

## Introdução

Quando dois ou mais processos leem ou escrevem algum dado compartilhado e o resultado depende das informações de quem executa e também de quando é executado, isso é chamado de condição de disputa. Esse problema pode ser evitado encontrando um modo de impedir que mais de um processo leia e escreva ao mesmo tempo na memória compartilhada. Desta forma entra a exclusão mútua. Ela assegura que outros processos sejam impedidos de usar uma variável ou um arquivo compartilhado que já esteja em uso por um processo. A seguir serão vistos alguns exemplos disponibilizados pelo professor Maziero para obter exclusão mútua, que podem ou não funcionar:

### Funcionamento do Algoritmo de “Sem coordenação

```
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ gcc -Wall me1-none.c -o me1 -lpthread  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ time ./me1  
Sum should be 10000000 and is 10000000  
  
real    0m0,030s  
user    0m0,019s  
sys     0m0,004s  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$
```

## Funcionamento do Algoritmo de “Ingênuo”

Esta solução consiste em definir uma variável de controle `busy`, que controla a entrada na seção crítica (incremento da variável `sum`).

```
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ time ./me2  
Sum should be 10000000 and is 10000000  
  
real    0m0,868s  
user    0m0,853s  
sys     0m0,000s  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$
```

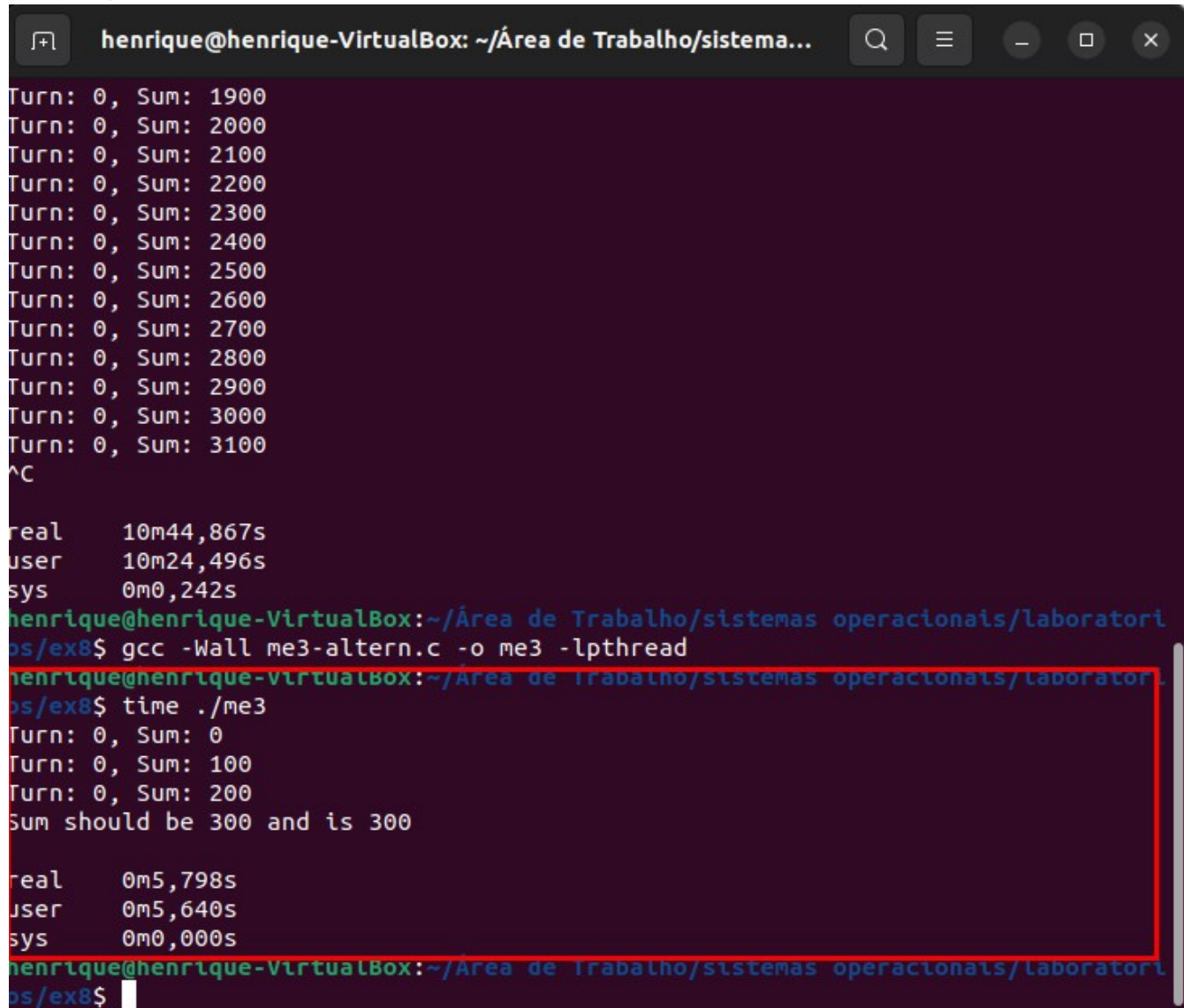
O teste contínuo da variável `busy` (chamado *busy wait*) consome muito processamento. Se houver muitas tarefas tentando entrar em uma seção crítica será gasto muito tempo de processamento.

