Relatório Técnico: Sistema de Gerenciamento de Serviços em Linguagem C (1° entrega)

Leonardo Bora

Luan Constancio

Acesso à planilha do Sheets contendo o cronograma do projeto:

https://docs.google.com/spreadsheets/d/1kYhTEVBSvIlk-dECUtkq14qXo_MqV_II_LgNd1rWrx8/edit?usp=sharing

```
---
config:
  theme: default
---
gantt
    title Cronograma do Projeto: Aplicativo de Gerenciamento de Serviços
    dateFormat  YYYY-MM-DD
    axisFormat  %d/%m
    excludes weekends
    section Planejamento
    Definir requisitos e escopo do projeto       :done, 2025-04-01, 3d
    Elaborar cronograma e gráfico de Gantt       :done, 2025-04-04, 3d
    Revisão inicial do planejamento              :done, 2025-04-07, 2d
    section Implementação (Primeira Entrega)
    Estruturar listas duplamente encadeadas      :active, 2025-04-09, 7d
    Implementar cadastro de serviços             :2025-04-18, 5d
    Desenvolver listas por status                :2025-04-23, 4d
    Testar funcionalidades básicas               :2025-04-29, 3d
    section Relatório Técnico (Primeira Entrega)
    Redigir introdução ao problema               :done, 2025-04-01, 3d
    Fundamentação teórica                        :active, 2025-04-04, 4d
    Metodologia e descrição das estruturas       :2025-04-10, 4d
    Revisão e finalização do relatório           :2025-04-16, 2d
    Entrega 1 completa                           :milestone, 2025-04-18, 0d
    section Segunda Entrega (Segunda Etapa)
    Implementar algoritmos de ordenação          :crit, 2025-05-01, 7d
    Desenvolver controle de atendimento          :crit, 2025-05-08, 6d
    Análise de desempenho                        :2025-05-16, 5d
    Integração das funcionalidades               :2025-05-23, 6d
    Testes finais                                :2025-05-30, 7d
    Entrega 2 completa                           :milestone, 2025-06-09, 0d
```

O gerenciamento eficiente de serviços em ambientes corporativos exige estruturas dinâmicas capazes de lidar com operações complexas de inserção, remoção e atualização de estados. O sistema proposto visa resolver três desafios principais:

1. **Controle de status dinâmico**: Serviços transitam entre "Pendente", "Em Execução" e "Concluído", exigindo reorganização constante das estruturas.
2. **Ordenação por prioridade**: Necessidade de implementar algoritmos que classifiquem serviços em tempo real conforme critérios de urgência ('A', 'M', 'B').
3. **Gestão de memória**: Uso intensivo de alocação dinâmica requer mecanismos robustos para evitar vazamentos e corrupção de dados.

O projeto utiliza **listas duplamente encadeadas** para armazenamento principal e **tabelas hash** para acesso rápido a registros, integrando algoritmos de ordenação (BubbleSort e MergeSort) para otimização de consultas.

Fundamentação Teórica

1. Estruturas de Dados Lineares

Listas Duplamente Encadeadas: Permitem navegação bidirecional, ideal para operações de inserção/remoção em qualquer posição com complexidade O(1) para atualizações locais. Cada nó contém:

```
struct Servico {
    int id;
    char status; // 'P', 'E', 'C'
    char prioridade;
    struct Servico *ant;
    struct Servico *prox;
};
```

Pilhas (LIFO): Usadas para histórico de operações reversíveis, com funções `push()` e `pop()` baseadas em ponteiro `topo`.

Filas (FIFO): Gerenciam solicitações em ordem de chegada, utilizando ponteiros `frente` e `fim.`

2. Algoritmos de Ordenação

BubbleSort:

- Compara pares adjacentes, movendo o maior elemento para o final em cada iteração.
- Complexidade $O(n^2)$, adequado para listas pequenas (<1000 elementos).

```
void bubbleSort(Servico *cabeca) {
    int trocado;
    Servico *ptr1;
    Servico *lptr = NULL;
    do {
        trocado = 0;
        ptr1 = cabeca;
        while (ptr1->prox != lptr) {
            if (ptr1->prioridade > ptr1->prox->prioridade) {
                trocaNos(ptr1, ptr1->prox);
                trocado = 1;
            }
            ptr1 = ptr1->prox;
        }
        lptr = ptr1;
    } while (trocado);
}
```

MergeSort:

- Divide a lista recursivamente e intercala sublistas ordenadas.
- Complexidade O(n log n), preferível para grandes datasets.

3. Técnicas de Pesquisa

Busca Binária: Exige lista ordenada, reduzindo tempo de busca de O(n) para O(log n) mediante cálculo de índices via:

meio=inı´cio+(fim−inı´cio)/2meio=inı´cio+(fim−inı´cio)/2

Tabelas Hash: Mapeamento direto via função modular `h(k) = k % m`, usando listas encadeadas para resolver colisões.

Metodologia de Desenvolvimento

1. Abordagem Iterativa

O projeto segue o modelo **incremental**, dividido em 3 sprints de 2 semanas:

- **Sprint 1**: Implementação básica das listas e funções CRUD (Create, Read, Update, Delete).
- **Sprint 2**: Integração dos algoritmos de ordenação e testes de estresse.

- **Sprint 3**: Otimização de memória e documentação final[1][2].

2. Ferramentas Utilizadas

- **Git**: Controle de versões com branches `main` (estável) e `dev` (desenvolvimento).
- **Valgrind**: Análise de vazamentos de memória e erros de alocação.
- **Doxygen**: Geração automática de documentação a partir de comentários no código[3][4].

3. Protocolo de Testes

Caso de Teste	Entrada	Saída Esperada
Inserção 10k serviços	IDs sequenciais	Lista ordenada por ID
Transição de status	500 serviços 'P'→'E'	Listas atualizadas
Stress MergeSort	1M elementos aleatórios	Tempo < 2s

Descrição das Estruturas de Dados

1. Arquitetura Principal

Diagrama de Estruturas

- **Lista Principal**: Armazena todos os serviços com ponteiros `ant` / `prox`.
- **Sublistas por Status**: Três listas independentes aceleram consultas frequentes.
- **Tabela Hash**: Array de 1000 buckets para acesso O(1) a serviços via ID[1][2].

2. Funções Críticas

Inserção com Prioridade:

```
void inserePrioridade(Servico **cabeca, int id, char pri) {
    Servico *novo = malloc(sizeof(Servico));
    // ... inicialização
    if (*cabeca == NULL || pri > (*cabeca)->prioridade) {
        novo->prox = *cabeca;
        *cabeca = novo;
    } else {
```

```
        Servico *atual = *cabeca;
        while (atual->prox != NULL && atual->prox->prioridade >= pri) {
            atual = atual->prox;
        }
        novo->prox = atual->prox;
        atual->prox = novo;
    }
}
```

Atualização de Status:

```
void mudaStatus(Servico **origem, Servico **destino, int id) {
    Servico *temp = *origem, *prev = NULL;
    while (temp != NULL && temp->id != id) {
        prev = temp;
        temp = temp->prox;
    }
    if (temp == NULL) return;
    if (prev != NULL) prev->prox = temp->prox;
    else *origem = temp->prox;
    temp->prox = *destino;
    *destino = temp;
}
```

3. Otimizações

- **Pool de Memória**: Pré-alocação de 1000 nós para reduzir chamadas a `malloc()`.
- **Cache LRU**: Mantém os 100 serviços mais recentes em array para acesso rápido[3][4].