**APLICAÇÃO DE SEMÁFOROS NA COORDENAÇÃO DOS PROCESSOS DE DEPÓSITO E SAQUE EM UMA CONTA BANCÁRIA ACESSADA POR QUATRO CLIENTES SIMULTANEAMENTE**

SILVA, Matheus Augusto Veríssimo1; CALDAS, Chrystian Victor da Silva2; CASANOVA, Dalcimar3

*1Student of Electronic Engineering at Amazonas State University, Brazil.*

*2Student of Electronic Engineering at Amazonas State University, Brazil.*

*3Ph.D, Professor of Electric Engineering at Federal Technologic University of Paraná*

***ABSTRACT:*** *This work presents the importance of controlling the access of shared data in a program through an example of four users picking and putting money on a shared bank account. First, it begins by explaining what are critical sections and mutual exclusion, followed by the fundaments of semaphores: the wait() and signal() functions. Second, it proceeds showing the limitations of the semaphores and the need to control the access of the critical sections while maintaining a generous amount of CPU available. Third, it shows a quick step by step of how to use the VM Oracle Virtual Machine. And finally, it begins to show the example given by professor Dalcimar Casanova and its implementation.*

Tabela de Figuras

[Figura 1 – Comparação do funcionamento dos semáforos reais com os virtuais 4](#_Toc103102610)

[Figura 2 - Aula de semáforos realizada pela Neso Academy 6](#_Toc103102611)

[Figura 3 - Explicação do controle de acesso às seções críticas 7](#_Toc103102612)

[Figura 4 - Explicação das funções wait() e signal() 8](#_Toc103102613)

[Figura 5 - Aula de demonstração das limitações dos semáforos tipo spinlock realizada pela Neso Academy 9](#_Toc103102614)

[Figura 6 - Demonstração do funcionamento de um semáforo utilizando uma lista de espera 10](#_Toc103102615)

[Figura 7 – Tela de início do Oracle VM Virtual Box 11](#_Toc103102616)

[Figura 8 - Tela inicial do Ubuntu na máquina virtual criada pelo Oracle VM Virtual Box 12](#_Toc103102617)

[Figura 9 - Demonstração do funcionamento do código realizada pelo professor Dalcimar 13](#_Toc103102618)

[Figura 10 - Código do projeto 14](#_Toc103102619)

[Figura 11 – Execução do programa sem o semáforo 15](#_Toc103102620)

[Figura 12 - Execução do programa com o semáforo 16](#_Toc103102621)

1. **INTRODUÇÃO**

No início da computação, as arquiteturas das máquinas eram simples e capazes de realizar apenas uma operação por vez. Chamamos arquiteturas assim como *monolíticas*.

A necessidade de realização de um alto volume de tarefas de uma maneira ágil e conveniente estimulou o desenvolvimento de arquiteturas computacionais mais complexas. As máquinas começaram a ser capazes de realizar múltiplas tarefas a uma altíssima velocidade.

Apesar de todo esse avanço, mesmo o computador mais avançado fica sobrecarregado com todo o trabalho que lhe é designado.

Semelhante ao que acontece em um atendimento médico, caso não haja regulamentos para os atendimentos, um paciente irá impossibilitar o atendimento do outro. Como resultado, haverá um paciente com diagnóstico insuficiente; um segundo pode receber um diagnóstico errado e um terceiro pode até nem mesmo ser atendido devido ao tempo.

Com a intenção de garantir que os computadores permaneçam sendo uma ferramenta confiável e conveniente foram desenvolvidas técnicas de tratam da concorrência das tarefas. Todas elas têm como base os *semáforos*. Este presente artigo dará ênfase a essa base tão importante para a computação moderna.

Figura – Comparação do funcionamento dos semáforos reais com os virtuais



Fonte: CC BY-NC1

1. **MATERIAIS E MÉTODOS**

Na primeira etapa foi realizado um aprofundamento teórico por meio da leitura da apostila Sistemas Operacionais: Conceitos e Mecanismos e vídeos sobre os semáforos da Neso Academy. Então foi possível pesquisar e compreender as soluções para o Problema de Produtores e Consumidores.

O melhor resultado da pesquisa foi um exemplo publicado no YouTube pelo professor Professor Ph.D Dalcimar Casanova.

