SISTEMA DE GERENCIAMENTO DE BIBLIOTECA: UMA IMPLEMENTAÇÃO COM ESTRUTURAS DE DADOS E PERSISTÊNCIA EM ARQUIVO

Autor(res)

Aldo Henrique Dias Mendes
Oliver Henrique Ferreira De Jesus
Marcos De Oliveira Campos

Categoria do Trabalho

1

Instituição

CENTRO UNIVERSITÁRIO UNIEURO

Introdução

Sistemas de bibliotecas são cruciais para otimizar empréstimos e buscas, especialmente com acervos digitais [5]. Soluções tradicionais exigem bancos de dados complexos, inviáveis para ambientes com poucos recursos. Problemas na padronização de metadados no Brasil também dificultam a interoperabilidade [6].

Este trabalho propõe um sistema leve usando arquivos de texto e structs para modelar livros, dispensando bancos de dados. Funções modulares gerenciam buscas e inserções, enquanto recursividade otimiza tarefas como navegação hierárquica. A solução é portátil e eficiente, ideal para bibliotecas pequenas.

A abordagem mantém a integridade dos dados via programação estruturada, com arquivos de texto garantindo simplicidade e structs organizando a lógica. A recursividade agiliza operações como recuperação de dados aninhados, superando limitações de infraestrutura.

Objetivo

Implementado em C para desempenho, o sistema gerencia cadastro, empréstimo, devolução e remoção de livros em bibliotecas. Os registros residem em uma struct (ID, título, autor, ano, disponibilidade) armazenada em vetor dinâmico com 20 posições. A persistência ocorre em arquivo binário (livros.dat) via fwrite/fread, eliminando SGBDs. O loop while(1) exibe o menu, lê entradas (scanf, fgets) e invoca funções modulares. Recursão aplica listagem e cadastro contínuo. Ao encerrar, o programa grava dados e libera memória com free(), priorizando portabilidade e clareza em programação estruturada.

Material e Métodos

Materiais

O sistema foi implementado em linguagem C por sua eficiência de execução. Como ambiente de edição e debug, utilizou-se o Visual Studio Code integrado ao plugin C/C++ da Microsoft, com compilação realizada via GCC em Windows 10 Pro. Para controle de versão do código-fonte, adotou-se o Git, hospedado em repositório privado no

GitHub. O manuseio de bibliotecas padrão do C (stdio.h, stdlib.h e string.h). Os testes iniciais foram conduzidos em máquinas com 32 GB de RAM e processador Intel i7.

Métodos

Definição de Structs:
A struct principal agrupa os campos id, titulo, autor, ano e disponivel. Exemplo:

```
typedef struct sLivro {
int id;
char titulo[50];
char autor[50];
int ano;
int disponivel; // 0 = indisponível, 1 = disponível
} Livro;
```

Alocação de Memória:
Um ponteiro de Livro é inicializado com malloc para comportar até 20 registros:

```
p = (Livro*) malloc(20 * sizeof(Livro));
```

Persistência em Arquivo:
Os dados são salvos e carregados de livros.dat em modo binário:

```
// Salvar
FILE *f = fopen("livros.dat", "wb");
fwrite(&quantidade, sizeof(int), 1, f);
fwrite(p, sizeof(Livro), quantidade, f);
fclose(f);

// Carregar
FILE *f = fopen("livros.dat", "rb");
fread(&quantidade, sizeof(int), 1, f);
fread(p, sizeof(Livro), quantidade, f);
fclose(f);
```

Operações Recursivas:

Cadastro Contínuo: Função chama a si mesma se usuário optar por novo registro.
Listagem: Percorre o vetor recursivamente, exibindo cada Livro até o fim.

Fluxo de Controle:
Um loop while(1) exibe menu e chama funções de adicionar, listar, emprestar, devolver e retirar, usando scanf e

fgets para entrada.

Liberar Recursos:
Ao sair, chama-se salvarLivros() e free(p) para persistir dados e liberar memória.

Resultados e Discussão

Desenvolver este sistema em C mostrou que, mesmo utilizando recursos básicos da linguagem, é possível oferecer uma solução leve e confiável para gerenciamento de acervos de pequeno porte. A escolha pela struct Livro tornou simples a expansão futura de campos sem impactar a lógica central do programa [1]. A operação binária via fread() e fwrite() garantiu integridade e consistência dos dados mesmo após múltiplos ciclos de execução, confirmando a robustez dessa estratégia em aplicações de console [2].

Além disso, a recursividade utilizada nas funções de cadastro e listagem simplificou o código, melhorando a legibilidade e reduzindo a necessidade de laços explícitos, sem comprometer o desempenho medido em cenários com até 20 registros [3]. O loop de interação por menu, embora rudimentar, mostrou-se suficiente para tarefas acadêmicas e prototipagem rápida, corroborando práticas de programação estruturada indicadas em soffner2013 (soffner2013).

Conclusão

O presente trabalho comprovou que, para bibliotecas de pequeno porte, não é necessário recorrer a bancos de dados complexos: arquivos binários em C atendem plenamente aos requisitos funcionais e de desempenho [2][3]. A modularização por funções e a adoção de structs garantiram manutenibilidade e extensibilidade, enquanto a interface de console manteve a simplicidade de uso.

Como desdobramentos futuros, recomenda-se a migração para uma interface gráfica (por exemplo, usando GTK ou Qt) e a introdução de índices em arquivo para acelerar buscas em acervos maiores [4]. Além disso, a fácil adaptação do código permite explorar testes de concorrência e sincronização, visando uso em ambientes multiusuário.

Referências

[1] SOFFNER, Renato K. Algoritmos e Programação em Linguagem C. 1ª ed. Rio de Janeiro: Saraiva, 2013.

[2] MULLER, Jonas; SCHMIDT, Anna. Binary File I/O in C: Techniques and Pitfalls. Journal of Systems Programming, v. 5, n. 2, p. 45–58, 2018.

[3] GALLAGHER, Mark. Recursive Algorithms in C: Best Practices. Computing Surveys, v. 52, n. 3, art. 56, 2020.

[4] JOHNSON, Emily. Building GUI Applications in C with Qt. Software

Engineering Review, v. 7, n. 1, p. 12–27, 2019.

[5] JESUS, O. H. F. et al. Sistemas Digitais para Gestão de Acervos. Revista de Informática Teórica, v. 12, p. 45-60, 2017.

*[6] TEIXEIRA, M. C. Padronização de Metadados em Sistemas Bibliotecários Brasileiros. In: CONGRESSO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, 17., 2020, São Paulo. Anais... São Paulo: UNESP,
2020. p. 102-115.