Pthreads parte 2

Nome:Hao Yue Zheng-10408948
Nome:Smuel Zheng-10395781
Nome:Vitor Pasquarelli Cinalli-10401806

Linker de `Git`
https://github.com/mauriciohao/Sistema-operacional.git

Entregue a solução realizada em sala para o problema do cálculo de pi. Para a solução implementada apresente os logs de execução contendo:

* Quantidade de threads;

* Valor de pi calculado pelo algoritmo.

Codigo

- **sem mutex**

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

long n = 1000000;
int amount = 4;

long double sum = 0;

void *Thread_sum(void *rank) {
    long my_rank = (long)rank;
    double factor;
    long long i;
    long long my_n = n / amount;
    long long my_first_i = my_n * my_rank;
    long long my_last_i = my_first_i + my_n;

    if (my_first_i % 2 == 0) {
        factor = 1.0;  // Índice inicial é par
    } else {
        factor = -1.0; // Índice inicial é ímpar
    }
    for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
        sum += factor / (2 * i + 1);
    }
    return NULL;
}

int main() {
    pthread_t *threads;
    threads = (pthread_t *)malloc(amount * sizeof(pthread_t));
    for (long i = 0; i < amount; i++) {
        pthread_create(&threads[i], NULL, Thread_sum, (void *)i);
    }
```

```
    for (long i = 0; i < amount; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Quantidade de threads: %d\n", amount);
    printf("Estimativa de π com %d threads (sem mutex): %Lf\n", amount, sum * 4);
    free(threads);
    return 0;
}
```

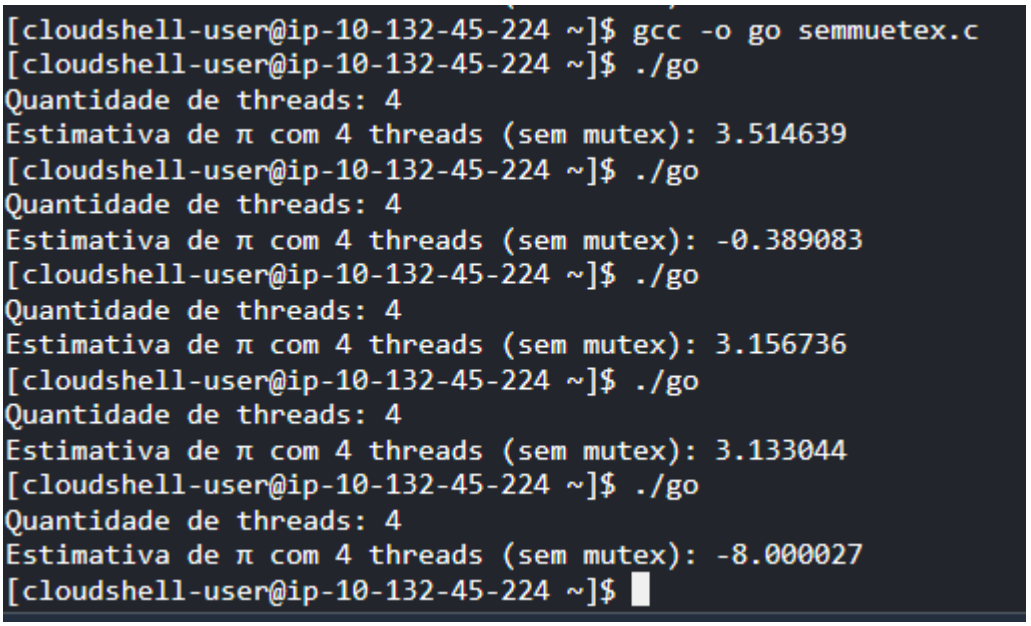
- **com mutex**

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

long n = 1000000;
int amount = 4;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

long double sum = 0;

void *Thread_sum(void *rank) {
    long my_rank = (long)rank;
    double factor;
    long long i;
    long long my_n = n / amount;
    long long my_first_i = my_n * my_rank;
    long long my_last_i = my_first_i + my_n;

    if (my_first_i % 2 == 0) {
        factor = 1.0;  // Índice inicial é par
    } else {
        factor = -1.0; // Índice inicial é ímpar
    }
    for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
        pthread_mutex_lock(&mutex);
        sum += factor / (2 * i + 1);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    pthread_t *threads;
```

```
    threads = (pthread_t *)malloc(amount * sizeof(pthread_t));
    for (long i = 0; i < amount; i++) {
        pthread_create(&threads[i], NULL, Thread_sum, (void *)i);
    }
    for (long i = 0; i < amount; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Quantidade de threads: %d\n", amount);
    printf("Estimativa de π com %d threads (com mutex): %Lf\n", amount, sum * 4);
    free(threads);
    return 0;
}
```

Solução

sem mutex



com mutex

```
[cloudshell-user@ip-10-132-45-224 ~]$ gcc -o run m
mack.pem.txt  muetx.c
[cloudshell-user@ip-10-132-45-224 ~]$ gcc -o run muetx.c
[cloudshell-user@ip-10-132-45-224 ~]$ ./run
Quantidade de threads: 4
Estimativa de π com 4 threads (com mutex): 3.141592
[cloudshell-user@ip-10-132-45-224 ~]$ ./run
Quantidade de threads: 4
Estimativa de π com 4 threads (com mutex): 3.141592
[cloudshell-user@ip-10-132-45-224 ~]$ ./run
Quantidade de threads: 4
Estimativa de π com 4 threads (com mutex): 3.141592
[cloudshell-user@ip-10-132-45-224 ~]$ ./run
Quantidade de threads: 4
Estimativa de π com 4 threads (com mutex): 3.141592
[cloudshell-user@ip-10-132-45-224 ~]$ ./run
Quantidade de threads: 4
Estimativa de π com 4 threads (com mutex): 3.141592
[cloudshell-user@ip-10-132-45-224 ~]$
```

a) Implemente uma solução utilizando Mutex e comparece com a anterior. O que mudou? Por quê?
R:
A principal mudança com o uso do Mutex é evitar condições de corrida ao atualizar a variável compartilhada pi_total. Sem o Mutex, múltiplas threads podem modificar pi_total simultaneamente, o que pode levar a resultados incorretos. Com o Mutex, garantimos que apenas uma thread por vez possa atualizar pi_total, mantendo a corretude do resultado.

b) No final, entregue no README do repositório uma explicação resumida sobre as diferenças entre os valores atingidos. Quais foram as causas das divergências? Por que elas aconteceram?
R:
Cálculo de Pi usando Pthreads

Este repositório contém duas implementações do cálculo de Pi usando o método Gregory-Leibniz. Ambas as versões utilizam múltiplas threads, mas uma usa Mutex para garantir a segurança na atualização de variáveis compartilhadas.

Diferenças nos Resultados

As diferenças entre os resultados das duas versões podem ser atribuídas à presença de condições de corrida na versão sem Mutex. A ausência de Mutex pode resultar em cálculos imprecisos devido a atualizações concorrentes na variável compartilhada.

Conclusão

O uso de Mutex, apesar de adicionar algum overhead, é essencial para garantir a precisão dos resultados em programas multithread.