# Лекция 0x12

11 апреля

# Символы в процессе компоновки

- Глобальные символы
  - Символы определенные в одном модуле таким образом, что их можно использовать в других модулях.
  - Например: не-`static` Си-функции и не-`static` глобальные переменные.

- Внешние (неопределенные) символы
  - Глобальные символы, которые используются в модуле, но определены в каком-то другом модуле.

- Локальные символы
  - Символы определены и используются исключительно в одном модуле.
  - Например: Си-функции и переменные, определенные с модификатором `static`.
  - **Локальные символы *не являются* локальными переменными Си-программы**

# Задача: разрешение символов

Глобальные

main.c

```
int buf[2] = {1, 2};
char str[] = "Hello";
char *p    = "world";

int swap(char*, char*);

int main()
{
  swap(str, p);
  return 0;
}
```

Внешний

Глобальный

Внешний

Локальный

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

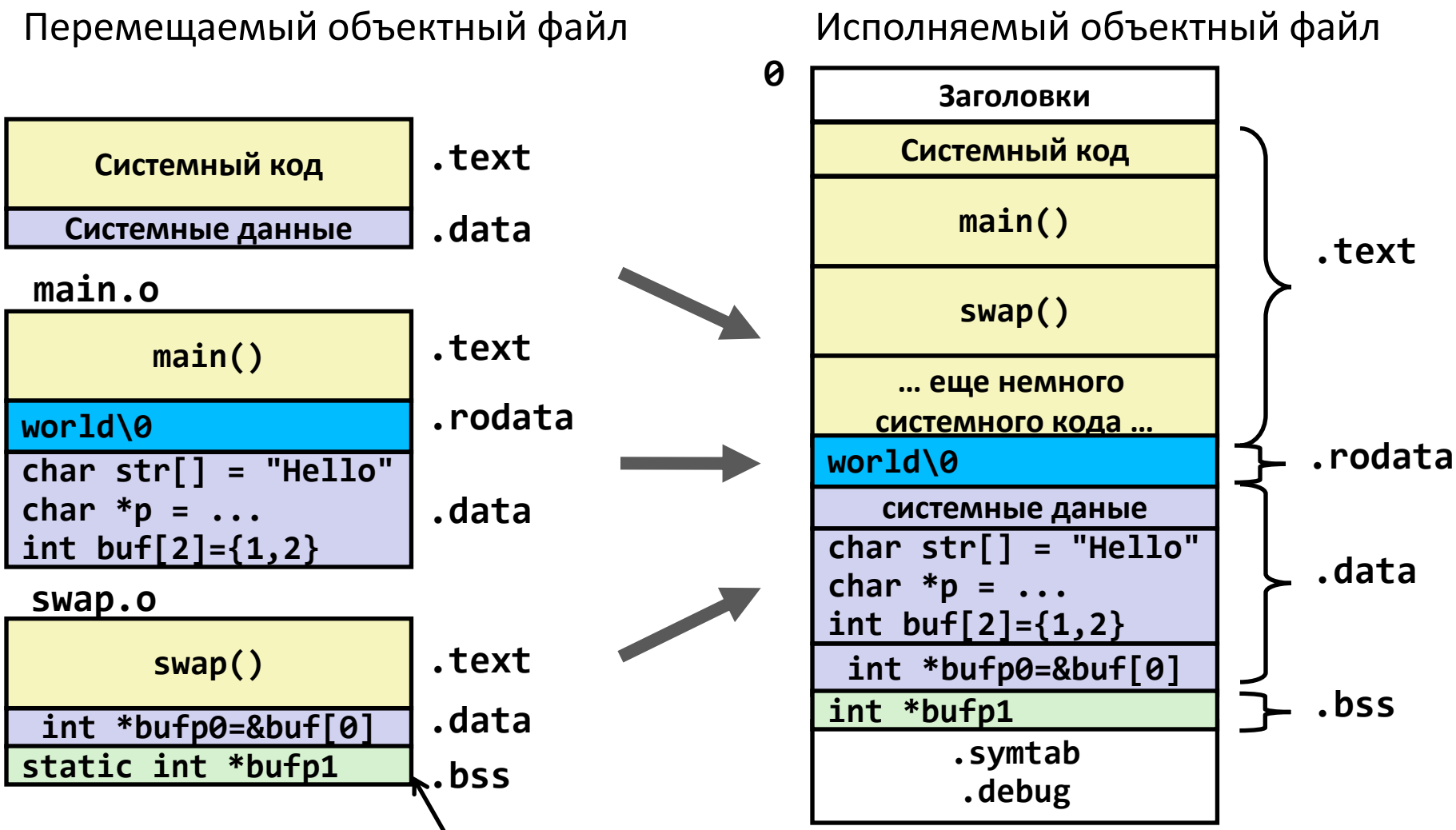
Глобальный

Для компоновщика переменной temp не существует

swap.c

- Тип символа: `local`, `global`, `extern`
- Присутствует ли символ в `.symtab`?
- Модуль, в котором символ определен?
- Размер?
- Смещение внутри секции?

# Перемещение кода и данных




**Даже приватные данные файла swap, требуют размещения в .bss**

# Предварительное определение переменных

```
int i1 = 10;          /* определение, внешнее связывание */
static int i2 = 20;   /* определение, внутреннее связывание */
extern int i3 = 30;   /* определение, внешнее связывание */
int i4;               /* предварительное определение, внешнее связывание */
static int i5;        /* предварительное определение, внутреннее связывание */

int i1;               /* корректное предварительное определение */
int i2;               /* ошибка, неувязка с уже заданным связыванием */
int i3;               /* корректное предварительное определение */
int i4;               /* корректное предварительное определение */
int i5;               /* ошибка, неувязка с уже заданным связыванием */
```

- Связывание имен
  - Внешнее: переменная доступна во всех единицах трансляции
  - Внутреннее: переменная доступна только в текущей единице трансляции
- Определение
  - В отсутствии инициализатора, определение трактуется как «предварительное» (tentative definition). В отсутствии других определений в данной единице трансляции переменной присваивается нулевое значение.

# Сильные и слабые символы

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

**p1.c**

**сильный** →
```
int foo=5;
```
**сильный** →
```
p1() {
}
