Last updated: 2020-09-15

Занятие #4.

Задача 4.1.
Давайте-ка зададимся вопросом: сколько памяти может выделить malloc(3) за один вызов?
Параметр malloc (3) является целым числом типа данных size_t, поэтому логически максимальное число, которое мы можем передать в качестве параметра в malloc(3), является максимальным значением, которое size_t может принимать на платформе (`sizeof(size_t)`). Фактически, в 64-битном Linux size_t будет 8 байтов, что, конечно же, в битах равно 8 * 8 = 64. Следовательно, максимальный объем памяти, который может быть выделен за один вызов malloc(3), равен 2^64. Сколько это? Опробуйте его фактически на x86_64 и x86. (Совет: для ответа на вопрос касательно x86 не надо искать 32-битный Линукс. Подумайте!)
В 64-битной ОС malloc(3) может выделить максимум 16 эксабайт за один вызов. Теоретически, но не обольщайтесь очевидностью этого утверждения. Все на самом деле не так. Теоретический ответ на этот вопрос - 8 эксабайт (8 ЭБ). Почему?

Задача 4.2.
Что произойдет, если передать malloc(3) отрицательный аргумент? Тип данных параметра для malloc(3), size_t, представляет собой целое число без знака - оно не может быть отрицательным. Но программисты несовершенны, и ошибки Integer OverFlow действительно существуют...
Напишите test-case, который пытается вычислить количество выделяемых байтов, например: num = xa * xb. Что если num объявлено как целочисленная переменная со знаком, а xa и xb достаточно велики, чтобы результат операции умножения вызвал переполнение? Результат num станет отрицательным. Разумеется, malloc (3) не должен работать. Но если переменная num объявлена как size_t, отрицательное количество превратится в некоторое положительное. Запустите вашу программу на x86_64 и x86.

Задача 4.3.
Что произойдет, если использовать malloc(0)? Напишите test-case, в котором malloc (3) вернет
NULL, или указатель, отличный от NULL, который можно передать в free. Откомпилируйте и запустите через ltrace. Объсните поведение программы.

Задача 4.4.
Распространенное заблуждение относительно free(3) иногда приводит к его неправильному использованию. Взгляните на этот фрагмент псевдокода:
```
void *ptr = NULL;
```

```
[...]
while(<some-condition-is-true>) {
    if (!ptr)
        ptr = malloc(n);
    [... <use 'ptr' here> ...]
    free(ptr);
}
```

Вполне реоятно, что эта программа выйдет из строя в цикле (в коде <use 'ptr' here>) за несколько итераций. Почему? Напишите test-case, в котором продемонстрируйте ответ на вопрос. Покажите правильный вариант кода.
Еще раз о необходимости быть осторожными при написании кода. Ничего не предполагайте. Всякое предположение о поведении той или иной функции - это богатый источник ошибок.

Задача 4.5.

Пример фрагмента кода, использующий realloc:

```
void *ptr, *newptr;
ptr = calloc(100, sizeof(char)); // error checking code not shown here. Add
it.
newptr = realloc(ptr, 150);
if (!newptr) {
    fprintf(stderr, "realloc failed!");
    free(ptr);
    exit(EXIT_FAILURE);
}
/* ... do your stuff ... */
free(newptr);
```

Указатель, возвращаемый функцией realloc, является указателем на фрагмент памяти с измененным размером. Это может быть тот же адрес, что и исходный ptr. Фактически, теперь вы должны полностью игнорировать исходный указатель ptr и рассматривать возвращенный повторно указатель newptr как тот, с которым нужно работать.
В случае сбоя возвращается значение NULL, а исходный фрагмент памяти остается нетронутым. Напишите test-case, в котором проверьте это утверждение (создайте ситуацию, в которой будет возвращено NULL).
Содержимое только что измененного фрагмента памяти: был, например, размер выделенной памяти 100, а его увеличено до 150. При таком изменении содержимое памяти до 100 байт не изменится, а как насчет остатка (50 байт)?

Задача 4.6.

Пример фрагмента кода, использующий realloc:

```
void *ptr, *newptr;
ptr = calloc(100, sizeof(char)); // error checking code not shown here. Add
it.
newptr = realloc(NULL, 150);
```

Указатель, переданный в realloc, равен NULL. Что будет?

Другой фрагмент:

```
. . .
newptr = realloc(ptr, 0);
```

Параметр размера, переданный в realloc, равен 0. Что будет?

Задача 4.7.

Вы выделяете память для массива с помощью calloc(3); позже вы хотите изменить его размер, скажем, намного больше. Это можно сделать это с помощью realloc (3); например:

```
struct sbar *ptr, *newptr;
ptr = calloc(1000, sizeof(struct sbar)); // array of 1000 struct
sbar's
[...]
// now we want 500 more!
newptr = realloc(ptr, 500*sizeof(struct sbar));
```

Есть более простой способ - использовать `reallocarray(3)`. Его сигнатура:

```
void *reallocarray(void *ptr, size_t nmemb, size_t size);
```

Измените этот пример. Используйте `reallocarray(3)`. Сравните выводы `ltrace`. Какие отличия?