

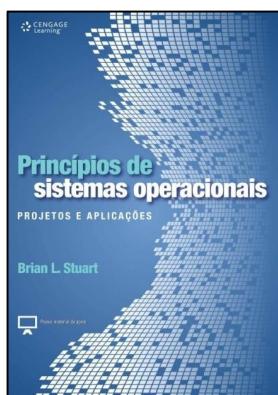
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
SISTEMAS OPERACIONAIS – Aula 01 – 1º SEMESTRE/2020
PROF. LUCIANO SILVA

TEORIA: ORGANIZAÇÃO E BOOTSTRAPPING DE SISTEMAS OPERACIONAIS



Nossos **objetivos** nesta aula são:

- conhecer o conceito de sistema operacional
- conhecer os principais componentes de um sistema operacional
- conhecer o processo de *boot(strapping)* típico de um sistema operacional



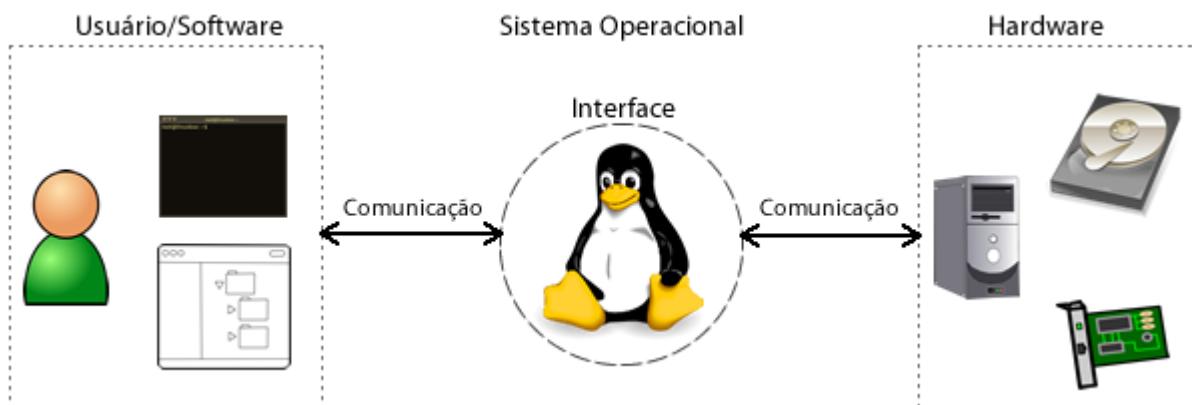
Para esta aula, usamos como referência o **Capítulo 1** do nosso livro-texto:

STUART, B.L., **Princípios de Sistemas Operacionais: Projetos e Aplicações**. São Paulo: Cengage Learning, 2011.

Não deixem de ler este capítulo depois desta aula!

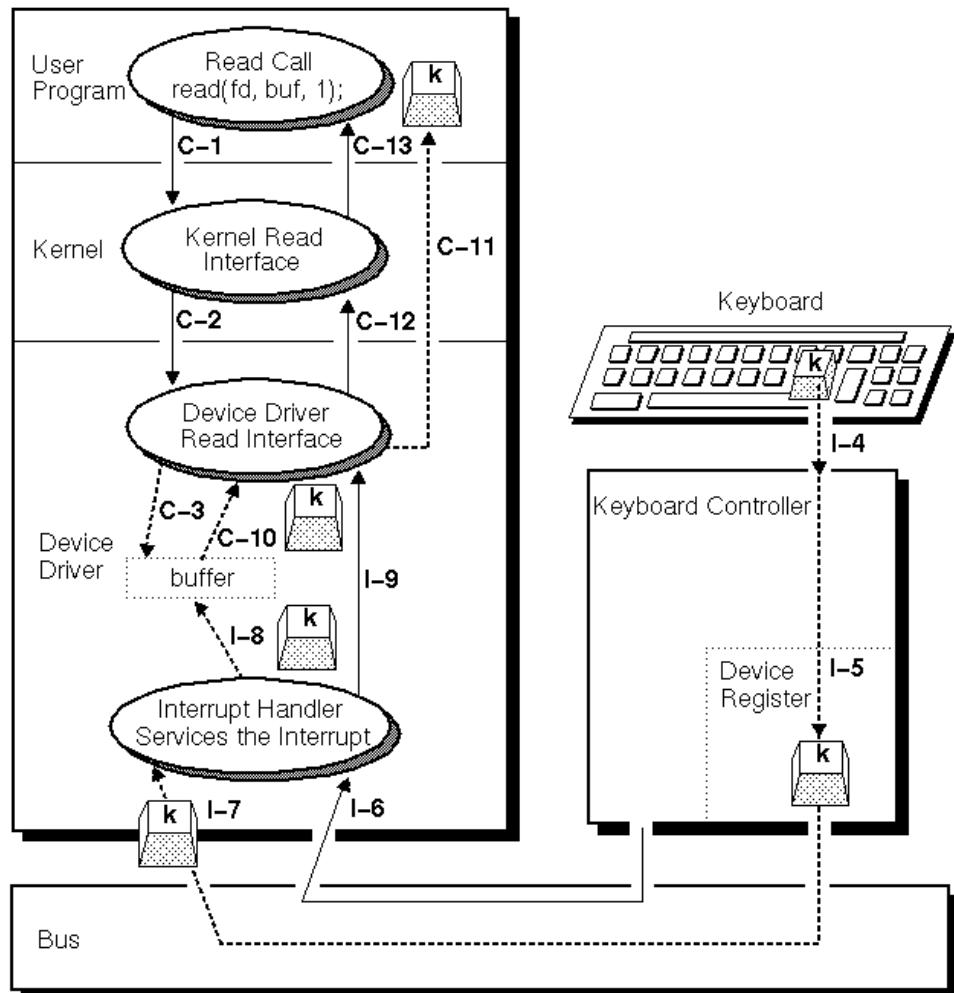
CONCEITO DE SISTEMA OPERACIONAL (S.O.)

- Um **sistema operacional** é um conjunto de um ou mais programas que fornece um **conjunto de serviços**, cria uma **interface** entre aplicações (software) e o hardware, assim como **aloca e gerencia** recursos compartilhados entre múltiplos programas em execução (processos). Além disto, é uma interface entre os usuários e os recursos de hardware.



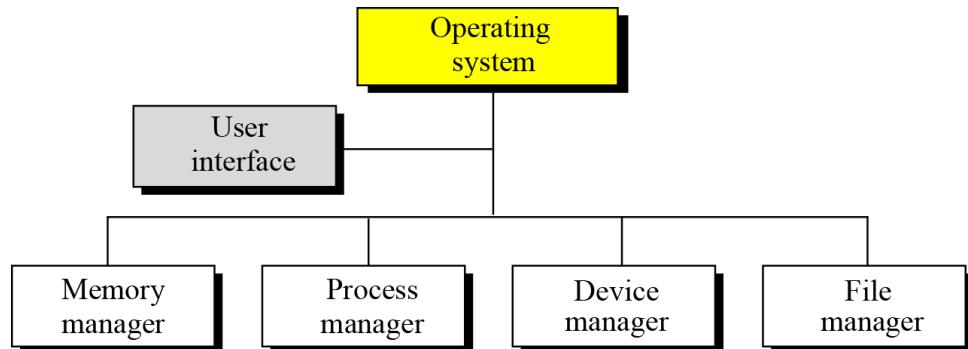
EXERCÍCIO TUTORIADO

Na figura abaixo, temos um usuário realizando a tarefa de pressionar uma tecla de um teclado. Descreva, brevemente, o funcionamento da execução desta tarefa, evidenciando as ações efetuadas pelo sistema operacional.