```

**p2.c**

```
int foo;
```
← **слабый**

```
p2() {
}
```
← **сильный**

# 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

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

```
snoop@jezek:~/samples/2017/linking$ gcc -c swap.c
snoop@jezek:~/samples/2017/linking$ readelf -s swap.o

Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
...
    10: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND buf
...
```

```c
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;
int buf[];
void swap()
{
  int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

- COMMON-символы в перемещаемом коде не относят к какой-либо реально существующей секции
- Место в памяти для таких символов выделяется при построении исполняемого кода в общей .bss секции
- При «слиянии» COMMON-символов есть возможность проверить их размер

```
snoop@jezek:~/samples/2017/linking$ gcc -c swap.c
snoop@jezek:~/samples/2017/linking$ readelf -s swap.o

Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
...
    10: 00000004     4 OBJECT  GLOBAL DEFAULT  COM buf
...
```

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;
int buf[];
void swap()
{
```

- Отказ от механизма COMMON-символов приводит к «принудительному» выделению памяти в для них в секции .bss
- В силу особенностей языка Си и правил построения исполняемого кода, программа выделит в памяти место для каждого предварительно определенного «экземпляра» переменной
- Последствия?

```
snoop@jezek:~/samples/2017/linking$ gcc -fno-common -c swap.c
snoop@jezek:~/samples/2017/linking$ readelf -s swap.o

Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
...
    10: 00000000     4 OBJECT  GLOBAL DEFAULT    5 buf
...
snoop@jezek:~/samples/2017/linking$ readelf -S swap.o
There are 13 section headers, starting at offset 0x24c:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
...
  [ 5] .bss              NOBITS          00000000 000070 000008 00  WA  0   0  4
...
```

# Задача

| | | |
|---|---|---|
| `int x;`<br>`p1() {}` | `p1() {}` | Ошибка компоновки: два сильных символа (**p1**) |
| `int x;`<br>`p1() {}` | `int x;`<br>`p2() {}` | Ссылки на **x** будут ссылаться на один и тот же неинициализированный **int**. Но какой? |
| `int x;`<br>`int y;`<br>`p1() {}` | `double x;`<br>`p2() {}` | Запись в **x** (**p2**) может поменять **y**! |
| `int x=7;`<br>`int y=5;`<br>`p1() {}` | `double x;`<br>`p2() {}` | Запись в **x** (**p2**) ***обязательно поменяет*** **y**! |
| `int x=7;`<br>`p1() {}` | `int x;`<br>`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.

