Лекция 13

26 марта

Возвращаемое значение - структура

```
typedef unsigned short uint16;

typedef struct {
   uint16 i;
   uint16 j;
   uint16 k;
} triple;

triple init_t() {
  return (triple){0xBADB, 0xADBA, 0xDBAD};
}

void h() {
   volatile triple t = init_t();
}
```

```
h:
   ???

init_t:
   ???
```

1. Реализуем h
   1. Распределяем память во фрейме
   2. Вычисляем значения аргументов вызова
   3. Вызываем init_t
2. Реализуем init_t

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 делается со стеком в функции main ?

и наконец …

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

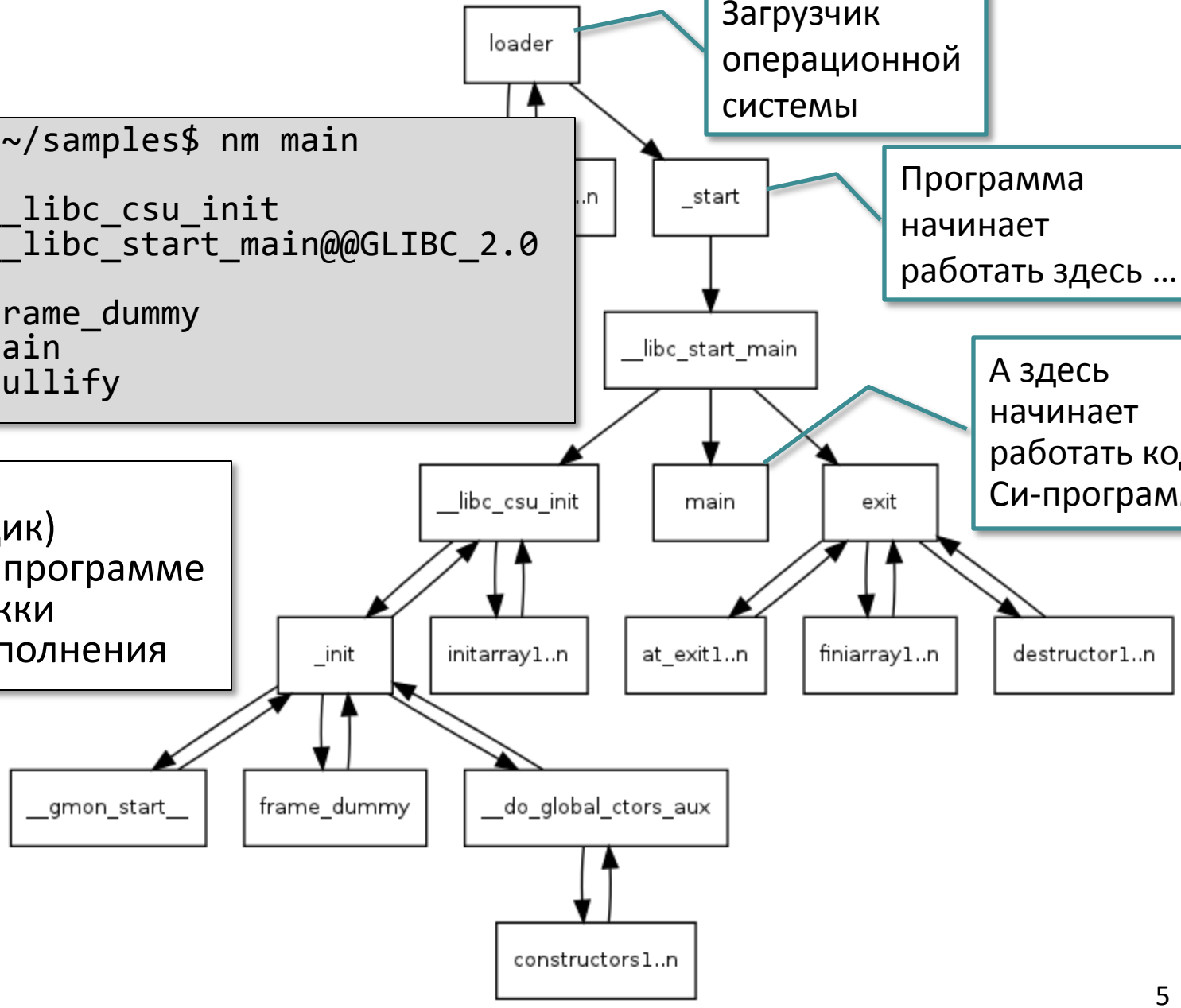
```
snoop@earth:~/samples$ gcc -g -o main main.c
snoop@earth:~/samples$ gdb main
(gdb) br main


Breakpoint 1 at 0x80483bf: file main.c, line 7.
(gdb) run
Starting program: /home/snoop/samples/main

Breakpoint 1, main (argc=1, argv=0xbffff804) at main.c:7
7           nullify(argc, argv);
(gdb) bt
#0  main (argc=1, argv=0xbffff804) at main.c:7
(gdb) set backtrace past-main on
(gdb) bt
#0  main (argc=1, argv=0xbffff804) at main.c:7
#1  0xb7e8dbd6 in __libc_start_main (main=0x80483b9 <main>, argc=1,
    ubp_av=0xbffff804, init=0x80483f0 <__libc_csu_init>,
    fini=0x80483e0 <__libc_csu_fini>, rtld_fini=0xb7ff1030
<_dl_fini>,
    stack_end=0xbffff7fc) at libc-start.c:226
#2  0x08048321 in _start ()
(gdb)
```

```
snoop@earth:~/samples$ nm main
...
080483f0 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
...
08048390 t frame_dummy
080483b9 T main
080483b4 T nullify
```

