Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

Лабораторная работа № 5

Тема: Динамические библиотеки

Студент: Туманов Георгий
Группа: 80-201
Преподаватель: Соколов А.А.
Оценка:
Подпись:

Москва, 2019

1. Постановка задачи

Требуется создать динамическую библиотеку, которая реализует определенный функционал. Далее использовать данную библиотеку 2-мя способами:

1. Во время компиляции (на этапе «линковки»/linking)
2. Во время исполнения программы, подгрузив библиотеку в память с помощью системных вызовов

В конечном итоге, программа должна состоять из следующих частей:

- Динамическая библиотека, реализующая заданных вариантом интерфейс;
- Тестовая программа, которая используют библиотеку, используя знания полученные на этапе компиляции;
- Тестовая программа, которая использует библиотеку, используя только местоположение динамической библиотеки и ее интерфейс.

Провести анализ между обоими типами использования библиотеки.

Вариант 8:

Структура данных, с которой должна обеспечивать работу библиотека:

1. Работа со списком

Тип данных, используемый структурой:

2. Вещественный 64-битный

2. Описание программы

Файл spisok.c реализует работу со списком для вещественных чисел. Он компилируется в динамическую библиотеку libmyspisok.so.1.0. Программы main1.c и main2.c реализуют простое меню для работы со списком. Первая программа подключается к библиотеке на этапе компиляции и требует ввода строки “export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.” перед запуском, чтобы программа видела библиотеку в папке, в которой она находится. Вторая

программа подключает библиотеку во время работы и выгружает её перед завершением. Компилируется программа двумя строками:
make lib -- скомпилировать библиотеку
make -- скомпилировать программы

3. Набор testcases

№	Описание	Ввод
1	Тест первой программы	3 0 0.1 3 1 0.3 3 1 0.2 2 4 1 2 0
2	Такой же тест для второй программы	---//---

4. Результаты выполнения тестов.

test 1:
1 - show menu
2 - show list contents
3 - insert element
4 - erase element
5 - clear list
6 - set value
0 - exit
> 3
Position: 0
Value: 0.1
> 3
Position: 1

Value: 0.3
> 3
Position: 1
Value: 0.2
> 2
0.100000 0.200000 0.300000 (size: 3)
> 4
Position: 1
> 2
0.100000 0.300000 (size: 2)
> 0

test 2:
1 - show menu
2 - show list contents
3 - insert element
4 - erase element
5 - clear list
6 - set value
0 - exit
> 3
Position: 0
Value: 0.1
> 3
Position: 1
Value: 0.3
> 3
Position: 1
Value: 0.2
> 2
0.100000 0.200000 0.300000 (size: 3)
> 4
Position: 1
> 2
0.100000 0.300000 (size: 2)
> 0

5. Листинг программы

spisok.h:

```
#pragma once

//list itself
struct List;
typedef struct List List;

//iterator
struct Iterator;
typedef struct Iterator Iterator;

//list functions
List* list_create();
void list_free(List *l);
//size
int list_size(List *l);
char list_empty(List *l);
//iterators
Iterator* list_begin(List *l);
Iterator* list_end(List *l);
//insert/erase elements
void list_clear(List *l);
void list_insert(List *l, Iterator *i, double val);
void list_erase(List *l, Iterator *i);

//iterator moving
void iterator_left(Iterator *i);
void iterator_right(Iterator *i);
//getting/seting values
double iterator_get(Iterator *i);
void iterator_set(Iterator *i, double val);
//remove iterator
void iterator_free(Iterator *i);
```

spisok.c:

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

//list itself
struct Node
{
	double value;
	struct Node *next, *prev;
};
struct Iterator
{
	struct Node *pointer;
};
struct List
{
```

```
	struct Node *head, *tail;
};

