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-214Б-24

Студент: Василянская А.Н.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 17.12.25

Москва, 2025

Постановка задачи

Вариант 30.

Требуется создать динамические библиотеки, которые реализуют заданный вариантом 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 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.

5. Расчет значения числа π при заданной длине ряда (k):

Сигнатура функции: float pi(int k);

- Реализация №1: Ряд Лейбница
- Реализация №2: Формула Валлиса

9. Отсортировать целочисленный массив:

Сигнатура функции: int *sort(int *array, size_t n);

- Реализация №1: Пузырьковая сортировка :^)
- Реализация №2: Сортировка Хоара (за использование qsort из libc или std::sort и его варианты из libstdc++ будет бан; реализуйте его самостоятельно)

Общий метод и алгоритм решения

Разрабатываются две динамические библиотеки с идентичным интерфейсом, но разными алгоритмами расчёта числа π (ряд Лейбница или формула Валлиса) и сортировки массива (пузырьковая или быстрая сортировка Хоара). Первая программа линкуется с библиотекой на этапе компиляции и вызывает функции напрямую, а вторая использует API dlopen/dlsym для явной загрузки .so-файла в память во время исполнения. Это позволяет второй программе менять реализацию «на лету»: по команде пользователя она выгружает текущую библиотеку через dlclose, загружает альтернативную и обновляет указатели на функции для выполнения новых расчётов.

Использованные системные вызовы:

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

Код программы

functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <stdlib.h>

float pi(int k);
int* sort(int* array, size_t n);

#endif
```

libfunc1.c

```
#include <math.h>
#include <stdlib.h>
#include <string.h>

float pi(int k) {
    if (k <= 0) return 0.0f;

    float result = 0.0f;
    for (int i = 0; i < k; i++) {
        float term = 1.0f / (2 * i + 1);
        result += (i % 2 == 0) ? term : -term;
    }
    return 4 * result;
}

int* sort(int* array, size_t n) {
    if (n == 0 || array == NULL) return NULL;

    int* result = malloc(n * sizeof(int));
    if (result == NULL) return NULL;

    memcpy(result, array, n * sizeof(int));
```

```
    for (size_t i = 0; i < n - 1; i++) {
        for (size_t j = 0; j < n - i - 1; j++) {
            if (result[j] > result[j + 1]) {
                int temp = result[j];
                result[j] = result[j + 1];
                result[j + 1] = temp;
            }
        }
    }

    return result;
}
```

libfunc2.c

```
#include <math.h>
#include <stdlib.h>
#include <string.h>

float pi(int k) {
    if (k <= 0) return 0.0f;

    float result = 1.0f;
    for (int i = 1; i <= k; i++) {
        float numerator = 4.0f * i * i;
        float denominator = (2.0f * i - 1.0f) * (2.0f * i + 1.0f);
        result *= numerator / denominator;
    }
    return 2.0f * result;
}

static void quick_sort_recursive(int* arr, int low, int high) {
    if (low < high) {
        int pivot = arr[(low + high) / 2];
        int i = low - 1;
```

```
        int j = high + 1;

        while (1) {
            do i++; while (arr[i] < pivot);
            do j--; while (arr[j] > pivot);
            if (i >= j) break;

            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }

        quick_sort_recursive(arr, low, j);
        quick_sort_recursive(arr, j + 1, high);
    }
}

int* sort(int* array, size_t n) {
    if (n == 0 || array == NULL) return NULL;

    int* result = malloc(n * sizeof(int));
    if (result == NULL) return NULL;

    memcpy(result, array, n * sizeof(int));
    quick_sort_recursive(result, 0, n - 1);