Компилятор (компоновщик) добавляет к программе код поддержки времени выполнения

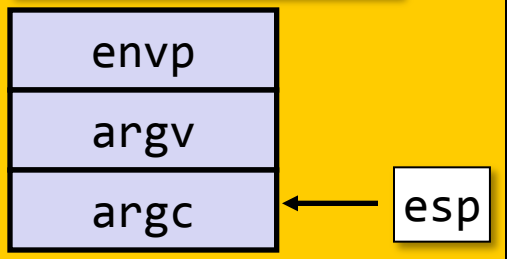


Функция _start

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

```
08048300 <_start>:
 8048300:  31 ed                xor    ebp,ebp
 8048302:  5e                   pop    esi
 8048303:  89 e1                mov    ecx,esp
 8048305:  83 e4 f0             and    esp,0xfffffff0
 8048308:  50                   push   eax
 8048309:  54                   push   esp
 804830a:  52                   push   edx
 804830b:  68 e0 83 04 08       push   0x80483e0
 8048310:  68 f0 83 04 08       push   0x80483f0
 8048315:  51                   push   ecx
 8048316:  56                   push   esi
 8048317:  68 b9 83 04 08       push   0x80483b9
 804831c:  e8 c7 ff ff ff       call   80482e8 <__libc_start_main@plt>
 8048321:  f4                   hlt
 8048322:  90                   nop
 ...
```

