

**Controle de concorrência:**

Data: 31 Mai 2024

**Discentes:**

Lucas Rabelo de Araujo Moraes

Raimundo Martins de Araújo Júnior

## 1 Introdução

Para a concepção desse projeto, desenvolvemos uma interface a fim de exemplificar de forma prática os conceitos de controle de concorrência utilizando os protocolos: ***Wait-die*** e ***Wound-wait***. Na parte de codificação, foi utilizada a linguagem de programação Python em sua versão 3 juntamente com a biblioteca ***TkInter***. Logo abaixo, listamos algumas das ferramentas/tecnologias utilizadas nesse projeto:

- ***Github*** - Ferramenta para versionamento de código e armazenamento do projeto;
- ***VS Code*** - IDE para escrita dos algoritmos;
- ***TkInter*** - Interface padrão da linguagem de programação Python baseada em Tcl/Tk. Pode ser instalada através da linha de comando abaixo:

- `pip install tkinter`;

## 2 Contextualização e Interface

### 2.1 Wait-Die e Wound-Wait

Os ***Deadlocks*** podem ser um grande problema em relação a sistemas gerenciadores de banco de dados (SGBDs) que realizam o gerenciamento de transações de forma concorrente/distribuído. Sendo assim, se faz necessário um planejamento a fim de realizar a prevenção, detecção de anomalias evitando assim os *deadlocks*. Resolvendo essa problemática, teremos um SGBD com desempenho satisfatório e deixando sua integridade em conformidade.

Em SGBDs, um *deadlock* ocorre quando duas ou mais transações ficam em uma espera contínua aguardando acesso a um ou mais recursos compartilhados, que só podem ser acessados pelas transações em espera quando forem liberados pelas transações que estão fazendo uso do recurso.

Nesse projeto, iremos trabalhar com dois protocolos que são baseados nas *timestamps* das transações: *Wait-die* e *Wound-wait*. Esses protocolos tomam a decisão se uma transação deve aguardar ou ser abortada, para selecionar um protocolo, é necessário selecioná-lo no radio button da interface, não é possível mudar de protocolo após a adição de uma transação.

- ***Wait-die*** - nesse protocolo, se uma transação (X) tiver o interesse de acessar um recurso que está bloqueado por outra transação (Y), o SGBD verifica o *timestamp* das duas transações deixando que a transação mais antiga, caso seja a solicitante,

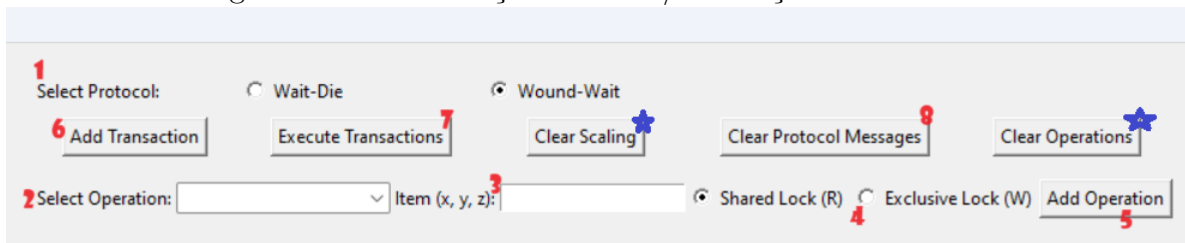
se mantenha na fila de espera aguardando até que o recurso esteja desbloqueado e disponível para execução da transação, se a transação solicitante for a mais nova, ela é abortada e recomeça mais tarde com mesmo timestamp;

- **Wound-wait** - já nesse protocolo, se uma transação (X) mais antiga tiver o interesse em acessar determinado recurso que está sendo usado por uma transação mais nova (Y), a transação mais antiga obriga a transação mais nova a encerrar suas operações a fim de liberar o recurso instantaneamente, caso contrário, se a mais nova for a solicitante ela aguarda a transação mais antiga desbloquear o recurso para utilizá-lo.

