НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА ПРОГРАМУВАННЯ»

«Безпечне програмування»

*Звiт з лабораторної роботи №1*

*Тема: «Робота з пам'ятю»*

Виконав:

ст. гр. 16Б

Башкатов В.Є.

Харків – 2019

***Мета:*** Дослідити особливості функцій malloc/calloc/free/new/delete на якнайменьш 2х ОС для якнайменьш двох компіляторах.

***Завдання***

Використовуючи середовище віртуалізації VirtualBox (VMWare, docker або іншу), продемонструвати особливості наступніх функцій на якнайменьш 2х ОС для якнайменьш двох компіляторах (напр. gcc4, gcc5+, Borland C, MSVC ...):

- malloc/calloc/free/new/delete .

Показати, що

1) не всі функції алокації пам'ятю очищують дані для користування

2) функції звільнення пам'яті не завжди знищують дані після звільнення, що може привести до можливості отримання інформації про значення знищеного об'єкту

**Текст програми**

#include <iostream.h>

#include <stdio.h>

#include <windows.h>

#include <stdlib.h>

using namespace std;

void main()

{

int \*checkMalloc = (int\*)malloc(sizeof(int));

cout << "Value of func MallocPreInit = " << \*checkMalloc << endl;

\*checkMalloc = 10;

cout << "Adress Value of func Malloc = " << checkMalloc << endl;

cout << "Value of func Malloc = " << \*checkMalloc << endl;

int \*checkPrevMalloc = checkMalloc;

free(checkMalloc);

cout << "Adress Value of PrevFuncMalloc = " << checkPrevMalloc << endl;

cout << "Value of checkPrevFuncMalloc = " << \*checkPrevMalloc << endl;

////

int \*checkNew = new int();

cout << "Value of checkFuncNewPreInit = " << \*checkNew << endl;

\*checkNew = 10;

cout << "Adress of checkFuncNew = " << checkNew << endl;

cout << "Value of checkFuncNew = " << \*checkNew << endl;

int \*checkPrevNew = checkNew;

delete checkNew;

cout << "Adress of PrevFuncNew = " << checkPrevNew << endl;

cout << "Value of PrevFuncNew = " << \*checkPrevNew << endl;

///

int \*checkCalloc = (int\*)calloc(1,sizeof(int));

cout << "Value of checkFuncCallocPreInit = " << \*checkCalloc << endl;

\*checkCalloc = 10;

cout << "Adress of checkFuncCalloc = " << checkCalloc << endl;

cout << "Value of checkFuncCalloc = " << \*checkCalloc << endl;

int \*checkPrevCalloc = checkCalloc;

free(checkCalloc);

cout << "Adress of PrevFuncCalloc = " << checkPrevCalloc << endl;

cout << "Value of PrevFuncCalloc = " << \*checkPrevCalloc << endl;

system("pause");

}

**Результати роботи програми у компіляторі Visual Studio 17:**

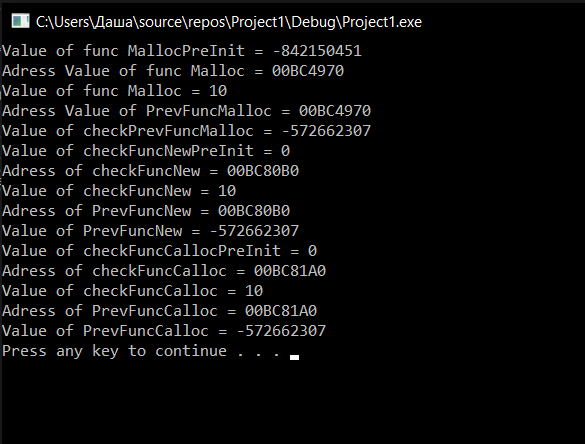


Рисунок 1 – Результати роботи програми

Як видно з рис.1, що функції алокації пам’яттю:

* + *malloc*;
  + *new*.

не очищують дані для користування на відміну від функції *calloc.*

**Результати роботи програми у компіляторі Borland C:**

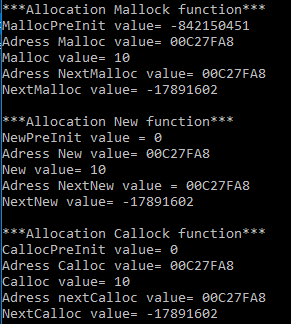


Рисунок 2 – Результати роботи програми

Як видно з рис.2, що функції алокації пам’яттю:

* + *malloc*;
  + *new*.

не очищують дані для користування на відміну від функції *calloc.*

Та функції звільнення пам’яті :

* + *free*;
  + *delete*.

Не знищують дані після звільнення, що може призвести до можливості отримання інформації про значення знищеного об'єкту.

**Результати роботи програми у компіляторі gcc++:**

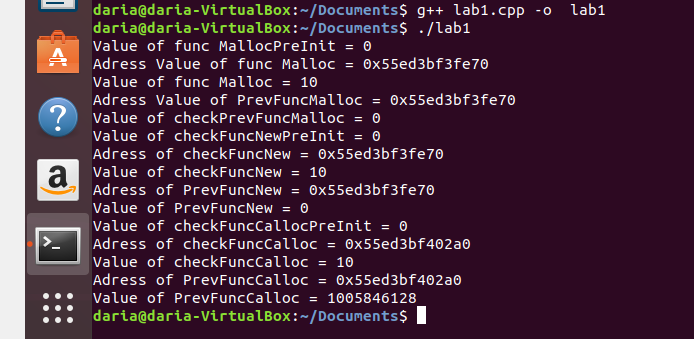


Рисунок 3 – Результати роботи програми

**Результати роботи програми у компіляторі gcc++-4.8:**



Рисунок 4– Результати роботи програми

Як видно з рис.3 та рис.4 , що після виділення пам’яті calloc та звільнення пам'яті free ми отримуємо сміття.

**Висновки:**

В результаті виконання лабораторної роботи було з’ясовано, що у різних компіляторах функції алокації та звільнення пам’яті працюють різноманітно, як приклад можна навести, що у компіляторі Visual Studio, функції звільнення пам’яті очищують дані після в звільнення, а у компіляторі Borland C - ні, це може призвести до можливості отримання інформації про значення знищеного об'єкту. Та другий приклад, що функції алокації пам’яті не завжди очищують дані після виділення пам’яті, так наприклад, у компіляторі Visual Studio та Borland C, функції malloc та new це роблять.