Creating a benchmark using C memory allocation functions

Lopatenco Andrei Kulev Mihail
Technical University of Moldova
icmcs@mail.utm.md, azzymd@gmail.com, mkmk@mail.ru

***Abstract* — Worldwide, C is the first or second most popular language in terms of number of developer positions or publicly available code.**

This thesis presents the ways, possibilities and types of memory management in C language.

Memory management is one of the most fundamental areas of computer programming. In many scripting languages, you don't have to worry about how memory is managed, but that doesn't make memory management any less important. Knowing the abilities and limitations of your memory manager is critical for effective programming. In most systems languages like C and C++, you have to do memory management.

***Index Terms* — allocators, c-style memory allocators, memory allocation, memory leak.**

I. INTRODUCTION

Today, everyone, who uses C/C++/C# languages or others, as his programming language, uses memory allocation. To understand how memory gets allocated within your program, you first need to understand how memory gets allocated to your program from the operating system. Each process on your computer thinks that it has access to all of your physical memory. Obviously, since you are running multiple programs at the same time, each process can't own all of the memory. What happens is that your processes are using virtual memory. If you've done much C programming, you have probably used malloc() and free() quite a bit. However, you may not have taken the time to think about how they might be implemented in your operating system.

II. MEMORY MANAGEMENT

In computer science, dynamic memory allocation is the allocation of memory storage for use in a computer program during the runtime of that program. It can be seen also as a way of distributing ownership of limited memory resources among many pieces of data and code.

Dynamically allocated memory exists until it is released either explicitly by the programmer, exiting a block, or by the garbage collector. This is in contrast to static memory allocation, which has a fixed duration. It is said that an object so allocated has a dynamic lifetime.

Usually, memory is allocated from a large pool of unused memory area called the heap (also called the free store). Since the precise location of the allocation is not known in advance, the memory is accessed indirectly, usually via a reference. The precise algorithm used to organize the memory area and allocate and deallocate chunks is hidden behind an abstract interface.

III. C-STYLE MEMORY ALLOCATORS

The C programming language provides two functions to fulfill our three requirements:

malloc: This allocates a given number of bytes and returns a pointer to them. If there isn't enough memory available, it returns a null pointer.

free: This takes a pointer to a segment of memory allocated by malloc, and returns it for later use by the program or the operating system (actually, some malloc implementations can only return memory back to the program, not to the operating system).

There also exist such functions as realloc and calloc, but these will be skipped at this time.

III.I *malloc*

The malloc function is one of the functions in standard C to allocate memory. Its function prototype is

```
void *malloc(size_t size);
```

which allocates size bytes of memory. If the allocation succeeds, a pointer to the block of memory is returned, otherwise a null pointer is returned.

malloc returns a void pointer (void *), which indicates that it is a pointer to a region of unknown data type.

Memory allocated via malloc is persistent: it will continue to exist until the program terminates or the memory is explicitly deallocated by the programmer (that is, the block is said to be "freed"). This is achieved by use of the free function. Its prototype is

```
void free(void *pointer);
```

which releases the block of memory pointed to by pointer. pointer must have been previously returned by malloc, calloc, or realloc and must only be passed to free once.

III.II *calloc*

malloc returns a block of memory that is allocated for the programmer to use, but is uninitialized. The memory is usually initialized by hand if necessary—either via the memset function, or by one or more assignment statements that dereference the pointer. An alternative is to use the calloc function, which allocates memory and then initializes it. Its prototype is

```
void *calloc(size_t nelements, size_t elementSize);
```

which allocates a region of memory, initialized to 0, of size nelements × elementSize.

III.III **realloc**

It is often useful to be able to grow or shrink a block of memory. This can be done using realloc which returns a pointer to a memory region of the specified size, which contains the same data as the old region pointed to by pointer (truncated to the minimum of the old and new sizes). If realloc is unable to resize the memory region in place, it allocates new storage, copies the required data, and frees the old pointer. If this allocation fails, realloc maintains the original pointer unaltered, and returns the null pointer value. The newly allocated region of memory is uninitialized (its contents are not predictable). The function prototype is

```
void *realloc(void *pointer, size_t size);
```

realloc behaves like malloc if the first argument is NULL:

```
void *p = malloc(42);
void *p = realloc(NULL, 42); /* equivalent */
```

IV. MEMORY LEAK

When a call to malloc, calloc or realloc succeeds, the return value of the call should eventually be passed to the free function. This releases the allocated memory, allowing it to be reused to satisfy other memory allocation requests. If this is not done, the allocated memory will not be released until the process exits — in other words, a memory leak will occur. Typically, memory leaks are caused by losing track of pointers, for example not using a temporary pointer for the return value of realloc, which may lead to the original pointer being overwritten with a null pointer.

V. CONCLUSION

First of all it's important to mention from the very beginning that without memory allocation functions we couldn't get well-optimized programs, as we do have now. Memory allocation function allows us to improve the speed of the program, and to store as much information in memory, as needed, therefore, to optimize the program while running, because the physical memory is not occupied with trash.

The C language brings a lot of possibilities in memory management at each place, and each time we need. Functions are easy in use, and very fast-working.

REFERENCES

[1] Bruce Eckel. Thinking in C++, 2nd Edition, Volume 1,2.Revision: 13 Last Modified: September 27, 2001.

[2] Dr. Kris Jamsa & Lars KlanderTotul despre C şi C++. Teora 2001.

[3] http://staff.um.edu.mt/csta1//courses/lectures/csa2060/c8a.html

[4] http://www.cantrip.org/wave12.html

[5] http://www.memorymanagement.org/glossary/c.html

[6] http://www.bradrodriguez.com/papers/ms/pat4th-c.html

[7] http://en.wikibooks.org/wiki/C_Programming/Memory_management

[8] http://www.mycplus.com/tutorials/cplusplus-programming-tutorials/memory-management/

[9] http://www.codeguru.com/forum/showthread.php?t=401848

[10] http://www.cs.wustl.edu/~schmidt/PDF/C++-mem-mgnt4.pdf