



Aspectos de Dados em Sistemas Operacionais

Visão Geral

Engenharia de Computação

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis M. L. Martins



Objetivos de Aprendizado

Ao final desta aula, você será capaz de:

Compreender abstrações de memória e arquivos em SO

Disclaimer

Parte do material apresentado a seguir foi adaptado de:

- [IT Systems – Open Educational Resource](#), produzido por [Jens~Lechtenböger](#); e
- [Open Education Hub – Operating Systems](#)

Imagens decorativas retiradas de [Unsplash](#)



Critérios para o SO

- Recuperar e armazenar o mais rápido possível
- Uso ótimo do espaço de memória
 - Quando os dados não estão sendo usados, a memória é liberada imediatamente
 - Minimizar o tempo em que a memória está reservada, mas não utilizada
 - Os dados devem ocupar o menor espaço necessário
- Segurança
 - Correção dos dados
 - Isolamento dos dados



Uma Perspectiva do Programador sobre os Dados

- Dados = variáveis
- Operações: declarar/ler/escrever
- Variáveis são armazenadas na memória, portanto, dependendo da linguagem, você também pode:
 - Alocar memória
 - Desalocar memória

Exemplo

```
class Pokemon:
    def __init__(self, id, name):
        self.id = id
        self.name = name

    def __repr__(self):
        return "[id: " + str(self.id) + ", name: " + self.name + "]"

    def __del__(self):
        print("destroying " + self.name)

pokedex = []
pokedex.append(Pokemon(1, "Pikachu"))
pokedex.append(Pokemon(2, "Charizard"))
# ...
print(pokedex)
```

Desempenho: Depende de...

- Número de cópias de memória

```
for i in range(1000000): print(pokemon[i])
```

- Grau de reutilização da memória: Lembre de *Thread Pool*
- Número de alocações/desalocações de memória

```
arr->ptr = realloc(arr->ptr, sizeof(int)*(arr->len + 1));  
arr->ptr[arr->len] = elem;  
arr->len++;
```

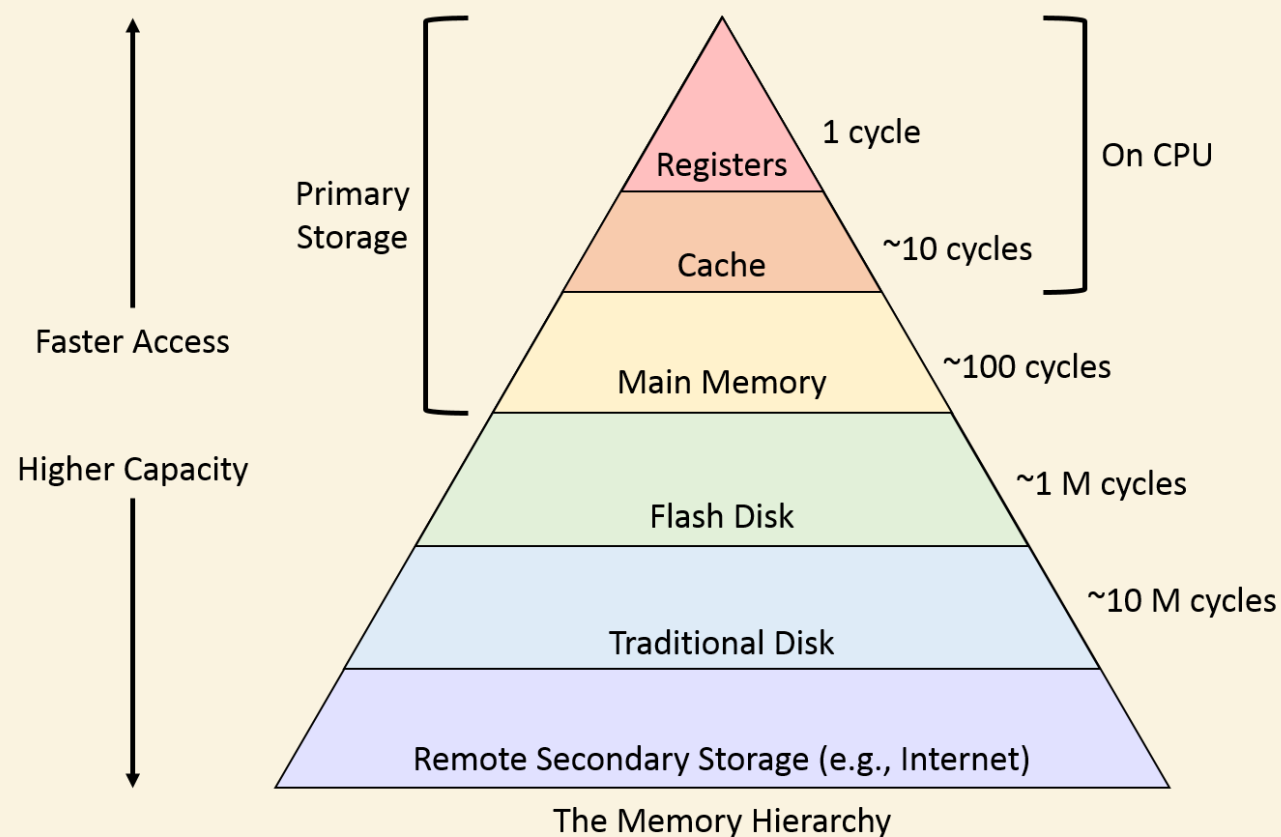
Uso de Espaço: Depende de...

