

Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа № 5
по курсу «Операционные системы»

Студент:	Валов В.В
Группа:	М8О-308Б-18
Вариант:	25
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва, 2020

1. Постановка задачи

Структура данных, с которой должна обеспечивать работу библиотека:

4. Работа с бинарным деревом поиска.

Тип данных, используемый структурой:

4. Md5 суммы.

Операционная система: Unix.

Цель работы

Приобретение практических навыков в:

- Создание динамических библиотек
- Создание программ, которые используют функции динамических библиотек

Задание

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 вариант интерфейса;
- Тестовая программа, которая использует 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.

2. Решение задачи

Реализованная стандартная библиотека для бинарного дерева поиска с функциями: создание и удаление дерева, вставка в дерево, удаление из дерева, поиск в дереве, проверка дерева на пустоту, а также печать дерева.

В первом случае линкования во время компиляции указываем путь до библиотеки и ее название с стандартным использованием функций. А во втором случае, рантайм линковки нужно явно открывать библиотеку с помощью утилиты `dlopen()`, а затем присваивать указателям на функции результат утилиты `dsym()`, который функции по имени в библиотеке.

Используемые системные вызовы:

- **void exit(int status)** — функция выхода из процесса с заданным статусом.
- **void *dlopen(const char *filename, int flag)** - открывает файл по пути *filename* если NULL, то по умолчанию открывается `main()` со свойствами `flag`. Если библиотека

имеет зависимости, то они также подключаются с теми же свойствами. В случае ошибки возвращает NULL. *Flag* обязательно должен иметь либо RTLD LAZY, либо RTLD NOW, которые отвечают за загрузку библиотеки.

- **char *dlderror(void)** - возвращает строку, которая описывает ошибку. Если ошибки не было, то возвращает NULL.
- **void *dlsym(void *handle, const char *symbol)** – поиск функции в дереве, подключенных через dlopen() библиотек строку symbol, если подходит, то возвращает void* участок памяти, связанный с функцией. В случае ошибки возвращает NULL, однако может вернуть NULL и в случае успеха, поэтому обязательна проверка с помощью dlderror(), которая в свою очередь устанавливает ошибку.
- **int dlclose(void *handle)** – уменьшает количество ссылок на подключенную динамическую библиотеку, если он становится равным 0, то библиотека отсоединяется. При успешном выполнении возвращает 0.

Тесты программы:

```
vadim@vadim:~/Desktop/os/lab5$ export LD_LIBRARY_PATH=$(pwd)
```

```
vadim@vadim:~/Desktop/os/lab5$ echo $LD_LIBRARY_PATH  
/home/julia/Рабочий стол/2 курс/oc/lab5/src/25 в
```

```
vadim@vadim:~/Desktop/os/lab5$ make  
gcc -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare -pedantic -lm -o  
run-static mainStat.o -L. -lbtree -Wl,-rpath,.  
gcc -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare -pedantic -lm -o  
run-dynamic mainDyn.o -ldl
```

```
vadim@vadim:~/Desktop/os/lab5$ ./run-static
```

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

1

Enter key: r

Error: insert correct MD5

74c557c8ba571c8e518f9593e9c8e9cb

1

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

Enter key: 4

Error: insert correct MD5

74c557c8ba571c8e518f9593e9c8e9cb

4

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

74c557c8ba571c8e518f9593e9c8e9cb

74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

4

74c557c8ba571c8e518f9593e9c8e9cb

74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

4

74c557c8ba571c8e518f9593e9c8e9cb

74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

1

Enter key: 2db95e8e1a9267b7a1188556b2013b33

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

4

2db95e8e1a9267b7a1188556b2013b33

74c557c8ba571c8e518f9593e9c8e9cb

74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

1

Enter key: 3d3d7232bca83b8c711deacf7d5f19f5

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree

Press 0 to Exit

4

2db95e8e1a9267b7a1188556b2013b33

3d3d7232bca83b8c711deacf7d5f19f5

74c557c8ba571c8e518f9593e9c8e9cb

74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:

