

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики  
Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу  
«Операционные системы»**

**Потоки в операционных системах**

Студент	Гришин Алексей Юрьевич
Группа	М8О-208Б-21
Вариант	25
Преподаватель	Соколов Андрей Алексеевич
Оценка	5
Дата	28.11.2022
Подпись	

Москва, 2022.

## Постановка задачи

### Цель работы

Целью является приобретение практических навыков в:

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

### Задание

Требуется создать динамические библиотеки, которые реализуют определенный функционал. Далее использовать данные библиотеки 2-мя способами:

1. Во время компиляции (на этапе «линковки» / linking)
2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками

В конечном итоге, в лабораторной работе необходимо получить следующие части:

- Динамические библиотеки, реализующие контракты, которые заданы вариантом
- Тестовая программа (программа №1), которая использует одну из библиотек, используя знания полученные на этапе компиляции;
- Тестовая программа (программа №2), которая загружает библиотеки, используя только их местоположение и контракты.

Провести анализ двух типов использования библиотек. Пользовательский ввод для обеих программ должен быть организован следующим образом:

1. Если пользователь вводит команду “0”, то программа переключает одну реализацию контрактов на другую (необходимо только для программы №2);
2. «1 arg1 arg2 ... argN», где после “1” идут аргументы для первой функции, предусмотренной контрактами. После ввода команды происходит вызов первой функции, и на экране появляется результат ее выполнения;
3. «2 arg1 arg2 ... argM», где после “2” идут аргументы для второй функции, предусмотренной контрактами. После ввода команды происходит вызов второй функции, и на экране появляется результат ее выполнения.

## Вариант 25

№	Описание	Сигнатура	Реализация 1	Реализация 2
4	Подсчёт наибольшего общего делителя для двух натуральных чисел	<code>Int GCF(int A, int B)</code>	Алгоритм Евклида	Наивный алгоритм. Пытаться разделить числа на все числа, что меньше A и B.
8	Перевод числа x из десятичной системы счисления в другую	<code>char* translation(long x)</code>	Другая система счисления двоичная	Другая система счисления троичная

## Общие сведения о программе

Реализации 1 и 2 первой библиотеки компилируется из файлов `gcf/gcf1.c` и `gcf/gcf2.c` соответственно. Используется заголовочный файл `stdio.h` а также заголовочный файл `gcf/gcf.h`, который играет роль контракта между основной программой и библиотекой.

Реализации 1 и 2 второй библиотеки компилируются из файлов `translation/translation1.c` и `translation/translation2.c` соответственно. Используют вспомогательный заголовочный файл `translation/convert/convert.h`, так как логика реализаций практически одинаковая за исключением оснований системы счисления. Из стандартной библиотеки используются заголовочные файлы `string.h`, `stdlib.h`

Программа №1 компилируется из файла `main1.c` и использует контрактные заголовочные файлы `gcf/gcf.h` и `translation/translation.h`. Также используются заголовочные файлы `stdio.h` и `stdlib.h`.

Программа №2 компилируется из файла `main2.c` и использует те же контрактные заголовочные файлы, что и программа №1. Из стандартной библиотеки используются заголовочные файлы `stdio.h` и `stdlib.h`.

## Системные вызовы

Используются следующие библиотечные вызовы:

1. `dlopen(filename, flag)` – загружает динамическую библиотеку, соответствующую имени, переданному в качестве параметра `filename`. В параметре `flag` указываются флаги при загрузке динамической библиотеки. Флаг `RTLD_NOW` означает, что требуется немедленное разрешение всех неопределенных символов. При ошибке возвращается `NULL`. В случае успеха возвращается указатель на начало библиотеки.
2. `dldclose(handle)` – убирает ссылку на открытую динамическую библиотеку, переданную в качестве параметра `handle`. Если количество ссылок стало равным нулю, то динамическая библиотека выгружается из памяти
3. `dlsym(handle, sym)` – осуществляет поиск символа, указанного в качестве параметра `sym`, в открытой динамической библиотеке, переданной в качестве параметра `handle`. В частном случае, в роли символа выступает название функции. При успешном выполнении возвращает указатель, соответствующий символу. В случае ошибки возвращается `NULL`

## Общий метод и алгоритм решения

1. Изучить методы работы с динамическими библиотеками
2. Реализация библиотек согласно контрактам, указанным в задании
3. Реализовать программу №1 и №2

