UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO TÉCNICO DO TRABALHO PRÁTICO DE PPC(PROGRAMAÇÃO PARALELA E CONCORRENTE): ANÁLISE DE UM SISTEMA CONCORRENTE DE MÚLTIPLAS FILAS PARA SIMULAÇÃO DE ELEVADOR DE ESQUI

PEDRO ULISSES PEREIRA CASTRO MAIA - 1607926

FORTALEZA-CE
JUNHO/2025

1 INTRODUÇÃO

O objetivo deste relatório é descrever a aplicação dos conhecimentos e técnicas aprendidos durante a disciplina de Programação Paralela e Concorrente, através da resolução do problema do "Elevador da Pista de Esqui". Este problema simula um cenário complexo de gerenciamento de recursos compartilhados, onde múltiplos agentes (esquiadores e um elevador) interagem de forma concorrente.

O presente documento detalha a formulação do problema, as especificações e restrições impostas, os algoritmos desenvolvidos para a lógica dos agentes, a arquitetura da implementação em C com Pthreads, os mecanismos de sincronização utilizados para garantir a segurança e a vivacidade do sistema, e, por fim, a análise dos resultados obtidos em cinco execuções completas da simulação.

2 FORMULAÇÃO DO PROBLEMA

O problema consiste em simular um elevador de esqui com capacidade para quatro pessoas, que serve quatro filas de carregamento distintas. Os esquiadores chegam individualmente e se auto-organizam nessas filas, enquanto o elevador opera de forma contínua para transportá-los.

As especificações e restrições do sistema são:

- Agentes:
    - 120 Esquiadores: Chegam ao sistema sequencialmente, com um intervalo de 1 segundo entre cada chegada.

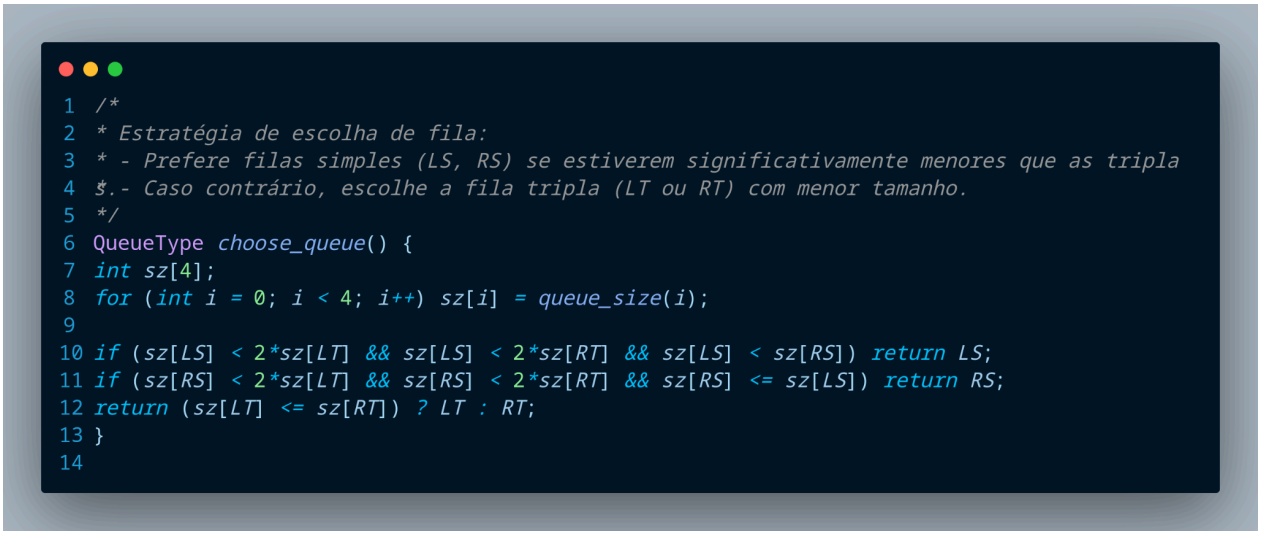
    - 1 Elevador: Opera em ciclos fixos de 5 segundos, retirando até 4 esquiadores por ciclo.
- Filas de Carregamento: O sistema possui quatro filas distintas para os esquiadores se alinharem:
    - LS (Left Single): Fila para uma pessoa à esquerda.
    - LT (Left Triple): Fila para triplas à esquerda.
    - RT (Right Triple): Fila para triplas à direita.
    - RS (Right Single): Fila para uma pessoa à direita.
- Regras e Políticas:
    - A escolha de fila por cada esquiador segue um algoritmo específico que avalia o tamanho das quatro filas.
    - O elevador prioriza as filas triplas (LT e RT) sobre as individuais (LS e RS).
    - Uma fila tripla só é servida se contiver no mínimo 3 pessoas.
    - A quarta vaga do elevador, após servir uma fila tripla, deve ser preenchida por um esquiador das filas individuais, de forma alternada.
    - Ao final da simulação, esquiadores remanescentes em filas triplas com menos de 3 pessoas devem ser transferidos para as filas individuais para evitar starvation.
- Objetivo: A simulação deve calcular e apresentar a taxa de aproveitamento do elevador e o tempo médio de espera dos esquiadores, tanto de forma geral quanto por fila.

