Лекция 0xC

23 марта

ABI – двоичный интерфейс приложения

- Двоичный интерфейс приложения (application binary interface) задает правила взаимодействия между модулями программы на уровне исполняемого кода
  - Библиотеки и функции операционной системы (системные вызовы) также рассматриваются как модули, входящие в состав программы
- ABI определяет
  - Использование ISA и регистров процессора, организацию стека и т.п.
  - Соглашение вызова функций
  - Размещение данных в памяти, требования по выравниванию
  - Выполнение системных вызовов
  - Формат объектных файлов, состав библиотек (в случае, если ABI описывает всю среду выполнения программ)
- При соблюдении ABI можно разрабатывать модули программы на разных языках программирования

Пример ABI: http://www.sco.com/developers/devspecs/abi386-4.pdf

Вставляем элемент в непустой список,
Указатели на элемент и список не нулевые.

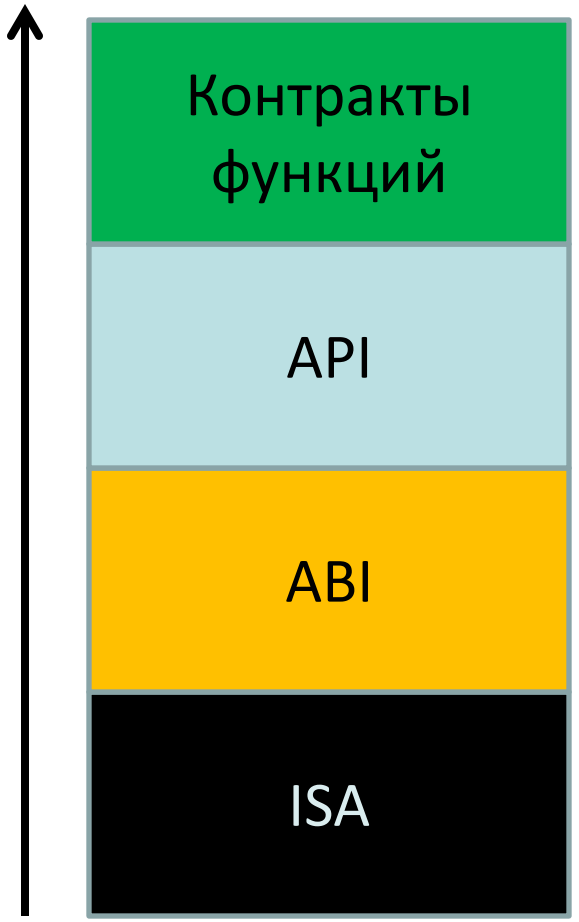
```
typedef struct chain chain;
struct chain {
   int    payload;
   chain *next;
};

chain* insert(chain* list, chain* elem) {
 if (0 == list->next) {
    ...
```

```
insert:
   push    ebx
   sub     esp, 8
   mov     ebx, dword [esp+16]
   mov     eax, dword [ebx+4]
   test    eax, eax
   je      .L22
   ...
```

```
esp   ← esp-4
[esp] ← ebx
...
```

Уровни интерфейсов



Функция main

```
#include <stdio.h>

void nullify(int argc, char* argv[]) {
}

int main(int argc, char* argv[]) {
   nullify(argc, argv);
   return 0;
}
```

- Есть ли отличия у функции main от остальных функций?
- Откуда берутся параметры argc и argv ?
- Что происходит со стеком в функции main ?

и наконец …

- Как можно воспользоваться стандартной библиотекой языка Си в ассемблерной программе?

```
snoop@earth:~/samples$ gcc -g -o main main.c
snoop@earth:~/samples$ gdb main
```

```
(gdb) br main
Breakpoint 1 at 0x80483f3: file main.c, line 7.
(gdb) run
Starting program: /home/snoop/samples/main

Breakpoint 1, main (argc=1, argv=0xbffff184) at main.c:7
7           nullify(argc, argv);
(gdb) bt
#0  main (argc=1, argv=0xbffff184) at main.c:7
(gdb) set backtrace past-main on
(gdb) bt
#0  main (argc=1, argv=0xbffff184) at main.c:7
#1  0xb7e1972e in __libc_start_main (main=0x80483f0 <main>, argc=1,
argv=0xbffff184, init=0x8048410 <__libc_csu_init>,
    fini=0x8048470 <__libc_csu_fini>, rtld_fini=0xb7feb210
<_dl_fini>, stack_end=0xbffff17c) at libc-start.c:289
#2  0x08048311 in _start ()
(gdb)
```

```
snoop@jezek:~/samples$ gcc -v
...
Target: i686-linux-gnu
...
gcc version 4.9.2 (Ubuntu 4.9.2-10ubuntu13)
```

```
snoop@earth:~/samples$ nm main
...
08048470 T __libc_csu_fini
08048410 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080483f0 T main
080483eb T nullify
...
080482f0 T _start
...
```

