## Name: Jack Brennan

**Task 1**

I have decided to use a 5MB memory with 4KB pages.

The Memory will be split into 16 blocks of 32 pages, 16 blocks of 16 pages, 16 blocks of 8 pages, 16 blocks of 4 pages, 16 blocks of 2 pages and 8 blocks of 1 page. Since I am using the buddy system for allocation this will save time from splitting large chunks of memory repeatedly. While also keeping the size as a power of two.

I will use a Hast Table to implement free memory management. I choose to use a hash table as its expected search is O(1) and I am using variable block sizes.

I’ll use the buddy system to implement memory allocation. Less time is spent designing the memory blocks for the specific purpose of the machine. The memory is static enough to not have a lot of external fragmentation yet dynamic enough to not have a lot of internal fragmentation.

For Memory Replacement I have decided to implement Not Recently Used with the Clock Algorithm. This is because it seems to be the most implementable within the time given, while still being a good system.

Get free block.

Allocation

Buddy System

Free Memory Manager

Hashtable ADT

If no block available and can’t split any.

If memory not full.

Check if memory full.

Replacement

Clock Not Recently Used

Request Queue

Queue ADT

If memory full.

**Task 2 Pseudocode**

*# set up a block class with attributes for (pages, size, what request its currently holding, its buddy, BitA and BitM)*

*# set up a Request class with attributes for (id and size)*

*#*

*# import HashTable*

*# import Queue*

*#*

*# class Manager:*

*# def init():*

*# requests = new Queue()*

*# free = new HashTable()*

*# # main and is just a representation of the hardware the cpu would know where the memory of its requestId is*

*# main = new HashTable()*

*# inMemory = new Hashtable()*

*# # set up memory partitions*

*# blockArray = [16,16,16,16,16,8]*

*# pageArray = [32,16,8,4,2,1]*

*# for i in range(len(blockArray)):*

*# for b in range(blockArray[i]):*

*# block = new Block(pageArray[i],pageArray[i]\*4)*

*# main.add(block,block)*

*# free.add(block.size,block)*

*#*

*# def addRequest(request):*

*# requests.add(request)*

*#*

*# def run():*

*# for request in requests:*

*# # if the request is allocation/replacement*

*# if request.type == "A":*

*# if free.size > 0:*

*# allocate(request)*

*# else:*

*# replacement(request)*

*# # if its a deallocation deallocate*

*# elif request.type == "D":*

*# deAllocate(request)*

*# # otherwise must be a write*

*# else:*

*# write(request)*

*#*

*# def deAllocate(request):*

*# # find block*

*# block = main.search(request.block)*

*# # deallocate*

*# block.requestId = None*

*# # check if pair available*

*# block.pair()*

*# free.add(block.size,block)*

*#*

*# def allocate(request):*

*# # check if already in memory*

*# if inMemory(request.id) is None:*

*# # get size*

*# nSize = request.size*

*# # select a size*

*# if request.size < 4:*

*# block = free.search(4)*

*# else:*

*# # round up to nearest power of 2 using bitwise*

*# nSize -= 1*

*# nSize |= n >> 1*

*# nSize |= n >> 2*

*# nSize |= n >> 4*

*# nSize |= n >> 8*

*# nSize |= n >> 16*

*# nSize += 1*

*# block = free.search(nSize)*

*# if block is None:*

*# block = block.split(nSize)*

*# block.requestId = request.id*

*# inMemory.add(request.id, block)*

*# else:*

*# block = inMemory.search(request.id)*

*# block.bitA = True*

*#*

*# def write(request):*

*# block = main.search(request.id)*

*# block.bitM = True*

*#*

*# def replacement(request):*

*# select a size of block in a simlar way as above in allocation*

*# # get all the blocks of a size*

*# clock = main.search(blocksize)*

*# # set up hands*

*# handA = clock.top()*

*# handM = clock.top()*

*# for i in range(clock.size//2):*

*# handM = handM.next*

*# # clock*

*# while (handA.\_item.bitA or handA.\_item.bitM) is True:*

*# if handA.bitA == 1:*

*# handA.bitA = 0*

*# if handM.bitM == 1:*

*# handM.bitM = 0*

*# handA = handA.next*

*# handM = handM.next*

*# # replace the block selected*

*# inMemory.remove(handA.request)*

*# handA.requestId = request.id*

*# inMemory.add(handA.requestId)*

*#*

*#*

*# def pair(block):*

*# if block.buddy.requestId == None and block.buddy.size == self.size:*

*# block.pages = block.pages \* 2*

*# block.size = block.size \* 2*

*# bud = block.buddy*

*# main.remove(bud)*

*# free.remove(bud)*

*# del(bud)*

*#*

*# def split(block, sizeN):*

*# block.pages = block.pages//2*

*# block.size = block.size//2*

*# bud = Block(block.pages,block.size)*

*# block.buddy = bud*

*# bud.buddy = self*

*# main.add(bud)*

*# free.add(bud)*

*# if size != sizeN:*

*# return split(block, sizeN)*

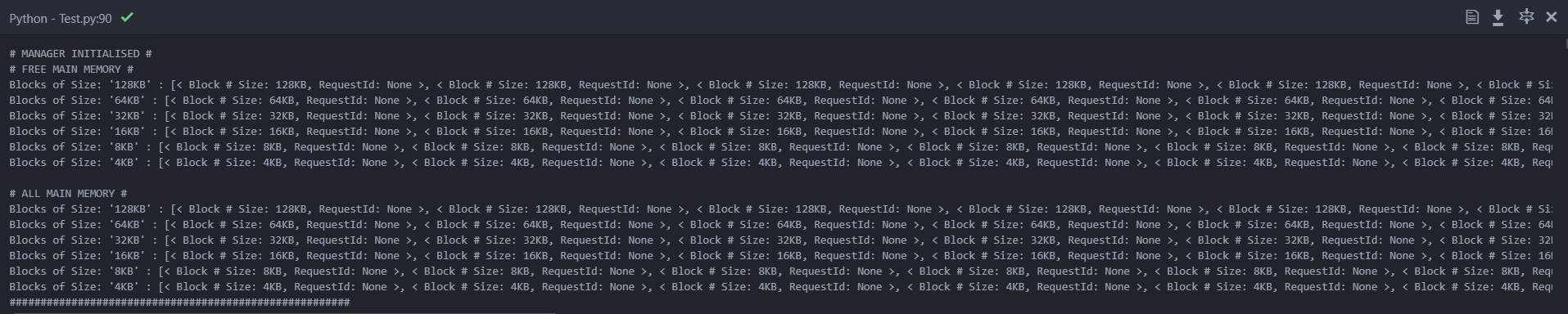
*# else:*

*# return self*

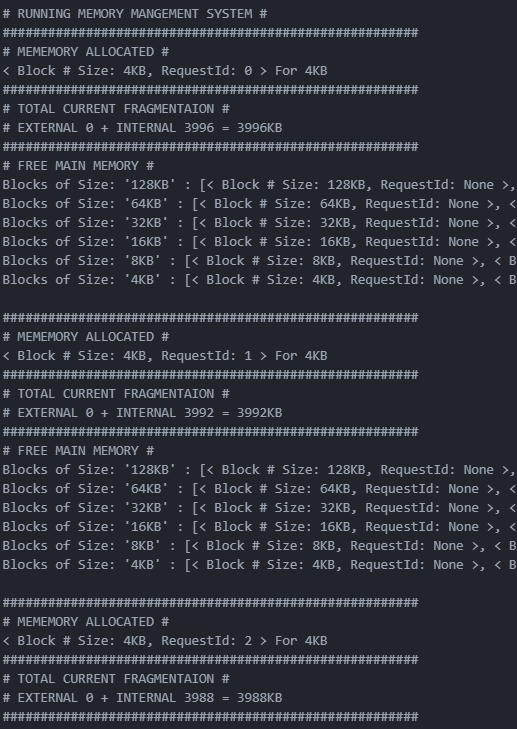
**NOTE:**

**I originally confused the label for Internal and External Fragmentation. Instead of retaking the 46 screenshots below, I just put a picture with the correct text over any of the wrong text in the screenshots. You should not notice it, but if the print for fragmentation seems unaligned this is the reason**

