Documentação Trabalho Prático 1

Equipe:

Nome: Michael Willian Pereira Vieira
Nome: Luiz Gabriel Antunes Sena (Luiz Sena)

1 Arquitetura:

Como ponto de partida para o trabalho, decidimos antes de tudo focar como seria a arquitetura dos diretórios do nosso trabalho. Com isso, optamos por seguir o conceito de tipos abstratos de dados, composto por headers e sources. A partir dessa escolha, focamos em que tipo de composições, desse trabalho, seria necessário uma espécie de pacotes.. Em seguida, visto que é um trabalho em grupo, optamos por um diretório específico para testes dos métodos construídos, dessa forma economizamos tempo, ao colidir com erros e na análise de códigos para descobrir a utilidade de certos métodos.

Estrutura:

```
├── bin/
│   ├── consumidor_produtor.o
│   ├── leitor_escritor.o
│   ├── consumidorXescritor.o
│   └── test_<biblioteca>.o
├── include/
│   └── <biblioteca>.h
├── lib/
│   └── <biblioteca>.c
├── src/
│   ├── consumidor_produtor.o
│   ├── leitor_escritor.o
│   └── main.o
├── tests/
│   └── <biblioteca>.test.c
├── Makefile
└── README.md
```

2. Instruções para Compilação e Execução

Para replicar o ambiente de teste e executar os problemas de concorrência implementados, utilize os comandos make a partir do diretório raiz do projeto.

2.1 Compilação do Projeto

O projeto utiliza um Makefile para automatizar a compilação de todos os arquivos de código-fonte (.c) e a vinculação da biblioteca pthread.

Comando:

Bash

```
make build
```

Resultado:

O executável principal, main, será gerado no diretório bin/.

2.2 Execução dos Testes de Concorrência

Foram criados comandos de atalho no Makefile para executar diretamente cada conjunto de testes (Leitor/Escritor ou Produtor/Consumidor) com configurações padrão, além da execução manual via bin/main.

A. Teste 1: Problema Leitor X Escritor

Este comando executa as três versões do problema Leitor/Escritor (Sem Prioridade, Com Prioridade, Sem Controle).

Comando:

Bash

```
make leitorEscritor
```

Configuração Padrão:

| Parâmetro | Valor |

| :--- | :--- |

| Escritores (-w) | 5 |

| Leitores (-r) | 10 |

B. Teste 2: Problema Produtor X Consumidor

Este comando executa as três versões do problema Produtor/Consumidor (V1: N:1, V2: N:M Seguro, V3: N:M Inseguro).

Comando:

make consumidorProdutor

Configuração Padrão:

| Versão | Produtores | Consumidores |

| :--- | :--- | :--- |

| V1 e V3 (V1/V3) | 5 | 1 |

| V2 (N:M Seguro) | 10 | 5 |

2.3 Execução Manual com Parâmetros Customizados

Flag	Forma Longa	Descrição	Status
-w	--writer	Contagem de Escritores (L/E) ou Produtores (P/C V1/V3). **Obrigatório**.	
-r	--reader	Contagem de Leitores (L/E) ou Consumidores (P/C V1/V3). **Obrigatório**.	
-s	--switch	**Habilita** o modo Produtor X Consumidor.	Opcional
-p	--prod-v2	Contagem de Produtores **apenas para P/C V2**.	Opcional
-c	--cons-v2	Contagem de Consumidores **apenas para P/C V2**.	Opcional

Para configurar o número de *threads* para cada cenário, utilize o executável diretamente, passando os argumentos obrigatórios (-w e -r) e as *flags* opcionais.**Exemplo de Comando (P/C Customizado)**

Para rodar os testes de Produtor/Consumidor, definindo:

- V1/V3 com 5 Produtores e 1 Consumidor.
- V2 com 15 Produtores e 10 Consumidores.