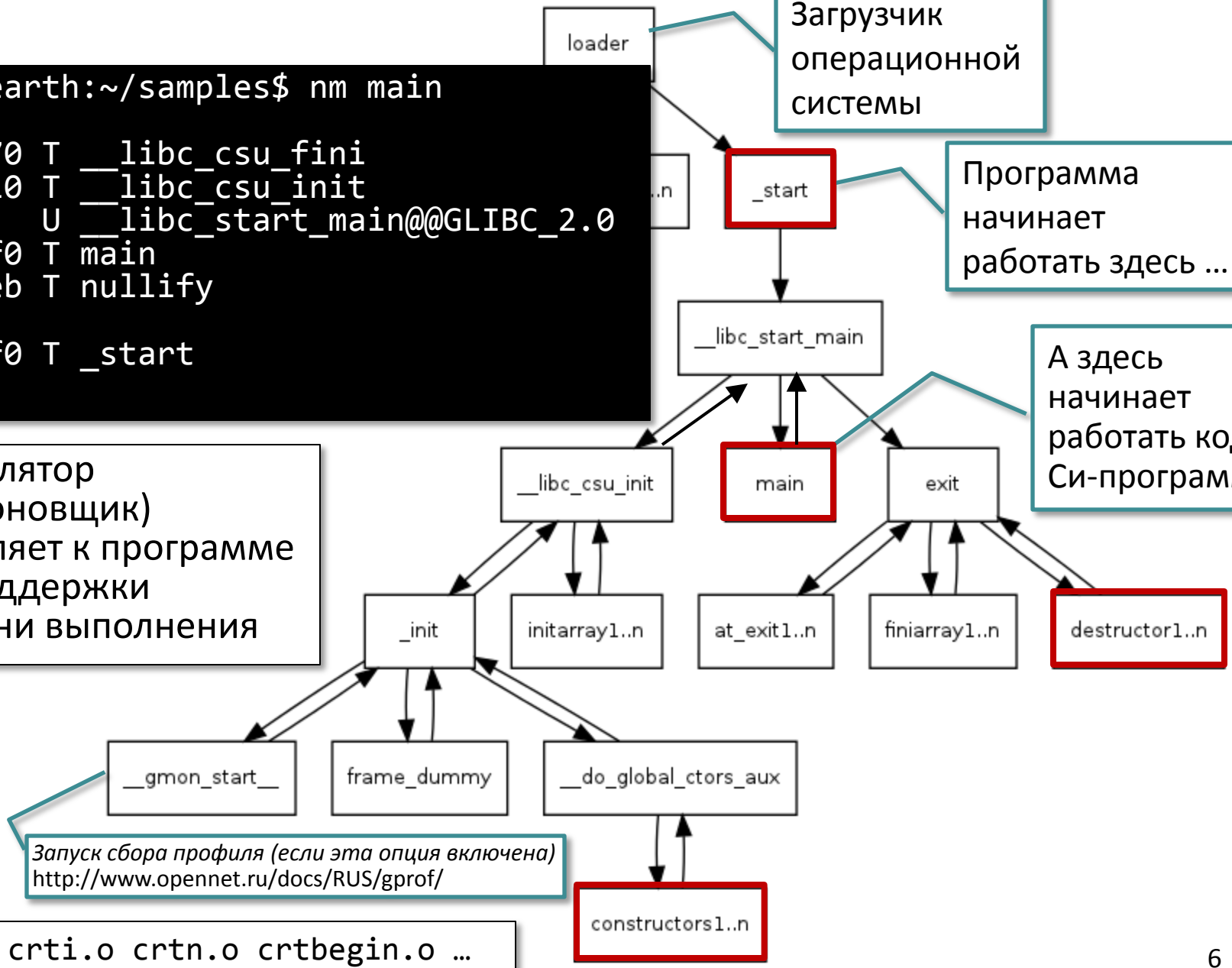
Загрузчик операционной системы

Программа начинает работать здесь …

А здесь начинает работать код Си-программы

Компилятор (компоновщик) добавляет к программе код поддержки времени выполнения

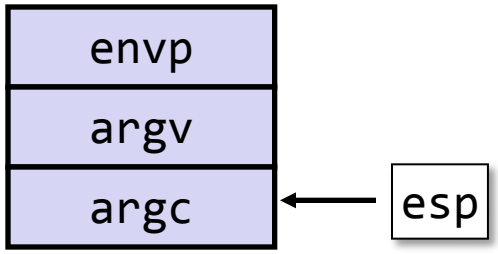
```
crt1.o crti.o crtn.o crtbegin.o …
```



Запуск сбора профиля (если эта опция включена)
http://www.opennet.ru/docs/RUS/gprof/

Функция _start

Начальное состояние стека




```
080482f0 <_start>:
 80482f0:   31 ed              xor    ebp,ebp
 80482f2:   5e                 pop    esi
 80482f3:   89 e1              mov    ecx,esp
 80482f5:   83 e4 f0           and    esp,0xfffffff0
 80482f8:   50                 push   eax
 80482f9:   54                 push   esp
 80482fa:   52                 push   edx
 80482fb:   68 70 84 04 08     push   0x8048470
 8048300:   68 10 84 04 08     push   0x8048410
 8048305:   51                 push   ecx
 8048306:   56                 push   esi
 8048307:   68 f0 83 04 08     push   0x80483f0
 804830c:   e8 cf ff ff ff     call   80482e0 <__libc_start_main@plt>
 8048311:   f4                 hlt
 8048312:   66 90              nop
 ...
```

Выравнивание стека (границ фрейма) по 16 байт

Адрес функции main

```
snoop@earth:~/samples$ objdump -M intel -d main
```

Функция _start

```
int __libc_start_main( int (*main) (int, char **, char **),
