



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.



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE

Exercício 4 - EMB 5632 - Sistemas Operacionais

2.2 - Explicando as funções utilizadas

O código possui as quatro operações básicas da lista, e também algumas funções auxiliares. As quatro operações da 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 realizar 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 realizar essas operações críticas na memória. Um exemplo disso seria retirar o *mutex* apenas da função que insere no início, fato que já causaria condições de corrida e a perda de dados.

Figura 3 - Main utilizada para testar o código

```
24 int main() {
25     pthread_t threads[6];
26
27     pthread_mutex_init(&lock, NULL);
28     head = initialize_list(4, 5);
29
30     my_list *p1 = (my_list *)malloc(sizeof(my_list));
31     p1->header = list;
32     p1->value = 10;
33     p1->function_type = INSERT_FIRST;
34     my_list *p2 = (my_list *)malloc(sizeof(my_list));
35     p2->header = list;
36     p2->value = 20;
37     p2->function_type = INSERT_FIRST;
38     my_list *p3 = (my_list *)malloc(sizeof(my_list));
39     p3->header = list;
40     p3->value = 30;
41     p3->function_type = INSERT_LAST;
42     my_list *p4 = (my_list *)malloc(sizeof(my_list));
43     p4->header = list;
44     p4->value = 0;
45     p4->function_type = REMOVE_FIRST;
46     my_list *p5 = (my_list *)malloc(sizeof(my_list));
47     p5->header = list;
48     p5->value = 0;
49     p5->function_type = REMOVE_FIRST;
50     my_list *p6 = (my_list *)malloc(sizeof(my_list));
51     p6->header = list;
52     p6->value = 0;
53     p6->function_type = REMOVE_LAST;
54     pthread_create(&threads[0], NULL, &handle_threads_function, p1);
55     pthread_create(&threads[1], NULL, &handle_threads_function, p2);
56     pthread_create(&threads[2], NULL, &handle_threads_function, p3);
57     pthread_join(threads[0], NULL);
58     pthread_join(threads[1], NULL);
59     pthread_join(threads[2], NULL);
60
61     printf("Printing after insertion: \n");
62     print_list(list);
63
64     pthread_create(&threads[3], NULL, &handle_threads_function, p4);
65     pthread_create(&threads[4], NULL, &handle_threads_function, p5);
66     pthread_create(&threads[5], NULL, &handle_threads_function, p6);
67     pthread_join(threads[3], NULL);
68     pthread_join(threads[4], NULL);
69     pthread_join(threads[5], NULL);
70     printf("Printing all elements after removing stuff: \n");
71     print_list(list);
72
73     printf("Freeing memory\n");
74     free(p1);
75     free(p2);
76     free(p3);
77     free(p4);
78     free(p5);
79     free(p6);
80     free(list);
81
82     printf("Program exit successfully\n");
83
84     return 0;
85 }
```



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE

Exercício 4 - EMB 5632 - Sistemas Operacionais

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

```
lucasbernardino@pop-os: ~/Documentos/sistemas_operacionais/ex4
% ./sem_mutex_insert_first
Printing after insertion:
Value stored: 20.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 30.000000
Printing all elements fater removing stuff.:
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Freeing memory
Program exit sucessfully

lucasbernardino@pop-os: ~/Documentos/sistemas_operacionais/ex4
% ./sem_mutex_insert_first
Printing after insertion:
Value stored: 20.000000
Value stored: 10.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 30.000000
Printing all elements fater removing stuff.:
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Freeing memory
Program exit sucessfully

lucasbernardino@pop-os: ~/Documentos/sistemas_operacionais/ex4
% ./sem_mutex_insert_first
Printing after insertion:
Value stored: 10.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 30.000000
Printing all elements fater removing stuff.:
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Freeing memory
Program exit sucessfully

lucasbernardino@pop-os: ~/Documentos/sistemas_operacionais/ex4
%

lucasbernardino@pop-os: ~/Documentos/sistemas_operacionais/ex4
% ./com_mutex_insert_first
Printing after insertion:
Value stored: 20.000000
Value stored: 10.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 30.000000
Printing all elements fater removing stuff.:
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Freeing memory
Program exit sucessfully

lucasbernardino@pop-os: ~/Documentos/sistemas_operacionais/ex4
% ./com_mutex_insert_first
Printing after insertion:
Value stored: 20.000000
Value stored: 10.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 30.000000
Printing all elements fater removing stuff.:
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Value stored: 5.000000
Freeing memory
Program exit sucessfully
```

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 início, 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*.



