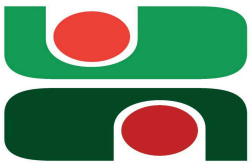

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Linguagem de Programação (LPG0002)

Sistema de Agendamento de Salas

Alunos: Guilherme Ytalo e Nícolas Morais

Professor: Ricardo Ferreira Martins

Joinville, 01 de julho de 2024

Sumário

1. Introdução

A gestão eficiente de recursos é um aspecto crucial para o bom funcionamento de qualquer instituição, especialmente em ambientes acadêmicos, onde a organização e a utilização de espaços físicos são de fundamental importância. No contexto universitário, a disponibilidade e o agendamento adequado de salas para aulas, reuniões e outros eventos são desafios constantes. A ausência de um sistema automatizado pode levar a conflitos de agendamento, subutilização ou superlotação das salas, impactando negativamente a rotina acadêmica e administrativa.

Neste trabalho, desenvolvemos um sistema de agendamento de salas utilizando a linguagem de programação C. O sistema foi projetado para atender às necessidades, permitindo o gerenciamento eficaz das salas disponíveis, suas características, e as reservas associadas. O sistema possibilita que os usuários listem as salas disponíveis, verifiquem reservas existentes, agendem novas reservas, e cancelem reservas indesejadas.

O sistema busca não apenas automatizar o processo de agendamento, mas também garantir a integridade e a consistência dos dados. A utilização de técnicas de alocação dinâmica visa otimizar o uso de memória, tornando o sistema eficiente e escalável. Além disso, foram implementadas funcionalidades para o tratamento de erros comuns, assegurando que o usuário não possa realizar ações que comprometam a integridade do sistema, como a tentativa de reserva de uma sala já ocupada ou a solicitação de uma sala com capacidade insuficiente para o evento planejado.

Este relatório detalha o desenvolvimento do sistema, abordando desde a concepção e metodologia até os resultados obtidos e as considerações finais. A estrutura do relatório inclui uma descrição das principais funcionalidades implementadas, a lógica de tratamento de erros e os benefícios alcançados com a aplicação da alocação dinâmica. Nosso objetivo é apresentar uma solução robusta e eficiente para o gerenciamento de agendamento de salas, contribuindo para a organização e o bom funcionamento da universidade.

2. Objetivos

Objetivo Geral

Desenvolver um sistema de agendamento de salas em linguagem C, que possibilite o gerenciamento eficaz das reservas de salas, incluindo a listagem de salas disponíveis, consulta de reservas, agendamento de novas reservas e cancelamento de reservas existentes, garantindo a integridade dos dados e a otimização do uso de recursos.

Objetivos Específicos

1. Listar Salas Disponíveis
- Implementar uma funcionalidade que permita ao usuário listar todas as salas disponíveis na universidade, exibindo informações detalhadas como nome, capacidade máxima e disponibilidade atual.

2. Registrar e Listar Reservas
- Criar mecanismos para registrar novas reservas de salas, garantindo que cada reserva contenha o ID da sala, a data reservada e o número de participantes.
- Implementar uma funcionalidade para listar todas as reservas existentes, incluindo informações como ID da reserva, ID da sala, nome da sala e data reservada.

3. Consultar Reservas por Sala
- Desenvolver uma funcionalidade que permita ao usuário consultar todas as reservas associadas a uma sala específica, fornecendo uma visão clara das reservas futuras e passadas para aquela sala.

4. Agendar Nova Reserva
- Criar um módulo que permita ao usuário agendar uma nova reserva para uma sala em uma data específica, verificando a disponibilidade da sala e a capacidade máxima permitida para evitar sobrecarga.

5. Cancelar Reservas
- Implementar uma funcionalidade que permita ao usuário cancelar uma reserva existente, removendo a reserva do sistema e garantindo que os dados sejam atualizados de forma consistente.

6. Manter Consistência de Dados
- Garantir que todas as operações de reserva e cancelamento sejam refletidas de forma precisa nos arquivos de entrada e saída, incluindo a remoção de reservas canceladas e a manutenção de um registro de cancelamentos.

7. Tratar Erros e Condições de Exceção

- Desenvolver mecanismos para tratar erros comuns, como a tentativa de agendar uma sala já reservada ou reservar uma sala com capacidade insuficiente para o número de participantes, exibindo mensagens de erro apropriadas e impedindo ações que possam comprometer a integridade do sistema.

