Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №4 по курсу

«Операционные системы»

Группа: М8О-211БВ-24

Студент: Бабицкий И.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.11.25

Москва, 2025

Постановка задачи

Вариант 17.

Требуется создать динамические библиотеки, которые реализуют заданный вариантом функционал.

Далее использовать данные библиотеки 2 способами:

- Во время компиляции (на этапе линковки/**linking**);
- 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 используют одну из библиотек, используя информацию полученные на этапе компиляции;
- Тестовая программа (программа №2), которая загружает библиотеки, используя только их относительные пути и контракты.

Провести анализ двух типов использования библиотек.

Пользовательский ввод для обоих программ должен быть организован следующим образом:

- Если пользователь вводит команду `"0"`, то программа переключает одну реализацию контрактов на другую (необходимо только для программы №2). Можно реализовать лабораторную работу без данной функции, но максимальная оценка в этом случае будет «хорошо»;
- `"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;
- `"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.

Контракты и реализации функций:

- №3. Подсчёт количества простых чисел на отрезке `[a, b]` (`a`, `b` – натуральные):
  - Сигнатура функции: `int prime_count(int a, int b);`
  - Реализация №1: Наивный алгоритм. Проверить делимость текущего числа на все предыдущие числа;
  - Реализация №2: Решето Эратосфена.
- №5. Расчет значения числа π при заданной длине ряда (`k`):
  - Сигнатура функции: `float pi(int k);`
  - Реализация №1: Ряд Лейбница;
  - Реализация №2: Формула Валлиса.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `void *dlopen(const char *filename, int flags);` – загружает динамическую библиотеку в память и открывает её;
- `void *dlsym(void *handle, const char *symbol);` – возвращает адрес функции или переменной из загруженной динамической библиотеки;
- `int dlclose(void *handle);` – выгружает из памяти ранее загруженную динамическую библиотеку.

Я составил программу на языке **C** для реализации библиотеки с двумя контрактами двух функций, которую можно подключать к программе статически или динамически.

Код программы

contracts.h

```
#ifndef CONTRACTS_H
#define CONTRACTS_H

int prime_count(int a, int b);
float pi(int k);

#endif
```

simpleBool.h

```
#ifndef SIMPLE_BOOL_H
#define SIMPLE_BOOL_H

#define TRUE  1
#define FALSE 0

#endif
```

utilities.h

```
#ifndef UTILITIES_H
#define UTILITIES_H

typedef enum {
  OK,
  NO_COMMAND,
  INVALID_MODE,
  INVALID_ARGS_COUNT
} commandType;

commandType checkCommand(int mode, int argsCount);

void printMessage(const char *message);
void printError(const char *error);

#endif
```

lib1.c

```
#include "../../include/contracts.h"
#include "../../include/simpleBool.h"

int is_prime(int number) {
  if (number == 1) {
    return FALSE;
  }
  for (int d = 2; d * d <= number; ++d) {
    if (number % d == 0) {
      return FALSE;
    }
  }
  return TRUE;
}

int prime_count(int a, int b) {
  if (b == 1 || a > b) {
    return 0;
  }

  int count = 0;
  for (int number = a; number <= b; ++number) {
    if (is_prime(number)) {
      ++count;
    }
  }
  return count;
}

float pi(int k) {
  float sum = 0.0;
  for (int i = 0; i != k; ++i) {
    sum += (i % 2 == 0 ? 1.0 : -1.0) / (2.0 * i + 1.0);
  }
  return sum * 4.0;
}
```

lib2.c

```
#include "../../include/contracts.h"
#include "../../include/simpleBool.h"
#include <stdlib.h>

int prime_count(int a, int b) {
  if (b == 1 || a > b) {
    return 0;
  }
```

```
  char *is_prime = malloc(sizeof(char) * (b + 1));
  is_prime[0] = FALSE, is_prime[1] = FALSE;
  for (size_t i = 2; i <= b; ++i) {
    is_prime[i] = TRUE;
  }

  for (int d = 2; d * d <= b; ++d) {
    if (is_prime[d]) {
      for (int _d = d * d; _d <= b; _d += d) {
        is_prime[_d] = FALSE;
      }
    }
  }

  int count = 0;
  for (int d = a; d <= b; ++d) {
    if (is_prime[d]) {
      ++count;
    }
  }

  free(is_prime);
  return count;
}

float pi(int k) {
  float composition = 1.0;
  for (int i = 1; i <= k; ++i) {
    composition *= (4.0 * i * i) / (4.0 * i * i - 1.0);
  }
  return composition * 2.0;
}
```

utilities.c

```
#include "../../include/utilities.h"
#include <unistd.h>
#include <string.h>

commandType checkCommand(int mode, int argsCount) {
  if (argsCount == 0) {
    return NO_COMMAND;
  }
  if (mode < 0 || mode > 2) {
    return INVALID_MODE;
  }
  int correctArgsCount[3] = {1, 3, 2};
  if (argsCount != correctArgsCount[mode]) {
    return INVALID_ARGS_COUNT;
  }
```

```
  return OK;
}


void printMessage(const char *message) {
  write(STDOUT_FILENO, message, strlen(message));
}

void printError(const char *error) {
  write(STDERR_FILENO, error, strlen(error));
}
```

static.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "../include/contracts.h"
#include "../include/utilities.h"

