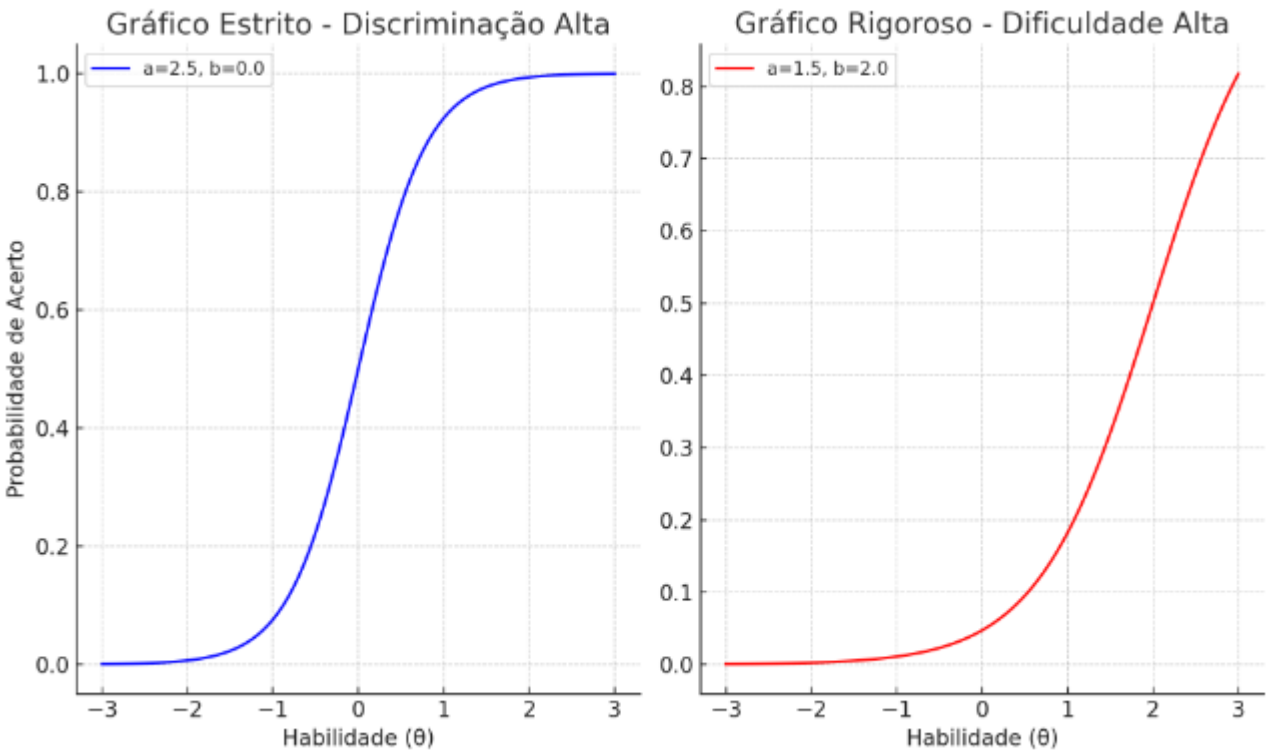
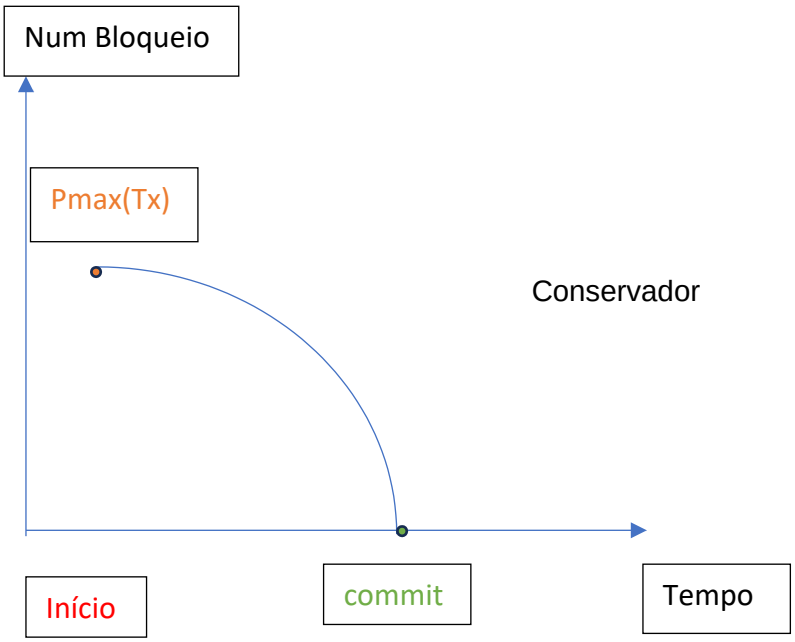
IFBA - INSTITUTO FEDERAL DA BAHIA