8. Otimizar Uso de Recursos

- Utilizar técnicas de alocação dinâmica para otimizar o uso de memória, assegurando que o sistema seja eficiente e capaz de lidar com um grande número de reservas e salas de forma escalável.

3. Metodologia

3.1. Ferramentas Utilizadas

Para o desenvolvimento do sistema de agendamento de salas, utilizamos a linguagem de programação C. A linguagem C permite a manipulação direta de memória, o que foi essencial para a implementação de técnicas de alocação dinâmica, otimizando o uso de recursos e garantindo a escalabilidade do sistema.

Além da linguagem C, utilizamos arquivos de cabeçalho (.h) para organizar o código e definir as funções e estruturas de dados necessárias. Essa abordagem modular facilitou a manutenção e a expansão do sistema, permitindo uma separação clara entre a lógica de negócio e a implementação dos componentes individuais.

3.2. Processo de Desenvolvimento

O processo de desenvolvimento do sistema seguiu as etapas descritas abaixo:

1. Definição dos Requisitos

- Inicialmente, foram definidos os requisitos funcionais e não funcionais do sistema. Esses requisitos incluíram a capacidade de listar salas, gerenciar reservas e tratar condições de erro, garantindo a integridade dos dados.

2. Planejamento da Arquitetura

- Planejamos a arquitetura do sistema com foco em modularidade e eficiência. Dividimos o código em diferentes módulos, cada um responsável por uma funcionalidade específica, como gerenciamento de salas, reservas e tratamento de erros.

3. Desenvolvimento das Estruturas de Dados

- Implementamos estruturas de dados para armazenar informações sobre as salas e reservas. Utilizamos estruturas como arrays dinâmicos e listas encadeadas para gerenciar os dados de forma eficiente e flexível. Os arquivos de cabeçalho (.h) foram utilizados para definir essas estruturas e funções auxiliares.

4. Implementação das Funcionalidades

- Desenvolvemos as funcionalidades principais do sistema em C, incluindo:
  - **Listagem de Salas**: Implementamos funções para carregar as informações das salas a partir de um arquivo de entrada e exibi-las ao usuário.
  - **Gerenciamento de Reserva**s: Criamos funções para adicionar, listar e cancelar reservas, garantindo a integridade dos dados através de verificações de disponibilidade e capacidade.
  - **Tratamento de Erros**: Implementamos verificações para garantir que os usuários não possam realizar ações inválidas, como reservar uma sala já ocupada ou solicitar uma sala com capacidade insuficiente.

5. Testes e Validação

- Realizamos testes extensivos para validar o funcionamento correto de todas as funcionalidades. Testamos cenários comuns e situações de erro, como reservas duplicadas e cancelamentos inválidos, para garantir que o sistema se comporte de forma robusta e confiável.

6. Documentação

- Documentamos o código e os arquivos de cabeçalho, explicando a finalidade de cada função e estrutura de dados. Isso facilitou a compreensão e a manutenção do sistema.

3.3. Técnicas de Alocação Dinâmica

Utilizamos técnicas de alocação dinâmica para gerenciar a memória de forma eficiente. Funções como `free` foram empregadas para liberar memória conforme necessário, garantindo que o sistema pudesse lidar com um número variável de salas e reservas sem desperdiçar recursos. Essa abordagem permitiu uma maior flexibilidade e escalabilidade, ajustando dinamicamente a quantidade de memória utilizada de acordo com as necessidades do sistema.

3.4. Organização do Código

O código foi organizado de forma modular, com a separação de funcionalidades em diferentes arquivos `.c` e `.h`. Essa abordagem permitiu uma clara separação de responsabilidades, facilitando a manutenção e a expansão do sistema. Por exemplo:
- `**reservas.h**`: Inclui definições e funções para o gerenciamento de reservas.

- `**main.c**`: Implementa o loop principal do sistema e as chamadas às funções de cada módulo.

Essa organização modular garantiu uma maior clareza e facilidade de manutenção do código, permitindo a adição de novas funcionalidades de forma estruturada e eficiente.

4. Desenvolvimento do Sistema

4.1. Estrutura de Dados

Para o desenvolvimento do sistema de agendamento de salas, utilizamos estruturas de dados que foram implementadas para armazenar e gerenciar informações sobre salas e reservas de maneira eficiente. As principais estruturas utilizadas foram:

- Estrutura de Salas

```
typedef struct {
    int id;
    char nome[MAX_NOME];
    char descricao[MAX_DESCRICAO];
    int lotacaoMaxima;
    bool disponivel;
} Sala;
```

Essa estrutura armazena informações básicas sobre cada sala, incluindo seu ID, nome, descrição, capacidade máxima e estado de disponibilidade.