## 2.2 Interface desenvolvida

Nessa subseção, apresentamos as funcionalidades da interface do sistema que foi desenvolvido. Na Figura 1, são apresentadas todas as funcionalidades do sistema desenvolvido, as estrelas representam uma seleção opcional. Após a Figura 1, temos a descrição de cada funcionalidade:

Figure 1: Recomendações de Uso / Instruções da Interface

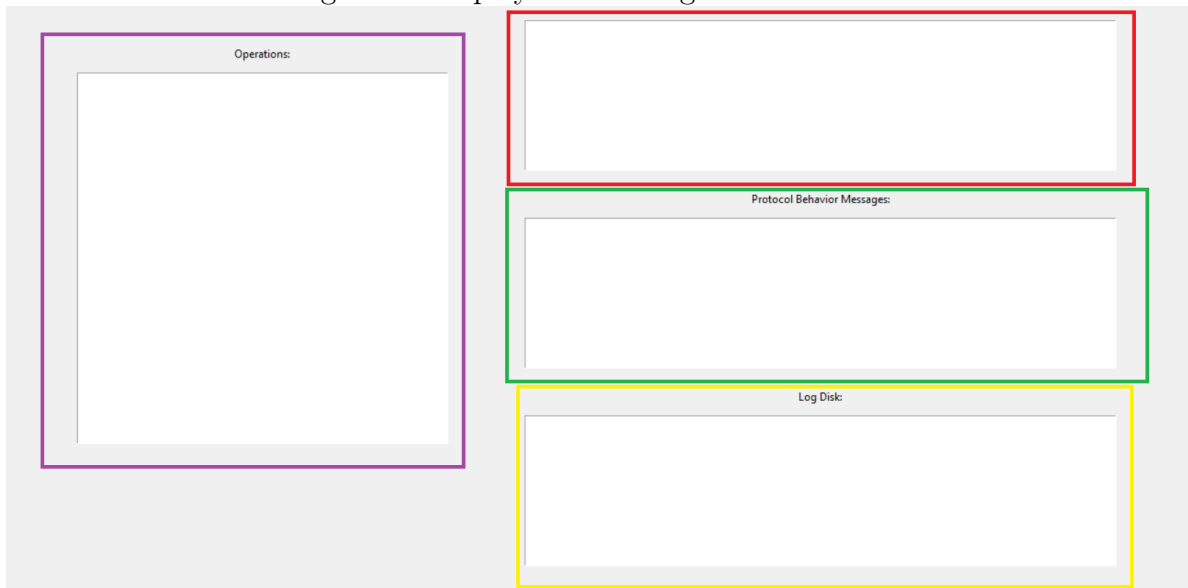


Fonte: Autoria própria.

1. – Campo do tipo *RadioButton* que tem como objetivo selecionar o tipo de protocolo que será utilizado na simulação;
2. – Campo do tipo *OptionMenu* que tem como objetivo selecionar o tipo de operação que será realizada: Leitura (R - *Read*) ou Escrita (W - *Write*);
3. – Campo do tipo *Label* que tem como objetivo inserir os itens de cada operação: x, y ou z;
4. – Campo do tipo *RadioButton* que tem como objetivo selecionar o tipo de compartilhamento que será utilizado nas operações: bloqueio compartilhado ou exclusivo;
5. – Campo do tipo *Button* que tem como objetivo adicionar cada operação de cada transação;
6. – Campo do tipo *Button* que tem como objetivo adicionar cada transação para realizar as simulações;
7. – Campo do tipo *Button* que tem como objetivo executar as transações que foram adicionadas para simulação;
8. – Campo do tipo *Button* que tem como objetivo limpar a tela de mensagens do protocolo.

Na Figura 2 temos o *display* do sistema desenvolvido. Logo abaixo da Figura 2, apresentamos a função de cada um dos itens:

Figure 2: Displays de mensagem da Interface



Fonte: Autoria própria.

