Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Операционные системы»

Работа с динамическими библиотеками

Студент: Симонов Сергей Яковлевич
Группа: М80 – 206Б-18
Вариант: 25
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019

Содержание

Постановка задачи

Целью является приобретение практических навыков в:

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

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

Общие сведения о программе, метод и алгоритм решения

Использовалась библиотека для бинарного дерева поиска. В ее интерфейс входит вставка в дерево, печать дерева, удаление узла из дерева, поиск элемента в дереве. Для линковки во время компиляции указываем путь до библиотеки и указываем название библиотеки, пользуемся функциями как обычно. В случае использования библиотеки во время использования нужно открыть библиотеку с помощью dlopen(), а затем присваивать указателям на функции результат dlsym(), который ищет по названию функции в библиотеке.
Использованные системные вызовы:
void *dlopen(const char *filename, int flag); – открывает файл по пути filename 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 *dlerror(void); – возвращает строку, которая описывает ошибку. Если ошибок не было, то возвращает NULL.
void *dlsym(void *handle, const char *symbol); – ищет в дереве подключенных через dlopen() библиотек строку symbol, если находит, то возвращает void * участок памяти, связанный с функцией. В случае ошибки возвращает NULL и устанавливает ошибку для dlerror().

Основные файлы программы

mainDyn.c

```
#include <stdlib.h>
#include <stdio.h>
#include "Tree.h"
#include <dlfcn.h>


int main(int argc, char *argv[]) {
```

```
tree* (*search)(tree *parent, char* value);
tree* (*min)(tree *parent);
tree* (*max)(tree *parent);
void (*printTree)(tree *root, int tabs);
void (*traverse)(tree *parent);
void (*insert)(tree **root, char* value, tree *parent);
tree* (*_delete)(tree *t, char* x);
int (*get_int)(char *prompt);
char* (*get_key)(char *prompt);
void (*flush_stdin)(void);

char *err;
void *libHandle;
libHandle = dlopen("libtree.so", RTLD_LAZY);
if (!libHandle) {
   fprintf(stderr, "%s\n", dlerror());
   exit(1);
}

search = dlsym(libHandle, "search");
min = dlsym(libHandle, "min");
max = dlsym(libHandle, "max");
printTree = dlsym(libHandle, "printTree");
traverse = dlsym(libHandle, "traverse");
insert = dlsym(libHandle, "insert");
_delete = dlsym(libHandle, "_delete");
get_int = dlsym(libHandle, "get_int");
flush_stdin = dlsym(libHandle, "flush_stdin");
get_key = dlsym(libHandle, "get_key");


int cmd = 0;
char* value;
tree *root = NULL;

do {
   printf("Commands\n\n"
         "1 - Insert\n"
         "2 - Search\n"
         "3 - Delete\n"
         "4 - Print tree\n"
         "0 - Quit\n\n");

   cmd = get_int("Command");
```

```
        switch (cmd) {
            case 1:
                value = get_key("value to insert");
                insert(&root, value, NULL);
                break;
            case 2:
                value = get_key("value to find");
                if (search(root, value) == NULL) {
                    printf("%s is not found in tree!\n", value);
                } else {
                    printf("%s is found in tree!\n", value);
                }
                break;
            case 3:
                value = get_key("value to delete");
                root = _delete(root, value);
                break;
            case 4:
                printf("tree:\n");
                printTree(root, 0);
                break;
        }

    } while (cmd != 0);

    return 0;
}
```

