When user call *malloc()*:

* *void\* malloc(uint nbytes)* defined at **umalloc.c:64** first be called, in this function.
* Calculate the size of memory for a block include header and saved in local variable “nunits”.
* Check the free list (which is circular linked list), if list is null (freep=null) just initial it as an empty list.
* Search in the free list to find the first block that not smaller than the target size. Split the founded block to the size we want (if any needed) and return the pointer point to the top of this block (not include Header).
* If no suitable block founded, call *morecore()* to get more heap space and add the new space in free list, then search again; if system cannot get more space, return 0.

Furthermore------------------------------------------------------------------------------------

* + *static Header\** *morecore(uint nu)* defined in **umalloc.c:47**
  + Set the target size larger than 4096, then call *sbrk()* to increase the heap size of nu\*8 bytes;
    - sbrk is a syscall, defined in **usys.S:29**, then jump to kernel mode and start at **syscall.c:132**. The function *void syscall()* will handle which syscall to use by get information from current proc.
    - For sbrk, *sys\_sbrk(void)* defined at **sysproc.c:46** will be called. First use *argint()* to get parameter (same way as pa2), if the parameter’s value is smaller than 0, return -1 for error.
    - Save the current size of proc heap in variable addr, and call *growproc()* to increase the heap size.
      * *int growproc(int n)* defined in **proc.c:159**,
      * First het process state by using*myproc()*;(same as pa2).
      * Then check the size change: if change > 0, call *allocuvm()* to increase the heap size; else if change <0, use *deallocuvm()* to decrease the heap size. If success, these two functions will return new size of process heap, or return -1 for failed.
        + In *int allocuvm(pde\_t \*pgdir, uint oldsz, uint newsz)* defined in **vm.c:221**
        + Check the newsize: if it is larger, then KERNBASE. return -1 for error; if newsize < oldsize, just return old size.
        + Use the oldsize and macro PGROUNDUP to get the last page address in current process.
        + In a for loop, just try to add new pages in current heap:

In loop, first call *char\* kalloc(void)* defined at **kalloc.c:84** to get an address of 4096-byte page of physical memory. It just tries to pick up the first page in the freelist of physical memory and remove it.

If kalloc failed, print error message and return 0 for error.

Use memset to make the new page empty.

Use mapepages to create PTEs for new page

*static int mappages(pde\_t \*pgdir, void \*va, uint size, uint pa, int perm)* defined in **vm.c:61**

Use macro PGROUNDDOWN, the given virtual address to get the range of the new virtual address for new pages,

For each page call *walkpgdir()* defined in **vm.c:36** with last parameter as 1 to create PTE. In *walkpgdir()*, first get the indexes for multiple layer page tables, as the past parameter alloc is 1, the secondary table will be generate if the secondary layer table does not exist. Then, this function returns the address of PTE if success or 0 for failed.

After get PTE address, check its usability, if the pte is in use, print error message using *panic()*;

Put the physical address and mark bits into pte.

This function returns -1 for error or 0 for success.

If *mappages()* failed, use *kfree()* defined in **kalloc.c:61** to re-add the page into free list of physical memory.

If any one of kalloc of mappages return error, it needs deallocuvm to change the heap size back, and return 0 for error.

Return new size of proc heap.

* + - * + For *int deallocuvm(pde\_t \*pgdir, uint oldsz, uint newsz)* defined at **vm.c 255**, it’s just try to free PTEs between the virtual address of oldsize and newsize.
      * If the “% allocuvm” success, use switchuvm defined in **vm.c:157** to finish TSS operation and load page table into cr3 register. Then return 0.
      * If failed return -1 for error.
    - If *growproc()* success (return value > 0), return the addr; else return -1 for error.
    - Syscall finished, return to user mode.
  + Get the address from *sbrk()*, and build a header into this block,
  + Use *free()* to add this block into process free list.
  + If sbrk failed (return -1), return 0 for error; else return the freep.

Get back to malloc-----------------------------------------------------------------------------------

* If *morecore()* success, search the free list again; else return 0 for error.
* Malloc finished.

When user call *free()*:

* First get the header from the given virtual address.
* Then search the free list in a for loop to find the place for new free block that keeps the address increase.
* After getting the right position for the new free block, check the block which is after it; if the two blocks are next to each other, merge them together.
* Check the block before it: if they are next to each other, merge them.
* Set the head of the free list to where this block is.