COMP2211: Operating Systems

COURSEWORK 1: Memory Management

Name: Ayesha Rahman

Student ID – 201522771

Username – sc21ar

Introduction – Xv6

Operating systems are software that help to share a computer among multiple programs and shares the hardware among multiple programs to run them at the same time [1]. It acts like the intermediary between the computer user and hardware [2]. This report will focus on the explanation of my code based on c programming for malloc() and free() in xv6-riscv based system and a reflection of the skills learnt. Xv6 was created for teaching purposes in MIT's Operating System Engineering course in 2006. It is a modern implementation of the Sixth Edition Unix in ANSI C for multiprocessor x86 and RISC-V systems [3 (Wikipedia)].
Malloc function allocates size requested memory and returns a pointer to it where the memory is not initialized. It returns NULL or a unique pointer value which is passed to free() when the size is 0 [4].
Malloc uses the synopsis: Void *malloc(size_t size);
Free function deallocates the memory space which has been pointed by ptr. This memory space is returned by a pervious call to malloc(), calloc() or realloc(). The ptr is NULL or no operation performed when the free(ptr) has already been called before. This is when undefined behavior occurs [4].
Free uses the synopsis: Void free(void *ptr);

Code Explanation

The malloc() function will return a pointer to the memory which is allocated suitably aligned for any bult-in type. The malloc() called will return a NULL where there is an error or when size is 0. The free() function will not return any value.
The void *malloc(int size) starts by checking if the size of the memory allocation needed is 0 or not. If it is 0 it returns a false or NULL (here returns 0). Once checked that the size is 1 or more, initializing of the first block is done using the concept of Circular Linked List. Linked lists are linear data structures where the elements are linked using pointers and not stored at a contiguous location [5]. I have used linked list as to fix the size of the array created and to insert new data every time malloc() is called. The type of linked list used in the code is Circular Linked List. In a Circular linked lists the last node of a block points to the first node of the next block [6].

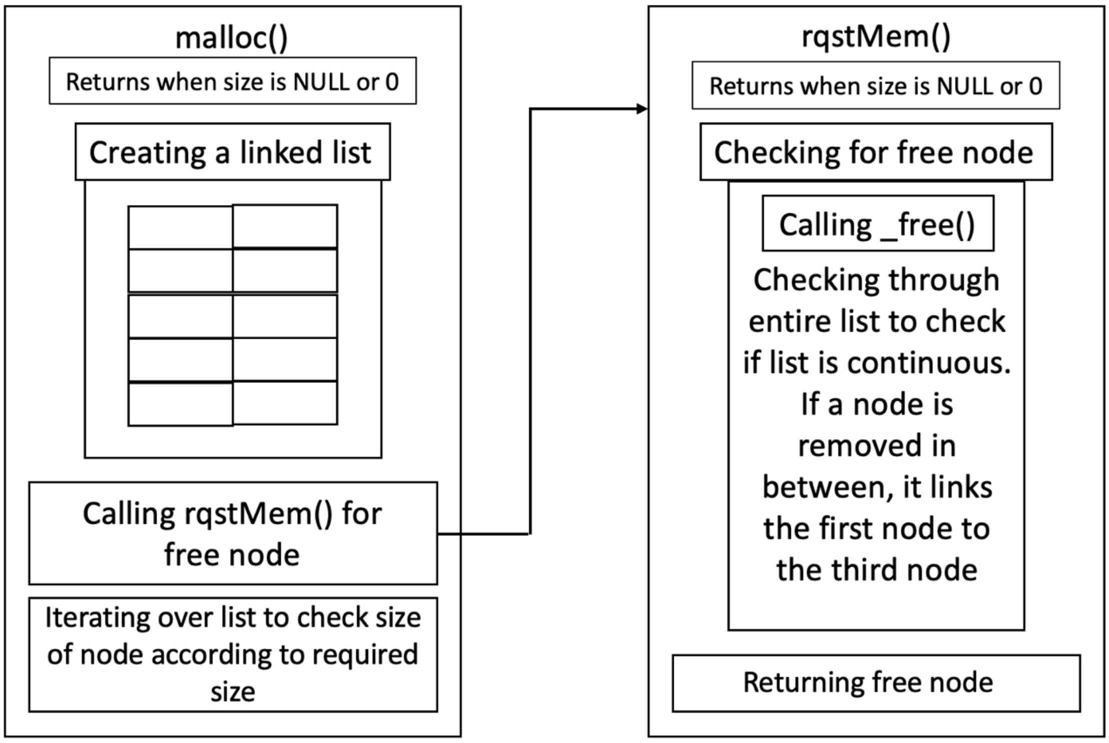
To initialize the first memory space required an if loop is created where the function checks for the existence of any previous allocated space or free node with value 0. If no such space is found the function creates a memory allocated space called block1 with size 0. Then it checks for any value allocated with block1.n->f=1. This block1 is initialised to 1. Once block1 is initialized and value is given, a new pointer is assigned. This is a free node. The pointer shifts to the next block to node1 named node2 which is initialized to 1. Now both the node1 and node2 has been initialized to 1. An if loop is run to check for existence of free node. If free node is not found then memory space is requested by the loop from the operating system by calling rqstMem() function.

