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»

Выполнил: Д. Н. Дружинин
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы:

- Приобретение практических навыков разработки динамических библиотек.
- Освоение методов использования динамически подключаемых библиотек в программах на C/C++ двумя способами: на этапе компиляции (linking) и на этапе выполнения программы (dynamic loading).
- Закрепление навыков отделения интерфейса от реализации, организации кроссплатформенных решений и использования обёрток системных вызовов.

Задание:

1. Реализовать набор динамических библиотек согласно выбранному варианту: каждая функция должна иметь две реализации (v1 и v2).
2. Реализовать программу №1, которая использует одну реализацию библиотек, подключённых на этапе компиляции.
3. Реализовать программу №2, которая загружает динамические библиотеки в момент выполнения, используя только относительные пути и контракты (заголовочные файлы). Обеспечить переключение между реализациями v1 и v2 по команде пользователя.
4. Все системные вызовы загрузки библиотек должны быть инкапсулированы в кроссплатформенную обёртку `os_wrapper`.
5. Входные данные для обеих программ должны вводиться пользователем в соответствии с форматом задания для вариантов.

Вариант: 10

Функция 1: Производная функции $\cos(x)$ в точке A с приращением Δx.

Функция 2: НОД двух натуральных чисел A и B.

Метод решения

В данной работе требуется реализовать две программы, использующие динамические библиотеки двумя различными способами: на этапе компиляции (статическое связывание с конкретной библиотекой) и на этапе выполнения программы (динамическая загрузка библиотек с помощью системных вызовов).

Для решения задачи была применена следующая архитектура:

1. **Контракты функций.** Интерфейсы всех функций собраны в заголовочном файле `contracts.h`. Это позволяет отделить объявление функций от их реализации и обеспечить совместимость между библиотеками `v1` и `v2`. Используется конструкция `extern "C"` для предотвращения искажения имён функций (name mangling) в C++.
2. **Две реализации функций.** Для каждой функции были созданы две независимые динамические библиотеки:

- версия `v1` использует один алгоритм решения;
- версия `v2` использует альтернативный алгоритм.

Для производной функции реализованы методы односторонней (v1) и центральной (v2) разности. Для НОД реализован алгоритм Евклида (v1) и метод перебора делителей (v2).

3. **Кроссплатформенная обёртка системных вызовов.** Все операции загрузки библиотек и получения адресов функций вынесены в модуль `os_wrapper`. На Linux используются вызовы:
   - `dlopen()`;
   - `dlsym()`;
   - `dlclose()`.

   На Windows применяются:
   - `LoadLibrary()`;
   - `GetProcAddress()`;
   - `FreeLibrary()`.

   Обёртка скрывает различия между операционными системами и предоставляет единый API.

4. **Программа №1 (компиляционное связывание).** В этой программе требуемые динамические библиотеки подключаются на этапе сборки проекта с помощью директивы `target_link_libraries`. Выбор реализации (v1 или v2) фиксирован в `CMakeLists.txt`.

5. **Программа №2 (динамическая загрузка).** Эта программа загружает необходимые библиотеки во время выполнения при помощи функций `os_wrapper`. Пользователь может переключать реализацию командой `0`, и программа повторно загружает библиотеки `v1` или `v2` без необходимости пересборки.

Результаты

В ходе выполнения лабораторной работы были созданы две независимые программы, демонстрирующие различные способы использования динамических библиотек в операционных системах.

1. Реализованные динамические библиотеки

Для каждой из двух функций были разработаны по две реализации (v1 и v2), что позволяет сравнивать различные алгоритмические подходы:

- функция вычисления производной $\cos(x)$:
  - v1 — метод односторонней разности;
  - v2 — метод центральной разности (более точный).

- функция вычисления НОД двух натуральных чисел:
  - v1 — классический алгоритм Евклида;
  - v2 — перебор делителей.

Все реализации собраны в виде динамических библиотек (`.so` на Linux и `.dll` на Windows), с корректной экспортируемой таблицей символов и единым интерфейсом, заданным контрактами функции.

2. Программа №1 (compile-time linking)

Была реализована программа, использующая функции из динамических библиотек, подключённых на этапе компиляции. Реализация функций выбирается в `CMakeLists.txt` и не может быть изменена пользователем в процессе работы.

