</> Abstração em Gerência de Arquivos

Exemplo aplicado: Prontuários Hospitalares

👥 Autores

Nome	RGM
Camilla Macedo Alves	43713548
Gabriel Oliveira de Souza	43319327
João Gabriel Rocha Cerqueira	43912672
Virgínia Mayumi Furushima de Freitas	42114331

Tema: Abstração em Gerência de Arquivos
Aplicação: Prontuários Hospitalares

📅 **Data da Apresentação:** 13/Nov/2025

Curso: Ciência da Computação
Disciplina: Sistemas Operacionais

Objetivo da Apresentação

Apresentar cada um dos níveis de funcionamento do sistema através das seguintes flags:

- 🔴 **Baixo Nível:** Revisar operações que ocorrem por baixo do SO.
- 🟡 **Abstração SO:** Explicar abstrações de sistemas feitos pelo Sistema Operacional.
- 🟢 **Abstração Programática:** Apresentar uma segunda camada de abstração por cima do SO através da programação.

Tópicos Chave

1. Abstração
2. Estrutura e Diretórios
3. Permissões
4. Mecanismos de Acesso
5. Armazenamento

Caso prático: Prontuários Hospitalares

0 - Conceito de Abstração

Antes de falar sobre os meios de abstração, é importante entender o que é a abstração na computação.

O que é?

Abstração na computação é o processo de simplificar sistemas complexos.

- Ocultar detalhes de implementação e expor apenas as funcionalidades essenciais.
- Permitir que usuários e desenvolvedores trabalhem com conceitos de alto nível, facilitando o entendimento, reutilização e manutenção do sistema.
- Baixa curva de aprendizado, sem a necessidade de conhecer todos os detalhes internos de como algo funciona para conseguir usufruir.

Como é feito?

- 🟡 Pelo Sistema Operacional (SO):
  - Expõe chamadas simples (open, read, write) no lugar de comandos de hardware.
  - Usa drivers para traduzir pedidos genéricos para cada dispositivo físico.
  - Unifica discos e partições em um único sistema de arquivos.
  - Isola programas com processos/threads e memória virtual.
  - Aplica permissões e políticas de segurança sem que o app conheça detalhes do hardware.
- 🟢 Por Aplicativos (camada acima do SO):
  - Agrupam várias chamadas do SO em funções de alto nível (ex.: “Gerar Relatório”).
  - Usam bibliotecas/frameworks para esconder protocolos, formatos e rede.
  - Trabalham com modelos do domínio (Paciente, Consulta) em vez de pastas/arquivos brutos.
  - Impõem regras de negócio e padronizam nomes/locais de dados.
  - Oferecem UI e APIs próprias para uso simples por pessoas e outros sistemas.

Resumo em uma frase: o SO padroniza o acesso ao hardware; os apps padronizam o acesso ao problema do usuário.

1 - Estrutura e Diretórios

🔴 Baixo Nível

Os diretórios não ficam abaixo do sistema operacional, e sim dentro dele, na camada do sistema de arquivos, que é quem realmente organiza e controla os dados no disco.

- Os inodes são criados.
- Os blocos de dados são alocados.
- As entradas de diretório (nome → inode) são gravadas no disco.

Essa parte conversa diretamente com o hardware de armazenamento (HD, SSD, pendrive, etc).

🟡 Abstração do SO

- O sistema de arquivos é organizado em forma de árvore.
- O topo da árvore é o diretório raiz (“/”).
- Cada diretório pode conter arquivos e outros diretórios (subdiretórios).
- Os diretórios criam uma estrutura hierárquica que facilita o acesso e organização.
- Um diretório é, internamente, um arquivo especial que armazena nomes e referências para outros arquivos.

Estrutura do diretório raiz `prontuarios/` na visualização do OS:

```
In [6]: list_directory_tree("prontuarios/")
```

```
prontuarios/
├── pacientes_arquivados/
│   ├── P0-Olavo_de_Carvalho/
│   │   ├── consultas/
│   │   │   └── dificuldade_respiratória.pdf
│   │   ├── exames/
│   │   │   └── covid19.pdf
│   │   ├── profile/
│   │   │   └── P0.jpg
│   │   └── tratamentos/
│   │       └── internação_covid.pdf
│   └── P1-Everson_Olhos/
│       ├── cirurgias/
│       │   └── extração_dente.pdf
│       ├── consultas/
│       │   └── dor_de_dente.pdf
│       └── profile/
│           └── P1.jpg
└── pacientes_ativos/
    ├── P3-Maria_Hipotetica/
    │   ├── cirurgias/
    │   │   └── hernia_de_disco.pdf
    │   ├── consultas/
    │   │   └── dor_na_coluna.pdf
    │   ├── exames/
    │   │   └── raiox_coluna.pdf
    │   └── profile/
    │       └── P3.jpg
    └── P4-Allexandre/
        ├── exames/
        │   └── exame_de_prostata.pdf
        └── profile/
            └── P4.jpg
```