mainStat.c

```
#include <stdlib.h>
#include <stdio.h>
#include "Tree.h"
#include <dlfcn.h>


int main(int argc, char *argv[]) {

    tree* (*search)(tree *parent, char* value);
    tree* (*min)(tree *parent);
    tree* (*max)(tree *parent);
    void (*printTree)(tree *root, int tabs);
    void (*traverse)(tree *parent);
    void (*insert)(tree **root, char* value, tree *parent);
```

```
tree* (*_delete)(tree *t, char* x);
int (*get_int)(char *prompt);
char* (*get_key)(char *prompt);
void (*flush_stdin)(void);

char *err;
void *libHandle;
libHandle = dlopen("libtree.so", RTLD_LAZY);
if (!libHandle) {
   fprintf(stderr, "%s\n", dlerror());
   exit(1);
}

search = dlsym(libHandle, "search");
min = dlsym(libHandle, "min");
max = dlsym(libHandle, "max");
printTree = dlsym(libHandle, "printTree");
traverse = dlsym(libHandle, "traverse");
insert = dlsym(libHandle, "insert");
_delete = dlsym(libHandle, "_delete");
get_int = dlsym(libHandle, "get_int");
flush_stdin = dlsym(libHandle, "flush_stdin");
get_key = dlsym(libHandle, "get_key");


int cmd = 0;
char* value;
tree *root = NULL;

do {
   printf("Commands\n\n"
         "1 - Insert\n"
         "2 - Search\n"
         "3 - Delete\n"
         "4 - Print tree\n"
         "0 - Quit\n\n");

   cmd = get_int("Command");

   switch (cmd) {
      case 1:
         value = get_key("value to insert");
         insert(&root, value, NULL);
         break;
      case 2:
```

```
            value = get_key("value to find");
            if (search(root, value) == NULL) {
               printf("%s is not found in tree!\n", value);
            } else {
               printf("%s is found in tree!\n", value);
            }
            break;
         case 3:
            value = get_key("value to delete");
            root = _delete(root, value);
            break;
         case 4:
            printf("tree:\n");
            printTree(root, 0);
            break;
      }

   } while (cmd != 0);

   return 0;
}
```

Makefile

```
CC = gcc
FLAGS = -std=c99 -w -Werror -Wall -pedantic

all: run

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

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

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

libtree.so: Tree.o
	$(CC) $(FLAGS) -shared -o libtree.so Tree.o

Tree.o: Tree.c
	$(CC) -c -fPIC $(FLAGS) Tree.c
```

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

Tree.c

```
#include "Tree.h"
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>

// Binary search in a tree
tree *search(tree *parent, char* value) {
   if (parent == NULL) {
      return NULL;
   }

   if (strcmp(value,parent->value)==0) {
      return parent;
   }

   if (strcmp(value,parent->value)<0) {
      return search(parent->left, value);
   } else {
      return search(parent->right, value);
   }
}

// Find minimum item in the tree
tree *min(tree *parent) {
   if (parent == NULL) {
      return NULL;
   }

   tree *min = parent;
   while (min->left != NULL) {
      min = min->left;
   }

   return min;
}

// Find maximum item in the tree
tree *max(tree *parent) {
```

```
    if (parent == NULL) {
        return NULL;
    }

    tree *max = parent;
    while (max->right != NULL) {
        max = max->right;
    }

    return max;
}

// Walk through the tree using In-Order
void traverse(tree *parent) {
    if (parent != NULL) {
        traverse(parent->left);
        printf("(%d)->", parent->value);
        traverse(parent->right);
    }
}

// Insert an element in the tree
void insert(tree **root, char* value, tree *parent) {
    tree *p = NULL;

    // Insert to an empty tree
    if (*root == NULL) {
        p = malloc(sizeof(tree));

        p->value = value;
        p->left = NULL;
        p->right = NULL;
        p->parent = parent;

        *root = p;

        return;
    }

    if (strcmp(value, (*root)->value) < 0) {
        insert(&((*root)->left), value, *root);
    } else {
        insert(&((*root)->right), value, *root);
    }
}
```

```
tree * _delete(tree *t, char* x) {
    tree * tmp;
    if (t == NULL) {
        return t;
    }
    if (strcmp(x, t->value)<0) {
        t->left = _delete(t->left, x);
    } else {
        if (strcmp(x, t->value)>0) {
            t->right = _delete(t->right, x);
        } else {
            if (t->left && t->right) {
                tmp = min(t->right);
                t->value = tmp->value;
                t->right = _delete(t->right, t->value);
            } else {
                tmp = t;
                if (t->left == NULL) {
                    t = t->right;
                } else {
                    if (t->right == NULL) {
                        t = t->left;
                    }
                    free(tmp);
                }
            }
        }
    }
    return t;
}

// Get int
int get_int(char *prompt) {
    int value = 0;

    printf("%s: \n", prompt);
    scanf("%d", &value);

    flush_stdin();

    return value;
}

// Get key
```

```
char* get_key(char *prompt) {

    printf("%s: \n", prompt);

    int capacity_app_elem = 3;
    int size_app_elem = 0;
    char* app_element = ( char* ) malloc( sizeof( char ) * capacity_app_elem );
    char c;
    while ( 1 ) {
        if ( size_app_elem == capacity_app_elem ) {
            capacity_app_elem *= 2;
            app_element = ( char* ) realloc( app_element, sizeof( char ) *
capacity_app_elem );
        }
        c = getchar();
        if ( c == '\n' || c == ' ' ) break;
        app_element[size_app_elem] = c;
        size_app_elem++;
    }
    app_element[size_app_elem] = '\0';



    return app_element;
}

// Flush input buffer
void flush_stdin(void) {
    while (getc(stdin) != '\n');
}

void printTree(tree *root, int tabs) {
    if(root == NULL) return;

    printTree(root->right, tabs+1);
    for ( int i = 0; i < tabs; ++i ) {
        printf("\t");
    }
    printf("%s\n",root->value);
    printTree(root->left, tabs+1);
}
```

