

Семинар #14: Сегмент памяти текст. Классные задания.

Часть 1: Сегмент памяти Текст. Указатели на функцию.

Сегмент памяти Text

- В этом сегменте хранится машинный код программы (Код на языке C, сначала, переводится в код на языке Ассемблера, а потом в машинный код. Как это происходит смотрите ниже.).
- Адрес функции - адрес первого байта инструкций в этом сегменте.

Указатели на функции

Пример работы с указателем на функцию:

```
#include <stdio.h>

void print(int a)
{
    printf("%d\n", a);
}

int main ()
{
    // Создадим указатель на функцию ( вместо названия функции - *p )
    void (*p)(int a) = print;

    // Теперь с p можно работать также как и с print
    p(123);
}
```

Подробнее в файле funcpointers/0funcpointer.c. Задачи на указатели на функцию:

- В файле funcpointers/1foreach.c лежит заготовка исходного кода. Вам нужно написать функцию `void foreach(int* array, int size, int (*f)(int))`, которая будет принимать на вход массив размера `size` и применять к каждому элементу функцию `f`.
- В файле funcpointers/2foreach_second_argument.c лежит заготовка исходного кода. Вам нужно написать функцию `void foreach(int* array, int size, int (*f)(int, int), int b)`, которая будет принимать на вход массив размера `size` и применять к каждому элементу функцию `g(x) = f(x, b)`.

Стандартная функция qsort

В библиотеке `stdlib.h` уже реализована функция `qsort`, которая сортирует произвольные элементы, используя быструю сортировку. Пример использования этой функции:

```
#include <stdio.h>
#include <stdlib.h>

int cmp(const void* a, const void* b)
{
    // В этот компаратор передаются указатели на void,
    // Поэтому их нужно привести в нужный нам тип:
    int* pa = (int*)a;
    int* pb = (int*)b;
    return (*pa - *pb);
}
```

```

}

int main()
{
    int arr[] = {163, 624, 7345, 545, 41, 78, 5, 536, 962, 1579};
    qsort(arr, 10, sizeof(int), cmp);
    // qsort( массив, количество элементов, размер каждого элемента, компаратор )
    // Функция принимает на вход указатель на функцию cmp

    print_array(10, arr);
}

```

Функция-компаратор стандартной функции `qsort` отличается от той, что была написана нами для сортировки городов и звёзд только тем, что она принимает на вход указатели типа `void*`. Это сделано для того, чтобы эта функция была более общей. С помощью неё можно отсортировать как массив чисел, так и массив указателей или массив любых структур. В функции `cmp` нужно привести указатель `void*` к указателю нужного типа.

Задача на стандартную функцию `qsort`:

- Перепишите сортировку звёзд с использованием функции `qsort`.

Часть 2: Как код превращается в последовательность байт.

a.c

```
int a = 0x1234;  
a *= 0x7755;  
a += 0x99aa88;
```

a.exe (или a.out)

```
c7 45 fc 34 12 00 00  
8b 45 fc  
69 c0 55 77 00 00  
89 45 fc  
81 45 fc 88 aa 99 00
```

a.s

```
mov  DWORD PTR [rbp-0x4],0x1234  
mov  eax,DWORD PTR [rbp-0x4]  
imul eax,eax,0x7755  
mov  DWORD PTR [rbp-0x4],eax  
add  DWORD PTR [rbp-0x4],0x99aa88
```

Из кода на C в код ассемблера:

- Код на языке C (a.c) переводится в код на языке ассемблера (a.s). Эту операцию можно сделать командой

```
gcc -S -masm=intel ./a.c
```

- Регистры процессора – это сверхбыстрая память, которая находится внутри процессора. Её размер очень мал(десятки байт), но процессор может достигаться к ней очень быстро (за 1 такт). В примере выше используются 2 регистра: `rbp` и `eax` (`eax` это часть регистра `rax`).
- Процессор может делать множество различных операций. Например, он может переместить некоторое количество байт из одного места в другое. Такие операции называются `mov`. Он может прибавить число (`add`) или умножить на целое (`imull`) и многое другое. `DWORD PTR` просто означает, что операция будет работать с 4-мя байтами.
- В примере выше в регистре `rbp` содержится некоторый адрес. Квадратные скобочки означают разыменовывание. Поэтому строка

```
mov DWORD PTR [rbp-0x4],0x1234
```

означает, что нужно положить число 0x1234 в 4 байта по адресу `rbp-0x4`

- `mov eax,DWORD PTR [rbp-0x4]`
означает, что нужно переместить 4 байта, которые хранятся по адресу `rbp-0x4` в регистр `eax`.
- `imull eax,eax,0x7755`
означает, что нужно умножить содержимое `eax` на 0x7755 и сохранить результат в `eax`.
- `mov DWORD PTR [rbp-0x4],eax`
означает, что нужно переместить содержимое `eax` в память по адресу `rbp-0x4`.
- `add DWORD PTR [rbp-0x4],0x99aa88`
означает, что нужно добавить к числу по адресу `rbp-0x4` число 0x99aa88.
- В отличие от кода на языке C, код на языке ассемблера различается на разных процессорах. Код с вычислительной системы одной архитектуры скорее всего не будет работать на другой.

Из кода ассемблера в бинарный код (.exe):

- Код на языке ассемблера (**a.s**) переводится в исполняемый файл. Эту операцию можно сделать командой `gcc a.s`
- Каждая операция кодируется некоторым числом, называемым кодом операции (**opcode**).
- Код операции **mov** на процессорах архитектуры **x86-64** может равняться **c7** или **8b** или **89** или некоторым другим значениям(в зависимости от того куда и откуда мы копируем).
- Например в строке:

`c7 45 fc 34 12 00 00`

- **c7** означает, что это операция **mov** (присвоить число переменной в памяти)
- **45** кодирует регистр **rbp**
- **fc** кодирует смещение **-0x4**
- **34 12 00 00** – это 4-х байтовое число **0x1234** (порядок байт – Little Endian)

- `8b 45 fc`
 - **8b** означает, что это операция **mov** (записать число, хранящееся в памяти, в **eax**)
 - **45** кодирует регистр **rbp**
 - **fc** кодирует смещение **-0x4**
- Все коды можно посмотреть тут ref.x86asm.net/coder64.html
- Получается, что в результате компиляции программы код превращается в последовательность байт (инструкций процессора). Эта последовательность байт и хранится в сегменте Текст.
- А указатель на функцию является просто номером первого байта, с которого начинается функция в этом сегменте.
- Менять сегмент Текст во время выполнения программы в большинстве современных операционных систем нельзя.