Bash

```
./bin/main -w 5 -r 1 -p 15 -c 10 -s
```

Exemplo de Comando (L/E Customizado)
Para rodar o teste de Leitor/Escritor com 20 Leitores e 2 Escritores:

Bash

```
../bin/main -w 2 -r 20
```

3 Linguagem e Metodologia de Implementação:

O trabalho foi implementado em Linguagem C, utilizando a biblioteca Pthreads (POSIX Threads) para gerenciar *threads*. Os mecanismos de sincronização utilizados incluem:

- Mutexes (*Mutual Exclusion*): Para garantir o acesso exclusivo a regiões críticas.
- Semáforos: Para controlar a ordem e a capacidade do *buffer* (Produtor/Consumidor).
- Variáveis de Condição: Para permitir que *threads* esperem por condições específicas dentro de uma seção crítica (uso comum em algumas versões de Leitor/Escritor).

4. Problema 1: Leitor e Escritor

4.1 Cenário Prático: Gestão de Contas Bancárias

O problema Leitor/Escritor foi adaptado para simular um cenário de **Gestão de Contas Bancárias**.

- **Escritores:** Representam operações de **Débito/Crédito** (modificam o saldo).
- Leitores: Representam operações de Consulta (apenas leem o saldo).
  A região crítica é o saldo da conta bancária compartilhada.

4.2 Implementação das Versões e Análise

O Mutex foi a principal ferramenta de controle, utilizada para garantir a atomicidade nas operações de leitura e escrita do saldo.

Versão 1: Sem Prioridade (Acesso Justo)

Conceito	Explicação
Sincronização	Uso simples de **Mutex** para proteger o acesso à conta. O Mutex é adquirido por qualquer *thread* (leitor ou escritor) antes de acessar o saldo.
Comportamento	Oferece um acesso justo, mas **não é ideal em performance**, pois leitores (que não modificam o dado) bloqueiam outros leitores.

Versão 2: Escritores com Prioridade sobre Leitores

Conceito	Explicação
Sincronização	Implementação de lógica que usa **Variáveis de Condição e Contadores** (e Mutex) para priorizar Escritores. Se houver Escritores esperando, novos Leitores são bloqueados até que todos os Escritores pendentes terminem.
Comportamento	Garante que as atualizações críticas (Escritores) sejam feitas rapidamente, prevenindo a **Fome (*Starvation*)** do Escritor. Contudo, em cenários de alta taxa de escrita, pode levar à Fome do Leitor.

Versão 3: Sem Controle de Concorrência

Conceito	Explicação
Sincronização	Nenhuma. As *threads* acessam o saldo diretamente.
Consequência	Geração de **Condições de Corrida** durante a operação de débito/crédito (operação de LME).

5. Problema 2: Produtor e Consumidor

5.1 Cenário Prático: Sistema de Processamento de Débitos

O problema Produtor/Consumidor simula um sistema de processamento de transações bancárias (Débitos).

- ***Produtores:*** *Geram novos débitos e os inserem no* ***Buffer Compartilhado****.*
- *Consumidores: Retiram os débitos do buffer e os Executam (atualizando o saldo da conta destino/origem).*
  A região crítica é o Buffer Compartilhado.

5.2 Implementação das Versões e Análise

O mecanismo central de sincronização é o ***Semáforo****, complementado por um* ***Mutex*** *para proteger o acesso ao buffer em si.*

Versão 1: Vários Produtores e 1 Consumidor

Conceito	***Explicação***
Sincronização	*Usa **Semáforo empty** (vagas) e **Semáforo full** (itens) para Produtores e Consumidor, respectivamente. Um **Mutex** é usado para garantir que apenas uma thread por vez acesse a estrutura do buffer.*
Comportamento	*É a implementação fundamental. O único Consumidor é o **gargalo (bottleneck)** do sistema. O sistema é correto e os dados são íntegros.*

```
====================================
         MONITOR DE THREADS
====================================
[PRODUTOR 1] -> ESTADO: FINALIZADO
[PRODUTOR 2] -> ESTADO: FINALIZADO
[PRODUTOR 3] -> ESTADO: FINALIZADO
[PRODUTOR 4] -> ESTADO: FINALIZADO
[PRODUTOR 5] -> ESTADO: FINALIZADO
[CONSUMIDOR 6] -> ESTADO: FINALIZADO

====================================
           STATUS DO BUFFER
====================================
Itens no Buffer: 0/10 (Entrada: 5 | Saída: 5)
cor out      cor in
Conteúdo do Buffer (ID Transação):
[0] 1024 [1] 1019 [2] 1009 [3] 1014 [4] 1004
[5] 1023 [6] 1018 [7] 1008 [8] 1013 [9] 1003

====================================
[MANAGER] Enviando sinal de cancelamento para 6 threads...

[RESULTADO] Buffer Final: 0 itens (Idealmente: 0)
[RESULTADO] Total de débitos gerados: 25
[RESULTADO] Total do valor dos débitos processados: 1285.00
[RESULTADO] Saldo final da Conta Origem: 98715.00 (Idealmente: 98715.00)
[RESULTADO] Saldo final da Conta Destino: 51285.00 (Idealmente: 51285.00)
[Mon Oct 20 03:06:11 2025] (PROGRAMA PRINCIPAL): Testando Versão 2 (SEGURA – Varios Consumidores e Produtores)
```