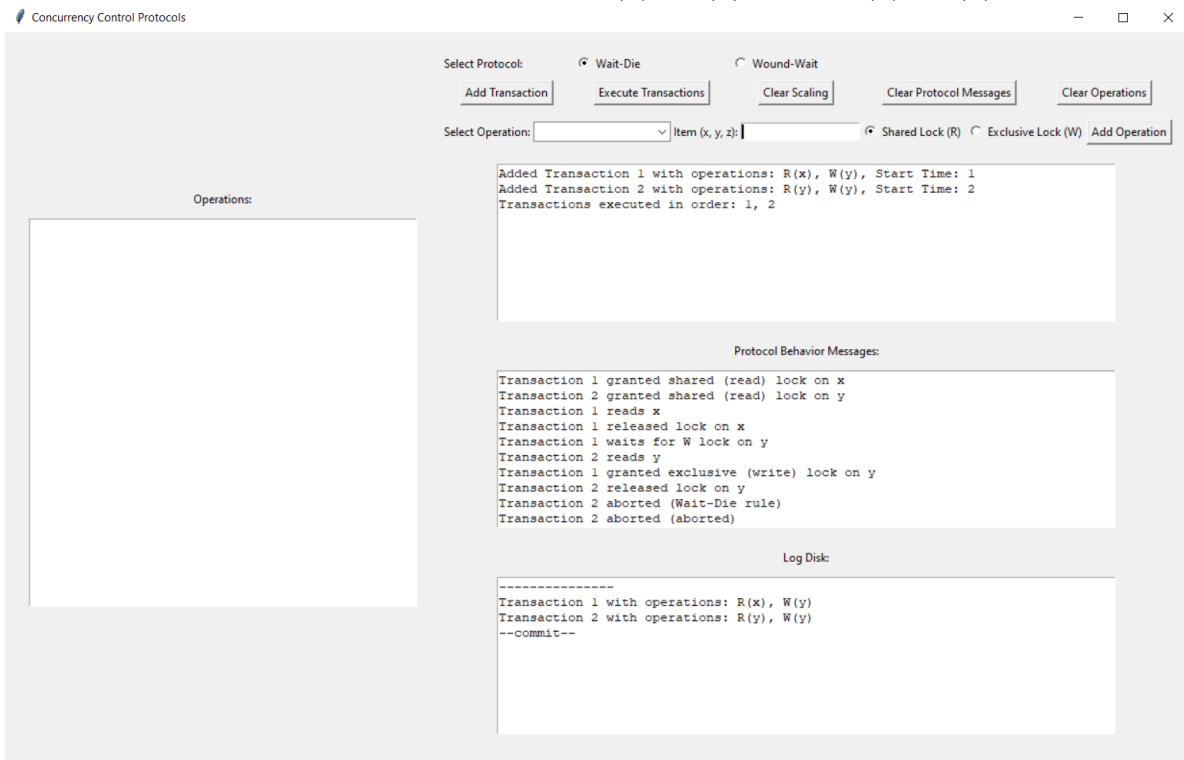
- **Lilás** - é a tela que lista todas as operações inseridas para cada transação;
- **Vermelho** - é a tela de escalonamento (Log da memória);
- **Verde** - é a tela que apresenta o comportamento do protocolo utilizado;
- **Amarelo** - é a tela que representa o Log do disco.

## 2.3 Execução das simulações

### 2.3.1 Wait-die

Nesse primeiro exemplo, utilizando o protocolo *Wait-die*, são criadas duas transações. Na primeira transação, temos as operações de Leitura (R - *Read*) do item X e Escrita (W - *Write*) do item Y. Na segunda transação, temos as operações de Leitura e escrita no item Y.

Figure 3: Simulação de T1: R(x), W(y) e T2: R(y), W(y)



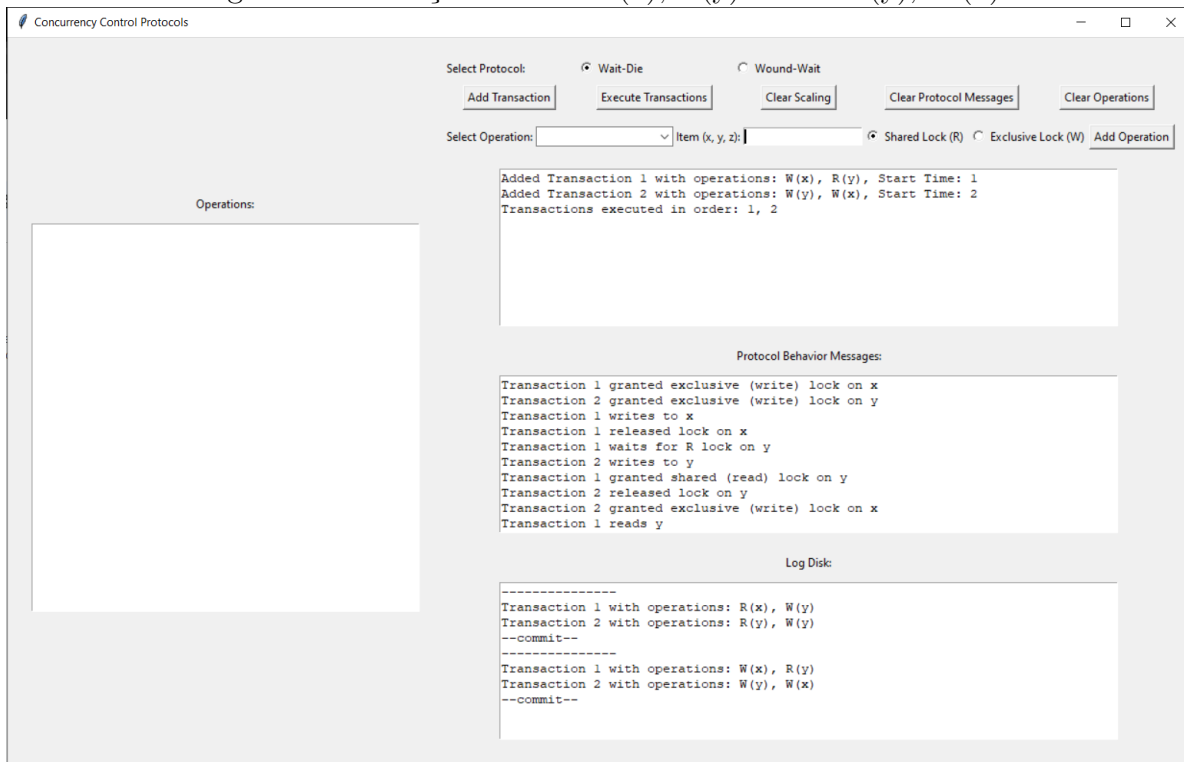
Fonte: Autoria própria.