Начальное состояние стека



Функция _start

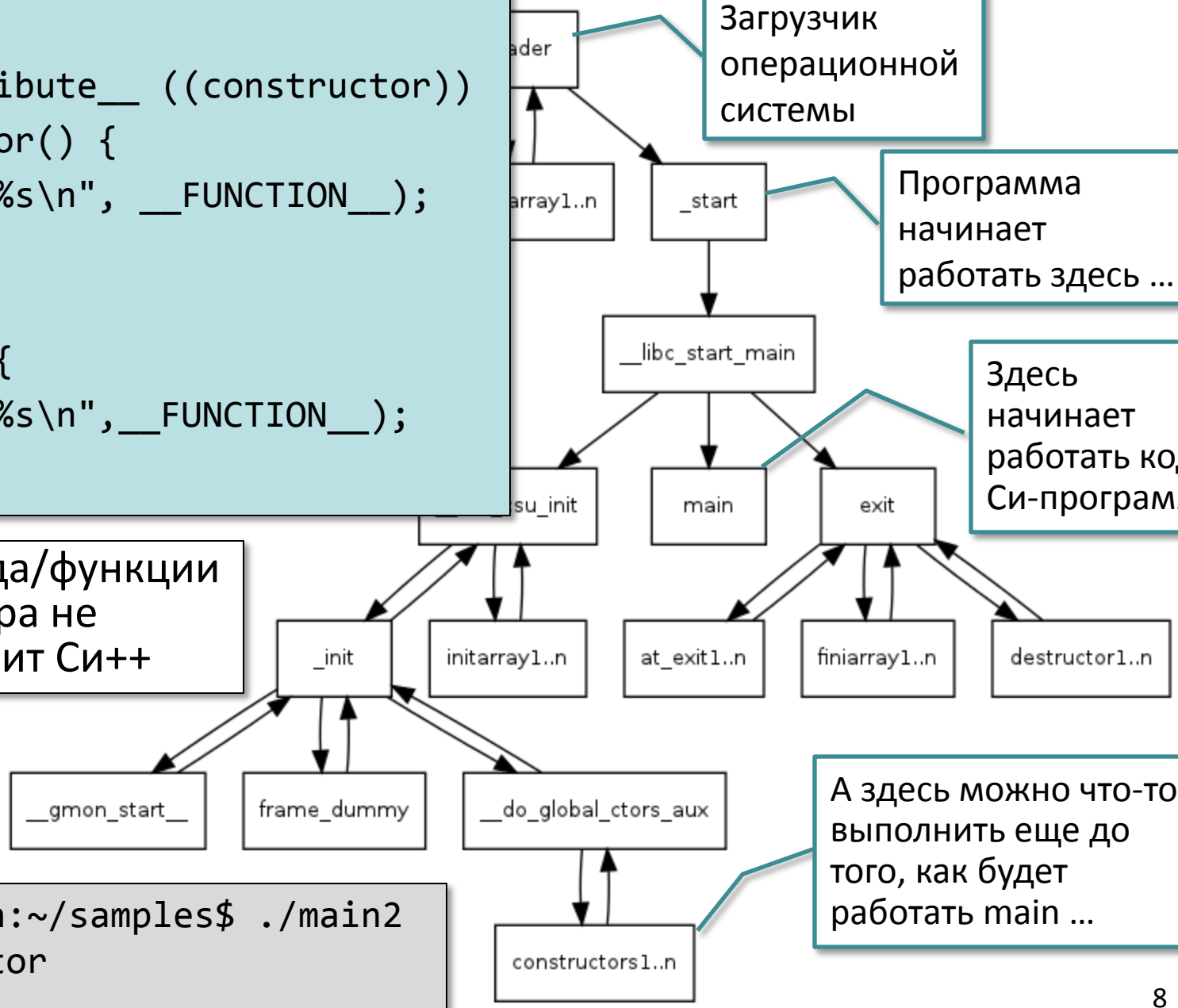
```
int __libc_start_main( int (*main) (int, char **, char **),
                                                  int argc,
                                          char ** ubp_av,
                                     void (*init) (void),
                                     void (*fini) (void),
                                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   0x80483e0
 push   0x80483f0
 push   ecx
 push   esi
 push   0x80483b9
 call   80482e8 <__libc_start_main@plt>
 hlt
 nop
 ...
```

```
#include <stdio.h>

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

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



Загрузчик операционной системы

Программа начинает работать здесь …

Здесь начинает работать код Си-программы

Идея метода/функции конструктора не принадлежит Си++

А здесь можно что-то выполнить еще до того, как будет работать main …

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

Функция main

