Entrega de refeições com Programação Concorrente

Mathaus C. Huber

[1]Universidade Federal de Pelotas (UFPEL) – Discente do Curso Superior de Ciência da Computação
R. Gomes Carneiro, 1 - Centro – 96075-630 – Pelotas – RS – Brazil

mchuber@inf.ufpel.edu.br

Abstract. *This article describes the report of the first practical evaluation of the discipline of Operating Systems, given to the fifth semester of the Computer Science course, of the Federal University of Pelotas, in which it refers to the solution of the proposed problem of delivery of meals aiming at saving fuel using the concept of concurrent programming.*

Resumo. *Este artigo descreve o relatório da primeira avaliação prática da disciplina de Sistemas Operacionais, conferida ao quinto semestre do curso de Ciência da Computação, da Universidade Federal de Pelotas, no qual se refere a solução do problema proposto de entrega de refeições visando economia de combustível utilizando o conceito de programação concorrente.*

1. Informação Geral

O problema proposto consiste em utilizar Programação Concorrente para simular a entrega de alimentos de um ponto A até um ponto B, tendo o funcionário como o criador das refeições (produtos), e o entregador fazendo o transporte das mesmas. Contudo, o entregador somente poderá fazer o transporte quando o número de refeições acumuladas em sua sacola chegar em 10 (dez). Naturalmente notamos a sua analogia com o problema do Produtor e do Consumidor, também conhecido como problema do buffer limitado, sendo o funcionário analogo ao produtor, o entregador ao consumidor e, a sacola ao buffer.

É notável que, este é um cenário que busca exemplificar de forma clara, situações de impasses que ocorrem no gerenciamento de processos de um sistema operacional, e existem alguns problemas a serem identificados, como acessos ilegais a certos recursos e manter um sincronismo entre as threads. Algumas soluções já são muito conhecidas quando se trata do problema do produtor e consumidor, bem como, o uso de semáforos para o controle de acesso a recursos compartilhados, o que vai ser bem abordado, e melhor elucidado ao longo do relatório.

2. Objetivos

O objetivo principal desse relatório é elucidar o conceito de programação concorrente no problema de entregas de refeições com economia de combustível, que faz analogia ao problema do produtor e do consumidor. Confesso que, para quem está acostumado com a programação procedural, esse problema pode se tornar um pouco complexo, contudo existem algumas vantagens de se fazer uso desse conceito. Algumas delas são:

- Pode-se ter uma visão mais clara de certas etapas dos programas;
- Dependendo da programação, erros ”fatais”podem afetar apenas seu processo, preservando a integridade dos demais;
- Até mesmo com um único núcleo de processador, pode-se otimizar processos de E/S, pois uma parte do programa não precisará esperar o disco responder para continuar executando;

3. Estratégia

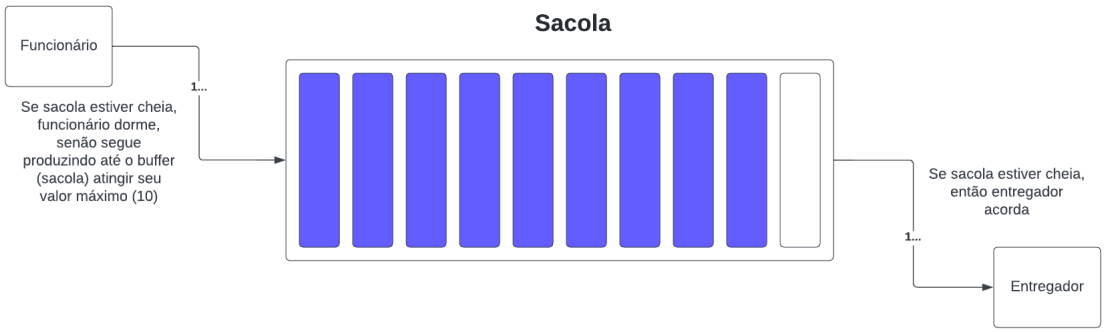
De forma que, eu conseguisse resolver o problema proposto, primeiro teria que ser montado uma estratégia, como em todos os problemas da vida real de um programador, não é possível sairmos apenas codando. Pensando nisso, me propus a identificar alguns problemas e possíveis soluções que teriam que ser implementadas, sendo elas:

- O produtor insere em posição: Ainda não consumida
- O consumidor remove de posição: Já foi consumida
- Espera ociosa X Escalonamento do processo X Uso CPU

Possíveis soluções para estes problemas:

- Exclusão mutua (semáforos)
- Fim da espera ociosa: Dormir(dow)/acordar(up) X Semáforos full/empty ou Mutex (Mutual exclusion)

Com o intuito de melhor elucidar a estratégia da entrega de refeições, criei um diagrama na ferramenta Lucidchart, onde simula os processos funcionário, entregador e a sacola. Basicamente o funcionário, pode ter mais de um, produz alimentos e põem na sacola, dormindo se a sacola estiver cheia e acordando se a sacola estiver vazia. Por outro lado, o entregador, pode ter mais de um, entrega os alimentos, vamos dizer aqui que ele consome, na vida real não seria a melhor palavra, mas apenas para fazer analogia ao consumidor, se a sacola estiver cheia, ou seja, com dez produtos, ele acorda, caso contrário ele dorme.



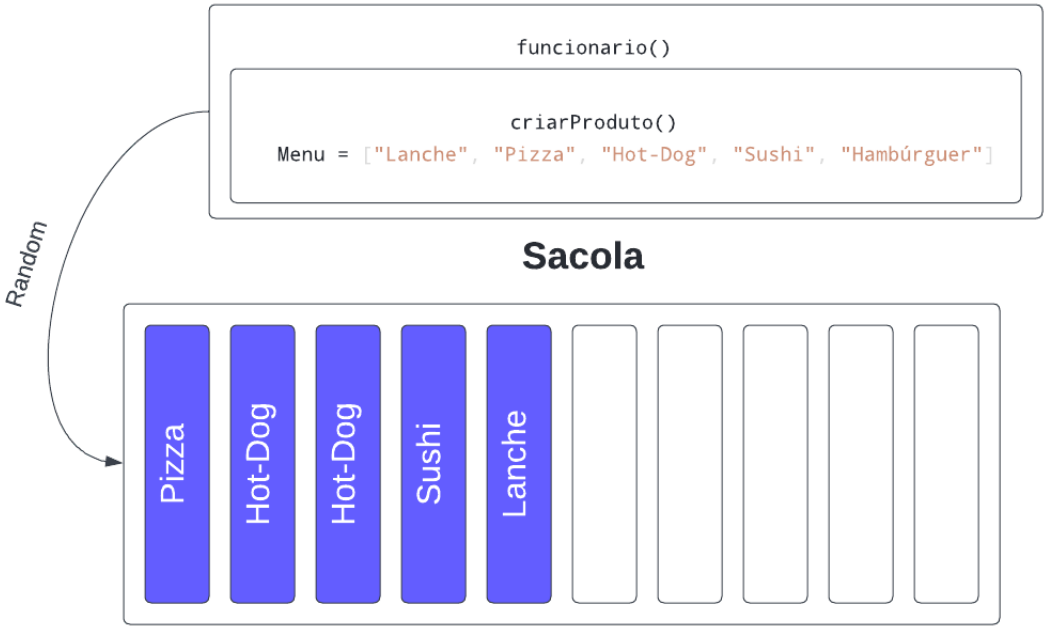
4. Implementação

Para fazer a implementação da solução do problema proposto utilizando o conceito de programação concorrente, que diferentemente da programação sequencial, a que estamos habituados a programar, tem por base a separação de processos dentro um mesmo programa, através de threads para otimização do processamento, utilizei a linguagem python, por motivos de maior familiaridade e prática com seus recursos e bibliotecas.