Após a execução das duas transações, temos as seguintes informações:

- A **Transação 1** realiza o bloqueio compartilhado do item X;
- A **Transação 2** realiza o bloqueio compartilhado do item Y;
- A **Transação 1** realiza a leitura do item X e depois libera o acesso ao item X;
- A **Transação 1** precisa aguardar na fila para acessar o item Y, visto que ele está bloqueado por estar sendo acessado pela **transação 2**;
- A **Transação 2** realiza a leitura do item Y e depois libera o acesso ao item Y;
- A **Transação 1** realiza o bloqueio exclusivo do item Y;
- A **Transação 2** é abortada visto que, pela regra do protocolo *Wait-die*, como o recurso está sendo utilizado pela **Transação 1**, a **Transação 2** deve ser abortada e reiniciar o processo;
- A **Transação 1** realiza a escrita no item Y e depois realiza o desbloqueio do item;
- Por fim, com o item Y desbloqueado, a **Transação 2** executa a operação de leitura no item Y.

No segundo exemplo, utilizando o protocolo *Wait-die*, novamente são criadas duas transações. Entretanto, na primeira temos as operações de Escrita (W - *Write*) no item X e Leitura (R - *Read*) do item Y. Na segunda temos as operações de escrita nos itens Y e X, respectivamente.

Figure 4: Simulação de T1: W(x), R(y) e T2: W(y), W(x)



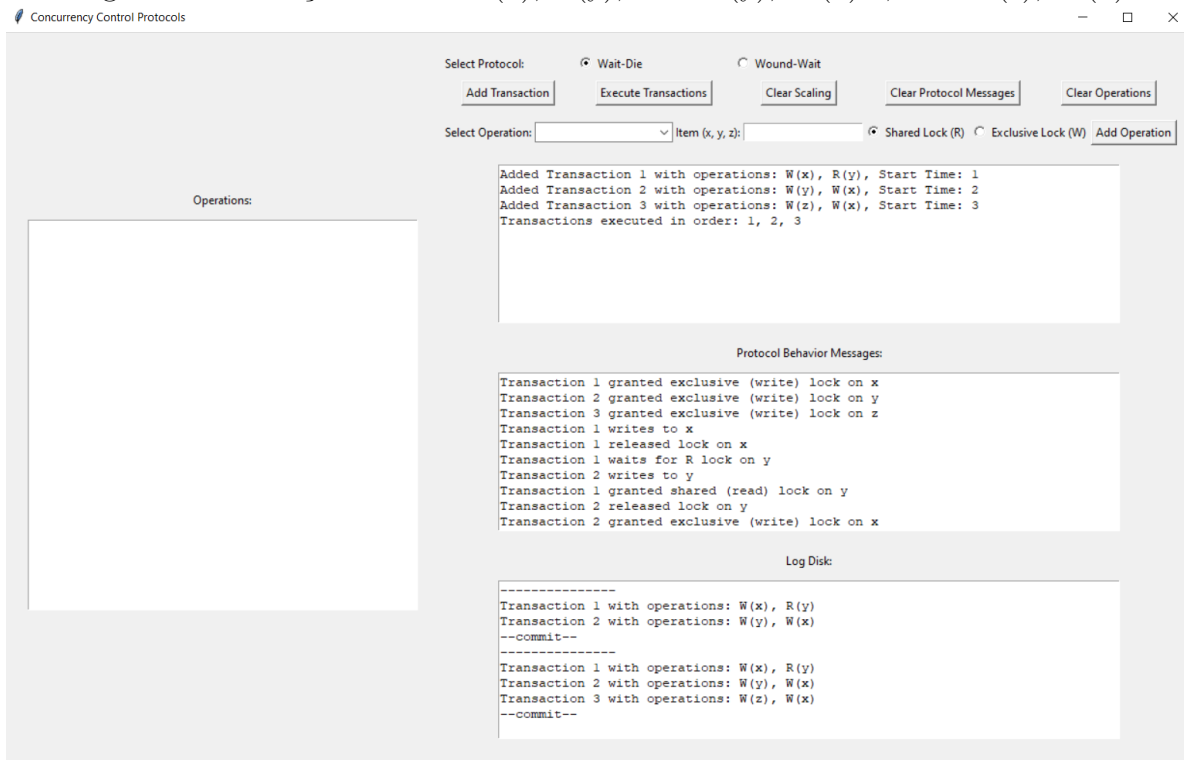
Fonte: Autoria própria.

