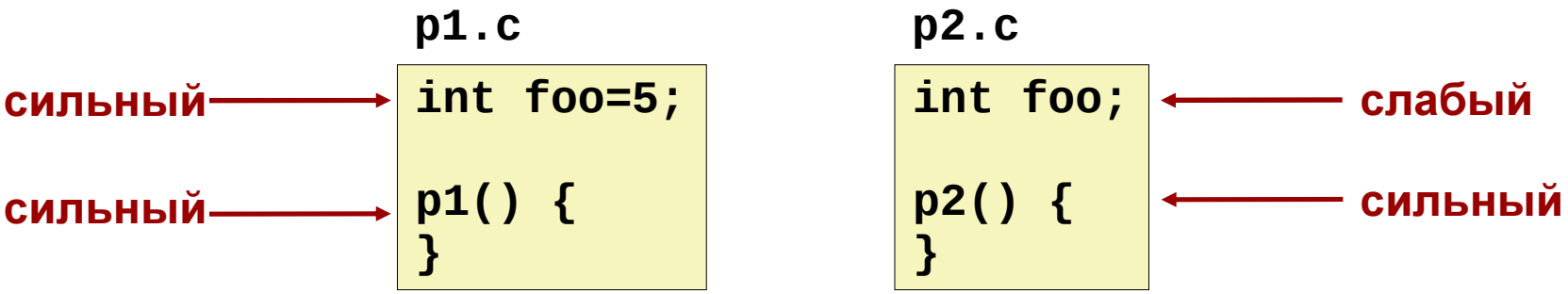
Лекция 23

30 апреля 2011

Сильные и слабые символы

- Каждый символ в программе либо «сильный», либо «слабый»
  - ***Сильные***: функции и инициализированные глобальные переменные
  - ***Слабые***: неинициализированные глобальные переменные



Правила работы с символами

- Правило 1: Одинаковые сильные символы запрещены
  - Каждый элемент может быть определен только один раз
  - В противном случае ошибка компоновки
- Правило 2: Один сильный символ и несколько слабых – выбираем сильный символ
  - Ссылки на слабые символы заменяются ссылками на сильный символ
- Правило 3: Если несколько слабых символов, выбираем произвольный
  - Поведение можно поменять gcc –fno-common

Вопросы к залу

`int x;` `p1() {}`	`p1() {}`	Ошибка компоновки: два сильных символа (**p1**)
`int x;` `p1() {}`	`int x;` `p2() {}`	Ссылки на **x** будут ссылаться на один и тот же неинициализированный int. Но какой?

`int x;`
`int y;`
`p1() {}`
	`double x;`
	`p2() {}`
		Запись в **x** (**p2**) может поменять **y**!

`int x=7;`
`int y=5;`
`p1() {}`
	`double x;`
	`p2() {}`
		Запись в **x** (**p2**) обязательно поменяет **y**!

`int x=7;` `p1() {}`	`int x;` `p2() {}`	Ссылка на **x** будет ссылаться на один и тот же инициализированный int.

Наихудший сценарий: два одинаковые «слабые» структуры, Откомпилированные разными компиляторами с разными правилами выравнивания.

Пример с заголовочным файлом

c1.c

```
#include "global.h"

int f() {
  return g+1;
}
```

global.h

```
#ifdef INITIALIZE
int g = 23;
static int init = 1;
#else
int g;
static int init = 0;
#endif
```

c2.c

```
#include <stdio.h>
#include "global.h"

int main() {
  if (!init)
    g = 37;
  int t = f();
  printf("Calling f yields %d\n", t);
  return 0;
}
```