                                                  int argc,
                                        char ** ubp_av,
                                   void (*init) (void),
                                   void (*fini) (void),
                              void (*rtld_fini) (void),
                                   void (* stack_end));
```

```
08048300 <_start>:
 xor    ebp,ebp
 pop    esi
 mov    ecx,esp
 and    esp,0xfffffff0
 push   eax
 push   esp
 push   edx
 push   0x8048470
 push   0x8048410
 push   ecx
 push   esi
 push   0x80483f0
 call   80482e0 <__libc_start_main@plt>
 hlt
 nop
 ...
```

Функция main

ассемблерный листинг адаптирован под синтаксис nasm

```
080483f0 <main>:
 80483f0:      55                      push   ebp
 80483f1:      89 e5                   mov    ebp,esp
 80483f3:      ff 75 0c                push   dword [ebp+0xc]
 80483f6:      ff 75 08                push   dword [ebp+0x8]
 80483f9:      e8 ed ff ff ff          call   80483eb <nullify>
 80483fe:      83 c4 08                add    esp,0x8
 8048401:      b8 00 00 00 00          mov    eax,0x0
 8048406:      c9                      leave
 8048407:      c3                      ret
 8048408:      66 90                   nop
 ...
```

```
int main(int argc, char* argv[]) {
   nullify(argc, argv);
   return 0;
}
```