**Task 4 Output**



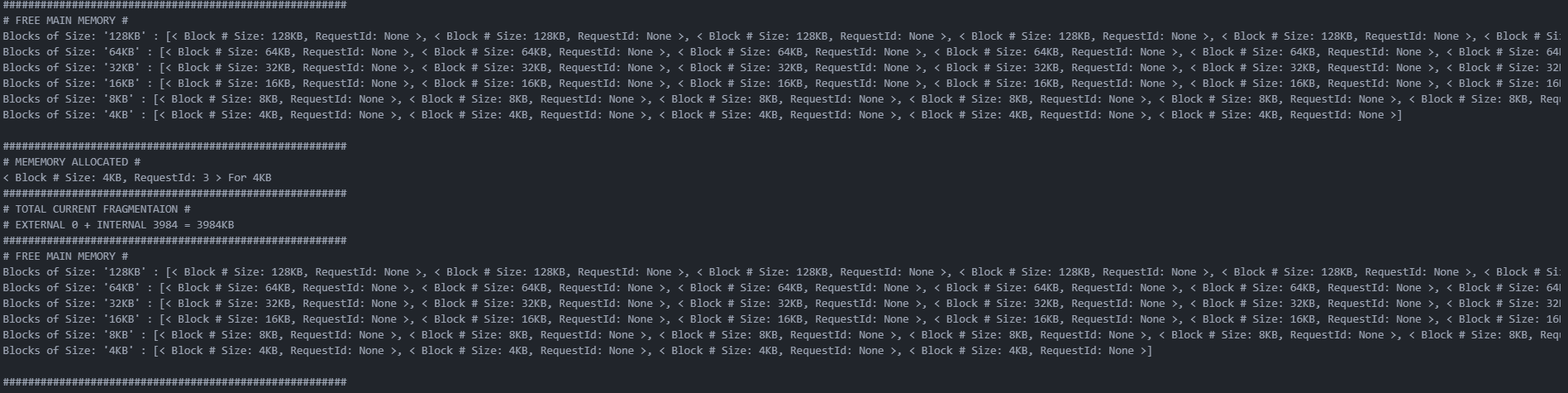
Memory initialised with partitions described above.

****

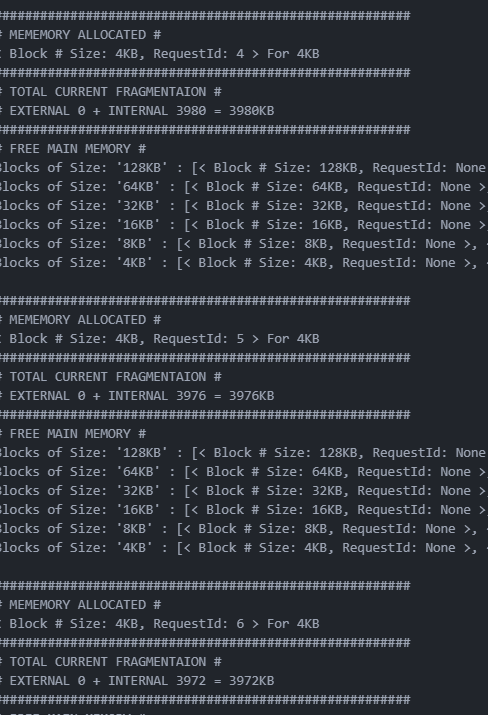
Representation of the free memory cut off as it’s very big will show example of it changing later.

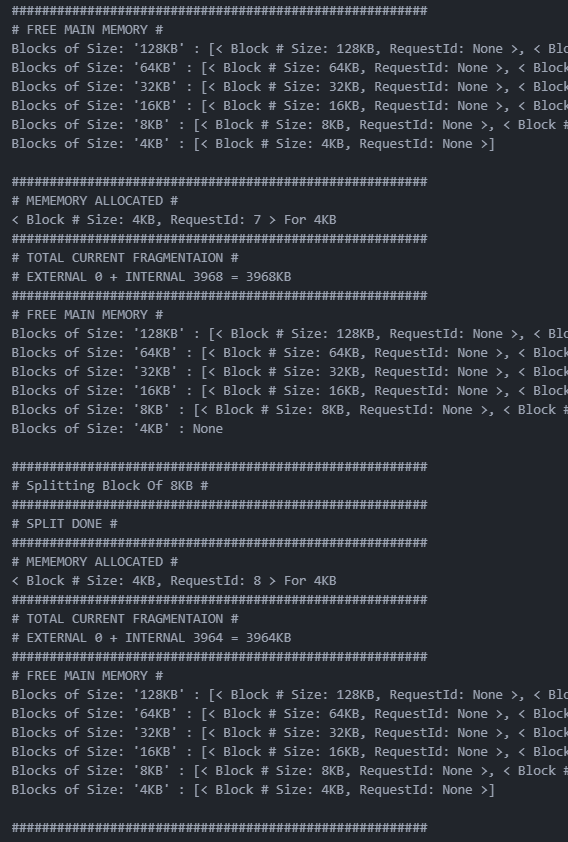
External fragmentation decreases by block amount as it’s now being used.

Allocation of a block of 1 page you can tell it’s one page as 4KB/4KB = 1

****

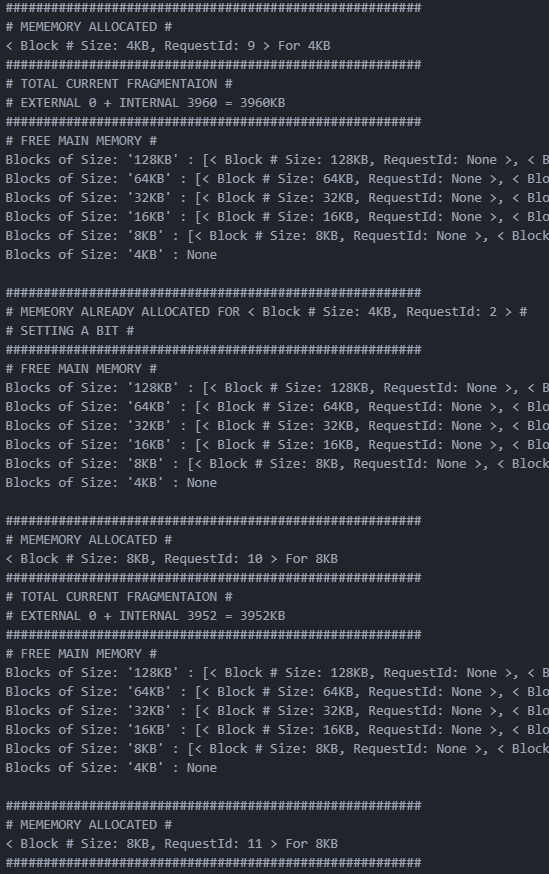
Example of allocation leaving the free memory.

