operações Críticas

X

Tempo de execução


Professor : José Carlos Althoff

Um problema pode ser resolvido através de diversos algoritmos.

O fato de um algoritmo resolver um dado problema não significa que seja aceitável na prática.

# Exemplo:
# A solução de sistemas lineares.

| ordem | Método de Cramer | Método de Gauss |
|---|---|---|
| 2 | $22\mu s$ | $50\mu s$ |
| 3 | $102\mu s$ | $159\mu s$ |
| 4 | $456\mu s$ | $353\mu s$ |
| 5 | $2.35ms$ | $666\mu s$ |
| 10 | $1.19min$ | $4.95ms$ |
| 20 | 15255 séculos | $38.63ms$ |

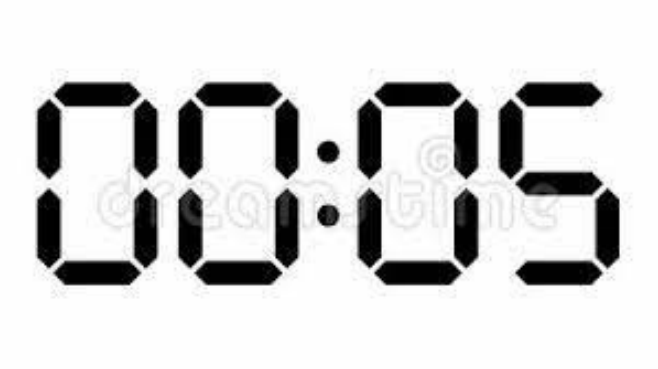
- Na maioria das vezes, a escolha de um algoritmo é feita através de critérios subjetivos como

    1) facilidade de compreensão, codificação e depuração;

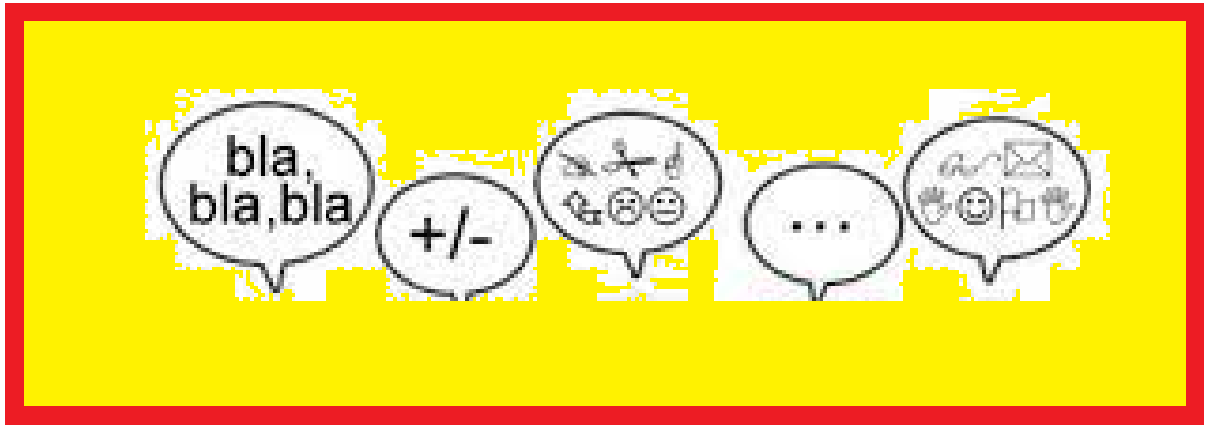
    2) eficiencia na utilização dos recursos do computador e rapidez.

Pergunta:

Será que medir o tempo em (mili-segundos, segundos) é adequado para verificar o desempenho de um algoritmo?



Vamos falar um pouco sobre o assunto:
Tempo



Uma entrada de dados muito grande **pode não** ser muito eficaz utilizando um algoritmo e mais eficaz usando outro algoritmo.