O professor Dalcimar fez uma associação do Problema de Produtores e Consumidores com movimentações bancárias de saque e depósito em uma conta bancária que é utilizada por quatro clientes. Como a quantia sacada e o depósito são limitados, é necessário criar um controle.

Figura 2 - Aula de semáforos realizada pela Neso Academy

Texto

Descrição gerada automaticamente

Fonte: Canal da Neso Academy no YouTube2

2Disponível em: <https://www.youtube.com/>[watch?v=XDIOC2EY5JE&list=PLeL6fbUgUKxz8-ADEazyMmRgzokCSxVpx](https://www.youtube.com/watch?v=XDIOC2EY5JE&list=PLeL6fbUgUKxz8-ADEazyMmRgzokCSxVpx). Acesso em: 11 de abril de 2022.

Um programa A pode ser executado por mais de um processo ao mesmo tempo. Porém, caso as rotinas e sub-rotinas do programa A sejam executadas simultaneamente, erros irão ocorrer. Dessa situação, surge a necessidade de estabelecer uma hierarquia de acesso aos dados compartilhados, seguindo ao princípio de FIFO (First In First Out = O primeiro que entra é o primeiro que sai). O cerne do funcionamento desse controle de acesso está no conceito de **Seções Críticas** e **Exclusão Mútua.**

Os trechos do código de um programa que utiliza dados compartilhados são chamados de **Seções Críticas** porque é onde os dados compartilhados estão armazenados.

