refere-se ao kernel (núcleo) do sistema operacional
- Multiusuário e Multitarefa; livre distribuição; Disponível para diversas arquiteturas: x86; MIPS; PowerPC; ARM; System Z (Mainframe IBM)

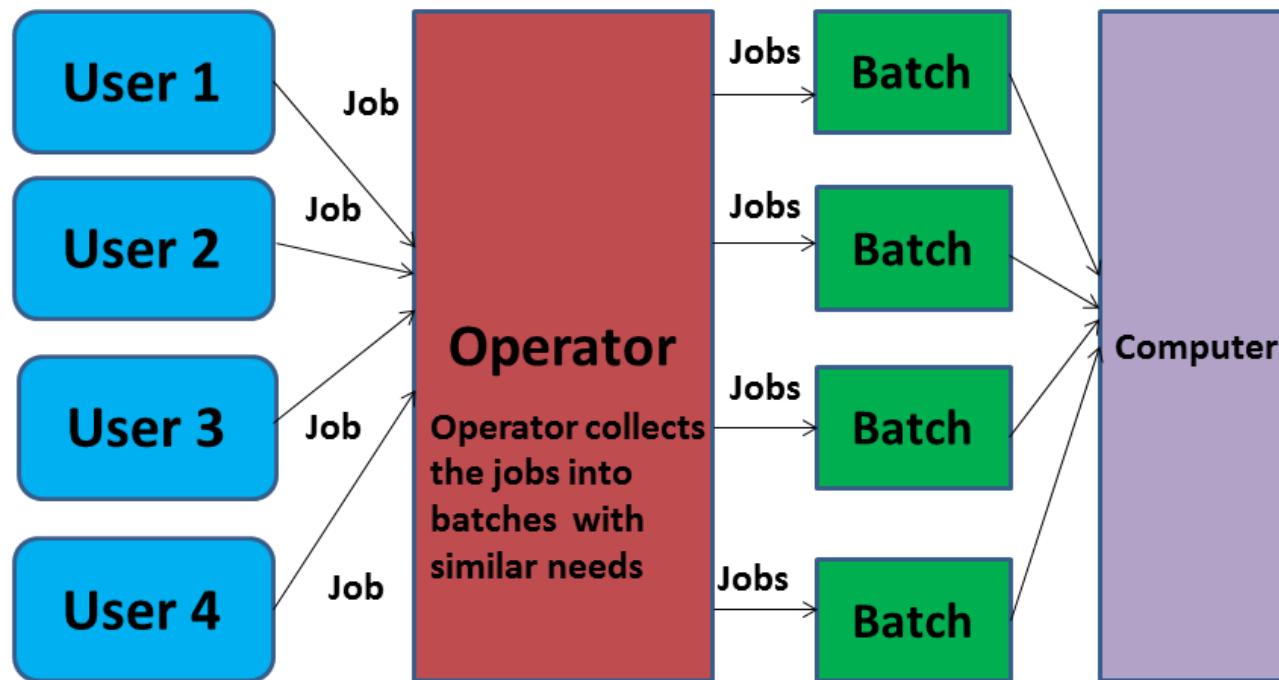


© 1997-2017 Volnys Bernal

## Classificação dos Sistemas Operacionais

### ➤ S.O. em lote (batch OS)

- ✓ Execução de programas em fila (um após o outro) – processamento em sequência

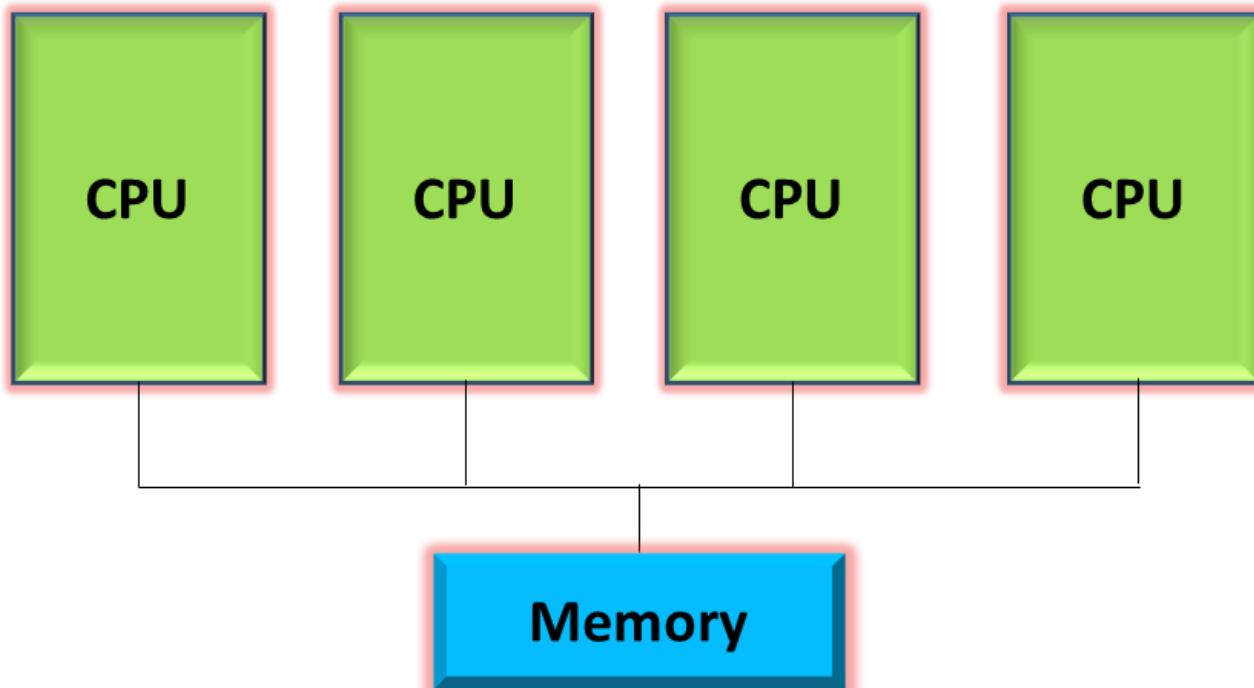


<https://www.guru99.com/operating-system-tutorial.html>

## Classificação dos Sistemas Operacionais

### ➤ S.O. de Múltiplos processadores (multiprocessor OS)

- ✓ Múltiplos processadores utilizados para aprimoramento da performance computacional e tarefas de processamento.

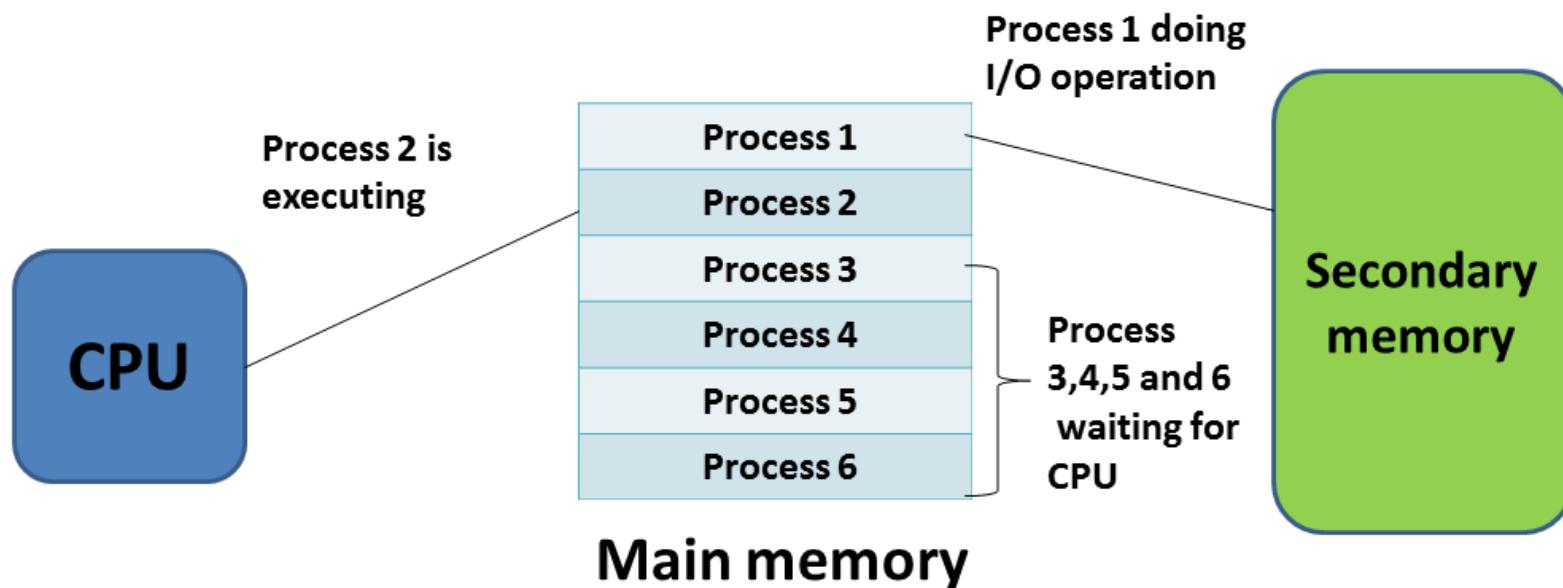


<https://www.naukri.com/learning/articles/types-of-operating-systems/>

## Classificação dos Sistemas Operacionais

### ➤ S.O. multiprogramação

- ✓ Múltiplos processos executados em um único processador

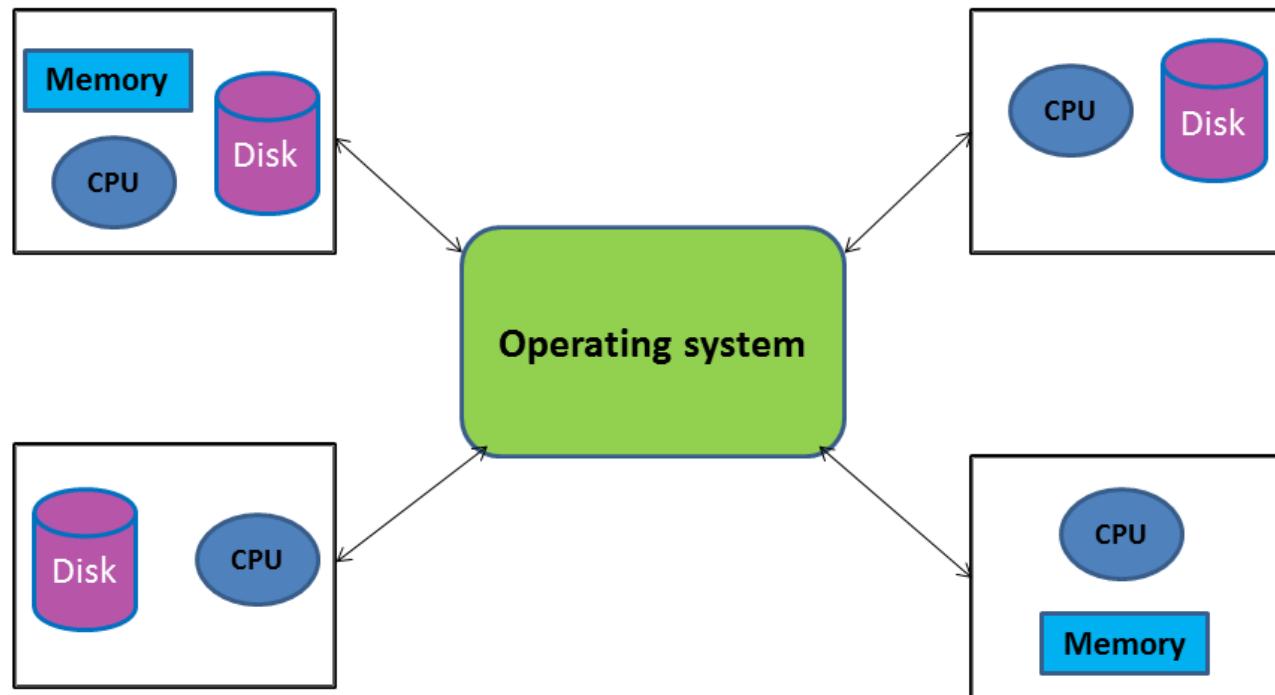


<https://www.naukri.com/learning/articles/types-of-operating-systems/>

## Classificação dos Sistemas Operacionais

### ➤ S.O. distribuído

- ✓ Único S.O. operando em uma rede de computadores

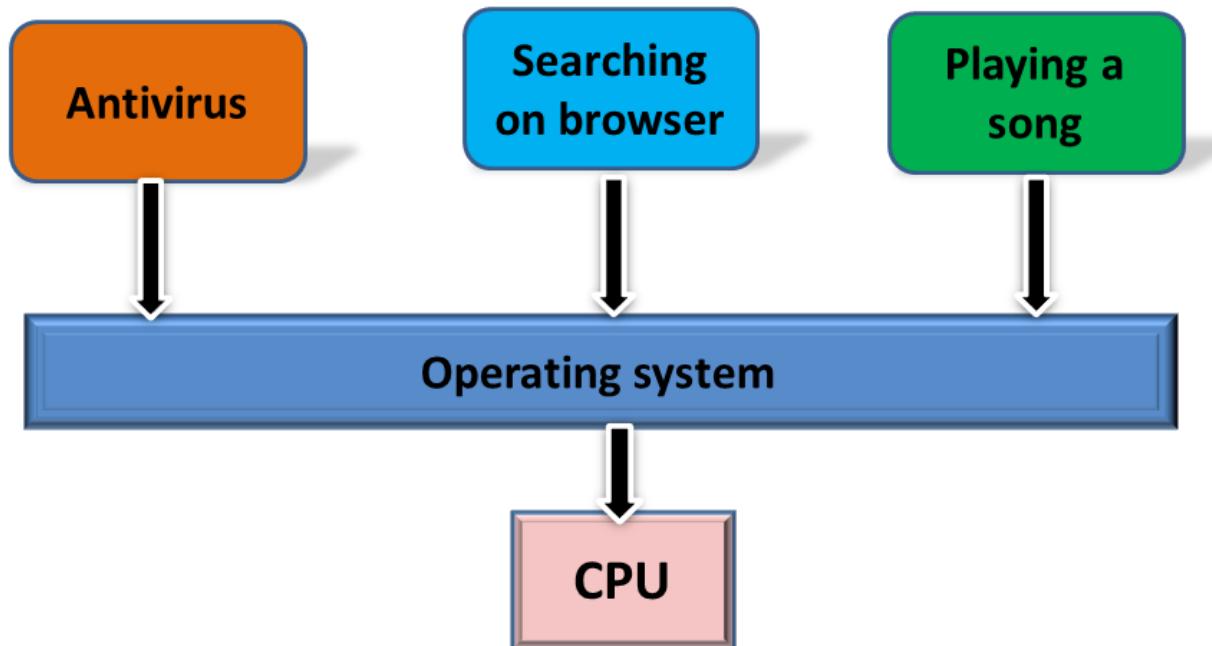


<https://www.naukri.com/learning/articles/types-of-operating-systems/>

## Classificação dos Sistemas Operacionais

### ➤ S.O. multitarefas

- ✓ Múltiplas aplicações em execução simultaneamente

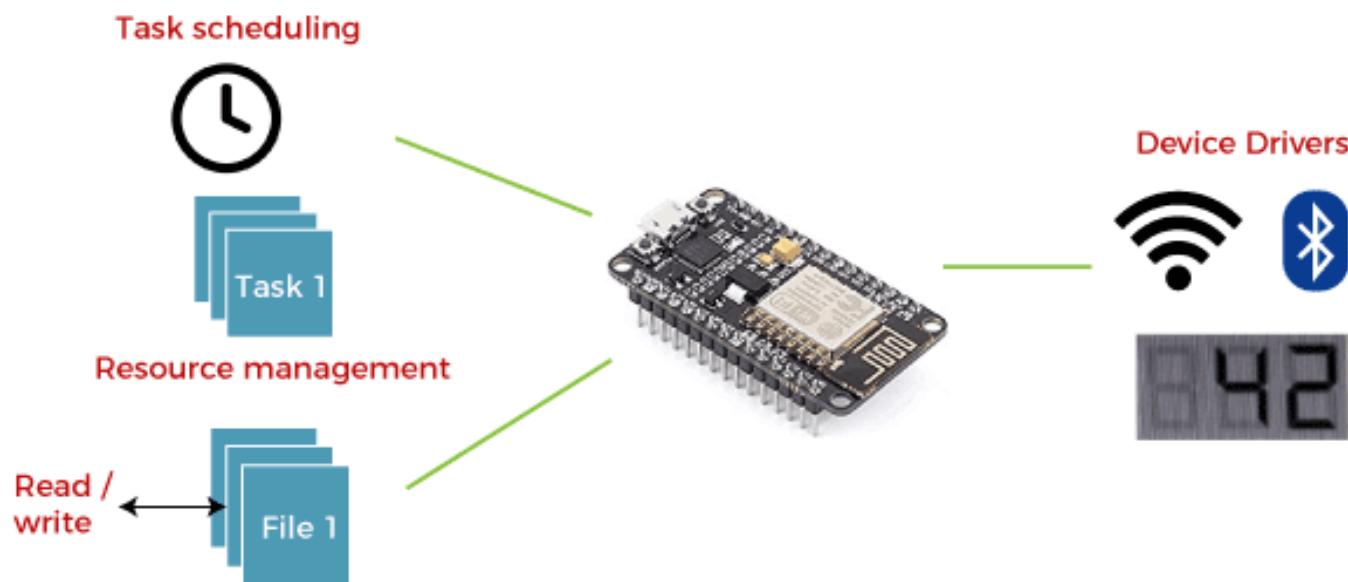


<https://www.naukri.com/learning/articles/types-of-operating-systems/>

## Classificação dos Sistemas Operacionais

### ➤ S.O. de tempo real (*real time operating systems*)

- ✓ Conforme abordado no capítulo 1, são sistemas que devem executar tarefas com deadlines, isto é, o parâmetro de tempo de resposta é fundamental e as tarefas podem ser de tempo crítico (**hard real-time**) ou de tempo real não crítico (**soft real-time**)