Особенности:

- фиксированный выбор реализации;
- подключение библиотек через `target_link_libraries`;
- работа функций полностью аналогична обычному статическому связыванию.

Программа корректно принимает команды пользователя, вычисляет производную и НОД, демонстрируя работу выбранной реализации.

3. Программа №2 (dynamic loading)

Вторая программа позволяет переключать реализацию библиотек в момент выполнения, используя только относительные пути и системные вызовы загрузки динамических библиотек. Подгрузка производится через созданный модуль `os_wrapper`, который скрывает различия между Linux и Windows.

Ключевые особенности:

- поддержка горячего переключения реализаций командой `0`;
- загрузка библиотек функцией `os_load_library`;
- получение адресов экспортируемых функций через `os_load_symbol`;
- корректное освобождение ресурсов.

Пользователь может выполнять вычисления в любой из реализаций без пересборки программы. Отличия между результатами v1 и v2 особенно заметны при вычислении производной: метод центральной разности (v2) показывает более точное значение.

4. Кроссплатформенность

Благодаря модулю `os_wrapper` проект успешно выполняется как на Linux, так и на Windows. В зависимости от платформы используются:

- Linux: `dlopen`, `dlsym`, `dlclose`;

- Windows: `LoadLibrary`, `GetProcAddress`, `FreeLibrary`.

Это подтверждает корректность архитектуры и универсальность разработанного решения.

5. Итог

В результате работы была получена полностью функционирующая система, включающая набор динамических библиотек, две тестовые программы, кроссплатформенный слой работы с динамическими модулями и механизм переключения реализаций во время выполнения программы. Реализация полностью соответствует требованиям задания и демонстрирует оба способа использования динамических библиотек.

Выводы

В ходе выполнения лабораторной работы были исследованы и practically реализованы два принципиально различных подхода к использованию динамических библиотек: компиляционное связывание и динамическая загрузка во время выполнения программы.

Были разработаны две независимые реализации каждой функции (v1 и v2), а также две тестовые программы, демонстрирующие работу с библиотеками различными способами. Программа №1 показала использование библиотек, подключаемых на этапе компиляции, что обеспечивает простоту, но не позволяет изменять алгоритмы без пересборки. Программа №2 реализует динамическую загрузку библиотек, давая возможность переключать реализации в процессе работы и тем самым повышая гибкость и расширяемость приложения.

Особое внимание было уделено созданию кроссплатформенного модуля `os_wrapper`, скрывающего различия между системными вызовами Linux и Windows. Это позволило обеспечить переносимость решения и корректную работу программы на обеих платформах.

В результате работы приобретены практические навыки:

- разработки и экспорта динамических библиотек;
- использования механизмов загрузки библиотек во время выполнения;
- организации архитектуры с разделением интерфейса и реализации;
- построения кроссплатформенных приложений с использованием системных вызовов.

Полученное решение полностью соответствует требованиям задания и подтверждает эффективность динамической загрузки как инструмента для построения гибких и расширяемых программных систем.

Исходная программа

Ниже приведены все исходные файлы лабораторной работы, включая контракты, реализации функций, кроссплатформенную оболочку для загрузки динамических библиотек, тестовые программы и файл сборки. В конце раздела добавлена трассировка системных вызовов, подтверждающая корректную работу как компиляционного, так и динамического связывания библиотек.

contracts.h

```
#pragma once

#ifdef _WIN32
#ifdef BUILD_DLL
#define API __declspec(dllexport)
#else
#define API __declspec(dllimport)
#endif
#else
#define API
#endif

extern "C"
{
  API float Derivative(float A, float deltaX);
  API int GCF(int A, int B);
}
```

os_wrapper.h

```
#pragma once
#include <string>

#ifdef _WIN32
#include <windows.h>
typedef HMODULE LibHandle;
#else
#include <dlfcn.h>
typedef void* LibHandle;
#endif

LibHandle os_load_library(const std::string& path);
void* os_load_symbol (LibHandle lib, const std::string& name);
void os_close_library(LibHandle lib);
```