# 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

hello1.c

```
extern void func();

char *buf = "Hello, world!\n";

int main() {
  int ret_code = 0;
  func();
  return ret_code;
}
```

hello2.c

```
#include <stdio.h>

extern char* buf;

void func() {
   printf("%s", buf);
}
```

Makefile

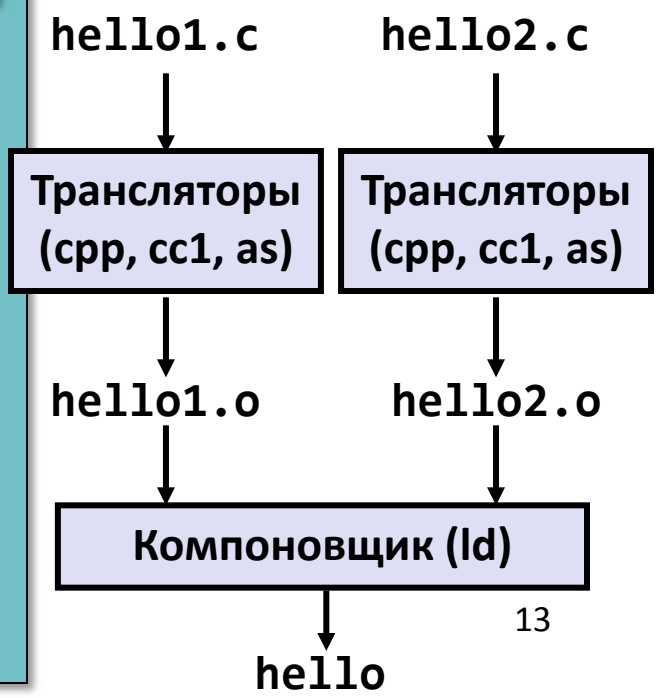
```
all: hello

hello: hello1.o hello2.o
   gcc -Xlinker -M -o hello hello1.o hello2.o

hello1.o: hello1.c
   gcc -c -o hello1.o hello1.c

hello2.o: hello2.c
   gcc -c -o hello2.o hello2.c

clean:
   rm -f hello hello1.o hello2.o
```



hello1.c

```
extern void func();

char *buf = "Hello, world!\n";

int main() {
  int ret_code = 0;
  func();
  return ret_code;
}
```

```
snoop@earth:~/samples/2014$ nm hello1.o
00000000 D buf
         U func
00000000 T main
```

hello1.c

```
extern void func();

char *buf = "Hello, world!\n";

int main() {
  int ret_code = 0;
  func();
  return ret_code;
}
```

```
snoop@earth:~/samples/2014$ readelf -s hello1.o

Symbol table '.symtab' contains 11 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS hello1.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 00000000     0 SECTION LOCAL  DEFAULT    8
     7: 00000000     0 SECTION LOCAL  DEFAULT    7
     8: 00000000     4 OBJECT  GLOBAL DEFAULT    3 buf
     9: 00000000    28 FUNC    GLOBAL DEFAULT    1 main
    10: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND func
```

hello1.c

```
extern void func();

char *buf = "Hello, world!\n";

int main() {
  int ret_code = 0;
  func();
  return ret_code;
}
```

```
snoop@earth:~/samples/2014$ readelf -r hello1.o

Relocation section '.rel.text' at offset 0x388 contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000012  00000a02 R_386_PC32        00000000   func

Relocation section '.rel.data' at offset 0x390 contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000000  00000501 R_386_32          00000000   .rodata
```

