Язык C++

Работа с памятью

Работа программ

- Архитектура Фон Неймана\Гарвардская
- Виды памяти
- Процессор
- Прерывания

Процессы\потоки

Процессы

- Независимое адресное пространство
- Объекты ядра (файловые дескрипторы, объекты синхронизации и т.д.)

Потоки

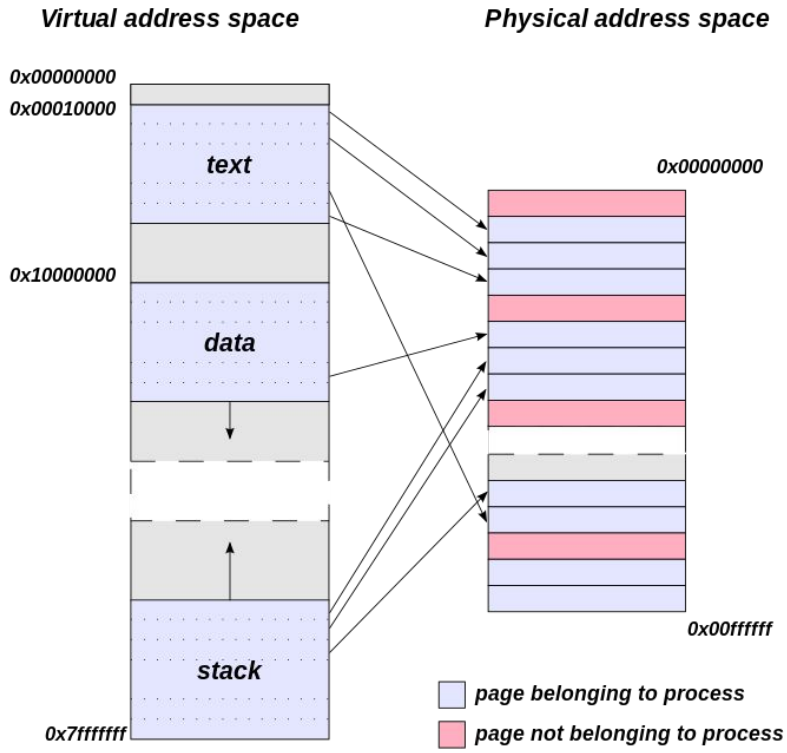
- Набор команд
- Стек

Виртуальное адресное пространство

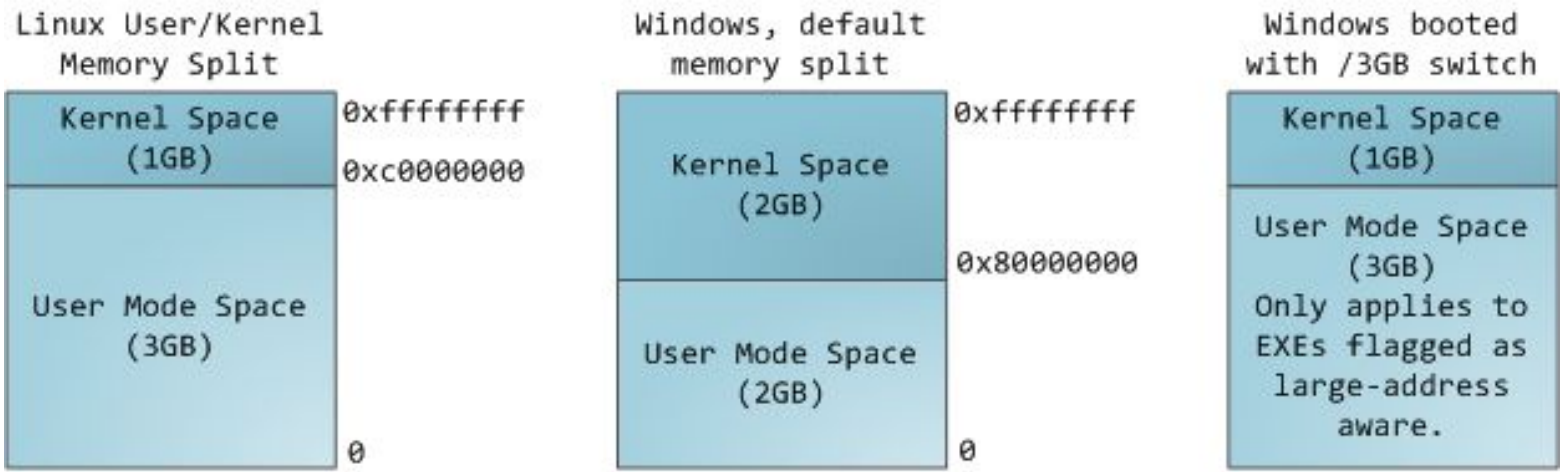
1. У каждого процесса “своя” память
2. Иллюзия доступности всех ресурсов
3. Осуществляется мапинг на физическую память
4. Page Table
5. Segments
6. ОС также реализует данную логику