<https://www.javatpoint.com/types-of-operating-systems>

## Classes de aplicação

### ➤ Sistemas operacionais de computadores de propósito gerais

- ✓ Utilizados em computadores pessoais
- ✓ Fornece interface intuitiva para tarefas de propósito gerais: acesso a internet, aplicativos em geral, games, edição de texto, envio de e-mails etc.
- ✓ Exemplos: Windows 11 (Microsoft), Distribuições Linux (Gratuitas: Ubuntu, Debian, Mint, ArchLinux etc.), MacOS (Apple)



Windows 11



Linux

@debian



Ubuntu

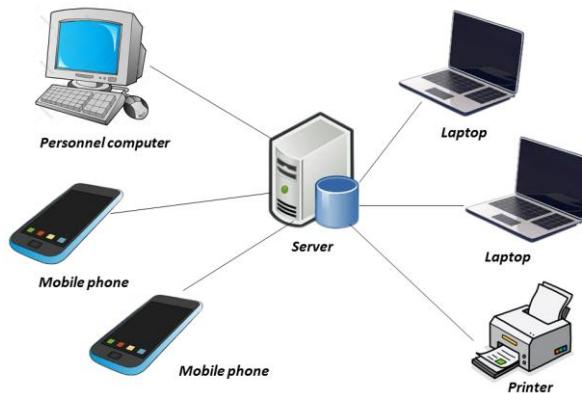


Mac OS

## Classes de aplicação

### ➤ Sistemas operacionais de servidores

- ✓ Desenvolvidos para computadores servidores com propósito de fornecer diversos tipos serviços para um grande número de usuários.
- ✓ Serviços: hospedagem de site, compartilhamento e backup de dados, serviços de e-mail, nuvem,
- ✓ Necessitam de hardware específico com maior processamento.
- ✓ Exemplos: Windows Server, Ubuntu Server, CentOS, Fedora, IBM.



## Classes de aplicação

### ➤ Sistemas operacionais portáteis (mobile systems)

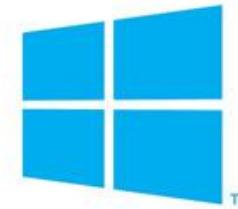
- ✓ Desenvolvidos para smartphones, Tablets e outros dispositivos móveis.
- ✓ Funcionalidades: interface adaptada para computadores portáteis, sem necessidade de periféricos (teclado, mouse etc.), com recurso touch.
- ✓ Exemplos: iOS (Apple – iPhones); Android (Google – vários modelos de smartphones); FirefoxOS e Windows Phone (obsoletos)



ANDROID



iOS



Windows Phone

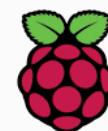


Firefox OS

## Classes de aplicação

### ➤ Sistemas operacionais embarcados

- ✓ Desenvolvidos para sistemas embarcados, isto é, para propósitos computacionais específicos, com objetivos simples, sem grandes alterações.
- ✓ Geralmente instalados de fábrica sem permitir alterações, com restrições de memória, consumo de energia, tamanho etc.
- ✓ Exemplo: Linux Embocado, QNX, VXworks, FreeRTOS
- ✓ OBS. As definições atuais consideram que smartphones modernos são considerados sistemas computacionais embarcados altamente sofisticados e usados em tarefas de propósito geral, portanto, são enquadrados na classe de “dispositivos portáteis” do slide anterior.



Raspberry Pi OS



## Sistema operacional embarcado

- A necessidade do sistema operacional em sistemas embarcados
- Sistemas avançados como set-top boxes, que podem ser reproduzidos em 1000 canais, podem gravar vídeos de alta definição e suportar mídia de dispositivos externos como USB, definitivamente requerem S.O. e sistema embarcado sofisticado (microprocessador, CPU multicore, GPU etc).



Linux

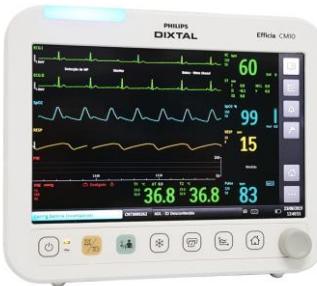


Windows IoT

## Sistema operacional embarcado

➤ Classes de produtos que incorporam sistemas embarcados com sistema operacional de propósito geral (Linux, por exemplo):

- ✓ Set-top-boxes - receptores de TV e serviços de streaming Roteadores e switches de rede
- ✓ Servidores de armazenamento em rede (NAS) e domésticos
- ✓ Centrais de multimídia em veículo
- ✓ Monitores médicos de monitoramento de saúde

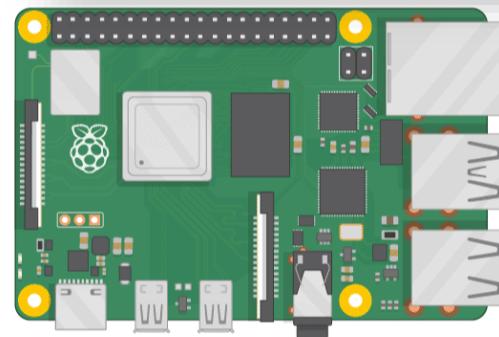
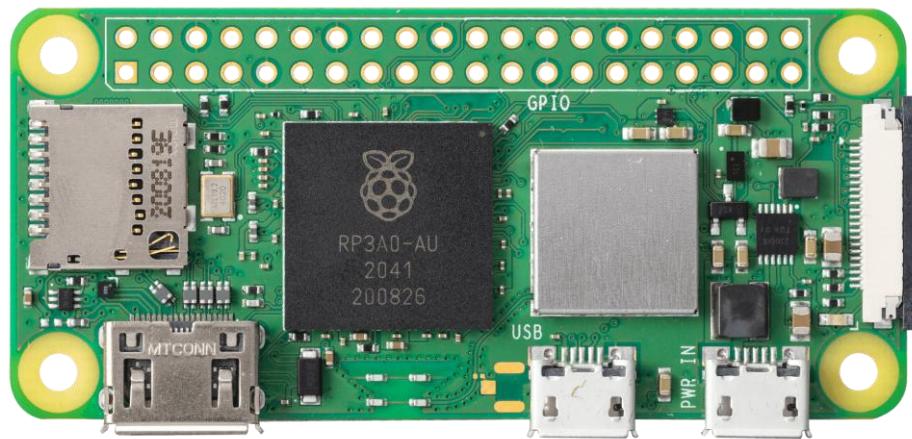
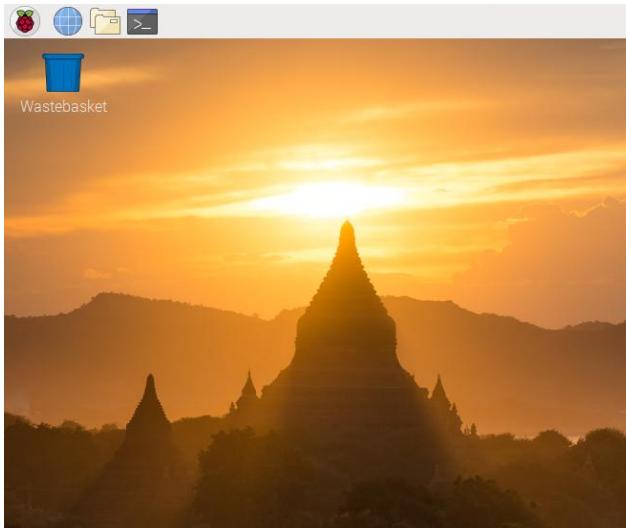


## Sistema operacional embarcado

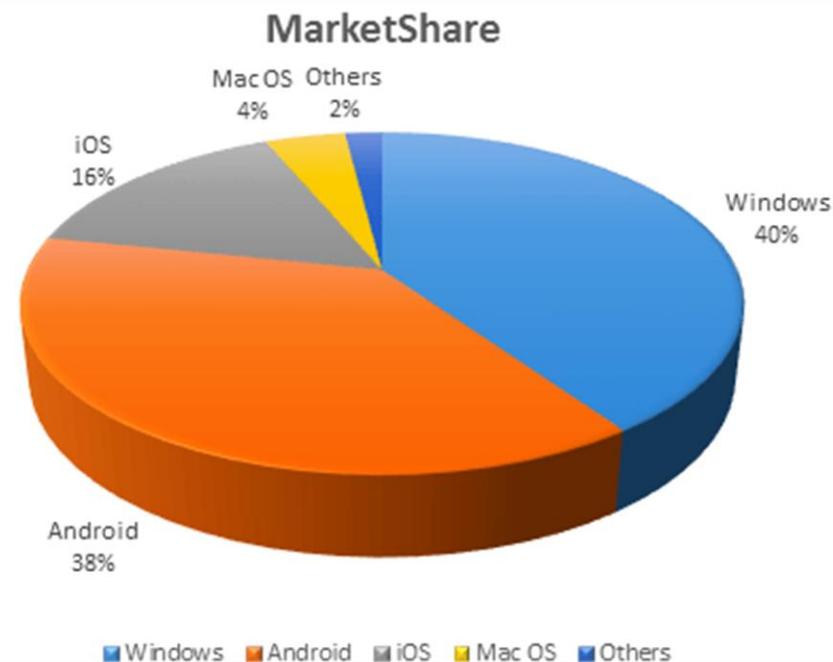
- ✓ Drones
- ✓ Sistemas de videovigilância
- ✓ Robótica
- ✓ Cluster de computação e estações de trabalho de baixo consumo
- ✓ Video games



# Sistema operacional embarcado - Raspberry Pi



## Mercado de Sistemas Operacionais

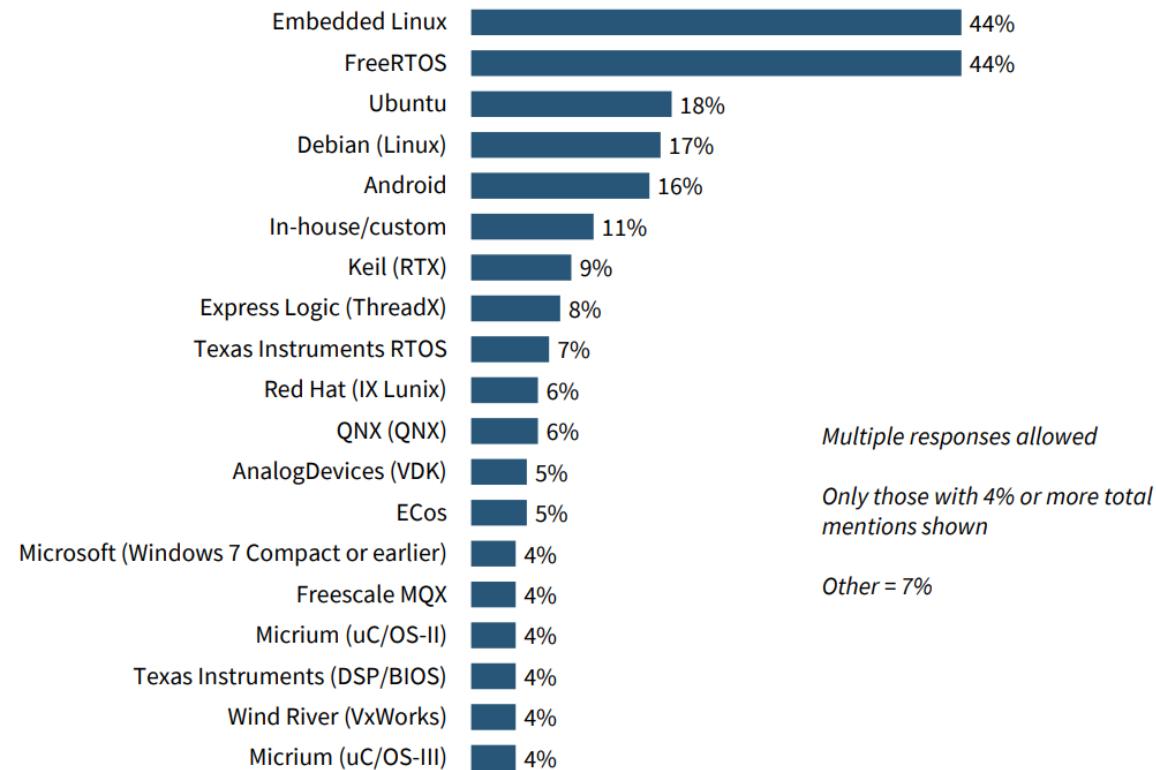


<https://www.guru99.com/operating-system-tutorial.html>

## Mercado de Sistemas Operacionais Embarcados

Most popular embedded OSs – Embedded Linux, FreeRTOS and Ubuntu

Top 3 OSs are especially popular in APAC, while Embedded Linux is used more in the Americas



Embedded Survey- "The current state of embedded development"- 2023

<https://www.embedded.com/wp-content/uploads/2023/05/Embedded-Market-Study-For-Webinar-Recording-April-2023.pdf>

## Alguns Sistemas Operacionais populares e suas características

### Origem e licença:

- ✓ O **Windows** é desenvolvido pela Microsoft e é um sistema proprietário, ou seja, requer o pagamento de uma licença para uso.
- ✓ O **Linux** é um kernel (núcleo) de código aberto, que pode ser modificado e distribuído livremente por qualquer pessoa, incorporado em uma série de sistemas operacionais (**Ubuntu, Debian, CentOS, SUSE, Fedora** etc.).
- ✓ O **Android** é baseado no Linux e também é de código aberto, mas possui algumas restrições impostas pelo Google.
- ✓ O **MACOS** é desenvolvido pela Apple e é um sistema exclusivo para os dispositivos da marca.

