UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE

Exercício 4 - EMB 5632 - Sistemas Operacionais

Lucas Gabriel Bernardino - 22101935

1 - Introdução

Esse relatório busca propor uma explicação do funcionamento do meu código para o quarto exercício sistemas operacionais. A avaliação proposta exigia que o aluno implementasse uma lista com operações básicas - inicializar, inserir no início e fim e remover no início e fim - porém utilizando *mutex* para proteger as regiões críticas da lista, protegendo as *threads* criadas.

2 - Desenvolvimento

2.1 - Explicando as *structs* utilizadas

Para a implementação da lista, foram definidas três *structs*.

Figura 1 - Definição das três *structs*.

```
3   typedef struct list_t {
4     double value;
5     struct list_t *next;
6   } list;
7
8   typedef struct head_t {
9     size_t size;
10    list *list;
11  } head;
12
13  typedef struct my_list_t {
14    head* header;
15    double value;
16    int function_type;
17  } my_list;
```

Fonte: Elaborado pelo autor (2024).

A *struct* representa o nó da lista encadeada, contendo o seu valor e um ponteiro para o próximo nó, A *head* é a cabeça da lista, que contém o tamanho atual e o ponteiro para o início dela. Já a *my_list* será utilizada para ser usada como parâmetro nas *threads*, contendo assim um ponteiro para a cabeça da lista, um valor que será utilizado nas funções de inserção e o tipo de função que a *thread* deve performar.

2.2 - Explicando as funções utilizadas

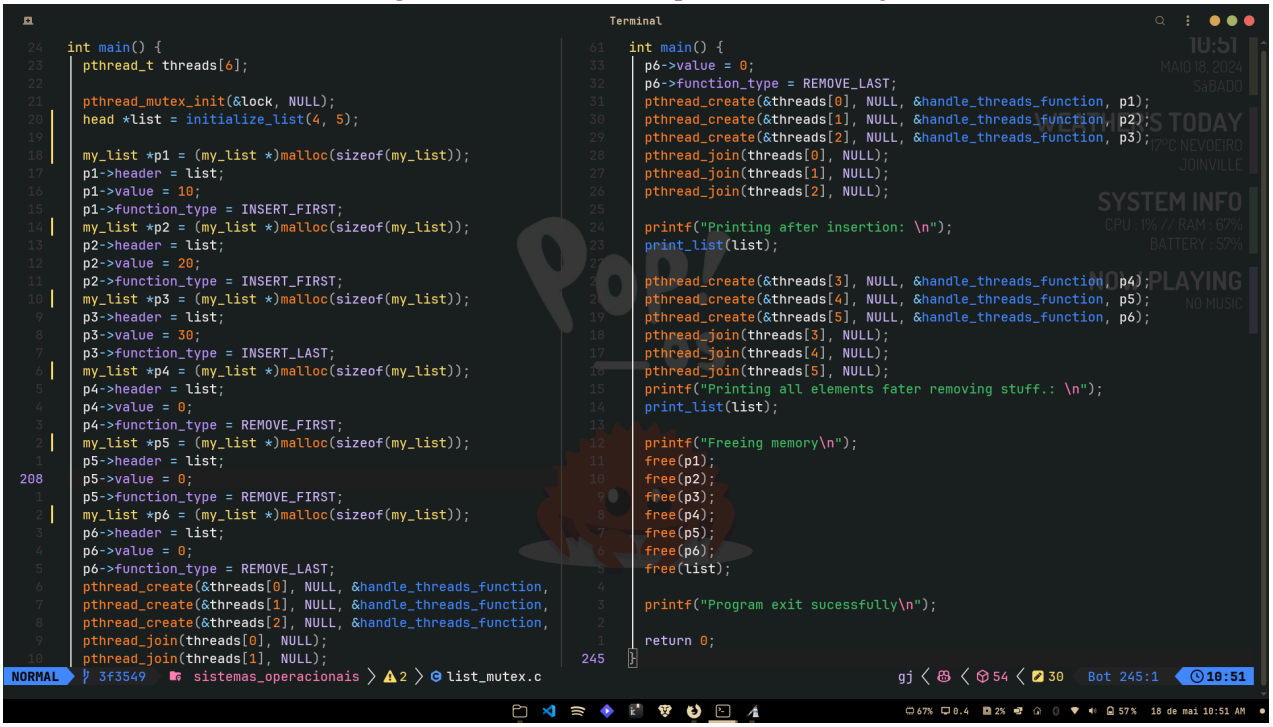
O código possui as quatro operações básicas da lista, e também algumas funções auxiliares. As quatro operações das lista recebem *void* arg* como argumento, o qual será convertido para o tipo *my_list,* explicado anteriormente, para realizar as devidas operações. A função que inicializa a lista recebe o tamanho inicial da lista e um valor para preenchê-la com esse tamanho. Resolvi implementar a função *insert_empty_list* para poupar linhas de código em casos onde será preciso inserir um dado em uma lista vazia. A função *print_list* ajuda a visualizar o estado atual da lista. Por fim, a função *handle_threads_function* é a função que cada thread chamará. Ela recebe como parâmetro *void* arg* que será convertido para o tipo *my_list* para que se possa verificar qual função a thread precisa chamar, verificando dessa maneira o valor guardado no campo *function_type* dessa *struct*.