****

****

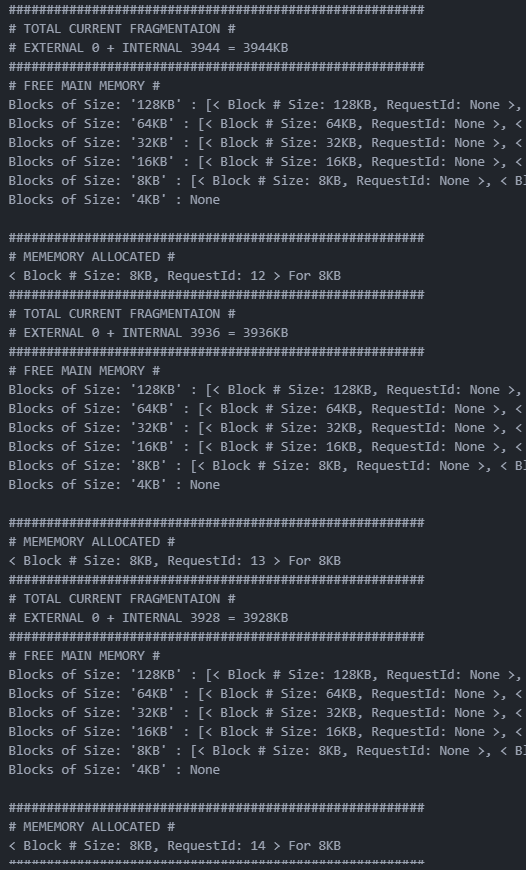
Another 4KB available because of the split

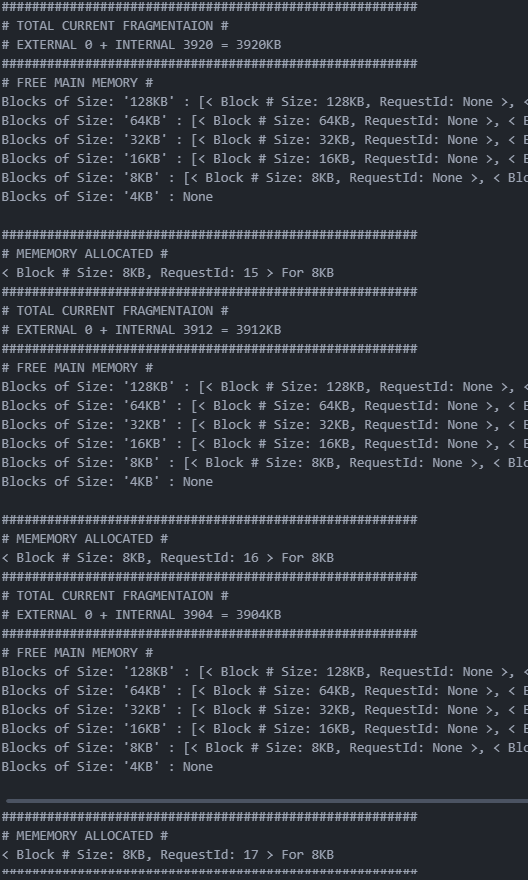
no 4KB left for a 4KB request so split an available block above it.

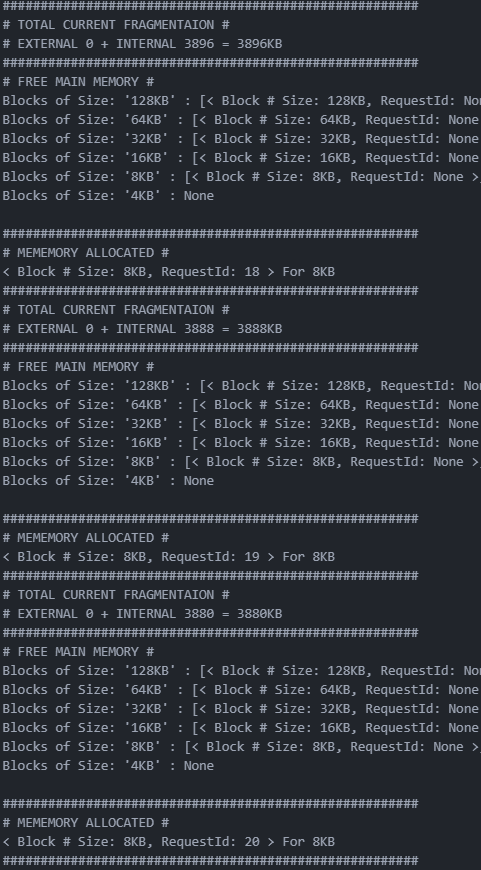
****

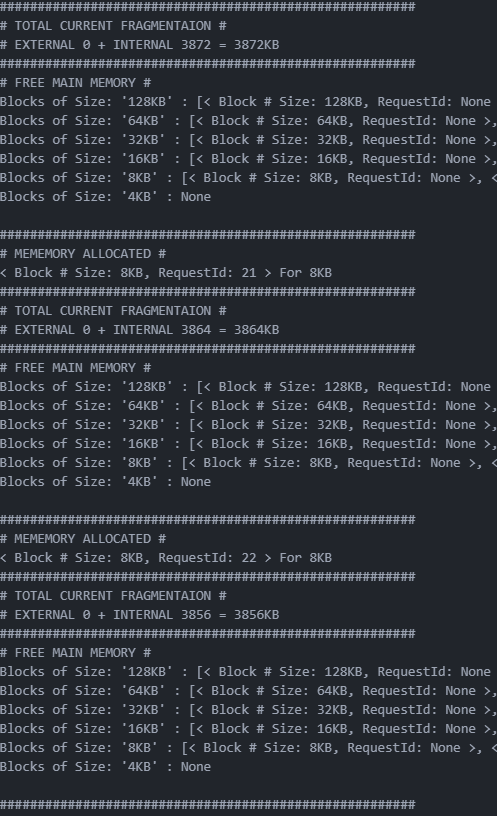
Attempted allocation of a request already in memory. Reset it’s A bit to true.

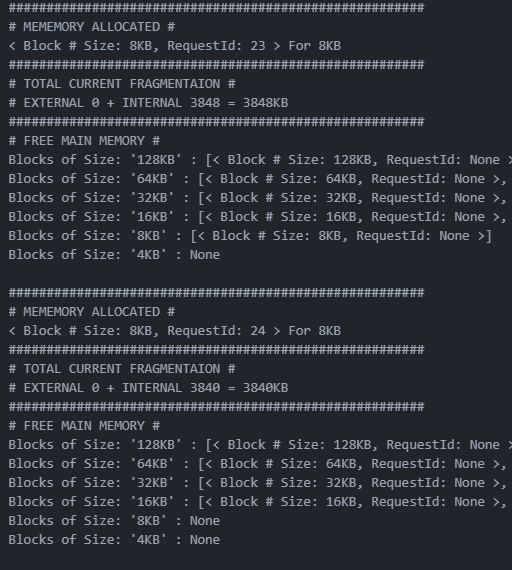
**NEXT NODE PAGE 27**

****

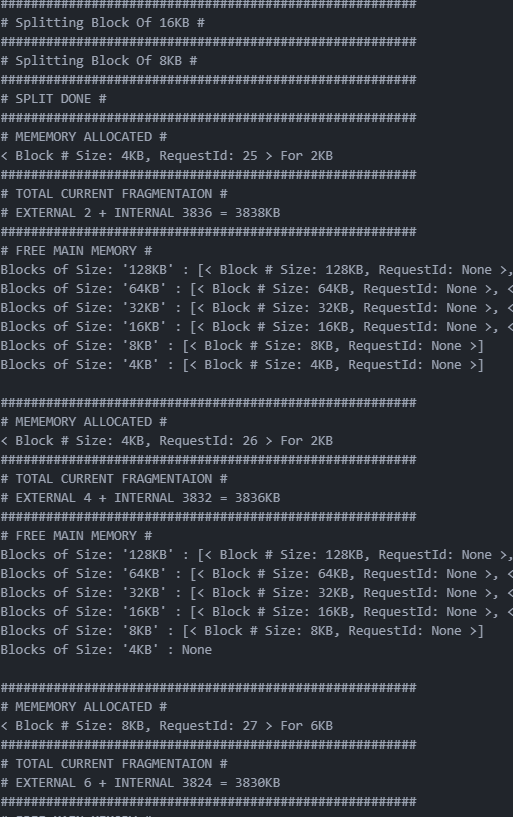
****

****

****

****

Example of an allocation

****

Request size not a power of two, so round up to nearest one and allocate that block.