### Interface e usabilidade:

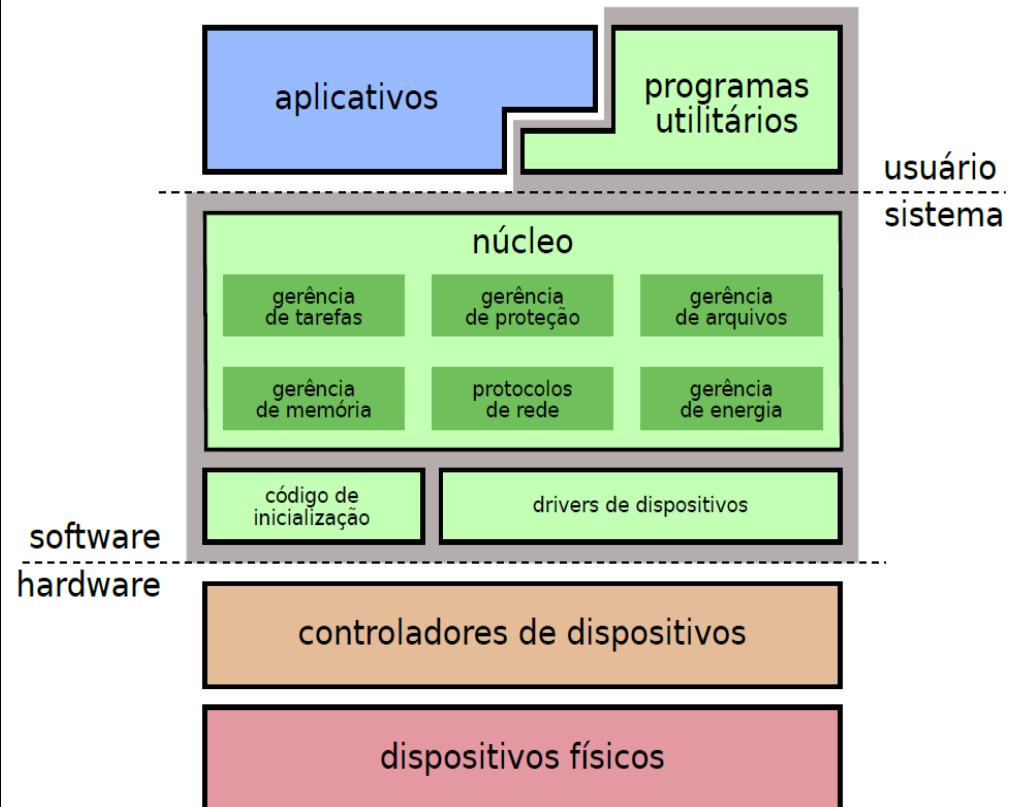
- ✓ O **Windows** é uma opção popular para usuários com pouco conhecimento técnico, pois possui uma interface gráfica intuitiva e compatibilidade com a maioria dos programas e dispositivos.
- ✓ O **Linux** possui várias interfaces gráficas diferentes, que podem ser escolhidas pelo usuário de acordo com sua preferência.
- ✓ O **Android** é um sistema voltado para dispositivos móveis, que possui uma interface baseada em toque e acesso a uma grande variedade de aplicativos.
- ✓ O **MACOS** é frequentemente utilizado por profissionais criativos, pois possui uma interface elegante e funcional, além de integrar-se facilmente com outros produtos da Apple.

### Segurança e estabilidade:

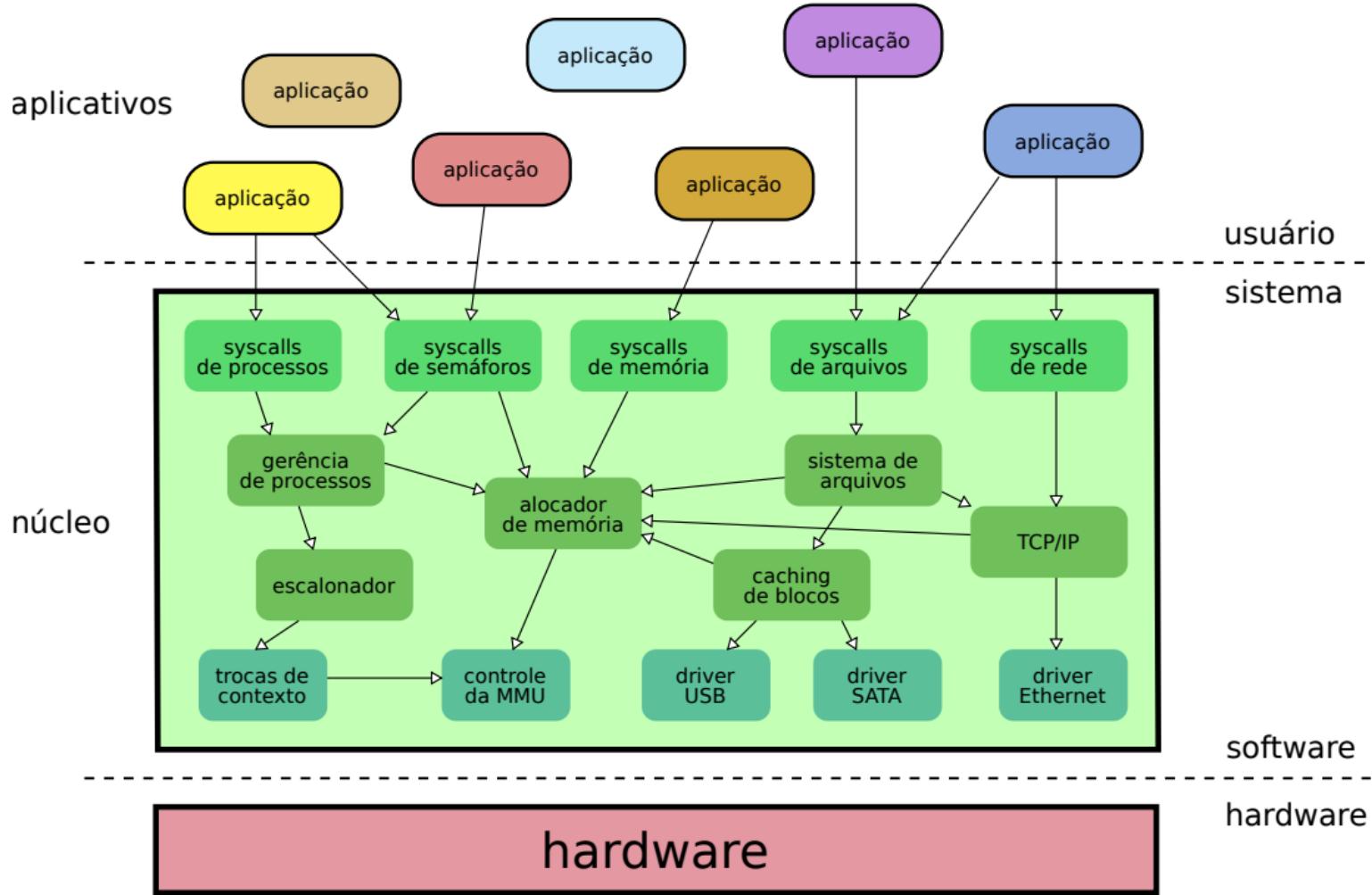
- ✓ O **Windows** é o sistema mais vulnerável a vírus e malwares, pois é o mais utilizado no mundo e também o mais visado pelos hackers.
- ✓ O **Linux** é considerado o sistema mais seguro e estável, pois possui um controle rigoroso das permissões de acesso e uma comunidade ativa que corrige rapidamente as falhas.
- ✓ O **Android** também possui um bom nível de segurança, mas depende da atualização dos fabricantes dos dispositivos para receber as correções. O **MACOS** é menos suscetível a ataques do que o Windows.

## Arquiteturas do Sistema Operacional

- ✓ **Núcleo (kernel)** – gerencia recursos do hardware e implementa abstrações usados pelos aplicativos e programas utilitários;
- ✓ **Boot Code:** iniciação do hardware com testes funcionais e carregamento do kernel do S.O.
- ✓ **Drivers:** módulos específicos para acessar dispositivos de diversos fabricantes (USB, SATA, periféricos);
- ✓ **Programas utilitários:** implementam funcionalidades do S.O., manipulação de arquivos, interpretador, interface gráfica, configuração de mídias.



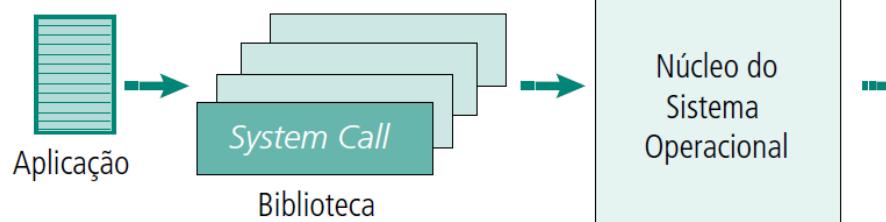
## Estrutura do Kernel



## Funcionamento do Kernel

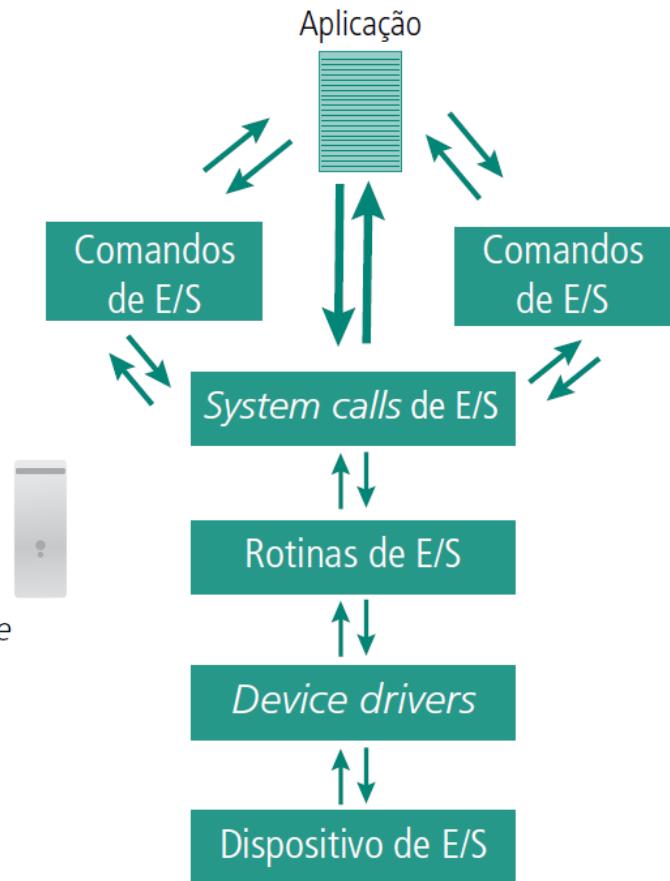
### ➤ Chamadas do sistema (system calls)

- ✓ Controle de processo;
- ✓ Manipulação de arquivos e dispositivos
- ✓ Manutenção de informações
- ✓ Comunicações



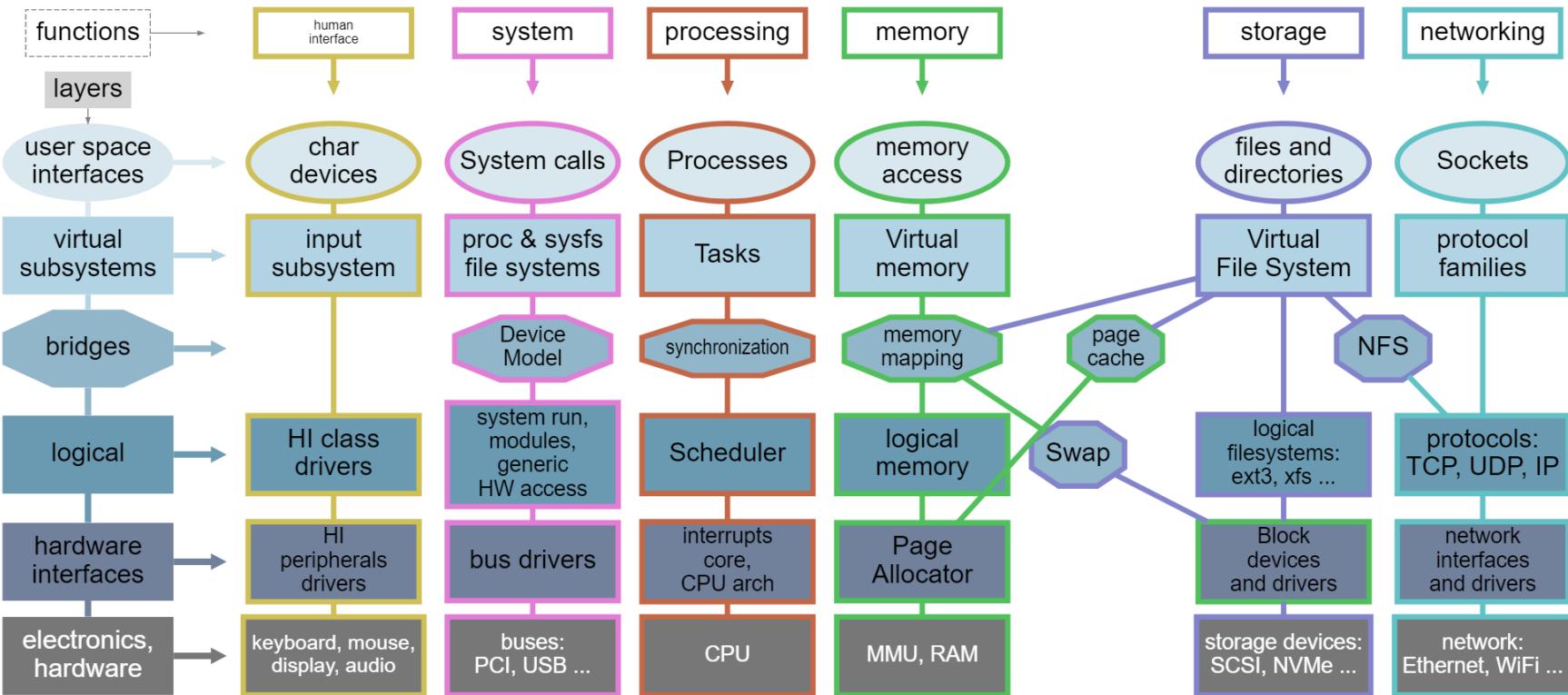
#### Exemplos de System Call

- **`fork()`** - Cria um novo processo
- **`open()`** - Abre um arquivo
- **`write()`** - Escreve em um arquivo
- **`mkdir()`** - Cria uma pasta no FS
- **`read()`** - Lê de um arquivo



## Kernel Linux

Linux kernel diagram

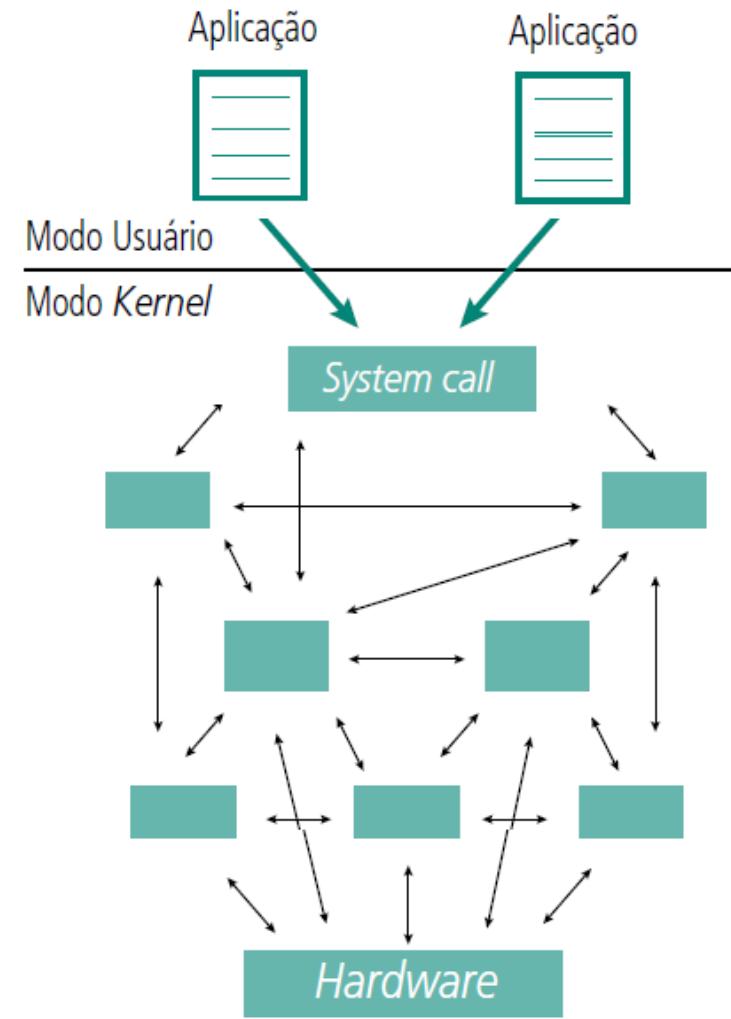
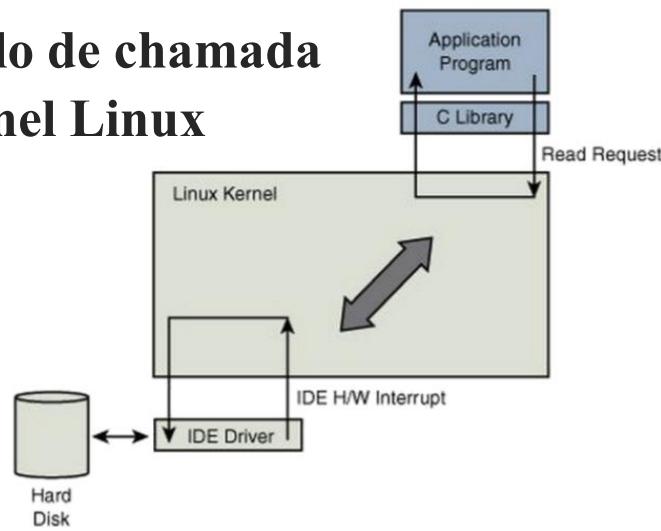