Banco de dados II

Aluna: Emillyn Souza Panserini





Questão 2

```
#include <stdio.h>
#include <stdbool.h>

#define NUM_RECURSOS 3
#define NUM_PROCESSOS 5

int max_recursos[NUM_PROCESSOS][NUM_RECURSOS] = {
    {7, 5, 3},
    {3, 2, 2},
    {9, 0, 2},
    {2, 2, 2},
    {4, 3, 3}
};

int alocados[NUM_PROCESSOS][NUM_RECURSOS] = {
    {0, 1, 0},
    {2, 0, 0},
    {3, 0, 2},
    {2, 1, 1},
    {0, 0, 2}
};

int disponiveis[NUM_RECURSOS] = {3, 3, 2};

bool is_deadlocked() {
    int i, j;
    int alocados_temp[NUM_PROCESSOS][NUM_RECURSOS];
    int disponiveis_temp[NUM_RECURSOS];
    bool processo_concluido[NUM_PROCESSOS] = {false};

    for (i = 0; i < NUM_PROCESSOS; i++) {
        for (j = 0; j < NUM_RECURSOS; j++) {
            alocados_temp[i][j] = alocados[i][j];
        }
    }

    for (i = 0; i < NUM_RECURSOS; i++) {
        disponiveis_temp[i] = disponiveis[i];
```