#define BUFFER_SIZE 128

int main() {
  printMessage(
    "1 <start> <end> | calculate count of prime numbers in range [start; end]\n"
    "2 <accuracy>    | calculate PI\n"
  );

  char buffer[BUFFER_SIZE];
  int mode = 0;
  int arg1 = 0;
  int arg2 = 0;
  ssize_t len;
  int argsCount;

  printMessage("Choose a mode: ");
  while ((len = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1)) > 0) {
    buffer[len] = '\0';
    argsCount = sscanf(buffer, "%d%d%d", &mode, &arg1, &arg2);

    switch(checkCommand(mode, argsCount)) {
      case NO_COMMAND: printError("Invalid input\n"); break;
      case INVALID_MODE: printError("Invalid mode\n"); break;
      case INVALID_ARGS_COUNT: printError("Invalid count of arguments\n"); break;
      case OK:
        switch (mode) {
          case 1:
            snprintf(buffer, BUFFER_SIZE, "Result: %d\n", prime_count(arg1, arg2));
            write(STDOUT_FILENO, buffer, strlen(buffer));
```

```
            break;
          case 2:
            snprintf(buffer, BUFFER_SIZE, "Result %f\n", pi(arg1));
            write(STDOUT_FILENO, buffer, strlen(buffer));
            break;
        }
    }

    printMessage("Choose a mode: ");
  }

  return 0;
}
```

dynamic.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dlfcn.h>
#include "../include/contracts.h"
#include "../include/utilities.h"

#define BUFFER_SIZE 128

typedef int prime_count_func(int, int);
typedef float pi_func(int);

static prime_count_func *prime_count_ptr = NULL;
static pi_func *pi_ptr = NULL;
static void *library = NULL;

static int current_implementation = 1;

static void load_library() {
  if (library) {
    dlclose(library);
  }

  const char *libraryName = (current_implementation == 1) ? "./lib1.so" : "./lib2.so";
  library = dlopen(libraryName, RTLD_LOCAL | RTLD_NOW);
  if (library == NULL) {
    printError("Can't load the library\n");
    exit(EXIT_FAILURE);
  }

  prime_count_ptr = dlsym(library, "prime_count");
  if (prime_count_ptr == NULL) {
    printError("Can't find the function \"prime_count\"\n");
```

```
    exit(EXIT_FAILURE);
  }

  pi_ptr = dlsym(library, "pi");
  if (pi_ptr == NULL) {
    printError("Can't find the function \"pi\"\n");
    exit(EXIT_FAILURE);
  }
}

int main() {
  printMessage(
    "0               | switch implementation of functions\n"
    "1 <start> <end> | calculate count of prime numbers in range [start; end]\n"
    "2 <accuracy>    | calculate PI\n"
  );

  load_library();

  char buffer[BUFFER_SIZE];
  int mode = 0, arg1 = 0, arg2 = 0, argsCount;
  ssize_t len;

  printMessage("Choose a mode: ");
  while ((len = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1)) > 0) {
    buffer[len] = '\0';
    argsCount = sscanf(buffer, "%d%d%d", &mode, &arg1, &arg2);

    switch(checkCommand(mode, argsCount)) {
      case NO_COMMAND: printError("Invalid input\n"); break;
      case INVALID_MODE: printError("Invalid mode\n"); break;
      case INVALID_ARGS_COUNT: printError("Invalid count of arguments\n"); break;
      case OK:
        switch (mode) {
          case 0:
            current_implementation = (current_implementation == 1) ? 2 : 1;
            load_library();
            snprintf(buffer, BUFFER_SIZE, "Implementation switched to library #%d\n",
                     current_implementation);
            printMessage(buffer);
            break;
          case 1:
            snprintf(buffer, BUFFER_SIZE, "Result from library #%d: %d\n",
                     current_implementation, prime_count_ptr(arg1, arg2));
            printMessage(buffer);
            break;
          case 2:
            snprintf(buffer, BUFFER_SIZE, "Result from library #%d: %f\n",
                     current_implementation, pi_ptr(arg1));
```

```
            printMessage(buffer);
            break;
        }
    }

    printMessage("Choose a mode: ");
  }

  if (library) {
    dlclose(library);
  }

  return 0;
}
```

Протокол работы программы

```
$ ./staticLib1.out
1 <start> <end> | calculate count of prime numbers in range [start; end]
2 <accuracy>    | calculate PI
Choose a mode: 1 2 20
Result: 8
Choose a mode: 2 100
Result 3.131593
$ ./staticLib2.out
1 <start> <end> | calculate count of prime numbers in range [start; end]
2 <accuracy>    | calculate PI
Choose a mode: 1 2 20
Result: 8
Choose a mode: 2 100
Result 3.133787
$ ./dynamic.out
0               | switch implementation of functions
1 <start> <end> | calculate count of prime numbers in range [start; end]
2 <accuracy>    | calculate pi
Choose a mode: 1 2 20
Result from library #1: 8
Choose a mode: 2 100
Result from library #1: 3.131593
Choose a mode: 0
Implementation switched to library #2
Choose a mode: 1 2 20
Result from library #2: 8
Choose a mode: 2 100
Result from library #2: 3.133787
Choose a mode: 0
Implementation switched to library #1
Choose a mode: 0
Implementation switched to library #2
```

Вывод

В ходе выполнения лабораторной работы я изучил на практике создание динамических библиотек и их использование в собственных программах, а также алгоритмы нахождения простых чисел при помощи решето Эратосфена и числа $\boldsymbol{\pi}$ при помощи ряда Лейбница и формулы Валлиса.