Лекция 5
Работа с памятью

Буфер строки

- Буфер – область памяти, отведенная для хранения строки
- Буфер имеет ограниченный размер, но размер может изменяться
- При обработке строки “на чтение” достаточно только указателя на строку
- При формировании строки в памяти важен и адрес буфера, и размер буфера

Переполнение буфера

- Если не контролируется размер данных, записываемых в буфер, возможно **переполнение буфера**
- Может иметь катастрофические последствия для безопасности системы (arbitrary code execution)

Переполнение буфера

- Если не контролируется размер данных, записываемых в буфер, возможно **переполнение буфера**
- Может иметь катастрофические последствия для безопасности системы (arbitrary code execution)
- CVE-2015-2712 (Firefox)
- CVE-2010-1117 (IE)
- CVE-2016-5157 (Chrome)

Good vs evil

- “Плохие” функции: записывают строку, но не принимают параметр размера буфера: gets, scanf(“%s”, …), strcpy, sprintf
  - gets, scanf – запрещены; strcpy, sprintf – крайне осторожно
- “Хорошие” функции: записывают строку и принимают размер буфера строки: fgets, snprintf, scanf(“%100s”, ...)

Выравнивание

- Выравнивание — гарантирует размещение переменной (простого или сложного типа) так, чтобы адрес размещения был кратен размеру выравнивания
- Дополнение — добавление в структуру скрытых полей так, чтобы поля структуры были правильно выровнены

Невыровненные данные

- Недопустимы на некоторых платформах (попытка обращения вызовет Bus Error)
- На других платформах (x86) обращение к невыровненным данным требует два цикла обращения к памяти вместо одного
- Работа с невыровненными данными **не атомарна**
- **UNDEFINED BEHAVIOR!**

Правильное выравнивание

- Тип char не требует выравнивания
- Short — выравнивание по двум байтам
- Int, long (x86), long long (x86), double (x86) — выравнивание по 4 байтам
- Long (x64), long long (x64), double (x64) — выравнивание по 8 байтам
- Выравнивание по границе 16 байтов — для стека в Linux x86
- Выравнивание по 64 байтам – для cache line
- Выравнивание по границе 4096 — размер страницы (mmap)

Базовые типы и их свойства

type	X86 Linux		X64 Linux	
	size	alignment	size	alignment
`char`	1	1	1	1
`short`	2	2	2	2
`int`	4	4	4	4
`long`	4	4	8	8
`long long`	8	4	8	8
`void *`	4	4	8	8
`float`	4	4	4	4
`double`	8	4	8	8
`long double`	12	4	16	16

Пример:

```
struct s {
    char f1;
    long long f2;
    char f3;
};
```

- X86: sizeof(s) == 16
- X64: sizeof(s) == 24

```
struct s {
    long long f2;
    char f1;
    char f3;
};
```

- X86: sizeof(s) == 12
- X64: sizeof(s) == 16

Пример для x64

```
struct s {
    char f1;      // смещение от начала - 0
    // + 7 байт на выравнивание (alignment)
    long long f2; // смещение от начала - 8
    char f3;      // смещение от начала – 9
    // + 7 байт на дополнение (padding)
};
```

- Максимальное требуемое выравнивание – 8 (для поля f2), поэтому:
  - struct s требует выравнивания 8
  - sizeof(struct s) должен быть кратен 8
- Смещение первого поля всегда равно 0
- Смещение каждого поля должно быть выровнено соответственно (быть кратным выраваниванию) типу этого поля

Динамическая память

- Область динамической памяти заданного размера нужно выделять явно
- Получаем указатель на начало области
- В динамической памяти могут размещаться и массивы элементов, и одиночные элементы
- Динамическая память должна освобождаться явно

Динамическая память

- Выделение:
  void *malloc(size_t size);
  void *calloc(size_t nelem, size_t elsize);
- Освобождение:
  void free(void *ptr);
- Изменение размера:
  void *realloc(void *ptr, size_t newsize);

Блоки динамической памяти

- Адрес, возвращенный malloc, должен быть выровнен корректно выровнен, то есть кратен 4 для x86 и кратен 16 для x64
- malloc выделяет память блоками чуть большего размера, чтобы обеспечить выравнивание (x86: 12, 20, 28… + 4 байта на служебный указатель; x64: 24, 40, 56 + 8 байт на служебный указатель) – для glibc
- Operator new (C++) работает поверх malloc

Small string optimization

- Короткие строки храним в самой структуре

```
enum { STRING_OPT_SIZE = 8 };
struct String
{
    size_t size;
    union
    {
        char *str;
        char data[STRING_OPT_SIZE];
    };
};
```

Функция realloc

- Определена в <stdlib.h>
  void *realloc(void *ptr, size_t newsize);
- Изменяет размер ранее выделенного блока ptr, возвращает адрес нового местоположения
- ptr может остаться на своем месте, но может быть и перемещен
- Если ptr перемещен, ptr разыменовывать нельзя
- Если не хватает памяти, возвращается NULL, ptr остается сохранным
- Если ptr == NULL, работает как malloc
- Если newsize == 0, работает как free

Vector implementation

- reserved – сколько памяти выделено
- size – сколько памяти используется
- data – данные
- При полном использовании выделенной памяти она расширяется в C раз с помощью realloc: C = 2 или C = 3/2 или другое

Vector vs List

- Вектор
  - (+) расположен в памяти последовательно
  - (+) оптимальнее использует кучу
  - (?) вставка и удаление из середины за O(n)
-
- Список предпочтительнее при больших размерах одного элемента и добавлении/удалении из середины
- По умолчанию следует использовать вектор
- https://isocpp.org/blog/2014/06/stroustrup-lists