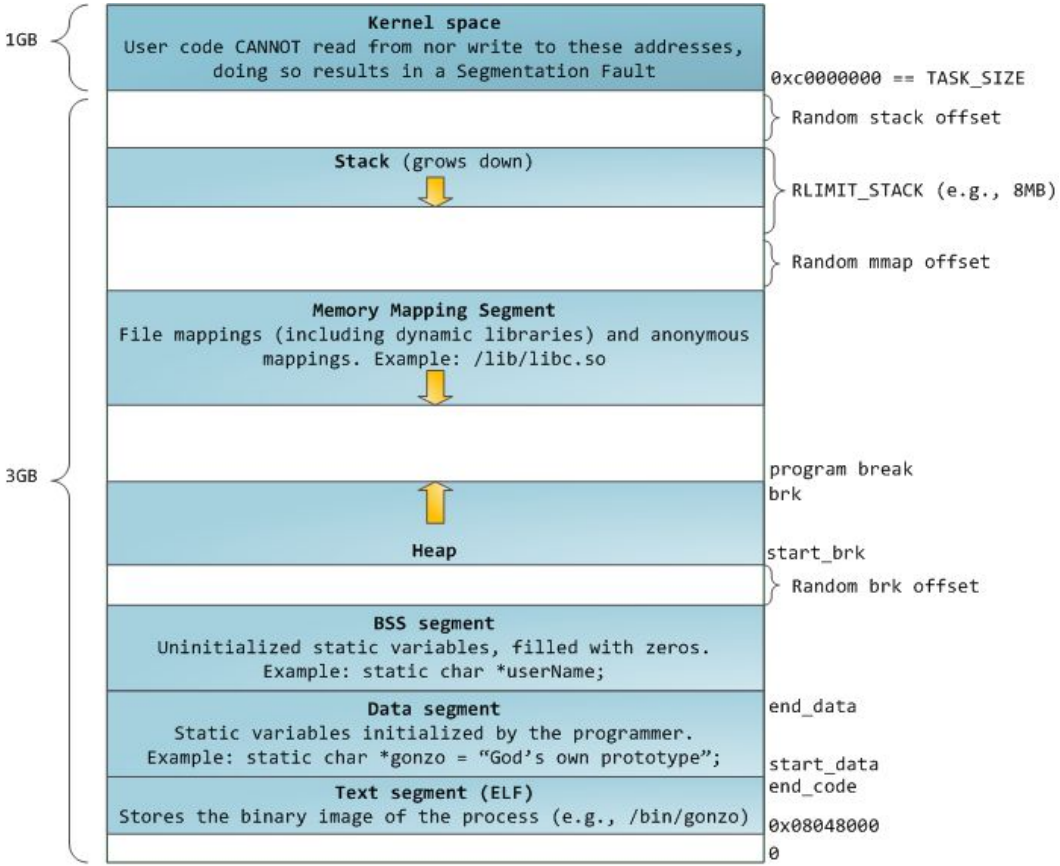
Page table

- Маппинг виртуального адреса на физически
- Изоляция процессов
- Memory-mapped file
- Обеспечение безопасного режима работы ОС
- swapping



Представление программы в памяти




1GB
3GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;
Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = "God's own prototype";
end_data
start_data
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
end_code
0x08048000
0

Segments

1. Stack
2. Heap
3. Memory Mapping
4. BSS
5. Data
6. Text

Segments

```
int main(int argc, char* argv[]) {
   int i = 0;
   char* str = "Hello world";

   std::printf("Process id: %d\n", getpid());

   std::printf("Data segment: %p\n", &PI);
   std::printf("BSS segment %p\n", &SomeGlobalValue);
   std::printf("Text segment %p\n", str);
   std::printf("Code segment %p\n", &SomeFunc);
   std::printf("Stack segment %p\n", &i);

   getchar();
   return 0;
}
```