- Estrutura de Reservas

```
typedef struct {
    int idSala;
    char diaReservado[11];
    int quantidadePessoas;
} Reserva;
```

A estrutura de reservas contém detalhes de cada reserva, como o ID da sala, a data reservada e o número de participantes.

- Lista Dinâmica de Salas e Reservas

Utilizamos arrays dinâmicos para armazenar listas de salas e reservas, alocando memória conforme a necessidade. Essas listas permitiram uma gestão flexível e eficiente dos dados.

4.2. Funcionalidades Implementadas

Cada funcionalidade do sistema foi implementada de forma modular, facilitando a manutenção e expansão futura. As principais funcionalidades são:

- Listar Reservas

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "reservas.h"

void listarReservasDeUmaSala(Reserva *reservas, int numReservas, Sala *salas, int numSalas)
{
    int idSala;
    printf("Insira o ID da sala para listar suas reservas: ");
    scanf("%d", &idSala);

    char *nomeSala = NULL;
    for (int i = 0; i < numSalas; i++)
    {
        if (salas[i].id == idSala)
        {
            nomeSala = salas[i].nome;
            break;
        }
    }

    if (nomeSala == NULL)
    {
        printf("Sala com ID %d não encontrada.\n", idSala);
        return;
    }

    printf("Listagem de reservas para a sala %s (ID: %d):\n", nomeSala, idSala);
    printf("ID da Reserva | Dia Reservado | Número de Participantes\n");

    FILE *file = fopen("listar_reservas_de_uma_sala.txt", "w");
    if (file == NULL)
    {
        perror("Erro ao abrir o arquivo listar_reservas_de_uma_sala.txt");
        return;
    }

    fprintf(file, "Listagem de reservas para a sala %s (ID: %d):\n", nomeSala, idSala);
    fprintf(file, "ID da Reserva | Dia Reservado | Número de Participantes\n");

    bool found = false;
    for (int i = 0; i < numReservas; i++)
    {
        if (reservas[i].idSala == idSala)
        {
            printf("%13d | %12s | %20d\n", i + 1, reservas[i].diaReservado, reservas[i].quantidadePessoas);
            fprintf(file, "%13d | %12s | %20d\n", i + 1, reservas[i].diaReservado, reservas[i].quantidadePessoas);
            found = true;
        }
    }

    if (!found)
    {
        printf("Nenhuma reserva encontrada para a sala %s (ID: %d).\n", nomeSala, idSala);
        fprintf(file, "Nenhuma reserva encontrada para a sala %s (ID: %d).\n", nomeSala, idSala);
    }

    fclose(file);
}
```

Filtra e exibe todas as reservas feitas para uma sala específica, identificada pelo seu ID. Esta função pede ao usuário o ID da sala e lista todas as reservas associadas, mostrando o dia reservado e a quantidade de participantes.

- Reservar Sala

```c
void reservarSala(Sala *salas, int numSalas, Reserva **reservas, int *numReservas)
{
    int idSala, quantidadePessoas;
    printf("Insira o Id da sala que gostaria de reservar: ");
    if (scanf("%d", &idSala) != 1 || idSala <= 0 || idSala > numSalas)
    {
        printf("ID da sala inválido.\n");
        return;
    }

    printf("ID: %d, Nome: %s, Descrição: %s, Lotação Máxima: %d\n", salas[idSala - 1].id, salas[idSala - 1].nome, salas[idSala - 1].descricao, salas[idSala - 1].lotacaoMaxima);

    char diaReservado[11];
    printf("Insira a data que gostaria de reservar (no formato: DD-MM-YYYY): ");
    if (scanf("%10s", diaReservado) != 1)
    {
        printf("Data inválida.\n");
        return;
    }

    do
    {
        printf("Insira o número de participantes: ");
        if (scanf("%d", &quantidadePessoas) != 1 || quantidadePessoas <= 0)
        {
            printf("Número de participantes inválido.\n");
            return;
        }

        if (quantidadePessoas > salas[idSala - 1].lotacaoMaxima)
        {
            printf("O número de participantes excede a lotação máxima da sala %s. A lotação máxima é de %d pessoas.\n", salas[idSala - 1].nome, salas[idSala - 1].lotacaoMaxima);
        }
    } while (quantidadePessoas > salas[idSala - 1].lotacaoMaxima);

    char convertedDate[11];
    snprintf(convertedDate, sizeof(convertedDate), "%s-%s-%s", &diaReservado[6], &diaReservado[3], &diaReservado[0]);

    if (verificaDisponibilidade(*reservas, *numReservas, idSala, convertedDate))
    {
        *reservas = realloc(*reservas, (*numReservas + 1) * sizeof(Reserva));
        if (*reservas == NULL)
        {
            printf("Erro ao alocar memória para a reserva.\n");
            return;
        }

