



FATORIAL USANDO MULTITHREADING

DCC403 -
SISTEMAS OPERACIONAIS I

KELVIN ARAÚJO FERREIRA
NATÁLIA RIBEIRO DE ALMADA

PROFESSOR: DR. HERBERT OLIVEIRA ROCHA

TEMA 9



O projeto a ser desenvolvido, consiste em calcular o fatorial de todos os números até 1.000.000.

Utilize threads para cada faixa de 1000 valores, crie uma thread e dispare o processo para cada uma delas. O fatorial de um número inteiro e positivo “n”, representado por “n!” é obtido a partir da multiplicação de todos os seus antecessores até o número um, cuja expressão genérica é $n! = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \dots 2, 1$.



$n!$

$$0! = 1$$

$$1! = 1$$

$$2! = 2 \cdot 1 = 2$$

$$3! = 3 \cdot 2 \cdot 1 = 6$$

$$N! = N \cdot (N-1) \cdot (N-2) \dots 3 \cdot 2 \cdot 1$$

**MAS O QUE É
FATORIAL?**

Fatorial é o produto dos números inteiros positivos consecutivos de um número natural n , menores ou iguais a n . A notação do fatorial de um número n é: $n!$

DESAFIO

OVERFLOW



UTILIZAÇÃO DE STRINGS

O RESULTADO DE FATORIAIS DE NÚMEROS MAIORES, ULTRAPASSA O LIMITE DE REPRESENTAÇÃO DE INTEIROS. NESTE PONTO, EM SISTEMAS DE 64 BITS O LIMITE DE REPRESENTAÇÃO SERIA O VALOR DE : (2^{64}) QUE É MENOR QUE O FATORIAL DE 21 POR EXEMPLO.

O RESULTADO DE FATORIAIS DE NÚMEROS MAIORES, ULTRAPASSA O LIMITE DE REPRESENTAÇÃO DE INTEIROS. NESTE PONTO, EM SISTEMAS DE 64 BITS O LIMITE DE REPRESENTAÇÃO SERIA O VALOR DE : (2^{64}) QUE É MENOR QUE O FATORIAL DE 21 POR EXEMPLO.

UTILIZANDO THREADS

Utilizamos a biblioteca ***pthread*** para a aplicação de threads, e as funções de ***pthread_mutex*** para sincronizar as threads.

Funções utilizadas:

- `pthread_create()`
- `pthread_join()`
- `pthread_mutex_init()`
- `pthread_mutex_lock()`
- `pthread_mutex_unlock()`

```
C: > Users > NATALIA > Desktop > ufrf 2022.1 > Sistemas Operacionais > C ThreadUmMilhão.c > ..
1  ✓ #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #define maxSize 60000000
6
7  int size;
8  int *buffer;
9
10 pthread_mutex_t mutex;
11 // As variáveis que irão passar na função das threads//
12 ✓ typedef struct {
13     int numero;
14     int thread;
15     int totalThreads;
16 }args;
17
18 // Assinaturas das funções//
19
20 ✓ /* A que vai multiplicar cada numero do fatorial pelo
21    produto, algarismo por algarismo*/
22    void multiply(int fatNum);
23
24    void* threadCalc(void* args);
25
26    // A que vai calcular o fatorial de um numero passado pelo parâmetro
27    void factorial(int numero, pthread_t *thread, int numT);
28
29    // As Implementações
```


UTILIZANDO THREADS

```
Users > NATALIA > Desktop > ufr 2022.1 > Sistemas Operacionais > C ThreadUmMilhão.c > threadCalc(void *)

// As Implementações

//Vai multiplicar os valores de 2 até o valor de numero, pelos algoritmos do buffer
void* threadCalc(void* param){
    args *info = (args*) param;
    // Aqui define o valor inicial do loop a partir da thread
    int start = 1 + (info->thread*1000);
    // Aqui define o valor final do loop a partir da thread
    int end = (info->thread + 1)*1000;
    if(info->numero%1000 != 0 && (info->thread == info->totalThreads - 1))
        end = start - 1 + info->numero%1000;

    printf("inicializando thread %d | calculo de %d a %d\n", info->thread, start, end);

    //É o que impede as outras threads de entrarem na regioao critica
    pthread_mutex_lock(&mutex);
    for(int i = start; i <= end; i++){
        multiply(i);
    }
    // Libera o acesso à regioao crítica
    pthread_mutex_unlock(&mutex);
}
```

