Лекция 23. Статические и динамические библиотеки языка Си.

создал Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем
читает Насонова Алина Олеговна
ассистент кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Библиотеки языка Си (1)

Библиотека - это набор готовых функций и процедур, которые можно использовать в своих программах программист.

На разных компьютерных системах используются различные версии стандартной библиотеки языка C. В операционных системах BSD эта библиотека является частью самой системы и хранится в общем хранилище кода. Обычно такую библиотеку можно найти под определенным именем.

Библиотеки языка Си (2)

Программа —> Линковщик —> Исполняемый файл

|

[Библиотеки]

|

[Заголовочные файлы]

Две основные категории:

Статические библиотеки (.a – Linux,
.lib - Winodows)

Динамические библиотеки (.so – Linux,
.dll – Winodows,
.dylib – MacOS)

Библиотеки языка Си (3)

```
project/
│
├── main.c   # Точка входа
├── libmath.h # Заголовочный файл библиотеки
└── libmath.c # Реализация функций библиотеки
```

Библиотеки языка Си (4)

Библиотеки позволяют решить ряд задач:

- Повторное использование кода
- Упрощение разработки
- Модульность и организованность проекта
- Оптимизация памяти и производительности

Библиотеки языка Си (5)

1. Экономия времени разработки:
   - Использование готового кода вместо написания с нуля
2. Упрощение поддержки:
   - Изменения требуются только в библиотеке, а не во всех местах использования
3. Оптимизация ресурсов:
   - Статические библиотеки уменьшают объем дублируемого кода
   - Динамические библиотеки экономят оперативную память
4. Модульность:
   - Программа становится более структурированной
   - Упрощается тестирование и отладка

Библиотеки языка Си (5)

1. Экономия времени разработки:
   - Использование готового кода вместо написания с нуля
2. Упрощение поддержки:
   - Изменения требуются только в библиотеке, а не во всех местах использования
3. Оптимизация ресурсов:
   - Статические библиотеки уменьшают объем дублируемого кода
   - Динамические библиотеки экономят оперативную память
4. Модульность:
   - Программа становится более структурированной
   - Упрощается тестирование и отладка

Библиотеки языка Си (5)

История развития библиотек в Си

- Первые версии языка Си не поддерживали библиотеки.
- В 1970-х годах появились первые статические библиотеки через архиватор **ar**.
- Динамические библиотеки появились в Unix системах в начале 1980-х годов.
- Современные системы управления библиотеками (например, **ld, dlopen**) развивались в 1990-х.

Библиотеки языка Си (6)

Создание статической библиотеки

ar rcs libmath.a math.o

Создание динамической библиотеки

gcc -shared -o libmath.so math.o

Библиотеки языка Си (7)

Преимущества и недостатки

Аспект	Статическая библиотека	Динамическая библиотека
Размер исполняемого файла	Больше (все функции включены)	Меньше (функции загружаются по необходимости)
Производительность	Выше (нет дополнительной загрузки)	Немного ниже (время загрузки)
Изменения	Требует компиляции программы	Можно обновить без компиляции
Память	Каждая программа имеет свою копию	Одна копия используется всеми программами

Библиотеки языка Си (8)

Статическая библиотека :

- Когда важна производительность.
- Когда нужно создать автономное приложение.
- Когда обновления библиотеки не планируются.

Динамическая библиотека :

- Когда важно экономить место.
- Когда нужно обновлять библиотеку без компиляции программы.
- Когда несколько программ используют одну и ту же библиотеку.

Библиотеки языка Си (9)

Стандартные библиотеки C :

- <stdio.h> — работа с потоками ввода-вывода.
- <stdlib.h> — аллокация памяти, конверсии, случайные числа.
- <math.h> — математические функции.

Популярные сторонние библиотеки :

- libcurl — работа с сетью.
- libpng — работа с изображениями PNG.
- SQLite — встраиваемая база данных.

Процесс линковки (1)

Линковка — процесс соединения объектных файлов и библиотек в один исполняемый файл.

Типы линковки:

- Статическая линковка : Библиотека включается в исполняемый файл.
- Динамическая линковка : Библиотека загружается при выполнении программы.