Препроцессирование

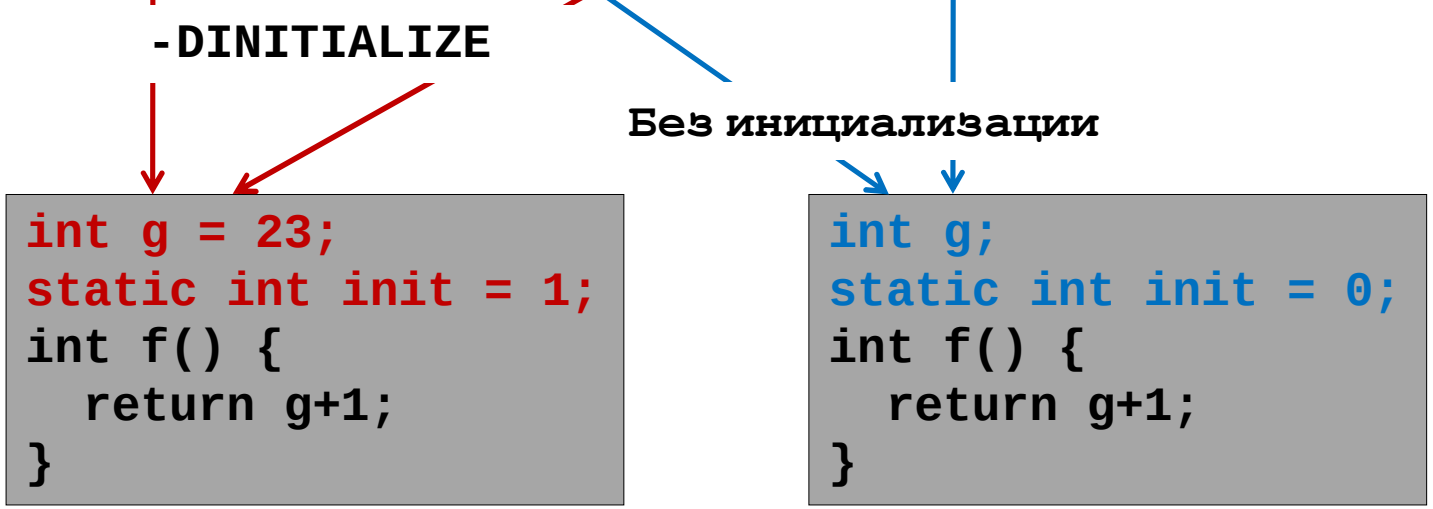
c1.c

```
#include "global.h"

int f() {
  return g+1;
}
```

global.h

```
#ifdef INITIALIZE
int g = 23;
static int init = 1;
#else
int g;
static int init = 0;
#endif
```



#include заставляет препроцессор Си выполнить подстановку файла

Роль заголовочных файлов (.h-файлов)

c1.c

```
#include "global.h"

int f() {
  return g+1;
}
```

global.h

```
#ifdef INITIALIZE
int g = 23;
static int init = 1;
#else
int g;
static int init = 0;
#endif
```

c2.c

```
#include <stdio.h>
#include "global.h"

int main() {
  if (!init)
    g = 37;
  int t = f();
  printf("Calling f yields %d\n", t);
  return 0;
}
```

Что произойдет:

```
gcc -o p c1.c c2.c
   ??
gcc -o p c1.c c2.c
   \
   -DINITIALIZE
   ??
```

Глобальные переменные

- Следует избегать, если только есть такая возможность

- В противном случае
  - Используйте **`static`** если это возможно
  - Если определяете глобальную переменную, инициализируйте ее
  - Используйте **`extern`** если ссылаетесь на внешнюю глобальную переменную