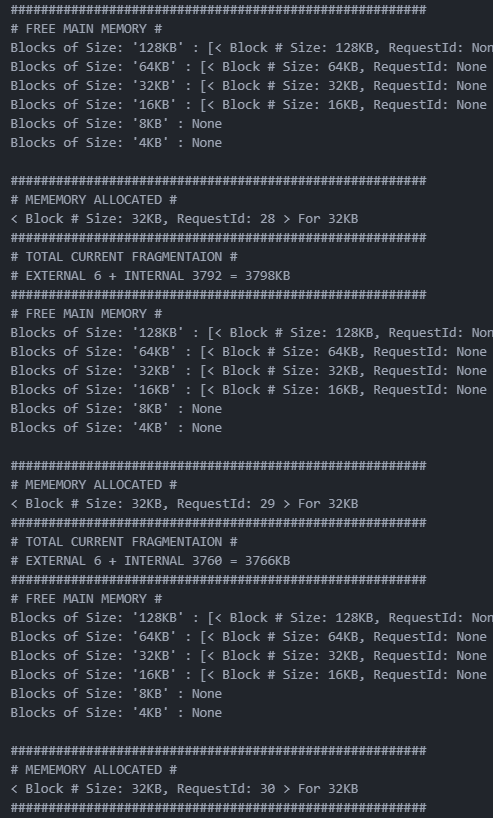
Blocks from the split. That haven’t been allocated

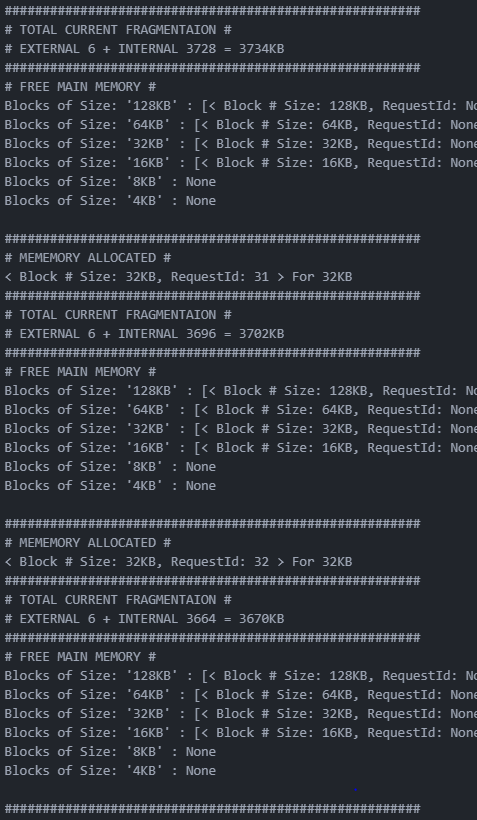
Internal fragmentation increased by 2 as 2KB not being used in the above allocation.

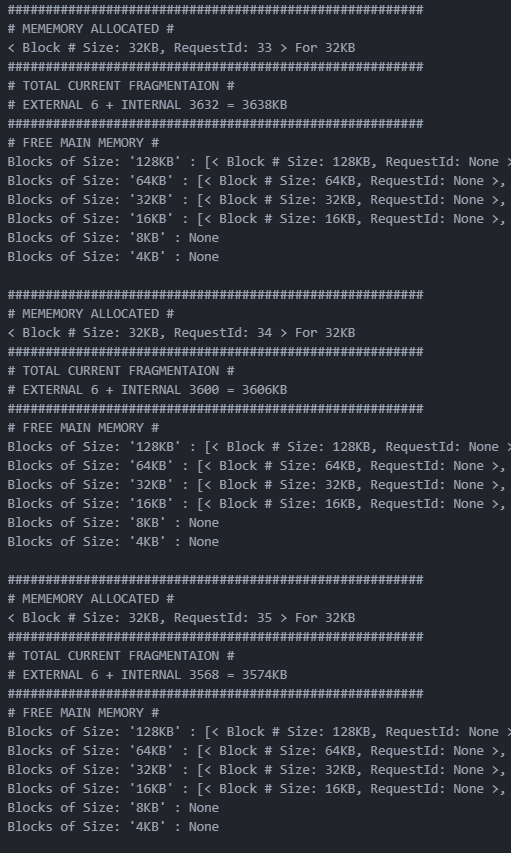
Request size below min allocation so allocated to a block of the min allocation size.

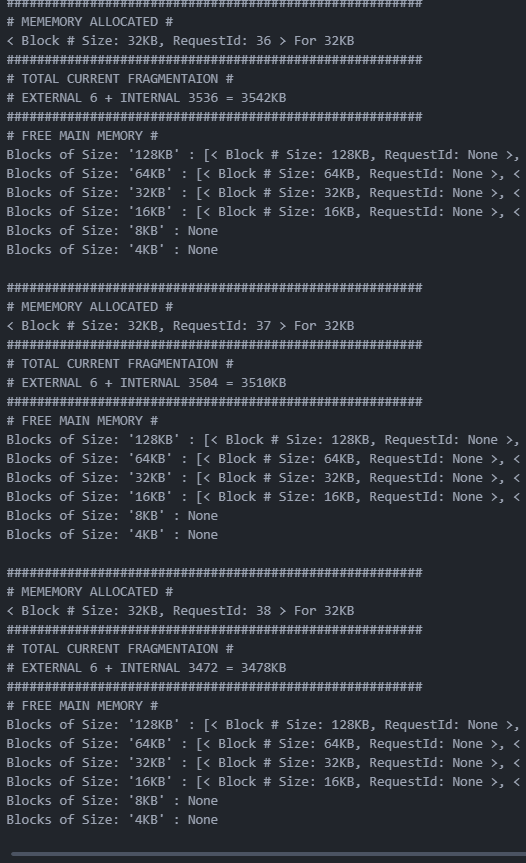
Example of a double split as there was no available block in the size directly above.

