Estudo de caso de ocorrência de DEADLOCK

Aluno: Gabriel Vieira Soriano Aderaldo
Matrícula: 1710538

SUMÁRIO

- Conhecendo mais o aluno
- Introdução
- Podemos evitar um deadLock?
- Vendo um pouquinho de código
- Conclusões

Conhecendo mais o aluno

Nome: Gabriel Vieira Soriano Aderaldo

Curso: Ciência da computação

Semestre: Melhor não perguntar

Stack: Front - End mobile

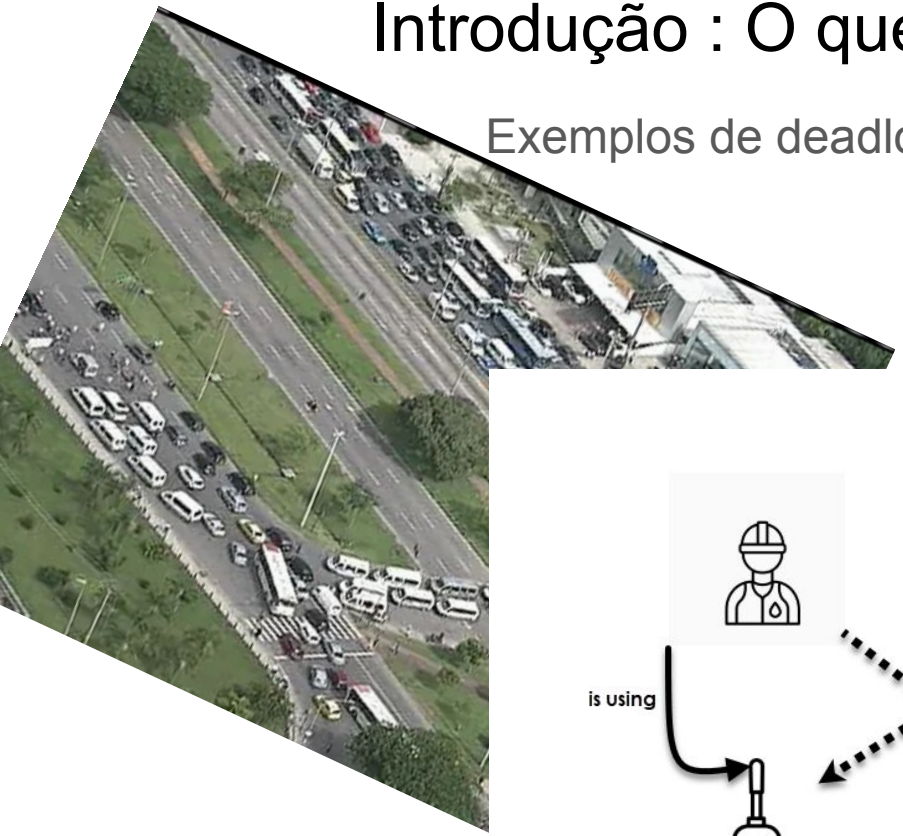
Especialidade: IOS

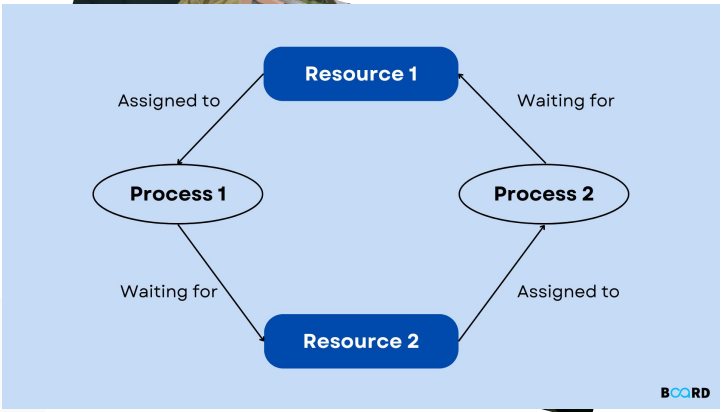
Empresa e cargo: Positivo - Dev JR

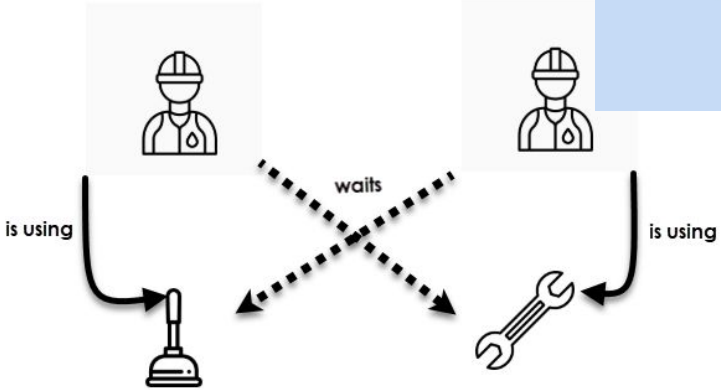
github:
https://github.com/gabrieladeraldo

Introdução : O que diabos é um deadLock?

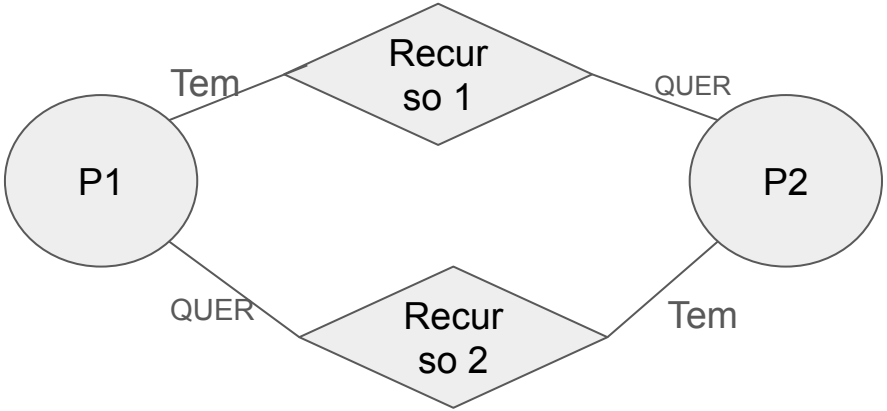
Exemplos de deadlock na vida real







Fluxo de um deadlock



Espera pra sempre

Podemos evitar um deadlock 100% ?

RESPOSTA CURTA: NÃO

- **Condição de não-preempção**: recursos já alocados a processos não podem ser tomados à força. Eles precisam ser liberados explicitamente pelo processo que detém a sua posse;
- **Condição de exclusividade mútua**: cada recurso ou está alocado a um processo ou está disponível;
- **Condição de posse-e-espera**: cada processo pode solicitar um recurso, ter esse recurso alocado para si e ficar bloqueado, esperando por um outro recurso;
- **Condição de espera circular**: deve existir uma cadeia circular de dois ou mais processos, cada um dos quais esperando por um recurso que está com o próximo integrante da cadeia.

Mas existem técnicas para evitar um!

Evitando um deadlock usando como exemplo o mutex

Exclusão Mútua:O uso do mutex (pthread_mutex_t mutex) garante que apenas um thread (seja produtor ou consumidor) possa acessar o buffer por vez. Isso evita condições de corrida e garante a consistência dos dados no buffer.