- Como armazenamos os dados
- Uso tipos apropriados para as variáveis
- Quão cedo a memória é liberada

Quem gerencia a memória?

- Você (o programador) - C/C++
- A linguagem de programação - Python, Java
- Uma implementação de biblioteca - C/C++
- O sistema operacional - para todas as linguagens

Hierarquia de memória



Fonte da Imagem: [CS31](#)

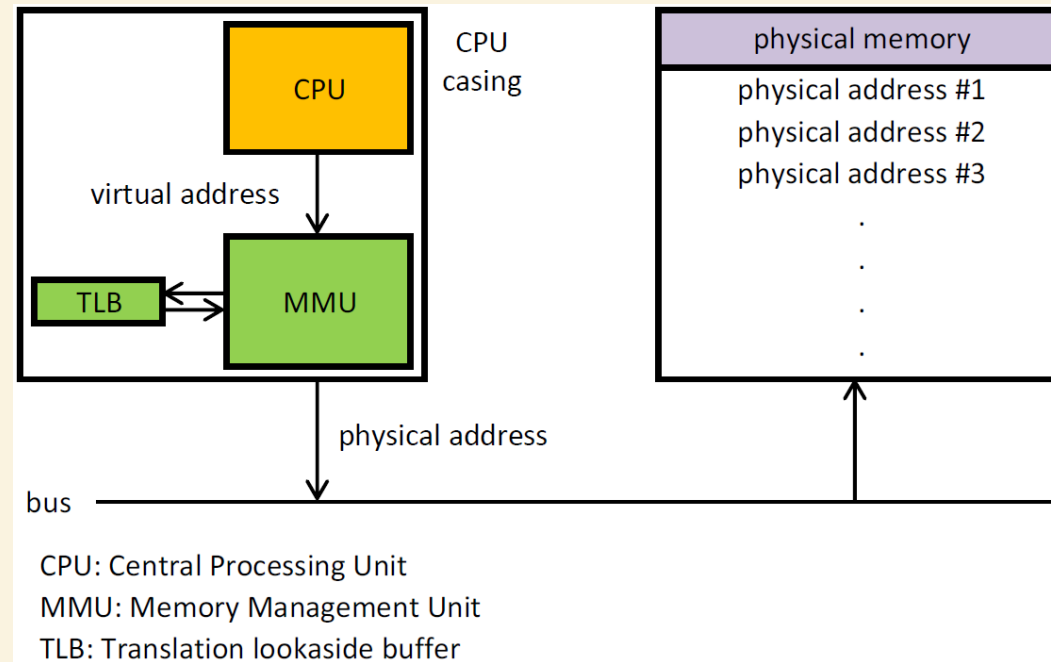
Ver também: [Latency by Collin Scott](#)



Objetivos Principais

- Maximizar a taxa de transferência de memória.
- Maximizar a utilização da memória.
- Garantir a consistência do espaço de endereço.
 - Prover "visão" uniforme de memória para processos
- Fornecer proteção de memória para processos.