    return result;
}
```

prog1.c

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>
#include "../include/functions.h"
```

```
#define BUFFER_SIZE 4096
#define OUTPUT_SIZE 1024

static void write_string(const char* str) {
    write(STDOUT_FILENO, str, strlen(str));
}

static void write_error(const char* str) {
    write(STDERR_FILENO, str, strlen(str));
}

static int parse_int(const char* str, int* value) {
    if (!str || !*str) return 0;

    *value = 0;
    int sign = 1;
    int i = 0;

    if (str[0] == '-') {
        sign = -1;
        i = 1;
    }

    while (str[i] >= '0' && str[i] <= '9') {
        *value = *value * 10 + (str[i] - '0');
        i++;
    }

    *value *= sign;
    return 1;
}

static void command_1(char* arg) {
```

```
    char buffer[OUTPUT_SIZE];
    int k;

    if (!parse_int(arg, &k)) {
        write_error("error: invalid argument for command 1\n");
        return;
    }

    float result = pi(k);
    int len = snprintf(buffer, OUTPUT_SIZE, "pi(%d) = %.6f\n", k, result);
    write(STDOUT_FILENO, buffer, len);
}

static void command_2(int n, char** args) {
    char buffer[OUTPUT_SIZE];

    if (n <= 0) {
        write_error("error: invalid array size\n");
        return;
    }

    int* array = malloc(n * sizeof(int));
    if (!array) {
        write_error("error: memory allocation failed\n");
        return;
    }

    for (int i = 0; i < n; i++) {
        if (!parse_int(args[i], &array[i])) {
            write_error("error: invalid array element\n");
            free(array);
            return;
        }
    }
```

```
    int* sorted = sort(array, n);
    if (!sorted) {
        write_error("error: sorting failed\n");
        free(array);
        return;
    }

    int len = snprintf(buffer, OUTPUT_SIZE, "sorted: ");
    write(STDOUT_FILENO, buffer, len);

    for (int i = 0; i < n; i++) {
        len = snprintf(buffer, OUTPUT_SIZE, "%d ", sorted[i]);
        write(STDOUT_FILENO, buffer, len);
    }
    write_string("\n");

    free(sorted);
    free(array);
}

int main() {
    char buffer[BUFFER_SIZE];
    const char* prompt = "> ";

    write_string("Static program (libfunc1.so)\n");
    write_string("Commands: 1 k - calculate pi(k)\n");
    write_string("          2 n a1 a2 ... an - sort array\n");
    write_string("          exit - exit program\n");
    write_string(prompt);

    while (1) {
        int bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1);
        if (bytes_read <= 0) break;
```

```
buffer[bytes_read] = '\0';

// Remove newline
if (buffer[bytes_read - 1] == '\n') {
    buffer[bytes_read - 1] = '\0';
    bytes_read--;
}

if (strcmp(buffer, "exit") == 0) {
    break;
}

// Skip empty lines
if (bytes_read == 0) {
    write_string(prompt);
    continue;
}

// Parse command
char* token = strtok(buffer, " ");
if (!token) {
    write_string(prompt);
    continue;
}

if (strcmp(token, "1") == 0) {
    char* arg = strtok(NULL, " ");
    command_1(arg);
}
else if (strcmp(token, "2") == 0) {
    char* size_str = strtok(NULL, " ");
    int n;
```

```
        if (!size_str || !parse_int(size_str, &n)) {
            write_error("error: invalid array size\n");
            write_string(prompt);
            continue;
        }

        char** args = malloc(n * sizeof(char*));
        if (!args) {
            write_error("error: memory allocation failed\n");
            write_string(prompt);
            continue;
        }

        int valid = 1;
        for (int i = 0; i < n; i++) {
            args[i] = strtok(NULL, " ");
            if (!args[i]) {
                write_error("error: not enough array elements\n");
                valid = 0;
                break;
            }
        }

        if (valid) {
            command_2(n, args);
        }

        free(args);
    }
    else {
        write_error("error: unknown command\n");
    }

    write_string(prompt);
}
```

```
    write_string("\n");
    return 0;
}
```

prog2.c

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>
#include <dlfcn.h>

#define BUFFER_SIZE 4096
#define OUTPUT_SIZE 1024

typedef float (*pi_func_t)(int);
typedef int* (*sort_func_t)(int*, size_t);

static void* lib_handle = NULL;
static pi_func_t pi_func = NULL;
static sort_func_t sort_func = NULL;
static int current_lib = 1;

static void write_string(const char* str) {
    write(STDOUT_FILENO, str, strlen(str));
}

static void write_error(const char* str) {
    write(STDERR_FILENO, str, strlen(str));
}

static int parse_int(const char* str, int* value) {
```

```
    if (!str || !*str) return 0;

