INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

# CS3.304.M22 Advanced Operating Systems

# Project Final Report

## Buddy Algorithm


**Team name: Fantastic Four**

**Team members:**

- Aman Khandelwal (2022201010)
- Nikhil Khemchandani (2022201042)
- Divyansh Negi (2022201014)
- Piyush Singh (2022201032)

**Instructor:** Prof Manish Shrivastava
**Mentor: Aditya Sharma**


**Github- https://github.com/wolfblunt/BuddyAlgorithm.git**

# 1.Malloc : -

The **“malloc”** or **“memory allocation”** method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. It doesn’t Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

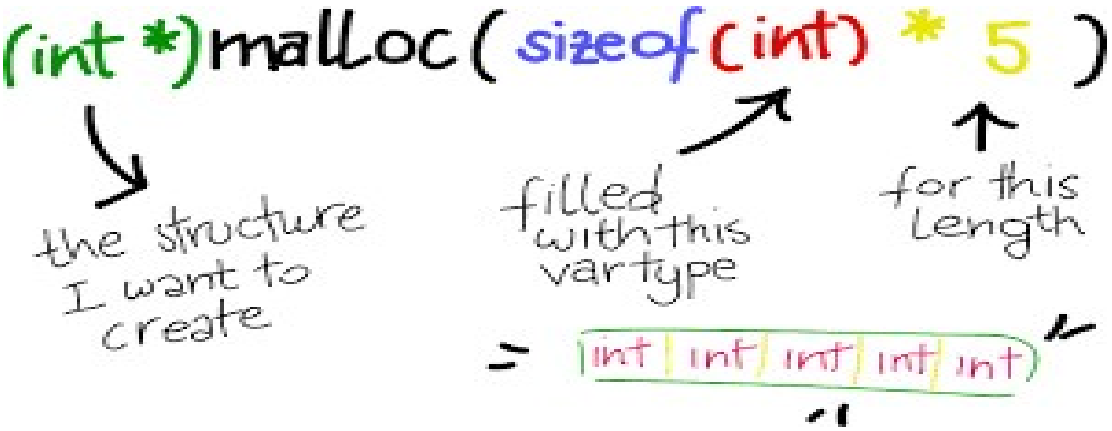
## Syntax:

ptr = (cast_type*) malloc(byte_size)

## For Example:

ptr = (int*) malloc(100 * sizeof(int));

*Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.*




If space is insufficient, allocation fails and returns a NULL pointer.

# 2.Calloc : -

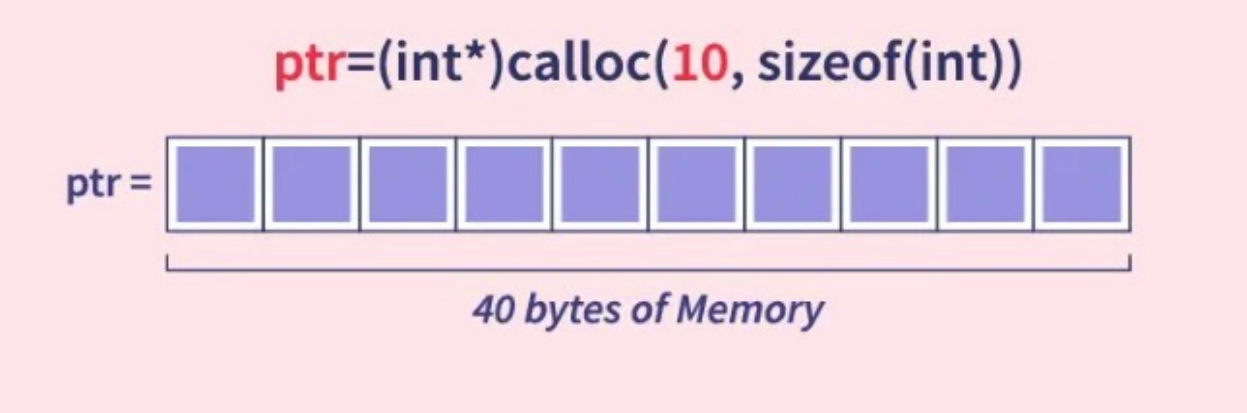
“calloc” or “contiguous allocation” method in C is used to dynamically allocate the specified number of blocks of memory of the specified type,it is very much similar to malloc() but has two different points and these are:
1.It initializes each block with a default value ‘0’.
2.It has two parameters or arguments as compare to malloc().

**Syntax:**

ptr = (cast-type*)calloc(n, element-size);
here, n is the no. of elements and element-size is the size of each element.