## Funcionamento do Algoritmo de “Alternância”

Esta solução consiste em definir uma variável `turn` que indica quem pode acessar a seção crítica a cada instante. Ao sair da seção crítica, cada thread incrementa o valor dessa variável, fazendo com que a próxima thread tenha acesso à seção crítica.

Usando os mesmos parametros de Threads e Steps do algoritmo anterior, para calcular sum de 1000000, levaria 27 dias.

Esta solução funciona, mas é muuuuito lenta. Por que?



```
henrique@henrique-VirtualBox: ~/Área de Trabalho/sistema...
Turn: 0, Sum: 1900
Turn: 0, Sum: 2000
Turn: 0, Sum: 2100
Turn: 0, Sum: 2200
Turn: 0, Sum: 2300
Turn: 0, Sum: 2400
Turn: 0, Sum: 2500
Turn: 0, Sum: 2600
Turn: 0, Sum: 2700
Turn: 0, Sum: 2800
Turn: 0, Sum: 2900
Turn: 0, Sum: 3000
Turn: 0, Sum: 3100
^C
real    10m44,867s
user    10m24,496s
sys     0m0,242s
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori
ps/ex8$ gcc -Wall me3-altern.c -o me3 -lpthread
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori
ps/ex8$ time ./me3
Turn: 0, Sum: 0
Turn: 0, Sum: 100
Turn: 0, Sum: 200
Sum should be 300 and is 300
real    0m5,798s
user    0m5,640s
sys     0m0,000s
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori
ps/ex8$
```

Nesse código é definido uma variável `turn` que indica de quem é a vez de entrar na seção crítica. Essa variável deve ser ajustada cada vez que uma tarefa sai da seção crítica, para indicar que a próxima tarefa pode usá-la. Nessa solução cada tarefa aguarda seu turno de usar a seção crítica, numa sequência circular. Isso garante a exclusão mútua entre as tarefas e independe de fatores externos, mas não atende ao critério de independência de outras tarefas, pois caso uma tarefa não deseje usar a seção crítica, todas as tarefas posteriores ficarão impedidas de acessar suas regiões críticas, pois a variável `turn` não será incrementada.

A taxa de acerto desse método é alta, porém é extremamente lento.

## Funcionamento do Algoritmo de “Operações atômicas”

Esta solução consiste em usar operações atômicas, como *Test-and-Set Lock* e similares. No exemplo abaixo é usada uma [operação OR atômica](#).

```
sys      0m0,000s
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori
os/ex8$ gcc -Wall me4-tsl.c -o me4 -lpthread
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori
os/ex8$ time ./me4
Sum should be 100000000 and is 100000000

real    0m1,320s
user    0m1,238s
sys     0m0,004s
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori
os/ex8$
```

Esse algoritmo consiste em testar e atribuir um valor a uma variável de forma atômica (indivisível, sem possibilidade de troca de contexto entre essas duas operações). A execução atômica das operações de teste e atribuição impede a ocorrência de condições de disputa sobre a variável. A instrução *Test-and-Set Lock* trava o dado antes de acessar e o libera assim que possível para que a próxima thread possa usá-lo. Com isso a taxa de acerto é grande, mas aumenta o tempo de execução.

## Funcionamento do Algoritmo de “XCHG”

Esta solução é similar à anterior, mas usa a instrução `XCHG` da plataforma Intel.

```
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ gcc -Wall me5-xchg.c -o me5 -lpthread  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ time ./me5  
Sum should be 10000000 and is 10000000  
  
real    0m1,265s  
user    0m1,201s  
sys     0m0,000s  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$
```

## Funcionamento do Algoritmo de “Com Semáforos”

Esta solução controla a entrada na seção crítica através de um semáforo genérico POSIX.

```
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ gcc -Wall me6-semaphore.c -o me6 -lpthread  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ time ./me6  
Sum should be 10000000 and is 10000000  
  
real    0m3,665s  
user    0m2,376s  
sys     0m1,224s  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$
```

## Funcionamento do Algoritmo de “Mutex”

Esta solução controla a entrada na seção crítica através de um mutex POSIX. .

```
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ gcc -Wall me7-mutex.c -o me7 -lpthread  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$ time ./me7  
Sum should be 10000000 and is 10000000  
  
real    0m0,285s  
user    0m0,256s  
sys     0m0,000s  
henrique@henrique-VirtualBox:~/Área de Trabalho/sistemas operacionais/laboratori  
os/ex8$
```