# Threads, concorrência e paralelismo

1st Lucas J. Cunha
*Escola do Mar, Ciência e Tecnologia - EMCT*
*Universidade do Vale do Itajaí - UNIVALI*
Itajaí, Brazil
lucas_cunha@edu.univali.br

1nd Luiz A.Z.Z.M. Pinto
*Escola do Mar, Ciência e Tecnologia - EMCT*
*Universidade do Vale do Itajaí - UNIVALI*
Itajaí, Brazil
luizzimmermann@edu.univali.br

***Index Terms*—threads, concorrencia, paralelismo.**

## I. Introdução

Concorrência, paralelismo, processos, threads, programação síncrona e assíncrona, são assuntos que permeiam o dia a dia dos desenvolvedores. A programação concorrente pode ser implementada em um processador único ou multicore onde processos e threads são criados a partir do compartilhamento destes. Para justificar a necessidade de concorrência é necessário dizer alguns pontos cruciais.

- Possibilidade de sincronização de entrada e saída em algumas situações.
- A algoritmo de controle não tolera espera longas.
- Ser capaz de projetar mais facilmente programas com alto paralelismo intrínseco.
- Um programa concorrente, onde cada tarefa cuida de uma das funcionalidades citadas antes, resulta em um design simples

## II. Problematica

Uma Indústria de Alimentos de Santa Catarina chamada FoodSec S.A. possui a tarefa de escanear alimentos por meio de câmeras e verificar se os mesmos estão corretos. Os alimentos podem passar por uma das três esteiras disponíveis. As três esteiras são controladas por um por um único computador centralizado. Esse computador recebe dados de um sensor em cada uma das esteiras que captura a contagem de itens que são identificados como seguros. A contagem é exibida em um display perto das esteiras (todos os displays são controlados pela mesma função, é apenas uma replicação). Diante disso, a empresa precisa que vocês implementem, por meio de aplicação para distribuição Linux, uma solução que consiga realizar as contagens nas três esteiras e exiba o resultado total (contagem esteira 1 + contagem esteira 2 + contagem esteira 3) em tempo real. A empresa comprou o computador com processador com 4 núcleos, possui uma distribuição Linux e não aceita atualizar sistema ou equipamento, mas aceita carregar novas aplicações. Além disso, os pesos dos itens que passam por cada uma das esteiras são armazenados em um único vetor de dados. A cada 1.500 unidades de produtos, das três esteiras, é necessário atualizar o peso total de itens processados. Sendo assim, a empresa aceita uma pausa na quantidade de itens sendo contados e pesados para realizar a pesagem total, mas ela necessita saber quanto tempo é necessário para isso.

## III. Requisitos

Diante da problemática apresentada, vocês terão que implementar uma aplicação (em nível de MVP) que possa lidar com tal situação usando duas abordagens: Pthreads e OpenMP. Deverá ser feitos um relatório especificando qual a taxa de atualização (capacidade das esteiras em conseguir identificar um novo item por meio do sensor), tempo de processamento da contagem, tempo que consegue atualizar o display. Além disso, como a abordagem utiliza seção crítica, é necessário implementar o uso de mutex na biblioteca Pthread. Como forma de estudar possíveis soluções, avalie o mutex dessa biblioteca usando os protocolos Herança de Prioridade e Teto de Prioridade, os quais podem ser configurados para uso na biblioteca Pthread. Avalie a abordagem single thread e multithread, especifique temporalmente a solução com, por exemplo 1 ou 2 núcleos. Além disso, ambas as bibliotecas oferecem a implementação de Barreira (problema da barreira) e será dado pontuação adicional na prova para quem utilizar de maneira apropriada (+1).

Para fundamentar e realizar os testes de codificação, foram utilizadas as seguintes configurações de hardware:

- Sistema Operacional: Linux Mint 19.3.
- Processador: Intel Core i5-7200U 2.5GHz de 4 núcleos.
- Memória RAM: 8GB.
- Armazenamento: SSD NVME 480GB Kingston.
- GPU: Nvidia Geforce 940MX

## IV. Pthreads

Para padronizar a utilização de threads em diversos sistemas o IEEE estabeleceu o padrão POSIX threads (IEEE_1003.1c), ou Pthreads. Esse padrão define mais de 60 funções para criar e gerenciar threads. Tais funções são definidos na biblioteca pthreads.h. Além disso, a biblioteca define estruturas de dados e atributos para configurar os threads. [1]

### A. Métodos utilizados

O processamento de cada esteira foi feito de maneira separada em cada thread, onde cada sensor de cada esteira é ativado a partir do pressionamento de uma tecla específica no teclado do usuário. Caso o usuário digite 1 será inserindo um produto na esteira A, se digitar '2' será inserido na esteira

B, e '3' na esteira C. Após pressionado alguma tecla, por meio do paralelismo a contagem total de produtos adicionados vai sendo atualizada no código e juntamente em um display mostrado ao usuário. Além da contagem de produtos, o peso também é um atributo a ser atualizado no display. O peso é calculado a partir de uma função que gera valores aleatórios 'rand()', que é então inserido no vetor de pesos. Para facilitar os testes, a cada entrada são adicionados 100 produtos nas esteiras.