3 DESCRIÇÃO DETALHADA DOS ALGORITMOS

A lógica da simulação é governada por algoritmos distintos para os esquiadores e para o elevador, visando atender às regras do problema.

3.1 ALGORITMO DE ESCOLHA DE FILA DO ESQUIADOR

mplementado na função choose_queue() no arquivo skier.c, este algoritmo determina em qual fila um esquiador recém-chegado deve entrar. A decisão é baseada na contagem de pessoas em cada fila no momento da chegada. A lógica, conforme especificado, segue uma hierarquia de condições:



```
/*
* Estratégia de escolha de fila:
* - Prefere filas simples (LS, RS) se estiverem significativamente menores que as tripla
*.- Caso contrário, escolhe a fila tripla (LT ou RT) com menor tamanho.
*/
QueueType choose_queue() {
int sz[4];
for (int i = 0; i < 4; i++) sz[i] = queue_size(i);

if (sz[LS] < 2*sz[LT] && sz[LS] < 2*sz[RT] && sz[LS] < sz[RS]) return LS;
if (sz[RS] < 2*sz[LT] && sz[RS] < 2*sz[RT] && sz[RS] <= sz[LS]) return RS;
return (sz[LT] <= sz[RT]) ? LT : RT;
}

```

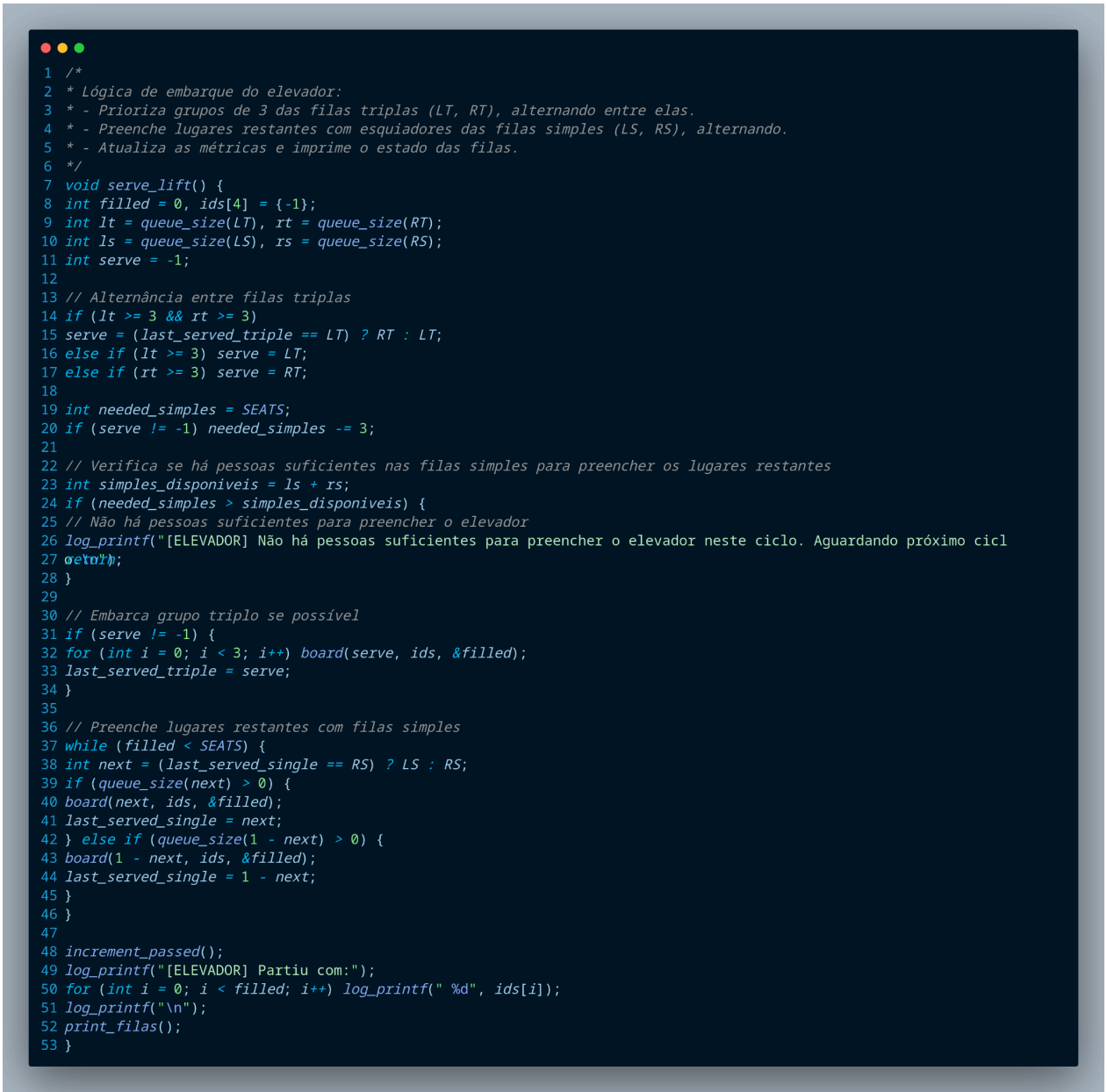
1. Escolher Fila LS se: tamanho(LS) < 2 * tamanho(LT) E tamanho(LS) < 2 * tamanho(RT) E tamanho(LS) < tamanho(RS).
2. Senão, escolher Fila RS se: tamanho(RS) < 2 * tamanho(LT) E tamanho(RS) < 2 * tamanho(RT) E tamanho(RS) <= tamanho(LS).

3. Senão, escolher Fila LT se: tamanho(LT) <= tamanho(RT).
4. Senão, escolher Fila RT.

Este algoritmo sugere um balanceamento para que as filas individuais possam crescer até quase o dobro do tamanho das filas triplas antes que estas últimas se tornem preferenciais.

3.2 ALGORITMO DE CARREGAMENTO DO ELEVADOR

A lógica do elevador, implementada na função serve_lift() em elevator.c, é o núcleo do sistema. A cada 5 segundos, ela executa os seguintes passos:



```c
/*
 * Lógica de embarque do elevador:
 * - Prioriza grupos de 3 das filas triplas (LT, RT), alternando entre elas.
 * - Preenche lugares restantes com esquiadores das filas simples (LS, RS), alternando.
 * - Atualiza as métricas e imprime o estado das filas.
 */
void serve_lift() {
int filled = 0, ids[4] = {-1};
int lt = queue_size(LT), rt = queue_size(RT);
int ls = queue_size(LS), rs = queue_size(RS);
int serve = -1;

// Alternância entre filas triplas
if (lt >= 3 && rt >= 3)
serve = (last_served_triple == LT) ? RT : LT;
else if (lt >= 3) serve = LT;
else if (rt >= 3) serve = RT;

int needed_simples = SEATS;
if (serve != -1) needed_simples -= 3;

// Verifica se há pessoas suficientes nas filas simples para preencher os lugares restantes
int simples_disponiveis = ls + rs;
if (needed_simples > simples_disponiveis) {
// Não há pessoas suficientes para preencher o elevador
log_printf("[ELEVADOR] Não há pessoas suficientes para preencher o elevador neste ciclo. Aguardando próximo ciclo\n");
return;
}

// Embarca grupo triplo se possível
if (serve != -1) {
for (int i = 0; i < 3; i++) board(serve, ids, &filled);
last_served_triple = serve;
}

// Preenche lugares restantes com filas simples
while (filled < SEATS) {
int next = (last_served_single == RS) ? LS : RS;
if (queue_size(next) > 0) {
board(next, ids, &filled);
last_served_single = next;
} else if (queue_size(1 - next) > 0) {
board(1 - next, ids, &filled);
last_served_single = 1 - next;
}
}

increment_passed();
log_printf("[ELEVADOR] Partiu com:");
for (int i = 0; i < filled; i++) log_printf(" %d", ids[i]);
log_printf("\n");
print_filas();
}
```