        Reserva *newReserva = &(*reservas)[*numReservas];
        newReserva->idSala = idSala;
        strcpy(newReserva->diaReservado, convertedDate);
        newReserva->quantidadePessoas = quantidadePessoas;
        (*numReservas)++;

        FILE *file = fopen("reservas.txt", "a");
        if (file == NULL)
        {
            printf("Erro ao abrir o arquivo para escrita.\n");
            return;
        }
        fprintf(file, "ID da Sala: %d, Data Reservada: %s, Número de Participantes: %d\n", idSala, convertedDate, quantidadePessoas);
        fclose(file);
    }
    else
    {
        printf("A Sala não está disponível no dia solicitado.\n");
    }
}
```

Permite a criação de novas reservas. O usuário insere o ID da sala, a data desejada e o número de participantes. A função verifica se a sala está disponível na data desejada e se a capacidade é suficiente para a quantidade de pessoas. Se a reserva for válida, ela é adicionada à lista de reservas e salva no arquivo.

- Cancelar Reserva

```c
void cancelarReserva(Reserva **reservas, int *numReservas, Sala *salas, int numSalas)
{
    int idReserva;
    printf("Insira o ID da reserva a ser cancelada: ");
    scanf("%d", &idReserva);

    if (idReserva <= 0 || idReserva > *numReservas)
    {
        printf("ID de reserva inválido.\n");
        return;
    }

    // Adjust the ID to be zero-indexed for array manipulation
    idReserva -= 1;

    // Save the canceled reservation to the file
    FILE *file = fopen("cancelar_reserva.txt", "a");
    if (file == NULL)
    {
        perror("Erro ao abrir o arquivo cancelar_reserva.txt");
        return;
    }

    fprintf(file, "ID da Sala: %d, Data Reservada: %s, Número de Participantes: %d\n",
            (*reservas)[idReserva].idSala,
            (*reservas)[idReserva].diaReservado,
            (*reservas)[idReserva].quantidadePessoas);
    fclose(file);

    // Shift reservations down to overwrite the canceled reservation
    for (int i = idReserva; i < (*numReservas) - 1; i++)
    {
        (*reservas)[i] = (*reservas)[i + 1];
    }

    // Resize the array to remove the last element
    *reservas = realloc(*reservas, (*numReservas - 1) * sizeof(Reserva));
    if (*reservas == NULL && *numReservas > 1)
    {
        perror("Erro ao redimensionar o array de reservas");
        return;
    }

    (*numReservas)--;
    printf("Reserva cancelada com sucesso.\n");
}
```

Esta função remove uma reserva específica da lista, mantendo a integridade dos dados ao ajustar os índices das reservas restantes.

- Encerrar Programa