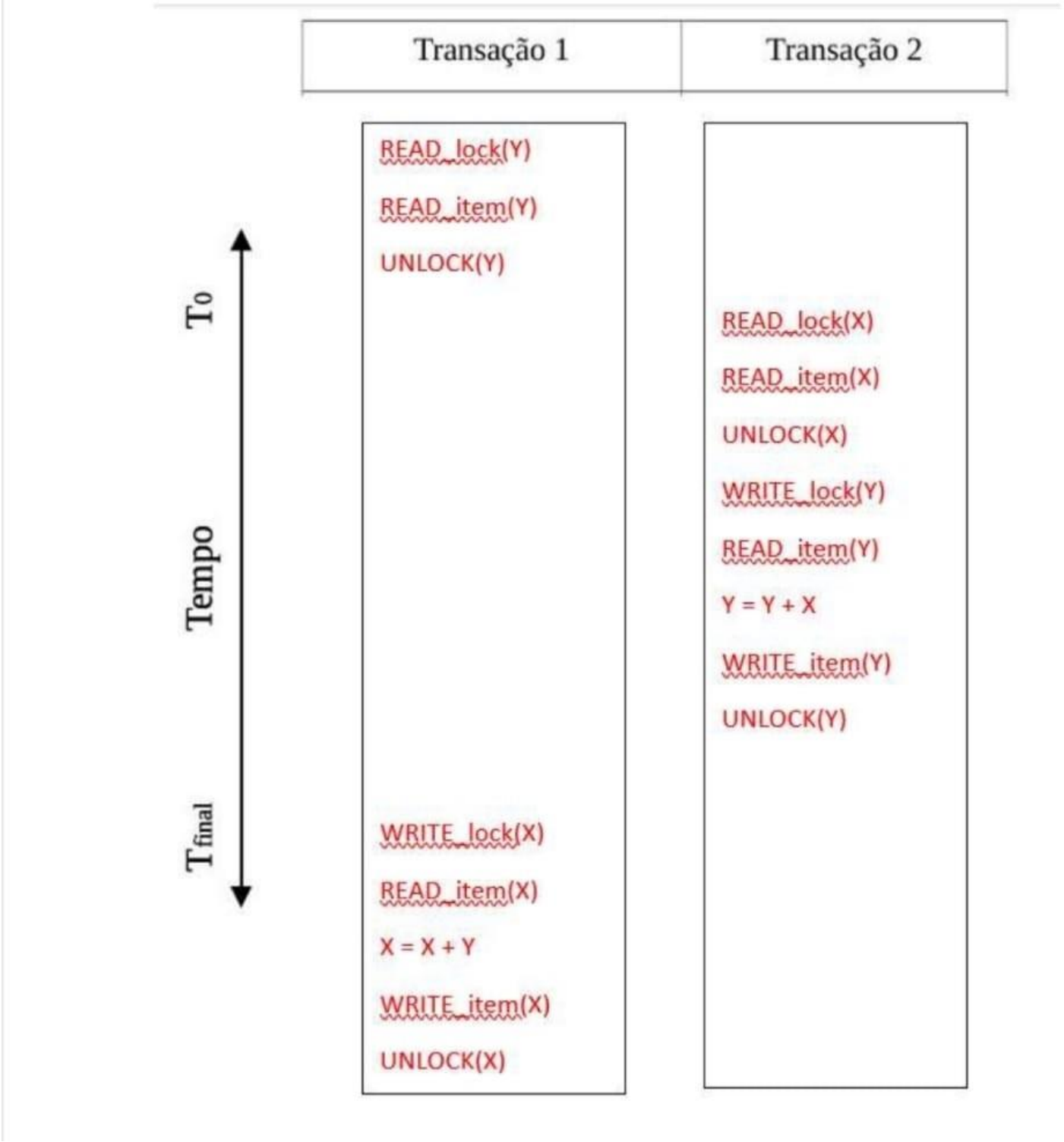
```
    }

    bool deadlock = false;
    while (true) {
        bool algum_processo_alocado = false;
        for (i = 0; i < NUM_PROCESSOS; i++) {
            if (!processo_concluido[i]) {
                bool pode_alocar = true;
                for (j = 0; j < NUM_RECURSOS; j++) {
                    if (alocados_temp[i][j] > disponiveis_temp[j]) {
                        pode_alocar = false;
                        break;
                    }
                }
                if (pode_alocar) {
                    processo_concluido[i] = true;
                    algum_processo_alocado = true;
                    for (j = 0; j < NUM_RECURSOS; j++) {
                        disponiveis_temp[j] += alocados_temp[i][j];
                    }
                    break;
                }
            }
        }
        if (!algum_processo_alocado) {
            deadlock = true;
            for (i = 0; i < NUM_PROCESSOS; i++) {
                if (!processo_concluido[i]) {
                    deadlock = false;
                    break;
                }
            }
            break;
        }
    }

    return deadlock;
}

int main() {
    if (is_deadlocked()) {
        printf("Deadlock detectado!\n");
    } else {
        printf("Sem deadlock.\n");
    }
    return 0;
}
```

Questão 3



Duas Transações:

A transação T2 lê um dado;

A transação T1 atualiza o valor deste dado;

A transação T2 também atualiza o valor deste dado;

No caso descrito precisa haver um controle de concorrência, para a transação T1 o valor do dado estará desatualizado. Uma solução é a serialização pelo Bloqueio em Duas Fases: A transação possui uma fase decrescimento, em que novos bloqueios nos itens podem ser adquiridos mas não liberados, e uma fase de encolhimento, aonde os bloqueios existentes podem ser liberados, mas novos. bloqueios não podem ser adquiridos.

Questão 4

T1 - Esta transação foi concluída antes do último checkpoint, então será mantida e não requer nenhuma ação adicional após a falha do sistema.

T2 Esta transação começou antes do último checkpoint e não foi concluída antes da falha do sistema. No entanto, como os efeitos da transação podem já ter sido parcialmente aplicados**,** T2 será refeita (REDO) para garantir que suas alterações estejam completamente aplicadas.

T3 Esta transação começou antes do último checkpoint e foi concluída antes da falha. Como essa transação foi finalizada, ela será mantida sem necessidade de REDO ou UNDO.

T4 Esta transação começou após o último checkpoint e não foi concluída antes da falha do sistema. Portanto, seus efeitos devem ser removidos. T4 será desfeita (UNDO).

T5 Esta transação também começou após o último checkpoint e não foi concluída. Da mesma forma que T4**,** T5 será desfeita (UNDO) para garantir a consistência.