Esse problema proposto, como já dito antes, no meu ponto de vista, é uma analogia ao problema do produtor e o consumidor, onde o produtor seria o funcionário responsável pela criação dos produtos, e o consumidor sendo o entregador, que leva o produto do ponto A até o ponto B, assim que a sacola estiver cheia. Basicamente, este problema consiste em um conjunto de processos que compartilham um mesmo buffer. Os processos chamados pela função funcionario() põem informação no buffer. Os processos chamados pela função entregador() retiram informação deste buffer. Deixo abaixo a saída de uma execução do programa de entrega de refeições utilizando programação concorrente. Para o cálculo do lucro do entregador foi levado em conta o preço da gasolina a 7,323R$ por litro, o valor da entrega cobrado pelo motoboy a 3,50R$, e levando em consideração que o entregador consegue fazer até cinco entregas com um litro de gasolina. Lembrando que, não era necessário calcular o lucro do entregar, contudo, achei mais interessante mostrar o seu lucro e o total de entregas da noite, de forma a simular um cenário real.

```
Funcionario produzindo Hot-Dog - Pedidos:  1
Funcionario produzindo Hot-Dog - Pedidos:  2
Funcionario produzindo Sushi - Pedidos:  3
Funcionario produzindo Lanche - Pedidos:  4
Funcionario produzindo Pizza - Pedidos:  5
Funcionario produzindo Sushi - Pedidos:  6
Funcionario produzindo Pizza - Pedidos:  7
Funcionario produzindo Lanche - Pedidos:  8
Funcionario produzindo Sushi - Pedidos:  9
Funcionario produzindo Sushi - Pedidos:  10
Motoboy entregando Sushi
Motoboy entregando Sushi
Motoboy entregando Lanche
Motoboy entregando Pizza
Motoboy entregando Sushi
Motoboy entregando Pizza
Motoboy entregando Lanche
Motoboy entregando Sushi
Motoboy entregando Hot-Dog
Motoboy entregando Hot-Dog
Funcionario produzindo Hambúrguer - Pedidos:  11
Funcionario produzindo Sushi - Pedidos:  12
Funcionario produzindo Hambúrguer - Pedidos:  13
Funcionario produzindo Lanche - Pedidos:  14
Funcionario produzindo Pizza - Pedidos:  15
Funcionario produzindo Hot-Dog - Pedidos:  16
Funcionario produzindo Pizza - Pedidos:  17
Funcionario produzindo Hambúrguer - Pedidos:  18
Funcionario produzindo Pizza - Pedidos:  19
Funcionario produzindo Pizza - Pedidos:  20
Motoboy entregando Pizza
Motoboy entregando Pizza
Motoboy entregando Hambúrguer
Motoboy entregando Pizza
Motoboy entregando Hot-Dog
Motoboy entregando Pizza
Motoboy entregando Lanche
Motoboy entregando Hambúrguer
Motoboy entregando Sushi
Motoboy entregando Hambúrguer
Encerrou a noite, Motoboy entregou um total de: 20 pedidos e obteve um lucro de: 40.71 R$
```

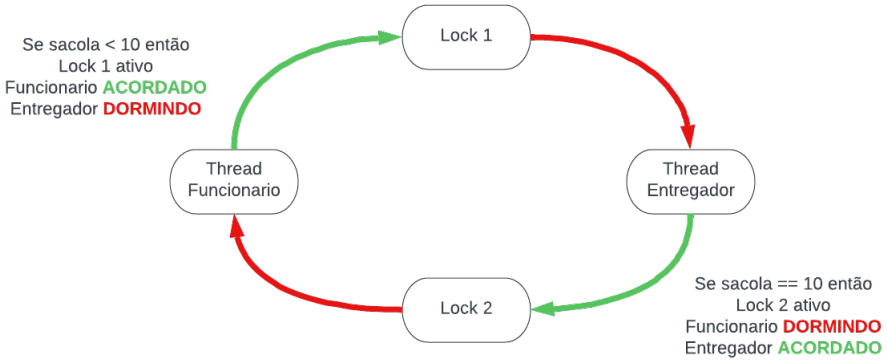
Minha ideia era deixar o programa o mais criativo possível, simulando os processos de um serviço de delivery da vida real. Dessa forma, utilizei uma função randômica para escolher um valor aleatório dentro do nosso menu, ou cardápio, que é um vetor de strings, vai sendo escolhido então o pedido do cardápio do menu (vetor em sua posição aleatória menu[aleatorio]), e posteriormente através da função append do python vai sendo colocado no nosso buffer uma escolha aleatória do cardápio. De maneira que, podemos parar a execução do nosso código, cologuei um limitador, sendo ele por limite de processos, ou seja, um laço de repetição que para com a condição de 40 processos feitos, sendo eles produção e entrega, gerando um total de 20 entregas.



