

# PROJETO 1 – SISTEMAS OPERACIONAIS

Turma: 04N - Noite

Curso: Ciencia da Computação

Nome: Hao Yue Zheng - RA: 10408948

Nome: Samuel Zheng - RA: 10395781

Nome: Vitor Pasquarelli Cinalli - RA: 10401806

Link no Github: <https://github.com/mauriciohao/Sistema-operacional>

-CODIGO CORRIGIDO:

```
#define _GNU_SOURCE
```

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

```
#include <malloc.h>
```

```
#include <sys/types.h>
```

```
#include <sys/wait.h>
```

```
#include <signal.h>
```

```
#include <sched.h>
```

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

```
#include <pthread.h> // Inclusão da biblioteca pthread
```

```
// 64kB stack
```

```
#define FIBER_STACK 1024*64
```

```
#define NUM_TRANSFERS 20
```

```
struct c {
```

```
    int saldo;
```

```
};
```

```
typedef struct c conta;
```

```
conta from, to;
```

```
// Variável para controlar o acesso concorrente
```

```
pthread_mutex_t transfer_mutex; // Declaração do mutex
```

```
// The child thread will execute this function
```

```
void* transferencia(void *arg) {
```

```
    int i;
```

```
    for(i = 0; i < NUM_TRANSFERS; i++) {
```

```
        // Bloqueia o mutex
```

```
        pthread_mutex_lock(&transfer_mutex);
```

```
        // Tranfere de c1 para c2
```

```
        if(from.saldo >= 10) {
```

```
            from.saldo -= 10;
```

```
            to.saldo += 10;
```

```
        }
```

```
        // Quando c1 zerar, inverte as contas
```

```
        if(from.saldo == 0) {
```

```
            conta tmp = from;
```

```
            from = to;
```

```
            to = tmp;
```

```
        }
```

```
        // Exibe saldos
```

```
        printf("c1: %d\nc2: %d\n", from.saldo, to.saldo);
```

```
    // Libera o mutex  
    pthread_mutex_unlock(&transfer_mutex);  
}  
return NULL;  
}
```

```
int main() {  
    void* stack;  
    pthread_t thread;  
    int i;  
  
    // Inicialização do mutex  
    pthread_mutex_init(&transfer_mutex, NULL);  
  
    stack = malloc(FIBER_STACK);  
    if (stack == NULL) {  
        perror("malloc: could not allocate stack");  
        exit(1);  
    }  
  
    from.saldo = 100;  
    to.saldo = 0;  
    printf("Transferindo 10 para a conta c2\n");  
  
    for (i = 0; i < 2; i++) {  
        pthread_create(&thread, NULL, transferencia, NULL);  
        pthread_join(thread, NULL); // Espera a thread terminar
```

```

}

// Destruição do mutex

pthread_mutex_destroy(&transfer_mutex);

// Free the stack

free(stack);

printf("Transferências concluídas e memória liberada.\n");

return 0;
}

```

#### -TESTES COM CODIGO CORRIGIDO:

```

[ec2-user@ip-172-31-88-88 projeto01]$ ./CodigoCorrigido
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-88-88 projeto01]$ ./CodigoCorrigido
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-88-88 projeto01]$ ./CodigoCorrigido
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-88-88 projeto01]$ ./CodigoCorrigido
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-88-88 projeto01]$ ./CodigoCorrigido
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-88-88 projeto01]$

```

Com a solução implementada, ocorre o acesso ao recurso de maneira organizada, fazendo com que cada thread complete uma operação antes de começar outra, evitando a corrida.