Press 1 to Add key

Press 2 to Remove key

Press 3 to Find key

Press 4 to Print tree
Press 0 to Exit
2
Enter key: 74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:
Press 1 to Add key
Press 2 to Remove key
Press 3 to Find key
Press 4 to Print tree
Press 0 to Exit
4

2db95e8e1a9267b7a1188556b2013b33
3d3d7232bca83b8c711deacf7d5f19f5
74c557c8ba571c8e518f9593e9c8e9cb

This is compile-time linking

Choose an operation:
Press 1 to Add key
Press 2 to Remove key
Press 3 to Find key
Press 4 to Print tree
Press 0 to Exit
0

vadim@vadim:~/Desktop/os/lab5\$

vadim@vadim:~/Desktop/os/lab5\$./run-dynamic

This is runtime linking

Choose an operation:
>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key
>> Press 4 to Print tree
>> Press 0 to Exit

1
Enter key: 3d3d7232bca83b8c711deacf7d5f19f5

This is runtime linking

Choose an operation:
>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key

>> Press 4 to Print tree
>> Press 0 to Exit

4

3d3d7232bca83b8c711deacf7d5f19f5

This is runtime linking

Choose an operation:

>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key
>> Press 4 to Print tree
>> Press 0 to Exit

1

Enter key: 74c557c8ba571c8e518f9593e9c8e9cb

This is runtime linking

Choose an operation:

>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key
>> Press 4 to Print tree
>> Press 0 to Exit

4

3d3d7232bca83b8c711deacf7d5f19f5
74c557c8ba571c8e518f9593e9c8e9cb

This is runtime linking

Choose an operation:

>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key
>> Press 4 to Print tree
>> Press 0 to Exit

1

Enter key: 2db95e8e1a9267b7a1188556b2013b33

This is runtime linking

Choose an operation:

>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key
>> Press 4 to Print tree
>> Press 0 to Exit

4

2db95e8e1a9267b7a1188556b2013b33
3d3d7232bca83b8c711deacf7d5f19f5
74c557c8ba571c8e518f9593e9c8e9cb

This is runtime linking

Choose an operation:

>> Press 1 to Add key
>> Press 2 to Remove key
>> Press 3 to Find key
>> Press 4 to Print tree
>> Press 0 to Exit

0

vadim@vadim:~/Desktop/os/lab5\$

3.Руководство по использованию программы

Компиляция и запуск программного кода в *Ubuntu* :

```
export LD_LIBRARY_PATH=$(pwd)
make clean
make
./run-static
./run-dynamic
```

4.Листинг программы

```
#ifndef _BTREE_H_
#define _BTREE_H_

#define SUCCESS 0
#define FAILURE 1
#define MD5 32

#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <stdbool.h>
#include <string.h>

typedef char ElemType;

typedef struct btree {
    struct btree *left;
    struct btree *right;
```



```

    ElemType key [MD5 + 1];
} *BTREE;

extern void TreeInsert(BTREE *root, ElemType* newKey);
extern BTREE TreeFind(BTREE root, ElemType* key);
extern BTREE TreeRemove(BTREE root, ElemType* key);
extern void TreePrint(BTREE root);
extern void TreeDestroy(BTREE root);
extern bool TreeIsEmpty(BTREE root);
#endif /* _BTREE_H */

#include "btree.h"

void TreeInsert(BTREE *root, ElemType* newKey)
{
    if (!(*root)) {
        BTREE newNode = (BTREE) malloc(sizeof(*newNode));
        if (!newNode) {
            printf("Error: no memory\n");
            exit(FAILURE);
        }

        newNode->left = newNode->right = NULL;
        strcpy(newNode->key, newKey);
        *root = newNode;

        return;
    }

    if (strcmp(newKey, (*root)->key) <= 0) {
        TreeInsert(&(*root)->left, newKey);
    } else {
        TreeInsert(&(*root)->right, newKey);
    }
}

BTREE TreeFind(BTREE root, ElemType* key)
{
    if (!root) {
        return root;
    }

    if (strcmp(key, root->key) < 0) {
        return TreeFind(root->left, key);
    } else if (strcmp(key, root->key) > 0) {
        return TreeFind(root->right, key);
    } else {
        return root;
    }
}

BTREE minValueNode(BTREE root)
{
    BTREE cur = root;
    while (cur->left)
        cur = cur->left;
    return cur;
}

BTREE TreeRemove(BTREE root, ElemType* key)
{
    if (!root)