Após a execução das duas transações, temos as seguintes informações:

- A **Transação 1** realiza o bloqueio do item X de forma exclusiva;
- A **Transação 2** realiza o bloqueio do item Y de forma exclusiva;
- A **Transação 1** realiza a escrita no item X e depois libera o acesso ao item X;
- A **Transação 1** precisa aguardar na fila para acessar o item Y, visto que ele está bloqueado por estar sendo acessado pela **transação 2**;
- A **Transação 2** realiza a escrita no item Y e depois libera o acesso ao item Y;
- A **Transação 1** realiza o bloqueio do item Y de forma compartilhada;
- A **Transação 2** realiza o bloqueio do item X de forma exclusiva;
- A **Transação 1** realiza a leitura do item Y e depois desbloqueia o item Y;
- A **Transação 2** realiza a escrita no item X e depois desbloqueia o item X.

Nesse terceiro exemplo, utilizando o protocolo *Wait-die*, novamente são criadas duas transações. Entretanto, na primeira temos as operações de Escrita (W - *Write*) no item X e Leitura (R - *Read*) do item Y. Na segunda transação, temos as operações de escrita nos itens Y e X, respectivamente. Por fim, na terceira transação temos a operação de escrita nos itens Z e X.

Figure 5: Simulação de T1: W(x), R(y); T2: W(y), W(x) e; T3: W(z), W(x)



Fonte: Autoria própria.

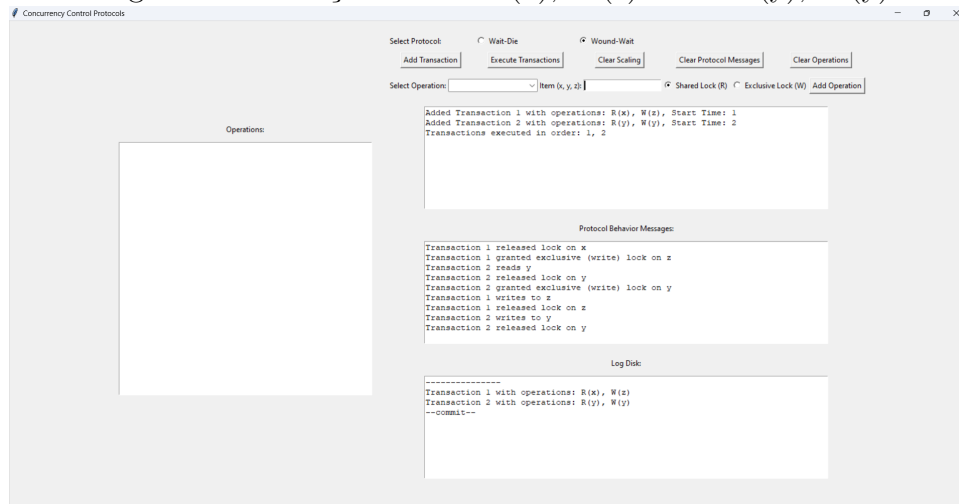
Após a execução das duas transações, temos as seguintes informações:

- A **Transação 1** realiza o bloqueio do item X de forma exclusiva;
- A **Transação 2** realiza o bloqueio do item Y de forma exclusiva;
- A **Transação 3** realiza o bloqueio do item Z de forma exclusiva;
- A **Transação 1** realiza a escrita no item X e desbloqueia o item x;
- A **Transação 1** aguarda o item y, pois ele está sendo utilizado pela transação 2;
- A **Transação 2** realiza a escrita no item Y de desbloqueia o item Y;
- A **Transação 1** realiza o bloqueio do item Y de forma compartilhada;
- A **Transação 2** realiza o bloqueio do item X de forma exclusiva;
- A **Transação 3** realiza a escrita no item Z e desbloqueia o item Z;
- A **Transação 3** é abortada, visto que o item X está sendo utilizado pela transação 2;
- A **Transação 1** realiza a leitura do item Y e desbloqueia o item Y;
- A **Transação 2** realiza a escrita no item X e desbloqueia o item X;
- A **Transação 3** realiza a escrita no item X.