🟢 Abstração Programática

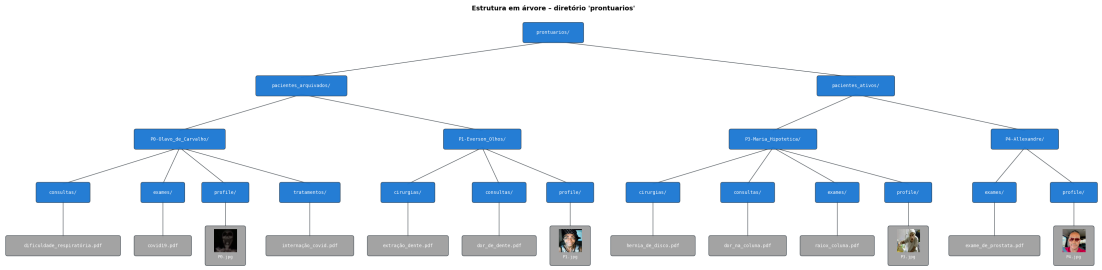
Representação de cores

- 🟦 **Azul:** Diretório / Pasta
- ⬛ **Cinza:** Arquivo

1º Nível Hierárquico

Estrutura completa no diretório raiz `prontuarios/` :

```
In [7]: draw_tree_auto("prontuarios/")
```



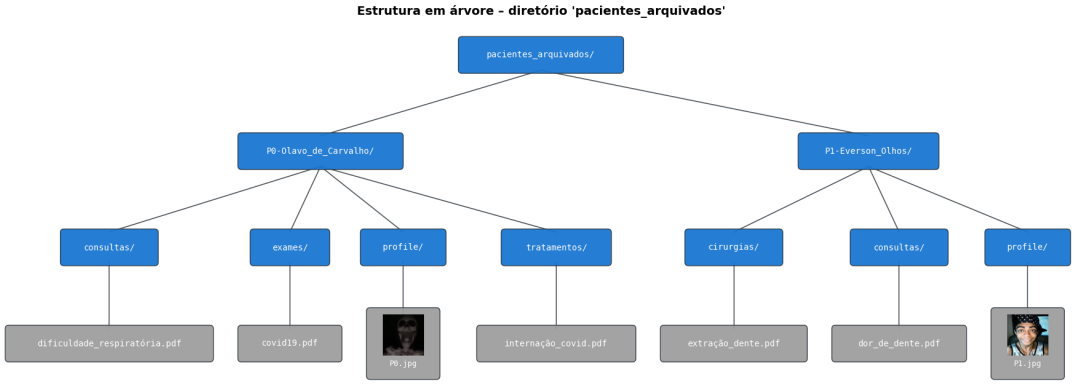
A Hierarquia Nº1 é composta por duas ramificações a partir do diretório raiz `prontuarios/` , sendo elas:

- `pacientes_arquivados/`
- `pacientes_ativos/`

2º Nível Hierárquico

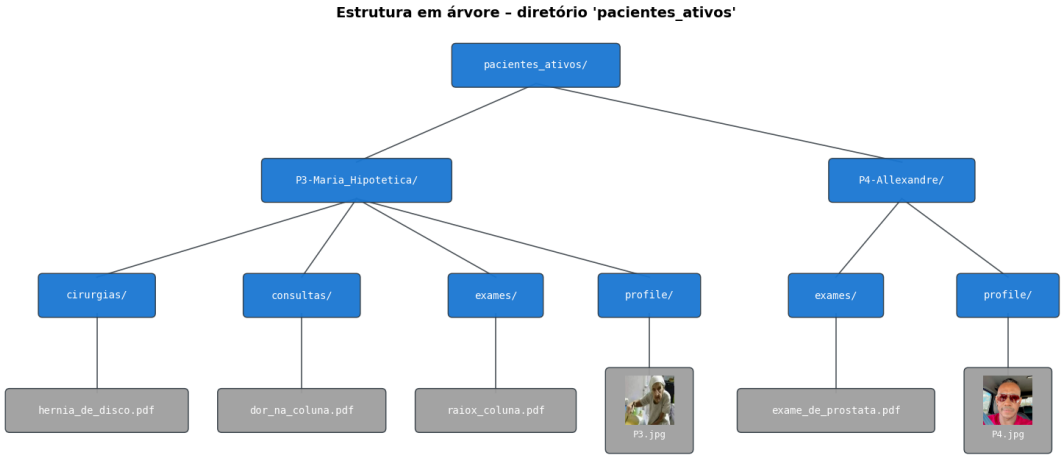
Estrutura no diretório `pacientes_arquivados/` :

```
In [8]: draw_tree_auto("prontuarios/pacientes_arquivados/")
```



Estrutura no diretório `pacientes_ativos/` :

In [9]:
```
draw_tree_auto("prontuarios/pacientes_ativos/")
```



A Hierarquia Nº2 é composta por uma ramificação para cada paciente **ativo** ou **arquivado**.

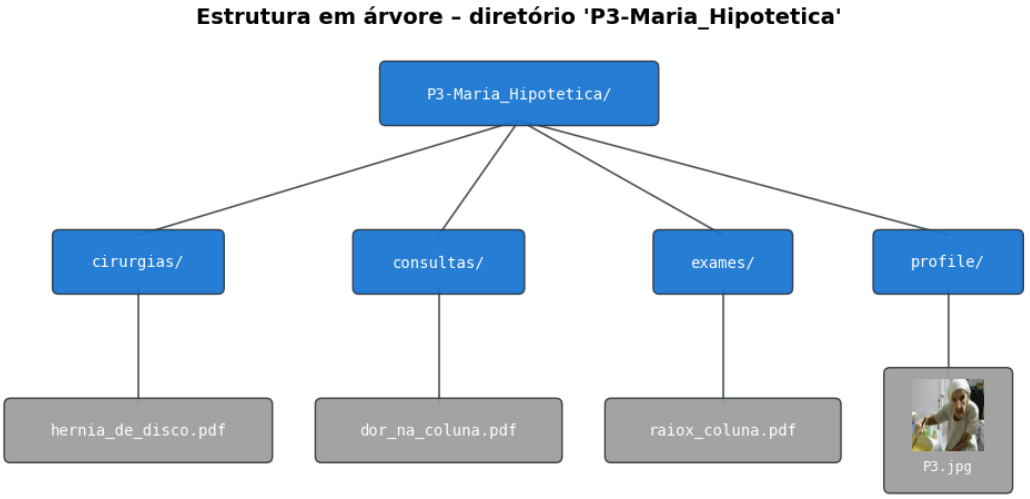
O nome do diretório é composto pelo índice e nome do paciente, no seguinte formato:

```
"P" + index + "-" + name
```