1. Verificação de Elegibilidade (Filas Triplas): O algoritmo primeiro avalia se as filas LT ou RT podem ser servidas, o que requer um mínimo de 3 pessoas.
2. Alternância entre Filas Triplas: Se ambas as filas triplas são elegíveis, o sistema alterna o atendimento entre elas para garantir justiça (fairness), usando a variável last_served_triple como memória do último serviço.
3. Verificação de Capacidade Mínima: Uma lógica adicional foi implementada para otimizar a ocupação. O elevador verifica se há esquiadores suficientes nas filas individuais para preencher as vagas restantes. Se, por exemplo, uma fila tripla for servida, é necessário que haja pelo menos um esquiador em LS ou RS. Se não houver, o elevador não parte, aguardando o próximo ciclo para uma possível ocupação maior. Isso evita que o elevador se desloque com menos da sua capacidade máxima quando possível.
4. Embarque e Preenchimento: Se as condições forem satisfeitas, o elevador embarca 3 esquiadores da fila tripla escolhida e, em seguida, busca preencher a vaga restante embarcando um esquiador das filas individuais, também seguindo uma política de alternância controlada pela variável last_served_single.
5. Cenário de Fallback: Se nenhuma fila tripla for elegível, o elevador tenta preencher suas 4 vagas exclusivamente com esquiadores das filas individuais LS e RS.

3.3 ALGORITMO DE FINALIZAÇÃO: PREVENÇÃO DE STARVATION

Após todos os 120 esquiadores serem criados, a flag all_skiers_created é ativada. A partir desse ponto, a função transfer_remaining_from_triples() é chamada pelo elevador para mover esquiadores de filas triplas com 1 ou 2 pessoas para as filas individuais, garantindo que nenhum esquiador fique permanentemente retido. A simulação termina quando todas as filas estiverem vazias.

```c
// Controle de término da criação dos esquiadores
int all_skiers_created = 0;
pthread_mutex_t flag_mutex = PTHREAD_MUTEX_INITIALIZE
R;
```