© 2007-2022 Costa Shulyupin <http://www.MakeLinux.net/kernel/diagram>

## Kernel Monolítico

- ✓ Composto por um conjunto de rotinas que podem interagir livremente umas com as outras;
- ✓ Vários procedimentos que são compilados separadamente e depois linkados, formando um grande e único programa executável

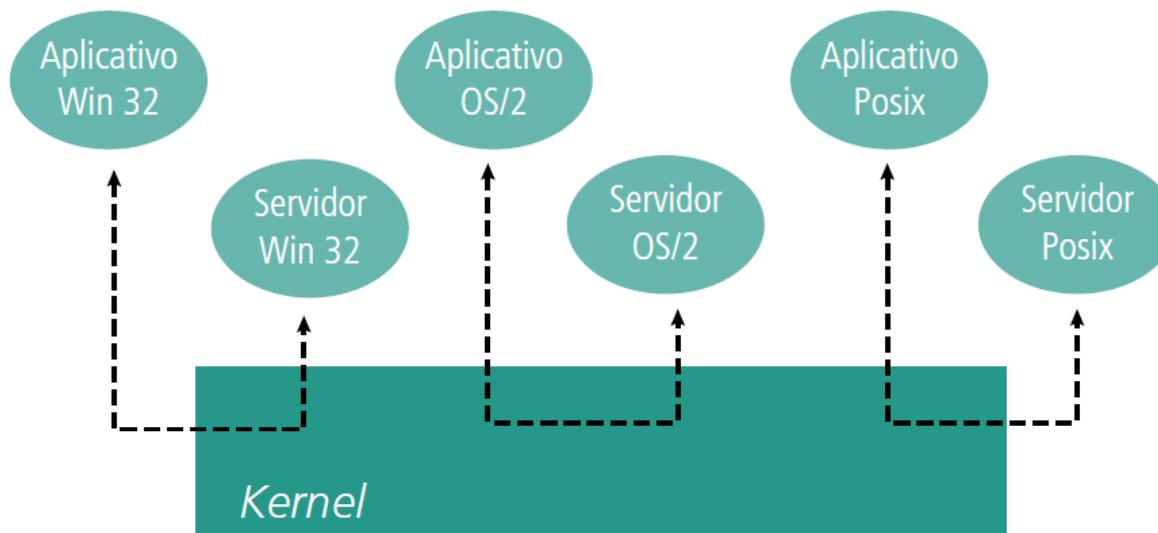
### Exemplo de chamada No kernel Linux



Bruno C. C. Sistemas Operacionais, IFES – 2010 E-TecBrasil

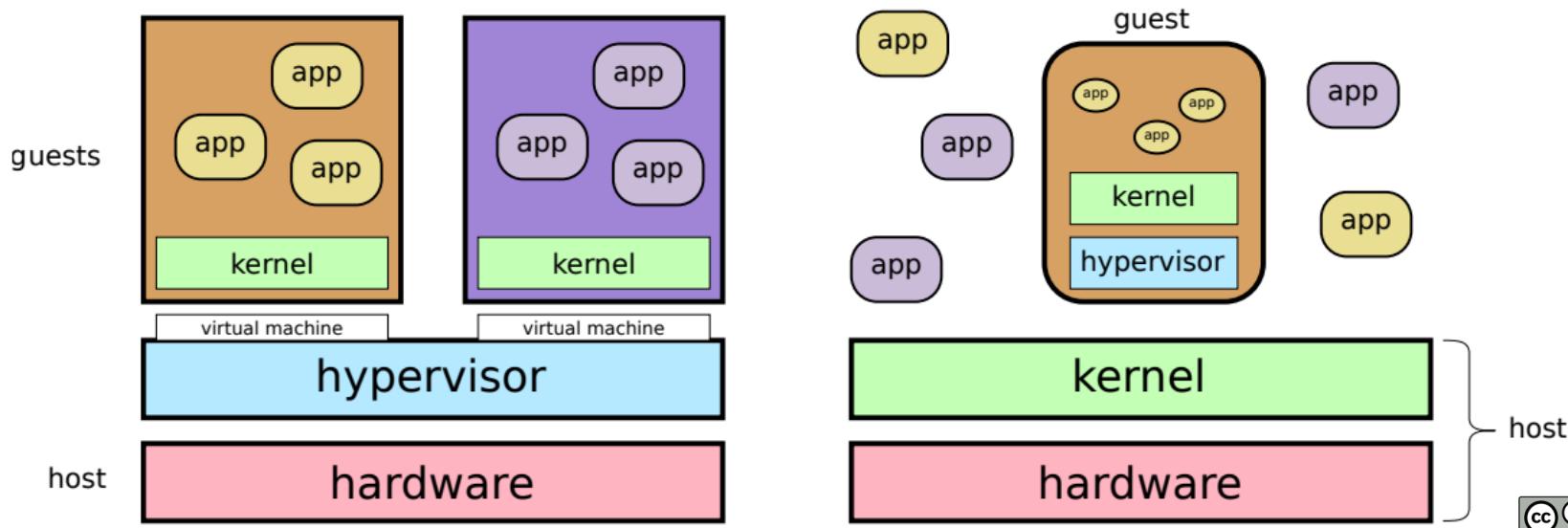
## MicroKernel

- ✓ Segue a tendência de um S.O. menor e o mais simples possível.
- ✓ Fornecem gerência mínima de memória e processos
- ✓ Divide-se em processos, sendo cada um responsável por oferecer um conjunto de serviços



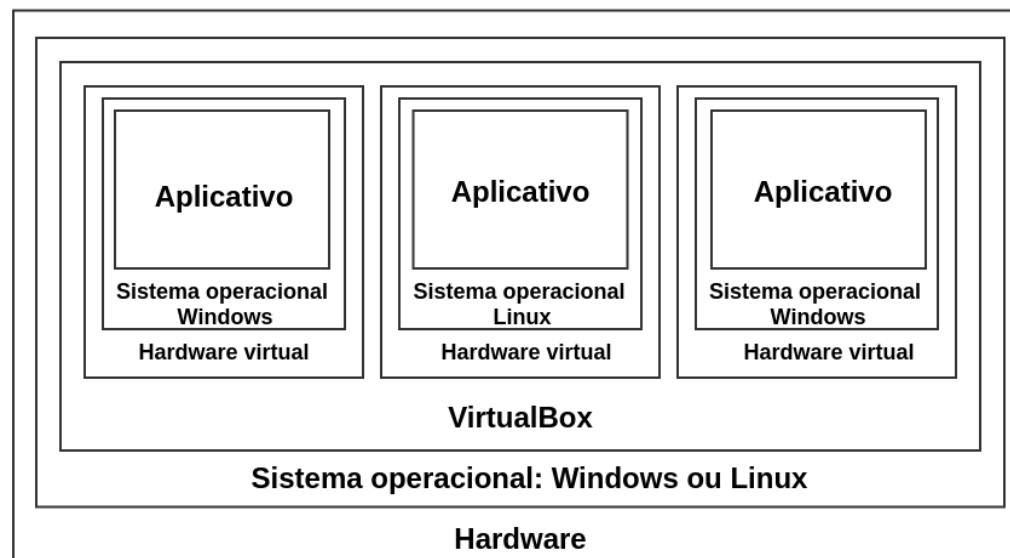
## Máquina virtual

- ✓ Uma máquina virtual é uma camada de software que “transforma” um sistema em outro, ou seja, que usa os serviços fornecidos por um sistema operacional (ou pelo hardware) para construir a interface de outro sistema.
- ✓ Por exemplo, o ambiente **JVM -Java Virtual Machine** (hypervisor) usa os serviços de um sistema operacional convencional (Windows ou Linux) para construir um computador virtual que processa bytecode, o código binário dos programas Java



## Máquina Virtual

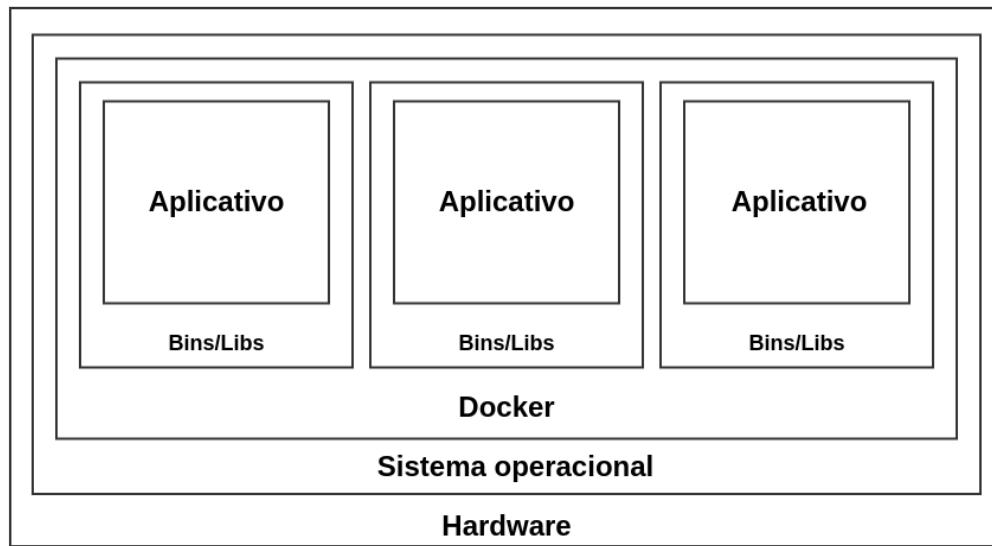
- ✓ A virtualização é um processo de abstração dos recursos da computação. Uma **máquina virtual** é uma máquina com seu próprio processador, memória, armazenamento e interface de rede, tudo criado de modo virtual. Este sistema virtual é criado com base em um sistema de hardware físico. Na imagem, temos um hardware físico (ex.: PC) com um S.O. (Windows ou Linux). O software **Virtual Box** (hypervisor) permite criar hardwares virtuais com seu próprio sistema operacional e aplicações instaladas nele.



<https://irias.com.br/blog/docker-virtualizacao-container/>

## Containers e Docker

Um container é um processo isolado do restante do sistema, que compartilha o mesmo kernel do sistema operacional da máquina. Todo o conjunto de arquivos necessários ao seu funcionamento é fornecido a partir de uma imagem, o que garante portabilidade e consistência. Cada container traz consigo bibliotecas e dependências próprias, mas sem a necessidade de simular hardware ou instalar um novo sistema operacional, como é feito na virtualização, já que aproveita o kernel existente.



O Docker popularizou o uso de containers ao fornecer uma plataforma prática para criação, empacotamento e distribuição de aplicações portáveis. Com ele, é possível iniciar e gerenciar containers rapidamente:

```
docker run hello-world # executa um container de teste  
docker ps           # lista containers em execução  
docker images
```

## Kernel Linux - módulos

- ✓ Os **módulos do kernel** são pedaços de código que podem ser carregados ou descarregados do kernel em tempo de execução. Eles fornecem funcionalidades específicas, como **drivers de dispositivo, sistemas de arquivos ou suporte para protocolos de rede**.

### Vantagens da Modularidade

- ✓ **Redução do Tamanho do Kernel:** A modularidade permite que apenas os módulos necessários sejam carregados. Isso reduz o tamanho do kernel em execução na memória.
- ✓ **Facilidade de Manutenção:** Atualizar ou corrigir um módulo específico é mais fácil do que recompilar o kernel inteiro. Isso agiliza a manutenção do sistema.
- ✓ **Suporte a Hardware:** Módulos de driver de dispositivo podem ser carregados sob demanda, permitindo que o kernel suporte uma ampla variedade de hardware.

## Kernel Linux - recursos

- ✓ O código-fonte do kernel Linux é mantido em um sistema de controle de versão e está hospedado no site oficial do kernel Linux ([www.kernel.org](http://www.kernel.org)).
- ✓ O repositório Git contém todos os arquivos fonte do kernel, incluindo código para várias arquiteturas, drivers, subsistemas e funcionalidades.

### Principais funcionalidades e recursos do Kernel Linux:

- ✓ Gerenciamento de processos do S.O. e a comunicação entre eles
- ✓ Gerenciamento de memória
- ✓ Sistema de arquivos
- ✓ Drivers de dispositivos
- ✓ Gerenciamento de interrupções
- ✓ Gestão de energia
- ✓ Segurança
- ✓ Rede
- ✓ Controle de E/S
- ✓ Gerenciamento de tarefas e de tempo
- ✓ Virtualização
- ✓ Suporte para Arquitetura específica (ARM, x86 etc.)

### The Linux Kernel Archives

[About](#)   [Contact us](#)   [FAQ](#)   [Releases](#)   [Signatures](#)   [Site news](#)



Protocol   Location  
HTTP   <https://www.kernel.org/pub/>  
GIT   <https://git.kernel.org/>  
RSYNC   <rsync://rsync.kernel.org/pub/>

Latest Release  
**6.4.10** 

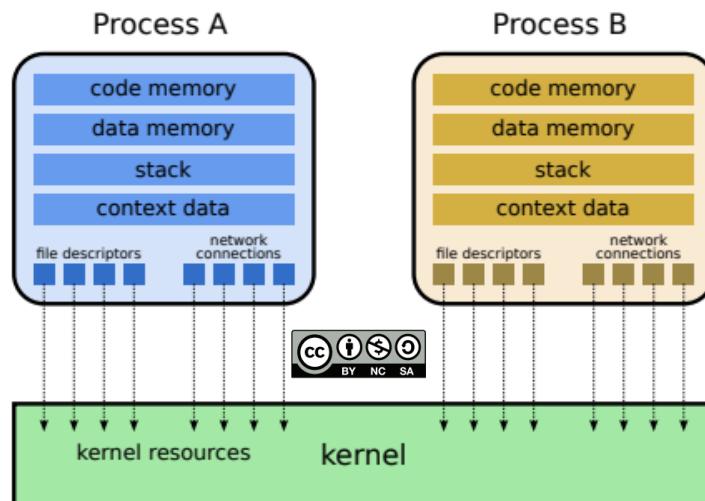
## Kernel Linux - módulos

- ✓ Exemplo de módulos do kernel Linux carregados no S. O.: digitar comando “`lsmod`” no terminal Linux; “`uname -r`” (versão do kernel); “`dmesg`” (mensagens do kernel relacionadas ao sistema)

```
sel@raspberrypi:~ $ lsmod
Module           Size  Used by
rfcomm          49152  4
cmac            16384  3
algif_hash      16384  1
aes_arm_bs     24576  2
crypto_simd    16384  1 aes_arm_bs
cryptd          24576  2 crypto_simd
algif_skcipher 16384  1
af_alg          28672  6 algif_hash,algif_skcipher
bnep            20480  2
hci_uart        40960  1
btbcm           20480  1 hci_uart
bluetooth       409600 31 hci_uart,bnep,btbcm,rfcomm
ecdh_generic    16384  2 bluetooth
ecc              40960  1 ecdh_generic
```

## Processos e threads

- ✓ Um processo é um contêiner de recursos utilizados por uma ou mais tarefas para sua execução: **áreas de memória (código, dados, pilha), informações de contexto e descritores de recursos do núcleo (arquivos abertos, conexões de rede, etc).**
- ✓ Um processo pode então conter várias tarefas, que compartilham esses recursos.
- ✓ Os processos são isolados entre si pelos mecanismos de proteção providos pelo hardware (isolamento de áreas de memória, níveis de operação e chamadas de sistema), impedindo que uma tarefa do processo PA acesse um recurso atribuído ao processo PB.