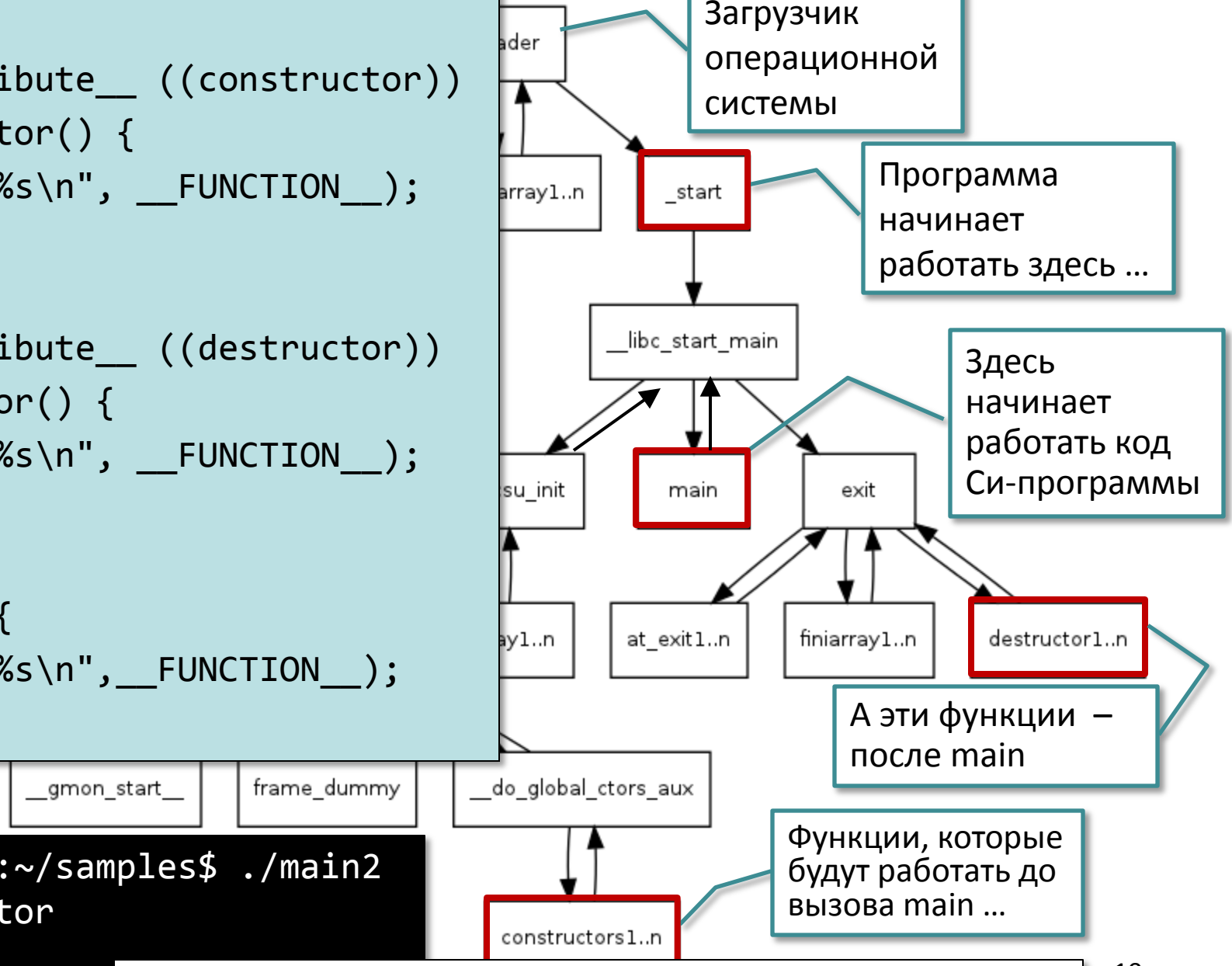
В зависимости от версии компилятора gcc выравнивание стека может выполняться в функции _start и/или main

```c
#include <stdio.h>

void __attribute__ ((constructor))
my_constructor() {
   printf("%s\n", __FUNCTION__);
}

void __attribute__ ((destructor))
my_destructor() {
   printf("%s\n", __FUNCTION__);
}

int main() {
   printf("%s\n",__FUNCTION__);
}
```




```
snoop@earth:~/samples$ ./main2
my_constructor
main
my_destructor
```