Sabemos que algoritmos diferentes podem executar a mesma tarefa, mas uns são mais eficientes do que outros.


Medir está eficiência em termos de tempo é muito complexo uma vez que temos muitas variáveis envolvidas.

Por exemplo:

- hardware;
- o algoritmo;
- a linguagem;
- entre outros.

É necessário uma medida que indique qual algoritmo é mais eficiente.

Medir o tempo pode não ser adequado.

# Operações Críticas

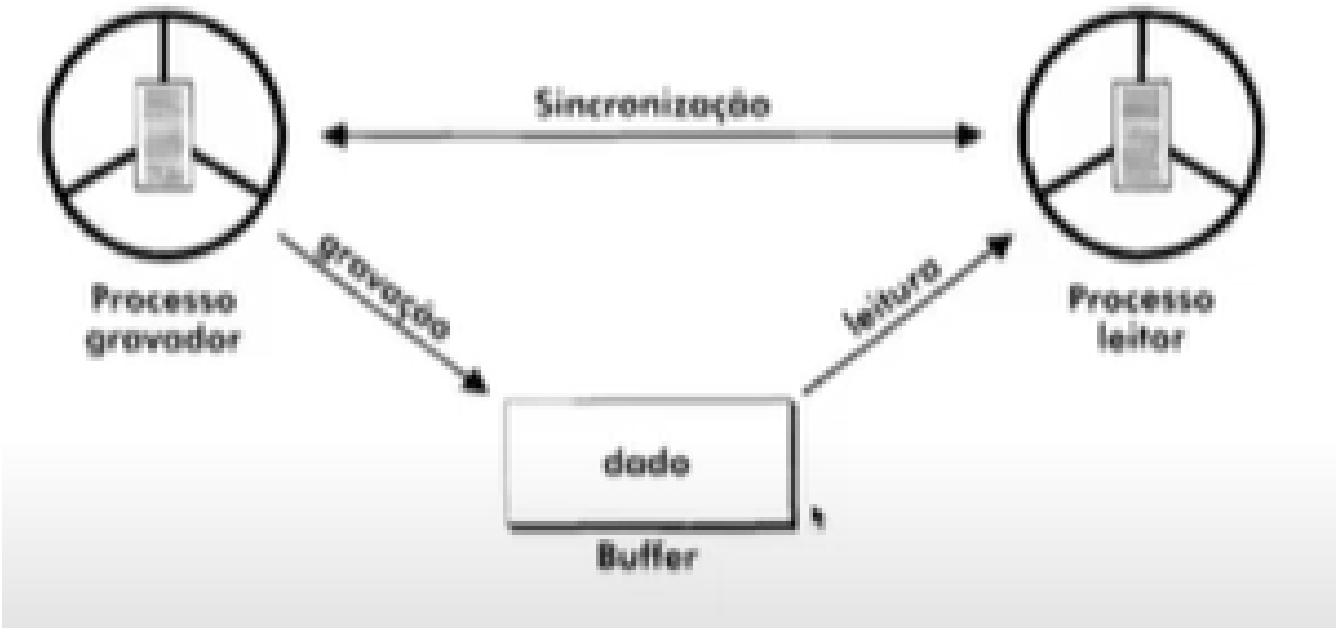
## IPC – Interprocess Communications.

É o fato de eu ter processos no sistema operacional que precisam disputar um determinado recurso compartilhado na CPU.

Mas lembramos que podemos ter sistemas distribuídos e nestes caso precisamos de acesso a rede.