## Исходный код

### main1.c

```
#include "gcf/gcf.h"
#include "translation/translation.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
    while (1) {
        int command_id;
        scanf("%d", & command_id);

        if (command_id == 1) {
            int a, b;
            scanf("%d %d", & a, & b);
            printf("GCF(%d, %d) = %d\n", a, b, gcf(a, b));
        } else if (command_id == 2) {
            long x;
            scanf("%ld", & x);
            printf("translation(%ld) = %s\n", x, translation(x));
        }
    }
}
```

```

    } else if (command_id == -1) {
        break;
    } else {
        printf("[Error] UnkownCommandId: Id %d is undefined\n",
command_id);
        exit(3);
    }
}

printf("Program exit...\n");
return 0;
}

```

## main2.c

```

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

int( * gcf)(int, int) = NULL;
char * ( * translation)(long) = NULL;

void * open_dynamic_library(char * filename) {
    void * handle = dlopen(filename, RTLD_NOW);

    if (handle == NULL) {
        printf("[Error] OpenDynamicLibrary: Can't open library '%s'\n",
filename);
        exit(1);
    }

    return handle;
}

void * get_symbol(void * handle, char * symbol) {
    void * symb = dlsym(handle, symbol);

    if (symb == NULL) {
        printf("[Error] GetSymbol: Can't get symbol '%s'\n", symbol);
        exit(2);
    }

    return symb;
}

void close_library(void * handle) {
    dlclose(handle);
}

int main() {

```

```

char * gcf_filenames[] = {
    "./libs/libgcf1.so",
    "./libs/libgcf2.so"
};
char * translation_filenames[] = {
    "./libs/libtranslation1.so",
    "./libs/libtranslation2.so"
};

int active_gcf = 0;
int active_translation = 0;

void * gcf_handle = open_dynamic_library(gcf_filenames[active_gcf]);
void * translation_handle =
open_dynamic_library(translation_filenames[active_translation]);

gcf = get_symbol(gcf_handle, "gcf");
translation = get_symbol(translation_handle, "translation");

while (1) {
    int command_id;
    scanf("%d", & command_id);

    if (command_id == 0) {
        close_library(gcf_handle);
        close_library(translation_handle);

        active_gcf ^= 1;
        active_translation ^= 1;

        gcf_handle = open_dynamic_library(gcf_filenames[active_gcf]);
        translation_handle =
open_dynamic_library(translation_filenames[active_translation]);
    } else if (command_id == 1) {
        int a, b;
        scanf("%d %d", & a, & b);
        printf("Result: GCF(%d, %d) = %d\n", a, b, gcf(a, b));
    } else if (command_id == 2) {
        long x;
        scanf("%ld", & x);
        printf("Result: translation(%ld) = %s\n", x, translation(x));
    } else if (command_id == -1) {
        break;
    } else {
        printf("[Error] UnkownCommandId: Id %d is undefined\n",
command_id);
        exit(3);
    }
}

close_library(gcf_handle);
close_library(translation_handle);

```

```
printf("Program exit...\n");  
return 0;  
}
```

### **gcf/gcf1.c**

```
#include "gcf.h"  
#include <stdio.h>  
  
int gcf(int a, int b)  
{  
    printf("[GCF] Using realization 1...\n");  
  
    while(a != 0 && b != 0)  
    {  
        if(a > b) a %= b;  
        else b %= a;  
    }  
  
    return a + b;  
}
```

### **gcf/gcf2.c**

```
#include "gcf.h"  
#include <stdio.h>  
  
int gcf(int a, int b)  
{  
    printf("[GCF] Using realization 2...\n");  
  
    int m = (a < b) ? a : b;  
  
    while(m > 1)  
    {  
        if(a % m == 0 && b % m == 0)  
            return m;  
  
        m--;  
    }  
  
    return 1;  
}
```

## gcf.gcf.h

```
#ifndef __GCF_H__
#define __GCF_H__

int gcf(int a, int b);

#endif
```

## translation/convert/convert.c

```
#include "convert.h"
#include <stdlib.h>
#include <string.h>

int __get_converted_length(long number, int base) {
    int size;
    for (size = 0; number > 0; number /= base) size++;
    return size;
}

void __reverse(char * str) {
    int n = strlen(str);

    for (int i = 0; i < n / 2; i++) {
        int j = n - i - 1;
        char tmp = str[i];
        str[i] = str[j];
        str[j] = tmp;
    }
}

char * convert_to_base(long x, int base) {
    int i = 0;
    int negative = 0;

    if (x < 0) {
        negative = 1;
        x *= -1;
    }

    int converted_length = __get_converted_length(x, base);
    char * converted = (char * ) malloc(sizeof(char) * converted_length);

    while (x > 0) {
        converted[i] = (x % base) + '0';
        x /= base;
        i++;
    }
}
```



```
    if (negative) {
        converted[i] = '-';
        i++;
    }

    converted[i] = '\0';
    __reverse(converted);

    return converted;
}
```