Идея метода/функции-конструктора не принадлежит Си++

```
snoop@earth:~/samples$ objdump -M intel -d main2
```

```
0804842b <my_destructor>:
 804842b:       55                      push   ebp
 804842c:       89 e5                   mov    ebp, esp
 804842e:       83 ec 18                sub    esp, 0x18
 8048431:       c7 44 24 08 49 85 04    mov    dword [esp+0x8], 0x8048549
 8048438:       08
 8048439:       c7 44 24 04 40 85 04    mov    dword [esp+0x4], 0x8048540
 8048440:       08
 8048441:       c7 04 24 01 00 00 00    mov    dword [esp], 0x1
 8048448:       e8 df fe ff ff          call   804832c <__printf_chk@plt>
 804844d:       c9                      leave
 804844e:       c3                      ret

0804844f <my_constructor>:
 804844f:       55                      push   ebp
 8048450:       89 e5                   mov    ebp,esp
 8048452:       83 ec 18                sub    esp,0x18
 8048455:       c7 44 24 08 57 85 04    mov    dword  [esp+0x8], 0x8048557
 804845c:       08
 804845d:       c7 44 24 04 40 85 04    mov    dword  [esp+0x4],0x8048540
 8048464:       08
 8048465:       c7 04 24 01 00 00 00    mov    dword [esp],0x1
 804846c:       e8 bb fe ff ff          call   804832c <__printf_chk@plt>
 8048471:       c9                      leave
 8048472:       c3                      ret
```

ассемблерный листинг адаптирован под синтаксис nasm

gcc version 4.4.3


```
snoop@earth:~/samples$ objdump -h main2
main2:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
...
 13 .text         000001cc  08048350  08048350  00000350  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
...
 17 .ctors        0000000c  08049f04  08049f04  00000f04  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 18 .dtors        0000000c  08049f10  08049f10  00000f10  2**2
                  CONTENTS, ALLOC, LOAD, DATA
...
```

```
snoop@earth:~/samples$ objdump -s -j .ctors -j .dtors main2
main2:     file format elf32-i386

Contents of section .ctors:
 8049f04 ffffffff 4f840408 00000000            ....O.......
Contents of section .dtors:
 8049f10 ffffffff 2b840408 00000000            ....+.......
```

В более поздних версиях gcc таблицы с адресами конструкторов и деструкторов были перенесены в секции `.init_array` и `.fini_array`

Функция `main` с выравниванием стека

ассемблерный листинг адаптирован под синтаксис nasm

```
08048360 <main>:
 8048360:      8d 4c 24 04              lea    ecx, [esp+0x4]
 8048364:      83 e4 f0                 and    esp, 0xfffffff0
 8048367:      ff 71 fc                 push   dword [ecx-0x4]
 804836a:      55                       push   ebp
 804836b:      89 e5                    mov    ebp, esp
 804836d:      51                       push   ecx
 804836e:      83 ec 10                 sub    esp, 0x10
 8048371:      68 10 85 04 08           push   0x8048510
 8048376:      e8 75 ff ff ff           call   80482f0 <puts@plt>
 804837b:      8b 4d fc                 mov    ecx, dword [ebp-0x4]
 804837e:      83 c4 10                 add    esp, 0x10
 8048381:      c9                       leave
 8048382:      8d 61 fc                 lea    esp, [ecx-0x4]
 8048385:      c3                       ret
```

Оболочка вокруг `main`

- **Выравнивание стека**
  - **Каждый вызов функции происходит на выровненном стеке**
  - **Необходимо формировать каждый фрейм таким образом, чтобы однажды выполненное выравнивание сохранялось**
- Поддержка повышенного уровня привилегий для данного запуска программы
- Поддержка многопоточного выполнения
- Запуск сбора профиля и запись в файл результатов профилирования после завершения работы деструкторов
  - Если программа была собрана с данной опцией
- Запуск конструкторов
- Запуск функции `main` с аргументами `argc` и `argv`
- Запуск деструкторов
- Передача результата функции `main` в функцию `exit`

Пример вызова malloc

```
#include <stdlib.h>

struct chain;

typedef struct chain {
   int val;
   struct chain *next;
} t_chain, *p_chain;
```

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

Пример вызова malloc

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
%include 'io.inc'

section .text

CEXTERN malloc

insert:
   push    ebp
   mov     ebp, esp
   sub     esp, 24
   ; ...
```

Пример вызова malloc

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
insert:
   ; ...
   mov      dword [ebp-4], esi
   mov      esi, dword [ebp+8]
   mov      dword [ebp-8], ebx
   mov      ebx, dword [ebp+12]
   ; ...
```