//list functions
List* list_create()
{
	List *list = malloc(sizeof(List)); //create list
	list->head = list->tail = malloc(sizeof(struct Node)); //create terminator
	list->tail->next = list->tail->prev = NULL; //setup
	return list;
}
void list_free(List *l)
{
	list_clear(l); //remove all elements
	free(l->tail); //remove terminator
	free(l); //remove list
}
//size
int list_size(List *l)
{
	int sz = 0;
	struct Node *node = l->head;
	while (node != l->tail) //how many elements between head and terminator
	{
	node = node->next;
	sz++;
	}
	return sz;
}
char list_empty(List *l)
{
	return l->head == l->tail; //no elements except terminator
}
//iterators
Iterator* list_begin(List *l)
{
	Iterator *iter = malloc(sizeof(Iterator)); //create iterator
	iter->pointer = l->head; //setup
	return iter;
}
Iterator* list_end(List *l)
{
	Iterator *iter = malloc(sizeof(Iterator)); //create iterator
	iter->pointer = l->tail; //setup
	return iter;
}
//insert/erase elements
void list_clear(List *l)
{
	struct Node *node;
	while (l->head != l->tail)
```

```
		{
		node = l->head->next;
		free(l->head); //remove element
		l->head = node; //go to next one
		}
		l->head->prev = NULL;
}
void list_insert(List *l, Iterator *i, double val) //insert BEFORE iterator
{
		struct Node *node = i->pointer; //current element
		struct Node *memory = malloc(sizeof(struct Node)); //allocate memory
		memory->value = val;

		//rearrange pointers
		memory->next = node;
		memory->prev = node->prev;
		node->prev = memory;

		if (memory->prev) memory->prev->next = memory; //not first element
		else l->head = memory; //first element
}
void list_erase(List *l, Iterator *i)
{
		struct Node *node = i->pointer; //current element
		if (!node->next) return; //attempt to erase terminator

		//redirect pointers
		node->next->prev = node->prev;
		if (node->prev) node->prev->next = node->next; //not first element
		else l->head = node->next; //first element

		free(node); //deallocate memory
}

//iterator moving
void iterator_left(Iterator *i)
{
		if (i->pointer->prev) i->pointer = i->pointer->prev;
}
void iterator_right(Iterator *i)
{
		if (i->pointer->next) i->pointer = i->pointer->next;
}
//getting/seting values
double iterator_get(Iterator *i)
{
		return i->pointer->value;
}
void iterator_set(Iterator *i, double val)
{
		i->pointer->value = val;
```

```
}
//remove iterator
void iterator_free(Iterator *i)
{
	free(i);
}
```

main1.c:

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

void print_menu()
{
	printf(
	"1 - show menu\n"
	"2 - show list contents\n"
	"3 - insert element\n"
	"4 - erase element\n"
	"5 - clear list\n"
	"6 - set value\n"
	"0 - exit\n"
	);
}

int main()
{
	List *l = list_create(); //create list

	char loop = 1;
	int com;

	print_menu(); //show menu at start

	while (loop)
	{
	printf("> ");
	scanf("%i", &com); //get user command
	switch (com)
	{
	case 0: //exit
	loop = 0;
	break;

	case 1: //show menu
	print_menu();
	break;

	case 2: //show list
	if (list_empty(l)) printf("*list is empty*\n");
	else
	{
	int n = list_size(l), sz = n;
```

```
Iterator *i = list_begin(l);

while (n--) //go through all list elements
{
        printf("%f ", iterator_get(i));
        iterator_right(i);
}
printf("(size: %i)\n", sz); //show size at the end
iterator_free(i);
}
break;

case 3: //insert element
{
int pos; double val; //get position and value
printf("Position: "); scanf("%i", &pos);
printf("Value: "); scanf("%lf", &val);

Iterator *i = list_begin(l);

while (pos--) iterator_right(i); //go to the pos
list_insert(l, i, val);

iterator_free(i);
break;
}

case 4: //erase element
{
int pos; //get position and value
printf("Position: "); scanf("%i", &pos);

Iterator *i = list_begin(l);

while (pos--) iterator_right(i); //go to the pos
list_erase(l, i);

iterator_free(i);
break;
}

case 5: //clear list
list_clear(l);
break;

case 6: //set value
{
int pos; double val; //get position and value
printf("Position: "); scanf("%i", &pos);
printf("Value: "); scanf("%lf", &val);
```

```
        Iterator *i = list_begin(l);

        while (pos--) iterator_right(i); //go to the pos
        iterator_set(i, val);

        iterator_free(i);
        break;
        }
        }
        }

        list_free(l); //free memory
        return 0;
}
```