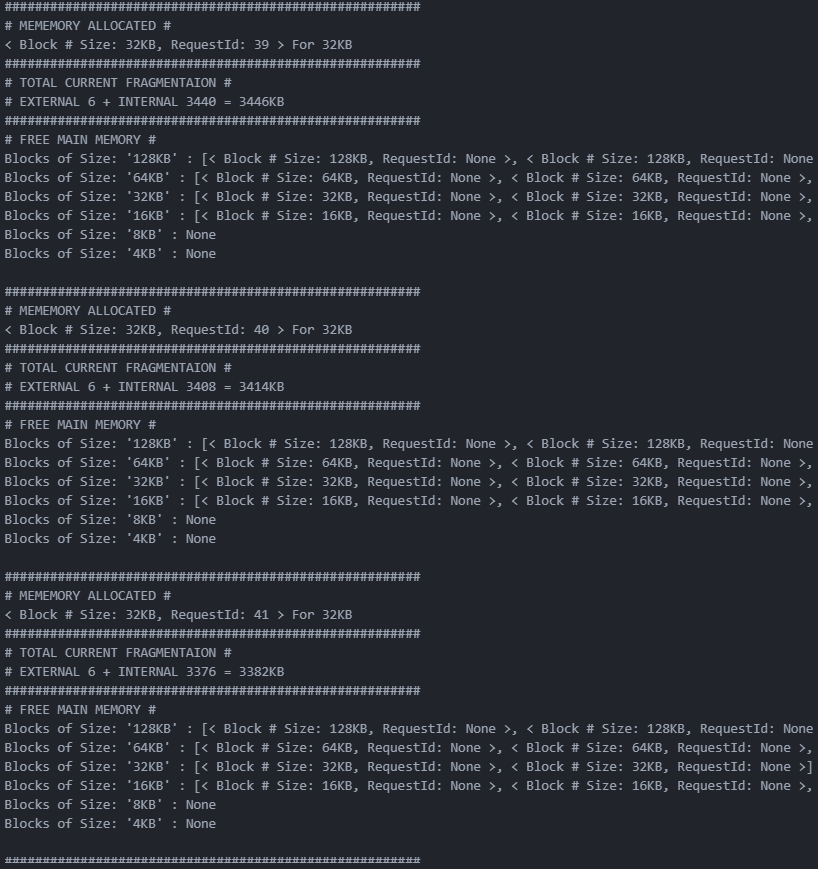
**NEXT NOTE PAGE 52**

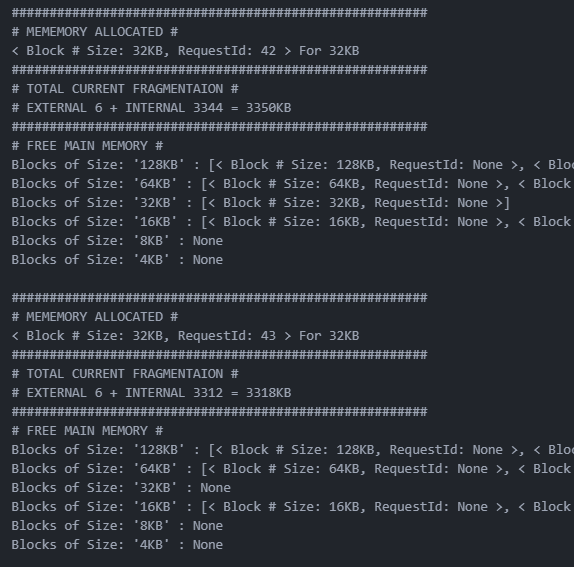
****

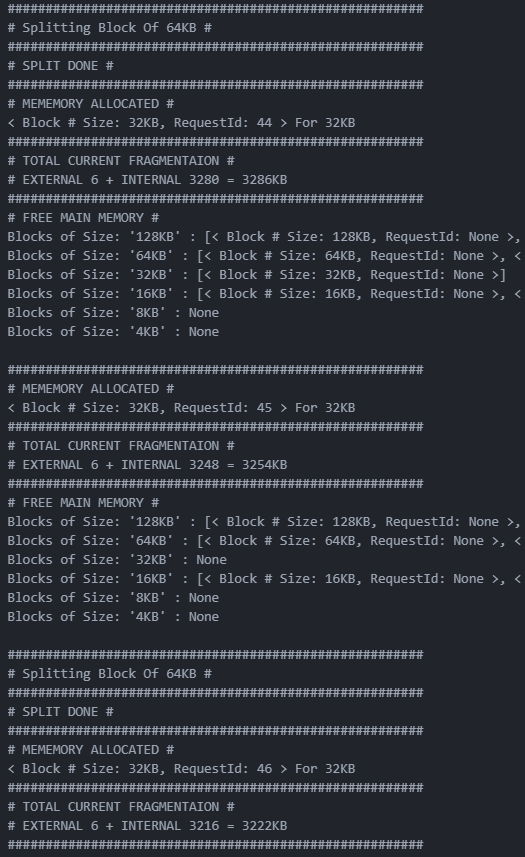
****

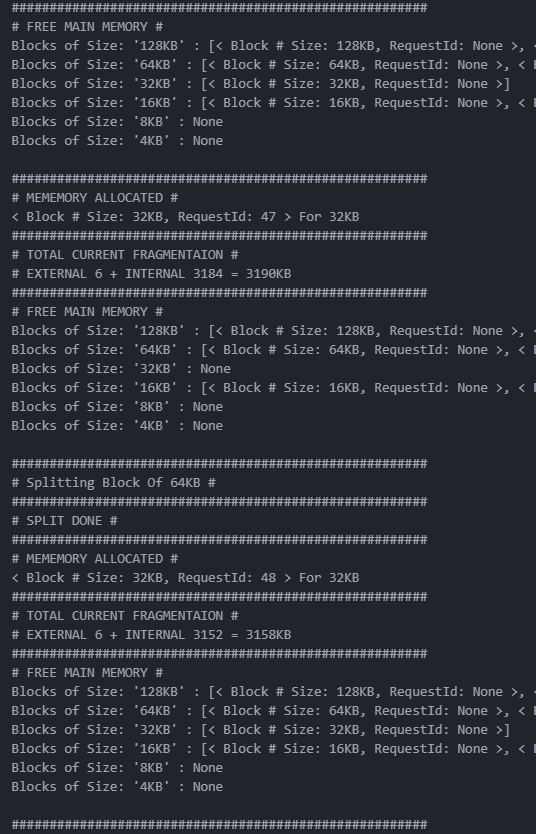
****

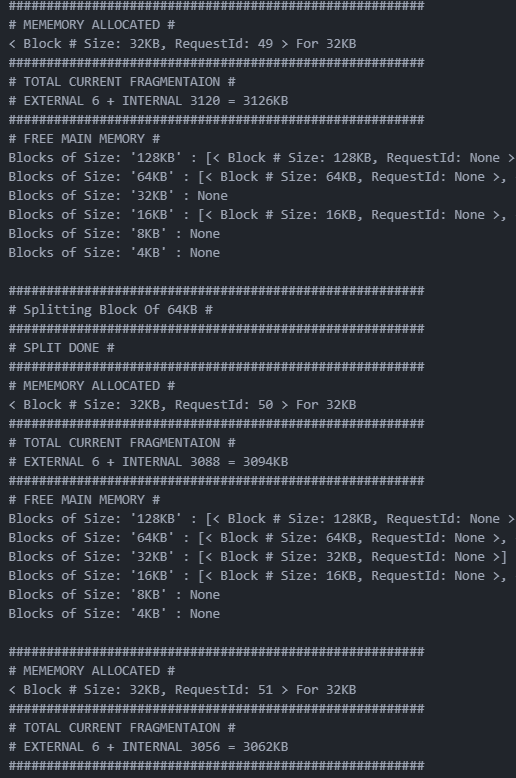
****

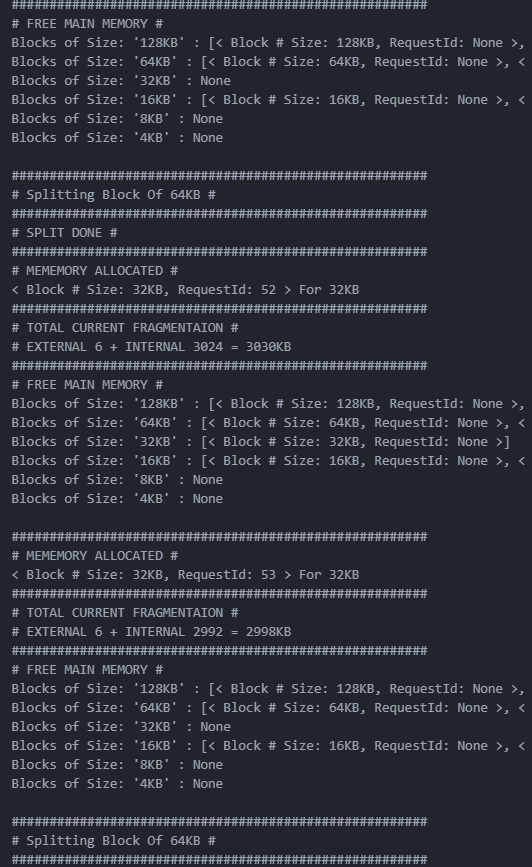