Процесс линковки (2)

Не стоит путать компиляцию и линковку!

Компиляция :

- Перевод исходного кода в машинный код.
- Создает объектные файлы.

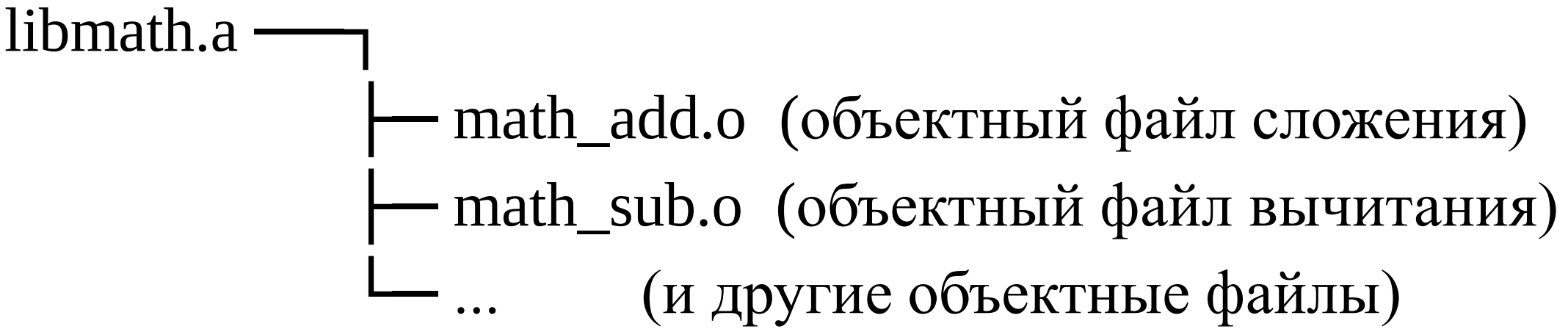
Линковка :

- Соединяет объектные файлы и библиотеки в один исполняемый файл.
- Разрешает внешние ссылки.

Создание статической библиотеки (1)

Статическая библиотека — это архив объектных файлов (.o), объединённых в один файл.

- Предкомпилированные функции и данные.
- Символы (имена функций и переменных), доступные для линковки.



Создание статической библиотеки (2)

1. Разработка функций и их реализация в **.c** файлах.
2. Перевод исходного кода в машинный код.
3. Создание архива с помощью утилиты **ar**.

```
gcc -c math.c -o math.o  # Компиляция в объектный файл
ar rcs libmath.a math.o  # Создание архива
```

Создание статической библиотеки (3)

```c
// libmath.c
int add(int a, int b) {
    return a + b;
}
int multiply(int a, int b) {
    return a * b;
}
int global_var = 42;
```

```c
// libmath.h
#ifndef LIBMATH_H
#define LIBMATH_H

int add(int, int);
int multiply(int, int);

extern int global_var;
#endif
```

```
gcc -c libmath.c -o libmath.o
ar rcs libmath.a libmath.o
```

Как быть если нужно скрыть внутренние реализации от возможного их использования в исходном коде?

Создание статической библиотеки (4)

Линковка приложения со статической библиотекой

gcc main.c -L. -lmath -o program

-L.: Указывает путь к библиотеке.

-lmath: Ищет libmath.a или libmath.so.

Альтернативный синтаксис

gcc main.c libmath.a -o program

Создание статической библиотеки (5)