    *value = 0;
    int sign = 1;
    int i = 0;

    if (str[0] == '-') {
        sign = -1;
        i = 1;
    }

    while (str[i] >= '0' && str[i] <= '9') {
        *value = *value * 10 + (str[i] - '0');
        i++;
    }

    *value *= sign;
    return 1;
}

static int load_library(int lib_num) {
    char buffer[OUTPUT_SIZE];

    if (lib_handle) {
        dlclose(lib_handle);
    }

    char lib_name[32];
    int len = snprintf(lib_name, sizeof(lib_name), "./libfunc%d.so", lib_num);

    lib_handle = dlopen(lib_name, RTLD_LAZY);
    if (!lib_handle) {
        len = snprintf(buffer, OUTPUT_SIZE, "error: failed to load library %s: %s\n",
                lib_name, dlerror());
```

```
        write_error(buffer);
        return 0;
    }

    pi_func = (pi_func_t)dlsym(lib_handle, "pi");
    sort_func = (sort_func_t)dlsym(lib_handle, "sort");

    if (!pi_func || !sort_func) {
        write_error("error: failed to find required functions in library\n");
        dlclose(lib_handle);
        lib_handle = NULL;
        return 0;
    }

    current_lib = lib_num;
    len = snprintf(buffer, OUTPUT_SIZE, "switched to library: %s\n", lib_name);
    write_string(buffer);
    return 1;
}

static void command_0() {
    int new_lib = (current_lib == 1) ? 2 : 1;
    load_library(new_lib);
}

static void command_1(char* arg) {
    char buffer[OUTPUT_SIZE];
    int k;

    if (!pi_func) {
        write_error("error: pi function not loaded\n");
        return;
    }
```

```
    if (!parse_int(arg, &k)) {
        write_error("error: invalid argument for command 1\n");
        return;
    }

    float result = pi_func(k);
    int len = snprintf(buffer, OUTPUT_SIZE, "pi(%d) = %.6f (lib %d)\n",
                       k, result, current_lib);
    write(STDOUT_FILENO, buffer, len);
}

static void command_2(int n, char** args) {
    char buffer[OUTPUT_SIZE];

    if (!sort_func) {
        write_error("error: sort function not loaded\n");
        return;
    }

    if (n <= 0) {
        write_error("error: invalid array size\n");
        return;
    }

    int* array = malloc(n * sizeof(int));
    if (!array) {
        write_error("error: memory allocation failed\n");
        return;
    }

    for (int i = 0; i < n; i++) {
        if (!parse_int(args[i], &array[i])) {
            write_error("error: invalid array element\n");
            free(array);
```

```
            return;
        }
    }

    int* sorted = sort_func(array, n);
    if (!sorted) {
        write_error("error: sorting failed\n");
        free(array);
        return;
    }

    int len = snprintf(buffer, OUTPUT_SIZE, "sorted: ");
    write(STDOUT_FILENO, buffer, len);

    for (int i = 0; i < n; i++) {
        len = snprintf(buffer, OUTPUT_SIZE, "%d ", sorted[i]);
        write(STDOUT_FILENO, buffer, len);
    }

    len = snprintf(buffer, OUTPUT_SIZE, "(lib %d)\n", current_lib);
    write(STDOUT_FILENO, buffer, len);

    free(sorted);
    free(array);
}

int main() {
    char buffer[BUFFER_SIZE];
    const char* prompt = "> ";

    if (!load_library(1)) {
        write_error("error: failed to load initial library\n");
        return 1;
    }
```