PRINCIPAIS COMPONENTES DE UM SISTEMA OPERACIONAL

- O núcleo (kernel) de um sistema operacional, geralmente, é formado por quatro módulos de gerenciamento (managers):



- Cada um destes gerenciadores é responsável por um tipo de objeto: **gerenciador de memória** (memória), **gerenciador de processos** (programas em execução), **gerenciador de dispositivos** (teclados, vídeo, discos,...) e **gerenciador de arquivos** (dados e pastas armazenados em memória secundária como, por exemplo, arquivos em discos).
- A interface com o usuário pode ser feita através de um **terminal simples (shell)** ou através de **sistemas de janelas**:

```
login as: tmp
tmp@192.168.1.1's password:
Last login: Mon Nov 13 23:48:43 2006 from 192.168.1.2
***** Welcome to devnull. *****
Hello tmp ! 

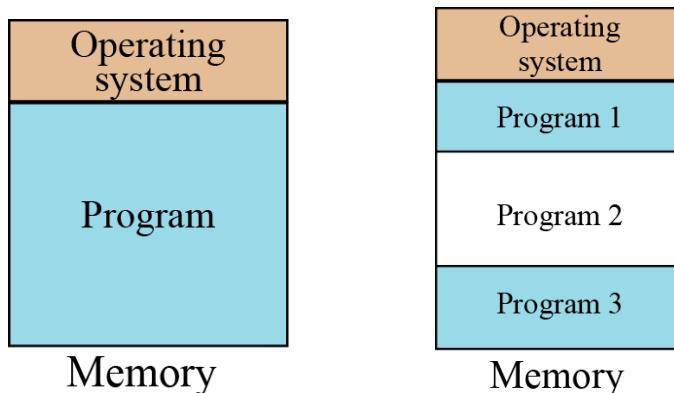
Today is: Mon Nov 13 23:50:47 EET 2006
Last login: Mon 13 Nov 2006 at 23:50:46 from 192.168.1.2
Loading system information ... done
Distro: Fedora Core release 6 (Zod)
Kernel: Linux 2.6.18-1.2798.fc6_i686
CPU: AMD Athlon(tm) XP 1600+
Speed: 1400.090 MHz
Load: 0.00, 0.00, 0.00
RAM: 515164 MB
Usage: 4.31700 %
IP:
Uptime: 1 day, 22 hours, 58 minutes
***** Enjoy your stay! *****

[tmp:devnull][~]$ ls
total 36K
4.0K .bash_history 4.0K .bash_profile 4.0K .emacs 4.0K test/ 4.0K .zshrc
4.0K .bash_logout 8.0K .bashrc 4.0K .kde/ 0 work.txt
[tmp:devnull][~]$
```

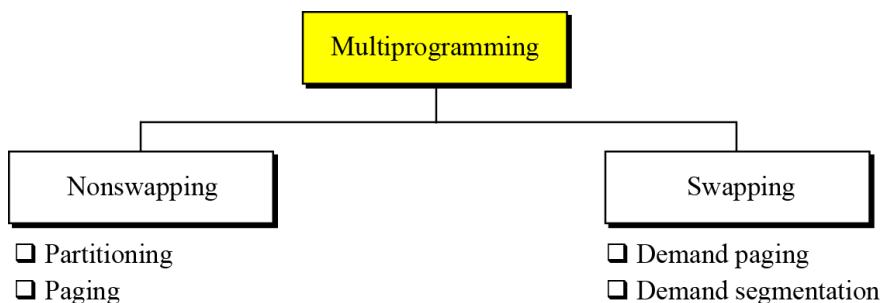


GERENCIADOR DE MEMÓRIA

- O gerenciador de memória de um SO é responsável pela alocação de espaços de memória aos programas (e seus dados) que serão executados pelo processador. Além da alocação, o gerenciador de memória é responsável de carga de programas e dados do disco para a memória.
- Em relação ao gerenciamento de memória, um SO é classificado em dois tipos: com **monoprogramação** ou com **multiprogramação**.



- Num **SO monoprogramado**, somente **um programa** pode estar carregado na memória para execução. Já num **SO multiprogramado**, podemos ter **vários programas** carregados na memória para execução.
- Grande parte dos SO's modernos (Windows, Linux, MacOS, UNIX,...) são multiprogramados. Existem diversas maneiras de se implementar multiprogramação em um SO, conforme mostrado na figura abaixo:



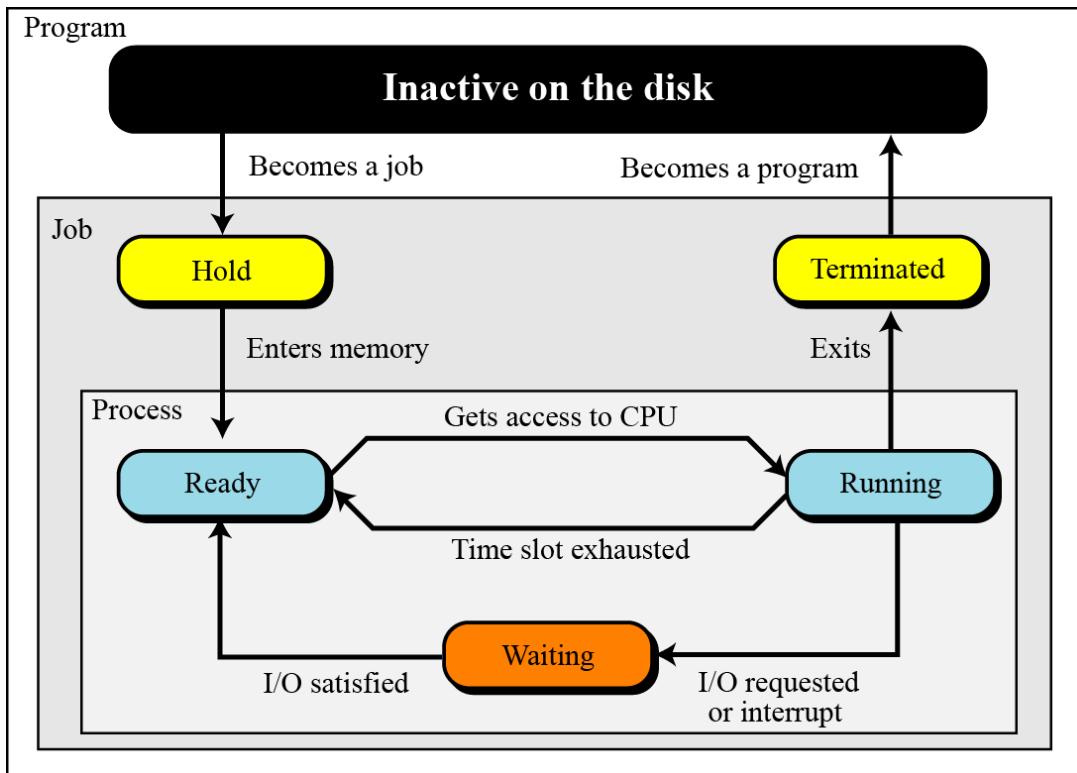
- A operação de **swap** (troca) permite utilizar uma partição especial do disco chamada **participação de swap** para realizar trocas de blocos da memória principal com o disco. Normalmente, recomenda-se que o tamanho da área de swap seja, no mínimo, duas vezes o tamanho da memória principal.