****

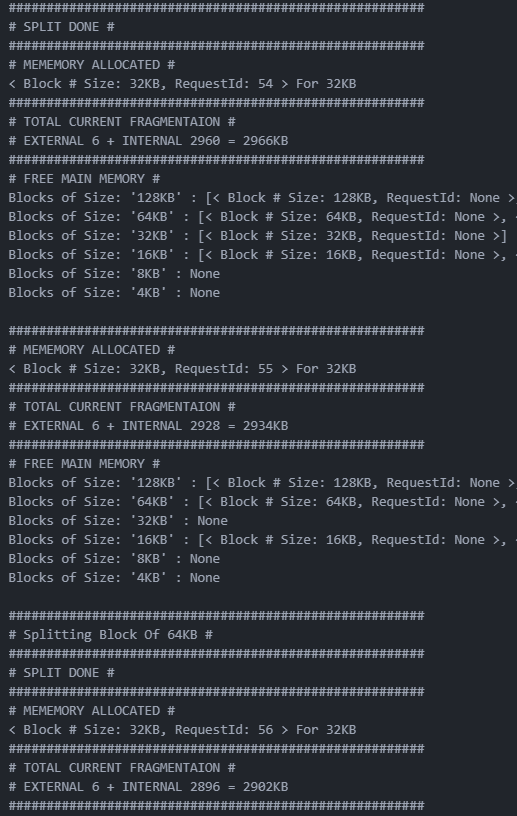
****

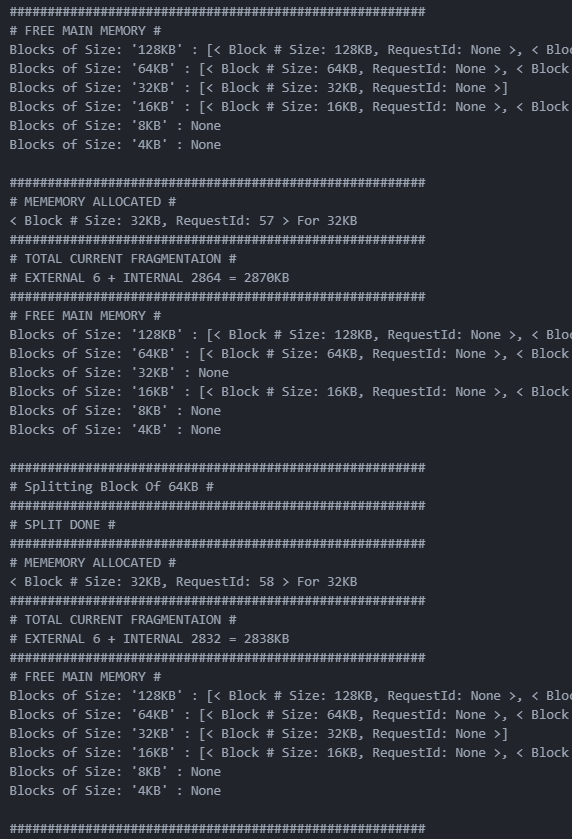
****

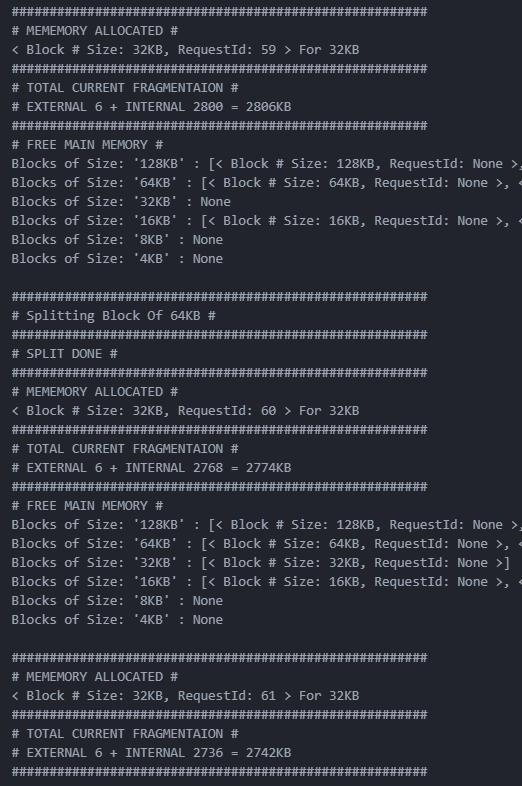
****

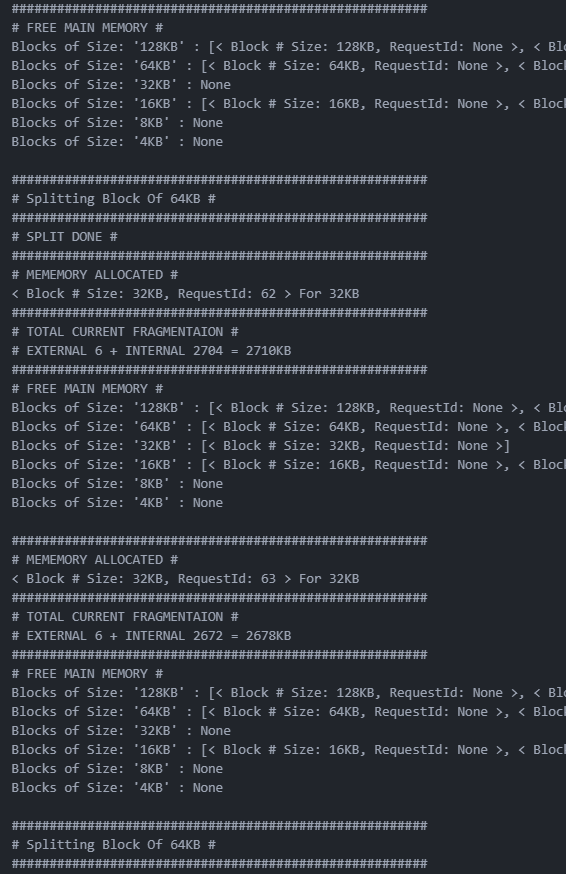
****

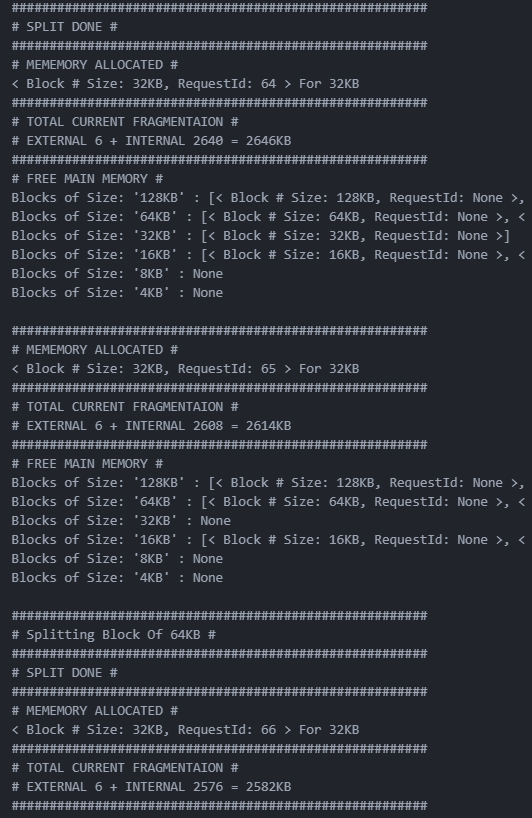