2.3 - Explicando o funcionamento das *threads* e *mutex*

Para proteger a lista, foi utilizado mutex. Eles são utilizados no começo de cada operação de inserção e de remoção, bem como no final dessas funções. Assim, no começo utiliza-se o *pthread_mutex_lock(&lock),* passando o mutex definido globalmente, para que somente essa thread possa realizar essa operação e as outras esperem-a até que ela acabe, operação que é realizada ao final pela função *pthread_mutex_unlock(&lock),* fazendo com que outras threads possam performar operações na lista.

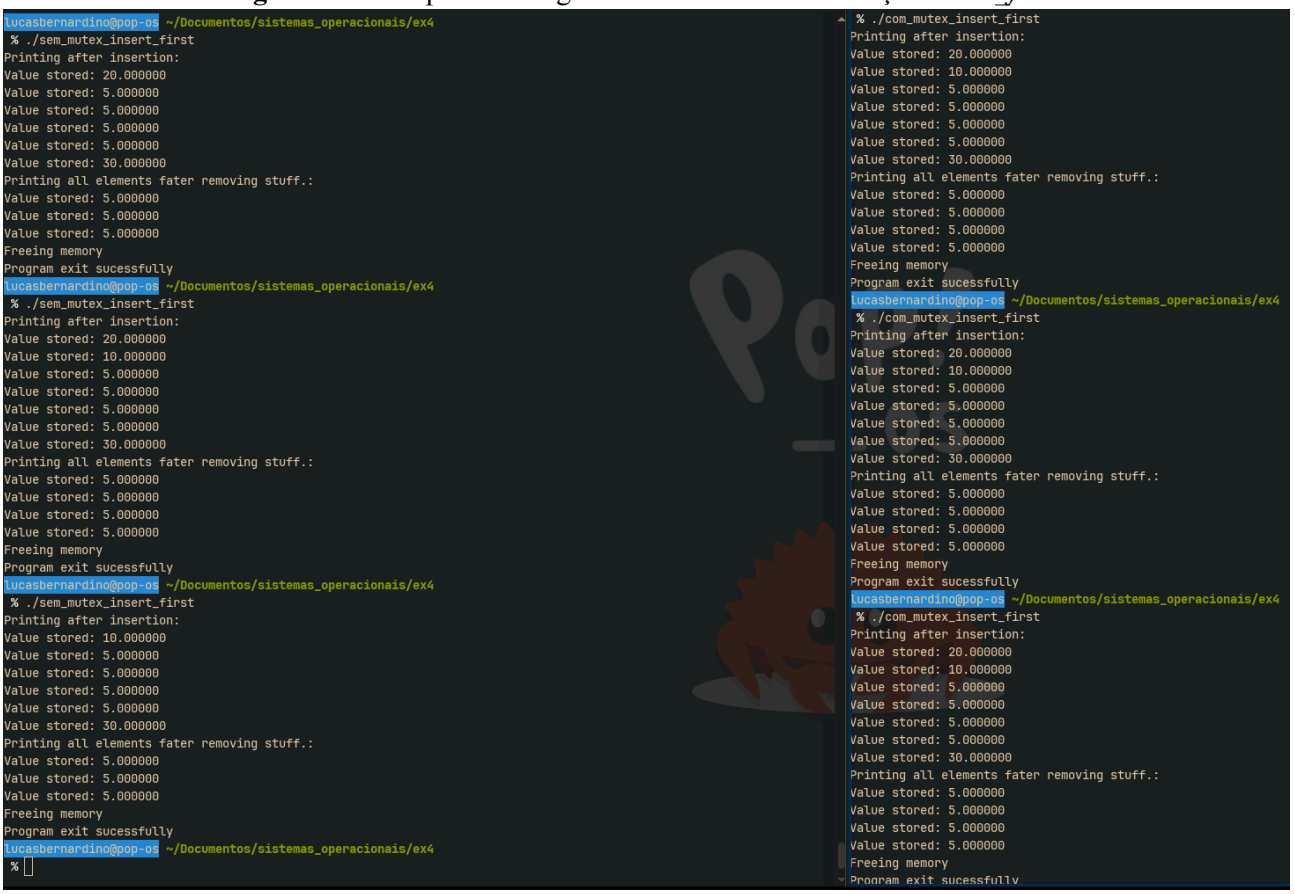
Caso as *threads* chamasse a *handle_threads_function* para realizar as operações na lista e não existisse o *mutex* para garantir a segurança delas, o código estaria repleto de condições de corrida. Elas iriam acontecer devido ao fato de que múltiplas *threads* estariam tentando acessar a lista e realizando operações simultaneamente na mesma área de memória com nenhuma garantia de que uma *thread* esperaria a outra terminar de performar essas operações críticas na memória. Um exemplo disso seria retirar o *mutex* apenas da função que insere no ínicio, fato que já causaria condições de corrida e a perda de dados.