main2.c:

```
#include <stdio.h>
#include <dlfcn.h>
//list itself
struct List;
typedef struct List List;

//iterator
struct Iterator;
typedef struct Iterator Iterator;

//list functions
List* (*list_create)();
void (*list_free)(List *l);
//size
int (*list_size)(List *l);
char (*list_empty)(List *l);
//iterators
Iterator* (*list_begin)(List *l);
Iterator* (*list_end)(List *l);
//insert/erase elements
void (*list_clear)(List *l);
void (*list_insert)(List *l, Iterator *i, double val);
void (*list_erase)(List *l, Iterator *i);

//iterator moving
void (*iterator_left)(Iterator *i);
void (*iterator_right)(Iterator *i);
//getting/seting values
double (*iterator_get)(Iterator *i);
void (*iterator_set)(Iterator *i, double val);
//remove iterator
void (*iterator_free)(Iterator *i);

void print_menu()
{
        printf(
```

```
		"1 - show menu\n"
		"2 - show list contents\n"
		"3 - insert element\n"
		"4 - erase element\n"
		"5 - clear list\n"
		"6 - set value\n"
		"0 - exit\n"
		);
}

int main()
{
		void *lib = dlopen("./libmyspisok.so", RTLD_LAZY);
		if (!lib)
		{
		printf("Cannot open lib because: %s\n", dlerror());
		return 1;
		}
		//list functions
		list_create = dlsym(lib, "list_create");
		list_free = dlsym(lib, "list_free");
		//size
		list_size = dlsym(lib, "list_size");
		list_empty = dlsym(lib, "list_empty");
		//iterators
		list_begin = dlsym(lib, "list_begin");
		list_end = dlsym(lib, "list_end");
		//insert/erase elements
		list_clear = dlsym(lib, "list_clear");
		list_insert = dlsym(lib, "list_insert");
		list_erase = dlsym(lib, "list_erase");

		//iterator moving
		iterator_left = dlsym(lib, "iterator_left");
		iterator_right = dlsym(lib, "iterator_right");
		//getting/seting values
		iterator_get = dlsym(lib, "iterator_get");
		iterator_set = dlsym(lib, "iterator_set");
		//remove iterator
		iterator_free = dlsym(lib, "iterator_free");

		List *l = list_create(); //create list

		char loop = 1;
		int com;

		print_menu(); //show menu at start

		while (loop)
		{
		printf("> ");
```

```
scanf("%i", &com); //get user command
switch (com)
{
case 0: //exit
loop = 0;
break;

case 1: //show menu
print_menu();
break;

case 2: //show list
if (list_empty(l)) printf("*list is empty*\n");
else
{
int n = list_size(l), sz = n;
Iterator *i = list_begin(l);

while (n--) //go through all list elements
{
        printf("%f ", iterator_get(i));
        iterator_right(i);
}
printf("(size: %i)\n", sz); //show size at the end
iterator_free(i);
}
break;

case 3: //insert element
{
int pos; double val; //get position and value
printf("Position: "); scanf("%i", &pos);
printf("Value: "); scanf("%lf", &val);

Iterator *i = list_begin(l);

while (pos--) iterator_right(i); //go to the pos
list_insert(l, i, val);

iterator_free(i);
break;
}

case 4: //erase element
{
int pos; //get position and value
printf("Position: "); scanf("%i", &pos);

Iterator *i = list_begin(l);

while (pos--) iterator_right(i); //go to the pos
```

```
            list_erase(l, i);

            iterator_free(i);
            break;
            }

            case 5: //clear list
            list_clear(l);
            break;

            case 6: //set value
            {
            int pos; double val; //get position and value
            printf("Position: "); scanf("%i", &pos);
            printf("Value: "); scanf("%lf", &val);

            Iterator *i = list_begin(l);

            while (pos--) iterator_right(i); //go to the pos
            iterator_set(i, val);

            iterator_free(i);
            break;
            }
            }
            }

            list_free(l); //free memory

            dlclose(lib);
            return 0;
}
```

Makefile:

```
all:
   gcc main1.c -o test1 -lmyspisok -L.
   gcc main2.c -o test2 -ldl
lib:
   gcc spisok.c -fPIC -c
   gcc -shared -Wl,-soname,libmyspisok.so.1 -o libmyspisok.so.1.0 spisok.o
   ldconfig -n .
   ln -sf libmyspisok.so.1 libmyspisok.so
```

6. Выводы:

Научился создавать динамические библиотеки и подключать их различными способами в своих программах.