Московский авиационный институт
Национальный исследовательский университет

Операционные системы
Кафедра 806

Лабораторная работа №5

Студент: Минибаев Айдар
Группа: М8О-301Б-18
Преподаватель: Миронов Е.С.
Дата: .
Подпись: .

Москва 2020

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)
2. Во время исполнения программы, подгрузив библиотеку в память с помощью системных вызовов.

В конечном счете, программа должно состоять из следующих частей:

1. Динамическая библиотека, реализующая заданный вариантом интерфейс.
2. Тестовая программа, которая использует библиотеку, используя знания полученные на этапе компиляции.
3. Тестовая программа, которая использует библиотеку, используя только местоположение динамической библиотеки и ее интерфейс.

Структура данных, с которой должна обеспечивать работу библиотека: работа с бинарным деревом поиска. Тип данных, используемый структурой: целочисленный 32-битный тип.

2. Метод решения

Написал библиотеку для работы с бинарным деревом поиска. В его интерфейс входит создание дерева, вставка в дерево, удаление из дерева, печать дерева, поиск в дереве, удаление дерева, проверка дерева на пустоту. В случае линковки во время компиляции указываем путь до библиотеки и указываем название библиотеки, пользуемся функциями как обычно. В случае рантайм линковки нужно явно открыть библиотеку с помощью dlopen(), а затем присваивать указателям на функции результат dlsym(), который ищет по названию функции в библиотеке.

3. Программная реализация

Makefile:

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

all: run

run: libbst.so mainStat.o mainDyn.o
	$(CC) $(FLAGS) -o run-stat mainStat.o -L. -lbst -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

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

libbst.so: bst.o
	$(CC) $(FLAGS) -shared -o libbst.so bst.o
```

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

bst.h:

```
#ifndef _BST_H_
#define _BST_H_

#define SUCCESS 0
#define FAILURE 1

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

typedef int32_t ElemType;

typedef struct _bst {
   struct _bst *left;
   struct _bst *right;
   ElemType key;
} *BST;

extern void TreeInsert(BST *root, ElemType newKey);
extern BST TreeFind(BST root, ElemType key);
extern BST TreeRemove(BST root, ElemType key);
extern void TreePrint(BST root);
extern void TreeDestroy(BST root);
extern bool TreeIsEmpty(BST root);

#endif /* _BST_H */
```

bst.c:

```
#include "bst.h"

void TreeInsert(BST *root, ElemType newKey)
{
   if (!(*root)) {
       BST newNode = (BST) malloc(sizeof(*newNode));
       if (!newNode) {
           printf("Error: no memory\n");
           exit(FAILURE);
       }
       newNode->left = newNode->right = NULL;
       newNode->key = newKey;
       *root = newNode;
       return;
   }

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

```
        TreeInsert(&(*root)->right, newKey);
    }
}

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

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

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

BST TreeRemove(BST root, ElemType key)
{
    if (!root)
        return root;

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

        BST tmp = minValueNode(root->right);
        root->key = tmp->key;
        root->right = TreeRemove(root->right, tmp->key);
    }
    return root;
}
```

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

void TreePrint(BST root)
{
   if (root) {
       TreeNodePrint(root, 0);
   } else {
       printf("BST died\n");
   }
}

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

bool TreeIsEmpty(BST root)
{
   return !root;
}
```

mainDyn.c:

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

#include "bst.h"

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

   void *libHandle;
   libHandle = dlopen("libbst.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 = 0;
BST tree = NULL;
printf("This is runtime linking\n\n");
printf("Choose an operation:\n");
printf("1) Add key\n");
printf("2) Remove key\n");
printf("3) Find key\n");
printf("4) Print tree\n");
printf("0) Exit\n");
while (scanf("%d", &act) && act) {
    switch(act) {
    case 1:
        printf("Enter key: ");
        scanf("%d", &key);
        (*TreeInsert)(&tree, key);
        break;
    case 2:
        printf("Enter key: ");
        scanf("%d", &key);
        if ((*TreeFind)(tree, key)) {
            tree = (*TreeRemove)(tree, key);
        } else {
            printf("This key doesn't exist\n");
        }
        break;
    case 3:
        printf("Enter key: ");
        scanf("%d", &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;
        }
        printf("Choose an operation:\n");
        printf("1) Add key\n");
        printf("2) Remove key\n");
        printf("3) Find key\n");
        printf("4) Print tree\n");
        printf("0) Exit\n");
    }
    (*TreeDestroy)(tree);
    dlclose(libHandle);
    return SUCCESS;
}
```

mainStat.c:

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

#include "bst.h"

int main(void)
{
    int act = 0;
    ElemType key = 0;
    BST tree = NULL;
    printf("This is compile-time linking\n\n");
    printf("Choose an operation:\n");
    printf("1) Add key\n");
    printf("2) Remove key\n");
    printf("3) Find key\n");
    printf("4) Print tree\n");
    printf("0) Exit\n");
    while (scanf("%d", &act) && act) {
        switch(act) {
        case 1:
            printf("Enter key: ");
            scanf("%d", &key);
            TreeInsert(&tree, key);
            break;
        case 2:
            printf("Enter key: ");
            scanf("%d", &key);
            if (TreeFind(tree, key)) {
                tree = TreeRemove(tree, key);
            } else {
                printf("This key doesn't exist\n");
            }
```

```
            break;
        case 3:
            printf("Enter key: ");
            scanf("%d", &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;
        }
        printf("Choose an operation:\n");
        printf("1) Add key\n");
        printf("2) Remove key\n");
        printf("3) Find key\n");
        printf("4) Print tree\n");
        printf("0) Exit\n");
    }
    TreeDestroy(tree);
    return SUCCESS;
}
```

4. Тестирование

```
$ make
gcc -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall
-Wno-sign-compare -pedantic -lm -o run-stat mainStat.o -L. -lbst
-Wl,-rpath,.
gcc -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall
-Wno-sign-compare -pedantic -lm -o run-dyn mainDyn.o -ldl
$ ./run-dyn
This is runtime linking

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

```
4
Tree is empty
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 10
This key doesn't exist
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 1
Key not found
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
```

```
0) Exit
4

       10
10

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 90
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

       10
10
       90

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 1
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4
```

```
               1
       10
10
       90

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

               1
       10
90

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4
```

```
       1
90

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

90

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

```
$ ./run-stat

This is runtime linking

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4
Tree is empty
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 10
This key doesn't exist
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 1
Key not found
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
```

```
3) Find key
4) Print tree
0) Exit
1
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

       10
10

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 90
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

       10
10
       90

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
```

```
0) Exit
1
Enter key: 1
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

                1
        10
10
        90

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

                1
        10
90

Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
```

```
0) Exit
2
Enter key: 10
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

       1
90

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

90

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

5. Выводы

Статическая линковка удобна 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 размерах. При динамической линковке мы получаем "голую" сборку без сторонних библиотек. Ее размер, безусловно, меньше, но при этом мы должны гарантировать, что на клиентской машине имеется библиотека, используемая в программе, и ее версия одинакова с той, которая была использована при сборке. В обоих способах есть минусы и плюсы, выбор зависит от результата, который мы хотим получить.