Пример вызова malloc

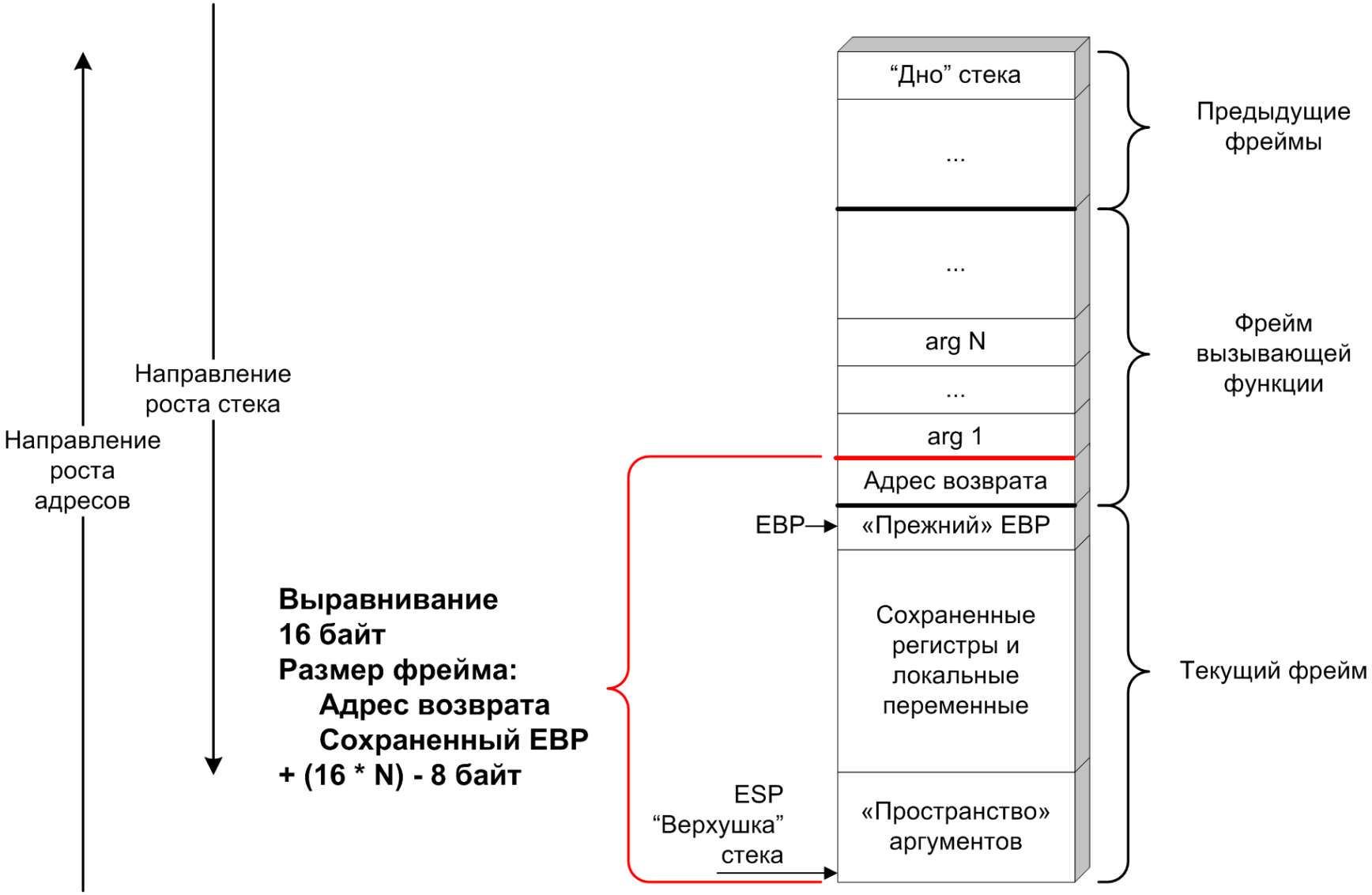
```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
   test    esi, esi
   je      .L2
   cmp     dword [esi], ebx
   jle     .L3
.L2:
   mov     dword [esp], 8
   call    malloc
   mov     dword [eax], ebx
   mov     dword [eax+4], esi
   mov     ebx, dword [ebp-8]
   mov     esi, dword [ebp-4]
   mov     esp, ebp
   pop     ebp
   ret
.L3:
```