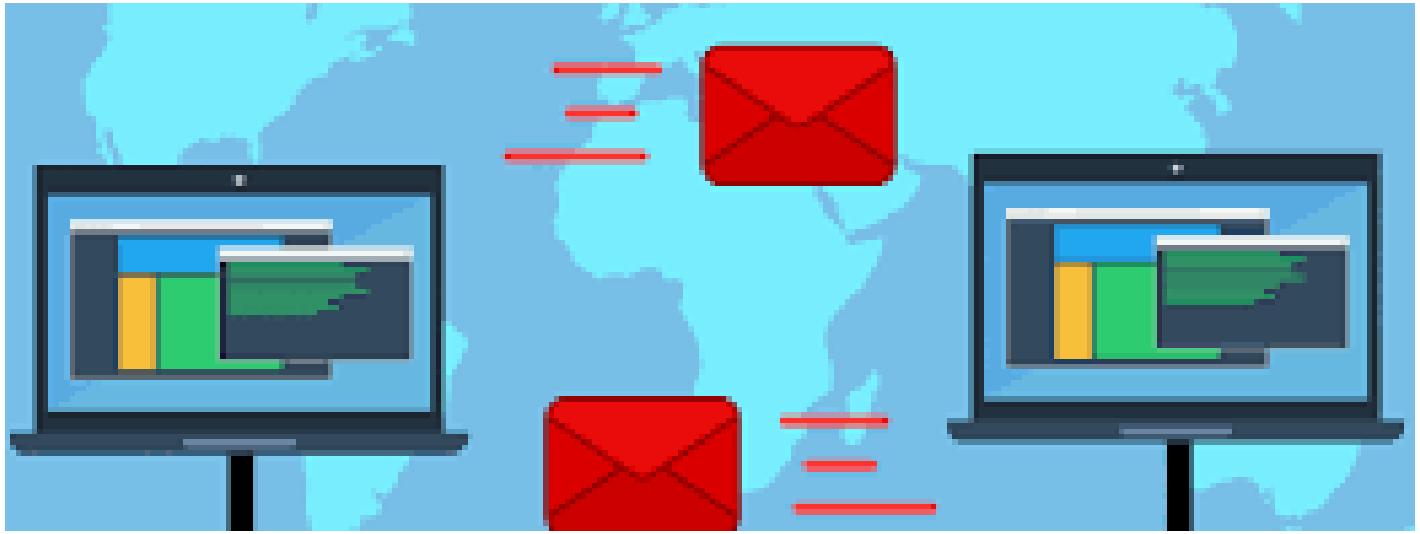
Mas primeiro vamos entender o problema.

Então dado vários processos disputando um recurso é o que chamamos de condição de corrida.



Muitas implementações de sistemas complexos são estruturadas com várias tarefas interdependentes, que cooperam entre si para atingir os objetivos da aplicação, como por exemplo em um navegador Web.

- Para que as várias tarefas que compõem a aplicação possam cooperar, elas precisam comunicar informações umas às outras e coordenar suas atividades, para garantir que os resultados obtidos sejam coerentes.



# O problema da concorrência.


- Quando duas ou mais tarefas acessam simultaneamente um recurso compartilhado, podem ocorrer problemas de consistência dos dados ou do estado do recurso acessado.

**Bom as condições de corrida é justamente o problema.**

- Ou seja, dois ou mais processos disputando um recurso. Por exemplo um arquivo, uma variável de memória, o valor de uma variável de memória.

Em programação, uma região **crítica** - também conhecida por seção **crítica** - é uma área de código de um algoritmo que acessa um recurso compartilhado que não pode ser acedido concorrentemente por mais de uma linha de execução.

**Então Região Crítica** é a área de um código ou recurso compartilhado que depende expressivamente que o acesso à mesma seja realizado de maneira sequencial.

Porém, sabemos da possibilidade de condição de corrida, essa **região** deverá ser tratada de maneira especial de maneira a evitar/tratar condições de corrida entre processos.

Na programação simultânea, os acessos simultâneos a recursos compartilhados podem levar a um comportamento inesperado ou errôneo, portanto, partes do programa onde o recurso compartilhado é acessado precisam ser protegidas de forma a evitar o acesso simultâneo. Não pode ser executado por mais de um processo por vez.