Figura 3 - Explicação do controle de acesso às seções críticas

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

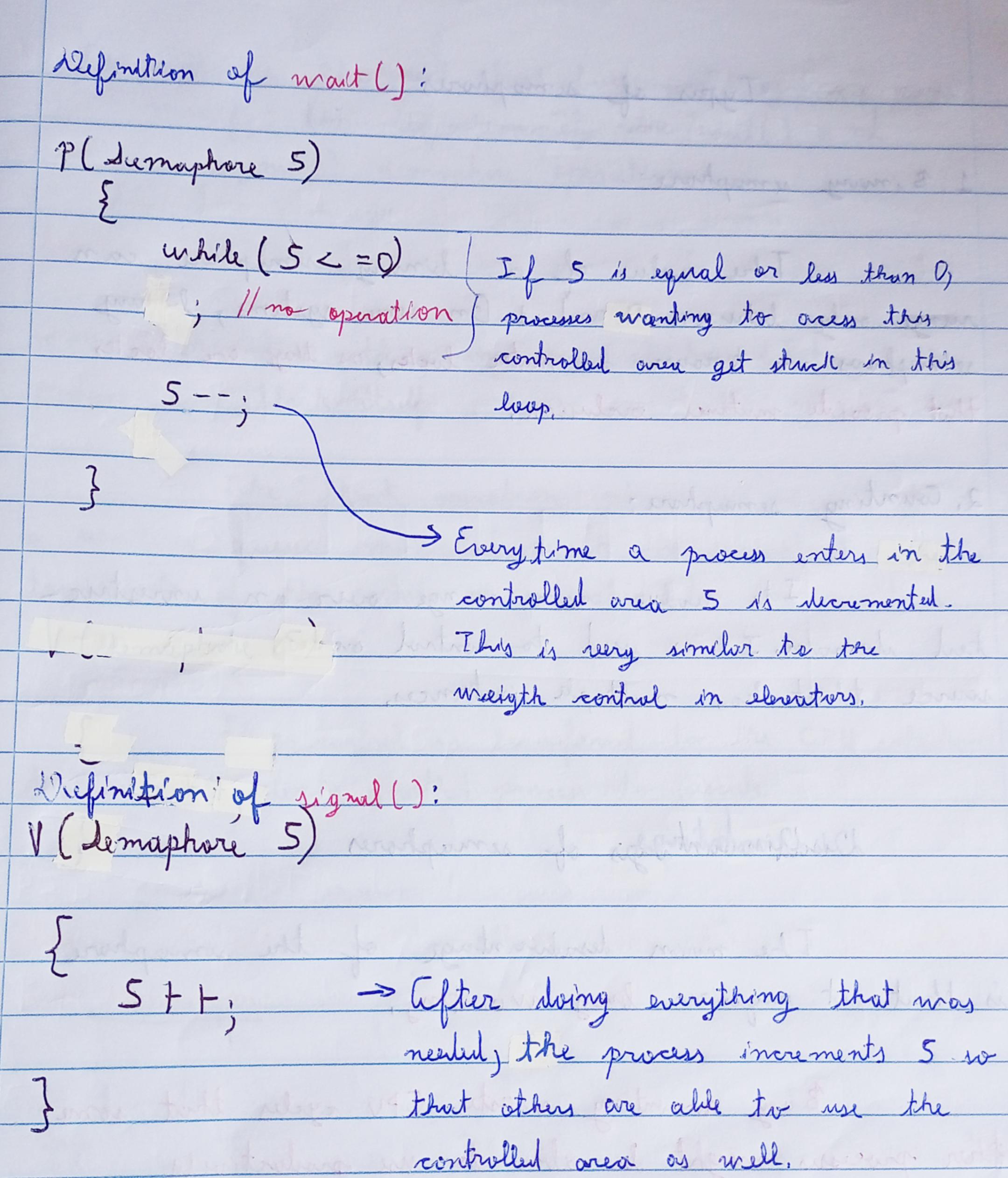
Descrição gerada automaticamente

Fonte: Canal da Neso Academy no YouTube3

3Disponível em: <https://www.youtube.com/>[watch?v=XDIOC2EY5JE&list=PLeL6fbUgUKxz8-ADEazyMmRgzokCSxVpx](https://www.youtube.com/watch?v=XDIOC2EY5JE&list=PLeL6fbUgUKxz8-ADEazyMmRgzokCSxVpx). Acesso em: 11 de abril de 2022.

**Exclusão mútua** refere-se ao fato de uma região crítica só poder ser acessada por um processo por vez.

Figura 4 - Explicação das funções wait() e signal()



Fonte: Autoria própria

S é a variável de controle. Normalmente ela armazena o número 1 inteiro.

Sempre que um processo acessa a uma seção crítica, a variável de controle S é decrementada em uma unidade, tornando-a nula. Isso faz com que qualquer outro processo que tente acessar à seção crítica que está em uso caia em um loop infinito **while(S<=0)**. Depois de sair da seção crítica, o processo incrementa S em uma unidade, liberando o acesso para os outros.

Figura 5 - Aula de demonstração das limitações dos semáforos tipo spinlock realizada pela Neso Academy

Texto

Descrição gerada automaticamente

Fonte: Canal da Neso Academy no YouTube5

5Disponível em: <https://www.youtube.com/watch?v=2cGo2HdA0dM>.

Semáforos que usam loops infinitos **while(S<=0)** são chamados de spinlock. Apesar de controlar bem o acesso às seções críticas, os processos que estão sendo barrados ficam consumindo espaço da CPU por terem que executar loops até chegar a vez de cada, podendo até mesmo atrapalhar os processos que estão utilizando as seções críticas.

Para liberar mais espaço da CPU, em vez de prender os processos em loops eles são postos em uma lista de espera. Então, o CPU Scheduler determina qual será o próximo processo a ser executado conforme foi instruído. Até mesmo o tempo que um processo leva para ser concluído é monitorado.

Figura 6 - Demonstração do funcionamento de um semáforo utilizando uma lista de espera

Uma imagem contendo Texto

Descrição gerada automaticamente

Fonte: Canal do YouTube da Neso Academy6

6Disponível em: <https://www.youtube.com/watch?v=2cGo2HdA0dM>.

Porém, os comandos wait() e signal() precisam ser cuidadosamente implementados para evitar que os processos boqueiem uns aos outros.

No exemplo acima, o processo P0 utiliza o semáforo S e o processo P1 utiliza o semáforo Q. No momento em que cada um tenta avançar para a sua próxima linha de comando, são impedidos por não terem permissão de acesso. Isso aconteceu porque apesar de cada processo já ter deixado de utilizar as seções críticas que precisavam utilizar, não liberaram a chave de acesso de imediato.

