Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

Студент: Шумилова Александра

Группа: М8О-207Б-21

Вариант: 21

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/tonsoleils/OS

**Постановка задачи**

**Цель работы**

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:

* 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 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 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 её 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 её выполнения.

**Общие сведения о программе**

Программа компилируется из файла main1.c, main2.c, functions1.c и fuinctions2.c с использованием заголовочного файла functions.h с помощью make и Makefile. Также используется заголовочные файлы: stdio.h, stdlib.h, dlfcn.h.

**Общий метод и алгоритм решения**

В файле functions.h находятся объявления функций подсчёта простых чисел и сортировки. В файле functions1.c находится реализация подсчёта простых чисел наивным алгоритмом и пузырьковой сортировки. В файле functions2.c находится реализация подсчёта простых числе с помощью решета Эратосфена и быстрая сортировка (сортировка Хоара).

В файле main1.c находится пример использвания библиотеки которая используют её, используя знания полученные на этапе компиляции. В файле main2.c находится пример использования библиотеки, которая загружает библиотеки, используя только их местоположение и контракты.

В файле Makefile описана стратегия компиляции в библиотеки: для main1.c используем первую библитеку с использованием знаний, полученных на этапе компиляции. Для файла main2.c компилируем библиотеки для их дальнейшей загрузки в рантайме.

**Исходный код**

**functions.h**

|  |
| --- |
| #pragma once  int PrimeCount(int a, int b); // объявление функции поиска простых чисел  int\* Sort(int\* array); // объявление функции сортировки |

**functions1.c**

|  |
| --- |
| #include "functions.h"  // Подсчёт количества простых чисел на отрезке [A, B] (A, B - натуральные)  // Наивный алгоритм  int PrimeCount(int a, int b) {  int count = 0; // объявляем счётчик простых чисел  for (int i = a; i <= b; i++) { // Идём по всем целым числам от А до Б  int flag = 1; // флаг, говорящий нам о том, простое ли данное число  if (i == 1) { // Игнорируем единицу  flag = 0;  }  for (int j = i - 1; j >= a; j--) { // идём по всем числам левее от текущего  if (i % j == 0 && j != 1 && j != 0) { // если они делятся без остатка, и делитель - не 1 и делитель не 0  flag = 0; // оно не простое  }  }  if (flag == 1) {  count++; // прибавляем счётчик если флаг не изменился  }  }  return count;  }  // Пузырьковая сортировка  int\* Sort(int\* array) {  // Для всех элементов  int size = array[0]; // получаем размер из нулевого элемента массива  for (int i = 1; i < size; i++) { // берем число  for (int j = size; j > i; j--) { // и сравниваем его с остальными  if (array[j - 1] > array[j]) { // если больше  int temp = array[j - 1]; // меняем их местами  array[j - 1] = array[j];  array[j] = temp;  }  }  }  return array;  } |

**functions2.c**

|  |
| --- |
| #include "functions.h"  #include <stdlib.h>  // Подсчёт количества простых чисел на отрезке [A, B] (A, B - натуральные)  // Решето эратосфена  int PrimeCount(int a, int b) {  int count = 0; // счётчик  int \*arr = malloc(sizeof(int) \* (b + 1)); // выделяем массив  for (int i = a; i <= b; i++) {  arr[i] = i; // заполняем числами от А до Б  }  // берем первое число, удаляем все числа от p^2 до B + 1 с шагом p (p^2, p^2 + p, p^2 + 2p и т.д.)  // далее находим первое ненулевое число и повторяем снова  for (int p = 2; p < b + 1; p++) {  if (arr[p] != 0) {  count++;  for (int j = p\*p; j < b + 1; j += p) {  arr[j] = 0;  }  }  }  return count;  }  void quicksort(int \*array, int first, int last);  // Сортировка Хоара  int\* Sort(int\* array) {  // Для всех элементов  int size = array[0];  quicksort(array, 1, size);  return array;  }  void quicksort(int \*array, int first, int last) {  int mid, count;  int f = first;  int l = last;  mid = array[(f + l) / 2]; // Выбираем опорный элемент  // Перераспределяем элементы в массиве таким образом,  // что элементы, меньшие опорного, помещаются перед ним, а большие или равные - после.  do {  while (array[f] < mid) f++;  while (array[l] > mid) l--;  if (f <= l) {  count = array[f];  array[f] = array[l];  array[l] = count;  f++;  l--;  }  } while (f < l);  // Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента.  // Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.  if (first < l) quicksort(array, first, l);  if (f < last) quicksort(array, f, last);  } |