Figura 3 - *Main* utilizada para testar o código



Fonte: Elaborado pelo autor (2024).

Figura 3 - Exemplo de código sem e com mutex na função *insert_first*

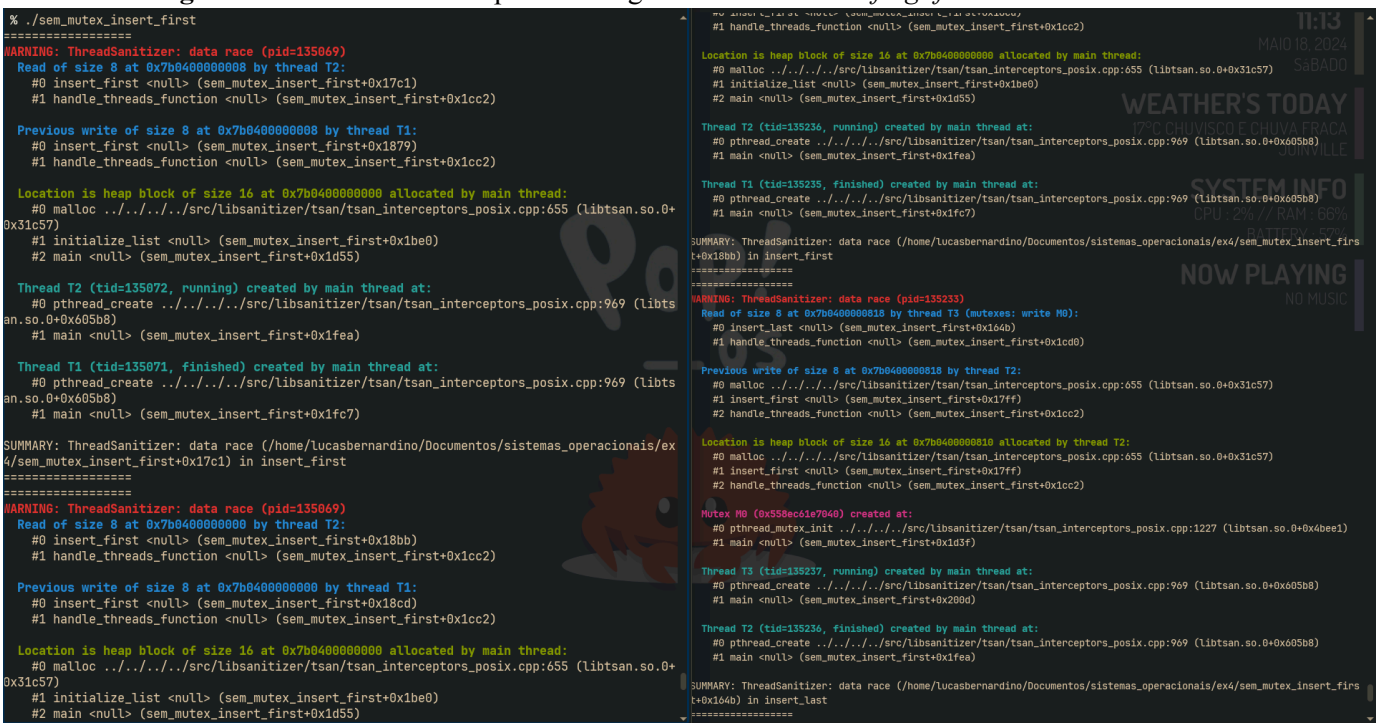


Fonte: Elaborado pelo autor (2024).

A figura 3 demonstra esse exemplo. Para testar o código, foi utilizado a *main* presente na figura 2. No terminal esquerdo, podemos observar a inconsistência da lista no que se refere aos dados que foram inseridos no início. O código foi executado três vezes e nessas três vezes houve diferenças nos elementos da lista. Embora a única diferença entre eles seja a remoção do mutex na função que insere no ínicio, isso já foi o suficiente para causar condições de corrida e deixar a lista inconsistente.

Por fim, a figura 4 representa o que acontece ao compilar e executar o código com a *flag -fsanitize=thread.* Quando o código está com o *mutex* não há nenhum problema e ao compilar com essa *flag* nenhum aviso aparece, enquanto o código sem *mutex* apresenta três avisos, devido a ausência de *mutex.*

Figura 4 - Resultado de compilar o código sem *mutex* com a *flag -fsanitize=thread*



Fonte: Elaborado pelo autor (2024).