****

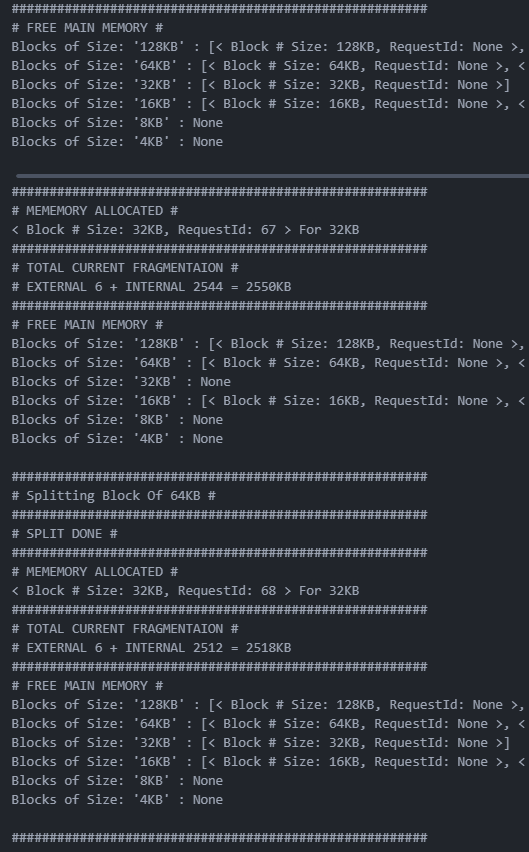
****

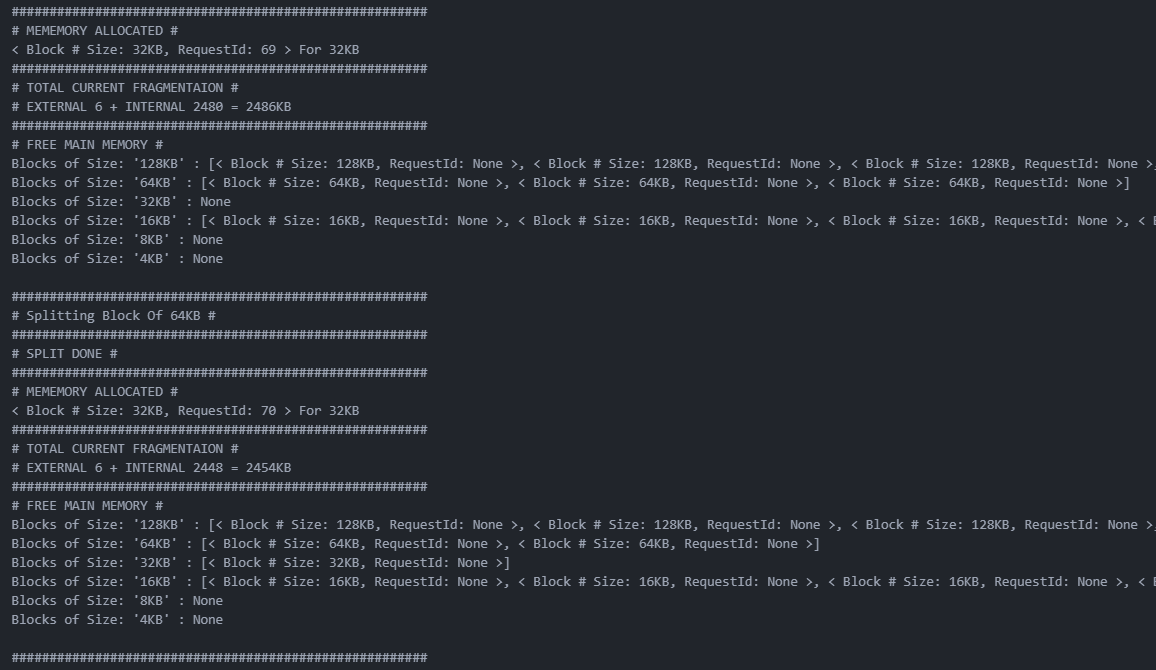
****

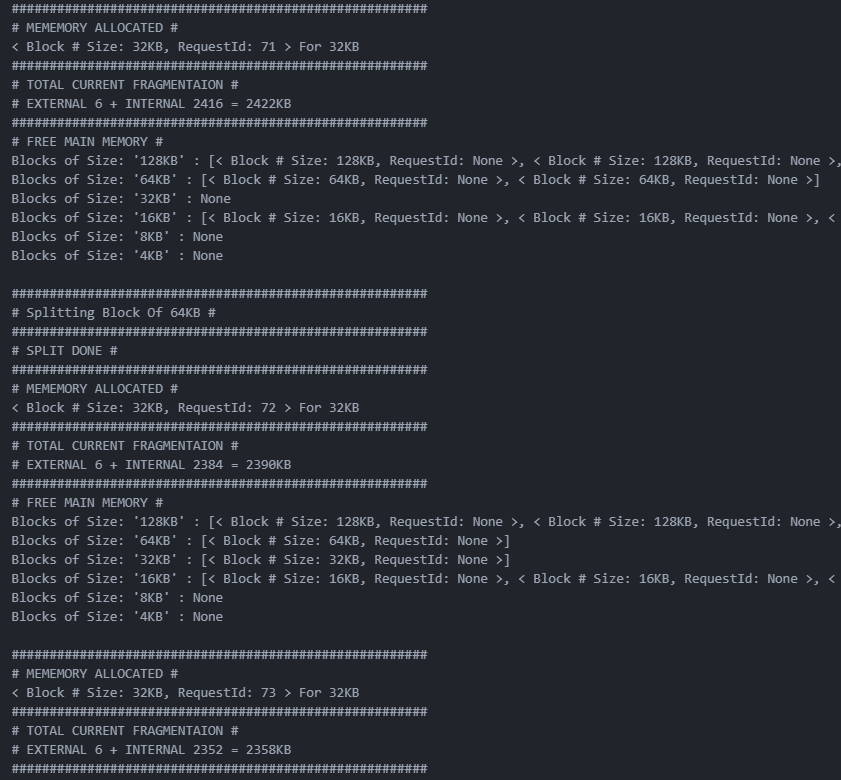
****

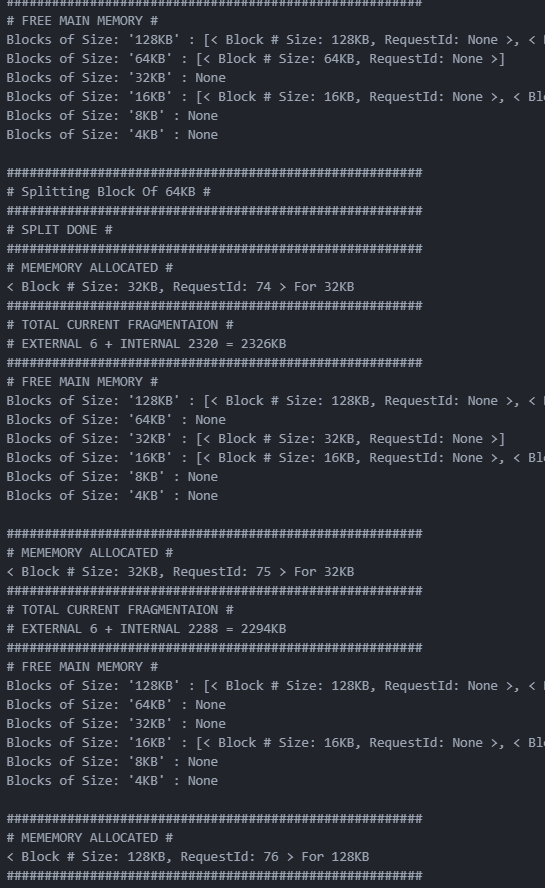
****

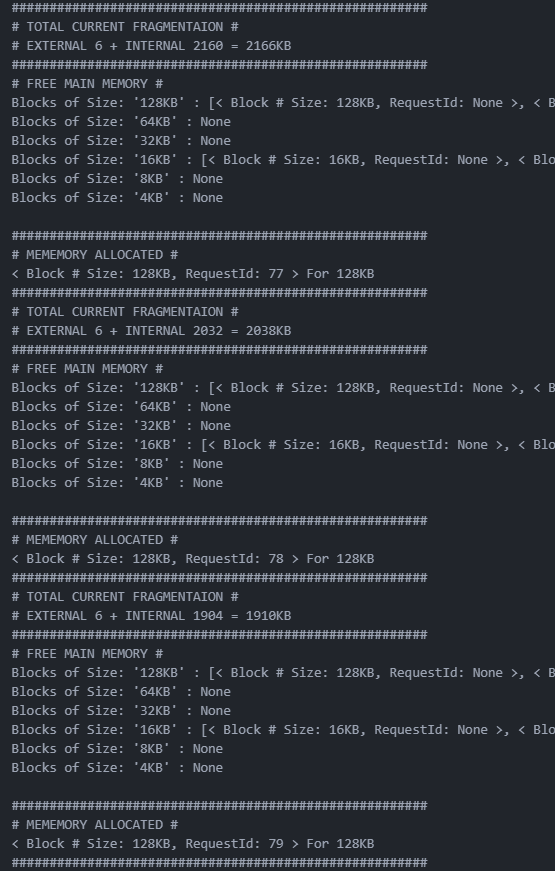
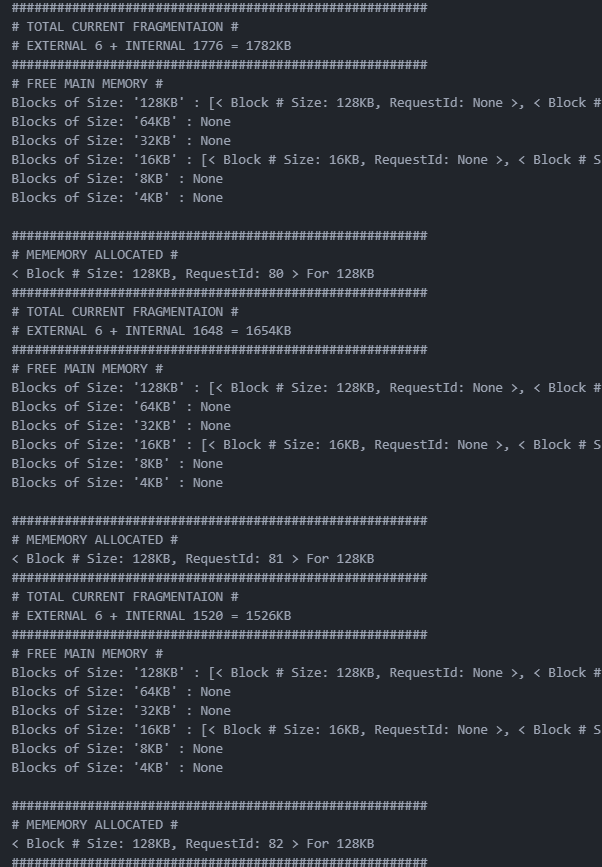
****