3º Nível Hierárquico

Estrutura no diretório de um paciente:

In [10]:
```
draw_tree_auto("prontuarios/pacientes_ativos/P3-Maria_Hipotetic
a/")
```



A Hierarquia Nº3 é composta por dois tipos de elementos: o subdiretório `profile/` contendo a imagem de identificação do paciente e subdiretórios para diferentes tipos de prontuários relacionados ao mesmo.

Os subdiretórios podem variar de acordo com o ciclo hospitalar do paciente, como:

- `consultas/`
- `exames/`
- `tratamentos/`
- `cirurgias/`
- `laboratoriais/`
- etc...

---

2 - Permissões

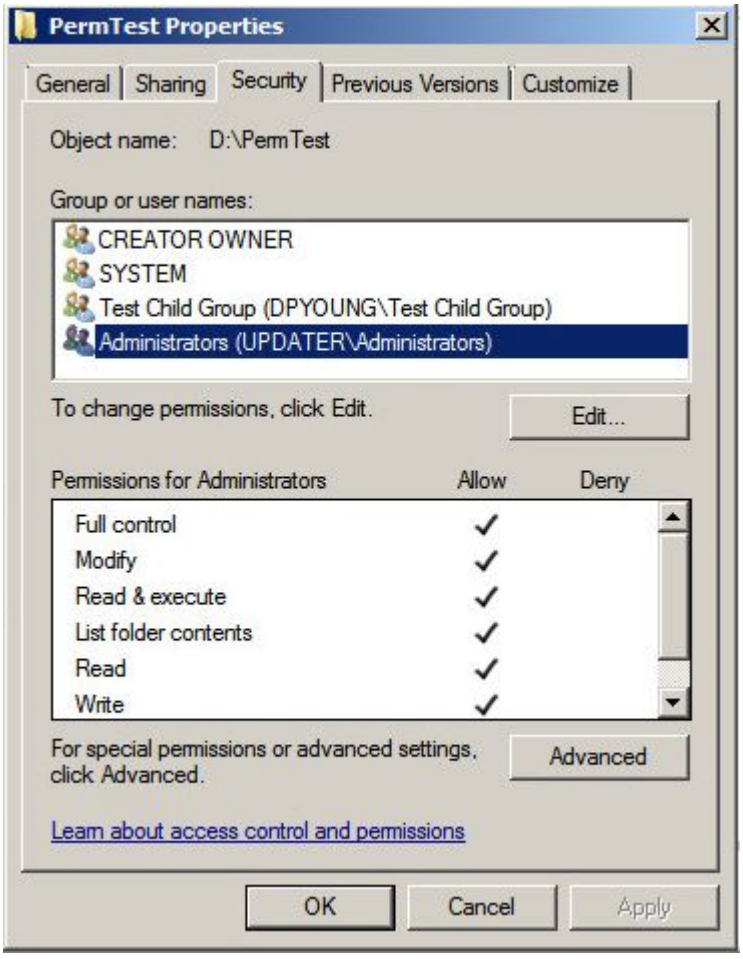
🔴 Baixo Nível

- No baixo nível, as permissões são apenas bits e identificadores armazenados nos metadados do arquivo (inode ou registro de diretório) pelo sistema de arquivos.
- O hardware do disco não entende permissões; ele só armazena os dados.
- O controle real de acesso só acontece quando o SO interpreta esses bits ao tentar acessar o arquivo.