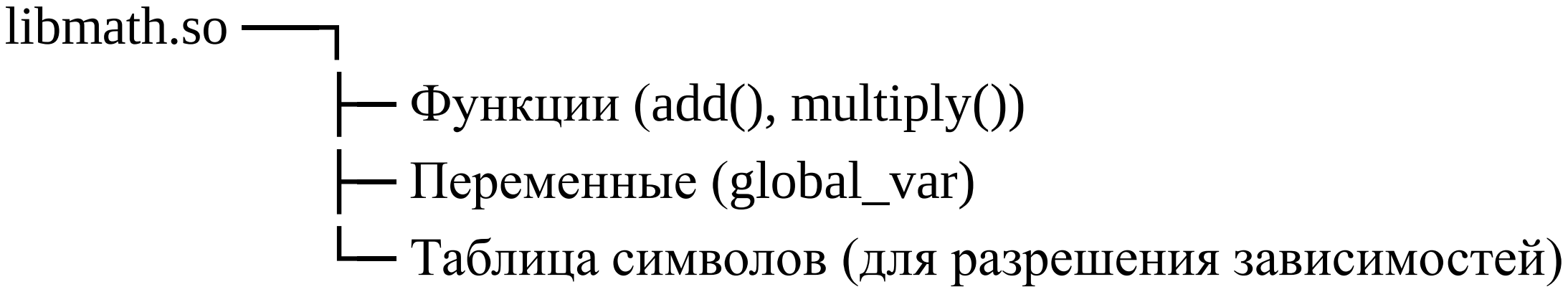
Отладка через компиляцию

```
gcc -g -c libmath.c -o libmath.o  # Компиляция с функциями отладки
nm libmath.a  # Просмотр символов в библиотеке
objdump -d libmath.a  # Просмотр дизассемблера
```

Создание динамической библиотеки (1)

Динамическая библиотека (.so, .dll) загружается в память при запуске приложения.

- Позиционно-независимый код (PIC/PIE).
- Символы (функции и переменные) доступны для всех программ, использующих её.
- Управление версиями через символические ссылки (например, libmath.so.1.0).



Создание динамической библиотеки (2)

- Написание исходного кода
- Компиляция с флагом PIC
  - gcc -fPIC -c math.c -o math.o
- Создание .so файла (в Linux)
  - gcc -shared -o libmath.so math.o

Исходный код (math.c) —> Объектный файл (math.o с PIC) —> Динамическая библиотека (libmath.so)

Создание динамической библиотеки (3)

Компилятор GCC

gcc -fPIC -c math.c -o math.o # Компиляция с PIC

Создание .so файла :

gcc -shared -Wl,-soname,libmath.so -o libmath.so.1.0 math.o

Утилиты для управления :

ldconfig — обновление кэша динамических библиотек.

ldd — проверка зависимостей.

Создание динамической библиотеки (4)

```c
// libmath.c
int add(int a, int b) {
    return a + b;
}
int multiply(int a, int b) {
    return a * b;
}
```

Флаг -fPIC обязателен для компиляции объектных файлов

```
gcc -fPIC -c libmath.c -o libmath.o
gcc -shared -o libmath.so libmath.o
```

```c
// libmath.h
#ifndef LIBMATH_H
#define LIBMATH_H

int add(int, int);
int multiply(int, int);

#endif
```

Позволяет библиотеке работать в разных адресных пространствах.

По умолчанию все функции экспортируются (кроме static).

Можно контролировать через файл версий

Создание динамической библиотеки (5)

Автоматический экспорт функций и глобальных переменных :

Все нестатические функции и переменные доступны. (по умолчанию)

Ручной экспорт функций и глобальных переменных :

Используйте файл libmath.version:

```
LIBMATH_1.0 {
    global:
        add;
        multiply;
    local:
        *;
};
```

gcc -shared -Wl,--version-script=libmath.version -o libmath.so math.o

Создание динамической библиотеки (6)

Линковка приложения с динамической библиотекой

gcc main.c -L. -lmath -o program

Загрузка библиотеки во время выполнения :
Операционная система (через ld-linux.so или loader) ищет libmath.so.
Пути поиска задаются через LD_LIBRARY_PATH.

gcc main.c -L/path/to/libs -lmath -o program

Создание динамической библиотеки (7)

Особенности отладки динамических библиотек

```
gcc -g -fPIC -c math.c -o math.o
gdb --args ./program  # Отладка приложения
ldd program  # Просмотр связанных библиотек
```

Работа с библиотеками (1)

Библиотеки по умолчанию находятся в Linux:

/lib, /usr/lib, /usr/local/lib.

Флаги компилятора :

-I — указывает пути к заголовочным файлам.

-L — указывает пути к библиотекам.

-l — имя библиотеки (например, -lmath → libmath.a или libmath.so)

gcc main.c -I/path/to/headers -L/path/to/libs -lmath -o program

Работа с библиотеками (2)

Путь к динамическим библиотекам, которые будут загружены при запуске.