Note que ao rodar a versão 1, contador do buffer está setado com 0, com deveria estar e além disso os débitos foram executados de forma adequada devido ao controle que foi utilizado.

Versão 2: Vários Produtores e Vários Consumidores

Conceito	***Explicação***
Sincronização	*Mantém os Semáforos e o **Mutex** de exclusão mútua sobre o acesso ao buffer. A diferença está no aumento da concorrência no lado da Consumo.*
Comportamento	*Aumenta a vazão (throughput) de débitos processados, pois múltiplos Consumidores podem processar simultaneamente. **O Mutex é essencial** para evitar que dois Consumidores tentem consumir o mesmo item (dupla retirada) ou corrompam a variável out.*

```
====================================
           STATUS DO BUFFER
====================================
Itens no Buffer: 5/10 (Entrada: 6 | Saída: 1)
cor out      cor in
Conteúdo do Buffer (ID Transação):
[0] 1000 [1] 1005 [2] 1015 [3] 1025 [4] 1035
[5] 1045 [6] 0 [7] 0 [8] 0 [9] 0

====================================
====================================
         MONITOR DE THREADS
====================================
[PRODUTOR 1] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 2] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 3] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 4] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 5] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 6] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 7] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 8] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 9] -> ESTADO: DORMINDO (Semáforo)
[PRODUTOR 10] -> ESTADO: DORMINDO (Semáforo)
[CONSUMIDOR 11] -> ESTADO: INICIADO
[CONSUMIDOR 12] -> ESTADO: CONSUMINDO (PROCESSANDO: ID 1000)
[CONSUMIDOR 13] -> ESTADO: DORMINDO (Semáforo)
[CONSUMIDOR 14] -> ESTADO: DORMINDO (Semáforo)
[CONSUMIDOR 15] -> ESTADO: DORMINDO (Semáforo)

====================================
           STATUS DO BUFFER
====================================
Itens no Buffer: 5/10 (Entrada: 6 | Saída: 1)
cor out      cor in
Conteúdo do Buffer (ID Transação):
[0] 1000 [1] 1005 [2] 1015 [3] 1025 [4] 1035
[5] 1045 [6] 0 [7] 0 [8] 0 [9] 0

====================================
```

Note a existência de múltiplos consumidores ao mesmo tempo.

```
====================================
====================================
         MONITOR DE THREADS
====================================
[PRODUTOR 1] -> ESTADO: FINALIZADO
[PRODUTOR 2] -> ESTADO: FINALIZADO
[PRODUTOR 3] -> ESTADO: FINALIZADO
[PRODUTOR 4] -> ESTADO: FINALIZADO
[PRODUTOR 5] -> ESTADO: FINALIZADO
[PRODUTOR 6] -> ESTADO: FINALIZADO
[PRODUTOR 7] -> ESTADO: FINALIZADO
[PRODUTOR 8] -> ESTADO: FINALIZADO
[PRODUTOR 9] -> ESTADO: FINALIZADO
[PRODUTOR 10] -> ESTADO: FINALIZADO
[CONSUMIDOR 11] -> ESTADO: FINALIZADO
[CONSUMIDOR 12] -> ESTADO: FINALIZADO
[CONSUMIDOR 13] -> ESTADO: FINALIZADO
[CONSUMIDOR 14] -> ESTADO: FINALIZADO
[CONSUMIDOR 15] -> ESTADO: FINALIZADO

====================================
           STATUS DO BUFFER
====================================
Itens no Buffer: 0/10 (Entrada: 0 | Saída: 0)
cor out     cor in
Conteúdo do Buffer (ID Transação):
[0] 1013 [1] 1023 [2] 1018 [3] 1042 [4] 1034
[5] 1024 [6] 1014 [7] 1019 [8] 1043 [9] 1044

====================================
[MANAGER] Enviando sinal de cancelamento para 15 threads...

[RESULTADO] Buffer Final: 0 itens (Idealmente: 0)
[RESULTADO] Total de débitos gerados: 50
[RESULTADO] Total do valor dos débitos processados: 3853.00
[RESULTADO] Saldo final da Conta Origem: 96147.00 (Idealmente: 96147.00)
[RESULTADO] Saldo final da Conta Destino: 53853.00 (Idealmente: 53853.00)
[Mon Oct 20 03:06:16 2025] (PROGRAMA PRINCIPAL): Testando Versão 3 (INSEGURA — sem sincronização)
[Mon Oct 20 03:06:16 2025] (PROGRAMA PRINCIPAL): ATENÇÃO: Versão insegura — esperado condições de corrida!
```