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Suponha que você tenha uma memória principal de 8 GB. Qual seria o tamanho mínimo recomendável para a partição de swap no seu disco ?

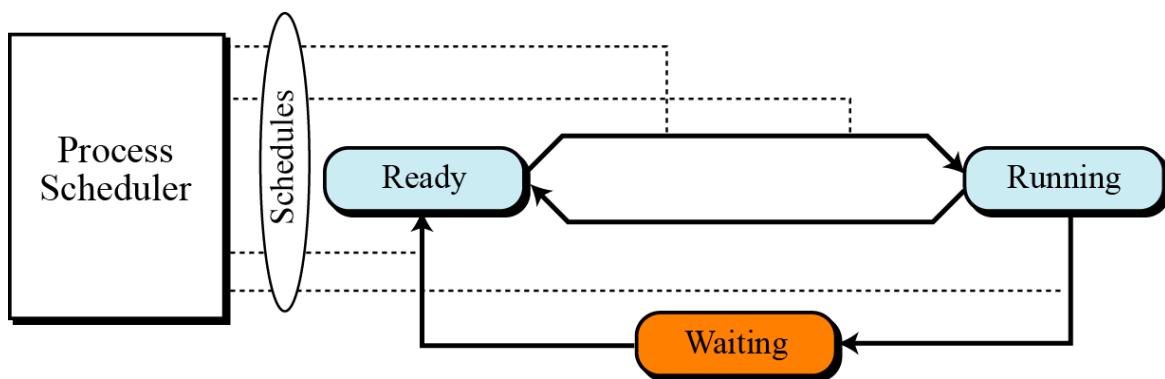
GERENCIADOR DE PROCESSOS

- Um gerenciador de processos controla a execução de programas. Quando temos um programa em execução, dizemos que temos um **processo** no SO. Para se transformar em processo, o programa passa por uma fase intermediária chamada **tarefa (job)**, quando ele deve ser preparado para executar (carga da memória) ou terminar a execução.

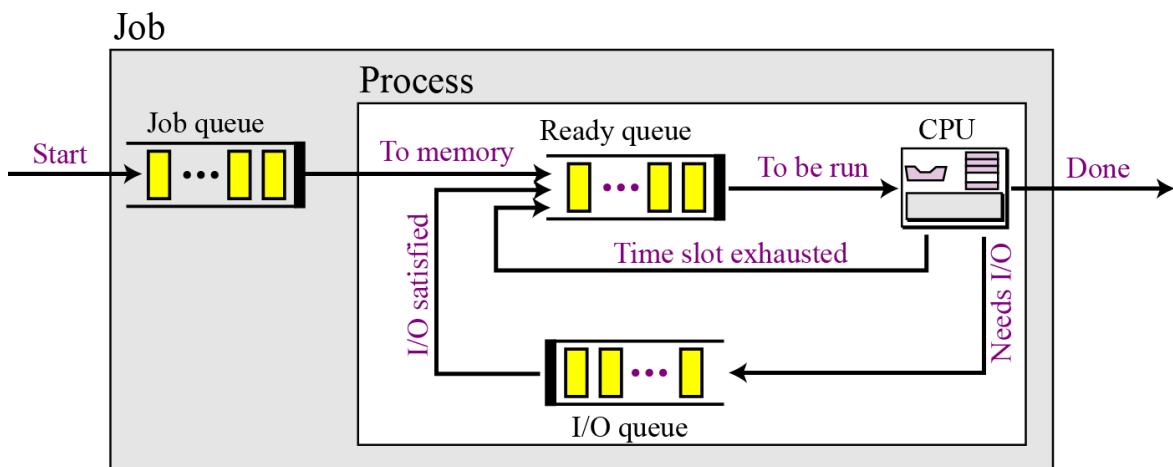


- Um processo pode estar em um dos seguintes estados:
 - pronto (ready):** está aguardando a CPU estar livre para ser executado
 - executando ou rodando (running):** está efetivamente usando a CPU
 - em espera (waiting):** está esperando por algum dispositivo de E/S (exemplo, esperando um usuário digitar um número no teclado)
- Um processo, quando ganha a CPU para execução, possui um tempo máximo de execução chamado **fatia de tempo (time slice)**. A fatia de tempo varia, geralmente, de 10 ms a 100 ms em vários SOs.
- Quando termina a fatia de tempo do processo, ele precisa deixar a CPU para outro processo possa usá-la.

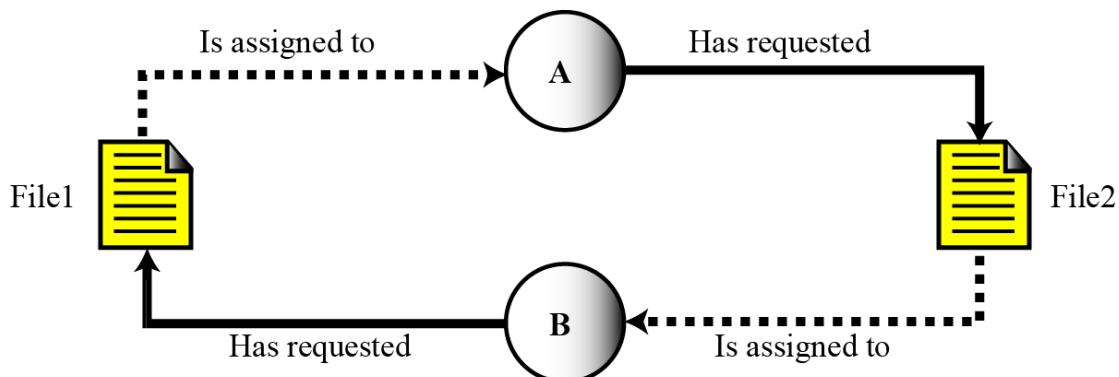
- A decisão de qual processo irá usar a CPU depende do **escalonador (scheduler)**. Existem diversas técnicas de escalonamento: FIFO, Round Robin, Prioridades,



- O controle de processos prontos e em espera é feito através de filas de processos.