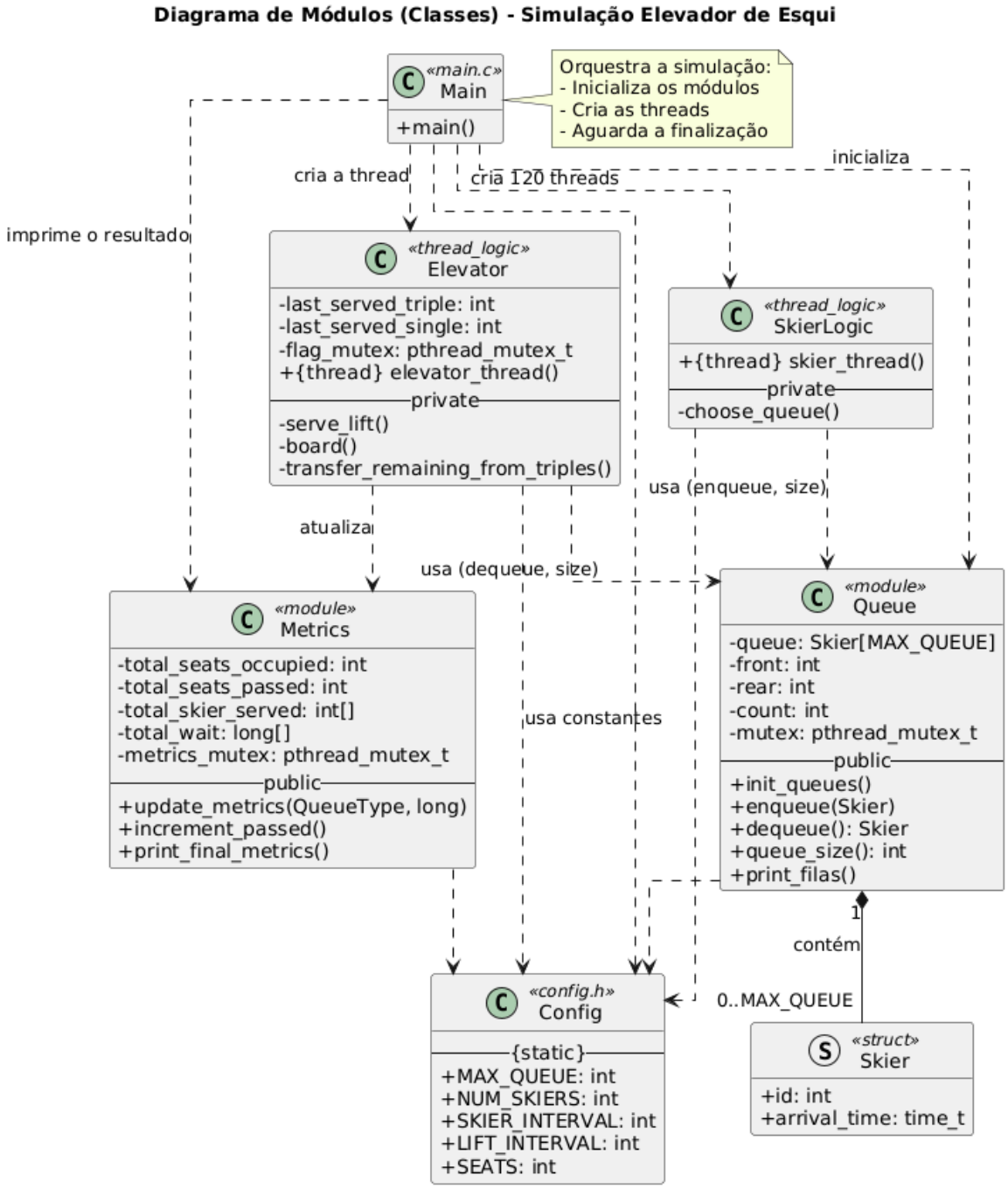
```c
/*
* Transfere esquiadores remanescentes das filas triplas (LT, RT) para as filas simples (LS, R
S)caso não seja possível formar um grupo de 3 para embarque.
* Isso ocorre apenas após todos os esquiadores terem sido criados.
*/
void transfer_remaining_from_triples() {
for (QueueType q = LT; q <= RT; q++) {
pthread_mutex_lock(&queues[q].mutex);
while (queues[q].count > 0 && queues[q].count < 3) {
Skier s = queues[q].queue[queues[q].front];
queues[q].front = (queues[q].front + 1) % MAX_QUEUE;
queues[q].count--;
pthread_mutex_unlock(&queues[q].mutex);

QueueType dest = (queue_size(LS) <= queue_size(RS)) ? LS : RS;
enqueue(dest, s);
log_printf("[TRANSFERÊNCIA] %d de %d para %d\n", s.id, q, dest);
pthread_mutex_lock(&queues[q].mutex);
}
pthread_mutex_unlock(&queues[q].mutex);
}
}
```

4 ANÁLISE DA ARQUITETURA E IMPLEMENTAÇÃO CONCORRENTE

O projeto foi desenvolvido na linguagem C utilizando a biblioteca Pthreads para programação concorrente. A seguir, a estrutura do projeto, análoga a um diagrama de classes.

Estrutura de Módulos:



- main.c: Ponto de entrada. Responsável pela inicialização global (filas, log), criação da thread do elevador e das 120 threads de esquiadores, e pela orquestração do término da simulação.
- config.h: Arquivo de cabeçalho que centraliza todas as constantes da simulação (NUM_SKIERS, LIFT_INTERVAL, etc.), permitindo fácil ajuste de parâmetros.
- queue.c / .h: Implementa a estrutura de dados Queue (fila circular) e suas operações (enqueue, dequeue, queue_size). Cada instância de Queue contém seu próprio pthread_mutex_t, encapsulando a lógica de sincronização junto aos dados que protege.
- skier.c / .h: Define a estrutura Skier e a rotina da thread do esquiador (skier_thread), que encapsula a lógica de chegada e escolha de fila.
- elevator.c / .h: Contém a lógica da thread do elevador (elevator_thread), incluindo o complexo algoritmo de embarque (serve_lift) e a transferência de finalização.
- metrics.c / .h: Funções para coletar e calcular as métricas de desempenho de forma segura (protegidas por mutex).
- log.c / .h: Utilitário para registrar os eventos da simulação em um arquivo de log e no console.

