
# 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 . (n – 1). (n – 2). (n – 3) ... 2,1.*

# *n!*

$0! = 1$

$1! = 1$

$2! = 2 . 1 = 2$

$3! = 3 . 2 . 1 = 6$

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

## 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

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.

## 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.

# 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 > ufrr 2022.1 > Sistemas Operacionais > C ThreadUmMilhão.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define maxSize 60000000

int size;
int *buffer;

pthread_mutex_t mutex;
 // As variáveis que irão passar na função das threads//
typedef struct {
    int numero;
    int thread;
    int totalThreads;
}args;

// Assinaturas das funções//

/* A que vai multiplicar cada numero do fatorial pelo
produto, algarismo por algarismo*/
void multiply(int fatNum);

void* threadCalc(void* args);

// A que vai calcular o fatorial de um numero passado pelo parâmetro
void factorial(int numero, pthread_t *thread, int numT);

// As Implementações
```

# UTILIZANDO THREADS

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

```c
// As Implementações

//Vai multiplicar os valores de 2 até o valor de numero, pelos algarismos 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 regiao critica
    pthread_mutex_lock(&mutex);
    for(int i = start; i <= end; i++){
        multiply(i);
    }
        // Libera o acesso à regiao 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 > ufrr 2022.1 > Sistemas Operacionais > C ThreadUmMilhão.c > threadCalc(void *)

```c
// As Implementações

//Vai multiplicar os valores de 2 até o valor de numero, pelos algarismos 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 regiao critica
    pthread_mutex_lock(&mutex);
    for(int i = start; i <= end; i++){
        multiply(i);
    }
        // Libera o acesso à regiao 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*.

SemTHREAD.c

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

```c
#include <stdio.h>
#include <stdlib.h>

#define maxSize 100000000

// Aqui as assinaturas das funções.
// Aqui multiplica cada numero de fatorial pelo produto algarismo por algarismo.
int multiply(int fatNum, int buffer[], int size);
//Aqui calcula o fatorial de um numero passado por parametro.
int factorial(int number);

// Aqui as implementações.

int factorial(int number){
    int *buffer = (int*)malloc(maxSize*sizeof(int));        // Aqui define o tamanho do buffer.
    int size = 1;                                           // Aqui inicializa o tamanho do buffer.
    buffer[0] = 1;                                          // Aqui inicializa o valor do fatorial com 1.

    int i;                                                 //Aqui define um contador.

    // Aqui multiplica os valores de 2 ate number e retorna o novo tamanho do buffer.
    for(i = 2; i <= number; i++){
        size = multiply(i, buffer, size);
    }

    // Aqui ele exibe o valor final do fatorial na tela.
    for(i = size - 1; i >= 0; i--){
        printf("%d", buffer[i]);
    }
    printf("\n\n");     //Aqui pula linha.
    free(buffer);       //Aqui libera a memória do buffer.
    return size;
}
```

# SEM USAR THREADS

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

SemTHREAD.c

Códigos > SemTHREAD.c > main()

```c
int multiply(int fatNum, int buffer[], int size){
    int carry = 0;      // Aqui o número que e carregado para outras casas do numero final.

    // Aqui multiplica o número passado, algorismo por algorismo.
    for(int i = 0; i < size; i++){
        // Aqui multiplica o algarismo do buffer[i] pelo valor passado e soma com o carry.
        int prod = buffer[i]*fatNum + carry;

        // Aqui vai trocar o valor do buffer[i] pelo resto da divisáo do produto anterior por 10.
        buffer[i] = prod%10;

        // Aqui define o valor do carry como o quociente a divisão inteira de prod por 10.
        carry = prod/10;
    }

    while(carry){
        buffer[size] = carry%10;
        carry /= 10;
        size++;
    }
    return size;
}
```

# SEM USAR THREADS

SemTHREAD.c

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

```c
int main(){
    int fat, size;

    while(1){
        printf("Fatorial de : ");
        scanf("%d", &fat);
        if(fat < 0) break;
        printf("O fatorial de %d e igual a :\n", fat);
        size = factorial(fat);
        printf("O fatorial de %d possui %d algarismos\n", fat, size);
    }
    return 0;
}
```

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

# CONSIDERAÇÕES

## COM THREADS:

| | |
|---|---|
| (100)! | 00:00.65 |
| (13000)! | 00:10.55 |
| (100000)! | 02:23.01 |

## SEM THREADS

| | |
|---|---|
| (100)! | 00:01.37 |
| (13000)! | 00:16.14 |
| (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