Пример вызова malloc

```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
   ; …
.L3:
   mov      dword [esp+4], ebx
   mov      dword [esp], esi
   call     insert
   mov      dword [esi+4], eax
   mov      eax, esi
   mov      ebx, dword [ebp-8]
   mov      esi, dword [ebp-4]
   mov      esp, ebp
   pop      ebp
   ret
```



Выравнивание
16 байт
Размер фрейма:
Адрес возврата
Сохраненный EBP
+ (16 * N) - 8 байт

Стандартная библиотека языка Си

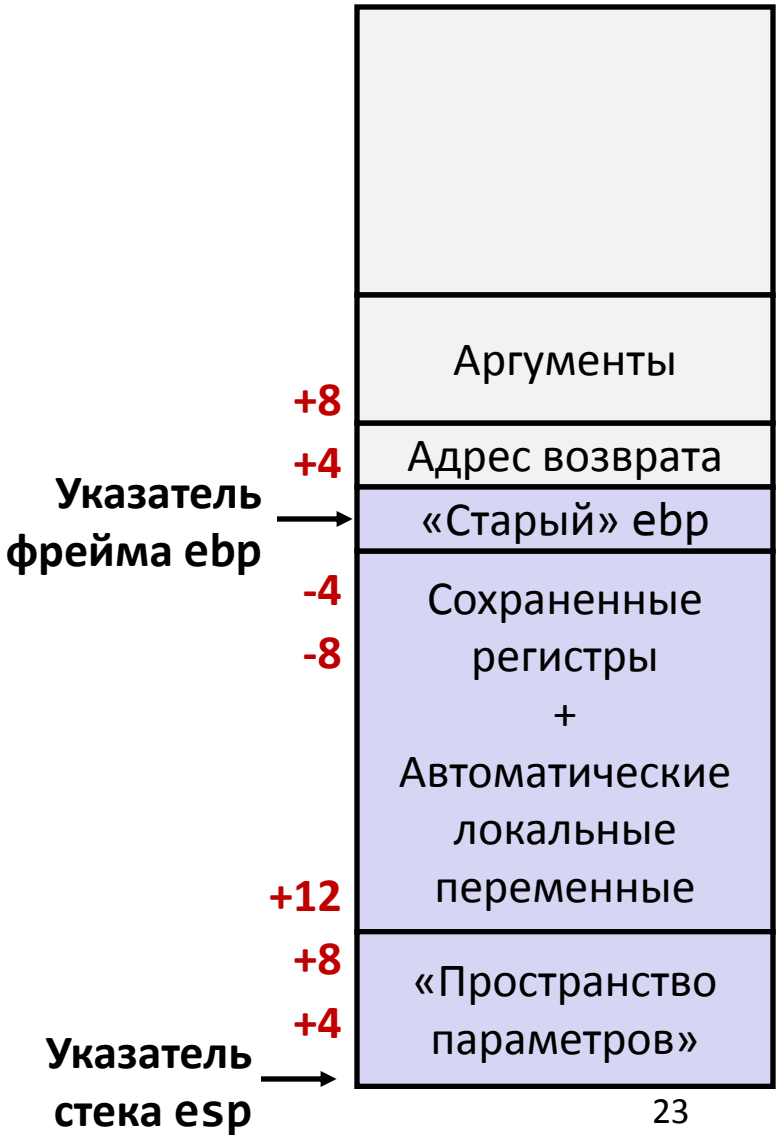
- 24 заголовочных файла
- stdlib.h
  - Преобразование типов: atoi, strtod, …
  - Генерация псевдослучайных последовательностей
  - Выделение и освобождение памяти
  - Сортировка и поиск
  - Математика
- stdio.h
  - Функции для файловых операций
  - Функции для операций ввода-вывода
- string.h
- …

Выравнивание стека

- На стек помещаем только машинные слова (4 байта)
- Выравнивание стека по границе в 16 байт IA-32/Linux/gcc
  - выполняется в функции main
  - остальные функции поддерживают выравнивание, формируя фрейм определенного размера
  - для листовых функций необязательно
- Не только производительность: команда `MOVDQA`
  - SSE2, Pentium4, 2001 г.
  - Быстрое копирование блока данных размером 16 байт
  - Аварийное завершение работы (#GP), если начальный адрес блока не выровнен по границе в 16 байт