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
./program
```

При запуске программы:

1. Проверяются пути в LD_LIBRARY_PATH
2. Затем стандартные пути (например, /lib, /usr/lib)

Работа с библиотеками (3)

Что бы проанализировать зависимости используется инструмент **ldd**

Показывает зависимости динамических библиотек. `ldd program`

linux-vdso.so.1 (0x7ffd2b3ff000)

libmath.so => /usr/local/lib/libmath.so
(0x7f8d1a1a0000)

libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x7f8d19f3d000)

Работа с библиотеками (4)

Просмотр списка функций, переменных в библиотеке.

nm libmath.so

0000000000001000 A __bss_start

0000000000000630 T add

0000000000000650 T multiply

Работа с библиотеками (5)

Дизассемблирование объектных файлов и библиотек.

```
objdump -d libmath.so
Disassembly of section .text:
0000000000000630 <add>:
    630:   89 7d fc    mov    %edi,%ebp
    633:   8b 45 08    mov    %eax,%edx
    ...
```

Работа с библиотеками (6)

Отладка проблем с библиотеками

Проблема: Не найдена библиотека :

```
error while loading shared libraries:
libmath.so: cannot open shared object
file
```

Решение: Убедитесь, что библиотека в LD_LIBRARY_PATH или системных каталогах.

Работа с библиотеками (7)

Отладка проблем с библиотеками

Проблема: Не найдена функция :

```
Error: undefined reference to `add`
```

Решение: Проверьте экспорт функций в библиотеке.

strace ./program 2>&1 | **grep** libmath

strace показывает все системные вызовы программы, которые та отправляет к системе во время выполнения

Работа с библиотеками (8)

Отладка проблем с библиотеками

Проблема: Ошибка загрузки динамической библиотеки

./program: error while loading shared libraries: libmath.so: cannot open shared object file: No such file or directory

Решение: export LD_LIBRARY_PATH=/path/to/lib:$LD_LIBRARY_PATH

Работа с библиотеками (9)

Отладка проблем с библиотеками

Проблема: Ошибка совместимости версии

./program: /usr/local/lib/libmath.so: version `LIBMATH_2.0' not found

Решение: Убедитесь, что версия библиотеки соответствует требованиям программы.

Динамическая загрузка библиотек (1)

Загрузка и использование библиотек во время выполнения программы, а не во время компиляции – динамическая загрузка.

`dlopen()` — загрузка библиотеки.

`dlsym()` — получение указателя на функцию/переменную.

`dlclose()` — разгрузка библиотеки.

`dlerror()` — проверка ошибок.

Динамическая загрузка библиотек (2)

```
void* dlopen(const char* filename,
             int flag)
```

Загружает библиотеку.

Флаги: RTLD_LAZY (ленивая загрузка),
RTLD_NOW (загрузка сразу).

```
void* dlsym(void* handle,
            const char* symbol)
```

Возвращает указатель на функцию/переменную.

Динамическая загрузка библиотек (3)

```
int dlclose(void* handle)
```

Отключает библиотеку

```
const char* dlerror()
```

Возвращает описание последней ошибки.

Динамическая загрузка библиотек (4)

```
void* handle = dlopen("libmath.so", RTLD_LAZY);
if (!handle) { fprintf(stderr, "Error: %s\n",
dlerror());     return 1;}
int (*add)(int, int) = dlsym(handle, "add");
if (!add) { fprintf(stderr, "Error: %s\n",
                                    dlerror());
dlclose(handle); return 1;}
printf("Result: %d\n", add(5, 3));
dlclose(handle); return 0;
```

Динамическая загрузка библиотек (5)

Преимущества динамической загрузки

1. Гибкость :

   Поддержка плагинов (например, в IDE или редакторах).

2. Модульность :

   Добавление/удаление функционала без перекомпиляции приложения.

3. Оптимизация памяти :

   Библиотеки загружаются только при необходимости.

4. Обновление без перезапуска :

   Например, в сетевых приложениях с *горячей* заменой модулей.

Динамическая загрузка библиотек (6)

Ограничения и проблемы

1.Сложность управления :

Нужно вручную загружать/разгружать библиотеки.

2.Проблемы совместимости :

Несовместимые версии библиотек могут вызвать ошибки.

3.Безопасность :

Динамическая загрузка из неизвестных источников может быть уязвимой.

4.Сложность отладки :

Ошибки возникают во время выполнения, а не компиляции.

Оптимизация и производительность (1)

Статическая библиотека :

Функции встраиваются в исполняемый файл → большой размер .

