Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Операционные системы»

Студентка: А. Довженко
Преподаватель: Е. С. Миронов
Группа: 08-207
Вариант: 4
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №5

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.

Вариант 4.
Структура данных, с которой должна обеспечивать работу библиотека: работа с бинарным деревом поиска.
Тип данных, используемый структурой: целочисленный 32-битный тип.

1 Описание

Была написана стандартная библиотека для бинарного дерева 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.

Системные вызовы:

void exit(int status); – функция выходит из процесса с заданным статусом.
void *dlopen(const char *filename, int flag); – открывает файл по пути filename (если NULL, то 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 (по частям, когда потребуется, либо все сразу).
char *dlerror(void); – возвращает строку, которая описывает ошибку. Если ошибки не было, то возвращает NULL.
void *dlsym(void *handle, const char *symbol); – имещ в дереве подключенных через dlopen() библиотек строку symbol, если находит, то возвращает void * участок памяти, связанный с функцией. В случае ошибки возвращает NULL (может вернуть NULL и в случае успеха, поэтому обязательна проверка через dlerror) и устанавливает ошибку для dlerror()).
int dlclose(void *handle); – уменьшает количество ссылок на подключенную динамическую библиотеку, если он становится равным нулю, то библиотека отсоединяется. В случае успеха возвращает 0, иначе – не нуль.

2 Исходный код

3 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 */
```

4 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;
```

```
14          return;
15      }
16
17      if (newKey <= (*root)->key) {
18          TreeInsert(&(*root)->left, newKey);
19      } else {
20          TreeInsert(&(*root)->right, newKey);
21      }
22  }
23
24  BST TreeFind(BST root, ElemType key)
25  {
26      if (!root) {
27          return root;
28      }
29
30      if (key < root->key) {
31          return TreeFind(root->left, key);
32      } else if (key > root->key) {
33          return TreeFind(root->right, key);
34      } else {
35          return root;
36      }
37  }
38
39  BST minValueNode(BST root)
40  {
41      BST cur = root;
42      while (cur->left)
43          cur = cur->left;
44      return cur;
45  }
46
47  BST TreeRemove(BST root, ElemType key)
48  {
49      if (!root)
50          return root;
51
52      if (key < root->key) {
53          root->left = TreeRemove(root->left, key);
54      } else if (key > root->key) {
55          root->right = TreeRemove(root->right, key);
56      } else {
57          if (!root->left) {
58              BST tmp = root->right;
59              free(root);
60              root = NULL;
61              return tmp;
62          } else if (!root->right) {
```

```
63              BST tmp = root->left;
64              free(root);
65              root = NULL;
66              return tmp;
67          }
68
69          BST tmp = minValueNode(root->right);
70          root->key = tmp->key;
71          root->right = TreeRemove(root->right, tmp->key);
72      }
73      return root;
74  }
75
76  void TreeNodePrint(BST node, int idx)
77  {
78      if (node) {
79          TreeNodePrint(node->left, idx + 1);
80          for (int j = 0; j < idx; ++j)
81              putchar('\t');
82          printf("%d\n", node->key);
83          TreeNodePrint(node->right, idx + 1);
84      }
85  }
86
87  void TreePrint(BST root)
88  {
89      if (root) {
90          TreeNodePrint(root, 0);
91      } else {
92          printf("BST died\n");
93      }
94  }
95
96  void TreeDestroy(BST root)
97  {
98      if (root) {
99          TreeDestroy(root->right);
100         TreeDestroy(root->left);
101     }
102     free(root);
103     root = NULL;
104 }
105
106 bool TreeIsEmpty(BST root)
107 {
108     return !root;
109 }
```

5 mainStat.c

```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");
```

```
48              } else {
49                  printf("Tree is empty\n");
50              }
51              break;
52          default:
53              printf("Incorrect command\n");
54              break;
55          }
56          printf("Choose an operation:\n");
57          printf("1) Add key\n");
58          printf("2) Remove key\n");
59          printf("3) Find key\n");
60          printf("4) Print tree\n");
61          printf("0) Exit\n");
62      }
63      TreeDestroy(tree);
64      return SUCCESS;
65  }
```

6 mainDyn.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <dlfcn.h>
4
5   #include "bst.h"
6
7   int main(void)
8   {
9       void (*TreeInsert)(BST *root, ElemType newKey);
10      BST (*TreeFind)(BST root, ElemType key);
11      BST (*TreeRemove)(BST root, ElemType key);
12      void (*TreePrint)(BST root);
13      void (*TreeDestroy)(BST root);
14      char *err;
15
16      void *libHandle;
17      libHandle = dlopen("libbst.so", RTLD_LAZY);
18      if (!libHandle) {
19          fprintf(stderr, "%s\n", dlerror());
20          exit(FAILURE);
21      }
22
23      TreeInsert = dlsym(libHandle, "TreeInsert");
24      TreeRemove = dlsym(libHandle, "TreeRemove");
25      TreeFind = dlsym(libHandle, "TreeFind");
26      TreePrint = dlsym(libHandle, "TreePrint");
27      TreeDestroy = dlsym(libHandle, "TreeDestroy");
28
```