Como sabemos, precisamos nos preocupar com acessos ilegais a certos recursos que são compartilhados entre os processos, e manter sincronismo entre os mesmos. Para controlarmos o acesso a essas variáveis e termos sincronismo nas operações, eu utilizei semáforos, que basicamente é um tipo abstrato de dados que tem como função o controle de acesso a recursos compartilhados num ambiente multitarefa. Outro controle importante é a espera ociosa que o programa irá gerar, pois quando um processo não está liberado deve entrar em estado de espera para não consumir processamento de graça, e deve ser avisado quando pode voltar a processar, com isso a ideia da analogia de acordar o entregador quando a sacola estiver cheia e voltar a dormir quando estiver vazia, e a mesma coisa com o processo do funcionário.

Foi dito no enunciado que poderíamos fazer uso de semafóros e/ou mutex, portanto, eu utilizei a função Condition() da biblioteca threading, para fazer os bloqueios necessários. Basicamente através das funções acquire(), wait(), release(), a thread funcionario produz e libera o bloqueio, então a thread entregador espera e depois adquire. Agora,

há uma condição de corrida entre o lançamento da thread entregador e a readquirição da thread funcionario após o retorno do wait(). Se a thread funcionario tentar readquirir primeiro, será desnecessariamente ressuspenso até que a thread entregador release() sejam completados. Basicamente o que é mostrado na figura 4, na função de trava entre as threads, isso é visto especificamente nas linhas 33, 38, 43, 55, 59 e 67 do código.



A biblioteca mais importante do projeto foi a threading do python, que nos proporciona a possibilidade de programação concorrente com as funções de: controle de threads, criação de threads, suspensão de threads, execução e controle de exclusão mútua por semáforos binários, para controle da seção crítica, através da função Semaphore().

5. Conclusão

Em vista dos argumentos apresentados, considerei uma atividade bem difícil de ser implementada, pois foi o meu primeiro projeto utilizando programação concorrente, e muitos problemas ocorreram durante a criação do código. Antes de resolver o problema de modo concorrente, implementei a solução de maneira procedural, dessa forma, não sendo uma analogia ao problema do produtor e consumidor, e sim, apenas a criação e entrega de alimentos por um restaurante, deixo o código feito de maneira procedural no próximo tópico desse relatório, ”Links Disponíveis e Referências”, assim como, o código utilizando programação concorrente.

Contudo, embora tenha sido uma atividade complexa, particularmente, gostei muito do contexto em que foi abordado o problema para o conceito de programação concorrente. Confesso que, foi um tanto divertido a implementação também. Achei que seria bem simples inserir o conceito de programação concorrente, pois resolvi de forma procedural de maneira rápida, contudo, tive alguns problemas na parte de espera ociosa que me roubaram um pouco de tempo e gastaram um alto consumo da minha CPU (cerébro).

6. Links Disponíveis e Referências

6.1. Links Disponíveis

6.1.1. Código-fonte

https://colab.research.google.com/drive/1PGEeFVAFC5s8XcNioeX1MMPEMPsP_7Fj?usp=sharing

6.2. Referências Bibliográficas

6.2.1. Bibliografia Referenciada

Silberschatz, A. **Fundamentos de sistemas operacionais. 8a. edição.** Rio de Janeiro: LTD. 2010

Tanenbaum, A. S. **Sistemas Operacionais: Projeto e Implementação. 3a. edição.** São Paulo: Prentice-Hall, 2010.

Silberschatz, A., Galvin, P., Gagne, G. **Sistemas Operacionais: Conceitos e Aplicações.** Rio de Janeiro: Campus, 2000.

6.3. Bibliografia para Consulta

Kirner, C. **Sistemas Operacionais Distribuídos.** São Paulo: Campus, 1998.

Problema de produtores e consumidores. Wikipedia, 2018. Disponível em: https://pt.frwiki.wiki/wiki/Probl%C3%A8me_des_producteurs_et_des_consommateurs Acesso em: 17/05/2022 às 15:32