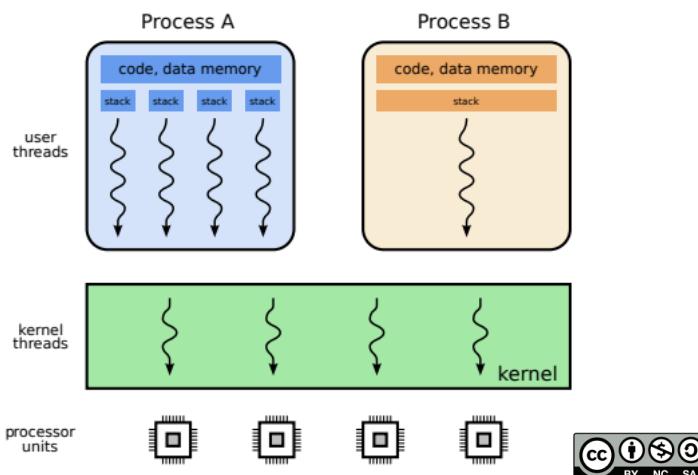


### Exemplos de processos:

- ✓ Navegador Web
- ✓ Processamento de imagem e vídeo
- ✓ Reprodutor de mídia
- ✓ Gerenciador de arquivos
- ✓ Processos de inicialização do S.O.
- ✓ Servidor Web
- ✓ Comunicação com dispositivos externos

## Processos e threads

- ✓ Uma **thread** é definida como sendo um fluxo de execução independente.
- ✓ Um processo pode conter uma ou mais threads, cada uma executando seu próprio código e compartilhando recursos com as demais **threads** localizadas no mesmo processo.
- ✓ Cada thread é caracterizada por um código em execução e um pequeno contexto local, o chamado **Thread Local Storage (TLS)**, composto pelos registradores do processador e uma área de pilha em memória, para que a **thread** possa armazenar variáveis locais e efetuar chamadas de funções.



### Exemplos de uso threads em processos:

- ✓ Tarefas em background
- ✓ Processamento paralelo de jogos
- ✓ Fila de impressão de documentos
- ✓ Processamento em lotes: edição de imagens, conversão de vídeos, áudio em tempo real

## Processos e threads

**Exemplo de processos e tarefas no sistema operacional:** Windows (“**tasklist**” – prompt de comando; gerenciador de tarefas); Linux (“**ps aux**” – shell)

Nome da imagem	Identifi	Nome da sessão	Sessão#	Uso de memór
System Idle Process	0	Services	0	8 K
System	4	Services	0	6.636 K
Secure System	140	Services	0	42.016 K
Registry	212	Services	0	22.596 K
smss.exe	720	Services	0	1.088 K
csrss.exe	1064	Services	0	5.372 K
wininit.exe	1196	Services	0	5.852 K
csrss.exe	1216	Console	1	6.500 K
services.exe	1268	Services	0	14.280 K
LsaIso.exe	1288	Services	0	3.016 K
lsass.exe	1304	Services	0	22.092 K
svchost.exe	1420	Services	0	34.256 K
fontdrvhost.exe	1448	Services	0	4.576 K
WUDFHost.exe	1468	Services	0	18.084 K
svchost.exe	1564	Services	0	21.492 K
svchost.exe	1612	Services	0	7.772 K
WUDFHost.exe	1668	Services	0	5.248 K
winlogon.exe	1744	Console	1	12.128 K
fontdrvhost.exe	1792	Console	1	16.008 K
dwm.exe	1872	Console	1	544.520 K
svchost.exe	1936	Services	0	5.016 K
svchost.exe	1956	Services	0	8.128 K
svchost.exe	1968	Services	0	12.356 K
svchost.exe	1984	Services	0	12.032 K
svchost.exe	1100	Services	0	9.772 K
svchost.exe	1116	Services	0	9.008 K
svchost.exe	1324	Services	0	8.508 K
IntelCpHDCPSvc.exe	1316	Services	0	6.952 K
svchost.exe	1220	Services	0	5.512 K
svchost.exe	2076	Services	0	8.164 K

Nome	Status	6%	77%	2%	0%
		CPU	Memória	Disco	Rede
Windows Explorer		0%	75,0 MB	0 MB/s	0 Mbps
<b>Processos em segundo plano ...</b>					
Acrobat Collaboration Synchr...		0%	1,6 MB	0 MB/s	0 Mbps
Acrobat Collaboration Synchr...		0%	0,1 MB	0 MB/s	0 Mbps
Acrobat Update Service (32 bits)		0%	0,1 MB	0 MB/s	0 Mbps
Adaptador Reverso de Desem...		0%	0,1 MB	0 MB/s	0 Mbps
AgentService.exe		0%	0,3 MB	0 MB/s	0 Mbps
Antimalware Service Executable		2,5%	134,9 MB	0 MB/s	0 Mbps
Aplicativo de subsistema de s...		0%	2,6 MB	0 MB/s	0 Mbps
Application Frame Host		0%	0,7 MB	0 MB/s	0 Mbps
Application Web Server Daem...		0%	0,3 MB	0 MB/s	0 Mbps
Bonjour Service (32 bits)		0%	0,3 MB	0 MB/s	0 Mbps
Carregador CTF		0%	3,2 MB	0 MB/s	0 Mbps
Cisco AnyConnect User Interf...		0%	0,7 MB	0 MB/s	0 Mbps
CodeMeter Runtime Server (3...		0%	1,9 MB	0 MB/s	0 Mbps

## Processos e threads

Exemplo de processos e tarefas na Raspberry Pi – comando “**top**” (mostra os processos em execução : identificação do processo, qual usuário está executando). **PR** – prioridade; **NI** “nice value” – afeta a prioridade; **VIR**, **RES**, **SHR**, **%MEM** – memória virtual e física usada e compartilhada; **S** – status; **%CPU** e **TEMPO+** consumo da CPU e tempo total do processo.

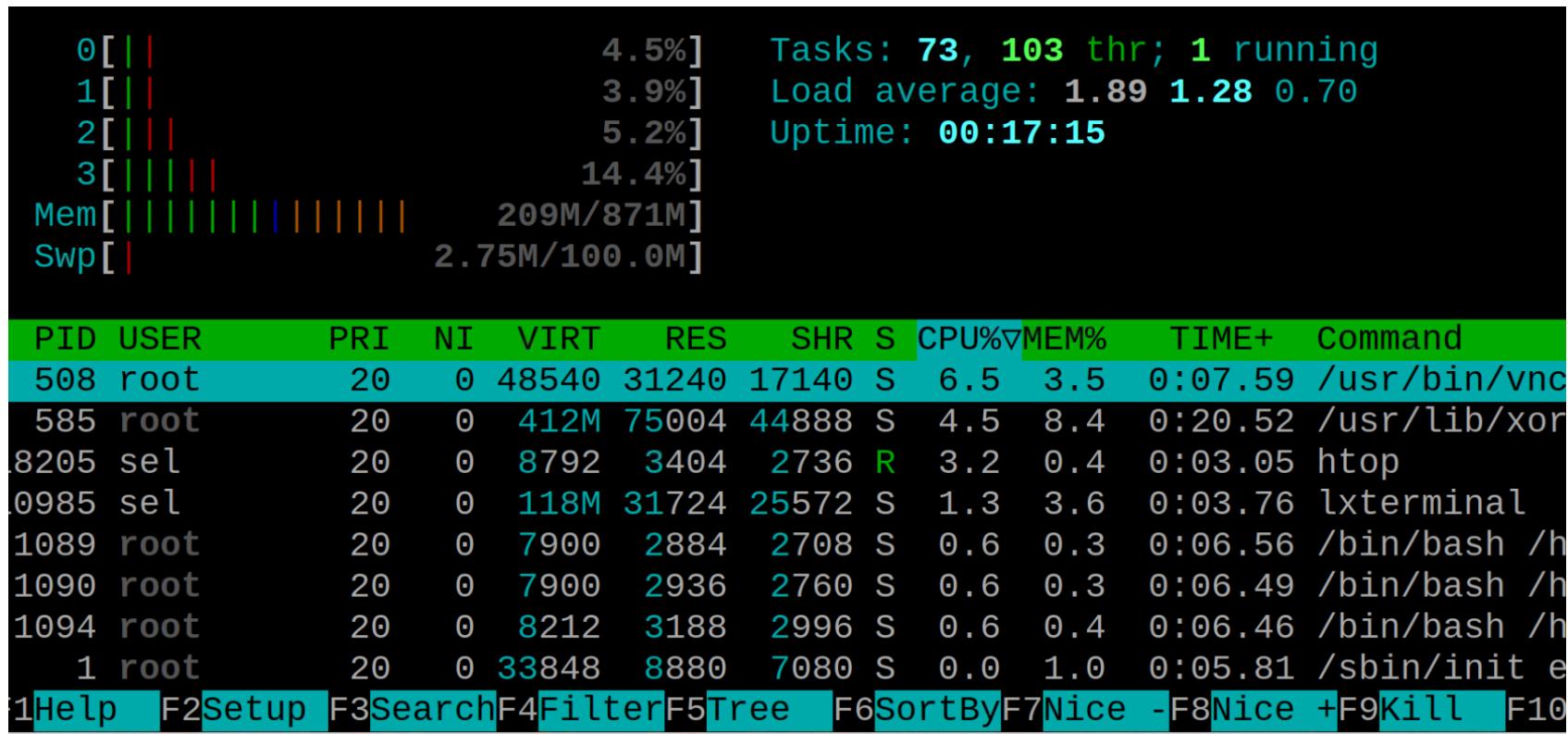
```
sel@raspberrypi:~ $ top
```

top - 15:36:13 up 19 min, 2 users, load average: 0,87, 1,13, 0,72  
Tarefas: 170 total, 1 em exec., 169 dormindo, 0 parado, 0 zumbi  
%Cpu(s): 5,2 us, 3,9 sy, 0,0 ni, 88,2 id, 2,3 wa, 0,0 hi, 0,4 si  
MB mem : 871,0 total, 327,1 livre, 191,4 usados, 352,5 buf  
MB swap: 100,0 total, 97,2 livre, 2,8 usados, 609,6 mem

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TEMPO+
508	root	20	0	48540	31240	17140	S	13,3	3,5	0:16.09
585	root	20	0	422752	74952	44888	S	4,3	8,4	0:25.46
22051	sel	20	0	120500	30788	24772	S	2,3	3,5	0:01.69
69	root	0	-20	0	0	0	I	1,3	0,0	0:02.70
<b>22248</b>	<b>sel</b>	<b>20</b>	<b>0</b>	<b>11456</b>	<b>3260</b>	<b>2696</b>	<b>R</b>	<b>1,0</b>	<b>0,4</b>	<b>0:00.49</b>
652	root	20	0	18460	11616	11096	S	0,7	1,3	0:01.66
1089	root	20	0	7900	2884	2708	S	0,7	0,3	0:35.01

## Processos e threads

Exemplo de gerencia de processos e tarefas na Raspberry Pi: o comando “**htop**” mostra a execução dos processos em paralelo nos 4 núcleos da Raspberry Pi. Outros comandos: “**pstree**” (visão hierárquica), **free -h** (memória disponível)



The screenshot shows the **htop** process monitor running on a Raspberry Pi. At the top, it displays CPU usage bars for four cores (0, 1, 2, 3) and system statistics: Tasks: 73, 103 thr; 1 running, Load average: 1.89 1.28 0.70, and Uptime: 00:17:15. Below this, it shows memory usage: Mem: 209M/871M and Swap: 2.75M/100.0M.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
508	root	20	0	48540	31240	17140	S	6.5	3.5	0:07.59	/usr/bin/vnc
585	root	20	0	412M	75004	44888	S	4.5	8.4	0:20.52	/usr/lib/xor
8205	sel	20	0	8792	3404	2736	R	3.2	0.4	0:03.05	htop
10985	sel	20	0	118M	31724	25572	S	1.3	3.6	0:03.76	lxterminal
1089	root	20	0	7900	2884	2708	S	0.6	0.3	0:06.56	/bin/bash /h
1090	root	20	0	7900	2936	2760	S	0.6	0.3	0:06.49	/bin/bash /h
1094	root	20	0	8212	3188	2996	S	0.6	0.4	0:06.46	/bin/bash /h
1	root	20	0	33848	8880	7080	S	0.0	1.0	0:05.81	/sbin/init e

At the bottom, there is a footer with various keyboard shortcuts:

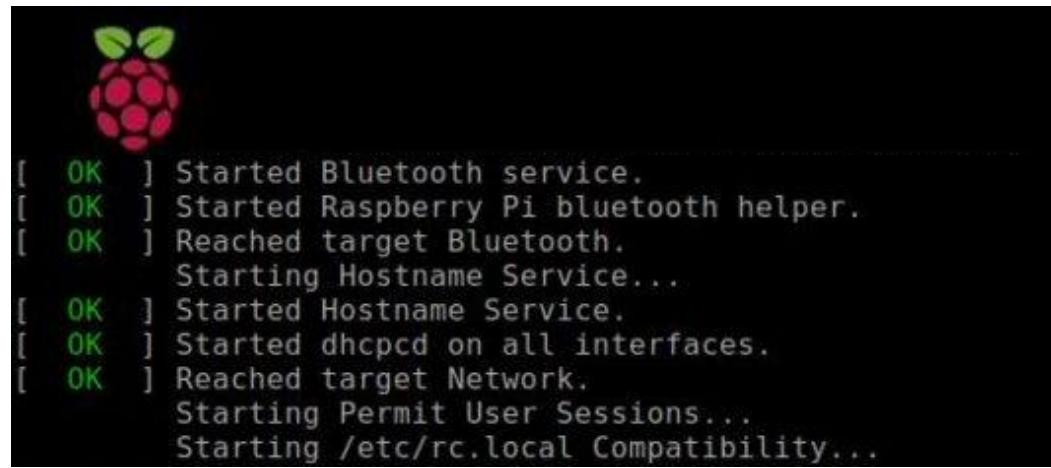
- F1 Help
- F2 Setup
- F3 Search
- F4 Filter
- F5 Tree
- F6 Sort By
- F7 Nice -
- F8 Nice +
- F9 Kill
- F10

## Processos e threads

**Exemplo de processos e tarefas na Raspberry Pi:** inicialização de processos no sistema operacional e visualização dos arquivos de configuração dos serviços, quando eles forem executados (processos)

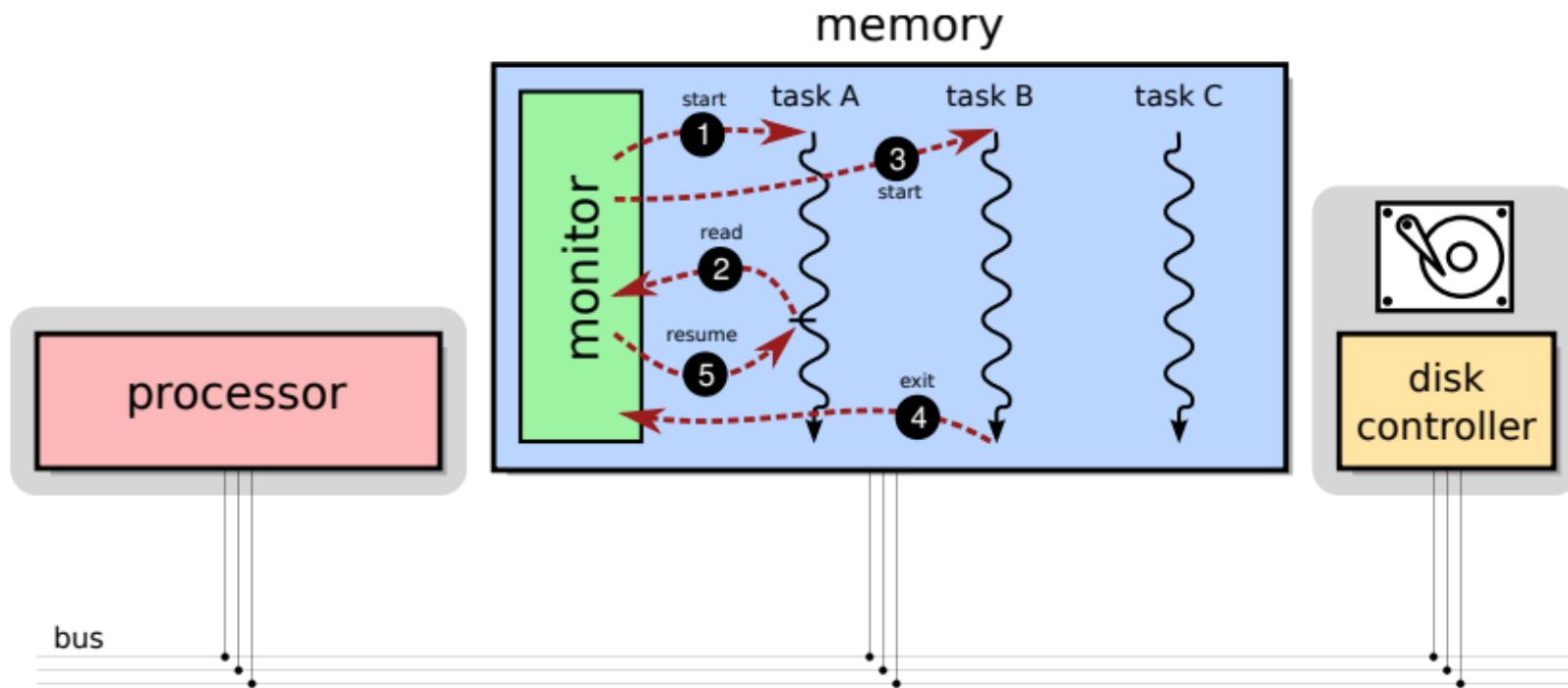
```
sel@raspberrypi:~ $ ls /lib/systemd/system
alsa-restore.service
alsa-state.service
alsa-utils.service
apply_noobs_os_config.service
apt-daily.service
apt-daily.timer
```

O *systemd* é usado na Raspberry Pi: ao ligar/desligar a placa, note que aparece a tela abaixo com “Ok” verde em cada operação no sistema operacional Raspbian, indicando que o referido serviço foi inicializado (ou finalizado) com sucesso (a lista de serviços disponíveis pode ser vista em <[/lib/systemd/system](#)>).