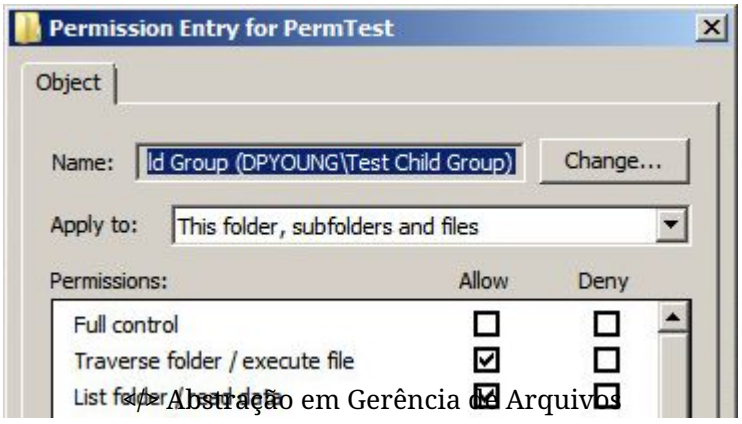
🟡 Abstração do SO

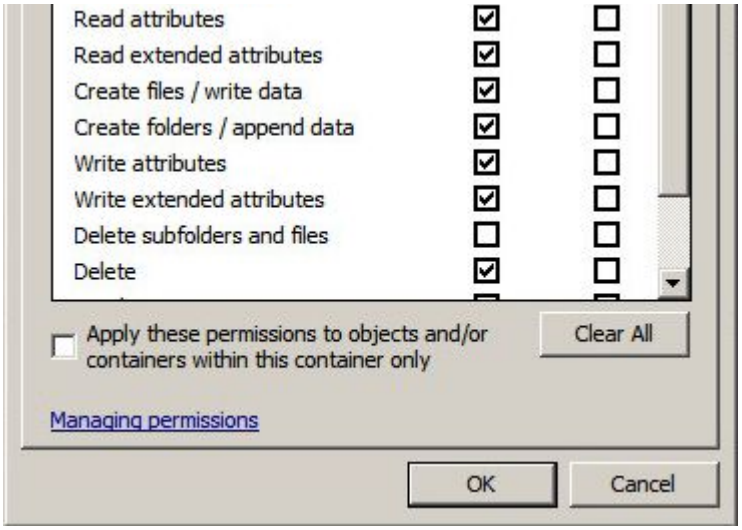
As permissões de arquivos são um mecanismo do sistema operacional para controlar quem pode acessar ou modificar arquivos e diretórios. Cada arquivo tem um usuário proprietário e um grupo, e as permissões são divididas em três níveis (usuário, grupo e outros), permitindo diferentes tipos de acesso e garantindo segurança e compartilhamento adequado.

Sistemas de arquivos como o NTFS (New Technology File System) organizam e controlam o armazenamento lógico dos dados, ocultando a complexidade do hardware. No NTFS, permissões são aplicadas a todos os arquivos e pastas do volume, normalmente herdadas da pasta raiz, mas essa herança pode ser desativada. As permissões NTFS valem para acessos locais e remotos, e incluem níveis como Leitura, Execução, Gravação, Modificação, Listagem e Controle Total.

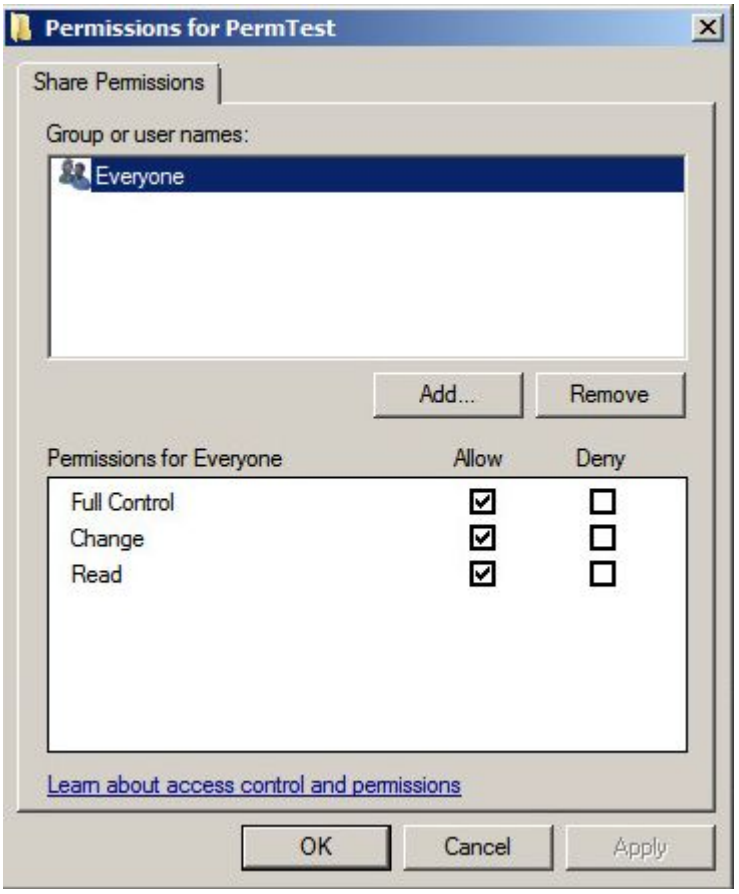


Além dessas, há um conjunto de permissões avançadas, que detalham ainda mais os níveis de acesso, permitindo configurações mais específicas conforme o tipo de objeto (arquivo ou pasta). Essas permissões avançadas oferecem maior granularidade e controle sobre as ações que cada usuário pode realizar.





Outro tipo de controle importante são as permissões de compartilhamento, aplicadas apenas a pastas compartilhadas em rede. Elas entram em vigor quando uma pasta é acessada remotamente e determinam o nível de acesso permitido sobre o conteúdo compartilhado. Embora sejam menos detalhadas do que as permissões NTFS, as permissões de compartilhamento também oferecem três níveis principais: Leitura, Modificação e Controle Total.



Assim, dentro da abstração de gerência de arquivos, o sistema operacional utiliza mecanismos como o NTFS e suas permissões para oferecer ao usuário uma forma segura, flexível e intuitiva de interagir com os dados, sem a necessidade de lidar diretamente com a complexidade física do armazenamento.

🟢 Abstração Programática

Organizações utilizam softwares de gerenciamento de permissões que adicionam camadas sobre o SO para facilitar controle e auditoria em larga escala:

Exemplos de soluções populares:

Varonis

- **Active Directory (Microsoft):** gerenciamento centralizado de usuários, grupos e políticas de acesso em domínios Windows.

Varonis

- **Varonis:** plataforma para análise, auditoria e proteção de dados com foco em detecção de anomalias e conformidade.

Varonis

- **SailPoint IdentityIQ:** solução enterprise para governança de identidades, auditoria e controle de acessos.