### B. Resultados obtidos

Para mensurar os resultados foi necessário utilizar a biblioteca time.h, que pega o uso do clock como referencia para resultar em tempo em segundos.

- Tempo de atualização: 0.000021 segundos.
- Tempo de contagem: 0.010500 segundos.
- Tempo de pesagem: 0.000007 segundos.

## V. OpenMP

OpenMP é uma implementação de multithreading, um método de paralelização no qual o "master thread"(uma série de instruções executadas consecutivamente) forks ("bifurca") um específico número de threads escravos e uma tarefa é dividida entre eles. Os thread são então executados simultaneamente, com ambiente de execução distribuindo as threads para diferentes processadores. [2]

### A. Métodos utilizados

A implementação da biblioteca OpenMP foi mais simplista se comparado a ultima (Pthread). Para obter uma entrada pelo usuário foi necessario o uso de uma função *input_generator()* que gera as entradas ciclicamente. Na função principal *main()* temos a determinação do número de threads que serão utilizados em *#pragma omp parallel shared()* e após isso o início de uma thread em *#pragma omp critical()* que realiza a tarefa de todas as threads em apenas uma utilização. A utilização desta biblioteca não permitiu a repetição da contagem, isso é somente possível se o usuário executar o código novamente.

### B. Resultados obtidos

Para mensurar os resultados foi necessário utilizar a biblioteca time.h, que pega o uso do clock como referencia para resultar em tempo em segundos.

- Tempo de atualização: 0.000038 segundos.
- Tempo de contagem: 0.00450000 segundos.
- Tempo de pesagem: 0.000005 segundos.

## VI. Single Thread

Programas sequenciais são responsaveis por executarem tarefas sozinhas, onde utiliza serviços de sistemas operacionais através de chamadas de sistema. Este tipo de programação se comporta de forma simples

### A. Métodos utilizados

Como dito anteriormente, a implementação de programas que utilizam estrutura sequenciais são simples. O funcionamento é dado por uma função *main()* que possui a verificação sequencial de qual tecla esta sendo pressionada e inclui o produto na lista total de produtos. Enquanto são inserindo os itens são exibidos os valores atualizados de peso e número de produtos no display. Após atingir o limite de produtos, são cronometrados os tempos de pesagem, contagem e atualização do display.

### B. Resultados obtidos

Para mensurar os resultados foi necessário utilizar a biblioteca time.h, que pega o uso do clock como referencia para resultar em tempo em segundos.

- Tempo de atualização: 0.000019 segundos.
- Tempo de contagem: 0.000900 segundos.
- Tempo de pesagem: 0.000006 segundos.

## VII. Conclusão

Em nosso entendimento, entre as duas bibliotecas usadas neste trabalho, a biblioteca OpenMP foi de mais fácil entendimento comparado ao uso da biblioteca Pthread. Porém se comparado as duas bibliotecas utilizadas e seus métodos derivados com o algoritmo sequencial, as duas são mais complexas devido ao uso de paralelização.

Já em relação aos tempo medidos, as duas bibliotecas não possuem uma diferença relevante entre si, mas entre os métodos, o single thread apresenta tempos de execução menores que os métodos paralelizados. Ao contrário do que muitas pessoas pensam, o paralelismo nem sempre apresenta um desempenho melhor que um algoritmo sequencial, isso pode ser devido ao processador presente na máquina onde os programas são executados, que pode não ser adequado para paralelismo, ou possuir um número de núcleos inferior ao número de threads presente no programa.

Os programas desenvolvidos para este trabalho podem ser encontrados no repositório: https://github.com/Lucasgb7/RTS, e a apresentação do mesmo pode ser vista em: https://youtu.be/69DVIsEYhTg.

## References


[1] F. D. GARCIA, "Threads POSIX" Embarcados, 2020. [Online]. Available: https://www.embarcados.com.br/threads-posix/

[2] Anônimo, "OpenMP" Wikipedia, 2020. [Online]. Available: https://pt.wikipedia.org/wiki/OpenMP