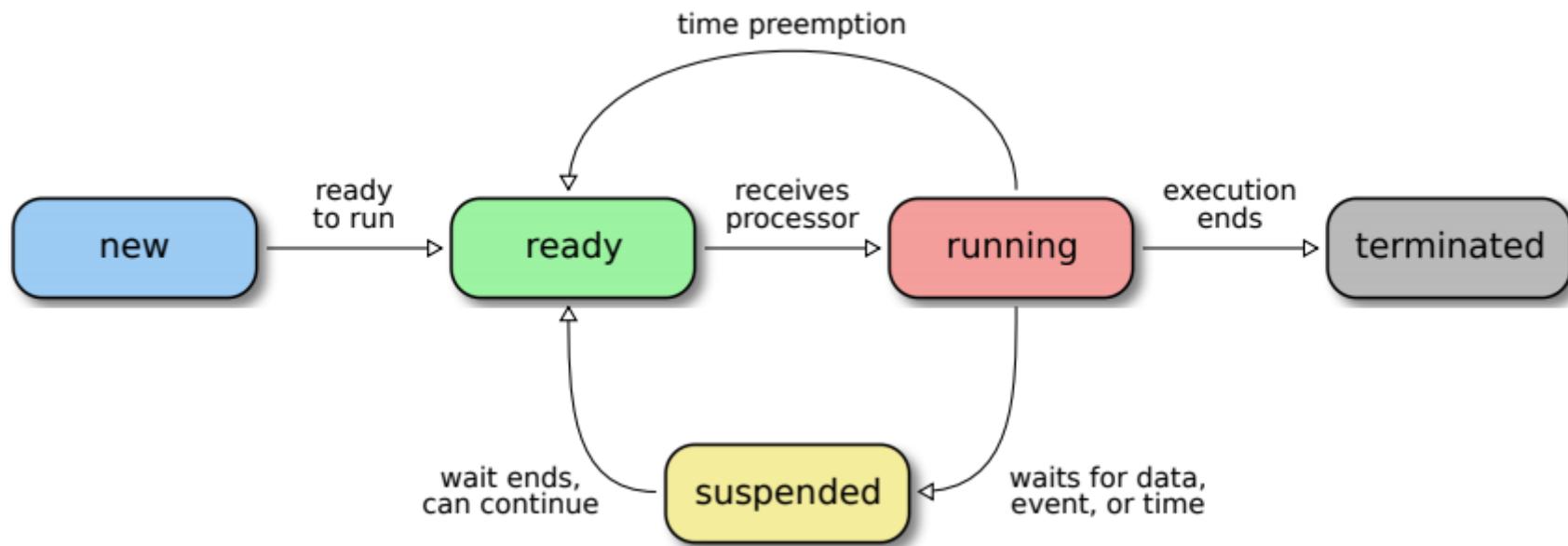
## Recursos do Kernel – gerenciamento de tarefas

- ✓ Os passos numerados representam a execução (start), suspensão (read), retomada (resume) e conclusão (exit) de tarefas pelo monitor. A tarefa A é suspensa ao solicitar uma leitura de dados, sendo retomada mais tarde, após a execução da tarefa B



## Recursos do Kernel – gerenciamento de tarefas

- ✓ Ciclo de vida de tarefas: new, ready, running, suspended, e terminated



## Recursos do Kernel – gerenciamento de tarefas e preempção

- ✓ O **Gerenciamento de Tarefas (TM)** é essencial em sistemas multitarefa para garantir que os recursos do sistema sejam alocados eficientemente entre os processos em execução. O TM atribui prioridades às tarefas, garantindo que as mais importantes recebam recursos suficientes e sejam executadas mesmo quando outras tarefas estão em andamento.

Um exemplo popular de algoritmo de escalonamento é o **Round Robin**, em que para cada tarefa é atribuída um pequeno intervalo de tempo de CPU. Quando esse intervalo expira, a próxima tarefa na fila é selecionada para execução. Isso garante uma distribuição justa do tempo de CPU entre todas as tarefas.

A preempção permite que tarefas críticas interrompam outras de menor prioridade ou evita que um algoritmo entre em loop infinito e monopolize a CPU.

## Recursos do Kernel – comunicação entre processos

- ✓ A **Comunicação entre Processos (IPC)** permite que diferentes processos cooperem, compartilhem informações e sincronizem suas ações.

### Filas de Mensagens: Transmitindo Informações

- ✓ Uma forma de IPC é a troca de mensagens por meio de filas. Processos podem enviar mensagens para uma fila e outros processos podem lê-las a partir dessa fila. Isso permite a transferência de informações entre processos de maneira organizada e segura.

### Semáforos: Coordenando Acesso a Recursos

- ✓ Um semáforo age como um sinal de trânsito. Ele controla o acesso a recursos compartilhados. Um processo pode usar um semáforo para indicar que está ocupando um recurso, impedindo que outros processos o acessem até que ele seja liberado.

## Sockets

Sockets são interfaces de comunicação que permitem a troca de dados entre processos, seja no mesmo computador ou através da rede.

Eles implementam a base da comunicação cliente-servidor, podendo operar com protocolos como TCP (confiável, orientado à conexão) ou UDP (leve, sem conexão).

No contexto de sistemas operacionais, sockets são um mecanismo clássico de IPC (Inter-Process Communication), como se fossem uma “tomada” que permite, por exemplo, que um navegador (cliente) se conecte a um servidor web.

```
ss -tulnp  # lista sockets TCP/UDP abertos e processos associados  
lsof -i    # mostra quais processos estão usando a rede
```

## Requisitos mínimos de hardware para rodar S.O.

- ✓ É necessário um **núcleo de 32 ou 64 bits** (ex: X86, ARM Cortex-A, RISC-V com suporte adequado), já que sistemas operacionais não são compatíveis com microcontroladores.
- ✓ São necessários algumas centenas de megabytes de **RAM** e **armazenamento não volátil** (como flash, disco, SSD) para o kernel, sistema de arquivos e aplicações.
- ✓ É necessário a presença de **interfaces de I/O** padronizadas (barramentos, Ethernet, USB, etc.), pois o sistema operacional (kernel) depende de drivers para interação com dispositivos e periféricos de hardware.
- ✓ Um programa é apenas um conjunto de instruções armazenado em disco (arquivo binário executável). Quando esse programa é carregado pelo sistema operacional e passa a ser executado, ele se torna um processo.
- ✓ Programas não acessam diretamente a memória física (endereços reais da RAM). Isso ocorre porque o kernel, por meio da **MMU** (Unidade de Gerenciamento de Memória), fornece a cada processo um espaço de endereçamento virtual.

## Recursos do Kernel – gerenciamento de memória

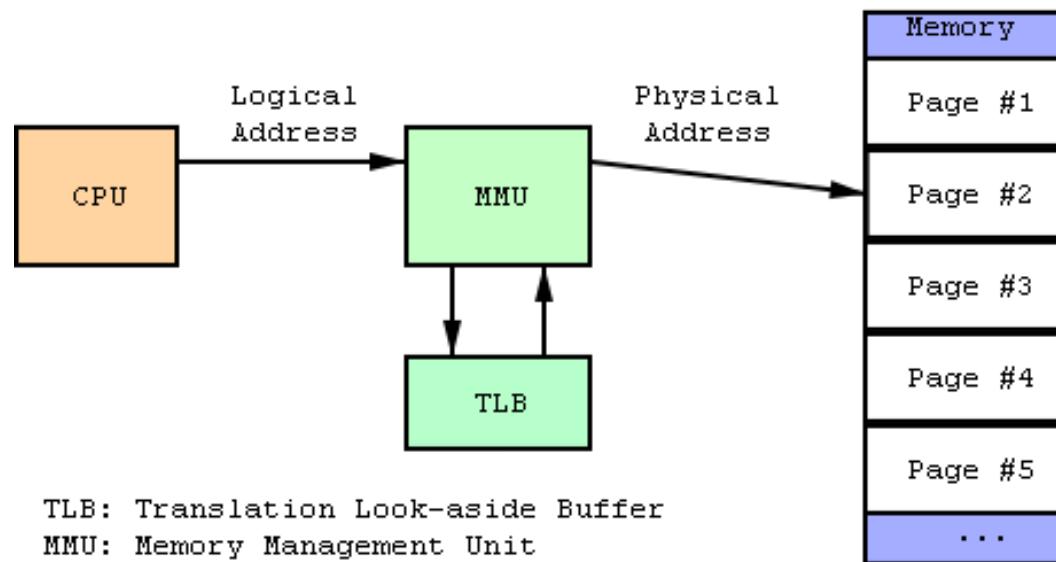
- ✓ O Gerenciamento de Memória (MM) é fundamental para garantir que os aplicativos tenham acesso à memória necessária para armazenar dados e instruções. À medida que os aplicativos são executados, a memória é alocada e liberada. Isso pode resultar em espaços vazios de memória, conhecidos como fragmentação. A fragmentação pode ser interna (espaço não utilizado dentro de um bloco alocado) ou externa (espaço não utilizado entre os blocos alocados). Estratégias de alocação visam minimizar a fragmentação.

**Memória Compartilhada** - A memória compartilhada é um mecanismo de IPC mais direto. Vários processos podem acessar a mesma área de memória, permitindo que compartilhem dados de maneira eficiente. Existem diferentes estratégias para alocar memória aos aplicativos:

- ✓ **Primeiro Encaixe (First Fit):** Aloca o primeiro espaço de memória que atende ao requisito do aplicativo. Suponha que um aplicativo solicite 20 KB de memória. O MM aloca o primeiro bloco de 20 KB que atende a esse requisito, mesmo que haja espaços menores disponíveis antes desse bloco. Isso pode resultar em fragmentação interna.
- ✓ **Melhor Encaixe (Best Fit):** Aloca o espaço de memória mais próximo do tamanho necessário. Se um aplicativo solicitar 20 KB de memória, o MM aloca o bloco mais próximo desse tamanho, mesmo que haja espaços menores disponíveis. Isso ajuda a minimizar a fragmentação interna, mas pode deixar espaços não utilizados.
- ✓ **Pior Encaixe (Worst Fit):** Aloca o maior espaço disponível, deixando um espaço fragmentado menor.

## Recursos do Kernel – gerenciamento de memória

- ✓ O Linux (kernel), por exemplo, trabalha com um mecanismo de **memória virtual** para gerenciar a memória do sistema.
- ✓ Todo o acesso à memória é realizado através de endereços virtuais, que são convertidos (em hardware) para endereços físicos durante o acesso à memória do sistema.
- ✓ A **MMU (Memory Management Unit)** é o hardware que implementa o mecanismo de memória virtual e realiza o gerenciamento acima referido, fornecendo: **proteção, eficiência, flexibilidade e paginação**.



## MMU (Memory Management Unit)

- Cada processo “enxerga” a memória como se fosse contínua e exclusiva para ele, mas na prática o kernel mapeia esses endereços virtuais para regiões específicas da memória física.  
Esse mecanismo:
- ✓ Evita que um processo invada a memória de outro (proteção de memória).
  - ✓ Permite usar recursos como memória virtual e swap, estendendo a RAM com espaço em disco.
  - ✓ Facilita o carregamento eficiente de programas e bibliotecas, já que endereços virtuais podem ser remapeados sem copiar dados fisicamente.

## Recursos do Kernel – gerenciamento de memória

- ✓ Visualizando informações de gerenciamento de memória no S.O e espaço em disco: exemplo na Raspberry Pi – “`cat /proc/meminfo`” – “`df -h`”

```
sel@raspberrypi:~ $ cat /proc/meminfo
MemTotal:       891872 kB
MemFree:        330104 kB
MemAvailable:   624108 kB
Buffers:        23088 kB
Cached:         325064 kB
SwapCached:     32 kB
Active:          114836 kB
Inactive:       378016 kB
Active(anon):   9868 kB
Inactive(anon): 152952 kB
Active(file):   104968 kB
Inactive(file): 225064 kB
Unevictable:    12 kB
Mlocked:        12 kB
```

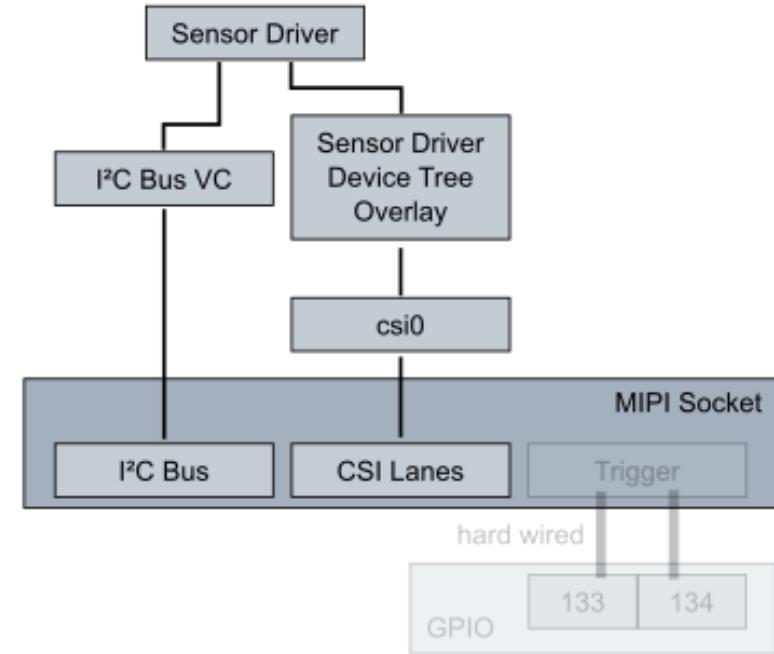
```
sel@raspberrypi:~ $ df -h
Sist. Arq.      Tam. Usado Disp. Uso% Montado em
/dev/root       29G  5,9G  22G  22% /
devtmpfs        403M   0   403M  0% /dev
tmpfs           436M   0   436M  0% /dev/shm
tmpfs           175M  988K  174M  1% /run
tmpfs           5,0M   0   5,0M  0% /run/lock
/dev/mmcblk0p1   255M   51M  205M  20% /boot
tmpfs           88M   20K   88M  1% /run/user/1000
```

## Recursos do Kernel – gerenciamento de dispositivos

- ✓ Quando se imprime um documento ou transfere arquivos para um dispositivo USB, o kernel interage com os drivers de dispositivo específicos para garantir que as operações sejam executadas corretamente.
- ✓ Por exemplo, ao imprimir, o kernel recebe os dados do aplicativo, os envia ao driver da impressora e garante que os dados sejam transmitidos de maneira adequada.