Что будет, если поменять код?

```
...
char buf[] = "Hello, world!\n";
...
```

**Напоминание**

Диалект Intel-синтаксиса, который используют программы binutils, отличается от диалекта, поддерживаемого nasm.

```
snoop@earth:~/samples/2014$ objdump -M intel -dr hello1.o

hello1.o:     file format elf32-i386


Disassembly of section .text:

00000000 <main>:
   0:   55                      push   ebp
   1:   89 e5                   mov    ebp,esp
   3:   83 e4 f0                and    esp,0xfffffff0
   6:   83 ec 10                sub    esp,0x10
   9:   c7 44 24 0c 00 00 00    mov    DWORD PTR [esp+0xc],0x0
  10:   00
  11:   e8 fc ff ff ff          call   12 <main+0x12>
                        12: R_386_PC32  func
  16:   8b 44 24 0c             mov    eax,DWORD PTR [esp+0xc]
  1a:   c9                      leave
  1b:   c3                      ret
```

```
snoop@earth:~/samples/2014$ readel

Relocation section '.rel.text' at
 Offset     Info    Type
00000012  00000a02 R_386_PC32

Relocation section '.rel.data' at
 Offset     Info    Type
00000000  00000501 R_386_32
```

```
snoop@earth:~/samples/2014$ objdump -s -j .data hello1.o

hello1.o:     file format elf32-i386

Contents of section .data:
 0000 00000000                             ....


snoop@earth:~/samples/2014$ objdump -s -j .rodata hello1.o

hello1.o:     file format elf32-i386

Contents of section .rodata:
 0000 48656c6c 6f2c2077 6f726c64 210a00    Hello, world!..
```

```
snoop@earth:~/samples/2014$ readelf -r hello1.o

Relocation section '.rel.text' at offset 0x388 contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000012  00000a02 R_386_PC32        00000000   func

Relocation section '.rel.data' at offset 0x390 contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000000  00000501 R_386_32          00000000   .rodata
```

hello2.c

```
#include <stdio.h>

extern char* buf;

void func() {
   printf("%s", buf);
}
```

```
snoop@earth:~/samples/2014$ nm hello2.o
         U buf
00000000 T func
         U printf
```

hello2.c

```c
#include <stdio.h>

extern char* buf;

void func() {
   printf("%s", buf);
}
```

```
snoop@earth:~/samples/2014$ readelf -s hello2.o

Symbol table '.symtab' contains 11 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS hello2.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    6
     8: 00000000    31 FUNC    GLOBAL DEFAULT    1 func
     9: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND buf
    10: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

hello2.c

```
#include <stdio.h>

extern char* buf;

void func() {
   printf("%s", buf);
}
```

```
snoop@earth:~/samples/2014$ readelf -r hello2.o

Relocation section '.rel.text' at offset 0x354 contains 3 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000008  00000901 R_386_32          00000000   buf
0000000d  00000501 R_386_32          00000000   .rodata
00000019  00000a02 R_386_PC32        00000000   printf
```

```
snoop@earth:~/samples/2014$ objdump -M intel -dr hello2.o

hello2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <func>:
   0:   55                      push   ebp
   1:   89 e5                   mov    ebp,esp
   3:   83 ec 18                sub    esp,0x18
   6:   8b 15 00 00 00 00       mov    edx,DWORD PTR ds:0x0
                        8: R_386_32     buf
   c:   b8 00 00 00 00          mov    eax,0x0
                        d: R_386_32     .rodata
  11:   89 54 24 04             mov    DWORD PTR [esp+0x4],edx
  15:   89 04 24                mov    DWORD PTR [esp],eax
  18:   e8 fc ff ff ff          call   19 <func+0x19>
                        19: R_386_PC32  printf
  1d:   c9                      leave
  1e:   c3                      ret
```

```
snoop@earth:~/samples/2014$ readelf -r hell

Relocation section '.rel.text' at offset 0x
 Offset     Info    Type            Sym.Val
00000008  00000901 R_386_32          000000
0000000d  00000501 R_386_32          000000
00000019  00000a02 R_386_PC32        000000
```

```
snoop@earth:~/samples/2014$ gcc -Xlinker -M -o hello hello1.o hello2.o
...
 .text           0x00000000080483e4       0x1c hello1.o
                 0x00000000080483e4                main
 .text           0x0000000008048400       0x1f hello2.o
                 0x0000000008048400                func
...
 .rodata         0x00000000080484e0        0xf hello1.o
 .rodata         0x00000000080484ef        0x3 hello2.o
...
 .data           0x000000000804a014        0x4 hello1.o
                 0x000000000804a014                buf
 .data           0x000000000804a018        0x0 hello2.o
```

```
snoop@earth:~/samples/2014$ nm hello1.o
00000000 D buf
         U func
00000000 T main
```

```
snoop@earth:~/samples/2014$ nm hello2.o
         U buf
00000000 T func
         U printf
```

Стандартная библиотека языка Си *printf.o*