```
080483b9 <main>:
 80483b9:       55                      push   ebp
 80483ba:       89 e5                   mov    ebp,esp
 80483bc:       83 ec 08                sub    esp,0x8
 80483bf:       8b 45 0c                mov    eax,dword [ebp+0xc]
 80483c2:       89 44 24 04             mov    dword [esp+0x4],eax
 80483c6:       8b 45 08                mov    eax,dword [ebp+0x8]
 80483c9:       89 04 24                mov    dword [esp],eax
 80483cc:       e8 e3 ff ff ff          call   80483b4 <nullify>
 80483d1:       b8 00 00 00 00          mov    eax,0x0
 80483d6:       c9                      leave
 80483d7:       c3                      ret
```

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

В предыдущих версиях компилятора gcc выравнивание стека происходило в функции main

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

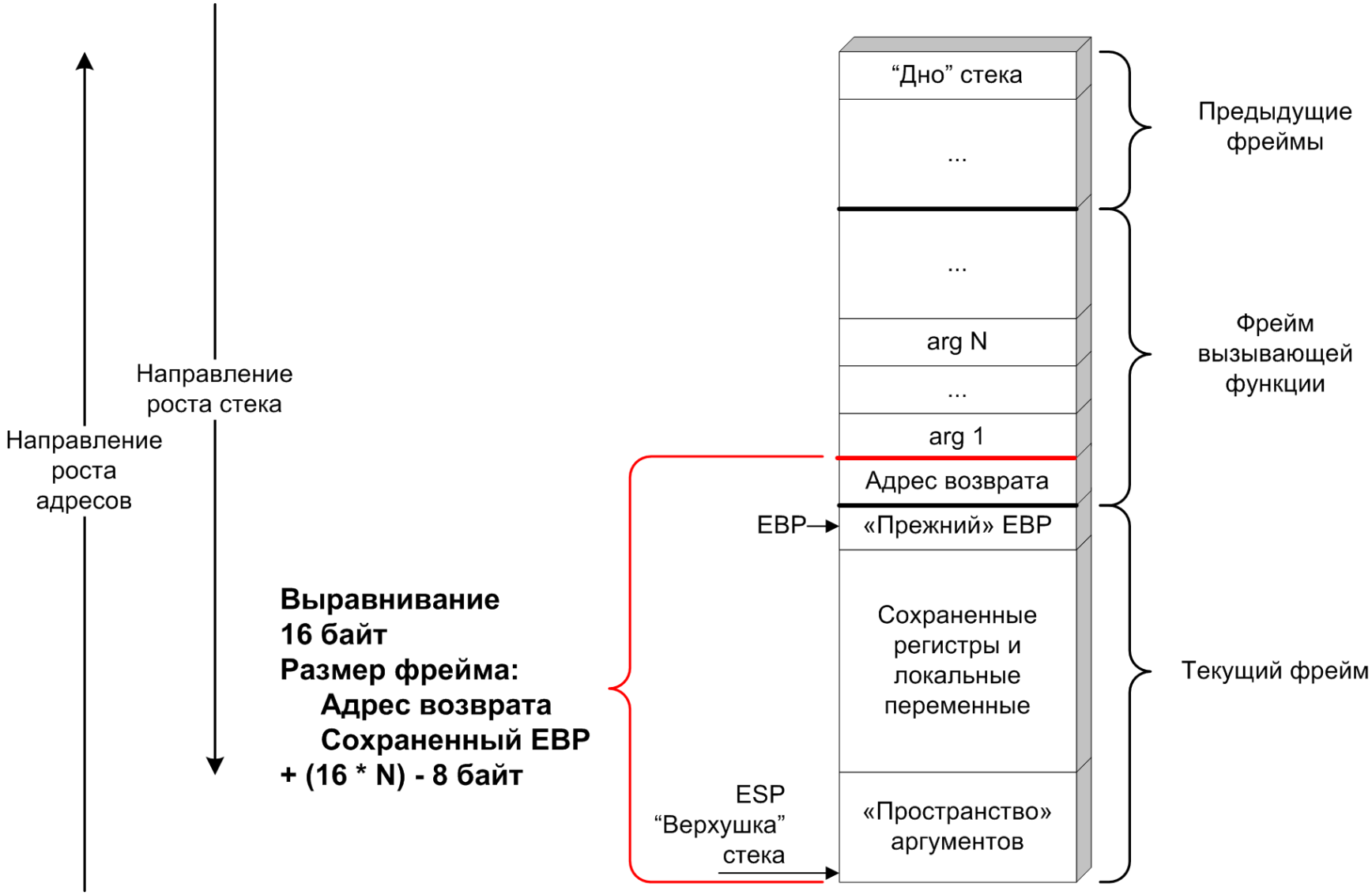
```
p_chain insert(p_chain p, int val) {
   if ((0 == p) || (p->val > val)) {
      p_chain np =
         (p_chain)malloc(sizeof(t_chain));
      np->val = val;
      np->next = p;
      return np;
   } else {
      p->next = insert(p, val);
      return p;
   }
}
```

```
   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
```



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
- …

Соглашение STDCALL

```c
#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;
}
```

```asm
sum:
   push    ebp
   mov     ebp, esp
   sub     esp, 16
   mov     edx, dword [ebp+12]
   mov     eax, dword [ebp+8]
   add     eax, edx
   mov     dword [ebp-4], eax
   mov     eax, dword [ebp-4]
   leave
   ret     8
```

Соглашение STDCALL

```c
#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;
}
```

```asm
CMAIN:
   ; …
   mov     eax, dword [ebp-12]
   mov     dword [esp+4], eax
   mov     eax, dword [ebp-16]
   mov     dword [esp], eax
   call    sum
   sub     esp, 8
   mov     dword [ebp-8], eax
   ; …
```

Соглашение FASTCALL

```c
#include <stdio.h>

__attribute__((fastcall))
int sub(int x, int y);

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

__attribute__((fastcall))
int sub(int x, int y) {
   int t = x - y;
   return t;
}
```

```asm
CMAIN:
   ; …
   mov     edx, 2
   mov     ecx, 1
   call    sub
   mov     dword [esp+20], eax
   ; …

sub:
   sub     ecx, edx
   mov     eax, ecx
   ret
```

Выравнивание стека - итоги

- 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
  - Быстрое копирование блока данных размером 16 байт
  - Аварийное завершение работы (#GP), если начальный адрес блока не выровнен по границе в 16 байт