- Funcionamento da *threadCalc()*:
- Passagem pelos parâmetros através da *struct*
- Definição do inicio de do fim do *loop*
- Sincronização da threads com mutex após entrar na região crítica



```

51
52 void factorial(int numero, pthread_t *thread, int numT){
53     buffer = (int*)malloc(maxSize*sizeof(int));           // Vai definir o tamanho do buffer
54     size = 1;                                             // Vai inicializar o tamanho do buffer
55     buffer[0] = 1;                                       // Vai inicializar o valor do fatorial com 1
56
57     int i;        //define um contador
58
59     pthread_mutex_init(&mutex, NULL);
60
61     args parametros[numT];
62
63     for(int i = 0; i < numT; i++){
64         printf("Thread %d\n",i);
65         parametros[i].thread = i;
66         parametros[i].numero = numero;
67         parametros[i].totalThreads = numT;
68         pthread_create(&thread[i], NULL, threadCalc,(void*) &parametros[i]);
69     }
70
71     for (int i = 0; i < numT; i++){
72         pthread_join(thread[i], NULL);
73     }
74
75     // Vai exibir o valor final do fatorial na tela
76     printf("O fatorial de %d e igual a :\n", numero);
77     for(i = size - 1; i >= 0; i--){
78         printf("%d", buffer[i]);
79     }
80     printf("\n\n");
81     free(buffer);
82 }

```



UTILIZANDO
THREADS

UTILIZANDO THREADS

```
Users > NATALIA > Desktop > ufr 2022.1 > Sistemas Operacionais > C ThreadUmMilhão.c > threadCalc(void *)

// As Implementações

//Vai multiplicar os valores de 2 até o valor de numero, pelos algoritmos do buffer
void* threadCalc(void* param){
    args *info = (args*) param;
    // Aqui define o valor inicial do loop a partir da thread
    int start = 1 + (info->thread*1000);
    // Aqui define o valor final do loop a partir da thread
    int end = (info->thread + 1)*1000;
    if(info->numero%1000 != 0 && (info->thread == info->totalThreads - 1))
        end = start - 1 + info->numero%1000;

    printf("inicializando thread %d | calculo de %d a %d\n", info->thread, start, end);

    //É o que impede as outras threads de entrarem na regioao critica
    pthread_mutex_lock(&mutex);
    for(int i = start; i <= end; i++){
        multiply(i);
    }
    // Libera o acesso à regioao crítica
    pthread_mutex_unlock(&mutex);
}
```

O que mudou na função *main()*:

- Declaração das *threads*
- O cálculo do numero de *threads* que serão utilizadas



SEM USAR THREADS

- A função Factorial():
- Multiplicação de valores
- *LOOP* do *buffer*.

C SemTHREAD.c X

Códigos > C SemTHREAD.c > multiply(int, int [], int)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define maxSize 100000000
5
6  // Aqui as assinaturas das funções.
7  // Aqui multiplica cada numero de fatorial pelo produto algarismo por algarismo.
8  int multiply(int fatNum, int buffer[], int size);
9  //Aqui calcula o fatorial de um numero passado por parametro.
10 int factorial(int number);
11
12 // Aqui as implementações.
13
14 int factorial(int number){
15     int *buffer = (int*)malloc(maxSize*sizeof(int)); // Aqui define o tamanho do buffer.
16     int size = 1; // Aqui inicializa o tamanho do buffer.
17     buffer[0] = 1; // Aqui inicializa o valor do fatorial com 1.
18
19     int i; //Aqui define um contador.
20
21     // Aqui multiplica os valores de 2 ate number e retorna o novo tamanho do buffer.
22     for(i = 2; i <= number; i++){
23         size = multiply(i, buffer, size);
24     }
25
26     // Aqui ele exibe o valor final do fatorial na tela.
27     for(i = size - 1; i >= 0; i--){
28         printf("%d", buffer[i]);
29     }
30     printf("\n\n"); //Aqui pula linha.
31     free(buffer); //Aqui libera a memória do buffer.
32     return size;
33 }
```

SEM USAR THREADS

- Na função *multiply()* é onde ocorre o cálculo de um produto em forma de numeração decimal

C SemTHREAD.c X

Códigos > C SemTHREAD.c > main()