Работа с общими функциями

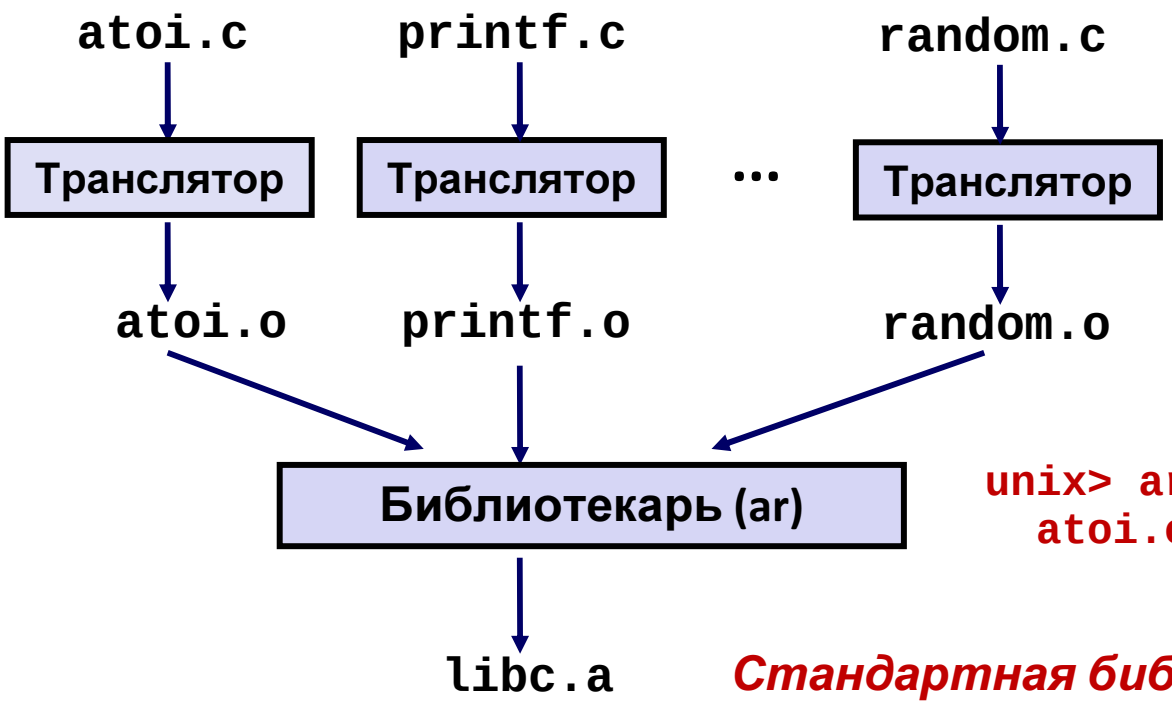
- Как следует размещать функции, часто используемые разными программами?
  - Математика, I/O, управление памятью, работа со строками, и т.д.

- Исходя из порядка компоновки:
  - **Вариант 1:** Поместить все функции в один файл
    - Компонуемся с одним большим объектным файлом
    - Неэффективно
  - **Вариант 2:** Поместить каждую функцию в отдельный файл
    - Во время компоновки явно указываем нужные объектные файлы
    - Более эффективно, но крайне неудобно для программиста

Решение: статические библиотеки

- Статические библиотеки (.a – файлы-архивы)
  - Близкие по смыслу перемещаемые объектные файлы группируются в одном файле, в т.н. называемом архиве.
  - Компоновщику указывают набор архивов для того, чтоб он попытался найти в них код с недостающими символами.
  - Если содержащийся в архиве файл помогает разрешить символ, то его автоматически включают в компоновку.

Создание статической библиотеки




```
unix> ar rs libc.a \
  atoi.o printf.o … random.o
```

Стандартная библиотека Си

- **Библиотекарь позволяет выполнять инкрементальное обновление**
- **Повторная компиляция функции и замена соответствующего o-файла в библиотеке.**

Часто используемые библиотеки

`libc.a` (Стандартная библиотека Си)

- 8 МБ архив из 1392 объектных файлов.
- I/O, управление памятью, работа со строками, даты и время, случайные числа, целочисленные математические функции