**For Example:**

*ptr = (float*) calloc(25, sizeof(float));*




*This statement allocates contiguous space in memory for 25 elements each with the size of the float.*

# ***Malloc V/S Calloc*** **: -**

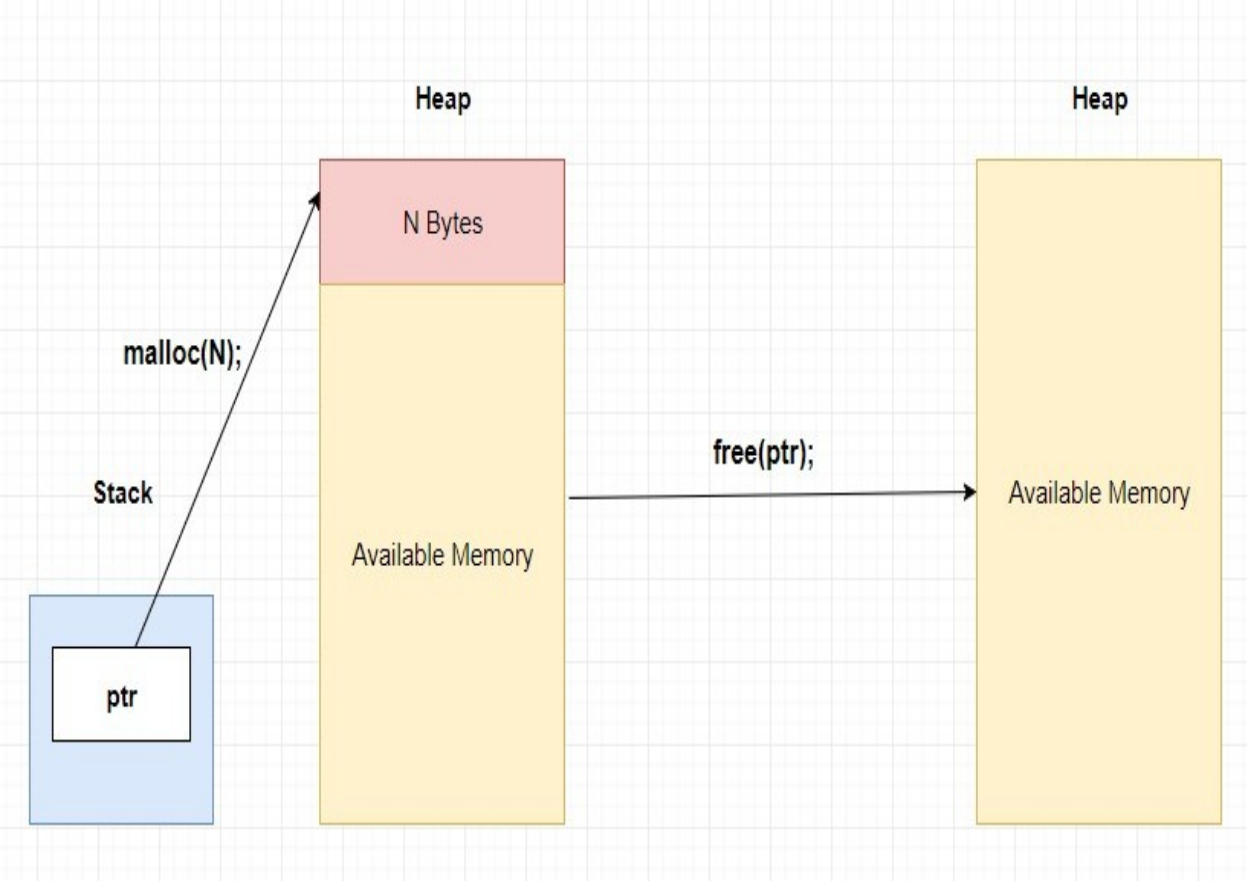
| | malloc() | calloc() |
|---|---|---|
| 1. | It is a function that creates one block of memory of a fixed size. | It is a function that assigns more than one block of memory to a single variable. |
| 2. | It only takes one argumemt | It takes two arguments. |
| 3. | It is faster than calloc. | It is slower than malloc() |
| 4. | It has high time efficiency | It has low time efficiency |
| 5. | It is used to indicate memory allocation | It is used to indicate contiguous memory allcoation |

# 3.free : -

**“free”** method in C is used to dynamically **de-allocate** the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

**Syntax:**

free(ptr);