In the rqstMem function an if loop is run to check whether unit is 0. If the condition is true then 0 is returned. Another if loop is run to check whether the units is less than the size of the page defined in the header file. If true, the units is made equal to the size of the page to ensure no wasting of memory address. Once the loop is terminated a pointer fmemory is created which is equal to the sbrk of units*sizeof(Node). Sbrk sets a new break value for the program by adding the value of increment to current break value [8]. After this an if loop is run to check whether the system is full or not. A new pointer freenode is defined which is equal to fmemory. The size of freenode is equal to units. Freenode is then run through the function _free() to check whether its full or not. Freenode is then taken as free and initialized. At the end of the function rqstMem, fnodes is returned as memory space is made.

When the _free() function is called in the rqstMem() function, an if loop is run to check if pointer is giving NULL for the address accessed. Then a while loop is run to iterate over nodes to find a free node. This checks a condition where the opposite of the conditions node2 is less than freednode and freednode less than the node next to node2. If loop is true, another if loop is run to check whether either node2 is less than freenode or freenode is less than node next to node2 and node2 is greater than or equal to node next to node2. If condition is true then the loop breaks as the sequence of the linked list is broken. Node2 is then taken equal to the node next to it and the while loop runs again through the entire linked list. After terminating while loop another if loop is run to check if freednode + size of freednode is equal to node next to node2. If true, size of freednode is equal to size of freednode plus size of node next to node2. The address of the new freenode size if changed to the address of the node next to the node which is next to node2. If the condition is false, the address of node next to freednode is considered equal to the address of the node next to node2. After this another if look is run where the condition is the node2 plus size of node2 is equal to freednode. If the condition is true, size of node2 is taken equal to size of node2 plus size of freednode. Then the node next to node2 is taken equal to node next to freednode. If the condition is false, the node next to node2 is taken as freednode. After the loop is done, fnode is taken equal to node2.

The code then iterates over all the free nodes using an infinity while loop as long as it is not returned. If the node is empty or 0 then the loop is not run. If the while loop is run, an if condition is run to check the size of the node2 is bigger than the size required when malloc() is run. If the size of the node2 is smaller than the required size, the if condition will be false. In this case, we are iterating over to the next node (node2 = node2->n) If this is true another if loop is run to check if the size of node2 is not equal to the required size. If the condition is true, size of node2 is subtracted by the required size which is made equal to the size of node2. As node2 is a pointer, the new node2 size is added to the address of the node2 to go to different address according to the new size. Then the new node2 size is taken as the required size (node2->size = size). After this the pointednode is taken as free and intialised as we have a free node now. The loop will then return node2 to void malloc() to run the function again.

The else loop is run after this with the condition when the size of node2 is equal to the required size. If the condition is true, the address next to the pointednode is taken as the same as the address next to node2 and the pointednode is taken as free and initialised. A similar pattern is then followed as the if loop above, the free node is initialised and node2 is returned to the void malloc() and the while loop is terminated to run the void malloc() function again.


malloc()
Returns when size is NULL or 0
Creating a linked list
Calling rqstMem() for free node
Iterating over list to check size of node according to required size
rqstMem()
Returns when size is NULL or 0
Checking for free node
Calling _free()
Checking through entire list to check if list is continuous. If a node is removed in between, it links the first node to the third node
Returning free node

Skills learnt

1. Creating malloc() and free() functions allowed me to understand better the working behind the allocation of memory spaces.
2. Debugging the code, I created and running through it multiple times showed me how to perfect my program to get the desired results.
3. Working on the code helped me practice working in c programming language and allowed me to have a higher understanding of the coding language.

References

1. Cox, R., Kaashoek, M.F. and Morris, R., 2011. Xv6, a simple Unix-like teaching operating system. *2013-09-05]. http://pdos. csail. mit. edu/6.828/2012/xv6. html*.
2. Silberschatz, A., Galvin, P.B. and Gagne, G., 2018. *Operating System Concepts, 10e Abridged Print Companion*. John Wiley & Sons.
3. XV6 (2022) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Xv6
4. Malloc(3) - linux manual page. Available at: https://man7.org/linux/man-pages/man3/malloc.3.html
5. What is linked list (2022) GeeksforGeeks. Available at: https://www.geeksforgeeks.org/what-is-linked-list/
6. Program for all operations on circular linked list in C (2022) GeeksforGeeks. Available at: https://www.geeksforgeeks.org/program-for-all-operations-on-circular-linked-list-in-c/#:~:text=In%20a%20Circular%20linked%20list,points%20to%20the%20first%20node.
7. Mit-Pdos (2019) *Xv6-riscv/umalloc.c at RISCV · mit-pdos/XV6-RISCV*, *GitHub*. Available at: https://github.com/mit-pdos/xv6-riscv/blob/riscv/user/umalloc.c **(reference for code)**
8. www.qnx.com. (n.d.). Help - Eclipse SDK. [online] Available at: http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_lib_ref%2Fs%2Fsbrk.html.