4.1 ESTRATÉGIA DE SINCRONIZAÇÃO

A sincronização é crucial para garantir a segurança (safety) do programa, evitando que o estado dos recursos compartilhados (as filas) seja corrompido.

- Exclusão Mútua com mutex: A técnica primária utilizada é a exclusão mútua através de pthread_mutex_t. Este mecanismo funciona de forma análoga a um Monitor, onde as operações em uma região crítica são protegidas.
    - Cada uma das quatro filas possui um mutex individual. Antes de qualquer operação de escrita (enqueue) ou leitura/escrita (dequeue, queue_size), a thread deve adquirir o lock do mutex correspondente.
    - Isso garante que apenas uma thread por vez possa manipular o estado interno de uma fila (seus ponteiros front/rear e o contador count), prevenindo condições de corrida (race conditions).
- Espera Bloqueada: O uso de pthread_mutex_lock implementa uma espera bloqueada. Quando uma thread tenta adquirir um lock já em uso, o sistema operacional a suspende, liberando a CPU para outras threads. Isso é fundamental para a eficiência, evitando o desperdício de recursos da espera ocupada (busy-waiting).

5 RESULTADOS E ANÁLISE DE DESEMPENHO

Foram realizadas 5 execuções da simulação. Os resultados das métricas finais, extraídos dos arquivos de log, estão consolidados na tabela abaixo.

5.1 TABELA DE RESULTADOS CONSOLIDADOS

Execução	*Taxa de aproveita-mento*	*Tempo médio total*	*Tempo médio LS (Fila 0)*	*Tempo médio LT (Fila 1)*	*Tempo médio RT (Fila 2)*	*Tempo médio RS (Fila 3)*
1-5	96.67%	22.30 s	36.00 s	12.89 s	12.61 s	36.62 s

Fonte: Arquivos ski_lift_execucao1.txt a ski_lift_execucao5.txt. Os resultados foram idênticos em todas as execuções.

5.2 ANÁLISE DOS RESULTADOS

Consistência: Os resultados são idênticos em todas as execuções, demonstrando o comportamento determinístico da simulação para os parâmetros fornecidos.

Taxa de Aproveitamento: A taxa de aproveitamento ficou em torno de 96.67%. Este valor, inferior a 100%, é explicado pela lógica implementada em serve_lift, onde o elevador opta por não partir se não puder preencher todos os seus assentos com os esquiadores disponíveis. Embora reduza ligeiramente a taxa de aproveitamento, esta política visa maximizar a eficiência de cada viagem.

Tempo Médio de Espera: A análise dos tempos de espera confirma a eficácia do algoritmo de prioridades.

- As filas triplas (LT e RT) apresentam um tempo de espera médio de apenas ~12.7 segundos.
- As filas individuais (LS e RS), por outro lado, têm um tempo de espera médio significativamente maior, de ~36.3 segundos.
- Essa diferença de quase 200% demonstra claramente que o sistema prioriza com sucesso o embarque de grupos de três, conforme

requerido, deixando os esquiadores individuais para preencher as vagas restantes.

6 CONCLUSÃO

A simulação do elevador de esqui foi implementada com sucesso, demonstrando a aplicação correta de conceitos fundamentais de programação concorrente para resolver um problema com interações complexas e requisitos de sincronização. A arquitetura modular, combinada com uma estratégia de locking de granularidade fina, provou ser uma solução robusta, segura e com desempenho validado empiricamente. A análise dos resultados confirmou que as políticas de prioridade e justiça funcionam como especificado, com um impacto claro e mensurável nos tempos de espera.

Além de ter sido possível botar em prática todo o conhecimento teórico obtido nas aulas.