```
29      if(err = dlerror()) {
30          fprintf(stderr, "%s\n", err);
31          exit(FAILURE);
32      }
33
34      int act = 0;
35      ElemType key = 0;
36      BST tree = NULL;
37      printf("This is runtime linking\n\n");
38      printf("Choose an operation:\n");
39      printf("1) Add key\n");
40      printf("2) Remove key\n");
41      printf("3) Find key\n");
42      printf("4) Print tree\n");
43      printf("0) Exit\n");
44      while (scanf("%d", &act) && act) {
45          switch(act) {
46          case 1:
47              printf("Enter key: ");
48              scanf("%d", &key);
49              (*TreeInsert)(&tree, key);
50              break;
51          case 2:
52              printf("Enter key: ");
53              scanf("%d", &key);
54              if ((*TreeFind)(tree, key)) {
55                  tree = (*TreeRemove)(tree, key);
56              } else {
57                  printf("This key doesn't exist\n");
58              }
59              break;
60          case 3:
61              printf("Enter key: ");
62              scanf("%d", &key);
63              if ((*TreeFind)(tree, key)) {
64                  printf("Key found\n");
65              } else {
66                  printf("Key not found\n");
67              }
68              break;
69          case 4:
70              if (tree) {
71                  printf("\n");
72                  (*TreePrint)(tree);
73                  printf("\n");
74              } else {
75                  printf("Tree is empty\n");
76              }
77              break;
```

```
78          default:
79              printf("Incorrect command\n");
80              break;
81          }
82          printf("Choose an operation:\n");
83          printf("1) Add key\n");
84          printf("2) Remove key\n");
85          printf("3) Find key\n");
86          printf("4) Print tree\n");
87          printf("0) Exit\n");
88      }
89      (*TreeDestroy)(tree);
90      dlclose(libHandle);
91      return SUCCESS;
92  }
```

7 Makefile

```
1  CC = gcc
2  FLAGS = -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare -
       pedantic -lm
3
4  all: run
5
6  run: libbst.so mainStat.o mainDyn.o
7      $(CC) $(FLAGS) -o run-stat mainStat.o -L. -lbst -Wl,-rpath,.
8      $(CC) $(FLAGS) -o run-dyn mainDyn.o -ldl
9
10 mainStat.o: mainStat.c
11     $(CC) -c $(FLAGS) mainStat.c
12
13 mainDyn.o: mainDyn.c
14     $(CC) -c $(FLAGS) mainDyn.c
15
16 bst.o: bst.c
17     $(CC) -c -fPIC $(FLAGS) bst.c
18
19 libbst.so: bst.o
20     $(CC) $(FLAGS) -shared -o libbst.so bst.o
21
22 clean:
23     rm -f *.o run-stat run-dyn *.so
```

8 Тестирование

Сборка.

```
karma@karma:~/mai_study/OS/lab5$ make
gcc -c -fPIC -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compa
-pedantic -lm bst.c
gcc -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare
-pedantic -lm -shared -o libbst.so bst.o
gcc -c -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare
-pedantic -lm mainStat.c
gcc -c -std=c99 -pthread -w -pipe -O2 -Wextra -Werror -Wall -Wno-sign-compare
-pedantic -lm mainDyn.c
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
```

- Тестирование статической библиотеки:

  Печать пустого дерева.

```
karma@karma:~/mai_study/OS/lab5$ ./run-stat
This is compile-time linking

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

Удаление несуществующего ключа.

```
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
```

Поиск несуществующего ключа.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 1
Key not found
```

Вставка одинаковых ключей.

```
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
```

Вставка ключа в правое поддерево.

```
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
```

Вставка ключа в левое поддерево.

```
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
```

Удаление корня.

```
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
```

Удаление нетерминальной вершины.

```
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
```

Удаление листа.

```
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
```

Построим такое дерево.

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

1
50
90
100
```

Поиск удаленного ключа.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 10
Key not found
```

Поиск ключа в корне.

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

Поиск ключа в левом поддереве.

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

Поиск ключа в правом поддереве.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 100
Key found
```

- Тестирование динамической библиотеки

Печать пустого дерева.

```
karma@karma:~/mai_study/OS/lab5$ ./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
```

Удаление несуществующего ключа.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: -1
This key doesn't exist
```

Поиск несуществующего ключа.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 12345
Key not found
```

Вставка одинаковых ключей.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 20
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
```

```
Enter key: 20
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

20
20
```

Вставка ключа в правое поддерево.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
1
Enter key: 40
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

20
20
40
```

Вставка ключа в левое поддерево.

```
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
20
20
40
```

Удаление корня.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 20
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

10
```

```
20
40
```

Удаление нетерминальной вершины.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
2
Enter key: 20
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
4

10
40
```

Удаление листа.

```
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

40
```

Построим такое дерево.

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

4
34
40
140
```

Поиск удаленного ключа.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 20
Key not found
```

Поиск ключа в корне.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 40
Key found
```

Поиск ключа в левом поддереве.

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

Поиск ключа в правом поддереве.

```
Choose an operation:
1) Add key
2) Remove key
3) Find key
4) Print tree
0) Exit
3
Enter key: 140
Key found
```

9 Выводы

В который раз были написаны интерфейс и реализация одного из часто используемых АТД, в моем случае – бинарного дерева поиска. Это настолько частая задача при написании программ, что никаких трудностей у меня не возникло. Основная сложность была, как не странно, в сборке, а точнее в правильной настройке переменной окружения LD LIBRARY PATH. Работа с динамическими библиотеками может быть полезна при низкоуровневой реализации паттерна plug-in, который заключается в том, что можно подключать и отключать библиотеки во время работы.

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.