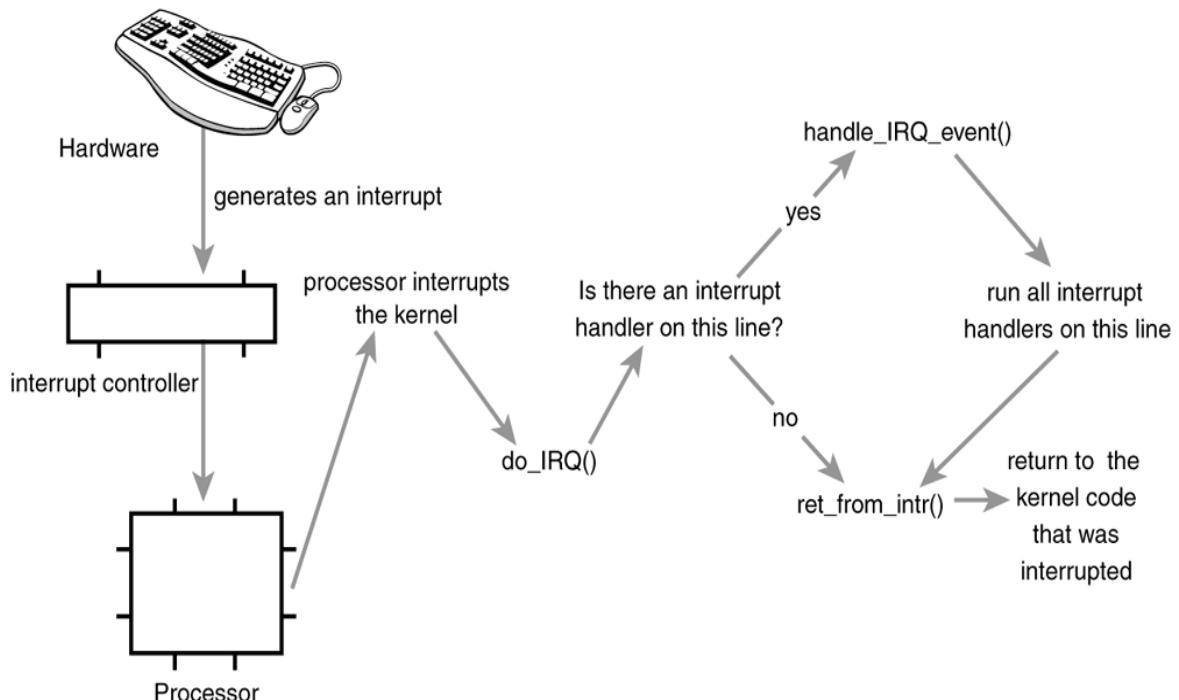
- Durante a execução de um processo, ele pode travar. Esta situação pode ser resultante de uma situação de **deadlock**, como o exemplo mostrado abaixo:



- Nesta situação, a única maneira de resolver o impasse é “matando” um dos processos que está em deadlock.

GERENCIADOR DE DISPOSITIVOS

- Cada um destes gerenciadores é responsável por um tipo de objeto: **gerenciador de memória** (memória), **gerenciador de processos** (programas em execução), **gerenciador de dispositivos** (teclados, vídeo, discos,...) e **gerenciador de arquivos** (dados e pastas armazenados em memória secundária como, por exemplo, arquivos em discos).
- O **gerenciador de dispositivos (ou gerenciador de E/S)**, é responsável pelo acesso a dispositivos de entrada/saída.
- O acesso a um dispositivo envolve:
 - operações realizadas no próprio hardware
 - auxílio de um **device driver**, já no nível de software e controlado pelo sistema operacional
- Embora existam várias técnicas de E/S, a mais comum é o mecanismo de **interrupção**, cujo esquema de funcionamento é mostrado abaixo:

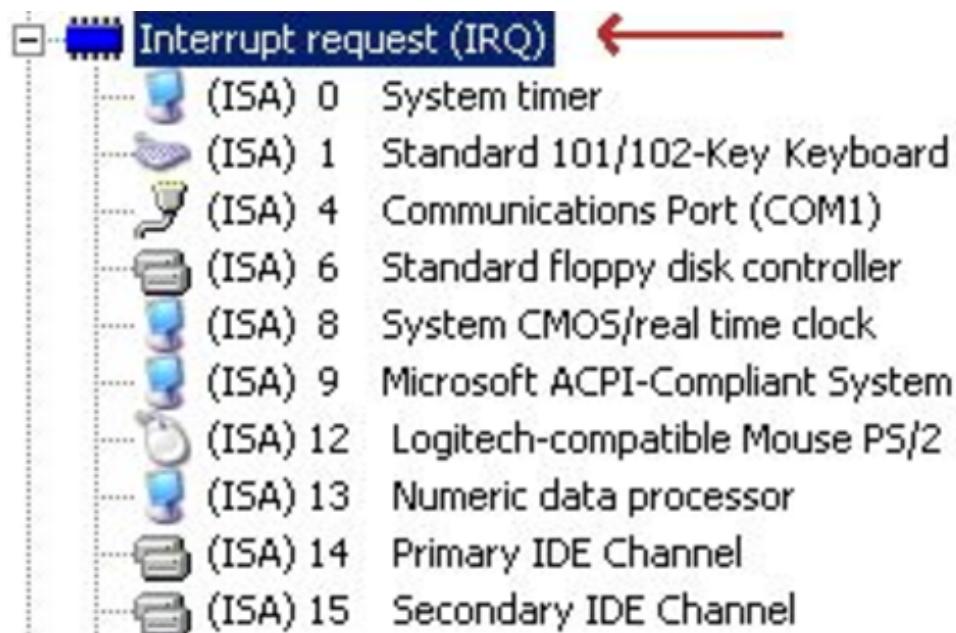


- No esquema acima, ocorre uma operação de entrada. Ao apertar uma tecla do teclado, geramos uma interrupção, que é capturada por um processador especializada chamado **controlador de interrupção** (*interrupt controller*). Este processador notifica a CPU de que há uma requisição de interrupção e, a CPU, envia esta solicitação para o gerenciador de dispositivos do SO.
- Estas requisições são chamadas de **IRQ** (*Interrupt ReQuest*). Cada dispositivo ou grupo de dispositivos possui um IRQ diferente para que o SO saiba quem está tentando se comunicar (ou com quem vai se comunicar).