Memory Management Unit (MMU)



Fonte da imagem: [Wikipedia](#)

- CPU não acessa diretamente endereços de memória física → Requisita da MMU endereços virtuais
- MMU traduz endereços virtuais em endereços físicos (**extremamente rápida**)
- Kernel envolvido para tarefas complexas (por exemplo, decidir o que remover da memória)

Interface do SO para Memória Virtual

- `/proc/<PID>/mem` - acesso à memória virtual
 - `demo/proc_mem/`
- `/proc/<PID>/page_map` - acesso aos mapeamentos de página
 - `demo/proc_pagemap/`
- `/dev/mem` - acesso à memória física
 - `demo/proc_pagemap/`

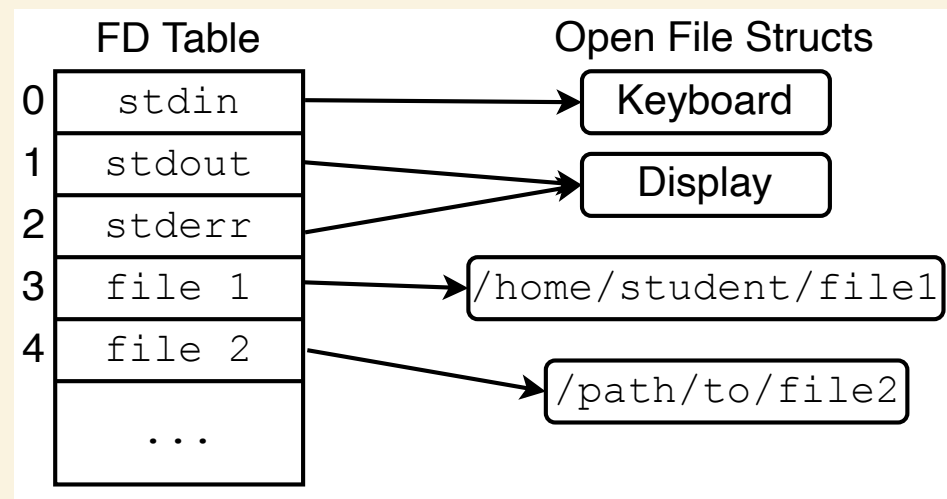


Arquivos

- Arquivos são abstrações comuns do SO para organizar dados e armazená-los de maneira persistente.
 - Persistência: SO deve manter dados mesmo que haja cortes de energia ou falhas no sistema.
- *File Descriptors*: SO representa arquivos via números inteiros chamados *descriptors*
 - Arquivos: *named streams of bytes*
 - Abstração: arquivos, diretórios, dispositivos de I/O, acesso de rede, etc.
- Operações: `open` , `close` , `read` , `write`

Arquivos

- O padrão POSIX descreve 3 descritores (numerados 0, 1, 2) para cada processo:
 - 0 : *Standard input*, `stdin` (e.g., entrada do teclado)
 - 1 : *Standard output*, `stdout` (e.g., imprimir na tela ou no terminal)
 - 2 : *Standard error*, `stderr` (e.g., imprimir mensagem de erro no terminal)



Fonte da Imagem: [OS Team - OS OER](#)

JULIA EVANS
@bork

file descriptors

Unix systems use integers to track open files



these integers are called **file descriptors**

lsof (list open files) will show you a process's open files

```
$ ls -p 4242 ← PID we're interested in
```