Questão 5

O bloqueio de duas fases conservativo é livre de deadlock e impede a ocorrência de impasse, pois o lock (bloqueio)é feito por uma transação de todos os itens necessários da execução e só é liberada após possuir todos os recursos necessários.

O bloqueio de duas fases rigoroso reduz a possibilidade de deadlock e impasse, pois os bloqueios são liberados apenas quando a transação é confirmada ou abortada. Os bloqueios são mantidos até que a transação chegue a um ponto de decisão final.

Questão 6

Neste resumo, será abordado serialização de transações e schedule. Definimos os sistemas de processamento de transações como sistemas com grande banco de dados e centenas de usuários simultâneos que executam transações, as quais incluem uma ou mais operações de acesso ao banco, sendo inserção, exclusão, modificação ou recuperação.
O processamento das transações do banco de dados deve ser concluído para que não haja inconsistência. Além disso, sistemas podem ser classificados tanto como monousuário, no máximo um usuário pode usar, quanto multiusuário, vários usuários podem acessar simultaneamente. Mesmo que os sistemas operacionais possam executar alguns comandos de um processo, a execução é intercalada.

Os limites de uma transação são devidamente explícitos, determinados por begin transaction e end transaction em um programa de aplicação. Todas e quaisquer operações de acesso ao banco de dados entre esses comandos são consideradas formando uma transação. Um programa pode conter várias transações, caso tenha vários limites de transações.
Chamamos de transação de leitura se não houver atualização no banco de dados, do contrário chamamos de leitura-gravação. E para que possamos acessar o banco de dados, usamos as operações read_item(X) e write_item(X). Ademais, operações adicionais como begin_transaction, read, write, end_transaction, commit_transaction e rollback.

Como as transações podem ocorrer simultaneamente é necessário um controle de concorrência, pois as atualizações podem ser perdidas, podem ter leituras sujas (atualização temporária), resumo incorreto, leitura não repetitiva. Também é necessário que tenha recuperação, o sistema de banco de dados é responsável por assegurar que as operações sejam concluídas com sucesso e tenha um registro permanente no banco de dados. As transações podem ser confirmadas ou abortadas, podendo ter estado ativo, parcialmente confirmado, estado confirmado, estado de falha e estado terminado. Falhas que podem ocorrer: falha do computador (do sistema), erro de transação ou do sistema, erros locais ou condições de exceção detectadas pela transação, imposição de controle de concorrência, falha de disco e problemas físicos e catástrofes.

Chamamos de log do sistema arquivos sequenciais, somente por inserção, que fica alocado no disco, não sendo afetado por falhas (a menos que seja falha de disco ou catastrófica). Sua função é de recuperação em caso de falhas. Nesse registro (log), temos identificadores como start_transaction, write_item, read_item, commit e abort. Além disso, transações possuem propriedades desejáveis, como atomicidade, preservação da consistência, isolamento e durabilidade ou permanência.

Por seguinte, definimos schedules (ou histórico) como ordenação das operações das transações, para duas operações devem ocorrer uma antes da outra, consideramos operações como totais ou parciais. Para que tenhamos recuperação e controle de concorrência destacamos interesse principal nas operações read_item, write_item, commit e abort. As operações em schedule entram em conflito com as seguintes condições: pertencem a diferentes transações, acessam o mesmo item e pelo menos uma delas é write_item. Podemos ter a intuição de entender que duas operações conflitam quando a mudança da ordem é relevante no resultado, chamado de conflito de leitura-gravação.
Contudo, schedules podem ser seriais, não seriais e serializáveis por conflito. Chamamos de seriais operações executadas em sequência (consecutivamente), sem intercalações. Consequentemente os não seriais possuem intercalações. Para entender a serialização por conflito é necessário saber que um schedule é serializável se for equivalente a algum schedule serial com a mesma quantidade de transações, ou seja, dois schedules são chamados de equivalentes no resultado se produzirem o mesmo estado final do banco de dados. Estedemos que schedules equivalentes em conflito são aqueles que se a ordem de duas operações em conflito for a mesma nos dois shedules. Portanto, se o schedule for equivalente (em conflito) com outro, chamamos de seralizável de conflito. Para além, podemos usar os shedules para o controle de concorrência usando a técnica de protocolos (conjunto de regras).