- Há tabelas de IRQ padronizadas, como a mostrada abaixo:

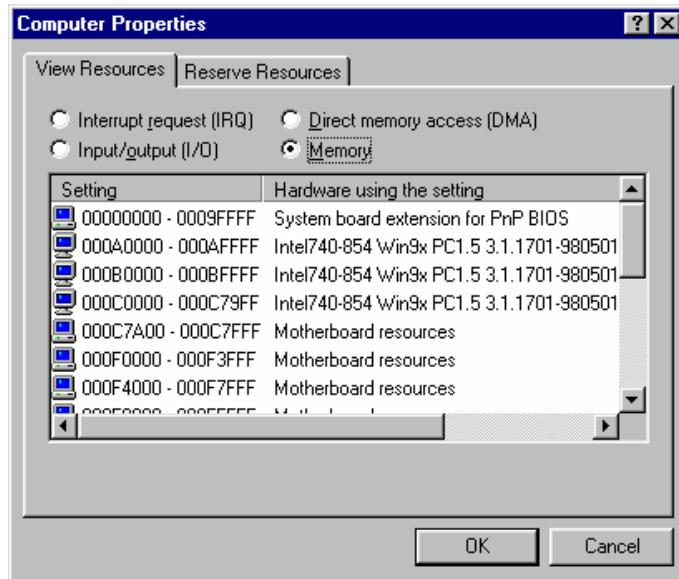
IRQ	Device
0	System Timer
1	Keyboard Controller
2	Second IRQ Controller
3	COM 2
4	COM 1
5	Sound Circuit
6	Floppy Drive
7	Parallel Port
8	Real-time Clock
9	Available
10	Available
11	Available
12	Mouse Port
13	Math Coprocessor
14	Primary Hard Drive
15	Secondary Hard Drive

- Quando um SO está rodando, é possível também se consultar as IRQs. No exemplo abaixo, temos algumas IRQs para alguns dispositivos:

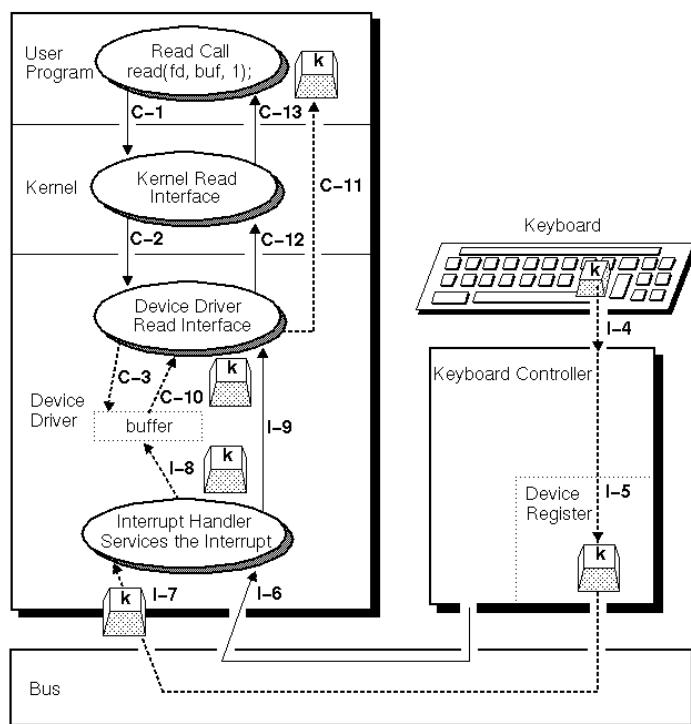


- Observem que o mapeamento da IRQs no SO segue a tabela padronizada mostrada anteriormente.

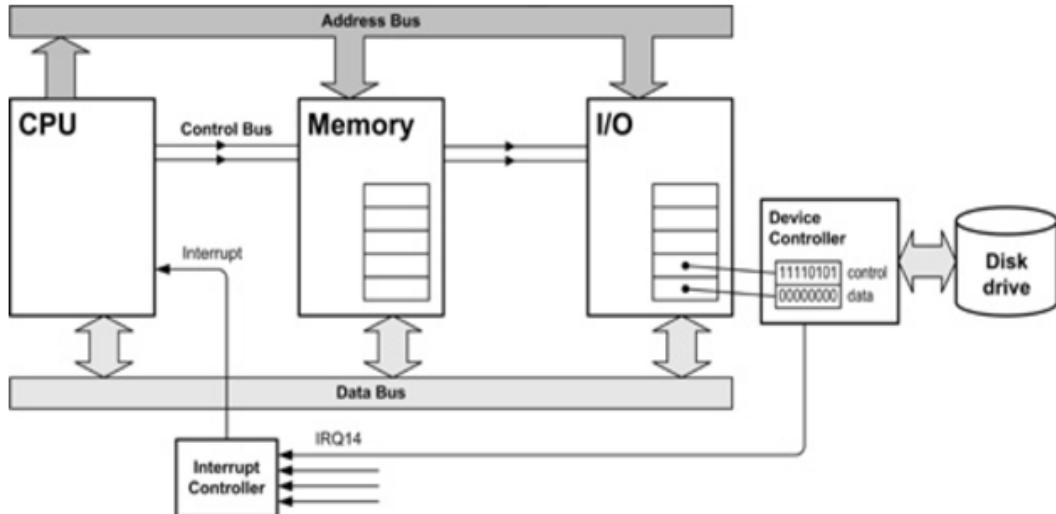
- Associada a uma IRQ, normalmente temos:
 - uma região de memória usada para comunicação (**buffer**) com o dispositivo. Esta área pode variar em função do tipo do dispositivo.



- um programa de controle de acesso ao dispositivo, chamado **device driver**. Os device drivers, geralmente, são fornecidos pelo fabricante do dispositivo e contém as instruções de inicialização, controle, leitura e escrita no dispositivo. Device drivers são programados, geralmente, em linguagem C (médio nível) ou em Assembly. A figura abaixo ilustra o funcionamento do device driver com o restante do SO e hardware num comando do tipo `c=input()`.



- A comunicação da CPU não é feita, geralmente, diretamente com o dispositivo. Há uma elemento intermediário, chamado **controladora de dispositivo** (**device controller**).
- As controladoras possuem memórias locais de comunicação com o dispositivo e, também, transformam as solicitações que vieram do SO, passaram pela CPU, em ações no dispositivo. Por exemplo, ler um arquivo armazenado em disco rígido.



- Abaixo, temos um exemplo de controladora ligada a um disco rígido.

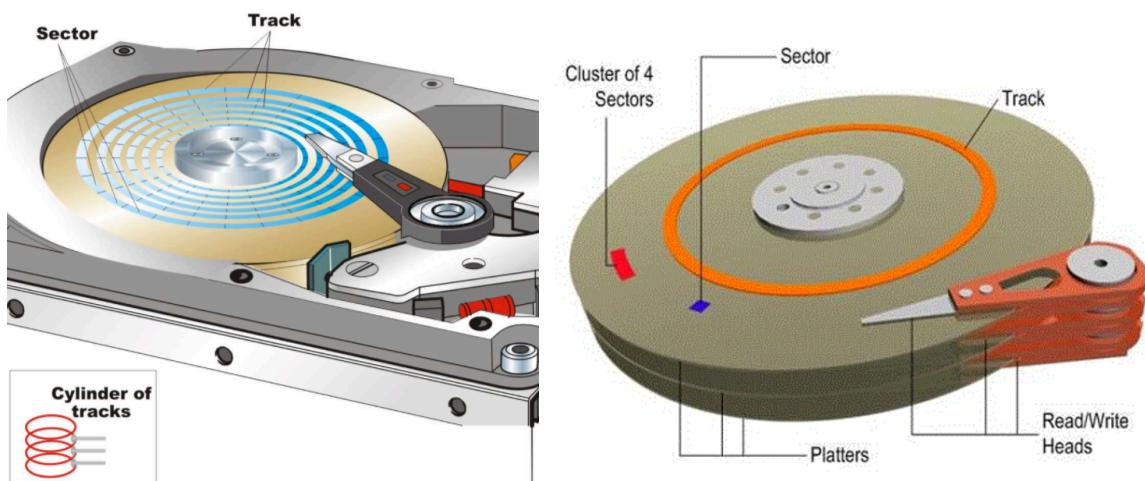


EXERCÍCIO COM DISCUSSÃO EM DUPLAS

O que pode ocorrer quando instalamos um **device driver genérico** ou **não compatível** com um determinado dispositivo ?