FD	NAME
0	/dev/pts/tty1
1	/dev/pts/tty1
2	pipe:29174
3	/home/bork/awesome.txt
5	/tmp/

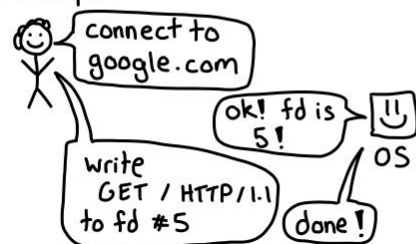
↑
FD is for file descriptor

file descriptors can refer to:

- files on disk
- pipes
- sockets (network connections)
- terminals (like xterm)
- devices (your speaker! /dev/null!)
- LOTS MORE (eventfd, inotify, signalfd, epoll, etc etc)

not EVERYTHING on Unix is a file, but lots of things are

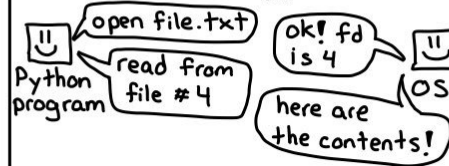
When you read or write to a file/pipe/network connection you do that using a file descriptor



Let's see how some simple Python code works under the hood:

```
Python:  
f = open("file.txt")  
f.read lines()
```

Behind the scenes:



(almost) every process has 3 standard FDs

```
stdin → 0  
stdout → 1  
stderr → 2
```

"read from stdin"
means

"read from the file descriptor 0"

could be a pipe or file or terminal

Descritores de Arquivo em `/proc/<pid>/fd`

Para um processo com ID `<pid>`, o subdiretório `/proc/<pid>/fd` indica seus descritores de arquivo.

- As entradas são links simbólicos que apontam para os destinos reais.
- Use `ls -l` para ver os números e seus destinos, por exemplo:

```
lrwx----- 1 root root 64 Jun 26 15:34 0 -> /dev/pts/3
lrwx----- 1 root root 64 Jun 26 15:34 1 -> /dev/pts/3
lrwx----- 1 root root 64 Jun 26 15:34 2 -> /dev/pts/3
lr-x----- 1 root root 64 Jun 26 15:34 3 -> /dev/tty
lr-x----- 1 root root 64 Jun 26 15:34 4 -> /etc/passwd
```

Permissões de Acesso

- Quem tem permissão para fazer o quê?
- O sistema controla o acesso a objetos por sujeitos.
- **Objeto:** qualquer coisa que precise ser protegida: por exemplo, uma região de memória, um arquivo, um serviço.
 - Com operações diferentes dependendo do tipo de objeto.
- **Sujeito:** entidade ativa que utiliza os objetos, ou seja, um processo.
 - Threads dentro de um processo compartilham as mesmas permissões de acesso.
 - O sujeito pode também ser o próprio objeto, por exemplo, terminar uma thread ou um processo.

JULIA EVANS
@bork

unix permissions

drawings.jvns.ca

There are 3 things you
can do to a file

↓ read ↓ write ↓ execute

ls -l file.txt shows you permissions
Here's how to interpret the output:

rw- rw- r-- bork staff
↑ ↑ ↑
bork (user) staff (group) ANYONE
can can can
read & write read & write read

File permissions are 12 bits

setuid setgid
↓ ↓
000 110 110 100
sticky rwx rwx rwx
user group all

For files:

r = can read
w = can write
x = can execute

For directories it's approximately:

r = can list files
w = can create files
x = can cd into & modify files

110 in binary is 6

So rw- r-- r--
= 110 100 100
= 6 4 4

chmod 644 file.txt
means change the
permissions to:

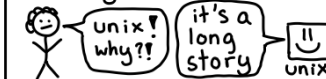
rw- r-- r--
simple!

setuid affects
executables

\$ls -l /bin/ping

rwS r-x r-x root root
↑
this means ping always
runs as root

setgid does 3 different
unrelated things for
executables, directories,
and regular files



Fonte da Imagem: [Julia Evans](#)

Dúvidas e Discussão