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE

Exercício 4 - EMB 5632 - Sistemas Operacionais

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

```
% ./sem_mutex_insert_first
=====
WARNING: ThreadSanitizer: data race (pid=135069)
  Read of size 8 at 0x7b0400000008 by thread T2:
    #0 insert_first <null> (sem_mutex_insert_first+0x17c1)
    #1 handle_threads_function <null> (sem_mutex_insert_first+0x1cc2)

  Previous write of size 8 at 0x7b0400000008 by thread T1:
    #0 insert_first <null> (sem_mutex_insert_first+0x1799)
    #1 handle_threads_function <null> (sem_mutex_insert_first+0x1cc2)

  Location is heap block of size 16 at 0x7b0400000008 allocated by main thread:
    #0 malloc ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:655 (libtsan.so.0+0x31c57)
    #1 initialize_list <null> (sem_mutex_insert_first+0x1be0)
    #2 main <null> (sem_mutex_insert_first+0x1d55)

  Thread T2 (tid=135072, running) created by main thread at:
    #0 pthread_create ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x605b8)
    #1 main <null> (sem_mutex_insert_first+0x1fea)

  Thread T1 (tid=135071, finished) created by main thread at:
    #0 pthread_create ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x605b8)
    #1 main <null> (sem_mutex_insert_first+0x1fc7)

SUMMARY: ThreadSanitizer: data race (/home/lucasbernardino/Documentos/sistemas_operacionais/ex4/sem_mutex_insert_first+0x17c1) in insert_first
=====
WARNING: ThreadSanitizer: data race (pid=135069)
  Read of size 8 at 0x7b0400000008 by thread T2:
    #0 insert_first <null> (sem_mutex_insert_first+0x18bb)
    #1 handle_threads_function <null> (sem_mutex_insert_first+0x1cc2)

  Previous write of size 8 at 0x7b0400000008 by thread T1:
    #0 insert_first <null> (sem_mutex_insert_first+0x18cd)
    #1 handle_threads_function <null> (sem_mutex_insert_first+0x1cc2)

  Location is heap block of size 16 at 0x7b0400000008 allocated by main thread:
    #0 malloc ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:655 (libtsan.so.0+0x31c57)
    #1 initialize_list <null> (sem_mutex_insert_first+0x1be0)
    #2 main <null> (sem_mutex_insert_first+0x1d55)

  Thread T2 (tid=135236, running) created by main thread at:
    #0 pthread_create ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x605b8)
    #1 main <null> (sem_mutex_insert_first+0x1fc7)

  Thread T1 (tid=135235, finished) created by main thread at:
    #0 pthread_create ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x605b8)
    #1 main <null> (sem_mutex_insert_first+0x1fc7)

SUMMARY: ThreadSanitizer: data race (/home/lucasbernardino/Documentos/sistemas_operacionais/ex4/sem_mutex_insert_first+0x18bb) in insert_first
=====
WARNING: ThreadSanitizer: data race (pid=135233)
  Read of size 8 at 0x7b0400000018 by thread T3 (mutexes: write M0):
    #0 insert_last <null> (sem_mutex_insert_first+0x164b)
    #1 handle_threads_function <null> (sem_mutex_insert_first+0x1cc0)

  Previous write of size 8 at 0x7b0400000018 by thread T2:
    #0 malloc ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:655 (libtsan.so.0+0x31c57)
    #1 insert_first <null> (sem_mutex_insert_first+0x17ff)
    #2 handle_threads_function <null> (sem_mutex_insert_first+0x1cc2)

  Location is heap block of size 16 at 0x7b0400000018 allocated by thread T2:
    #0 malloc ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:655 (libtsan.so.0+0x31c57)
    #1 insert_first <null> (sem_mutex_insert_first+0x17ff)
    #2 handle_threads_function <null> (sem_mutex_insert_first+0x1cc2)

  Mutex M0 (0x558ec61e7040) created at:
    #0 pthread_mutex_init ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:1227 (libtsan.so.0+0x4be1)
    #1 main <null> (sem_mutex_insert_first+0x1d3f)

  Thread T3 (tid=135237, running) created by main thread at:
    #0 pthread_create ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x605b8)
    #1 main <null> (sem_mutex_insert_first+0x200d)

  Thread T2 (tid=135236, finished) created by main thread at:
    #0 pthread_create ../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x605b8)
    #1 main <null> (sem_mutex_insert_first+0x1fea)

SUMMARY: ThreadSanitizer: data race (/home/lucasbernardino/Documentos/sistemas_operacionais/ex4/sem_mutex_insert_first+0x164b) in insert_last
=====
```

Fonte: Elaborado pelo autor (2024).