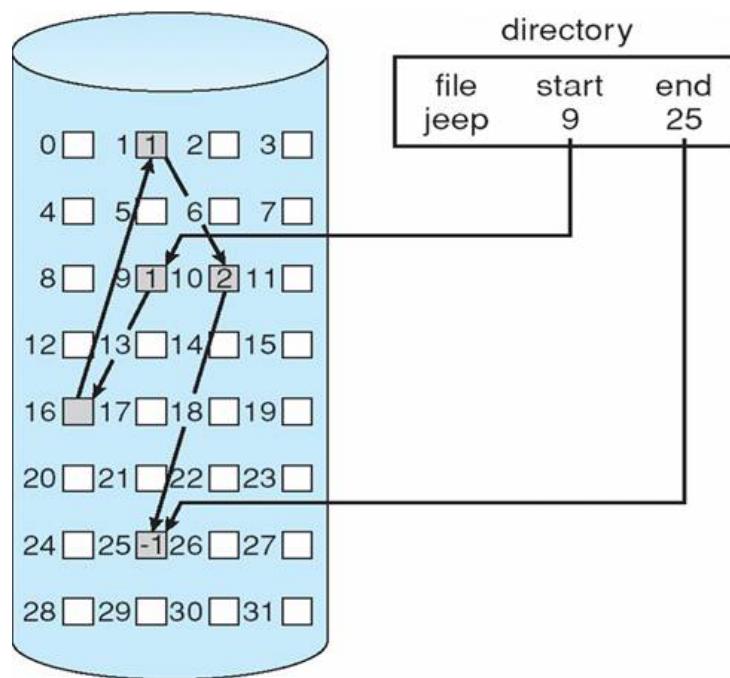
GERENCIADOR DE ARQUIVOS

- O gerenciador de arquivos é responsável pela criação, leitura, escrita e remoção de arquivos/diretórios em **dispositivos de memória secundária** (discos rígidos, pendrives, CD/DVD,...). Estes dispositivos também são conhecidos como dispositivos de armazenamento (storage devices).
- Um disco rígido, por exemplo, pode ser descomposto em **setores** (sectors), **trilhas** (tracks) e **cilindros** (cylinders). Um setor é uma região do disco onde armazenamos os dados. Um tamanho típico de setor é 512 bytes. O conjunto de setores num caminho circular é chamado de trilha. Assim, trilhas são formadas de setores. Pequenos conjuntos de setores também podem formar **aglomerados** (clusters). Finalmente, trilhas de mesmo raio formam um cilindro, em discos que possuem múltiplos pratos de armazenamento.

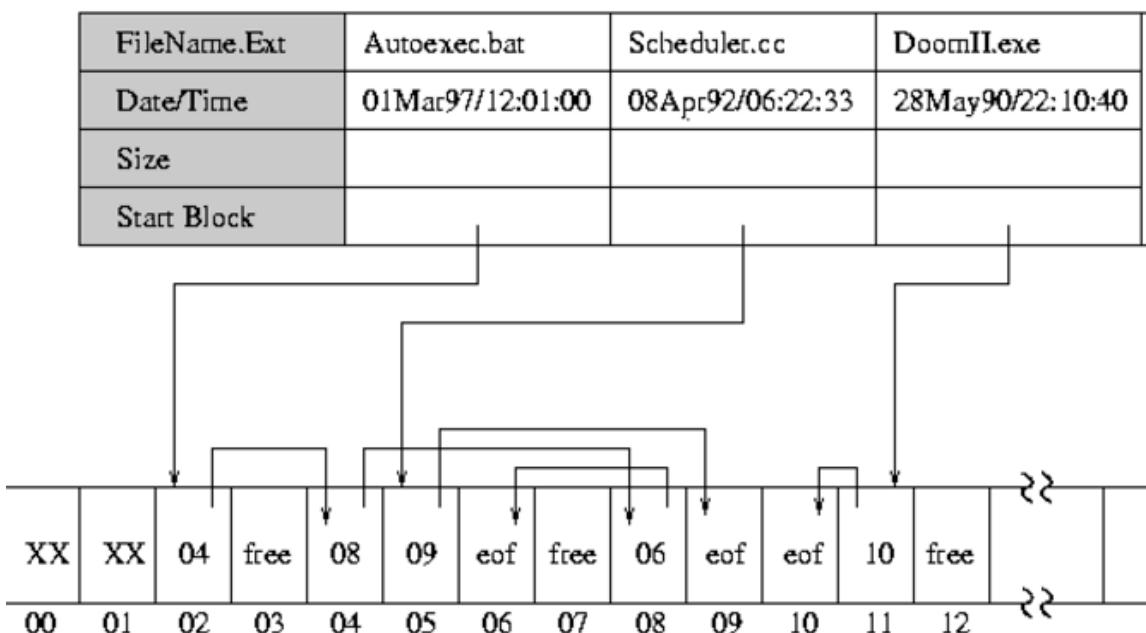


- Antes de realizarmos o armazenamento, o dispositivo de armazenamento precisa ser **formatado**. Na formatação, criamos um mapa de cilindros, trilhas e setores, que são armazenados numa tabela do disco. Dependendo do tipo de sistema operacional, esta tabela pode mudar (exercício extra-classe).

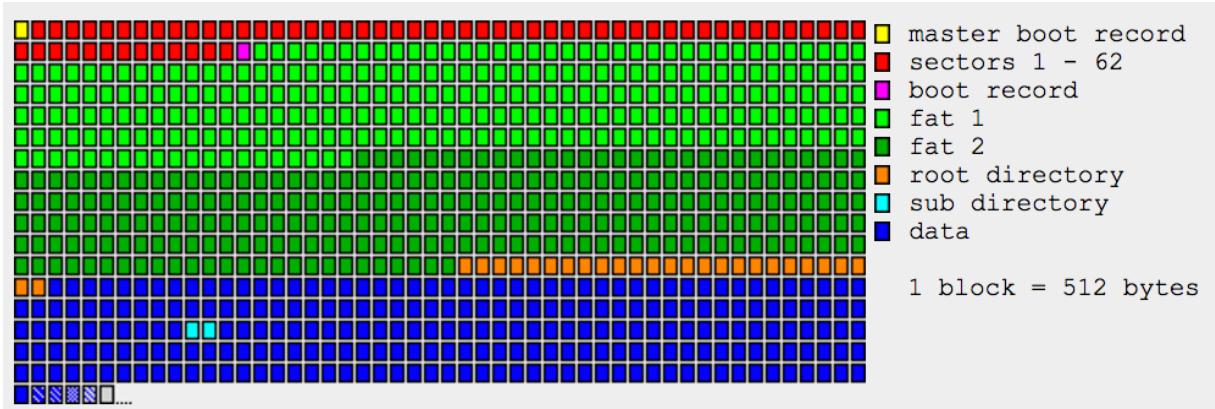
- Para o SO, um arquivo armazenado no disco é formado por **blocos** (blocks). Um bloco possui um endereço no disco (cilindro, trilha e setor(es)). Genericamente, um arquivo é armazenado como mostrado abaixo:



- Além de conter o nome do conjunto de blocos (nome do arquivo) e a localização dos blocos do arquivo, a tabela de arquivos também pode armazenar as propriedades do arquivo (data da criação/modificação, tamanho, permissões de acesso,...). Abaixo, temos parte da tabela FAT, usada pelos SOs DOS e Windows. Neste tipo de tabela, o final de arquivo é marcado com EOF (End Of File), ao invés de armazenar o bloco final.