Segments

```
hvost239@hvost239-ub16:~$ cat /proc/414335/maps
00400000-00401000 r--p 00000000 08:02 28974013                           /home/hvost239/ITMO/C/20_21/21.10.25/main
00401000-00402000 r-xp 00001000 08:02 28974013                           /home/hvost239/ITMO/C/20_21/21.10.25/main
00402000-00403000 r--p 00002000 08:02 28974013                           /home/hvost239/ITMO/C/20_21/21.10.25/main
00403000-00404000 r--p 00002000 08:02 28974013                           /home/hvost239/ITMO/C/20_21/21.10.25/main
00404000-00405000 rw-p 00003000 08:02 28974013                           /home/hvost239/ITMO/C/20_21/21.10.25/main
01bda000-01bfb000 rw-p 00000000 00:00 0                                  [heap]
7f2cc7874000-7f2cc7899000 r--p 00000000 08:02 24510538                   /lib/x86_64-linux-gnu/libc-2.31.so
7f2cc7899000-7f2cc7a11000 r-xp 00025000 08:02 24510538                   /lib/x86_64-linux-gnu/libc-2.31.so
7f2cc7a11000-7f2cc7a5b000 r--p 0019d000 08:02 24510538                   /lib/x86_64-linux-gnu/libc-2.31.so
7f2cc7a5b000-7f2cc7a5c000 ---p 001e7000 08:02 24510538                   /lib/x86_64-linux-gnu/libc-2.31.so
7f2cc7a5c000-7f2cc7a5f000 r--p 001e7000 08:02 24510538                   /lib/x86_64-linux-gnu/libc-2.31.so
7f2cc7a5f000-7f2cc7a62000 rw-p 001ea000 08:02 24510538                   /lib/x86_64-linux-gnu/libc-2.31.so
7f2cc7a62000-7f2cc7a68000 rw-p 00000000 00:00 0
7f2cc7a8b000-7f2cc7a8c000 r--p 00000000 08:02 24510512                   /lib/x86_64-linux-gnu/ld-2.31.so
7f2cc7a8c000-7f2cc7aaf000 r-xp 00001000 08:02 24510512                   /lib/x86_64-linux-gnu/ld-2.31.so
7f2cc7aaf000-7f2cc7ab7000 r--p 00024000 08:02 24510512                   /lib/x86_64-linux-gnu/ld-2.31.so
7f2cc7ab8000-7f2cc7ab9000 r--p 0002c000 08:02 24510512                   /lib/x86_64-linux-gnu/ld-2.31.so
7f2cc7ab9000-7f2cc7aba000 rw-p 0002d000 08:02 24510512                   /lib/x86_64-linux-gnu/ld-2.31.so
7f2cc7aba000-7f2cc7abb000 rw-p 00000000 00:00 0
7ffff3c0c000-7ffff3c2e000 rw-p 00000000 00:00 0                          [stack]
7ffff3cd1000-7ffff3cd4000 r--p 00000000 00:00 0                          [vvar]
7ffff3cd4000-7ffff3cd5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Stack (Стек вызова)

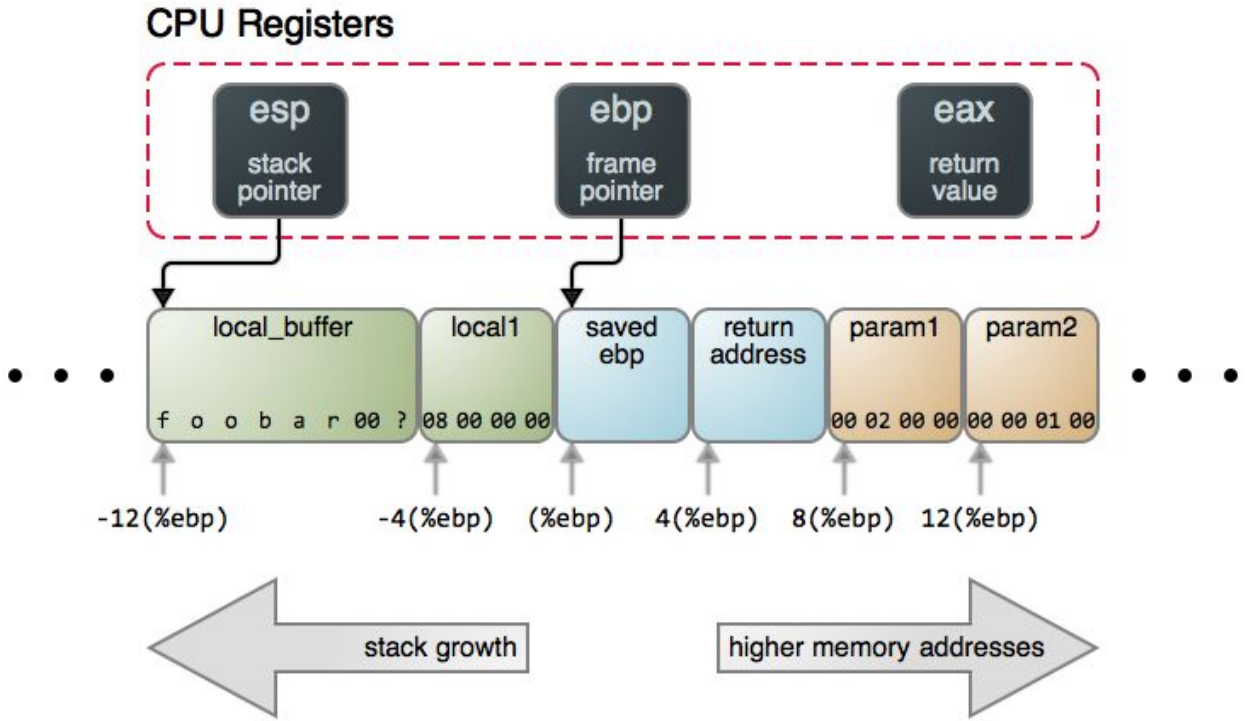
```
int add(int a, int b) {
    return a + b;
}

int main () {
    int result;
    result = add(100500, 1);
    return 0;
}
```

GodBolt.org