### Drivers de Dispositivo (device drivers):

- ✓ Permitem que o sistema operacional interaja com hardware específico, como placas de rede, discos rígidos e dispositivos USB.
- ✓ Cada driver fornece uma interface padronizada para que os aplicativos possam acessar e controlar o hardware.
- ✓ Por exemplo: **driver de comunicação externa (USB, rede, sensor, I2C etc)**



## Recursos do Kernel – gerenciamento de dispositivos

- ✓ **Visualizando drivers de dispositivo no S.O:** exemplo na Raspberry Pi com uso dos comandos no terminal Linux: “**lsusb**” (driver USB), “**lsmod**” (módulos do kernel Linux), “**ls /proc/device-tree**” (diretório da árvore de dispositivos do sistema) ; **cat /proc/devices** (dispositivos do hardware)

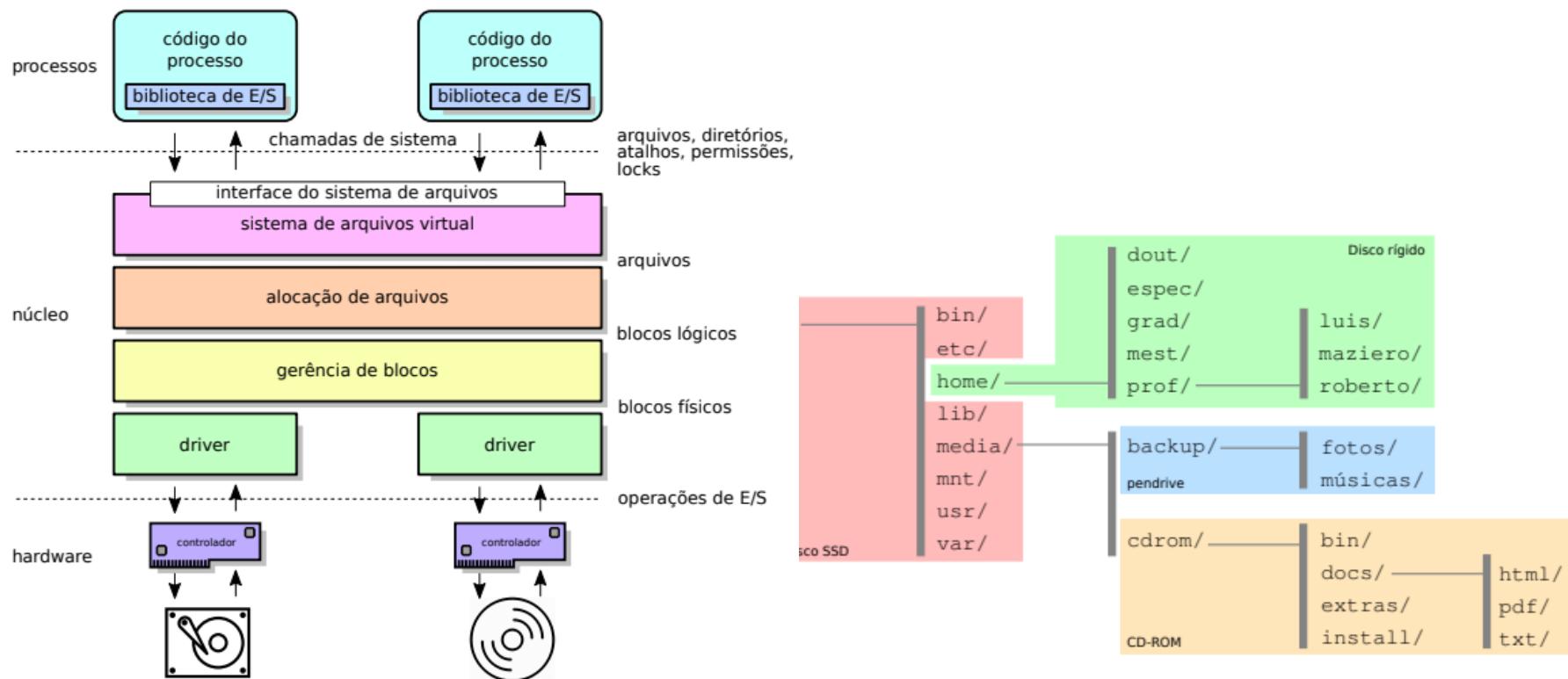
```
sel@raspberrypi:~ $ sudo lsusb
Bus 001 Device 005: ID 0424:7800 Microchip Technology, Inc. (for
  USB 2.0 Hub)
Bus 001 Device 003: ID 0424:2514 Microchip Technology, Inc. (for
  USB 2.0 Hub)
Bus 001 Device 002: ID 0424:2514 Microchip Technology, Inc. (for
  USB 2.0 Hub)
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

## Sistemas de arquivos

- ✓ Os sistemas de arquivos são responsáveis por organizar e armazenar os dados no disco rígido. A hierarquia de diretórios organiza os arquivos em uma árvore, com o diretório raiz no topo, o que permite que os arquivos sejam agrupados por categorias, facilitando a localização e o gerenciamento.
  
- ✓ **Ext3:** Um sistema de arquivos padrão para muitas distribuições Linux. Ele é confiável e robusto, mas pode ter desempenho mais lento em comparação com sistemas mais modernos.
- ✓ **Ext4:** Uma evolução do Ext3, com melhorias de desempenho e recursos adicionais, como suporte a tamanhos maiores de arquivos e sistemas de arquivos.
- ✓ **NTFS:** Usado no Windows, o NTFS é um sistema de arquivos rico em recursos, oferecendo suporte a permissões de arquivo, criptografia e compressão.
- ✓ **VFAT:** Um sistema de arquivos usado para dispositivos de armazenamento portáteis, como pen drives. Ele fornece compatibilidade entre sistemas operacionais, mas pode ter algumas limitações.

BOOT	RASPBERRY (rootfs)
FAT32	EXT4
<ul style="list-style-type: none"><li>• bootcode.bin</li><li>• start.elf</li><li>• kernel7.img</li><li>• fixup.dat</li><li>• config.txt</li><li>• cmdline.txt</li><li>• *.dtb</li></ul>	<ul style="list-style-type: none"><li>• /bin</li><li>• /boot</li><li>• /dev</li><li>• /etc</li><li>• /home</li><li>• /lib</li></ul> <ul style="list-style-type: none"><li>• /media</li><li>• /mnt</li><li>• /opt</li><li>• /proc</li><li>• /root</li><li>• /run</li></ul> <ul style="list-style-type: none"><li>• /sbin</li><li>• /srv</li><li>• /sys</li><li>• /tmp</li><li>• /usr</li><li>• /var</li></ul>

## Sistemas de arquivos e diretórios



## Sistemas de arquivos e diretórios

Exemplos de acesso a diretórios do S.O.: comandos **tree, ls, ls-a, ls -l, cd, pwd** etc.

```
SEL0337
├── P2.html
├── README.md
└── teste2.txt
    └── teste3.txt
teste.py
tutorial_git_github.txt
Untitled Sketch_bb.jpg
Untitled Sketch.fzz
Videos

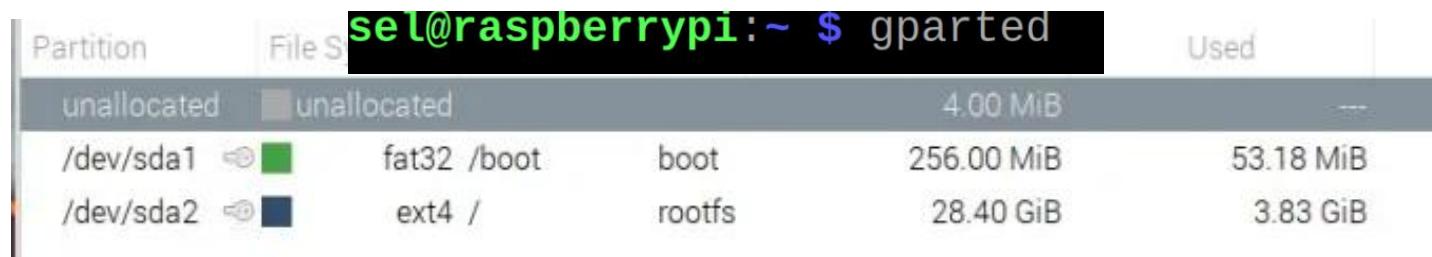
349 directories, 2757 files
sel@raspberrypi:~ $ tree
```

```
sel@raspberrypi:~ $ ls -a
.
..
2022-12-13-153212_1824x984_scrot.png
2022-12-13-153225_1824x984_scrot.png
2022-12-13-153226_1824x984_scrot_000.png
2022-12-13-153226_1824x984_scrot_001.png
2022-12-13-153226_1824x984_scrot_002.png
2022-12-13-153226_1824x984_scrot_003.png
2022-12-13-153226_1824x984_scrot_004.png
2022-12-13-153226_1824x984_scrot_005.png
2022-12-13-153226_1824x984_scrot_006.png
2022-12-13-153226_1824x984_scrot_007.png
2022-12-13-153226_1824x984_scrot_008.png
.cups
Desktop
Documentos
Downloads
fetch_local_weather.py
.gitconfig
git_github
git_passos.py
git_passos.py.save
git_projects
git_projetos
.gitignore
.history.txt
```

```
sel@raspberrypi:/ $ ls
bin      boot.bak   etc      lib          media     opt      root     sbin     sys      usr
boot     dev        home    lost+found   mnt       proc     run      srv      tmp      var
```

## Sistemas de arquivos de imagens do S.O

- ✓ Os arquivos de imagens são arquivos binários que contêm uma cópia exata de um sistema de arquivos ou dispositivo de armazenamento. Eles são usados para criar backups, clonar sistemas e distribuir sistemas operacionais.
  
- ✓ **dd:** data duplicator - é amplamente usada para copiar e converter arquivos de imagens. Pode ser usado para criar uma cópia exata de um disco ou para criar arquivos de imagens.
- ✓ **gparted:** é uma interface gráfica para redimensionar, criar, excluir e gerenciar partições em discos. Ele também suporta a criação e restauração de arquivos de imagens.
- ✓ **disks:** fornece uma interface gráfica simples para criar, formatar e montar partições e discos, bem como criar arquivos de imagens.



Partition	File S	sel@raspberrypi:~ \$ gparted	Used
unallocated	unallocated		4.00 MiB
/dev/sda1	 fat32	/boot	256.00 MiB
/dev/sda2	 ext4	/	28.40 GiB
		boot	53.18 MiB
		rootfs	3.83 GiB

## Sistemas de arquivos de imagens do S.O

- ✓ Visualizando informações sobre discos e partições montadas no sistema operacional: exemplo na Raspberry Pi com os comandos **gparted**, **lsblk**, **fdisk**, **parted**, e **mount**

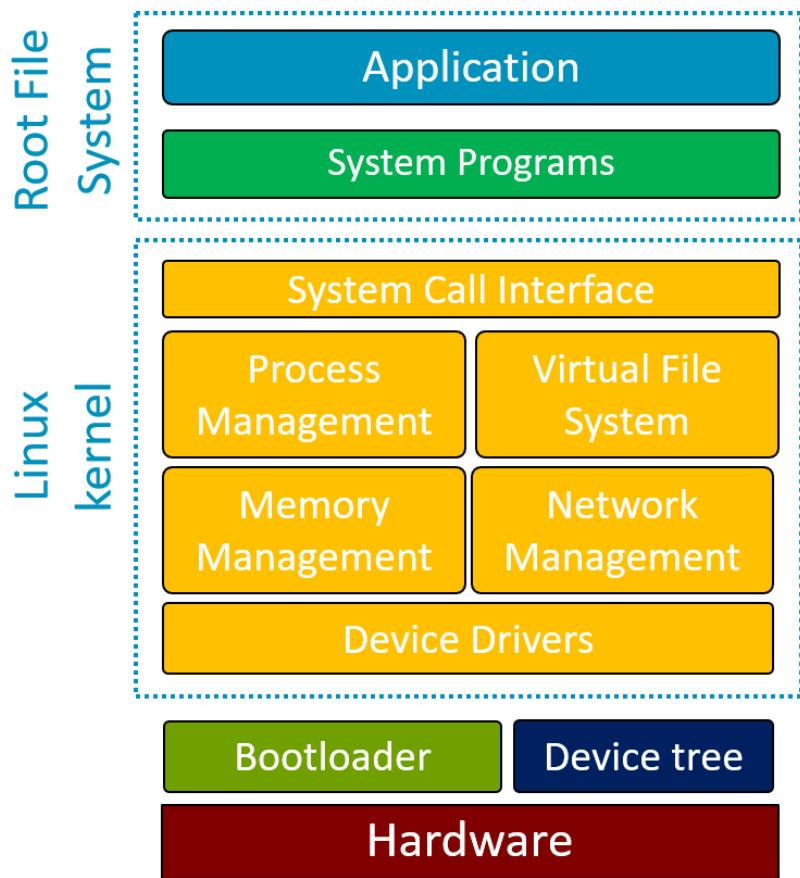
```
Disk /dev/mmcblk0: 28,89 GiB, 31016878080 bytes, 60579840 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x5986eb18

Device      Boot   Start     End   Sectors  Size Id Type
/dev/mmcblk0p1        8192   532479   524288  256M  c W95 FAT32 (LBA)
/dev/mmcblk0p2    532480 60579839 60047360 28,6G 83 Linux
sel@raspberrypi:~ $ sudo fdisk -l
```

