Exclusão Mútua - Mutex

Descrição do problema

O problema de exclusão mútua parte da necessidade de múltiplos clientes acessarem o mesmo servidor sem que haja problemas com a sua concorrência e mutualidade de acesso. Dessa forma, se mais de um cliente possuir acesso a um mesmo servidor, deve-se construir rotinas para que isso seja permitido sem que haja conflitos e impossibilidade de acesso por qualquer uma das partes.

Detalhes da implementação

Para que o problema de exclusão mútua fosse resolvido, foi instanciado um servidor (detalhes de instanciação foram mostrados anteriormente) e múltiplos clientes foram representados por requisições http reproduzidos através de uma plataforma de API, que no nosso caso foi o postman. Essa abordagem é uma implementação do algoritmo de mutex centralizado, onde cada cliente requisita a um “coordenador” o acesso a zona crítica. Cada requisição para acesso ao servidor possui uma resposta de acordo com o estado do servidor. Além disso, o servidor foi particionado em 3 regiões de acesso, e cada uma possui uma fila própria para acesso. Os detalhes em termos de código serão mostrados a seguir.

Cada uma das regiões da zona crítica foram definidas da seguinte forma:

```
let regions = {
 A: { state: enumeration.FREE, user: null, queue: [] },
 B: { state: enumeration.FREE, user: null, queue: [] },
 C: { state: enumeration.FREE, user: null, queue: [] },
};
```

Quando é realizado o método GET no servidor por um cliente em uma região específica, é feita uma análise da fila da região e sua disponibilidade. A função para análise do request é a seguinte:

```
APP.get("/open-session/:user/:region", (req, res) => {
```

Caso seja solicitado o acesso a uma região que não existe, o retorno ao cliente é “FORBIDDEN”

```
if (!regions.hasOwnProperty(region)) {
   message = enumeration.FORBIDDEN;
   res.json({ message, regions });
   return;
```

```
}
```

Caso o estado da região seja “BLOCKED” e o cliente em questão não está na fila, ele é inserido na fila e é retornado o estado “ENQUEUED”. Caso ele já esteja na fila, mesmo que ele seja o primeiro, é retornado o estado “WAITING”, pois ainda está esperando a região do servidor ficar livre.

```
if (regions[region].state === enumeration.BLOCKED)
  if (!regions[region].queue.includes(user)) {
    regions = {
      ...regions,
      [region]: {
        ...regions[region],
        queue: [...regions[region].queue, user],
      },
    };
    message = enumeration.ENQUEUED;
  } else message = enumeration.WAITING;
```

Se o cliente é o primeiro da fila e é feito um request de acesso a região por ele mesmo, e a região está disponível, é garantido o acesso com o estado “GRANTED”

```
else if (regions[region].queue.length > 0)
  if (regions[region].queue[0] === user) {
    regions = {
      ...regions,
      [region]: {
        state: enumeration.BLOCKED,
        user,
        queue: regions[region].queue.filter((u) => u !== user),
      },
    };
    message = enumeration.GRANTED;
```

Se o servidor está disponível porém o cliente que fez o request não é o primeiro da fila, é retornado o estado “WAITING” também

```
else {
    regions = {
      ...regions,
      [region]: {
        ...regions[region],
        queue: [...regions[region].queue.filter((u) => u !== user), user],
```

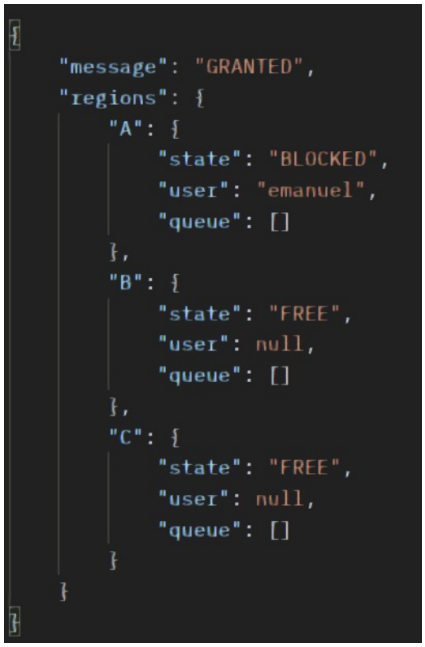
```
      },
    };
    message = enumeration.WAITING;
```

E caso não haja ninguém na fila e a região está disponível, o acesso é garantido e o estado da região do servidor passa a ser “BLOCKED”

```
else {
  regions = {
    ...regions,
    [region]: {
      ...regions[region],
      user,
      state: enumeration.BLOCKED,
    },
  };
  message = enumeration.GRANTED;
}
res.json({ message, regions });
});
```

Resultados e comentários

Inserção do usuário “Emanuel” na região A com a fila vazia e o servidor desocupado. A região é bloquada e o usuário com o acesso garantido.



Agora, o usuário “Lucca” tenta acessar a região A, porém ela já está sendo usada. Portanto é enfileirada

Sessão da região A é encerrada, e agora fica vazia

E por fim, usuário “Lucca” que estava na fila realiza outro request para saber se a região ainda está ocupada. Como não está. pode acessar a região do server, já que ele é o próximo na fila

Um comentário interessante é que poderia haver uma lógica do próximo usuário da fila automaticamente entrar no servidor que está liberado, porém da forma como foi feito já se consegue ter uma boa ideia de como o algoritmo deve funcionar.