```
1   add:
2           push    rbp
3           mov     rbp, rsp
4           mov     DWORD PTR [rbp-4], edi
5           mov     DWORD PTR [rbp-8], esi
6           mov     edx, DWORD PTR [rbp-4]
7           mov     eax, DWORD PTR [rbp-8]
8           add     eax, edx
9           pop     rbp
10          ret
11  main:
12          push    rbp
13          mov     rbp, rsp
14          sub     rsp, 16
15          mov     esi, 1
16          mov     edi, 100500
17          call    add
18          mov     DWORD PTR [rbp-4], eax
19          mov     eax, 0
20          leave
21          ret
```

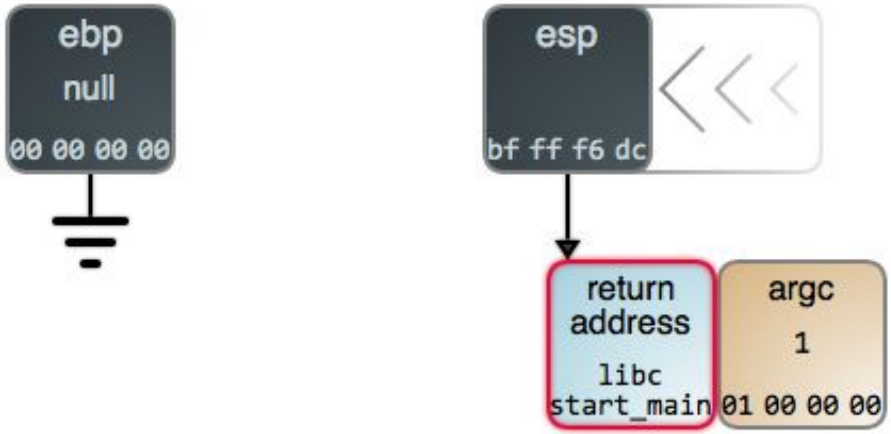
Стек вызова

1. StackFrame
   a. arguments
   b. local variable
   c. return point
2. cdecl, stdcall, fastcall
3. Регистры процессора
   a. esp (верхушка стека)
   b. ebp (начало кадра)
   c. eax (результат)

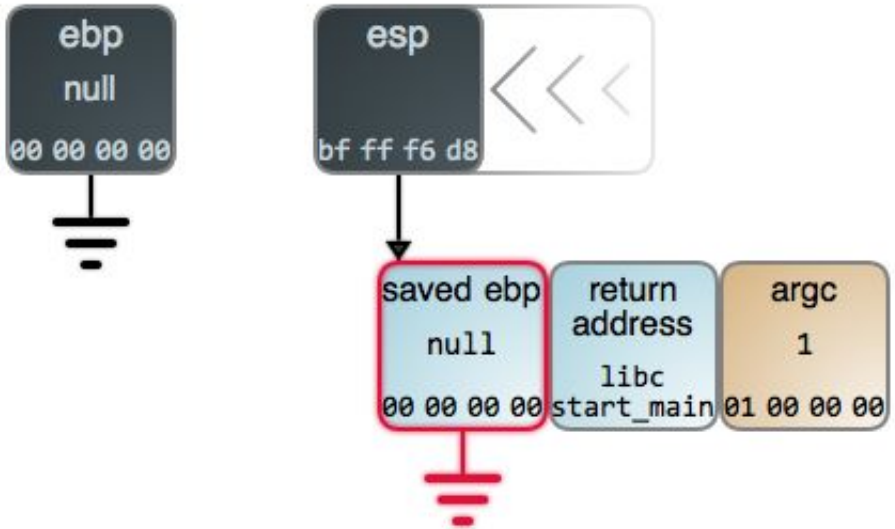

CPU Registers
esp
stack pointer
ebp
frame pointer
eax
return value
local_buffer
local1
saved ebp
return address
param1
param2
f o o b a r 00 ?
08 00 00 00
00 02 00 00
