COMPUTAÇÃO PARALELA - 05D

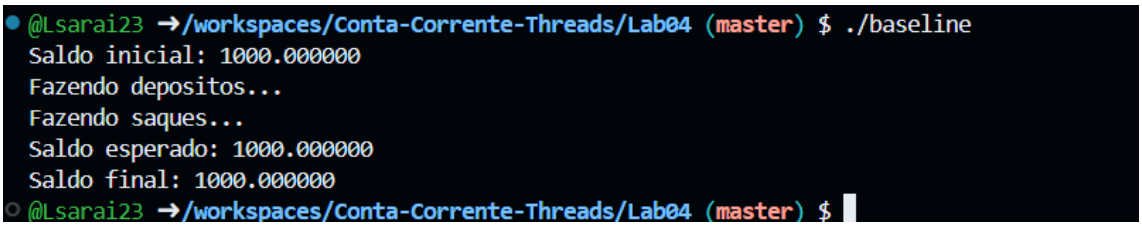
ATIVIDADE: CONTA CORRENTE

INTEGRANTES:

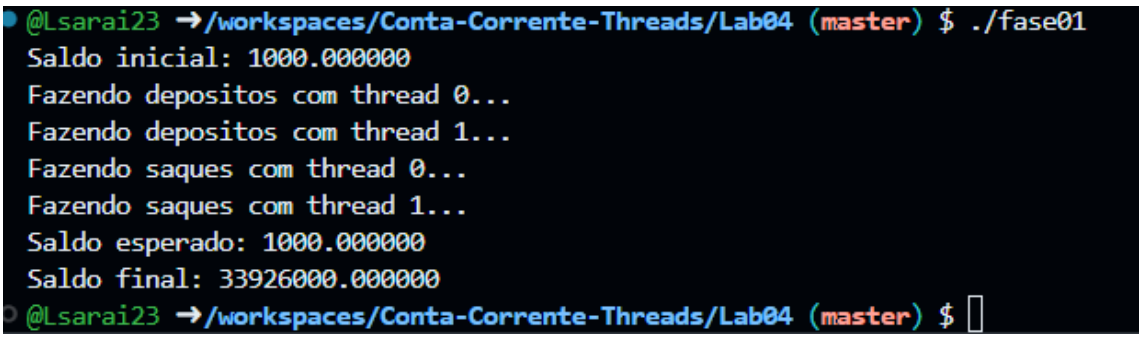
- LUCAS PIRES DE CAMARGO SARAI – 10418013
- LUCAS ZANINI DA SILVA – 10417361

CENÁRIOS

1. **Execução baseline.c (Sem threads)**



2. **Execução fase01.c (Com threads e sem mutex)**



3. **Execução fase02.c (Com threads e com mutex)**

CONCLUSÃO

Diante dos resultados apresentados, podemos chegar as seguintes conclusões sobre os resultados apresentados:

1. No programa baseline.c estamos realizando a as operações de depósitos (depositos()) e saques (saques() , respectivamente. Como a operação está sendo realizada de forma síncrona, ou seja, primeiro a execução da operação de depósitos e depois a operação de saques, conforme o número de repetições da mesma operação especificada pela definição ***NUM_OPERAÇÕES***. Dessa maneira, o resultado é igual ao valor de saldo inicial.

2. Já no programa fase01.c utilizamos do mesmo princípio da realização de múltiplas operações de depósitos e saques, conforme apresentado no tópico anterior. No entanto, para esse código estamos aplicando threads para simular essas operações. Com podemos observar em relação ao valor final da variável saldo apresentou-se bem distinto em relação ao valor inicial, definindo um comportamento não desejado. Esse resultado é devido a ocorrência de uma condição de corrida durante a execução do programa, sendo o acesso de mais de uma thread ao mesmo tempo a uma zona crítica, em que neste caso é a variável global saque.

3. Por fim, no programa fase02.c foi introduzido o conceito de mutex, especificado pela presença da função ***pthread_mutex_lock().*** O mutex é considerado um mecanismo de sincronização para garantir que apenas uma thread execute um seção crítica do código por vez. Dessa maneira, com cada thread acessando a variável global de saque na sua vez respectiva resulta na igualdade do valor da variável saque no final do programa em relação ao início.