```
sel@raspberrypi:~ $ lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0    179:0    0 28,9G  0 disk
└─mmcblk0p1 179:1    0  256M  0 part /boot
└─mmcblk0p2 179:2    0 28,6G  0 part /
sel@raspberrypi:~ $ mount
/dev/mmcblk0p2 on / type ext4 (rw,noatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=4126
ode=755)
proc on /proc type proc (rw,relatime)
```

## Resumindo a arquitetura, módulos e recursos do S.O.

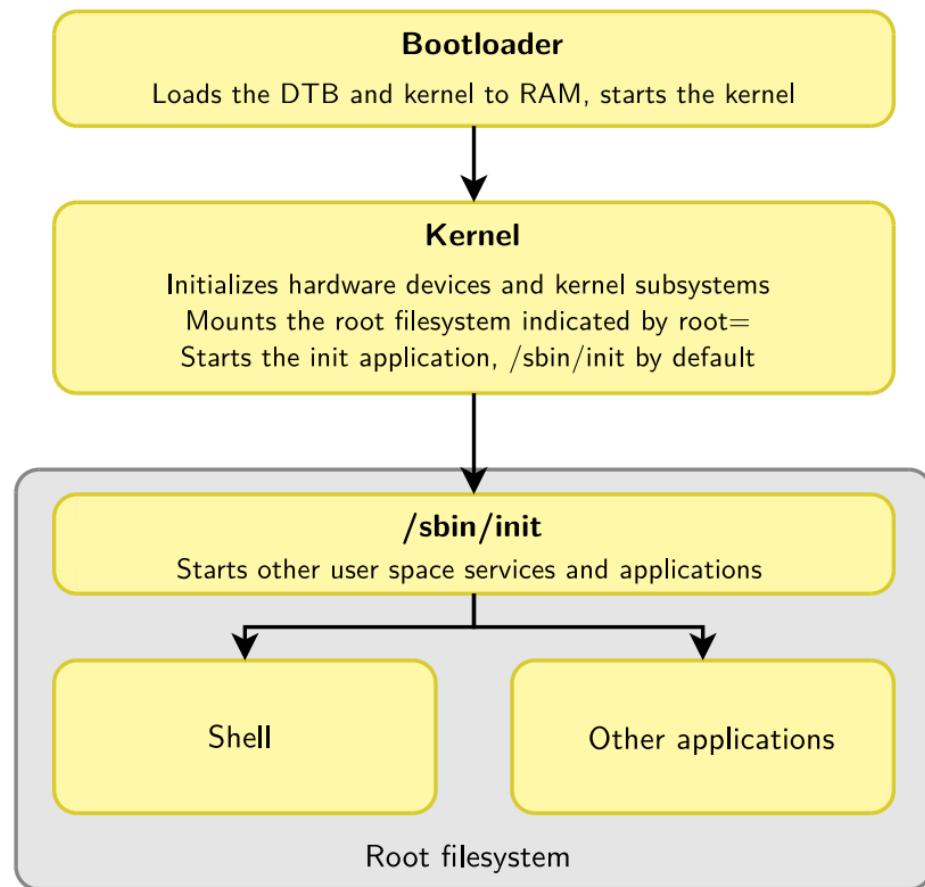
### Arquitetura de um Linux embarcado



® ARM Limited - 2017

Fonte: ® Bootlin - <https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf>

### Boot do Sistema Operacional



## Resumindo a arquitetura, módulos e recursos do S.O.

- ✓ **Bootloader:** carrega a inicialização (boot) do sistema operacional (firmware na ROM que inicializa a CPU, periféricos e o kernel)
- ✓ **Device tree:** estrutura de dados que descreve/mapeia a configuração do hardware no sistema operacional. Essa informação é usada pelo Kernel, durante o Boot, para reconhecer, carregar e gerenciar os drivers apropriados dos dispositivos de hardware.
- ✓ **Linux Device driver:** biblioteca de rotinas para manipulação em baixo nível de informações contidas na memória para o funcionamento e reconhecimento de dispositivos de hardware no sistema Linux.
- ✓ **System programs:** utilitários de acesso ao serviços do sistema operacional;
- ✓ **Root filesystem:** sistemas de arquivos em partições e contender com as configurações do Kernel, programa e aplicações

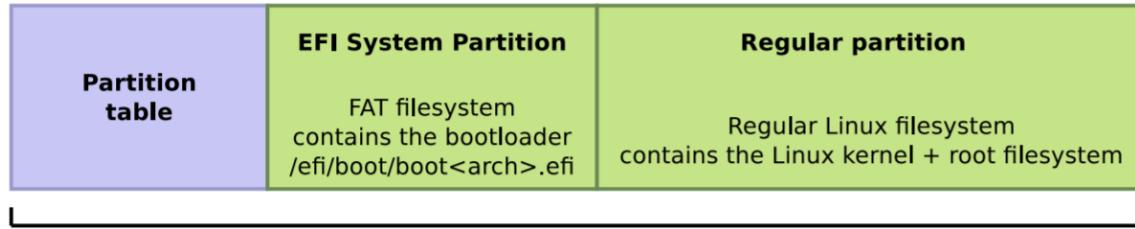
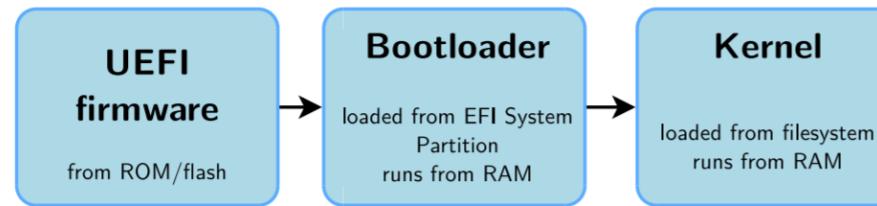
Bootloaders populares: **GRUB, LILO, U-Boot**



**U-Boot**

## Sequência de Boot – PC

- ✓ **Unified Extensible Firmware Interface (UEFI)** é um firmware moderno e mais flexível usado para inicializar sistemas operacionais (permite mais recursos, interface gráfica, substitui o BIOS tradicional) – **firmware gravado na ROM do computador pelo fabricante.**
- ✓ **Extensible Firmware Interface (EFI)** - **partição de disco especial usada pelo sistema de inicialização UEFI para armazenar arquivos e informações necessárias para a inicialização do sistema.**



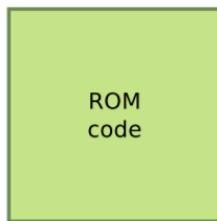
USB drive, SATA,  
SD card, eMMC, etc.

<https://bootlin.com>

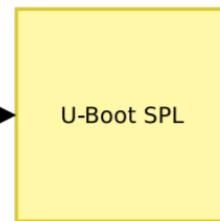
## Sequência de Boot do S.O. embarcado (ARM)

Firmware  
gravado na  
ROM no  
SoC

**1º estágio do  
bootloader na  
partição boot  
(SD card/ flash) –  
inicialização  
básica da RAM**



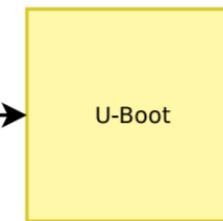
Stored in ROM



Loaded from a file  
called MLO in a FAT  
filesystem in the  
first bootable  
partition

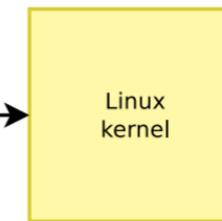
Runs from SRAM

**2º estágio do bootloader na  
partição boot (SD card/  
flash) – inicialização de  
drivers e carregamento do  
kernel (ex. firmware U-  
Boot)**



Loaded from a file  
called u-boot.img in a FAT  
filesystem

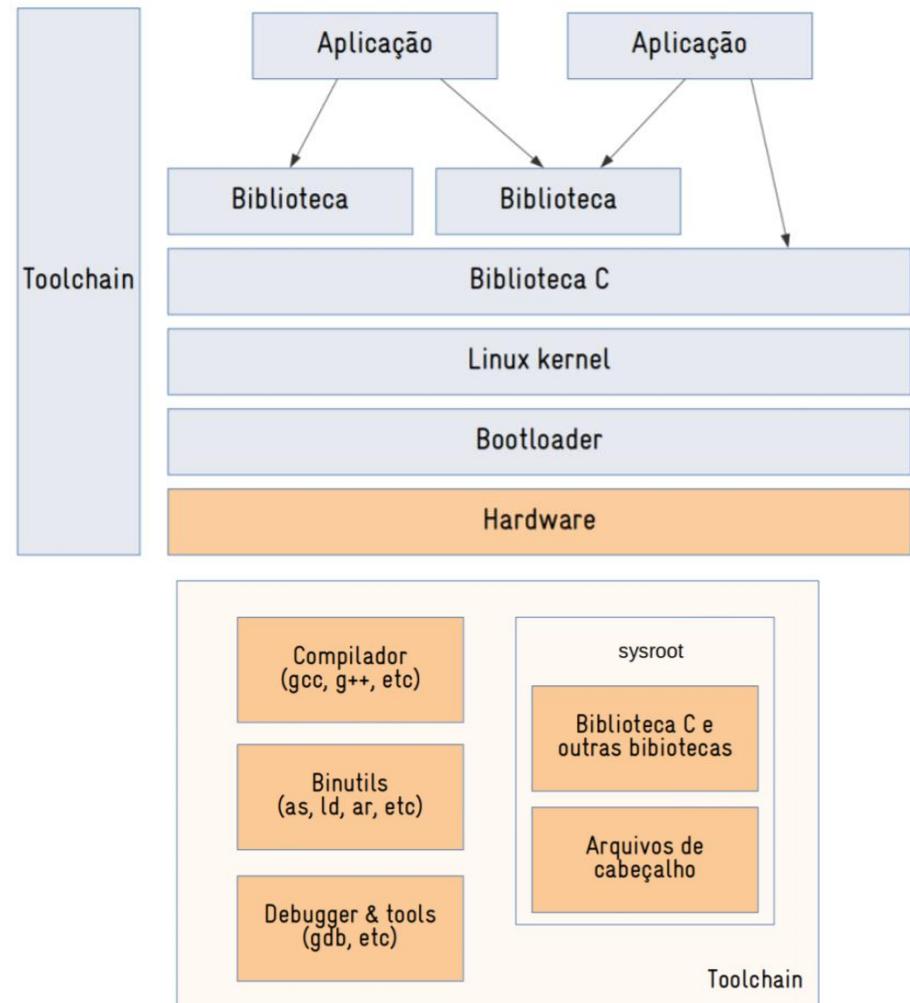
Runs from RAM



<https://bootlin.com>

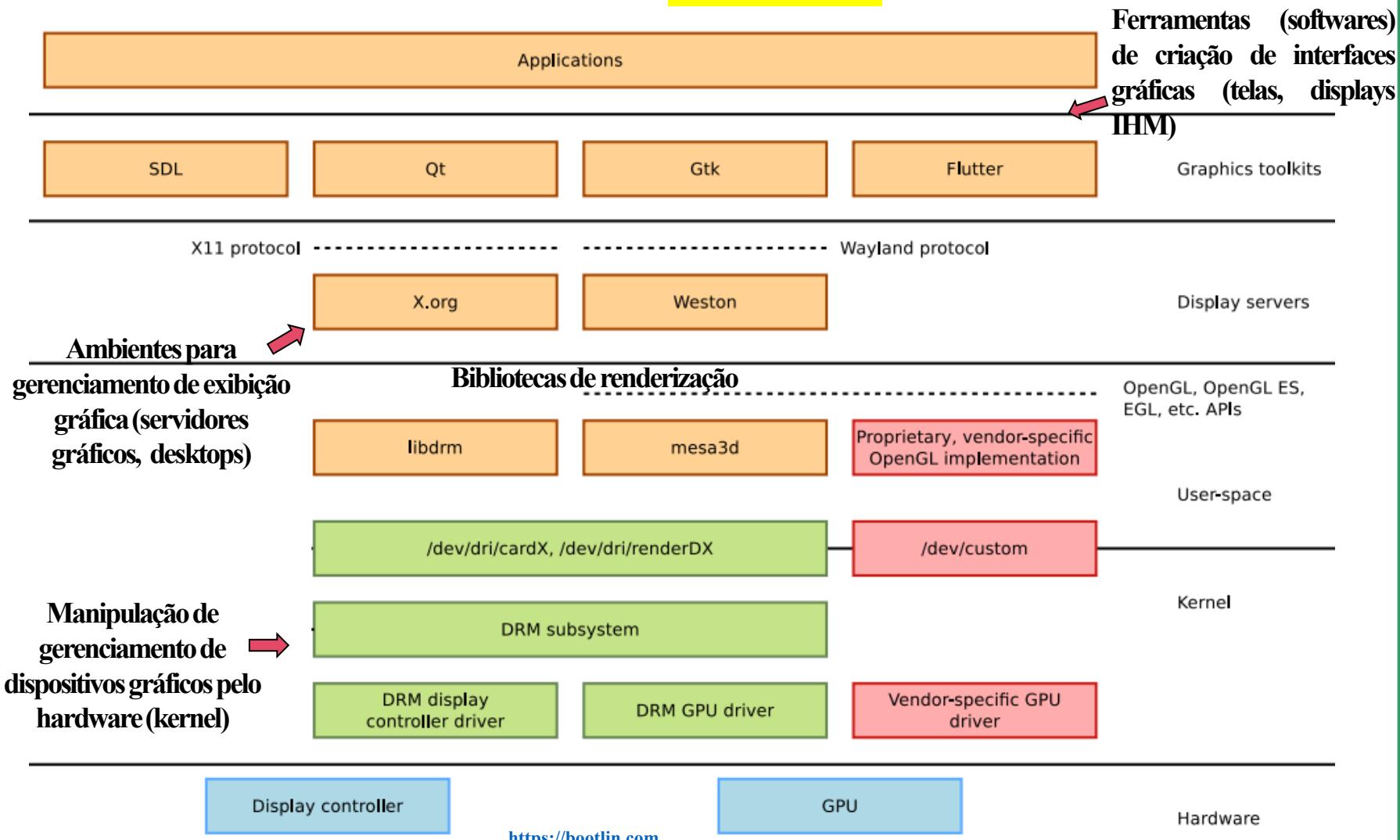
## Desenvolvimento de um Sistema Operacional (Linux)

- ✓ **Hardware:** plataforma alvo (target).
- ✓ **Bootloader**
- ✓ **Kernel Linux:** núcleo do sistema operacional. Gerencia CPU, memória e I/O, exportando serviços para a camada de aplicações.
- ✓ **Rootfs:** sistema de arquivos principal (camada de aplicações do usuário).
- ✓ **Biblioteca C:** API do sistema operacional, interface entre o kernel e as aplicações.
- ✓ **Bibliotecas e aplicações do usuário.**
- ✓ **Toolchain:** conjunto de ferramentas para manipular e gerar os artefatos do sistema operacional (bootloader, kernel, rootfs), conforme a arquitetura (ARM, X86...)



## Camada gráfica

Comando: “`ls -l /dev/dri/`”



## Informações e atualizações do Sistema Operacional

- ✓ Digitar: **neofetch** (informações técnicas do S.O, kernel e distribuição Linux)
- ✓ Digitar: **sudo apt update && sudo apt upgrade** para baixar e instalar atualizações

```
sel@raspberrypi
-----
OS: Raspbian GNU/Linux 11 (bullseye)
Host: Raspberry Pi 3 Model B Plus Re
Kernel: 5.15.76-v7+
Uptime: 1 hour, 34 mins
Packages: 1513 (dpkg)
Shell: bash 5.1.4
Resolution: 1824x984
DE: LXDE
WM: Openbox
Theme: Pixflat [GTK3]
Icons: Pixflat [GTK3]
Terminal: lxterminal
Terminal Font: Monospace 30
CPU: BCM2835 (4) @ 1.400GHz
Memory: 222MiB / 870MiB
```

```
sel@raspberrypi:~ $ sudo apt-get update
0% [Conectando a raspbian.raspberrypi.org]
```

```
sel@raspberrypi:~ $ sudo apt-get upgrade
Lendo listas de pacotes... Pronto
Construindo árvore de dependências... Pronto
Lendo informação de estado... Pronto
Calculando atualização... Pronto
Os pacotes a seguir serão atualizados:
  libcamera-apps libcamera-tools libcamera0 l
  libvlccore9 python3-libcamera pytho
  raspberrypi-ui-mods rpi-eeprom vlc vlc-bin
  vlc-plugin-access-extra vlc-plugin-base vlc
  vlc-plugin-samba vlc-plugin-skins2 vlc-plug
```

## Leitura recomendada

### Operating Systems Foundations with Linux on the Raspberry Pi

By Wim Vanderbauwhede and Dr. Jeremy Singer - ARM Education Media -

<https://www.arm.com/resources/education/books/operating-systems>

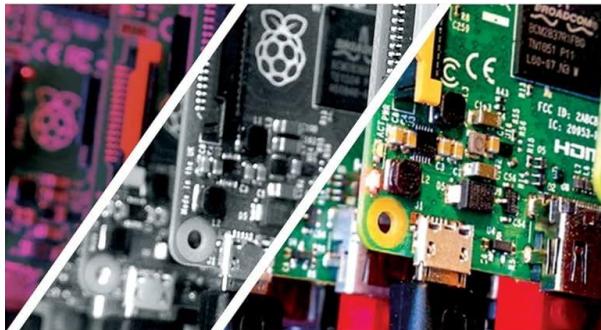
<https://armkeil.blob.core.windows.net/developer/Files/pdf/ebook/arm-operating-systems-foundations-with-linux-on-the-raspberry-pi.pdf>

arm Education Media

### Operating Systems Foundations

with Linux on the Raspberry Pi

TEXTBOOK



Wim Vanderbauwhede

Jeremy Singer

### Operating Systems Foundations with Linux on the Raspberry Pi



9781911531203>

#### Contents

- 1 A Memory-centric System Model
- 2 A Practical View of the Linux System
- 3 Hardware Architecture
- 4 Process Management
- 5 Process Scheduling
- 6 Memory Management
- 7 Concurrency and Parallelism
- 8 Input / Output
- 9 Persistent Storage
- 10 Networking
- 11 Advanced Topics

*"While the modern systems software stack has become large and complex, the fundamental principles are unchanging. Operating Systems must trade off abstraction for efficiency. In this respect, Linux on Arm is particularly instructive. The authors do an excellent job of presenting Operating Systems concepts, with direct links to concrete examples of these concepts in Linux on the Raspberry Pi. Please don't just read this textbook - buy a Pi and try out the practical exercises as you go."*

Steve Furber CBE FRS FREng  
ICL Professor of Computer Engineering,  
The University of Manchester

arm Education Media

Arm Education Media is a publishing operation with Arm Ltd, providing a range of educational materials for aspiring and practicing engineers. For more information, visit: [armedmedia.com](http://armedmedia.com)

## Leitura recomendada

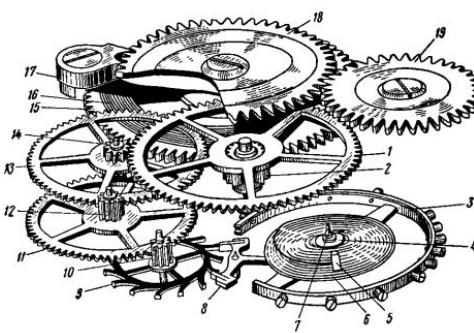
- Carlos Maziero – Sistemas Operacionais – Conceitos e Mecanismos –DINF UFPR

<https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-livro.pdf>

### SISTEMAS OPERACIONAIS: CONCEITOS E MECANISMOS

PROF. CARLOS A. MAZIERO

DINF - UFPR



## Referências e créditos

- ARM Limited <https://www.arm.com>
- ARM Education <https://www.arm.com/resources/education>
- Bootlin - <https://bootlin.com>
- Bruno C. C. Sistemas Operacionais. IFES – 2010 E-TecBrasil
- Carlos Maziero – Sistemas Operacionais – Conceitos e Mecanismos –DINF UFPR - <https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-livro.pdf>
- Embedded Survey- “The current state of embedded development”- 2023  
<https://www.embedded.com/wp-content/uploads/2023/05/Embedded-Market-Study-For-Webinar-Recording-April-2023.pdf>
- Embedded LabWorks - <https://e-labworks.com>
- Bootlin - <https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf>
- Raspberry Pi Foundation. Disponível em  
<https://www.raspberrypi.org> - <https://www.raspberrypi.com>