RESPOSTA CHECAGEM_04 - JANTAR DOS FILÓSOFOS

- **Professor:** Luis Vinicius Costa Silva
- **Aluno:** João Vitor de Souza Gonçalves
- **Data de avaliação:** 24/09/2025

Para a execução da checagem foi fornecido um código escrito na linguagem C. Seu funcionamento faz com que seja resolvido o Problema do Jantar dos Filósofos utilizando MPI. Usando MPI, os processos são enviados com MPI_Send e recebidos com MPI_Recv. O ideal é que todos os filósofos comam sem interrupções ou problemas.

1. COMANDOS PARA COMPILAR E EXECUTAR O CÓDIGO

Com o código copiado ao WSL foi necessário instalar o MPI:

```
# Instalar OpenMPI
sudo apt install -y build-essential openmpi-bin openmpi-common libopenmpi-dev
```

E logo em seguida conseguimos rodar o código com os seguintes comandos:

```
# Colocar o código no WSL
nano filosofos.c

# Compilar o programa
mpicc -o filosofos filosofos.c

# Verificar se o binário foi criado
ls -l filosofos

# Executar o programa
mpirun --oversubscribe -np 5 ./filosofos
```

1.1 BUG ENCONTRADO E SOLUÇÃO

No código fornecido encontra-se o seguinte bug:

```
MPI_Send(&estado, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD);
MPI_Send(&estado, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD);



MPI_Recv(&estado_esq, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
MPI_Recv(&estado_dir, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

O problema está no fato de usar diretamente o MPI_Send seguido de MPI_Recv entre processos vizinhos. Isso acaba podendo gerar deadlock porque todos os filósofos tentam enviar ao mesmo tempo com MPI_Send, sem nenhum processo pronto para receber.

Para solucionar o problema e evitar o deadlock, foi necessário alterar certos trechos do código fornecido. Foi utilizado MPI_Waitall, MPI_Isend e MPI_Irecv. Isso basicamente substitui o envio e recebimento síncrono (MPI_Send e MPI_Recv) por um assíncrono.

```
MPI_Irecv(&estado_esq, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
&requisicoes[0]);
MPI_Irecv(&estado_dir, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD, &requisicoes[1]);

double inicio = MPI_Wtime();

if (rank % 2 == 0) {
    MPI_Isend(&estado, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
&requisicoes[2]);
    MPI_Isend(&estado, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD, &requisicoes[3]);
} else {
    MPI_Isend(&estado, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD, &requisicoes[2]);
    MPI_Isend(&estado, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
&requisicoes[3]);
}

MPI_Waitall(4, requisicoes, MPI_STATUSES_IGNORE);
```

Também foi necessário alterar o código para registrar o tempo médio de espera de cada filósofo. Foi alterado os seguintes itens:

```
double inicio = MPI_Wtime();

# REGISTRO DE TEMPO FINAL
double fim = MPI_Wtime();
double tempo_rodada = fim - inicio;
acumulador_tempo += tempo_rodada;

# CÁLCULO DO TEMPO MÉDIO DE ESPERA
printf("Filósofo %d: tempo médio de espera = %.3f segundos\n", rank, acumulador_tempo
/ tentativas);
```

2. VERSÕES DO CÓDIGO

- VERSÃO BASE

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define N 5
enum { THINKING, HUNGRY, EATING };

int estado;

int esquerda(int rank) { return (rank + N - 1) % N; }
int direita(int rank) { return (rank + 1) % N; }

int main(int argc, char** argv) {
    int rank, size;
```

```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != N) {
        if (rank == 0) printf("Execute com exatamente %d processos.\n", N);
        MPI_Finalize();
        return 0;
    }

    srand(time(NULL) + rank);
    estado = THINKING;

    for (int step = 0; step < 10; step++) {

        printf("Filósofo %d pensando.\n", rank);
        sleep(rand() % 2);


        estado = HUNGRY;
        printf("Filósofo %d com fome.\n", rank);


        int estado_esq, estado_dir;
        MPI_Send(&estado, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD);
        MPI_Send(&estado, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD);


        MPI_Recv(&estado_esq, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        MPI_Recv(&estado_dir, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

        if (estado_esq != EATING && estado_dir != EATING) {
            estado = EATING;
            printf("Filósofo %d comendo.\n", rank);
            sleep(rand() % 2);
        } else {
            printf("Filósofo %d esperando.\n", rank);
        }

        estado = THINKING;
        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

- VERSÃO CORRIGIDA

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define N 5
enum { THINKING, HUNGRY, EATING };

int estado;

int esquerda(int rank) { return (rank + N - 1) % N; }
int direita(int rank) { return (rank + 1) % N; }

int main(int argc, char** argv) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != N) {
        if (rank == 0) printf("Execute com exatamente %d processos.\n", N);
        MPI_Finalize();
        return 0;
    }

    srand(time(NULL) + rank);
    estado = THINKING;

    // TRECHO PARA TEMPO DE ESPERA

    double acumulador_tempo = 0.0;

    int tentativas = 10;

    for (int step = 0; step < tentativas; step++) {

        printf("Filósofo %d pensando.\n", rank);
        sleep(rand() % 2);

        estado = HUNGRY;
        printf("Filósofo %d com fome.\n", rank);

        int estado_esq = THINKING, estado_dir = THINKING;
        MPI_Request requisicoes[4];

        MPI_Irecv(&estado_esq, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
&requisicoes[0]);
        MPI_Irecv(&estado_dir, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD,
&requisicoes[1]);

        double inicio = MPI_Wtime();
```

```
        if (rank % 2 == 0) {
            MPI_Isend(&estado, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
&requisicoes[2]);
            MPI_Isend(&estado, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD,
&requisicoes[3]);
        } else {
            MPI_Isend(&estado, 1, MPI_INT, direita(rank), 0, MPI_COMM_WORLD,
&requisicoes[2]);
            MPI_Isend(&estado, 1, MPI_INT, esquerda(rank), 0, MPI_COMM_WORLD,
&requisicoes[3]);
        }

        MPI_Waitall(4, requisicoes, MPI_STATUSES_IGNORE);

        if (estado_esq != EATING && estado_dir != EATING) {
            estado = EATING;

            double fim = MPI_Wtime();
            double tempo_rodada = fim - inicio;
            acumulador_tempo += tempo_rodada;

            printf("Filósofo %d está comendo. Esperou %.3f segundos nesta
tentativa.\n", rank, tempo_rodada);
            sleep(rand() % 2);
        } else {
            printf("Filósofo %d aguardando.\n", rank);
        }

        estado = THINKING;
        MPI_Barrier(MPI_COMM_WORLD);
    }

    printf("Filósofo %d: tempo médio de espera = %.3f segundos\n", rank,
acumulador_tempo / tentativas);

    MPI_Finalize();
    return 0;
}
```

3. RELATÓRIO

- Tempo médio de espera de cada filósofo Após a execução da versão corrigida do código, foi registrado o tempo médio de espera de cada filósofo. A varição ocorre devido ao comportamento randômico e à concorrência entre os próprios processos.
- Casos de bloqueio e deadlock observados Com o código original (com uso de MPI_Send e MPI_Recv), ocorria deadlock em várias execuções, pois todos os filósofos tentavam enviar mensagens simultaneamente sem processos prontos para receber. Na versão corrigida (MPI_Isend + MPI_Irecv + MPI_Waitall) não foram

observados deadlocks, garantindo que todos os filósofos conseguissem comer em todas as tentativas.