```
snoop@earth:~/samples/2014$ gcc -Xlinker -M -o hello hello1.o hello2.o
...
 .text           0x00000000080483e4       0x1c hello1.o
                 0x00000000080483e4                main
 .text           0x0000000008048400       0x1f hello2.o
                 0x0000000008048400                func
...
 .rodata         0x00000000080484e0        0xf hello1.o
 .rodata         0x00000000080484ef        0x3 hello2.o
...
 .data           0x000000000804a014        0x4 hello1.o
                 0x000000000804a014                buf
 .data           0x000000000804a018        0x0 hello2.o
```

- Правила пересчета для значений ссылок
  - Тип перебазирования R_386_32
    ***новое значение ссылки = S + A***
  - Тип перебазирования R_386_PC32
    ***новое значение ссылки = S + A – P***
- *S* – абсолютный адрес памяти, которому символ соответствует после перемещения
- *A* – дополнительное слагаемое (addend), хранимое непосредственно в байтах ссылки
- *P* – абсолютный адрес ссылки

```
snoop@earth:~/samples/2014$ objdump -s -j .data hello

hello:     file format elf32-i386

Contents of section .data:
 804a00c 00000000 00000000 e0840408           ............

snoop@earth:~/samples/2014$ objdump -d -M intel -s -j .text hello
...
080483e4 <main>:
 80483e4:       55                      push   ebp
 80483e5:       89 e5                   mov    ebp,esp
