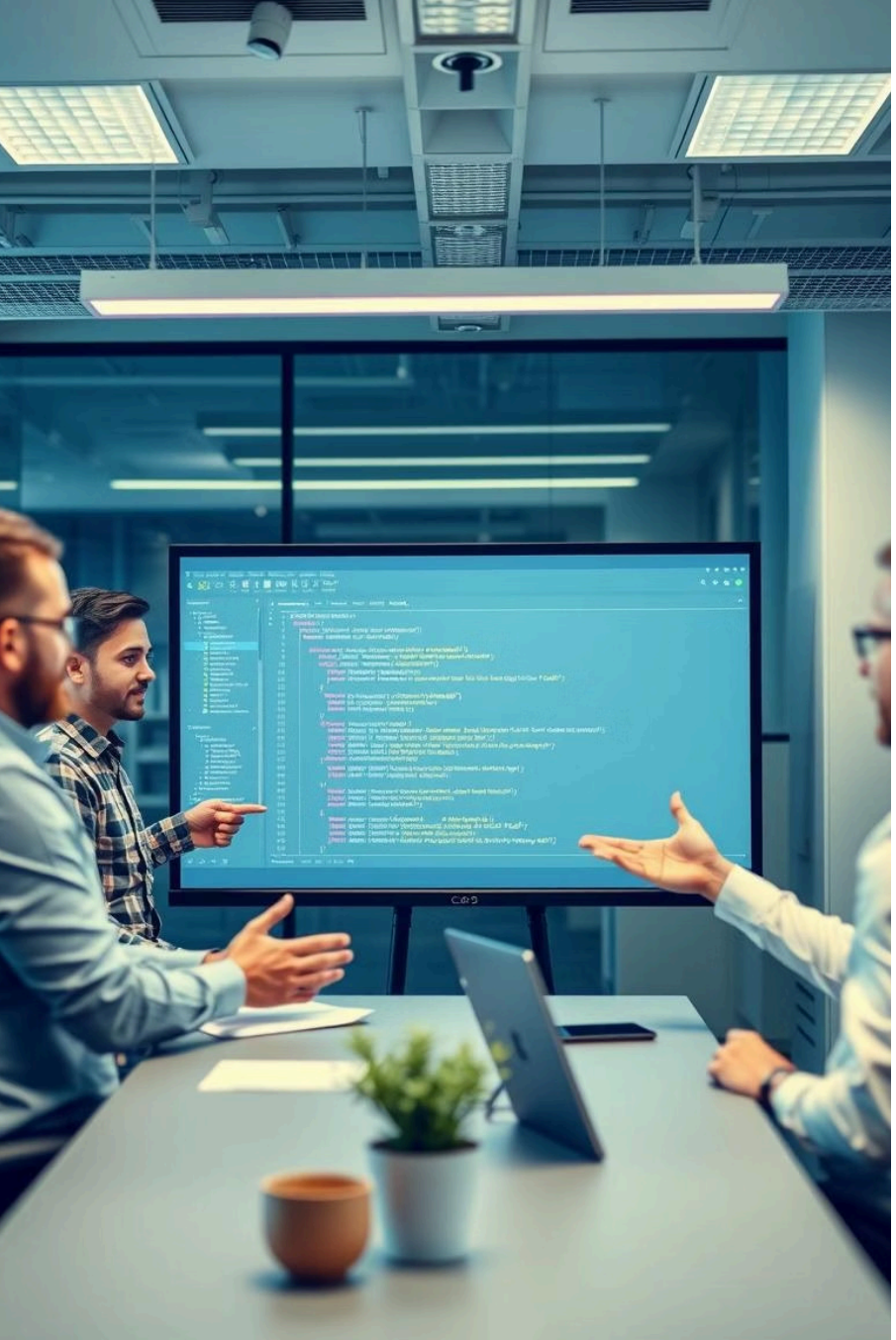

Sistema de Gerenciamento de Serviços

Este guia apoia a apresentação final do nosso projeto de Estruturas de Dados I. Abordaremos os aspectos técnicos e práticos do Sistema de Gerenciamento de Serviços.

Abertura da Apresentação e Resumo Executivo

- **Estrutura Principal: Lista Duplamente Encadeada**
- **Linguagem: C puro + interface GTK+**
- **Funcionalidades: CRUD completo com ordenação automática**
- **Persistência: Sistema de arquivos CSV**
- **Status: Funcional e estável**

Demonstração Prática do Sistema

1 Adicionar serviços

Com diferentes prioridades (Alta, Média, Baixa).