00 00 01 00
-12(%ebp)
-4(%ebp)
(%ebp)
4(%ebp)
8(%ebp)
12(%ebp)
stack growth
higher memory addresses

Источник

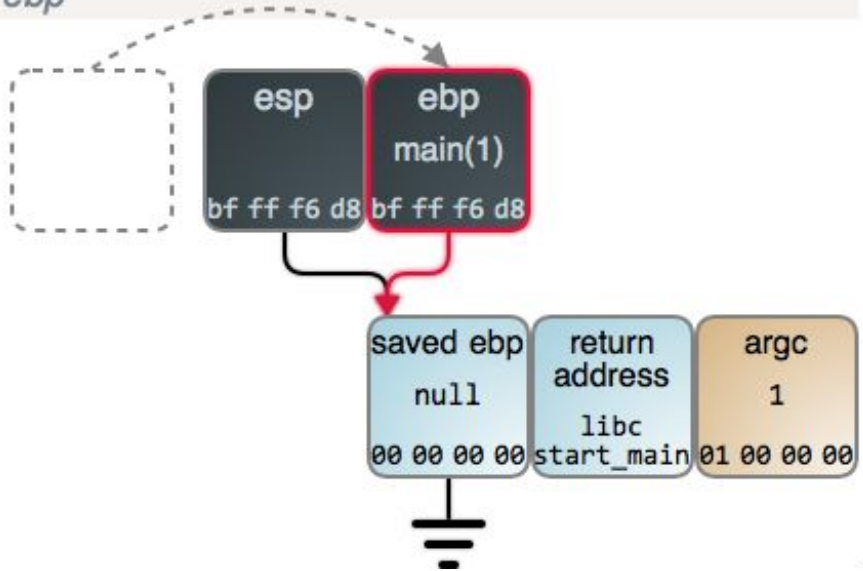
```
1.    call main # push return address onto stack, jump into main
```



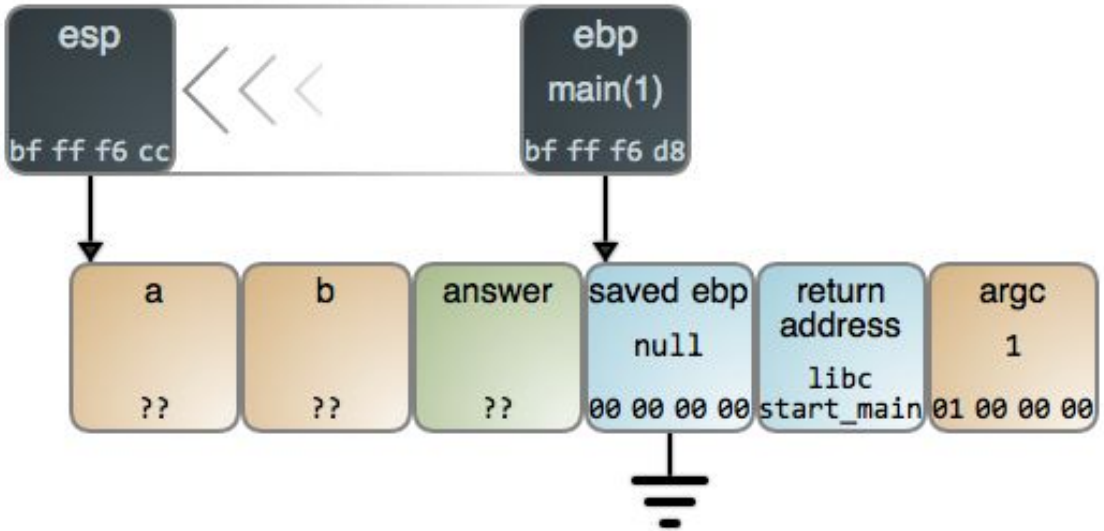
```
2.    pushl %ebp # save current ebp register value
```



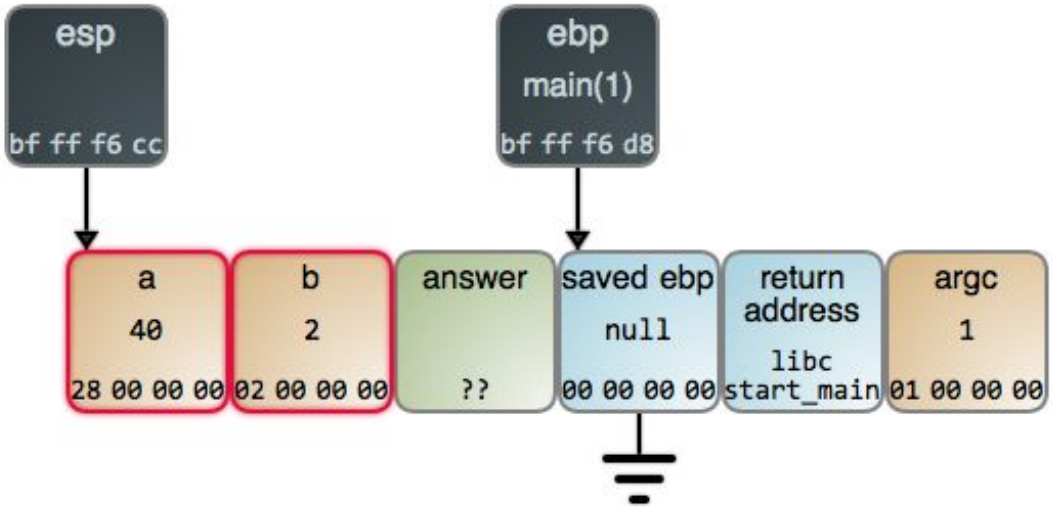
```
3.    movl %esp, %ebp # copy esp to ebp
```



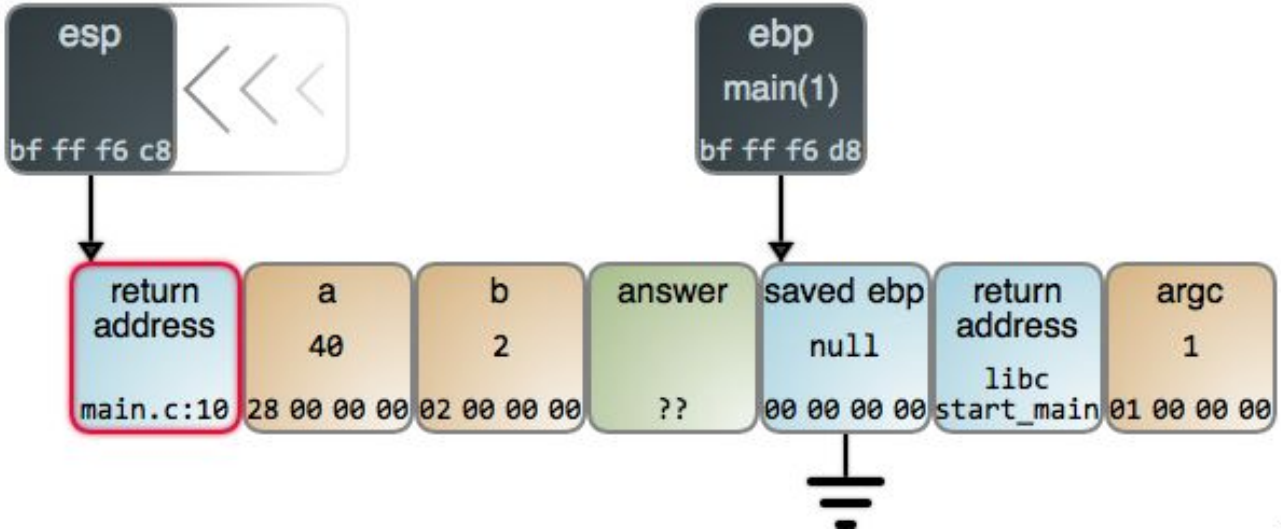
```
4.    subl $12, %esp # make room for stack data