 80483e7:       83 e4 f0                and    esp,0xfffffff0
 80483ea:       83 ec 10                sub    esp,0x10
 80483ed:       c7 44 24 0c 00 00 00    mov    DWORD PTR [esp+0xc],0x0
 80483f4:       00
 80483f5:       e8 06 00 00 00          call   8048400 <func>
 80483fa:       8b 44 24 0c             mov    eax,DWORD PTR [esp+0xc]
 80483fe:       c9                      leave
 80483ff:       c3                      ret
```

Как изменились ссылки, попавшие в исполняемый код из файла `hello1.o`?

# Работа с общими функциями

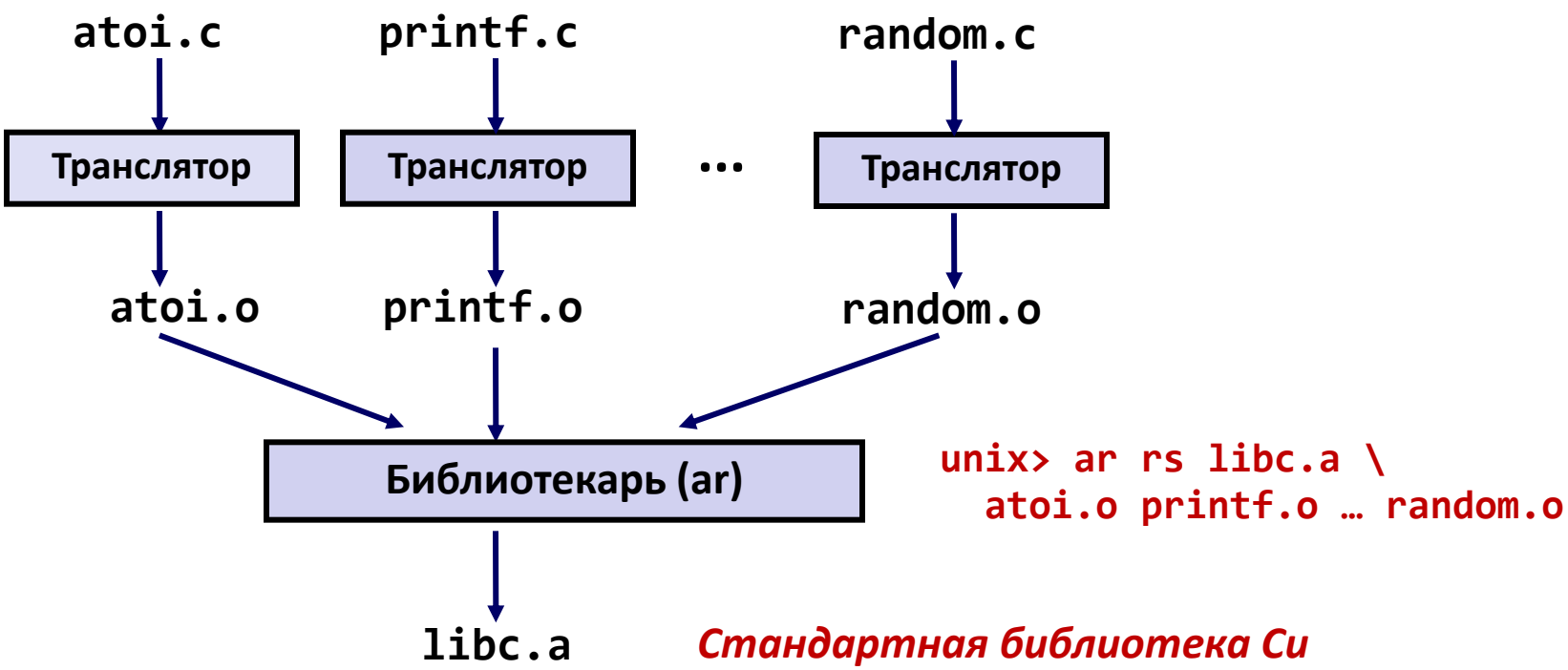
- Как следует размещать функции, часто используемые разными программами?
  - Математика, I/O, управление памятью, работа со строками, и т.д.

- 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




- Библиотекарь позволяет выполнять инкрементальное обновление
- Повторная компиляция функции и замена соответствующего о-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

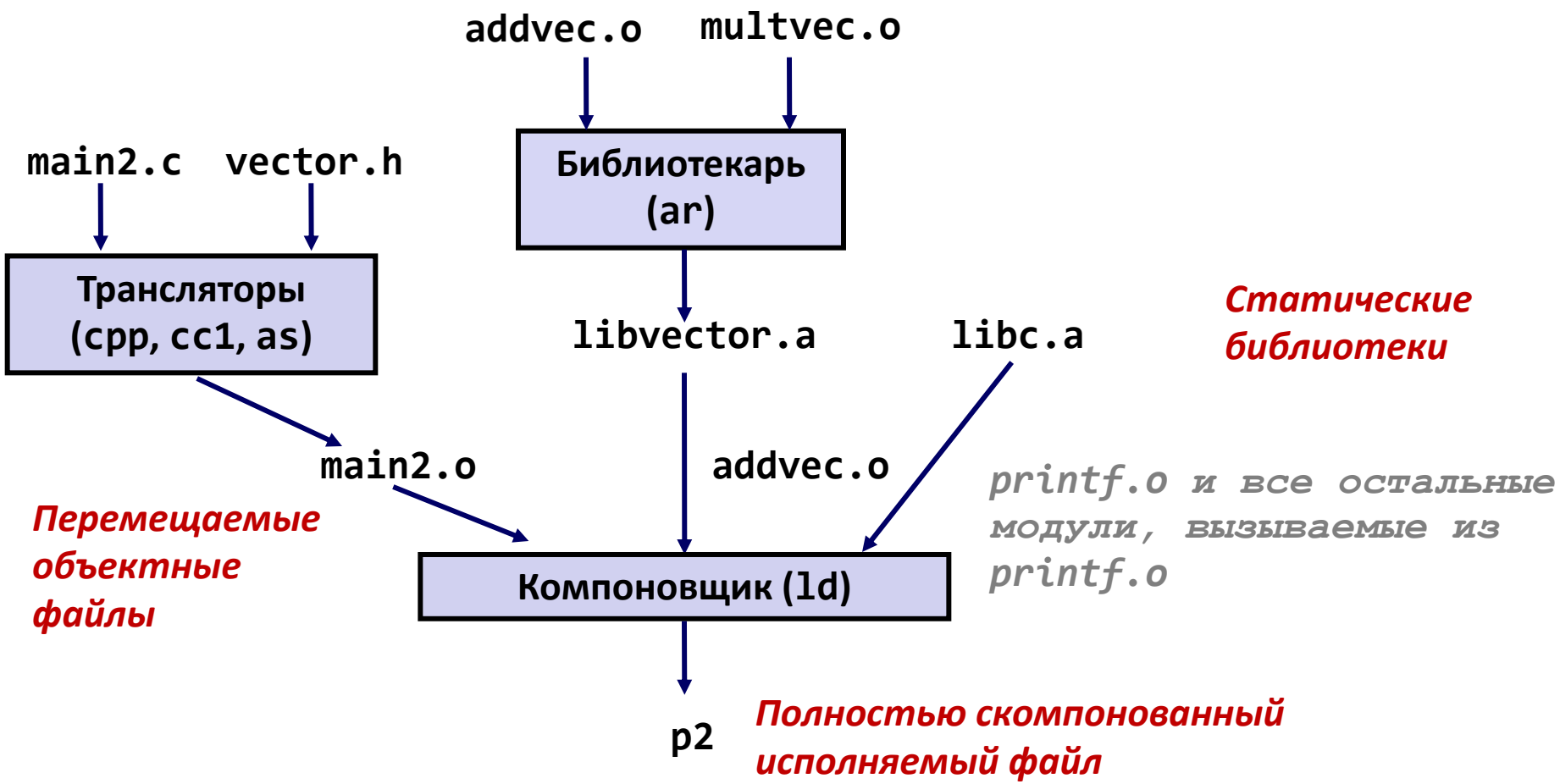
`libm.a` (Математическая библиотека Си)

- 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

- Подготовка адресного пространства для запускаемой программы
- Просмотр у заданного файла таблицы заголовков сегментов
  - Для каждого сегмента типа LOAD загружаем указанные байты из файла на указанные адреса памяти
- Передаем управление на точку входа в программу (символ `_start`)

