

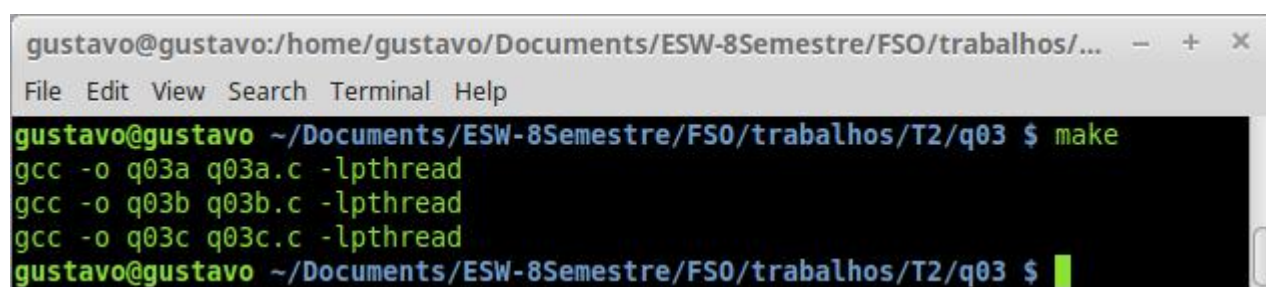
Nome: Gustavo Cavalcante Oliveira  
Matrícula: 130009920

Questão 03 do Trabalho 02 de Fundamentos de Sistemas Operacionais.

## 1 - Instruções

### 1.1 - Compilação

Para gerar os programas executáveis, basta digitar o comando 'make' no diretório da questão 03. Com isso serão criados três programas executáveis, chamados de q03a, q03b e q03c de acordo com a especificação da questão 03.



```
gustavo@gustavo:/home/gustavo/Documents/ESW-8Semestre/FSO/trabalhos/... - + x
File Edit View Search Terminal Help
gustavo@gustavo ~/Documents/ESW-8Semestre/FSO/trabalhos/T2/q03 $ make
gcc -o q03a q03a.c -lpthread
gcc -o q03b q03b.c -lpthread
gcc -o q03c q03c.c -lpthread
gustavo@gustavo ~/Documents/ESW-8Semestre/FSO/trabalhos/T2/q03 $
```

### 1.2 Executável

O formato de execução dos três programas é a mesmo. Basta escrever ./executável no diretório da questão 03, onde executável é o nome de um dos três executáveis gerados.

Será pedido os números de linhas e colunas das matrizes e depois os valores de cada célula das matrizes, sendo que esses valores devem ser informados da seguinte maneira:

- Em cada linha, deverão ser informados os valores de cada coluna pertencentes aquela linha separados por espaços.

Caso o tamanho das duas matrizes informadas seja inválido, a execução do programa será abortada.

No exemplo abaixo fica claro como deverá ser o input da questão. Por fim será impressa a matriz resultante na tela.

```
gustavo@gustavo:/home/gustavo/Documents/ESW-8Semestre/FSO/trabalhos/... - + x
File Edit View Search Terminal Help
gustavo@gustavo ~/Documents/ESW-8Semestre/FSO/trabalhos/T2/q03 $ ./q03a
enter the number of rows and columns for the first matrix.
3 2
enter the number of rows and columns for the second matrix.
2 3
enter the first matrix
1 2
3 4
5 6
enter the second matrix
1 2 3
4 5 6
Result:
 9 12 15
19 26 33
29 40 51
gustavo@gustavo ~/Documents/ESW-8Semestre/FSO/trabalhos/T2/q03 $
```

## 2 - Limitações

### 2.1 Memória

A memória necessária para execução dos programas não é alocada dinamicamente, logo tem-se um limite no tamanho da matriz resultante, que é uma matriz de tamanho 1000x1000.

### 2.2 - Variáveis

As variáveis utilizadas nos programas são do tipo int e possíveis overflows não são tratados, nem mesmo validação de entradas inválidas.

## 3 - Questão de análise

**3.1 “Compare o desempenho em termos de tempo de execução das três aplicações e justifique eventuais diferenças de desempenho obtidas. Em sua resposta, indique qual foi a implementação mais rápida em termos de tempo de execução.”:**

Para análise dos desempenho, foi utilizado como caso de teste uma matriz 90x90, que se encontra dentro do diretório da questão 3 no arquivo chamado in90x90. Para executar o caso de teste, basta escrever no terminal:

```
time cat in90x90 | ./programa > out
```

No qual programa é o nome de um dos três executáveis e out é o arquivo no qual a saída será escrita. O caso de teste foi executado 10 vezes em cada programa, levando em consideração somente o real time, obtivemos os seguintes resultados.

	q03a	q03b	q03c
T1	real0m0.006s	real0m0.147s	real0m0.061s
T2	real0m0.006s	real0m0.151s	real0m0.061s
T3	real0m0.007s	real0m0.143s	real0m0.061s
T4	real0m0.006s	real0m0.146s	real0m0.076s
T5	real0m0.006s	real0m0.139s	real0m0.061s
T6	real0m0.006s	real0m0.142s	real0m0.060s
T7	real0m0.006s	real0m0.145s	real0m0.060s
T8	real0m0.006s	real0m0.140s	real0m0.063s
T9	real0m0.006s	real0m0.143s	real0m0.060s
T10	real0m0.006s	real0m0.139s	real0m0.059s

Tabela 1 - Execução dos programas com uma tarefa simples.

Como esperado pelos resultados do problema 2, a implementação com somente uma thread foi a mais rápida, devido a simplicidade da tarefa ( poucas instruções na função `fill_array(void*)` ) dada a cada thread, a chamada de uma thread é mais custosa que a execução da tarefa.

Adicionando o seguinte trecho de código no final da função `fill_array(void*)` no qual é executada nos códigos fontes dos programas q03a, q03b e q03. Obtivemos os seguintes resultados:

The screenshot shows a terminal window with the following content:

```
gustavo@gustavo:/home/gustavo/Documents/ESW-8Semestre/FSO/trabalhos/... - + x
File Edit View Search Terminal Help

int teste2 = 1000000;
while(teste2--);

38,0-1 40%
```

	q03a	q03b	q03c
T1	real 0m19.016s	real0m6.370s	real0m9.014s
T2	real0m18.870s	real0m6.357s	real0m9.001s
T3	real0m18.877s	real0m6.367s	real0m9.002s
T4	real0m19.066s	real0m6.376s	real0m9.012s
T5	real0m19.266s	real0m6.357s	real0m9.019s
T6	real0m18.666s	real0m6.359s	real0m9.007s
T7	real0m18.797s	real0m6.356s	real0m9.010s
T8	real0m18.506s	real0m6.518s	real0m9.280s

T9	real0m18.926s	real0m6.430s	real0m8.909s
T10	real0m19.090s	real0m6.385s	real0m9.117s

Tabela 2 - Execução dos programas com uma tarefa um pouco mais complexa.

Com essa segunda amostragem de dados, fica claro que a execução em uma única thread não é a melhor solução para esse problema, pois a chamada de cada tarefa é bastante custosa, compensando a solução com múltiplas threads.

Com relação a solução de múltiplas threads, com a execução de tarefas simples, a solução que limita o número de threads ( quatro ) se saiu melhor, já na execução de tarefas um pouco mais complexas, a solução que não limita o número de threads se saiu melhor, provavelmente isso se deve ao fato do escalonador de processos do linux fazer um trabalho melhor do que a minha solução implementada em software para limitar o número de threads ao mesmo tempo.