### **translation/convert/convert.h**

```
#ifndef __CONVERT_H__
#define __CONVERT_H__

char *convert_to_base(long x, int base);

#endif
```

### **translation/translation1.c**

```
#include "translation.h"
#include "convert/convert.h"

char *translation(long x)
{
    return convert_to_base(x, 2);
}
```

### **translation/translation2.c**

```
#include "convert/convert.h"
#include "translation.h"

char *translation(long x)
{
    return convert_to_base(x, 3);
}
```

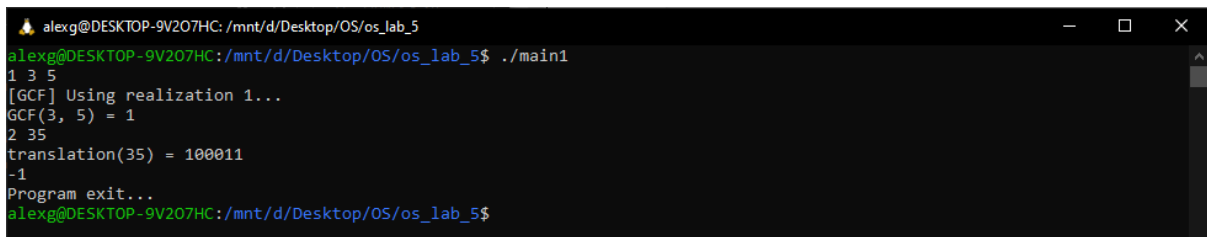
## translation/translation.h

```
#ifndef __TRANSLATION_H__
#define __TRANSLATION_H__

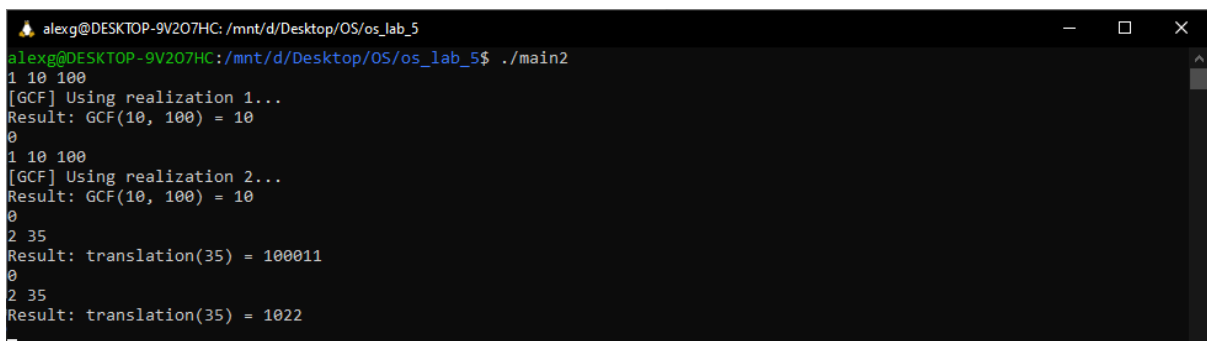
char *translation(long x);

#endif
```

## Пример работы



```
alexg@DESKTOP-9V207HC: /mnt/d/Desktop/OS/os_lab_5
alexg@DESKTOP-9V207HC:/mnt/d/Desktop/OS/os_lab_5$ ./main1
1 3 5
[GCF] Using realization 1...
GCF(3, 5) = 1
2 35
translation(35) = 100011
-1
Program exit...
alexg@DESKTOP-9V207HC:/mnt/d/Desktop/OS/os_lab_5$
```



```
alexg@DESKTOP-9V207HC: /mnt/d/Desktop/OS/os_lab_5
alexg@DESKTOP-9V207HC:/mnt/d/Desktop/OS/os_lab_5$ ./main2
1 10 100
[GCF] Using realization 1...
Result: GCF(10, 100) = 10
0
1 10 100
[GCF] Using realization 2...
Result: GCF(10, 100) = 10
0
2 35
Result: translation(35) = 100011
0
2 35
Result: translation(35) = 1022
```

## Вывод

В ходе выполнения данной лабораторной работы я ознакомился с созданием динамических библиотек, ознакомился с особенностями работы с ними как в операционной системе Linux, так и в Windows. Ознакомился с подключением динамических библиотек на этапе линковки и во время выполнения программы с помощью системных вызовов операционной системы.