```
typedef struct {
   Elf32_Word p_type;   /* LOAD, DYNAMIC, INTERP, PHDR, NULL, ... */
   Elf32_Off  p_offset; /* Смещение от начала файла */
   Elf32_Addr p_vaddr;  /* Начальный (базовый) адрес в памяти */
   Elf32_Addr p_paddr;  /* Не используется */
   Elf32_Word p_filesz; /* Размер в файле в байтах */
   Elf32_Word p_memsz;  /* Размер в памяти в байтах */
   Elf32_Word p_flags;  /* R/W/X */
   Elf32_Word p_align;  /* Величина выравнивания*/
} Elf32_Phdr;
```

# Связь секций и сегментов

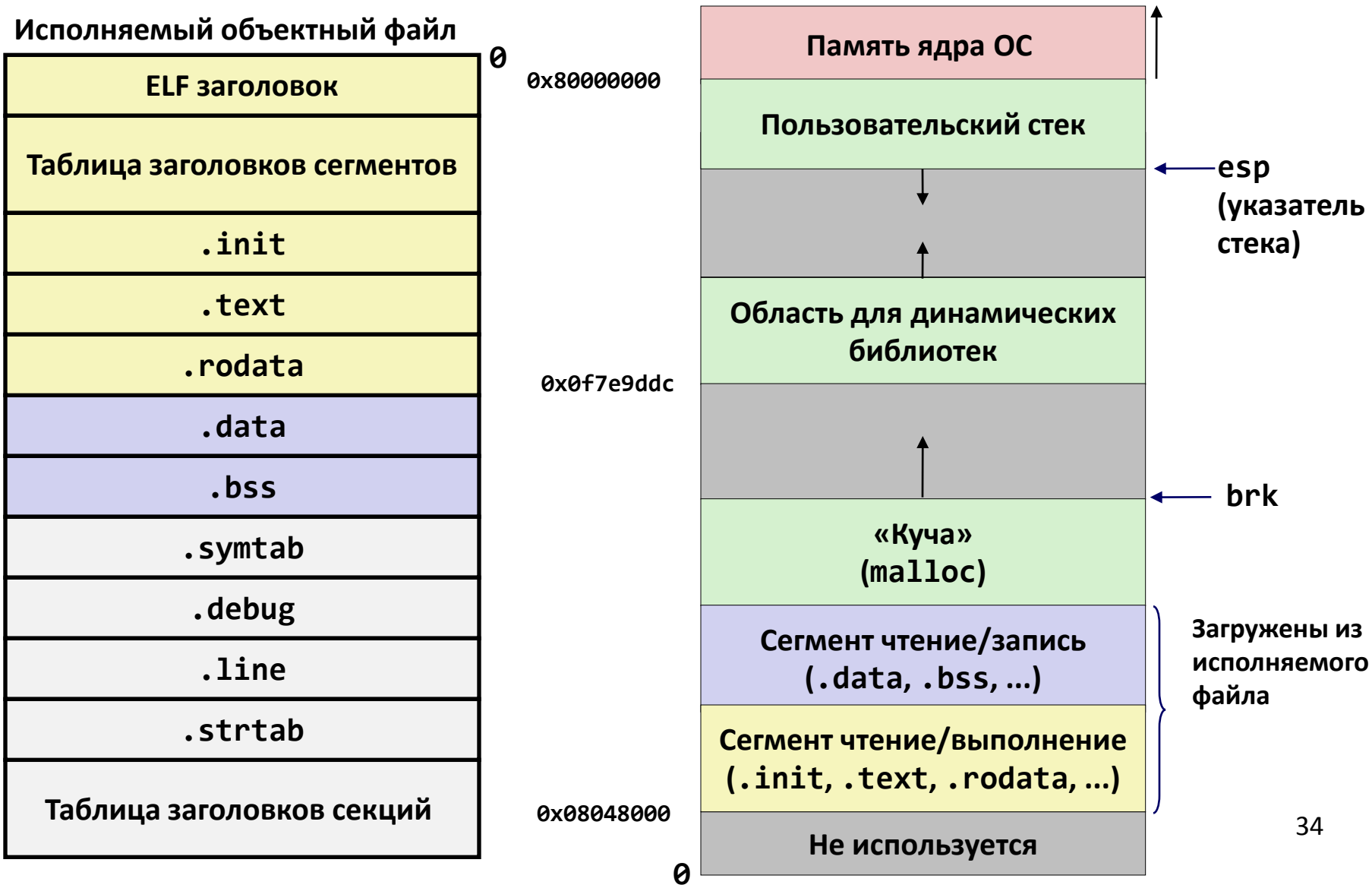
```
snoop@jezek:~/samples/2017/linking$ readelf -l hello-static