**main1.c**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include "functions.h"  int main() {  int choice; // переменная выбора  printf("Enter type of function (1 - PrimeCount, 2 - Sort): ");  scanf("%d", &choice); // считываем ее  if (choice == 1) {  printf("A = ");  int a;  scanf("%d", &a);  printf("B = ");  int b;  scanf("%d", &b);  printf("Prime count = %d\n", PrimeCount(a, b)); // запускаем и выводим результат работы  } else if (choice == 2) {  printf("Size = ");  int size;  scanf("%d", &size);  int\* array = malloc(sizeof(int) \* (size + 1)); // подготавливаем массив  array[0] = size; // прячем на первое место его размер  for (int i = 1; i <= size; i++) {  int temp;  scanf("%d", &temp); // считывем элементы массива  array[i] = temp;  }  int\* result = Sort(array); // сортируем  for (int i = 1; i <= size; i++) {  printf("%d ", result[i]); // выводим результат  }  printf("\n");  } else {  return 0;  }  return 0;  } |

**main2.c**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <dlfcn.h>  #include "functions.h"  int main() {  int choice;  int (\*PrimeCount)(int A, int B);  int\* (\*Sort)(int\* array);  char\* libs[] = {"libfunc1.so", "libfunc2.so"};  int lib = 0;  void\* handler = NULL;  while (printf("Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): ") && scanf("%d", &choice) > 0) {  if (choice == 0) {  if (handler) {  // dlclose(handle) - уменьшает на единицу счетчик ссылок на указатель динамической библиотеки handle.  dlclose(handler);  }  lib = 1 - lib; // переключатель  printf("Implementation changed!\n");  } else {  // dlopen(filename, flag) - dlopen загружает динамическую библиотеку,  // имя которой указано в строке filename, и возвращает прямой указатель  // на начало динамической библиотеки.  // flag RTLD\_LAZY, подразумевает разрешение неопределенных символов в виде кода,  // содержащегося в исполняемой динамической библиотеке  handler = dlopen(libs[lib], RTLD\_LAZY);  if (!handler) { // если произошла ошибка  fprintf(stderr, "dlopen() error: %s\n", dlerror()); // выводим её  exit(1); // и выходим  }  // Функции dlsym(handler, symbol) передаётся handler динамически загруженного объекта,  // возвращаемого dlopen и имя символа (с null в конце). В результате функция  // возвращает адрес, по которому символ расположен в памяти.  PrimeCount = dlsym(handler, "PrimeCount");  Sort = dlsym(handler, "Sort");  if (choice == 1) {  printf("A = ");  int a;  scanf("%d", &a);  printf("B = ");  int b;  scanf("%d", &b);  printf("Prime count = %d\n", PrimeCount(a, b));  } else if (choice == 2) {  printf("Size = ");  int size;  scanf("%d", &size);  int\* array = malloc(sizeof(int) \* (size + 1));  array[0] = size;  for (int i = 1; i <= size; i++) {  int temp;  scanf("%d", &temp);  array[i] = temp;  }  int\* result = Sort(array);  for (int i = 1; i <= size; i++) {  printf("%d ", result[i]);  }  printf("\n");  } else {  printf("Exit\n");  dlclose(handler);  break;  }  }  }  } |

**Makefile**

|  |
| --- |
| CC = gcc  CFLAGS = -g -c -Wall  all: main1 main2  main1: libfunc1.so src/main1.c  $(CC) src/main1.c -L. -lfunc1 -o main1  main2: libfunc1.so libfunc2.so src/main2.c  $(CC) src/main2.c -ldl -o main2  libfunc1.so: functions1.o  $(CC) -shared functions1.o -o libfunc1.so  libfunc2.so: functions2.o  $(CC) -shared functions2.o -o libfunc2.so  functions1.o: src/functions1.c  $(CC) -fPIC -c src/functions1.c -o functions1.o  functions2.o: src/functions2.c  $(CC) -fPIC -c src/functions2.c -o functions2.o  clean:  rm -r \*.so \*.o main1 main2 |

**Демонстрация работы программы**

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/op\_sys/os/lab5$** ./main2

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): 1

A = 1

B = 100

Prime count = 25

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): 0

Implementation changed!

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): 1

A = 1

B = 100

Prime count = 25

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): 2

Size = 5

4

2

6

4

3

2 3 4 4 6

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): 0

Implementation changed!

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): 2

Size = 5

4

2

6

4

3

2 3 4 4 6

Enter type of funtion (0 - Change implementation, 1 - PrimeCount, 2 - Sort): -1

Exit

**Выводы**

Проделав данную работу, я научилась принципам создание динамических библиотек создание программ, которые используют функции динамических библиотек. Научилась компилировать библиотеки и использовать их как на основе знаний с этапа компиляций, так и подгружать их во время работы программы.