os_wrapper.cpp

```
#include "os_wrapper.h"
#include <iostream>

LibHandle os_load_library(const std::string& path)
{
  #ifdef _WIN32
  return LoadLibraryA(path.c_str());
  #else
  return dlopen(path.c_str(), RTLD_LAZY);
  #endif
}

void* os_load_symbol(LibHandle lib, const std::string& name)
{
```

```
    #ifdef _WIN32
    return reinterpret_cast<void*>(GetProcAddress(lib, name.c_str()));
    #else
    return dlsym(lib, name.c_str());
    #endif
  }

  void os_close_library(LibHandle lib)
  {
    #ifdef _WIN32
    FreeLibrary(lib);
    #else
    dlclose(lib);
    #endif
  }
```

func1_derivative_v1.cpp

```
  #define BUILD_DLL
  #include "contracts.h"
  #include <cmath>

  extern "C" API float Derivative(float A, float deltaX)
  {
    return (cos(A + deltaX) - cos(A)) / deltaX;
  }
```

func1_derivative_v2.cpp

```
  #define BUILD_DLL
  #include "contracts.h"
  #include <cmath>

  extern "C" API float Derivative(float A, float deltaX)
  {
    return (cos(A + deltaX) - cos(A - deltaX)) / (2 * deltaX);
  }
```

func2_gcf_v1.cpp

```
  #define BUILD_DLL
  #include "contracts.h"

  extern "C" API int GCF(int A, int B)
  {
    while (B != 0) {
      int t = A % B;
      A = B;
      B = t;
    }
    return A;
  }
```

func2_gcf_v2.cpp

```
#define BUILD_DLL
#include "contracts.h"

extern "C" API int GCF(int A, int B)
{
  int minv = (A < B ? A : B);
  int g = 1;
  for (int i = 1; i <= minv; i++) {
    if (A % i == 0 && B % i == 0)
    g = i;
  }
  return g;
}
```

main_link.cpp

```
#include <iostream>
#include <sstream>
#include "contracts.h"

int main()
{
  std::cout << "Program #1 (compile-time linking)\n";
  std::cout << "Commands:\n";
  std::cout << " 1 A deltaX - compute Derivative(A, deltaX)\n";
  std::cout << " 2 A B - compute GCF(A, B)\n";
  std::cout << " q - quit\n\n";

  std::string line;
  while (true)
  {
    std::cout << "> ";
    if (!std::getline(std::cin, line)) break;
    if (line == "q") break;

    std::istringstream iss(line);
    int cmd;
    iss >> cmd;

    if (cmd == 1)
    {
      float A, dx;
      if (!(iss >> A >> dx))
      {
        std::cout << "Error: format is 1 A deltaX\n";
        continue;
      }
      std::cout << Derivative(A, dx) << "\n";

    }
    else if (cmd == 2)
    {
      int A, B;
      if (!(iss >> A >> B))
```

```
      {
        std::cout << "Error: format is 2 A B\n";
        continue;
      }
      std::cout << GCF(A, B) << "\n";
    }
    else
    {
      std::cout << "Unknown command\n";
    }
  }

  return 0;
}
```