Pontos fortes em comparação com o SO puro:

- **Visão centralizada:** interface única para gerenciar permissões em múltiplos servidores, aplicações e sistemas de arquivos.
- **Auditoria e compliance:** logs detalhados, relatórios de quem tem acesso a quê, e alertas de mudanças críticas.
- **Automação:** workflows de aprovação, provisionamento/revogação automática de acessos e integração com RH (onboarding/offboarding).
- **Análise de risco:** detecção de permissões excessivas, usuários com privilégios desnecessários e padrões anômalos de acesso.
- **Políticas de negócio:** aplicação de regras por departamento, cargo ou projeto, além do modelo simples rwx do SO.

---

3. Mecanismo de Acesso

🔴 Baixo Nível

No baixo nível, acessar arquivos envolve:

- Leitura e gravação de blocos diretamente no disco físico.
- Comunicação com o hardware via controladores e drivers (ex: SATA, NVMe).
- Uso de memória temporária (buffer cache) para acelerar operações.
- Sinais do disco (interrupções) para avisar o sistema quando uma operação termina.
- Métodos como o DMA (Direct Memory Access) permitem que dados sejam transferidos entre disco e RAM sem ocupar a CPU o tempo todo.

🟡 Abstração do SO

Syscalls open/read/write/close, descritores de arquivo e resolução de caminho via VFS (Virtual File System) padronizam o acesso e isolam a aplicação dos detalhes físicos.

Os mecanismos de acesso definem como os processos interagem com arquivos, abstraindo a complexidade do armazenamento físico através de interfaces padronizadas.

Nesta camada, descritores de arquivo, caminhos e chamadas de sistema (open, read, write, close) formam um CONTRATO estável que nosso trabalho reutiliza para oferecer operações de alto nível (listar, buscar, mapear) sem expor detalhes de baixo nível.

🚧 Pontos fracos do acesso manual de arquivos

- **Escalabilidade limitada:** Com o crescimento do volume de arquivos, navegar manualmente pela estrutura de diretórios torna-se cada vez mais demorado e propenso a erros
- **Busca ineficiente:** Localizar um prontuário específico requer conhecer exatamente sua localização na hierarquia de pastas, sem suporte para busca por conteúdo ou metadados
- **Inconsistência de nomenclatura:** Sem padronização automatizada, diferentes usuários podem criar arquivos com nomes inconsistentes, dificultando a organização
- **Dificuldade de integração:** Sistemas que dependem de acesso manual não se integram facilmente com outras ferramentas (relatórios, alertas, backup automatizado)

🟢 Abstração Programática

Abstraindo ainda mais o processo de acesso

Ao subir camadas de abstração, conseguimos mitigar os pontos fracos do gerenciamento manual de arquivos. Gerar a estrutura em um arquivo CSV adiciona uma camada estruturada de metadados que facilita consumo por outros sistemas.

Mapeamento de arquivos em formato tabular (CSV):

In [11]:
```
# Listar todos os arquivos dos prontuários
save_and_display_csv("prontuarios", "mapeamento_prontuarios.csv")
```

```
✓ Arquivo CSV salvo: mapeamento_prontuarios.csv
```

Out[11]:

Status Paciente	Nome Paciente	Categoria	Nome	Tipo	E
Arquivado			pacientes_arquivados	**Diretório**	
Arquivado	P0-Olavo_de_Carvalho		P0-Olavo_de_Carvalho	**Diretório**	
Arquivado	P0-Olavo_de_Carvalho	Consultas	consultas	**Diretório**	
Arquivado	P0-Olavo_de_Carvalho	Consultas	dificuldade_respiratória.pdf	Arquivo	.p
Arquivado	P0-Olavo_de_Carvalho	Exames	exames	**Diretório**	
Arquivado	P0-Olavo_de_Carvalho	Exames	covid19.pdf	Arquivo	.p
Arquivado	P0-Olavo_de_Carvalho	Profile	profile	**Diretório**	
Arquivado	P0-Olavo_de_Carvalho	Profile	P0.jpg	Arquivo	.j
Arquivado	P0-Olavo_de_Carvalho	Tratamentos	tratamentos	**Diretório**	
Arquivado	P0-Olavo_de_Carvalho	Tratamentos	internação_covid.pdf	Arquivo	.p
Arquivado	P1-Everson_Olhos		P1-Everson_Olhos	**Diretório**	
Arquivado	P1-Everson_Olhos	Cirurgias	cirurgias	**Diretório**	
Arquivado	P1-Everson_Olhos	Cirurgias	extração_dente.pdf	Arquivo	.p
Arquivado	P1-Everson_Olhos	Consultas	consultas	**Diretório**	

Status Paciente	Nome Paciente	Categoria	Nome	Tipo	E
Arquivado	P1-Everson_Olhos	Consultas	dor_de_dente.pdf	Arquivo	.p
Arquivado	P1-Everson_Olhos	Profile	profile	**Diretório**	
Arquivado	P1-Everson_Olhos	Profile	P1.jpg	Arquivo	.j
Ativo			pacientes_ativos	**Diretório**	
Ativo	P3-Maria_Hipotetica		P3-Maria_Hipotetica	**Diretório**	
Ativo	P3-Maria_Hipotetica	Cirurgias	cirurgias	**Diretório**	
Ativo	P3-Maria_Hipotetica	Cirurgias	hernia_de_disco.pdf	Arquivo	.p
Ativo	P3-Maria_Hipotetica	Consultas	consultas	**Diretório**	
Ativo	P3-Maria_Hipotetica	Consultas	dor_na_coluna.pdf	Arquivo	.p
Ativo	P3-Maria_Hipotetica	Exames	exames	**Diretório**	
Ativo	P3-Maria_Hipotetica	Exames	raiox_coluna.pdf	Arquivo	.p
Ativo	P3-Maria_Hipotetica	Profile	profile	**Diretório**	
Ativo	P3-Maria_Hipotetica	Profile	P3.jpg	Arquivo	.j
Ativo	P4-Allexandre		P4-Allexandre	**Diretório**	
Ativo	P4-Allexandre	Exames	exames	**Diretório**	