### 2.3.2 Wound-wait

Nesse primeiro exemplo, utilizando o protocolo *Wound-Wait*, selecionado no Radio-Button da interface, são criadas duas transações. Na primeira transação, temos as operações de Leitura (R - *Read*) do item X e Escrita (W - *Write*) do item Z. Na segunda transação, temos as operações de Leitura e escrita no item Y.

Figure 6: Simulação de T1: R(x), W(z) e T2: R(y), W(y)



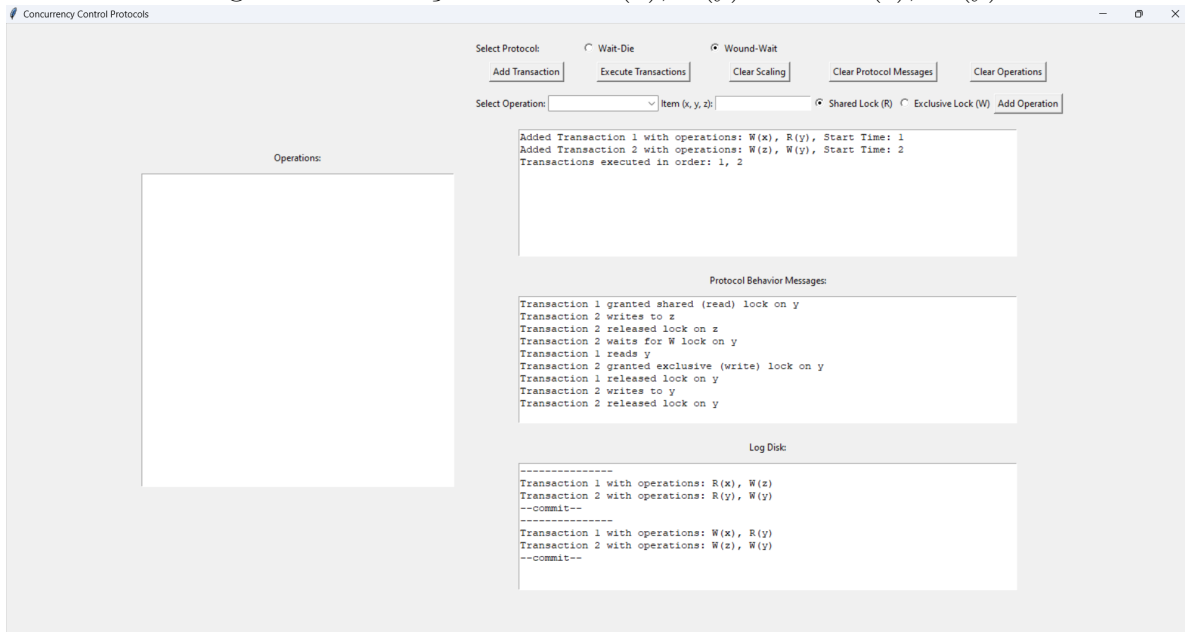
Fonte: Autoria própria.

Após a execução das duas transações, temos as seguintes informações:

- A **Transação 1** obtém um bloqueio compartilhado (leitura) sobre o item x;
- A **Transação 2** obtém um bloqueio compartilhado (leitura) sobre o item y;
- A **Transação 1** realiza a leitura do item x e libera o bloqueio;
- A **Transação 1** obtém um bloqueio exclusivo (escrita) sobre o item z;
- A **Transação 2** realiza a leitura do item y e libera o bloqueio sobre o item;
- A **Transação 2** obtém um bloqueio exclusivo (escrita) sobre o item y;
- A **Transação 1** escreve no item z e posteriormente libera o bloqueio;
- A **Transação 2** realiza a escrita no item y e depois libera o bloqueio;

No segundo exemplo, na primeira transação, temos as operações de Escrita (W - *Write*) do item X e Leitura (R - *Read*) do item y. Na segunda transação, temos as operações de escrita no item z e no item y.

Figure 7: Simulação de T1: W(x), R(y) e T2: W(z), W(y)



Fonte: Autoria própria.