2 Visualizar ordenação automática

A lista mantém a ordem (Alta → Média → Baixa).

3 Atualizar status

De serviços existentes em tempo real.

4 Excluir serviços

Com confirmação para evitar erros.

5 Filtros dinâmicos

Por status e prioridade, para melhor visualização.

6 Persistência dos dados

Fechar e reabrir o programa, mantendo os dados.

Durante a demonstração, observe a ordenação automática e o salvamento em CSV. O sistema possui interface de linha de comando e gráfica.

Perguntas e Respostas

Por que escolheram Lista Duplamente Encadeada?

- Inserção ordenada por prioridade
- Navegação bidirecional facilita busca e remoção
- Memória dinâmica (cresce conforme necessário)
- Complexidade O(n) aceitável para o contexto

Como implementaram o gerenciamento de memória?

- Verificação de **malloc()** antes de usar ponteiros
- Função **liberaMemoria()** percorre a lista e faz **free()**
- Ponteiros sempre inicializados como **NULL**
- Nenhum vazamento de memória detectado

Qual a complexidade dos algoritmos principais?

- Inserção: O(n) - encontrar posição por prioridade
- Busca: O(n) - busca linear por ID
- Remoção: O(n) + O(1) - localizar + ajustar ponteiros
- Atualização: O(n) - busca + modificação

Como funciona a inserção ordenada?

```
// Prioridade: A=3, M=2, B=1
if (novaPrioridadeVal > getPrioridadeValor(listaDeServicos->prioridade)) {
  // Inserir no início (maior prioridade)
} else {
  // Percorrer até encontrar posição correta
  while (atual->prox != NULL && getPrioridadeValor(atual->prox->prioridade) >= novaPrioridadeVal)
}
```

Como implementaram a persistência?

- Arquivo CSV simples: id,descricao,status,prioridade
- Salvamento automático após cada operação
- Carregamento na inicialização do programa
- Tratamento de erros de I/O implementado

Estrutura de Dados

Definição da Estrutura Servico:

```
typedef struct Servico {
  int id;         // Chave única
  char descricao[100]; // Descrição textual
  char status;      // P, E, C
  char prioridade;   // A, M, B
  struct Servico *ant;  // ← Ponteiro anterior
  struct Servico *prox; // → Ponteiro próximo
} Servico;
```

Vantagens da Lista Duplamente Encadeada:

- Inserção ordenada automática por prioridade.
- Remoção eficiente (O(1) após localização).
- Navegação bidirecional para percorrer a lista.
- Uso eficiente de memória (aloca conforme necessário).

Operações Principais:

Inserir

Encontra posição por prioridade e insere, mantendo a ordem.

Buscar

Percorre linearmente até encontrar o ID desejado.

Remover

Ajusta ponteiros dos nós adjacentes para exclusão.

Atualizar

Localiza o nó e modifica os campos in-place.

Interface e Usabilidade

CLI (Console)

Interface tradicional de linha de comando, para operações rápidas.

GTK+

Interface gráfica moderna com widgets interativos, fácil de usar.

Funcionalidades da Interface:

- Formulários para entrada de dados.
- Validação em tempo real para evitar erros.
- Filtros dinâmicos por status e prioridade.
- Contadores automáticos de serviços ativos.
- Confirmações para operações críticas.

Justificativas Técnicas

Por que não usaram Array/Vetor?

Arrays têm tamanho fixo; nossa solução é dinâmica. Inserção ordenada em array é O(n) e realocação custosa. Lista encadeada permite inserção O(1) na posição correta.

Por que não usaram Árvore/Heap?

Complexidade desnecessária para o escopo do projeto. A lista atende perfeitamente aos requisitos. O foco era demonstrar estruturas lineares da disciplina.

E se precisasse de performance melhor?

Hash table para busca O(1) por ID. Heap para prioridades, se o volume fosse muito grande. Para o contexto atual, a lista é adequada.

Métricas Finais e Objetivos Alcançados

O que foi entregue:

- Sistema funcional com todas as operações CRUD.
- Interface dupla (CLI + GTK+).
- Persistência de dados implementada.
- Gerenciamento de memória seguro.
- Código bem estruturado e documentado.

100%

Aplicação Prática

De estruturas de dados.

100%

Algoritmos Eficientes

De inserção, busca e remoção.

100%

Projeto Completo

E totalmente funcional.

100%

Boas Práticas

Código limpo e documentado.