```

```

        return root;

    if (strcmp(key, root->key) < 0) {
        root->left = TreeRemove(root->left, key);
    } else if (strcmp(key, root->key) > 0) {
        root->right = TreeRemove(root->right, key);
    } else {
        if (!root->left) {
            BTREE tmp = root->right;
            free(root);
            root = NULL;
            return tmp;
        } else if (!root->right) {
            BTREE tmp = root->left;
            free(root);
            root = NULL;
            return tmp;
        }

        BTREE tmp = minValueNode(root->right);
        strcpy(root->key, tmp->key);
        root->right = TreeRemove(root->right, tmp->key);
    }
    return root;
}

void TreeNodePrint(BTREE node, int idx)
{
    if (node) {
        TreeNodePrint(node->left, idx + 1);
        for (int j = 0; j < idx; ++j)
            putchar('\t');
        printf("%s\n", node->key);
        TreeNodePrint(node->right, idx + 1);
    }
}

void TreePrint(BTREE root)
{
    if (root) {
        TreeNodePrint(root, 0);
    } else {
        printf("Tree is empty\n");
    }
}

void TreeDestroy(BTREE root)
{
    if (root) {
        TreeDestroy(root->right);
        TreeDestroy(root->left);
    }
    free(root);
    root = NULL;
}

bool TreeIsEmpty(BTREE root)
{
    return !root;
}

#include <stdio.h>

```

```

#include <stdlib.h>

#include "btree.h"

void help()
{
    printf("\n\n");
    printf("This is compile-time linking\n\n");
    printf("Choose an operation:\n");
    printf(">> Press 1 to Add key\n");
    printf(">> Press 2 to Remove key\n");
    printf(">> Press 3 to Find key\n");
    printf(">> Press 4 to Print tree\n");
    printf(">> Press 0 to Exit\n");
    printf("\n");
}

int main(void)
{
    int act = 0;
    ElemType key [MD5 + 1] = "";
    BTREE tree = NULL;
    help();
    while (scanf("%d", &act) && act) {
        switch(act) {
            case 1:
                printf("Enter key: ");
                scanf("%s", &key);
                while( (strlen(key) != 32)) {
                    printf("Error: insert correct MD5\n");
                    scanf("%s\n", &key);
                }

                TreeInsert(&tree, key);
                break;

            case 2:
                if(TreelsEmpty(tree))
                    printf("Tree is empty\n");
                else {
                    printf("Enter key: ");
                    scanf("%s", &key);
                    while( (strlen(key) != 32)) {
                        printf("Error: insert correct MD5\n");
                        scanf("%s\n", &key);
                    }

                    if (TreeFind(tree, key)) {
                        tree = TreeRemove(tree, key);
                    } else {
                        printf("This key doesn't exist\n");
                    }
                }
                break;

            case 3:
                if(TreelsEmpty(tree))
                    printf("Tree is empty\n");
                else {
                    printf("Enter key: ");
                    scanf("%s", &key);
                    while( (strlen(key) != 32)) {
                        printf("Error: insert correct MD5\n");
                        scanf("%s\n", &key);
                    }

```

```

        }

        if (TreeFind(tree, key)) {
            printf("Key found\n");
        } else {
            printf("Key not found\n");
        }
    }
    break;

case 4:
    if (tree) {
        printf("\n");
        TreePrint(tree);
        printf("\n");
    } else {
        printf("Tree is empty\n");
    }
    break;

default:
    printf("Incorrect command\n");
    break;

    }
    help();
}

TreeDestroy(tree);
return SUCCESS;
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

```

```

#include "btree.h"

```

```

void help()
{
    printf("\n_____ \n");
    printf("This is runtime linking\n\n");
    printf("Choose an operation:\n");
    printf(">> Press 1 to Add key\n");
    printf(">> Press 2 to Remove key\n");
    printf(">> Press 3 to Find key\n");
    printf(">> Press 4 to Print tree\n");
    printf(">> Press 0 to Exit\n");
    printf("_____ \n");
}