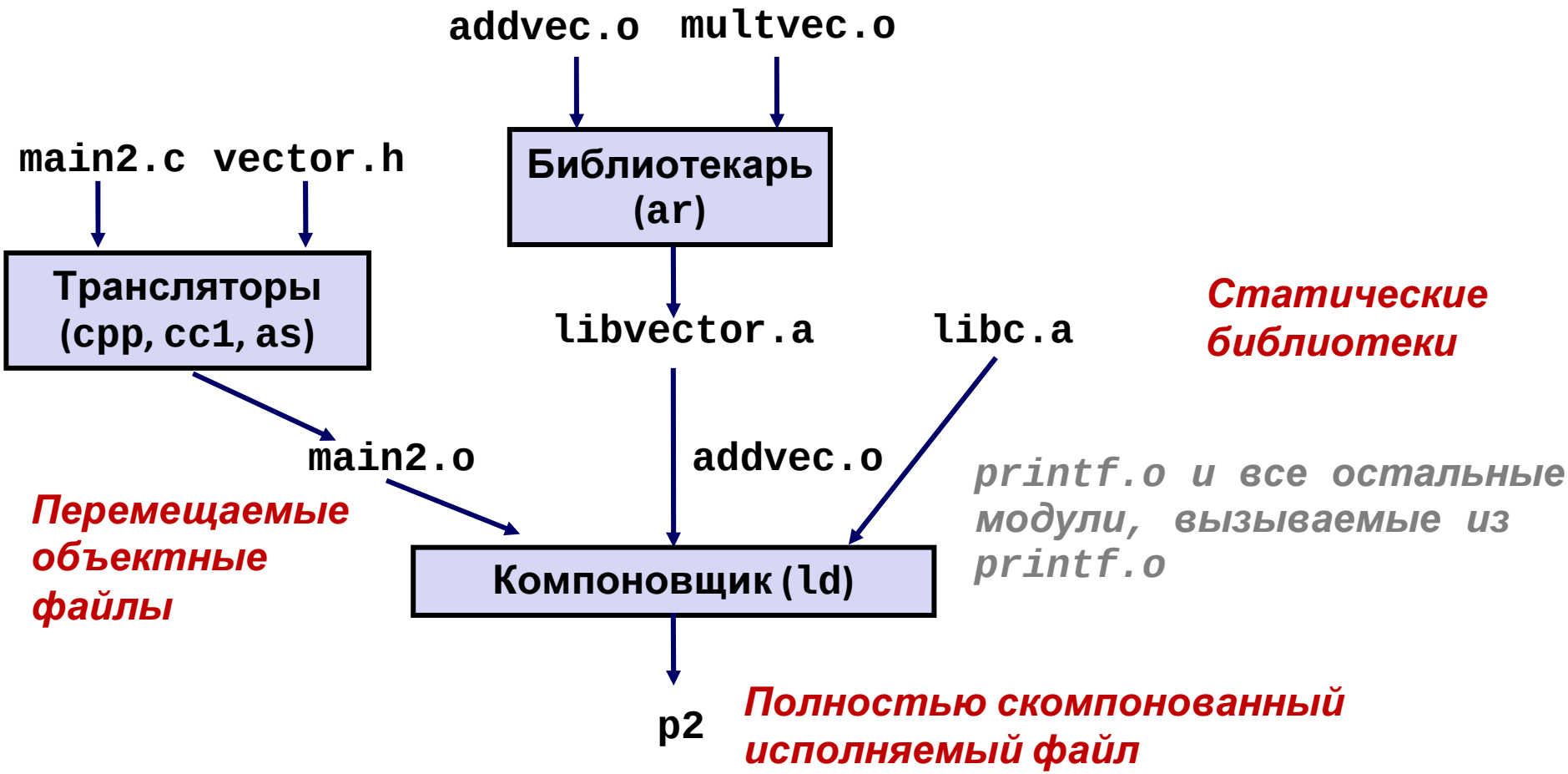
`libm.a` (Математическая библиотека Си)

- 1 МБ архив из 401 объектных файлов.
- Математические функции над числами с плавающей точкой (sin, cos, tan, log, exp, sqrt, …)

```
% ar -t /usr/lib/libc.a | sort
…
fork.o
…
fprintf.o
fpu_control.o
fputc.o
freopen.o
fscanf.o
fseek.o
fstab.o
…
```

```
% ar -t /usr/lib/libm.a | sort
…
e_acos.o
e_acosf.o
e_acosh.o
e_acoshf.o
e_acoshl.o
e_acosl.o
e_asin.o
e_asinf.o
e_asinl.o
…
```

Компоновка со статическими библиотеками



Использование статических библиотек

- Алгоритм компоновщика для разрешения внешних ссылок :
  - Просматриваем **.o** файлы и **.a** файлы в порядке их следования в командной строке.
  - В процессе просмотра, поддерживаем список неразрешенных в данный момент символов.
  - Как только появляется новый **.o** или **.a** файл, пытаемся разрешить каждый еще неразрешенный символ среди символов, определенных в найденном файле.
  - Ошибка линковки, если по окончании просмотра остался хоть один неразрешенный символ.
- Проблема:
  - Важен порядок объектных файлов в командной строке!
  - Решение: помещать все библиотеки в конец командной строки.

```
unix> gcc -L. libtest.o -lmine
unix> gcc -L. -lmine libtest.o
libtest.o: In function `main':
libtest.o(.text+0x4): undefined reference to `libfun'
```

Загрузка исполняемого объектного файла