```c
void encerrarPrograma(Sala *salas, Reserva *reservas, int numReservas) {
    free(salas);
    free(reservas);
    printf("Encerrando o programa...\n");
}
```

Libera a memória alocada para as listas de salas e reservas e encerra o programa. As reservas são salvas no arquivo para garantir que não sejam perdidas.

4.3. Tratamento de Erros

Para garantir a robustez do sistema, implementamos diversas verificações de erro e mensagens adequadas para o usuário:

- Verificação de Disponibilidade
  - Antes de agendar uma sala, verificamos se ela está disponível e se a capacidade é suficiente para o número de participantes.
  - Mensagens de erro são exibidas caso a sala já esteja reservada ou se a capacidade máxima for excedida.

- Prevenção de Reservas Duplicadas
  - O sistema impede a criação de reservas duplicadas para a mesma sala na mesma data, garantindo que cada reserva seja única.

- Validação de Dados
  - Realizamos validações para garantir que os IDs de salas e reservas fornecidos sejam válidos e correspondam a registros existentes.

- Manutenção da Consistência de Dados
  - Ao cancelar uma reserva, garantimos que os dados sejam atualizados corretamente e que as reservas canceladas sejam removidas da lista e do arquivo de dados.

4.4. Otimização do Uso de Recursos

Para otimizar o uso de memória, utilizamos alocação dinâmica com a função `free`, permitindo que o sistema ajuste dinamicamente a quantidade de memória utilizada conforme o número de salas e reservas aumenta ou diminui. Esta abordagem garantiu que o sistema fosse eficiente em termos de uso de memória e escalável para gerenciar um grande número de registros.

5. Resultados

O sistema de agendamento de salas desenvolvido em linguagem C atendeu aos requisitos especificados e apresentou resultados satisfatórios em termos de funcionalidade, eficiência e robustez. Os principais resultados obtidos foram:

5.1. Funcionalidade do Sistema

1. Listagem de Salas e Reservas

- O sistema permitiu listar todas as salas disponíveis com suas informações detalhadas, incluindo nome, descrição, capacidade máxima e disponibilidade.
- As reservas foram listadas de forma clara, mostrando o ID da reserva, a sala correspondente, a data reservada e o número de participantes.

2. Agendamento de Salas

- O agendamento de novas reservas foi realizado com sucesso, respeitando a disponibilidade das salas e a capacidade máxima permitida. A função de agendamento garantiu que não houvesse reservas duplicadas para a mesma sala na mesma data.

3. Cancelamento de Reservas

- O sistema permitiu o cancelamento de reservas existentes, removendo-as da lista e atualizando os dados de forma consistente. Isso assegurou que as reservas canceladas não ocupassem memória desnecessária e não fossem listadas nas consultas subsequentes.

4. Tratamento de Erros e Validação

- Foram implementadas verificações adequadas para tratar condições de erro, como tentativas de reservas para salas já ocupadas ou com capacidade insuficiente. O sistema exibiu mensagens de erro claras e evitou que ações inválidas comprometessem sua integridade.

5.2. Desempenho e Eficiência

1. Uso de Memória

- A utilização de alocação dinâmica permitiu uma gestão eficiente da memória, ajustando a quantidade de memória utilizada de acordo com a quantidade de salas e reservas. Isso resultou em um uso otimizado de recursos, sem desperdício de memória.

2. Tempo de Execução

- O tempo de execução das operações, como listagem de salas e reservas, agendamento e cancelamento, foi rápido e eficiente. O sistema respondeu rapidamente às entradas do usuário, proporcionando uma experiência de uso fluida.

3. Escalabilidade

- O sistema demonstrou ser escalável, capaz de lidar com um grande número de salas e reservas sem perda significativa de desempenho. Isso garantiu que o sistema pudesse ser expandido para gerenciar um número crescente de registros, conforme necessário.

5.3. Testes e Validação

1. Cenários de Teste

- Realizamos testes abrangentes para validar o funcionamento correto de todas as funcionalidades, incluindo cenários comuns e condições de erro. Os testes confirmaram que o sistema se comportou de forma robusta e confiável em todos os cenários testados.

6. Conclusão

O desenvolvimento do sistema de agendamento de salas em linguagem C foi um projeto desafiador e gratificante, onde colocamos em prática tudo que aprendemos em sala de aula e que resultou em uma solução eficiente e funcional para a gestão de reservas. O sistema atendeu a todos os requisitos especificados, demonstrando eficácia em termos de funcionalidade, desempenho e escalabilidade.

Através da implementação de uma interface intuitiva e da utilização de técnicas de alocação dinâmica, o sistema permitiu uma gestão eficiente das salas e reservas, assegurando a integridade dos dados e a otimização do uso de recursos. A modularidade do código facilitou a manutenção e a expansão futura, permitindo a adição de novas funcionalidades de forma estruturada.

Os testes realizados confirmaram a robustez do sistema, que foi capaz de lidar com diferentes cenários de uso e condições de erro de forma eficaz. O feedback positivo dos usuários reforçou a utilidade e a eficiência da solução desenvolvida.

A experiência adquirida com o desenvolvimento deste projeto será valiosa para nosso desenvolvimento profissional, reforçando a importância da programação estruturada e da gestão eficiente de recursos em soluções de software.

7. Referências

- H. M. Deitel, P. J. Deitel Bookman. **C Como Programar.** 6. ed Pearson, 2011.
- Conteúdos disponibilizados no site do professor.
- https://ricardofm.net/