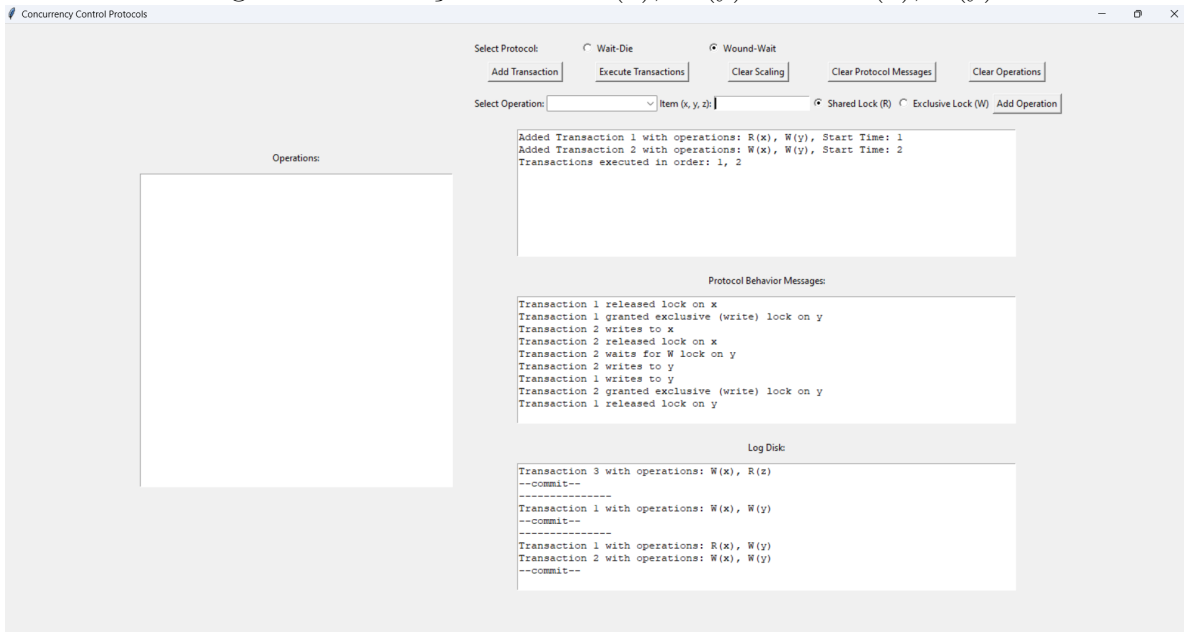
Após a execução das duas transações, temos as seguintes informações:

- A **Transação 1** obtém um bloqueio exclusivo (escrita) sobre o item x;
- A **Transação 2** obtém um bloqueio exclusivo (escrita) sobre o item z;
- A **Transação 1** escreve no item x e libera o bloqueio;
- A **Transação 1** obtém um bloqueio compartilhado (leitura) sobre o item y;
- A **Transação 2** escreve no item z e libera o bloqueio;
- A **Transação 2** aguarda para obter um bloqueio exclusivo (escrita) sobre o item y, que está sendo utilizado pela **Transação 1**;
- A **Transação 1** lê o item y e libera o bloqueio;
- A **Transação 2** obtém um bloqueio exclusivo (escrita) sobre o item y;
- A **Transação 2** escreve no item y e libera o bloqueio;

No terceiro exemplo, na primeira transação, temos as operações de Leitura (R - *Read*) do item X e Escrita (W - *Write*) do item y. Na segunda transação, temos as operações de Escrita no item x e Leitura no item y.



Figure 8: Simulação de T1: R(x), W(y) e T2: W(x), R(y)



Fonte: Autoria própria.

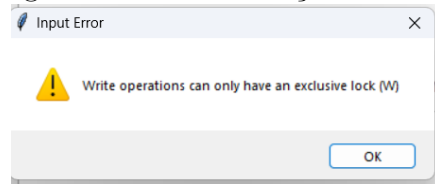
Após a execução das duas transações, temos as seguintes informações:

- A **Transação 1** obtém um bloqueio compartilhado (leitura) sobre o item x;
- A **Transação 2** aguarda para obter um bloqueio exclusivo (escrita) sobre o item x;
- A **Transação 1** lê o item x e libera o bloqueio;
- A **Transação 2** obtém um bloqueio exclusivo (escrita) sobre o item x;
- A **Transação 1** obtém um bloqueio exclusivo (escrita) sobre o item y;
- A **Transação 2** escreve no item x e libera o bloqueio;
- A **Transação 2** aguarda a mais antiga para obter um bloqueio exclusivo (escrita) sobre o item y;
- A **Transação 1** escreve no item y e libera o bloqueio;
- A **Transação 2** obtém um bloqueio exclusivo (escrita) sobre o item y;
- A **Transação 2** escreve no item y e libera o bloqueio;

### 3 Resultados e Discussão

Algumas mensagens de erro foram implementadas na interface para impedir que anomalias aconteçam. Na Figura 9, a mensagem de erro mostra que é impossível atribuir um lock compartilhado a uma operação de leitura, também está disponível uma mensagem de erro que proíbe a adição de um lock exclusivo para a operação de leitura.