```



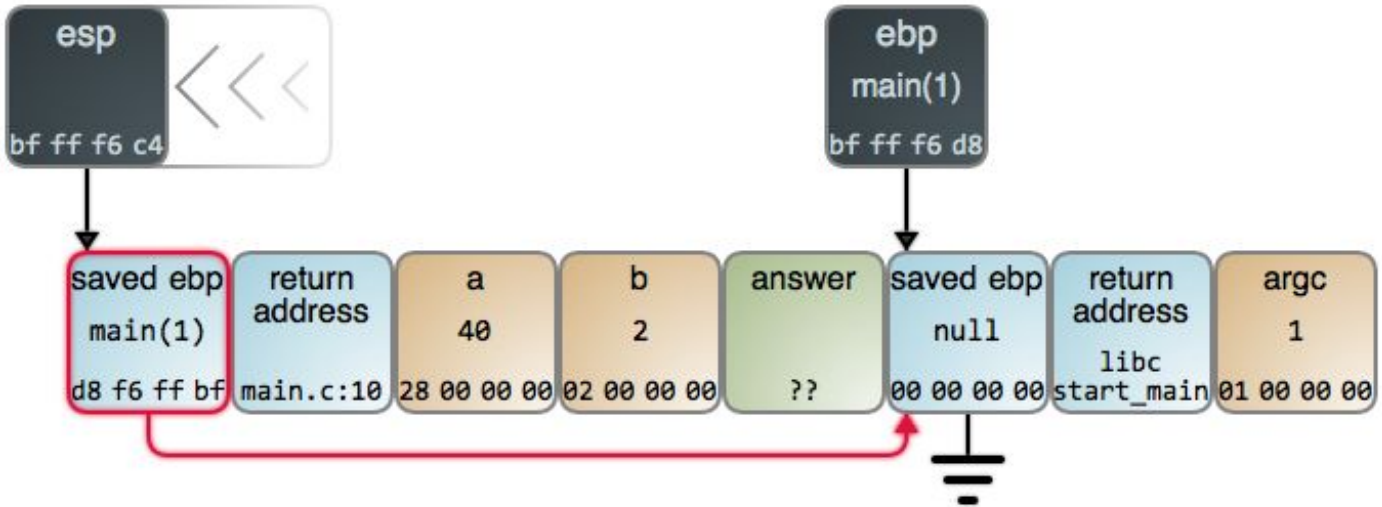
5.
```
movl $2, 4(%esp) # set b to 2
movl $40, (%esp) # set a to 40
```



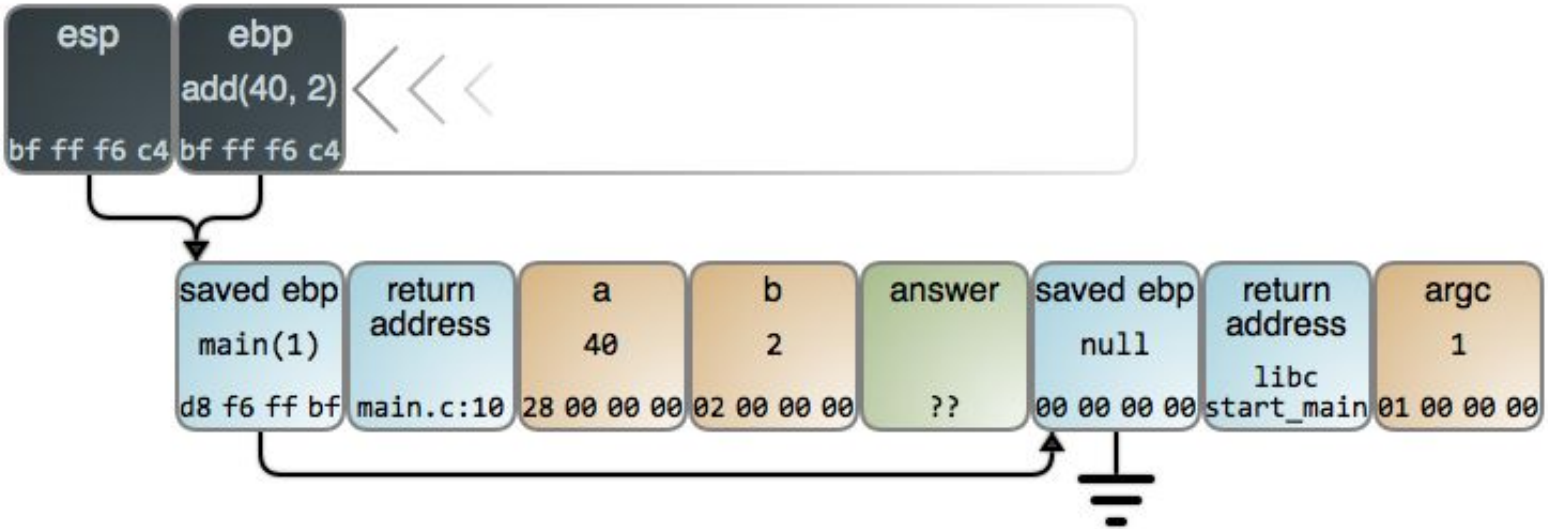
```
6.    call add  # push return address onto stack, jump into add
```



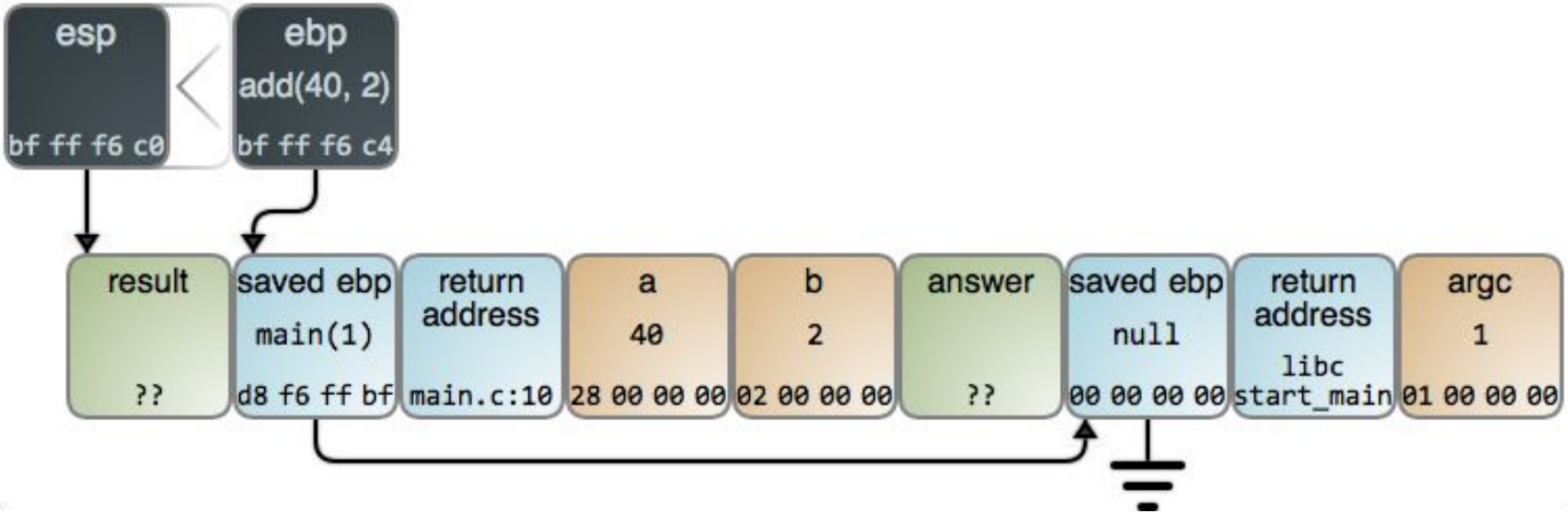
```
7.      pushl %ebp  # save current ebp register value
```



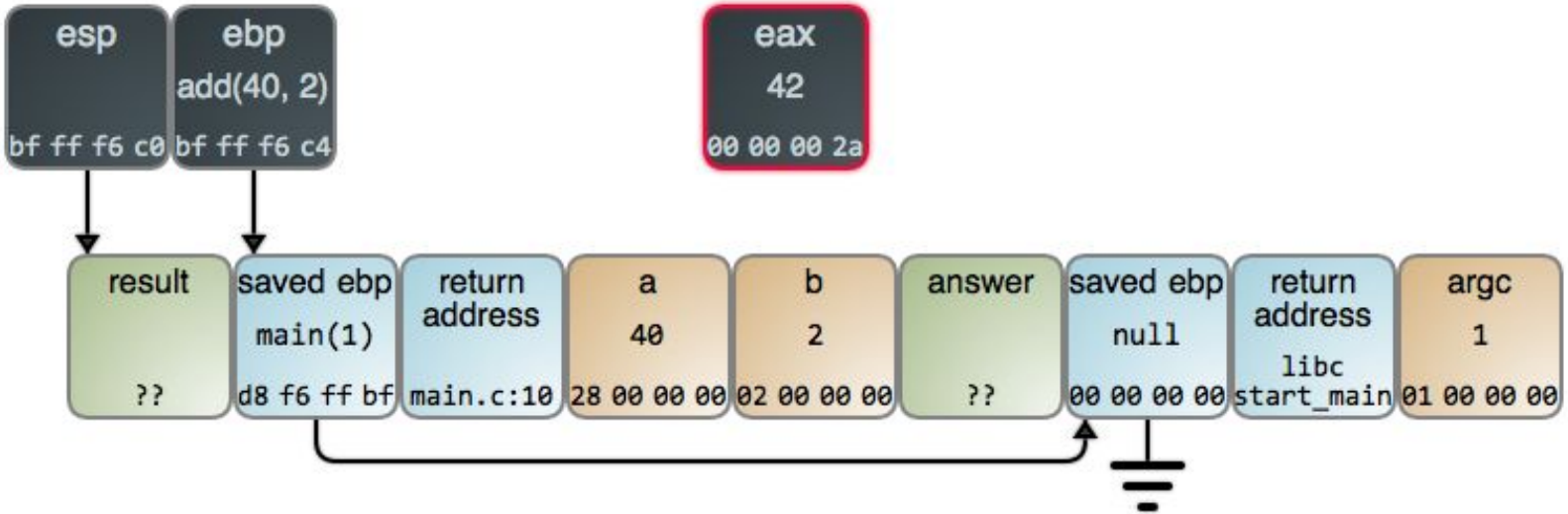
8. `movl %esp, %ebp # copy esp to ebp`



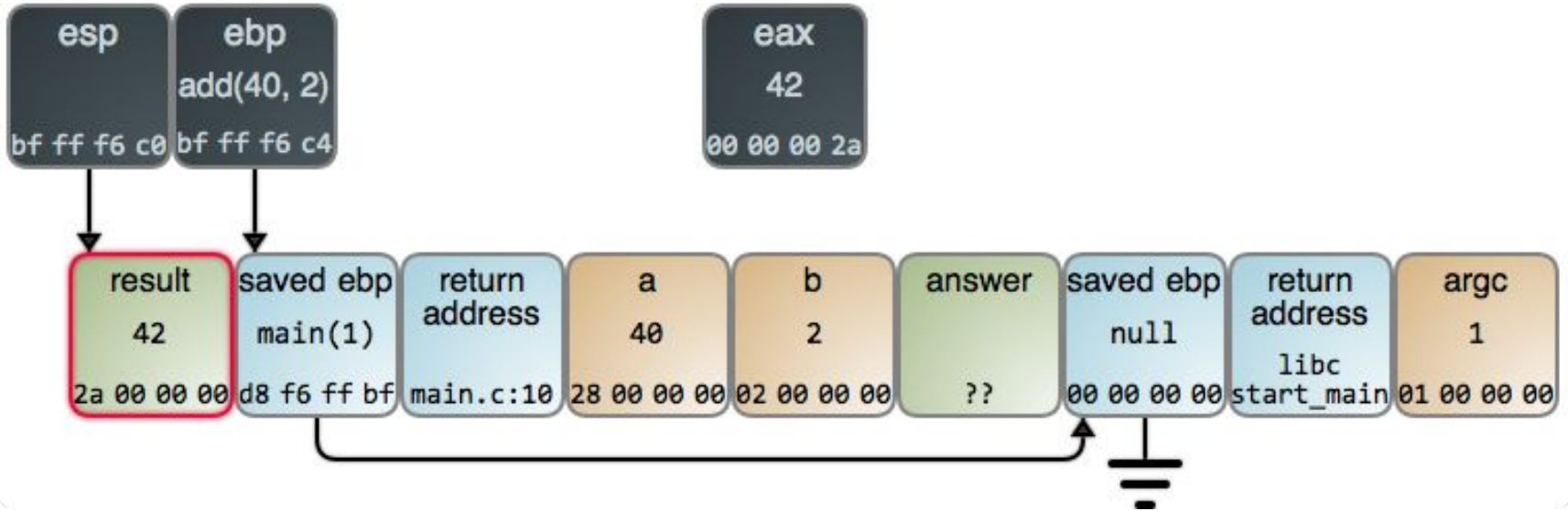
```
9.      subl $4, %esp # make room for result
```