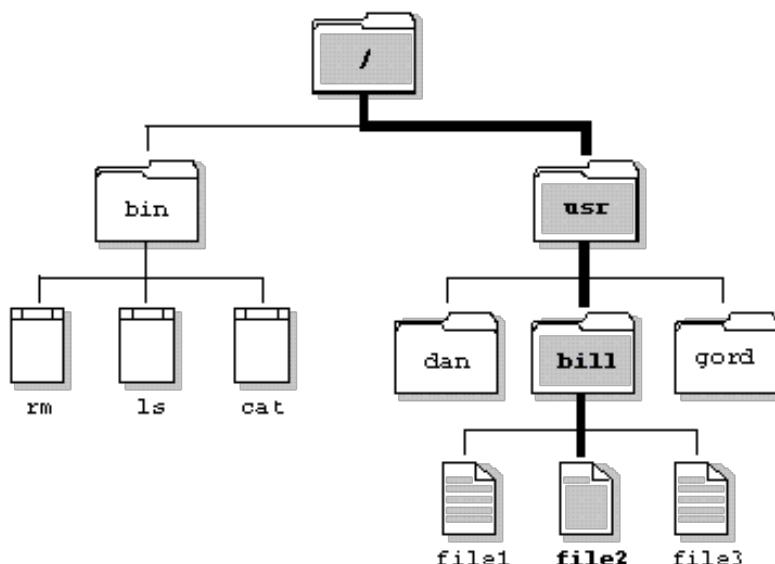
- A FAT, por exemplo, é armazenada logo no início do disco. Outras tabelas de arquivos podem ser armazenadas em outros pontos do disco (meio, final), para otimizar o acesso. Abaixo, temos um exemplo de localização da FAT e arquivos alocados em um disco.



- Se visualizarmos a FAT e arquivos numa ferramenta de visualização binária, teríamos algo como mostrado abaixo:

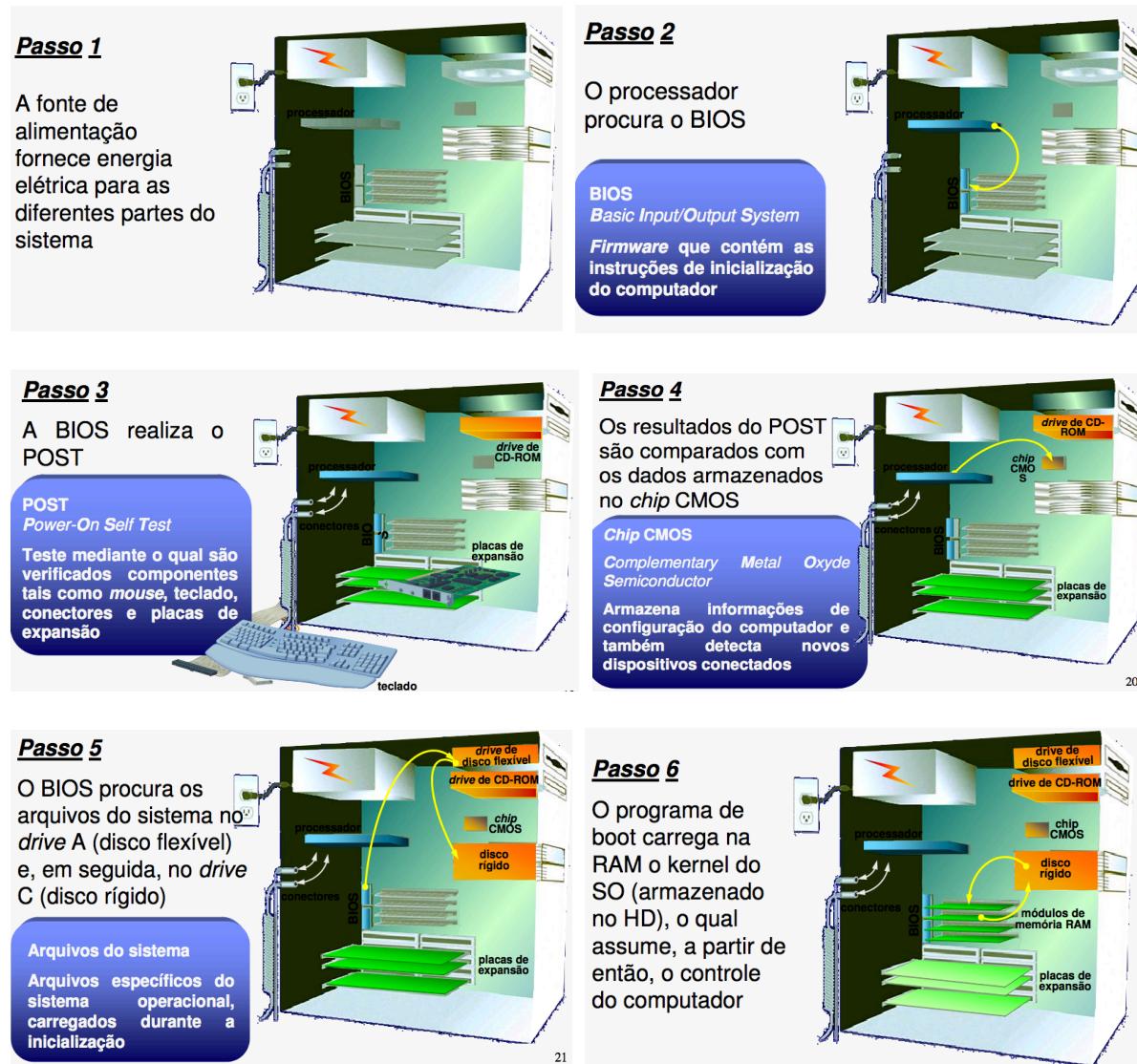
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
EB	3E	90	4D	53	57	49	4E	34	2E	31	00	02	04	01	00
02	00	02	00	00	F8	00	01	3F	00	F0	00	60	75	0C	00
10	EC	03	00	80	00	29	1E	3C	D9	19	4E	4F	20	4E	41
4D	45	20	20	20	20	46	41	54	31	36	20	20	20	F1	7D
FA	33	C9	8E	D1	BC	FC	7B	16	07	BD	78	00	C5	76	00
1E	56	16	55	BF	22	05	89	7E	00	89	4E	02	B1	0B	FC
F3	A4	06	1F	BD	00	7C	C6	45	FE	0F	8B	46	18	88	45
F9	FB	38	66	24	7C	04	CD	13	72	3C	8A	46	10	98	F7
66	16	03	46	1C	13	56	1E	03	46	0E	13	D1	50	52	89
46	FC	89	56	FE	B8	20	00	8B	76	11	F7	E6	8B	5E	0B
03	C3	48	F7	F3	01	46	FC	11	4E	FE	5A	58	BB	00	07
8B	FB	B1	01	E8	94	00	72	47	38	2D	74	19	B1	0B	56
8B	76	3E	F3	A6	5E	74	4A	4E	74	0B	03	F9	83	C7	15
3B	FB	72	E5	EB	D7	2B	C9	B8	D8	7D	87	46	3E	3C	D8