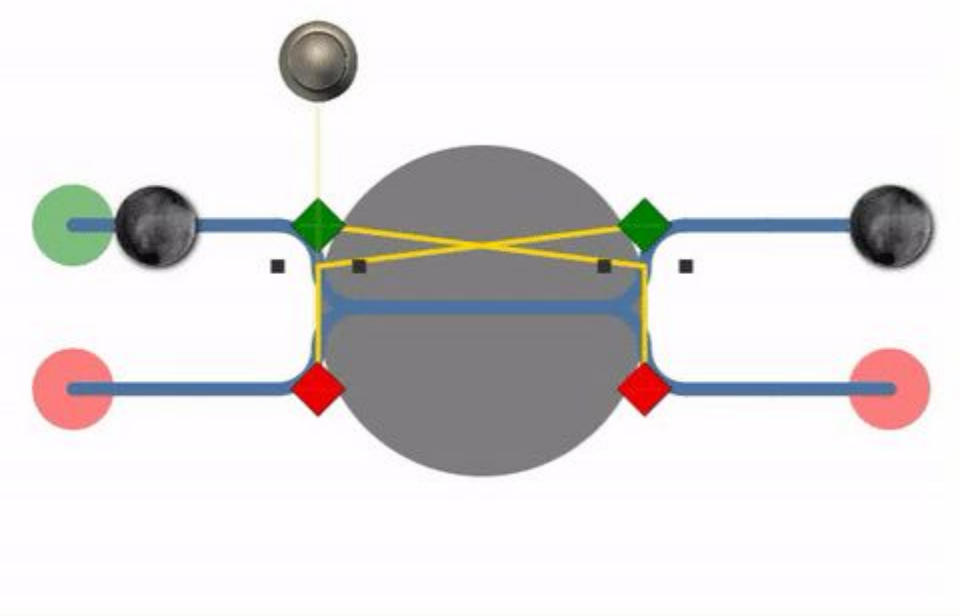
Hold and Wait (Aguarda e Mantém): Nenhum dos threads (produtores ou consumidores) mantém recursos enquanto espera por outros. Eles liberam e sinalizam os semáforos full e empty após cada operação, permitindo que outros threads possam prosseguir.

Sem Preempção:Não há preempção de recursos utilizados pelos threads, ou seja, um thread não pode tomar à força recursos que estão sendo utilizados por outro. O mutex é usado para garantir que somente um thread possa acessar o buffer por vez.

Circular Wait (Espera Circular): Não há uma situação em que um thread esteja esperando por recursos que são mantidos por outro thread na fila circular. A utilização dos semáforos full e empty permite que os threads produtores e consumidores sigam um protocolo específico ao acessar o buffer, evitando uma espera circular que poderia levar ao deadlock.

EXEMPLO VISUAL

Exemplo de um semáforo



Vendo um pouquinho de código

```c
void *producer(void *arg) {
    while (produced_count < MAX_PRODUCED) {
        // Solicitação de recursos
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);

        // Verifica se há espaço no buffer para produzir
        int i = 0;
        while (buffer[i] != 0 && i < BUFFER_SIZE) {
            i++;
        }

        if (i < BUFFER_SIZE) {
            buffer[i] = produced_count + 1;
            printf("Produtor produz um item:\nBarra: [ ");
            for (int j = 0; j < BUFFER_SIZE; ++j) {
                if (buffer[j] != 0) {
                    printf("# ");
                } else {
                    printf("  ");
                }
            }
            printf("]\n");

            produced_count++;
        }

        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }

    printf("Produtor encerrou a produção.\n");
    pthread_exit(NULL);
}
```

```c
void *consumer(void *arg) {
    int consumer_id = *((int *)arg);

    while (1) {
        // Solicitação de recursos
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        // Verifica se há itens no buffer para consumir
        int i = 0;
        while (buffer[i] == 0 && i < BUFFER_SIZE) {
            i++;
        }

        if (i < BUFFER_SIZE) {
            printf("Consumidor_%d consome o item %d:\nBarra: [ ", consumer_id, buffer[i]);
            buffer[i] = 0;
            for (int j = 0; j < BUFFER_SIZE; ++j) {
                if (buffer[j] != 0) {
                    printf("# ");
                } else {
                    printf("  ");
                }
            }
            printf("]\n");
            consumed_count++;
        } else {
            if (produced_count >= MAX_PRODUCED) {
                printf("Consumidor_%d: Não há mais produtos para consumir.\n", consumer_id);
                pthread_mutex_unlock(&mutex);
                sem_post(&full);
                break;
            }
            Gabriel Aderaldo, há 3 horas • [0] - codigo completo.
        }

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        // Liberando recursos
        sem_post(&full);
        sleep(2); // Adiciona um atraso de 2 segundos para demonstrar as operações
    }

    printf("Consumidor_%d encerrou o consumo.\n", consumer_id);
    pthread_exit(NULL);
}
```

Concluindo

Deadlocks são comuns quando usamos paralelismo, porém existem várias técnicas que já foram estudadas para garantirmos o mais próximo de conseguir evitar que eles ocorram.