Status Paciente	Nome Paciente	Categoria	Nome	Tipo	E
Ativo	P4-Allexandre	Exames	exame_de_prostata.pdf	Arquivo	.p
Ativo	P4-Allexandre	Profile	profile	**Diretório**	
Ativo	P4-Allexandre	Profile	P4.jpg	Arquivo	.j

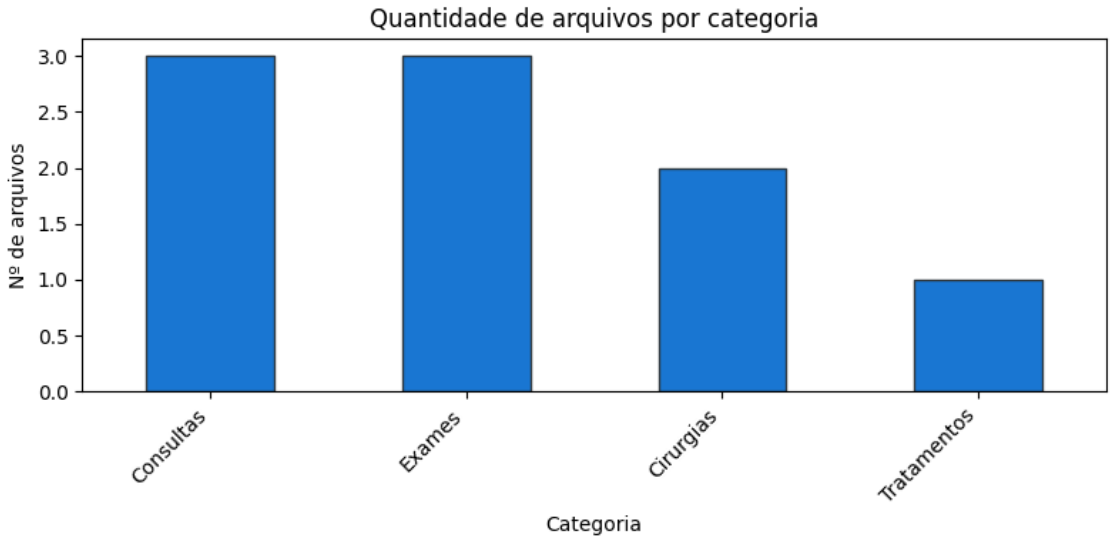
Benefícios práticos do CSV gerado:

- **Visão unificada e filtrável:** status (ativo/arquivado), paciente, categoria, tipo, extensão, tamanho e caminho em um único lugar
- **Busca e filtros rápidos:** encontrar arquivos por extensão (.pdf, .jpg), por categoria (exames, consultas) ou por tamanho (arquivos grandes)
- **Integração fácil:** abre no Excel/Google Sheets, alimenta dashboards (BI) e scripts Python (Pandas)
- **Auditoria de mudanças:** comparar versões do CSV no controle de versão para ver adições/remoções e reorganizações de pastas
- **Automação e relatórios:** gerar KPIs (total por categoria, espaço ocupado por paciente, top N arquivos maiores)
- **Detecção de inconsistências:** nomes fora do padrão, arquivos em pastas erradas, extensões inesperadas
- **Portabilidade e compartilhamento:** compartilhar o mapa da estrutura sem expor o conteúdo sensível dos arquivos
- **Base para indexação:** serve como entrada para mecanismos de busca/indice sem varrer o disco toda hora

Mini relatório: quantidade de arquivos por categoria

Este gráfico mostra quantos arquivos existem em cada categoria (consultas, exames, etc.). Ele ilustra como o CSV permite gerar estatísticas rápidas sem navegar manualmente pelas pastas.

In [12]:
```
gerar_relatorio_contagem_categoria()
```



Out[12]:
```
Categoria
Consultas      3
Exames         3
Cirurgias      2
Tratamentos    1
dtype: int64
```

---

4. Armazenamento

🔴 Baixo Nível

No baixo nível, o armazenamento de arquivos em disco é organizado através de estruturas de dados que determinam como localizar e acessar os blocos físicos onde os dados estão gravados. Uma das técnicas mais eficientes é a **alocação indexada**, que utiliza blocos de índice para mapear a localização dos dados no disco.

📋 Alocação Indexada com Lista de Índices

Diferente de outras técnicas, como a alocação encadeada (lista encadeada), a **lista com índice** resolve problemas como o "acesso lento" (percorrer bloco por bloco até encontrar um determinado bloco).

⚙️ Como Funciona?

Bloco de Índice:

- Não armazena dados do arquivo, mas sim **metadados** (tamanho do arquivo, permissões, proprietários) e principalmente uma **lista de endereços** que apontam para os blocos de dados.

Bloco de Dados:

- Contém o conteúdo real do arquivo. Exemplos: imagem, prontuários de pacientes, documentos médicos.

📖 Mecanismo de Leitura

Para acessar um bloco específico (exemplo: 20º bloco), o processo é dividido em **duas leituras**:

1º Leitura:

1. **Localizar o Índice:** O SO encontra o endereço do bloco índice no disco
2. **Carregar para RAM:** O Sistema Operacional realiza a primeira leitura para copiar o bloco de índice (do disco) para a memória RAM

Objetivo: O índice agora está na memória principal, pronto para consulta rápida.