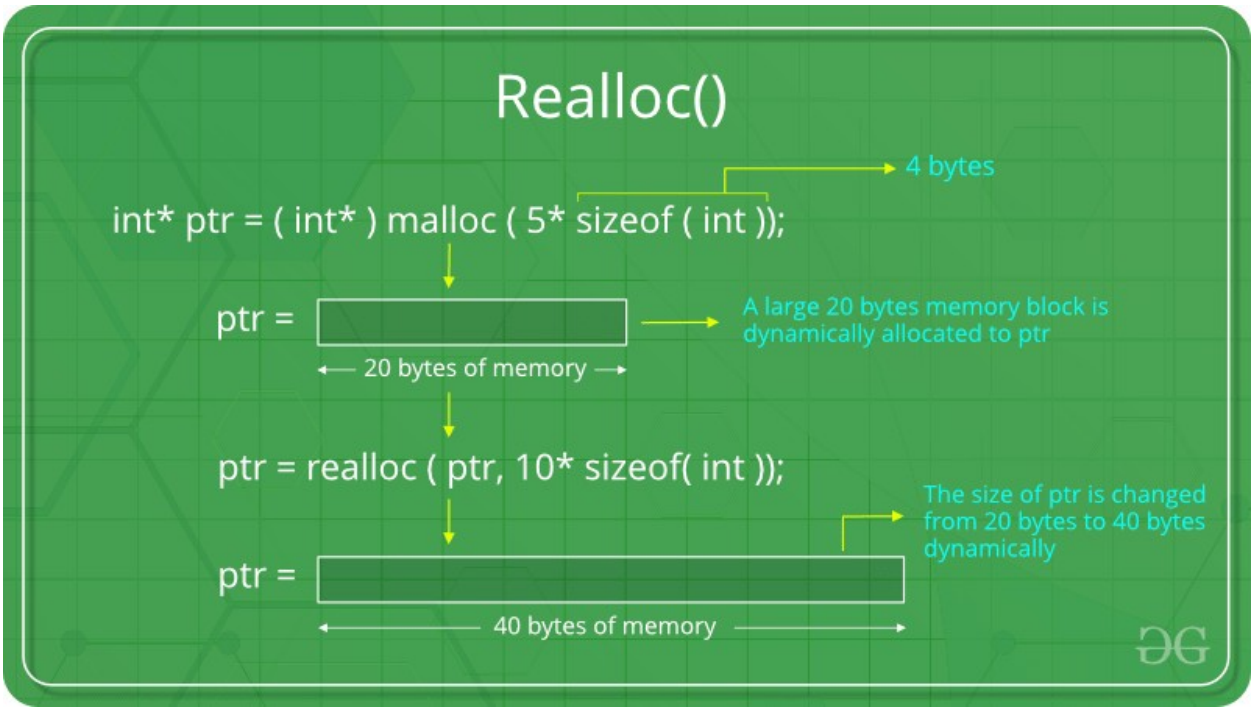
# 4.realloc : -

**“realloc”** or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

**Syntax:**

ptr = realloc(ptr, newSize);

where ptr is reallocated with new size 'newSize'.




If space is insufficient, allocation fails and returns a NULL pointer.

# 5.memalign : -

The memalign function allocates a block of size bytes whose address is a multiple of boundary. The boundary must be a power of two! The function memalign works by allocating a somewhat larger block, and then returning an address within the block that is on the specified boundary.The Address generated is a multiple of Alignment argument.

The memalign function returns a null pointer on error and sets errno to one of the following values:

ENOMEM - There was insufficient memory available to satisfy the request.

EINVAL - boundary is not a power of two.

# 6.posix_memalign : -

The *posix_memalign*() function shall allocate *size* bytes aligned on a boundary specified by *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment* shall be a power of two multiple of *sizeof*(void *).

Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned in *memptr*, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the the value returned in *memptr* is used to access an object. Upon successful completion, *posix_memalign*() shall return zero; otherwise, an error number shall be returned to indicate the error and the contents of *memptr* shall either be left unmodified or be set to a null pointer.

# 7.realloc_array : -

The function realloc is used to resize the memory block which is allocated by malloc or calloc before.

**syntax:**

void *realloc(void *pointer, size_t size)

Here,

pointer − The pointer which is pointing the previously allocated memory block by malloc or calloc.

size − The new size of memory block.

Here is an example of realloc() in C language,

Re- alloc Array , creates an array of  objects of size 'size' and the number of elements of the type object and returns void* pointing to memory.

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

# Learnings : -

- In Buddy System, the cost to allocate and free a block of memory is low compared to that of best fit or first fit algorithm
- Search for a block of right size is cheaper than best fit because we need only find the first available block on the block list for blocks of size $2^{k.}$

# Conclusion : -

This assignment helped us to understand handling memory management using the B algorithm. After completing the project, we can see the importance of how various memory management algorithms made tradeoffs in design in our case the way the algorithm handles fragmentation. Additionally, we learned how to allocate our memory allocation thread-safe in case of multiple memory allocation calls at a time by different threads of the same program.

# References : -


[1] Man page

https://man7.org/linux/man-pages/man3/malloc.3.html
https://linux.die.net/man/3/memalign
https://man7.org/linux/man-pages/man3/posix_memalign.3.html
https://man7.org/linux/man-pages/man3/free.3p.html