Normalmente, a seção crítica acessa um recurso compartilhado, como uma estrutura de dados, um dispositivo periférico ou uma conexão de rede, que não operaria corretamente no contexto de vários acessos simultâneos.

Por exemplo, se uma variável *x* deve ser lida pelo processo A e o processo B precisa gravar na mesma variável *x* ao mesmo tempo, o processo A pode obter o valor antigo ou o novo valor de *x* .

Em casos como esses, uma seção crítica é importante. No caso acima, se A precisar ler o valor atualizado de *x* , executar o Processo A e o Processo B ao mesmo tempo pode não dar os resultados necessários.

- Para evitar isso, a variável *x* é protegida por uma seção crítica. Primeiro, B obtém o acesso à seção. Assim que B terminar de escrever o valor, A terá acesso à seção crítica e a variável *x* poderá ser lida.

Controlando cuidadosamente quais variáveis são modificadas dentro e fora da seção crítica, o acesso simultâneo à variável compartilhada é evitado.

Uma seção crítica é normalmente usada quando um programa multi-thread deve atualizar várias variáveis relacionadas sem que um thread separado faça alterações conflitantes nesses dados.

**O que são Threads?**

Thread é uma forma de um processo dividir a si mesmo em duas ou mais tarefas, podendo executar elas concorrentemente.

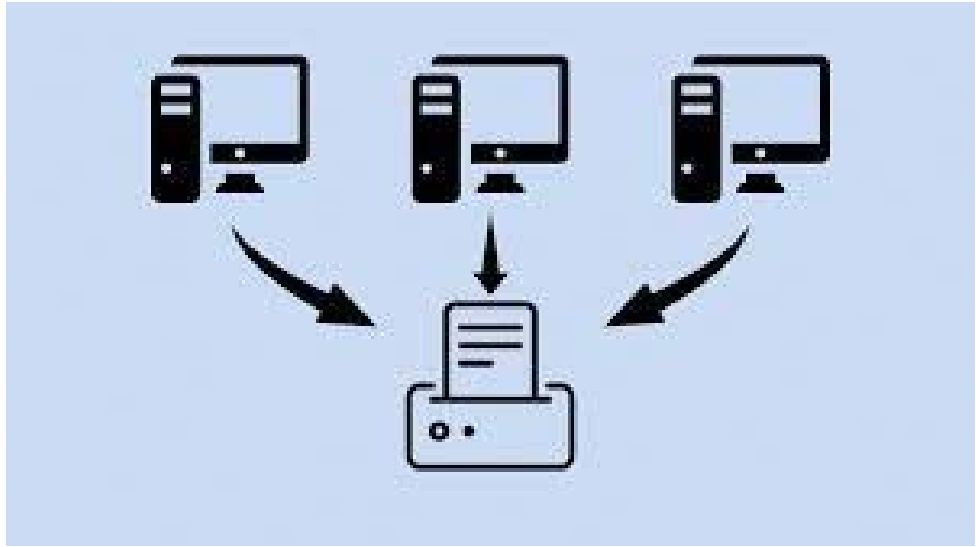
O suporte a threads é oferecido pelos Sistemas Operacionais, ou por bibliotecas de algumas linguagens de programação.

**Relembrando: O que são Aplicações Multithreads?**

Um programa single - thread (thread única) inicia na etapa 1 e continua seqüencialmente (etapa 2, etapa 3, o passo 4) até atingir a etapa final. Aplicações multithread permitem que você execute várias threads ao mesmo tempo, cada uma executando um passo por exemplo.

Exemplo: **Java , já nasceu com o conceito de Threads.**

Em uma situação relacionada, uma seção crítica pode ser usada para garantir que um recurso compartilhado, por exemplo, uma impressora, só possa ser acessado por um processo por vez.



Abaixo apresento um link de um vídeo que apresenta um pouco melhor o problema da medição de tempo de um algoritmo.
Quando tiverem tempo assistam.

https://www.youtube.com/watch?v=1cexU-mQMXU&t=2s