- Um **diretório** já é uma estrutura com um nível mais alto de abstração que um arquivo. Em vários sistemas operacionais, ele construído usando um tipo abstrato de dados (TAD) chamado **árvore** (tree). Por isto, normalmente podemos falar de um **diretório-raiz**, que o diretório que contém todos os outros diretórios e arquivos. A figura abaixo mostra um exemplo de estrutura de arquivos e diretórios no Linux:



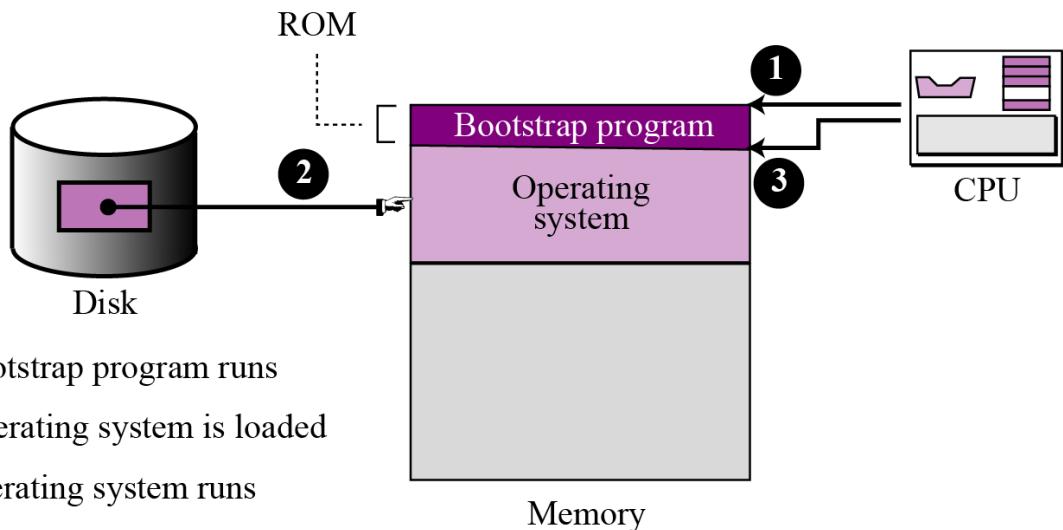
PROCESSO DE BOOT(STRAPPING)

- Uma das tarefas de um SO é carregar programas (e dados) na memória principal para execução. Como o SO também é um programa, quem o carrega na memória ?
- A carga de um SO, inicialmente, envolve diversos passos:



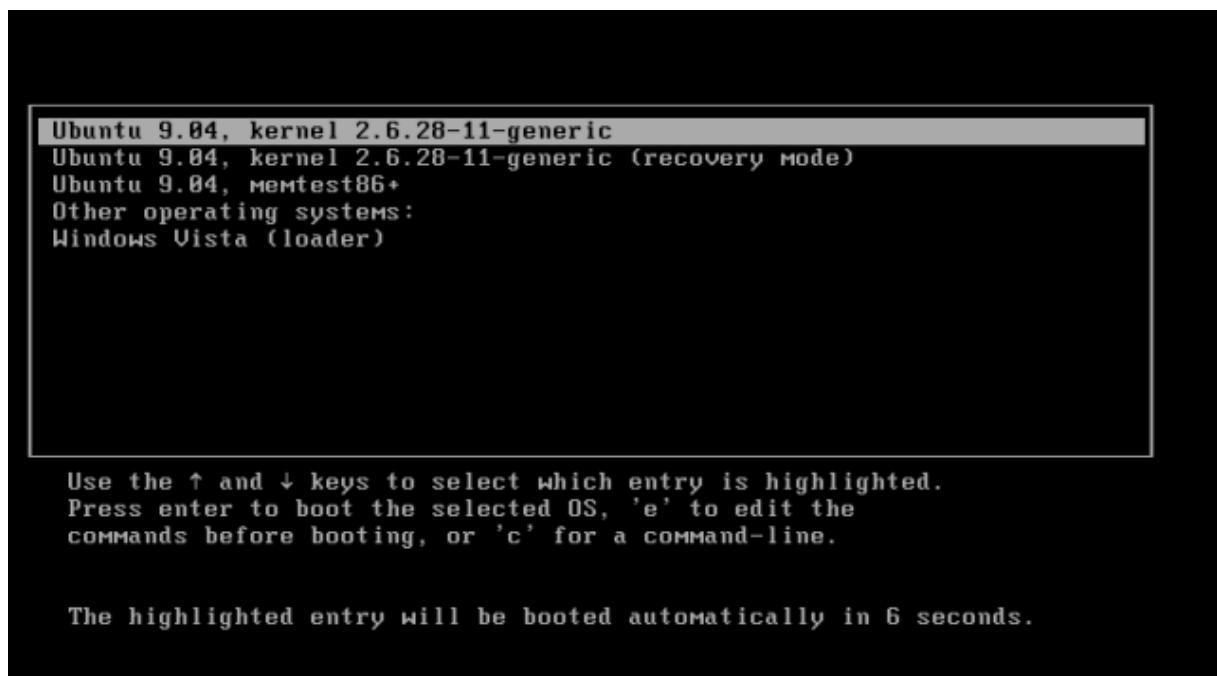
- O programa de boot também é conhecido como **bootstrap**, é um pequeno programa carregado na memória principal numa região somente de leitura (**ROM-Read Only Memory**).

- A única função do bootstrap é carregar o SO, usando os passos mostrados abaixo:



EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Atualmente, é muito comum termos mais de um SO instalado num computador. Por exemplo, conforme mostrado abaixo, temos um computador com Linux (Ubuntu) e Windows (Vista) instalados:



Como funciona o processo de boot em um computador como este ?

EXERCÍCIO EXTRA-CLASSE

1. Descreva a organização básica dos principais sistemas operacionais de mercado (UNIX, LINUX, WINDOWS, MacOS, ANDROID e iOS) e faça um resumo de como funciona seu processo de boot.