```
write_string("Dynamic program (dlopen/dlsym)\n");
write_string("Commands: 0 - switch library\n");
write_string("          1 k - calculate pi(k)\n");
write_string("          2 n a1 a2 ... an - sort array\n");
write_string("          exit - exit program\n");
write_string(prompt);

while (1) {
    int bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1);
    if (bytes_read <= 0) break;

    buffer[bytes_read] = '\0';

    // Remove newline
    if (buffer[bytes_read - 1] == '\n') {
        buffer[bytes_read - 1] = '\0';
        bytes_read--;
    }

    if (strcmp(buffer, "exit") == 0) {
        break;
    }

    // Skip empty lines
    if (bytes_read == 0) {
        write_string(prompt);
        continue;
    }

    // Parse command
    char* token = strtok(buffer, " ");
    if (!token) {
        write_string(prompt);
```

```
    continue;
}

if (strcmp(token, "0") == 0) {
    command_0();
}
else if (strcmp(token, "1") == 0) {
    char* arg = strtok(NULL, " ");
    command_1(arg);
}
else if (strcmp(token, "2") == 0) {
    char* size_str = strtok(NULL, " ");
    int n;

    if (!size_str || !parse_int(size_str, &n)) {
        write_error("error: invalid array size\n");
        write_string(prompt);
        continue;
    }

    char** args = malloc(n * sizeof(char*));
    if (!args) {
        write_error("error: memory allocation failed\n");
        write_string(prompt);
        continue;
    }

    int valid = 1;
    for (int i = 0; i < n; i++) {
        args[i] = strtok(NULL, " ");
        if (!args[i]) {
            write_error("error: not enough array elements\n");
            valid = 0;
            break;
        }
```

```
            }

            if (valid) {
                command_2(n, args);
            }

            free(args);
        }
        else {
            write_error("error: unknown command\n");
        }

        write_string(prompt);
    }

    if (lib_handle) {
        dlclose(lib_handle);
    }

    write_string("\n");
    return 0;
}
```

Протокол работы программы

Тестирование:

```
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab4$ ./prog1
Static program (libfunc1.so)
Commands: 1 k - calculate pi(k)
          2 n a1 a2 ... an - sort array
          exit - exit program
> 1 100
pi(100) = 3.131593
> 1 1000
pi(1000) = 3.140593
> 2 5 3 -2 8 10 -12
sorted: -12 -2 3 8 10
> 2 3 -1 1 0
sorted: -1 0 1
> 123 6
error: unknown command
> 26 0
error: unknown command
> exit

aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab4$ ./prog2
switched to library: ./libfunc1.so
Dynamic program (dlopen/dlsym)
Commands: 0 - switch library
          1 k - calculate pi(k)
          2 n a1 a2 ... an - sort array
          exit - exit program
> 1 1000
pi(1000) = 3.140593 (lib 1)
> 1 10000
pi(10000) = 3.141499 (lib 1)
> 0
switched to library: ./libfunc2.so
> 2 10 2 9 -2 5 8 0 -100 9 76 21
sorted: -100 -2 0 2 5 8 9 9 21 76 (lib 2)
> 2 2 0 -2
sorted: -2 0 (lib 2)
> 0
switched to library: ./libfunc1.so
> 2 4 9 -1 7 11
sorted: -1 7 9 11 (lib 1)
> 24 hjk
error: unknown command
> hjk
error: unknown command
> exit
```

Strace:

aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab4$ strace -f ./prog2

execve("./prog2", ["./prog2"], 0x7ffd8190aed8 /* 27 vars */) = 0

brk(NULL) = 0x5650650ab000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x779b88be5000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=20243, ...}) = 0

mmap(NULL, 20243, PROT_READ, MAP_PRIVATE, 3, 0) = 0x779b88be0000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x779b88800000

mmap(0x779b88828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE| MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x779b88828000

mmap(0x779b889b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED| MAP_DENYWRITE, 3, 0x1b0000) = 0x779b889b0000

mmap(0x779b889ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE| MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x779b889ff000

mmap(0x779b88a05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE| MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x779b88a05000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE| MAP_ANONYMOUS, -1, 0) = 0x779b88bdd000

arch_prctl(ARCH_SET_FS, 0x779b88bdd740) = 0

set_tid_address(0x779b88bdda10) = 246370

set_robust_list(0x779b88bdda20, 24)   = 0

rseq(0x779b88bde060, 0x20, 0, 0x53053053) = 0

mprotect(0x779b889ff000, 16384, PROT_READ) = 0

mprotect(0x5650608a4000, 4096, PROT_READ) = 0

mprotect(0x779b88c1d000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

munmap(0x779b88be0000, 20243)   = 0

getrandom("\x42\x32\xac\x7a\xf7\xf5\x78\x32", 8, GRND_NONBLOCK) = 8

brk(NULL)   = 0x5650650ab000

brk(0x5650650cc000)   = 0x5650650cc000

openat(AT_FDCWD, "./libfunc1.so", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0777, st_size=15624, ...}) = 0

getcwd("/mnt/c/Users/Alyona/2kurs/os/lab4", 128) = 34

mmap(NULL, 16416, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x779b88be0000

mmap(0x779b88be1000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x779b88be1000

mmap(0x779b88be2000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x779b88be2000

mmap(0x779b88be3000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x779b88be3000

close(3)   = 0

mprotect(0x779b88be3000, 4096, PROT_READ) = 0