Apesar do código do programa estar escrito em Linguagem C, há algumas funções que só existem no Linux. O Ubuntu é um sistema operacional escrito em Linux, mas o computador utilizado tem o Windows 10 Single Language instalado. Para contornar essa situação, foi utilizado o software Oracle VM Virtual Box.

Figura 7 – Tela de início do Oracle VM Virtual Box

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte: Autoria própria

O Oracle VM Virtual Box é um software que permite criar máquinas virtuais especializadas para Windows, Mac, Linux e muitos outros sistemas operacionais.

Quando em ação, uma máquina virtual age como um computador dentro do computador do usuário.

Figura 8 - Tela inicial do Ubuntu na máquina virtual criada pelo Oracle VM Virtual Box

Tela de computador

Descrição gerada automaticamente

Fonte: Autoria própria

1. I**MPLEMENTAÇÃO**

Figura 9 - Demonstração do funcionamento do código realizada pelo professor Dalcimar

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Fonte: Autoria própria

Nesse programa, quatro clientes acessam a mesma conta. Cada um pode depositar 10.000 unidades e sacar 5.000. Então, no total, 40.000 unidades serão depositadas e 20.000 serão sacadas. A conta deve estar armazenando 20.000 unidades no final.

Figura 10 - Código do projeto

Texto sobre foto de carro

Descrição gerada automaticamente com confiança média

Fonte: Autoria própria

Existe uma biblioteca pronta para a aplicação de semáforos: semaphore.h. Ela permite utilizar os comandos sem\_wait e sem\_post, que equivalem a wait() e signal(), respectivamente; &lock é a chave de acesso às seções críticas; sim\_init é o comando que habilita o semáforo e estabelece os valores que &lock irá assumir: 0 e 1.

A biblioteca pthread.h permite que vários processos coexistam em um mesmo programa. Como são quatro clientes NUM\_THREADS = 4.

O código do projeto foi armazenado na plataforma GitHub. O link de acesso é: <https://github.com/IntoTheUnkown/Controle-de-acesso-s-se-es-cr-ticas/commit/88d5c291ac5dce16411c9df42e96cd309cd199d4>. Lá estão a versão do programa sem o semáforo e com o semáforo.

1. **RESULTADOS E DISCUSSÃO**

Figura 11 – Execução do programa sem o semáforo

Texto

Descrição gerada automaticamente

Fonte: Autoria própria

Um cliente mais rápido acaba passando na frente de um mais lento, executando as operações de depósito e saque mais vezes do que devia.

Como há clientes que ficaram sem a vez, no momento do saque há uma quantidade menor disponível. No exemplo acima, havia tão pouco dinheiro armazenado que a conta bancária zerou.

Figura 12 - Execução do programa com o semáforo

Texto

Descrição gerada automaticamente

Fonte: Autoria própria

O semáforo garante que cada cliente tenha a sua vez, independente da sua velocidade de realização das transações bancárias. Assim, cada um deposita e saca apenas uma vez.

1. **CONCLUSÃO**

Através desse código simples, pôde-se verificar a importância da utilização de semáforos. Onde a aplicação desse mecanismo é essencial em situações em que é necessário o acesso exclusivo a um recurso compartilhado entre processos. Nesse caso, deve ser possível demonstrar que esse acesso exclusivo ao recurso é garantido, que não há intertravamento entre os processos que competem pelo recurso e que essa competição é justa (ou seja, deve-se garantir que todos os processos tenham a mesma oportunidade de adquirir o recurso Share).  Todos os procedimentos são constituídos de maneira que não haja condições de concorrência, já que o semáforo tem um contador a ser preservado.

**Referências**

CASANOVA, Dalcimar. **Sistemas Operacionais - L11 mecanismos de coordenação (semáforos)**. Disponível em: <https://www.youtube.com/watch?v=Qx49y6pE3NI&t=779s>. Acesso em: 12 de abril de 2022.

Neso Academy. **Semaphores**. Disponível em: <https://www.youtube.com/watch?v=XDIOC2EY5JE&list=PLeL6fbUgUKxz8-ADEazyMmRgzokCSxVpx>. Acesso em: 11 de abril de 2022.

Neso Academy. **Disadvantages of Semaphores**. Disponível em: <https://www.youtube.com/watch?v=2cGo2HdA0dM>. Acesso em: 11 de abril de 2022.