```
34
35 int multiply(int fatNum, int buffer[], int size){
36     int carry = 0;    // Aqui o número que e carregado para outras casas do numero final.
37
38     // Aqui multiplica o número passado, algoritmo por algoritmo.
39     for(int i = 0; i < size; i++){
40         // Aqui multiplica o algoritmo do buffer[i] pelo valor passado e soma com o carry.
41         int prod = buffer[i]*fatNum + carry;
42
43         // Aqui vai trocar o valor do buffer[i] pelo resto da divisão do produto anterior por 10.
44         buffer[i] = prod%10;
45
46         // Aqui define o valor do carry como o quociente a divisão inteira de prod por 10.
47         carry = prod/10;
48     }
49
50     while(carry){
51         buffer[size] = carry%10;
52         carry /= 10;
53         size++;
54     }
55     return size;
56 }
```

SEM USAR THREADS

C SemTHREAD.c X

Códigos > C SemTHREAD.c > multiply(int, int [], int)

```
57
58  int main(){
59      int fat, size;
60
61      while(1){
62          printf("Fatorial de : ");
63          scanf("%d", &fat);
64          if(fat < 0) break;
65          printf("O fatorial de %d e igual a :\n", fat);
66          size = factorial(fat);
67          printf("O fatorial de %d possui %d Algarismos\n", fat, size);
68      }
69      return 0;
70  }
```

- Na função *main()* é onde ocorre a interação do usuário com o programa

CONSIDERAÇÕES



COM THREADS:

SEM THREADS

(100)!

00:00.65

(100)!

00:01.37

(13000)!

00:10.55

(13000)!

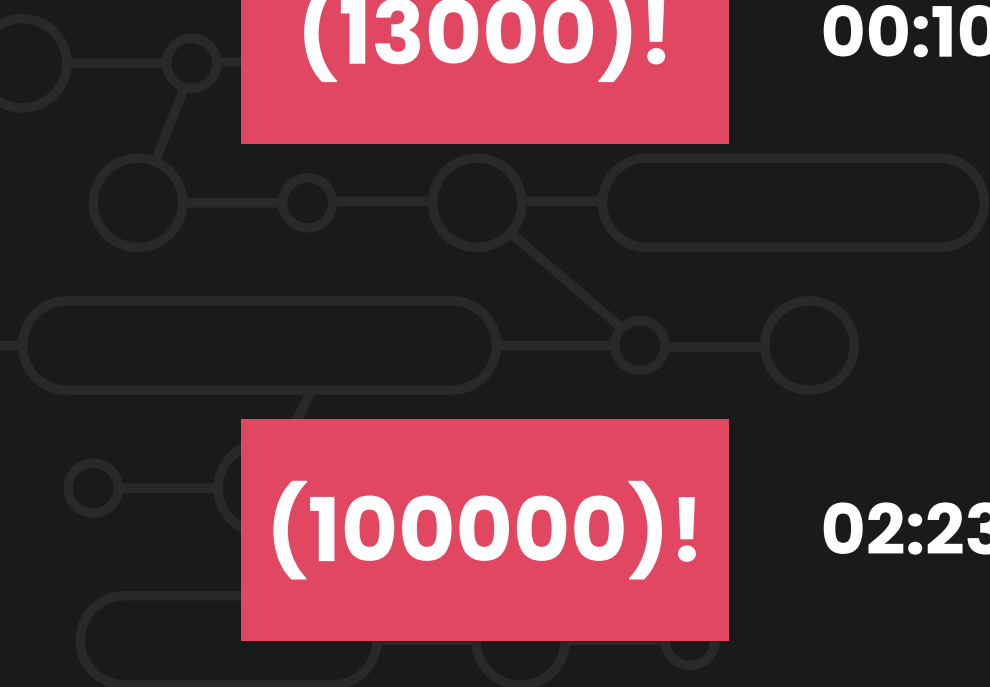
00:16.14

(100000)!

02:23.01

(100000)!

02:44.42



REFERÊNCIAS:

- <https://acervolima.com/fatorial-de-um-grande-numero/>
- <https://dev.to/dandyvica/using-threads-on-rust-part-3-2bpf>
- <https://users.rust-lang.org/t/parallel-product-for-factorial-surprised-by-the-results/30776/14>
- <https://programming-idioms.org/idiom/31/recursive-factorial-simple/450/rust>
- <https://replit.com/languages/c>