main_runtime.cpp

```
#include <iostream>
#include <sstream>
#include "os_wrapper.h"
#include "contracts.h"

typedef float (*DerivativeFunc)(float, float);
typedef int (*GCFFunc)(int, int);

DerivativeFunc derivative = nullptr;
GCFFunc gcf = nullptr;

LibHandle lib_deriv = nullptr;
LibHandle lib_gcf = nullptr;

bool load_impl(bool v1)
{
  #ifdef _WIN32
  const char* d = v1 ? "./libfunc1_derivative_v1.dll" : "./
      libfunc1_derivative_v2.dll";
  const char* g = v1 ? "./libfunc2_gcf_v1.dll" : "./libfunc2_gcf_v2.dll";
  #else
  const char* d = v1 ? "./libfunc1_derivative_v1.so" : "./
      libfunc1_derivative_v2.so";
  const char* g = v1 ? "./libfunc2_gcf_v1.so" : "./libfunc2_gcf_v2.so";
  #endif

  if (lib_deriv) os_close_library(lib_deriv);
  if (lib_gcf) os_close_library(lib_gcf);

  lib_deriv = os_load_library(d);
  lib_gcf = os_load_library(g);

  if (!lib_deriv || !lib_gcf)
  {
    std::cerr << "Failed to load libraries\n";
    return false;
  }

  derivative = (DerivativeFunc) os_load_symbol(lib_deriv, "Derivative");
```

```
38      gcf = (GCFFunc) os_load_symbol(lib_gcf, "GCF");
39
40      return derivative && gcf;
41    }
42
43    int main()
44    {
45      std::cout << "Program #2 (dynamic loading)\n";
46      std::cout << "Commands:\n";
47      std::cout << " 0 - switch implementation (v1 <-> v2)\n";
48      std::cout << " 1 A deltaX - compute Derivative(A, deltaX)\n";
49      std::cout << " 2 A B - compute GCF(A, B)\n";
50      std::cout << " q - quit\n\n";
51
52      bool v1 = true;
53      if (!load_impl(v1)) return 1;
54
55      std::string line;
56
57      while (true)
58      {
59        std::cout << "> ";
60        if (!std::getline(std::cin, line)) break;
61        if (line == "q") break;
62
63        std::istringstream iss(line);
64        int cmd;
65        iss >> cmd;
66
67        if (cmd == 0) {
68          v1 = !v1;
69          if (!load_impl(v1))
70          {
71            std::cout << "Error switching implementation\n";
72            return 1;
73          }
74          std::cout << "Switched to implementation: " << (v1 ? "v1" : "v2") << "\n"
              ;
75
76        } else if (cmd == 1) {
77          float A, dx;
78          if (!(iss >> A >> dx))
79          {
80            std::cout << "Error: format is 1 A deltaX\n";
81            continue;
82          }
83          std::cout << derivative(A, dx) << "\n";
84
85        } else if (cmd == 2) {
86          int A, B;
87          if (!(iss >> A >> B))
88          {
89            std::cout << "Error: format is 2 A B\n";
90            continue;
91          }
92          std::cout << gcf(A, B) << "\n";
93
94        } else {
```

```
95          std::cout << "Unknown command\n";
96        }
97      }
98
99      return 0;
100   }
```

CMakeLists.txt

```
1   cmake_minimum_required(VERSION 3.10)
2   project(Lab4_Var10 CXX)
3
4   set(CMAKE_CXX_STANDARD 17)
5
6   set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
7   set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
8
9   include_directories(include)
10
11  add_library(func1_derivative_v1 SHARED src/func1_derivative_v1.cpp)
12  add_library(func1_derivative_v2 SHARED src/func1_derivative_v2.cpp)
13  add_gallery(func2_gcf_v1 SHARED src/func2_gcf_v1.cpp)
14  add_library(func2_gcf_v2 SHARED src/func2_gcf_v2.cpp)
15
16  add_executable(main_link src/main_link.cpp)
17  target_link_libraries(main_link PRIVATE func1_derivative_v1 func2_gcf_v1)
18
19  add_executable(main_runtime
20  src/main_runtime.cpp
21  src/os_wrapper.cpp
22  )
23
24  if(UNIX AND NOT APPLE)
25  target_link_libraries(main_runtime PRIVATE dl)
26  endif()
```

strace (Program #1)

```
1   execve("./main_link", ["./main_link"], ...) = 0
2   openat(... "libfunc1_derivative_v1.so", ...) = 3
3   mmap(... libfunc1_derivative_v1.so ...)
4   openat(... "libfunc2_gcf_v1.so", ...) = 3
5   mmap(... libfunc2_gcf_v1.so ...)
6   write(1, "> ", 2)
7   read(0, "q\n", 1024)
8   exit_group(0)
9   +++ exited with 0 +++
```

strace (Program #2)

```
execve("./main_runtime", ["./main_runtime"], ...) = 0
write(1, "Program #2 (dynamic loading)\n", ...)
openat("./libfunc1_derivative_v1.so", O_RDONLY) = 3
mmap(... libfunc1_derivative_v1.so ...)
openat("./libfunc2_gcf_v1.so", O_RDONLY) = 3
mmap(... libfunc2_gcf_v1.so ...)
write(1, "> ", 2)
read(0, "q\n", 1024)
exit_group(0)
+++ exited with 0 +++
```