```
      movl 12(%ebp), %eax # move b to eax
10.   movl 8(%ebp), %edx  # move a to edx
      addl %edx, %eax     # add edx into eax. total is 42.
```



```
11.    movl %eax, -4(%ebp) # copy eax to result
```



Heap (Куча)

1. В отличии от стека позволяет создавать динамические структуры большого размера
2. Управление жизнью объектов в куче “ручное”

Функции работы с памятью StdLib.h

- malloc
- free
- calloc
- realloc

malloc

1. malloc не гарантирует выделение памяти
2. не забывать выставлять указатель в NULL после освобождения
3. free(NULL) ничего не делает

malloc

```
int main() {
   int* p1 = malloc(4 * sizeof(int));
   int* p2 = malloc(sizeof(int[4]));

   if(p1) {
      for(int n=0; n<4; ++n)
         p1[n] = n*n;
      for(int n=0; n<4; ++n)
         printf("p1[%d] == %d\n", n, p1[n]);
   }

   free(p1);
   free(p2);
}
```

malloc

```
int main() {
  int i = 0;
  int* p = malloc(sizeof(int));
  int* arr = calloc(sizeof(int), 10);

  printf("Sizeof(i): %lu \t Address of i %p\n", sizeof(i), &i);
  printf("Sizeof(p): %lu \t Address of p %p\n", sizeof(p), &p);
  printf("Sizeof(*p): %lu \t Address of *p %p\n", sizeof(*p), p);

  printf("Sizeof(arr): %lu \t Address of arr %p\n", sizeof(arr), &arr);
  printf("Sizeof(*arr): %lu \t Address of *arr %p\n", sizeof(*arr), arr);

  free(p);
  free(arr);
}
```

new\delete

```
int main() {
 int* pr = new int;
 delete pr;

 int* arr = new int[10];
 delete[] arr;

 return 0;
}
```

Segmentation fault

1. Обращение к несуществующему адресу
2. Обращение к сегменту, прав для которого нет
3. Попытка поменять данные в read-only сегменту
4. Обращения по нулевому указателю
5. Обращение по указателю на удаленный указатель
6. Переполнение стека
7. Переполнение буфера

Segmentation fault

```c
#include <stdio.h>
#include <stdint.h>


int main(int argc, char* argv[]) {
    uint64_t arr[1048570]; // 8Mb
    arr[10] = 1;
    return 0;
}
```

Segmentation fault

```
int main(int argc, char* argv[]) {
    char local_str[] = "Hello world";
   //char* local_str = "Hello world";

   local_str[1] = 'E';
   printf("%s\n", local_str);

   return 0;
}
```