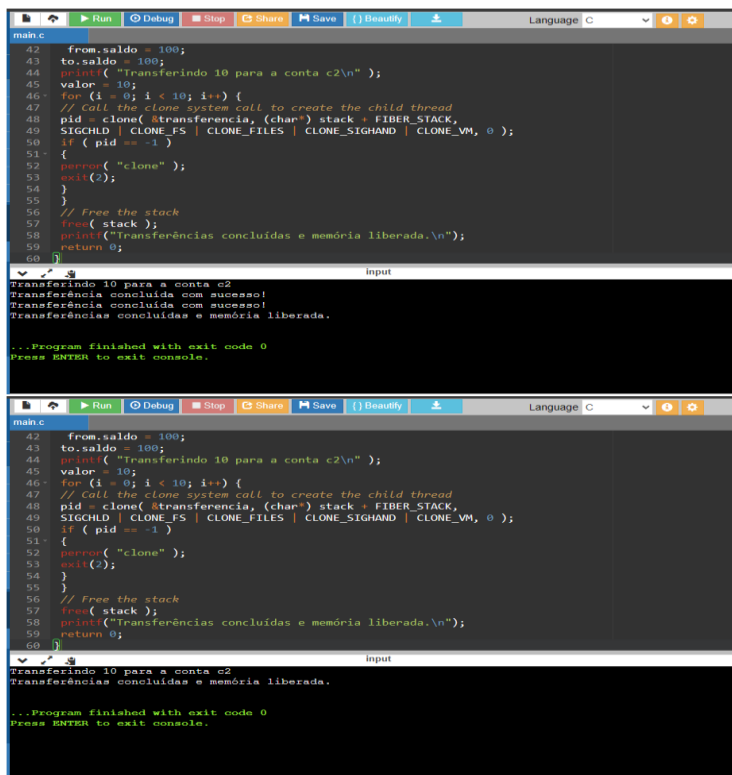
## -TESTES COM CODIGO ORIGINAL:

```
[ec2-user@ip-172-31-48-88 projetos]$ ./codigooriginal
Transferindo 10 para a conta c2
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-48-88 projetos]$ Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
^C
[ec2-user@ip-172-31-48-88 projetos]$ ./codigooriginal
Transferindo 10 para a conta c2
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-48-88 projetos]$ Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
^C
[ec2-user@ip-172-31-48-88 projetos]$ ./codigooriginal
Transferindo 10 para a conta c2
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-48-88 projetos]$ Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
^C
[ec2-user@ip-172-31-48-88 projetos]$ ./codigooriginal
Transferindo 10 para a conta c2
Transferências concluídas e memória liberada.
[ec2-user@ip-172-31-48-88 projetos]$ Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
^C
[ec2-user@ip-172-31-48-88 projetos]$ ]]
```

Durante a execução, os resultados apresentaram constância, também pode-se notar que o terminal foi chamado mais de uma vez a cada execução, indicando ser uma forma deste sistema operacional em lidar com o código.

Como citado no documento README, após a transferência não houve a liberação de memória obrigando o código a ser encerrado de maneira manual.

Por curiosidade, testei o código em outros compiladores e obtive diferentes comportamentos/resultados.

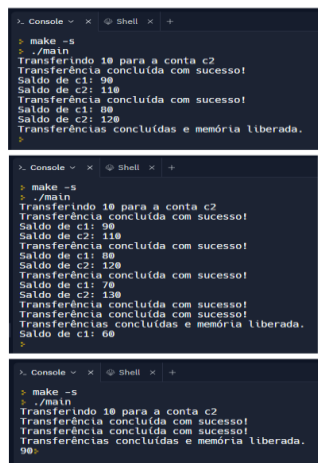


```
main.c
42 from.saldo = 100;
43 to.saldo = 100;
44 printf( "Transferindo 10 para a conta c2\n" );
45 valor = 10;
46 for ( i = 0; i < 10; i++ ) {
47 // Call the clone system call to create the child thread
48 pid = clone( &transferencia, (char*) stack + FIBER_STACK,
49 SIGCHLD | CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0 );
50 if ( pid == -1 )
51 {
52 perror( "clone" );
53 exit(2);
54 }
55 }
56 // Free the stack
57 free( stack );
58 printf( "Transferências concluídas e memória liberada.\n" );
59 return 0;
60 }
```

Input

```
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Transferência concluída com sucesso!
Transferências concluídas e memória liberada.

...Program finished with exit code 0
Press ENTER to exit console.
```



```
> make -s
> ./main
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 120
Transferências concluídas e memória liberada.
>
```

```
> make -s
> ./main
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Saldo de c1: 90
Saldo de c2: 110
Transferência concluída com sucesso!
Saldo de c1: 80
Saldo de c2: 120
Transferência concluída com sucesso!
Saldo de c1: 70
Saldo de c2: 130
Transferência concluída com sucesso!
Transferência concluída com sucesso!
Transferências concluídas e memória liberada.
Saldo de c1: 60
>
```

```
> make -s
> ./main
Transferindo 10 para a conta c2
Transferência concluída com sucesso!
Transferência concluída com sucesso!
Transferências concluídas e memória liberada.
90>
```

Por mera curiosidade, nestes outros compiladores o código original apresentou divergências nas saídas dos resultados.