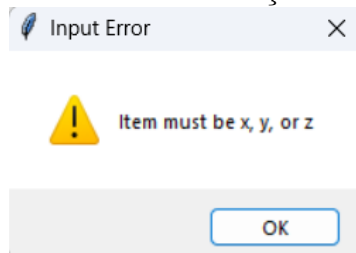
Figure 9: Erro na seleção de Locks



Fonte: Autoria própria.

Na Figura 10, é indicado que apenas os itens x, y e z podem ser selecionados, ou seja, para vias de simulação, não é possível fazer uma operação em um item diferente destes.

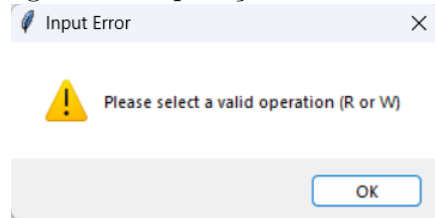
Figure 10: Erro na seleção de itens



Fonte: Autoria própria.

Na Figura 11, é indicado que apenas as operações R - Read ou W - Write podem ser selecionadas, portanto caso o usuário tente selecionar outra letra, não será possível.

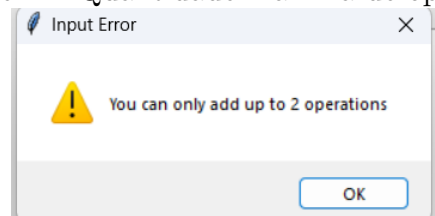
Figure 11: Operação nao existente



Fonte: Autoria própria.

Na Figura 12, apresentamos a configuração do número máximo de operações por transação. Esta configuração foi implementada para assegurar o funcionamento contínuo e estável da simulação, garantindo uma maior integridade da simulação.

Figure 12: Quantidade máxima de operações

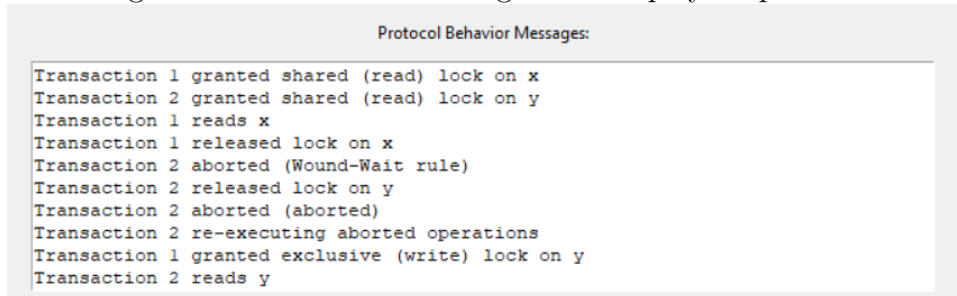


Fonte: Autoria própria.

Na Figura 13, é possível observar que, em determinadas circunstâncias, ocorre uma discrepância na ordem das mensagens, atribuída a um bug identificado na fila de prioridade do Python, juntamente com a aplicação. Este fenômeno se manifesta com maior

frequência durante a execução do protocolo wound-wait. Reconhecemos que essa irregularidade pode comprometer a integridade das operações em casos específicos. Estamos cientes da relevância da consistência no contexto do sistema desenvolvido e que essa limitação pode comprometer a experiência do usuário.

Figure 13: Ordem das mensagens no display do protocolo



```
Protocol Behavior Messages:
Transaction 1 granted shared (read) lock on x
Transaction 2 granted shared (read) lock on y
Transaction 1 reads x
Transaction 1 released lock on x
Transaction 2 aborted (Wound-Wait rule)
Transaction 2 released lock on y
Transaction 2 aborted (aborted)
Transaction 2 re-executing aborted operations
Transaction 1 granted exclusive (write) lock on y
Transaction 2 reads y
```

Fonte: Autoria própria.

Foi desenvolvida uma interface de usuário (UI) que simula a execução dos protocolos wound-wait e wait-die. A implementação destes protocolos é essencial para sistemas de gerenciamento de transações, pois ambos oferecem estratégias distintas para lidar com conflitos e evitar deadlocks, proporcionando uma maior confiabilidade e desempenho no processamento de dados. A UI desenvolvida não apenas demonstra visualmente essas interações, mas também permite aos usuários experimentar e entender as nuances de cada protocolo bem como simular sua execução em diferentes cenários.