```
ls -l program_static-rwxr-xr-x 1 user user 10MB program_static
```

Динамическая библиотека :

Функции загружаются из отдельного файла → меньший размер.

```
ls -l program_dynamic-rwxr-xr-x 1 user user 500KB program_dynamic
```

Размер исполняемого файла:

Статическая библиотека → 10MB

Динамическая библиотека → 500KB

Оптимизация и производительность (1)

Влияние на размер исполняемого файла

Статическая библиотека :

Функции встраиваются в исполняемый файл → большой размер .

```
ls -l program_static-rwxr-xr-x 1 user user 10MB program_static
```

Динамическая библиотека :

Функции загружаются из отдельного файла → меньший размер.

```
ls -l program_dynamic-rwxr-xr-x 1 user user 500KB program_dynamic
```

Размер исполняемого файла:

Статическая библиотека → 10MB

Динамическая библиотека → 500KB

Оптимизация и производительность (2)

Влияние на время загрузки программы

Статическая библиотека :

Вся функциональность встроена → **быстрая загрузка** (нет зависимости от внешних файлов).

Динамическая библиотека :

Загрузка библиотек происходит при запуске → **возможная задержка** .

Но кэширование ОС ускоряет повторные запуски.

Время загрузки:

Статическая: 0.1 сек

Динамическая: 0.2 сек (первый запуск)

Динамическая: 0.05 сек (последующие запуски, кеширована)

Оптимизация и производительность (3)

Влияние на использование памяти

Статическая библиотека :

Каждая копия программы хранит свою копию библиотеки → **больше потребление памяти .**

Динамическая библиотека :

Один экземпляр библиотеки используется всеми процессами → **экономия памяти .**

Оптимизация и производительность (4)

Влияние на использование памяти

Статическая:

- Программа 1 использует libmath.so → 1MB памяти
- Программа 2 использует libmath.so → 1MB памяти (статическая)
- Программа 3 использует libmath.so → 1MB памяти (статическая)

Всего: 3MB

Динамическая:

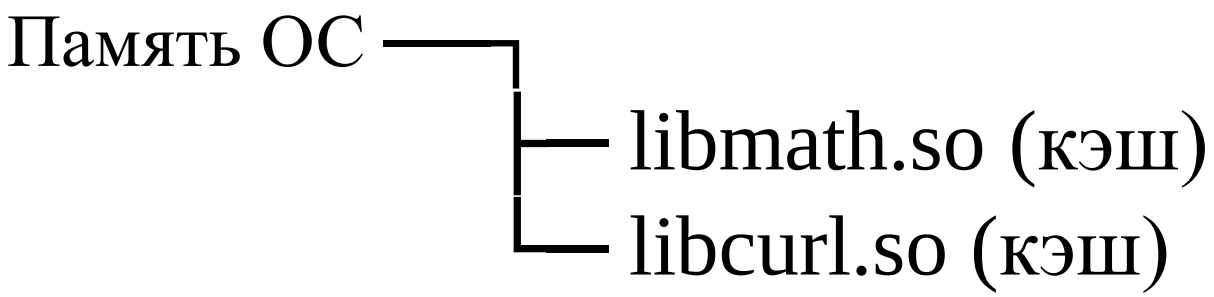
- libmath.so загружена один раз → 1MB памяти

Всего для всех программ: 1MB

Оптимизация и производительность (5)

Влияние на кэширование

Операционная система кэширует динамические библиотеки в память, а при запуске нескольких программ они используют один экземпляр из кэша. Это ускоряет загрузку повторно используемых библиотек и уменьшает нагрузки на ввод/вывод системы.



Спасибо за внимание

Насонова Алина Олеговна
ассистент кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)