Note que mesmo com a presença de múltiplos produtores e consumidores, o resultado final foi como o esperado, todos os débitos produzidos e executados de forma adequada a conta dos usuários.

Versão 3: Sem Controle de Concorrência

Conceito	***Explicação***
Sincronização	*Nenhuma sincronização de Mutex/Semáforo. O Consumidor usa* ***Espera Ocupada (Busy Waiting)*** *(laço while (contador <= 0)).*
Consequência	***1. Ineficiência:*** *Desperdício de CPU com Busy Waiting (visível no estado ESPERA_OCUPADA).* ***2. Corrupção de Dados:*** *Race Condition em buffer->contador e na operação de débito.*

```
=====================================
=====================================
         MONITOR DE THREADS
=====================================
[PRODUTOR 1] -> ESTADO: FINALIZADO
[PRODUTOR 2] -> ESTADO: FINALIZADO
[PRODUTOR 3] -> ESTADO: FINALIZADO
[PRODUTOR 4] -> ESTADO: FINALIZADO
[PRODUTOR 5] -> ESTADO: FINALIZADO
[CONSUMIDOR 6] -> ESTADO: FINALIZADO

=====================================
           STATUS DO BUFFER
=====================================
Itens no Buffer: -40/10 (Entrada: 5 | Saída: 5)
cor out      cor in
Conteúdo do Buffer (ID Transação):
[0] 1007 [1] 1018 [2] 1008 [3] 1019 [4] 1009
[5] 1014 [6] 1022 [7] 1023 [8] 1024 [9] 1017

=====================================
[MANAGER] Enviando sinal de cancelamento para 6 threads...

[RESULTADO] Buffer Final: -40 itens (Idealmente: 0)
[RESULTADO] Total de débitos gerados: 25
[RESULTADO] Total do valor dos débitos processados: 5154.00
[RESULTADO] Saldo final da Conta Origem: 92766.00 (Idealmente: 94846.00)
[RESULTADO] Saldo final da Conta Destino: 57234.00 (Idealmente: 55154.00)
[ALERTA] Buffer não esvaziado ou erro de contagem.
[Mon Oct 20 03:06:21 2025] (PROGRAMA PRINCIPAL): Testes de Produtor/Consumidor concluídos
[Mon Oct 20 03:06:21 2025] (PROGRAMA PRINCIPAL): Programa finalizado com sucesso
```

Note que nesta versão, pelo falta do controle de concorrência das threads, houve race condition pelo contador do buffer, pois o resultado final foi -40, além disso a presença da inconsistência do saldo das contas dos usuários também é algo de se enxergar, pois devido a race condition, entre os consumidores por débitos produzidos, fez com que fossem executadas de forma não adequada.

6. Conclusão

Este trabalho demonstrou que a implementação de mecanismos de sincronização não é apenas uma boa prática, mas um requisito fundamental para a integridade dos dados em sistemas concorrentes. A análise das versões sem controle (Versão 3) provou que, em cenários práticos (contas bancárias e processamento de débitos), a ausência de Mutexes e Semáforos leva inevitavelmente à falha lógica e financeira do sistema. O uso correto do `Mutex` *(para exclusão mútua) e de* `Semáforos` *(para controle de capacidade e ordem) garantiu a correção e a eficiência das versões seguras.*