2º Leitura:

- O SO acessa diretamente a 20ª entrada na lista do bloco de índice (que está na RAM)
- O SO extrai o endereço físico do disco (ex: bloco 415) armazenado naquela posição
- Realiza a segunda leitura para buscar o bloco de dados correspondente

Objetivo: O SO tem a coordenada exata de onde estão os dados no disco.

Varonis

✅ Vantagens:

- **Acesso Rápido (Aleatório):** Permite que o SO encontre qualquer bloco de dados do arquivo, independente do tamanho, utilizando apenas duas leituras
- **Sem Fragmentação Externa:** Diferente da alocação contígua, os blocos podem estar espalhados no disco
- **Crescimento Flexível:** É possível alocar novo bloco de dados em qualquer lugar do disco e adicionar seu endereço ao bloco de índice

🚧 **Desvantagens:**

- **Gasto de Espaço:** Cada arquivo, mesmo os pequenos, precisa de um bloco de índice inteiro
- **Complexidade da Indireção:** Para arquivos muito grandes, o uso de indireção multinível (índices que apontam para outros índices) torna a estrutura mais complexa de ser gerenciada pelo Sistema Operacional

---

🟡 Abstração do SO

O Sistema Operacional abstrai a complexidade do armazenamento físico através de ferramentas visuais e APIs que escondem os detalhes de baixo nível.

🗂️ Gerenciador de Arquivos

O **Gerenciador de Arquivos** é a interface gráfica do SO para manipulação de arquivos e diretórios:

Varonis

Principais Funcionalidades:

- Navegação em hierarquia de pastas
- Operações básicas: criar, copiar, mover, renomear, excluir
- Visualização de metadados (tamanho, data, permissões)
- Busca por nome, tipo ou conteúdo
- Integração com aplicativos do sistema

Exemplos: Windows Explorer, macOS Finder, Linux Nautilus

🎯 O que o SO Abstrai:

- Localização física dos dados no disco (blocos, setores)
- Sistema de arquivos utilizado (NTFS, ext4, FAT32)
- Técnicas de alocação (contígua, encadeada, indexada)
- Operações de I/O e comunicação com hardware

Resultado: O usuário manipula arquivos de forma intuitiva (arrastar e soltar), sem conhecer detalhes técnicos do armazenamento físico.

🟢 Abstração Programática

Funções de gerenciamento avançado da estrutura de prontuários:

Exemplos de uso:

Criar um novo paciente (com imagem placeholder no profile/):

```
In [13]: gp.criar_paciente("Vini_Junior", categorias=['consultas', 'exame
         s', 'cirurgias'])
```

```
📷 Imagem de perfil criada: profile.jpg
✅ Paciente 'P5-Vini_Junior' criado com sucesso!
   Localização: prontuarios/pacientes_ativos/P5-Vini_Junior
   Categorias: consultas, exames, cirurgias
```

```
Out[13]: PosixPath('prontuarios/pacientes_ativos/P5-Vini_Junior')
```

Listar pacientes ativos:

```
In [14]: gp.listar_pacientes(arquivado=False)
```

```
==================================================
Pacientes Ativos: 3
==================================================
  • P3-Maria_Hipotetica
  • P4-Allexandre
  • P5-Vini_Junior
==================================================
```

```
Out[14]: ['P3-Maria_Hipotetica', 'P4-Allexandre', 'P5-Vini_Junior']
```

Visualizar árvore de pacientes ativos atualizada:

```
In [15]: draw_tree_auto("prontuarios/pacientes_ativos")
```

Arquivar o paciente pelo ID (ex: arquivar P5):

In [16]:
```
gp.arquivar_paciente(5)
```

```
🗄️  Paciente 'P5-Vini_Junior' arquivado com sucesso!
   De: prontuarios/pacientes_ativos/P5-Vini_Junior
   Para: prontuarios/pacientes_arquivados/P5-Vini_Junior
```

Out[16]:
```
PosixPath('prontuarios/pacientes_arquivados/P5-Vini_Junior')
```

Listar pacientes arquivados:

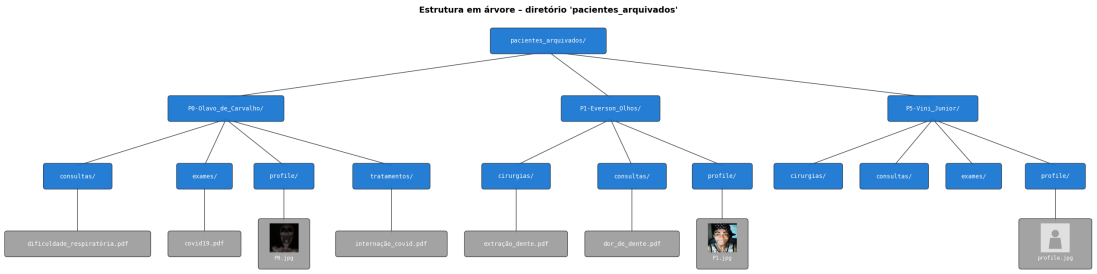
In [17]:
```
gp.listar_pacientes(arquivado=True)
```

```
==================================================
Pacientes Arquivados: 3
==================================================
  • P0-Olavo_de_Carvalho
  • P1-Everson_Olhos
  • P5-Vini_Junior
==================================================
```

Out[17]:
```
['P1-Everson_Olhos', 'P0-Olavo_de_Carvalho', 'P5-Vini_Junior']
```

Visualizar árvore de pacientes arquivados atualizada:

In [18]:
```
draw_tree_auto("prontuarios/pacientes_arquivados")
```



Desarquivar o paciente pelo ID (ex: desarquivar P5):

In [19]:
```
gp.desarquivar_paciente(5)
```

```
✅ Paciente 'P5-Vini_Junior' reativado com sucesso!
   De: prontuarios/pacientes_arquivados/P5-Vini_Junior
   Para: prontuarios/pacientes_ativos/P5-Vini_Junior
```

Out[19]:
```
PosixPath('prontuarios/pacientes_ativos/P5-Vini_Junior')
```

Listar pacientes ativos:

In [20]:
```
gp.listar_pacientes(arquivado=False)
```

```
==================================================
Pacientes Ativos: 3
==================================================
  • P3-Maria_Hipotetica
  • P4-Allexandre
  • P5-Vini_Junior
==================================================
```

Out[20]:
```
['P3-Maria_Hipotetica', 'P4-Allexandre', 'P5-Vini_Junior']
```

Visualizar árvore de pacientes ativos atualizada:

In [21]:
```
draw_tree_auto("prontuarios/pacientes_ativos")
```

Criar um documento para o paciente (ID 4, categoria 'consultas')

In [22]:
```
gp.criar_documento(
    id_paciente=4,
    categoria='consulta',
    nome_arquivo="consulta_2025-11-13.txt",
    conteudo="Consulta de rotina.\nPressão: 120/80\nPeso: 75kg\n")
```

```
📋 Documento criado com sucesso!
   Paciente: P4-Allexandre
   Categoria: consulta
   Arquivo: prontuarios/pacientes_ativos/P4-Allexandre/consulta/co
nsulta_2025-11-13.txt
```

Out[22]:
```
PosixPath('prontuarios/pacientes_ativos/P4-Allexandre/consulta/con
sulta_2025-11-13.txt')
```

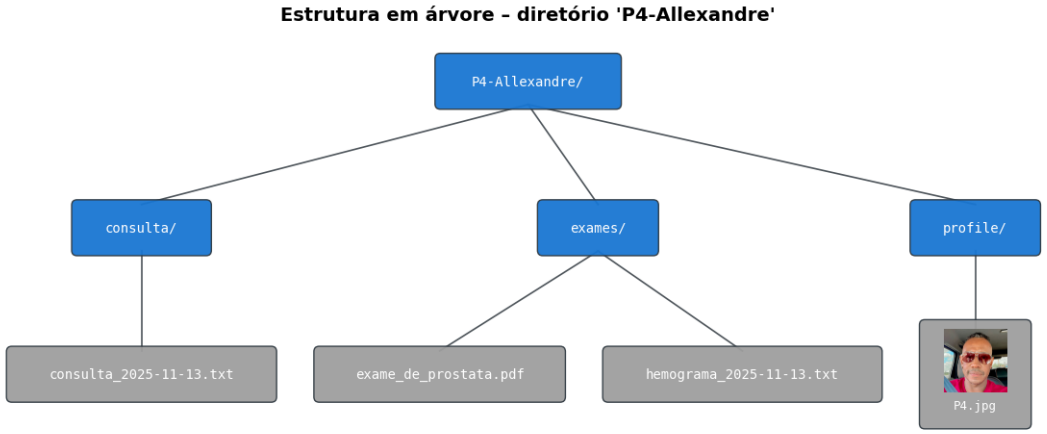
In [23]:
```
# Criar outro exame para paciente (ID 4, categoria 'exames')
gp.criar_documento(
    id_paciente=4,
    categoria='exames',
    nome_arquivo="hemograma_2025-11-13.txt",
    conteudo="Hemograma completo normal.\nLeucócitos: 7000/mm3\n")
```

```
📋 Documento criado com sucesso!
   Paciente: P4-Allexandre
   Categoria: exames
   Arquivo: prontuarios/pacientes_ativos/P4-Allexandre/exames/hemograma_2025-11-13.txt
```

Out[23]:
```
PosixPath('prontuarios/pacientes_ativos/P4-Allexandre/exames/hemograma_2025-11-13.txt')
```

Visualizar árvore do Paciente 4 atualizada:

In [24]:
```
draw_tree_auto("prontuarios/pacientes_ativos/P4-Allexandre")
```



🎯 Conclusão

A abstração em gerência de arquivos é fundamental para a usabilidade dos sistemas computacionais. Através do estudo de caso com prontuários hospitalares, demonstramos como três camadas de abstração transformam operações complexas de hardware em interfaces intuitivas.

🔑 Vantagens da Abstração:

- **Simplicidade:** Usuários manipulam arquivos sem conhecer detalhes técnicos de hardware
- **Portabilidade:** Mesmo código funciona em diferentes sistemas de arquivos e dispositivos
- **Manutenibilidade:** Mudanças no hardware não afetam aplicações existentes
- **Produtividade:** Desenvolvedores focam na lógica de negócio, não em detalhes de I/O

💡 Reflexão Final

A abstração permite que profissionais de saúde foquem no atendimento, desenvolvedores criem sistemas eficientes, e o SO gerencie a complexidade do armazenamento. **A abstração não esconde apenas complexidade — ela libera potencial humano para resolver problemas mais importantes.**

🙏 Obrigado!

Disciplina: Sistemas Operacionais
Curso: Ciência da Computação
Data: 13 de Novembro de 2025

Grupo:

- Camilla Macedo Alves
- Gabriel Oliveira de Souza
- João Gabriel Rocha Cerqueira
- Virgínia Mayumi Furushima de Freitas