```

```

int main(void)
{
    void (*TreeInsert)(BTREE *root, ElemType* newKey);
    BTREE (*TreeFind)(BTREE root, ElemType* key);
    BTREE (*TreeRemove)(BTREE root, ElemType* key);
    void (*TreePrint)(BTREE root);
    void (*TreeDestroy)(BTREE root);
    char *err;

    void *libHandle;
    libHandle = dlopen("libbtree.so", RTLD_LAZY);
    if (!libHandle) {
        fprintf(stderr, "%s\n", dlerror());
        exit(FAILURE);
    }
}

```

```

}

TreeInsert = dlsym(libHandle, "TreeInsert");
TreeRemove = dlsym(libHandle, "TreeRemove");
TreeFind = dlsym(libHandle, "TreeFind");
TreePrint = dlsym(libHandle, "TreePrint");
TreeDestroy = dlsym(libHandle, "TreeDestroy");

if(err = dlerror()) {
    fprintf(stderr, "%s\n", err);
    exit(FAILURE);
}

int act = 0;
ElemType key [MD5 + 1] = "";
BTREE tree = NULL;
help();
while (scanf("%d", &act) && act) {
    switch(act) {
        case 1:
            printf("Enter key: ");
            scanf("%s", &key);

            while( (strlen(key) != 32)) {
                printf("Error: insert correct MD5\n");
                scanf("%s\n", &key);
            }

            (*TreeInsert>(&tree, key);
            break;
        case 2:
            printf("Enter key: ");
            scanf("%s", &key);

            while( (strlen(key) != 32)) {
                printf("Error: insert correct MD5\n");
                scanf("%s\n", &key);
            }

            if ((*TreeFind)(tree, key)) {
                tree = (*TreeRemove)(tree, key);
            } else {
                printf("This key doesn't exist\n");
            }
            break;
        case 3:
            printf("Enter key: ");

            while( (strlen(key) != 32)) {
                printf("Error: insert correct MD5\n");
                scanf("%s\n", &key);
            }

            scanf("%s", &key);
            if ((*TreeFind)(tree, key)) {
                printf("Key found\n");
            } else {
                printf("Key not found\n");
            }
            break;
        case 4:

```

```

        if (tree) {
            printf("\n");
            (*TreePrint)(tree);
            printf("\n");
        } else {
            printf("Tree is empty\n");
        }
        break;
default:
    printf("Error: incorrect command\n");
    break;
}
help();
}
(*TreeDestroy)(tree);
dlclose(libHandle);
return SUCCESS;
}

CC = gcc
FLAGS = -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare -pedantic -lm

all: run

run: libbtree.so mainStat.o mainDyn.o
    $(CC) $(FLAGS) -o run-stat mainStat.o -L. -lbtree -Wl,-rpath,.
    $(CC) $(FLAGS) -o run-dyn mainDyn.o -ldl

mainStat.o: mainStat.c
    $(CC) -c $(FLAGS) mainStat.c

mainDyn.o: mainDyn.c
    $(CC) -c $(FLAGS) mainDyn.c

btree.o: btree.c
    $(CC) -c -fPIC $(FLAGS) btree.c

libbtree.so: btree.o
    $(CC) $(FLAGS) -shared -o libbtree.so btree.o

clean:
    rm -f *.o run-stat run-dyn *.o

```

5. Вывод

Статическое линкование удобно тем, что собирает программу и рантайм в один файл. После запуска программы, реализация используемых функций ищется в сборке, таким образом гарантируется переносимость программы. Как результат — сборка увеличивается в размере. При динамическом линковании мы получаем сборку без сторонних библиотек. Ее размер, определенно, меньше, однако мы должны быть уверены, что на машине клиента присутствовать библиотека, используемая в программе, и ее версия совпадает с той, что была использована при сборке. У обоих способов есть свои плюсы и свои минусы, выбор зависит лишь от требуемого результата.