Elf file type is EXEC (Executable file)
Entry point 0x8048a8d
There are 6 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x08048000 0x08048000 0xa3a31 0xa3a31 R E 0x1000
  LOAD           0x0a3f3c 0x080ecf3c 0x080ecf3c 0x01124 0x02608 RW  0x1000
  NOTE           0x0000f4 0x080480f4 0x080480f4 0x00044 0x00044 R   0x4
  TLS            0x0a3f3c 0x080ecf3c 0x080ecf3c 0x00010 0x00028 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x0a3f3c 0x080ecf3c 0x080ecf3c 0x000c4 0x000c4 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00     ... .init .text .fini .rodata ...
   01     ... .init_array .fini_array .data .bss
```

# Загрузка исполняемого объектного файла



# Динамические (разделяемые) библиотеки

- Статические библиотеки имеют следующие недостатки:
  - Многократное копирование кода в построенных исполняемых файлах (всем нужно std libc)
  - Копии кода в исполняющихся программах
  - Любое исправление в системных библиотеках требует повторной компоновки для **всех** приложений

```
snoop@earth:~/samples$ gcc -static -o hello-static hello1.o hello2.o
snoop@jezek:~/samples$ ls -l hello-static
-rwxrwxr-x 1 snoop snoop 743476 Apr 12 09:15 hello-static
snoop@earth:~/samples$ gcc -o hello hello1.o hello2.o
snoop@jezek:~/samples$ ls -l hello
-rwxrwxr-x 1 snoop snoop 7404 Apr 12 09:06 hello
```

- Способ преодолеть эти недостатки: динамические библиотеки (shared libraries)
  - Объектные файлы, в которых содержатся код и данные компонуются с приложением *динамически*, либо во время *загрузки*, либо во время *выполнения*
  - Практикуются названия: DLL-ки, `.so`-шники

# Динамические (разделяемые) библиотеки

- Динамическая компоновка происходит когда исполняемый файл в первый раз загружается и начинает работать (компоновка во время загрузки).
  - В Linux наиболее распространено, автоматически выполняется динамическим компоновщиком (`ld-linux.so`).
  - Стандартная библиотека языка Си (`libc.so`) обычно компонуется динамически.
- Динамическая компоновка может происходить когда программа уже работает (динамическая загрузка библиотек).

```
#include <dlfcn.h>
void *dlopen(const char *filename, int flag);
```

- Функции динамических библиотек могут одновременно использоваться несколькими процессами.

# Проблемы

- Динамическая библиотека может быть размещена в произвольном месте памяти
- Как обращаться из основной программы к переменным и функциям, размещение которых в памяти будет известно только в момент запуска программы?
  - До момента компоновки неизвестно, куда ведет ссылка – в перемещаемый код другой единицы трансляции или в динамическую библиотеку
  - Перемещаемый код с ссылками уже построен, в нем могут меняться только значения операндов (перебазируемые ссылки)
    - В случае динамической загрузки, адреса размещения будут определены еще позже – в уже работающей программе
- Как строить код динамической библиотеки, когда до момента начала выполнения ее кода неизвестно, по каким адресам будут размещены ее собственные функции и данные