```
write(1, "switched to library: ./libfunc1."..., 35switched to library: ./libfunc1.so
) = 35
write(1, "Dynamic program (dlopen/dlsym)\n", 31Dynamic program (dlopen/dlsym)
) = 31
write(1, "Commands: 0 - switch library\n", 29Commands: 0 - switch library
) = 29
write(1, "          1 k - calculate pi(k)\n", 32          1 k - calculate pi(k)
) = 32
write(1, "          2 n a1 a2 ... an - sor"..., 40          2 n a1 a2 ... an - sort array
) = 40
write(1, "          exit - exit program\n", 30          exit - exit program
) = 30
write(1, "> ", 2> )                      = 2
read(0, 2 5 -9 -23 19 0 2
"2 5 -9 -23 19 0 2\n", 4095)    = 18
write(1, "sorted: ", 8sorted: )                = 8
write(1, "-23 ", 4-23 )                    = 4
write(1, "-9 ", 3-9 )                      = 3
write(1, "0 ", 20 )                       = 2
write(1, "2 ", 22 )                       = 2
write(1, "19 ", 319 )                     = 3
write(1, "(lib 1)\n", 8(lib 1)
)               = 8
```

```
write(1, "> ", 2> )                     = 2
read(0, 2 5 -9 -12 19 0 2
"2 5 -9 -12 19 0 2\n", 4095)    = 18
write(1, "sorted: ", 8sorted: )                = 8
write(1, "-12 ", 4-12 )                  = 4
write(1, "-9 ", 3-9 )                   = 3
write(1, "0 ", 20 )                     = 2
write(1, "2 ", 22 )                     = 2
write(1, "19 ", 319 )                   = 3
write(1, "(lib 1)\n", 8(lib 1)
)               = 8
write(1, "> ", 2> )                     = 2
read(0, exit
"exit\n", 4095)                = 5
munmap(0x779b88be0000, 16416)          = 0
write(1, "\n", 1
)                     = 1
exit_group(0)                         = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения работы были изучены два способа использования динамических библиотек в Linux. Первый способ (статические зависимости на этапе компиляции) обеспечивает простоту разработки и высокую производительность за счёт прямого

вызова функций, но не позволяет менять реализацию без перекомпиляции. Второй способ (динамическая загрузка через dlopen/dlsym) предоставляет гибкость — возможность переключать реализации «на лету» и создавать плагинную архитектуру, хотя требует более сложной обработки ошибок и имеет накладные расходы на загрузку библиотек. Оба подхода имеют свои области применения: статическая линковка оптимальна для стабильных компонентов, а динамическая загрузка — для систем, требующих расширяемости и замены модулей во время выполнения.