Отказ от указателя фрейма

- Пролог и эпилог функций вносят накладные расходы
  - Особенно заметно на небольших функциях
- Содержимое фрейма распределяется во время компиляции
  - как правило, все размещаемые внутри фрейма данные могут быть адресованы через константные смещения
- Используем для адресации только ESP
  - Порядок использования EBP – часть ABI (Application Binary Interface)
  - До 2011 года gcc не включал опцию `-fomit-frame-pointer` в общие списки оптимизаций



Отказ от указателя фрейма

```
gcc –S -Os -fomit-frame-pointer -fno-optimize-sibling-calls -masm=intel length.c
```

```
typedef struct link link;

struct link {
   int payload;
   link* next;
};

int length(link *p) {
   return p? length(p->next) + 1 : 0;
}
```

```
length:
   sub      esp, 12
   xor      eax, eax
   mov      edx, dword [esp+16]
   test     edx, edx
   je       .L2
   mov      ecx, dword [edx+4]
   mov      [esp], ecx
   call     length
   inc      eax
.L2:
   add      esp, 12
   ret
```

Отказ от указателя фрейма

```
gcc -S -Os -fno-omit-frame-pointer -fno
```

```
typedef struct link link;

struct link {
   int payload;
   link* next;
};

int length(link *p) {
   return p? length(p->next) + 1 : 0;
}
```

```
length:
   push    ebp
   xor     eax, eax
   mov     ebp, esp
   sub     esp, 8
   mov     edx, dword [ebp+8]
   test    edx, edx
   je      .L2
   mov     ecx, dword [edx+4]
   mov     dword [esp], ecx
   call    length
   inc     eax
.L2:
   leave
   ret
```

Какие есть компиляторы и соглашения о вызове функций?

- Компиляторы
  - LLVM clang (open source)
  - GNU gcc (open source)
  - Microsoft Visual Studio vc
  - Intel icc
- еще компиляторы …
  - PGI
  - C++Builder (Borland C++)
  - Open Watcom
  - Digital Mars
  - Codeplay
- Модели памяти
  - 16, 32, 64 разряда
- Операционные системы
  - Linux (FreeBSD, …)
  - Windows
  - Mac OS

Дополнительная техническая информация - домашняя страница Agner Fog: http://www.agner.org/optimize/ сборник материалов по компиляторам и оптимизации.
Calling conventions for different C++ compilers and operating systems

Соглашение STDCALL

```
#include <stdio.h>

__attribute__((stdcall))
int sum(int x, int y);

int main() {
   int a = 1, b = 2, c;
   c = sum(a, b);
   printf("%d\n", c);
   return 0;
}

__attribute__((stdcall))
int sum(int x, int y) {
   int t = x + y;
   return t;
}
```

Очистку стека от аргументов вызова выполняет сама вызванная функция

```
sum:
   push    ebp
   mov     ebp, esp
   mov     eax, dword [ebp+12]
   add     eax, dword [ebp+8]
   pop     ebp
   ret     8
```

```
CMAIN:
   ; …
   mov     dword [esp+4], 2
   mov     dword [esp], 1
   call    sum
   sub     esp, 8
   mov     dword [ebp-8], eax
   ; …
```

При вызове `stdcall` из функции, имеющей соглашение `cdecl`, необходим код для компенсации освобождения стека.

Соглашение FASTCALL

- Первый и второй параметры размещаются в регистрах ECX и EDX
  - Если размер параметров позволяет
- Остальные параметры – на стеке, от них стек очищает вызванная функция, как и в stdcall
- Такая форма соглашения принята в gcc и MSVC

```
xorEmAll:
   test    ecx, ecx
   mov     eax, edx
   je      .L6
   push    ebx
   mov     ebx, ecx
   sub     esp, 8
   mov     ecx, dword [ecx+4]
   call    xorEmAll
   xor     eax, dword [ebx]
   mov     dword [ebx], eax
   add     esp, 8
   pop     ebx
 .L6:
   ret
```

```
typedef struct chain chain;

struct chain {
    unsigned data;
    chain    *next;
};

__attribute__((fastcall)) unsigned xorEmAll(chain *p, unsigned salt) {
   if (p) {
      return p->data ^= xorEmAll(p->next, salt);
   } else {
      return salt;
   }
}
```