Tree.h

```
#ifndef LAB_5_TREE_H
#define LAB_5_TREE_H

typedef struct tree {
    char* value;
    struct tree *parent;
    struct tree *left;
    struct tree *right;
} tree;

tree *search(tree *parent, char* value);
tree *min(tree *parent);
tree *max(tree *parent);

void printTree(tree *root, int tabs);
void traverse(tree *parent);
void insert(tree **root, char* value, tree *parent);
tree * _delete(tree *t, char* x);

int get_int(char *prompt);
char* get_key(char *prompt);
void flush_stdin(void);


#endif //LAB_5_TREE_H
```

Демонстрация работы программы

Статическая

```
sergey@sergey-RedmiBook-14:~/labs/OS/lab5$ make
gcc -c -std=c99 -w -Werror -Wall -pedantic mainStat.c
gcc -std=c99 -w -Werror -Wall -pedantic -o stat
mainStat.o -L. -ltree -Wl,-rpath,.
gcc -std=c99 -w -Werror -Wall -pedantic -o dyn mainDyn.o
-ldl -Wl,-rpath,.
sergey@sergey-RedmiBook-14:~/labs/OS/lab5$ ./stat
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit
```

```
Command:
1
value to insert:
5
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
4
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
9
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
3
Commands

1 - Insert
2 - Search
```

```
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
6
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
4
tree:
       9
                6
5
       4
                3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
2
value to find:
9
9 is found in tree!
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit
```

```
Command:
2
value to find:
1
1 is not found in tree!
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
3
value to delete:
9
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
4
tree:
       6
5
       4
              3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
6
Commands
```

```
1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
4
tree:
       6
5
       4
              3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
0
```

Динамическая

```
sergey@sergey-RedmiBook-14:~/labs/OS/lab5$ make
gcc -std=c99 -w -Werror -Wall -pedantic -o stat
mainStat.o -L. -ltree -Wl,-rpath,.
gcc -std=c99 -w -Werror -Wall -pedantic -o dyn mainDyn.o
-ldl -Wl,-rpath,.
sergey@sergey-RedmiBook-14:~/labs/OS/lab5$ ./dyn
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
```

```
value to insert:
5
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
9
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
1
value to insert:
6
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
```

```
0 - Quit

Command:
4
tree:
       9
              6
5
       3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
2
value to find:
9
9 is found in tree!
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
2
value to find:
1
1 is not found in tree!
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
```

```
4
tree:
       9
                6
5
       3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
6
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
6
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
4
tree:
       9
                6
5
       3
Commands

1 - Insert
```

```
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
3
value to delete:
6
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
4
tree:
       9
5
       3
Commands

1 - Insert
2 - Search
3 - Delete
4 - Print tree
0 - Quit

Command:
0
```

Вывод

Выполнив лабораторную работу, я приобрел практические навыки в создании динамических библиотек, создании программ, которые используют функции динамических библиотек. Статическая линковка при компиляции упаковывает библиотеку в исполняемый файл, таким образом гарантируется переносимость программы. Негативные стороны этого таковы: увеличение исполняемого файла в размерах и необходимость обновлять весь файл при апдейте библиотеки. При динамической линковке библиотека подключается во время исполнения. Различные копии программы и другие программы, будут использовать одну библиотеку, это позволяет экономить оперативную память, но данная библиотека должна быть установлена на используемой машине, это несколько затрудняет распространение данной библиотеки.