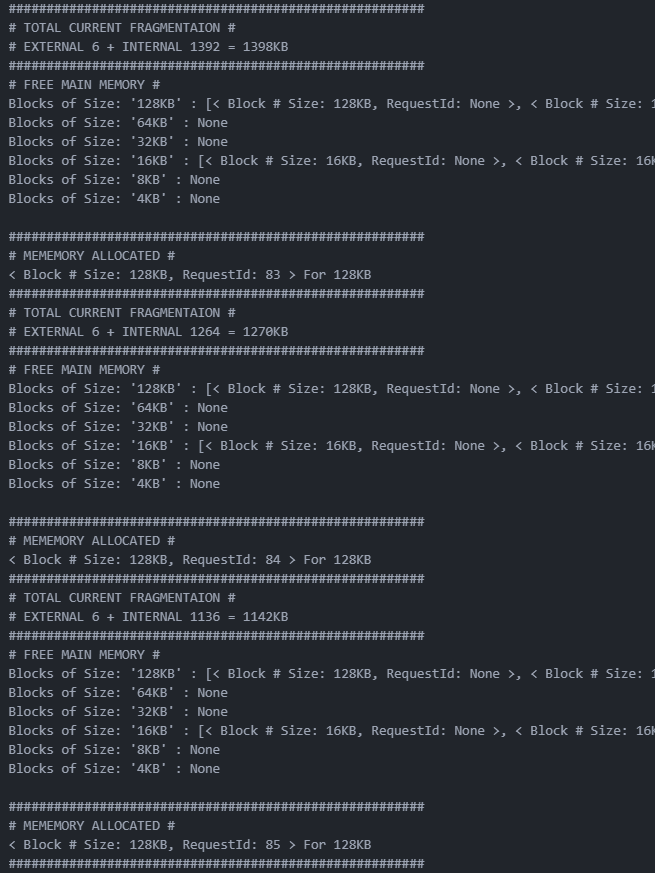
****

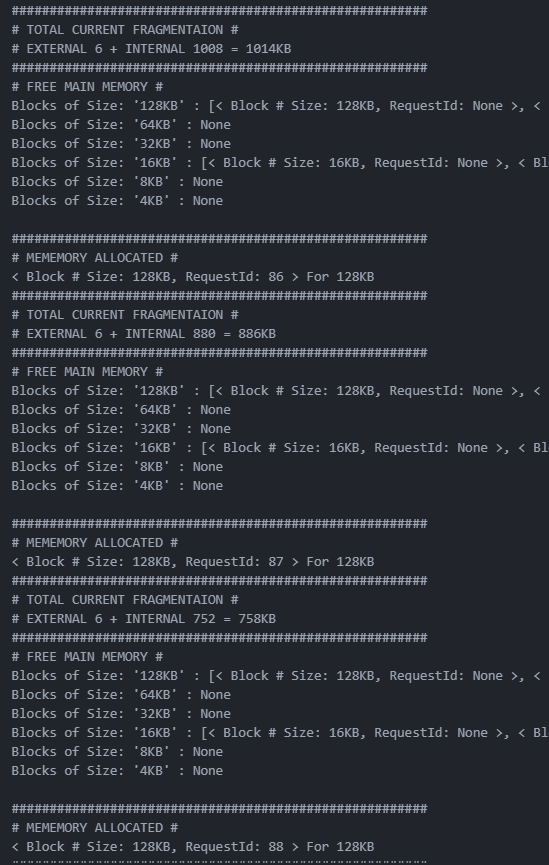
****

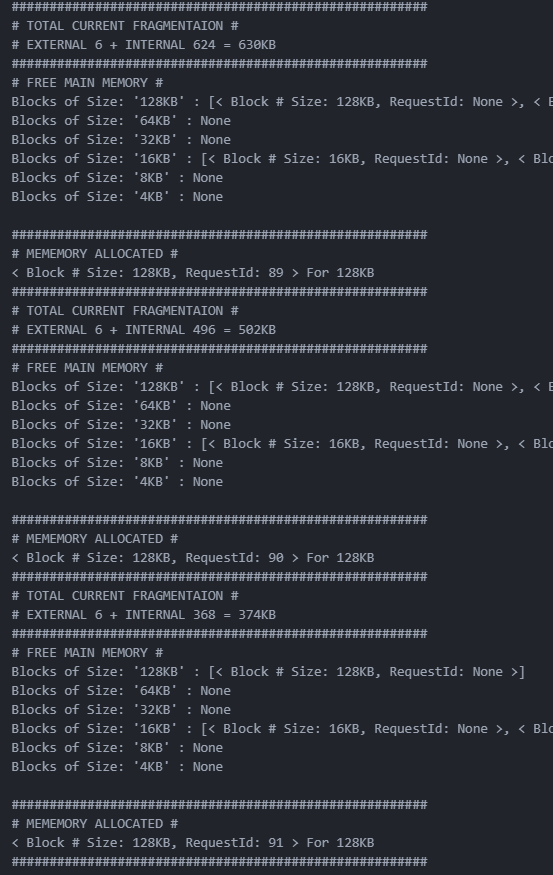
****

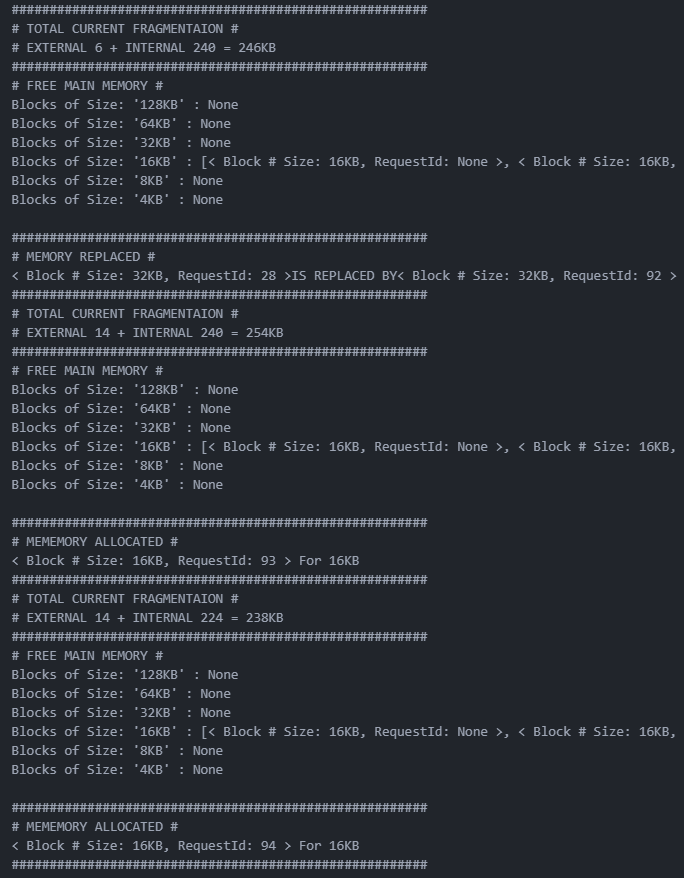
****

****

****

****

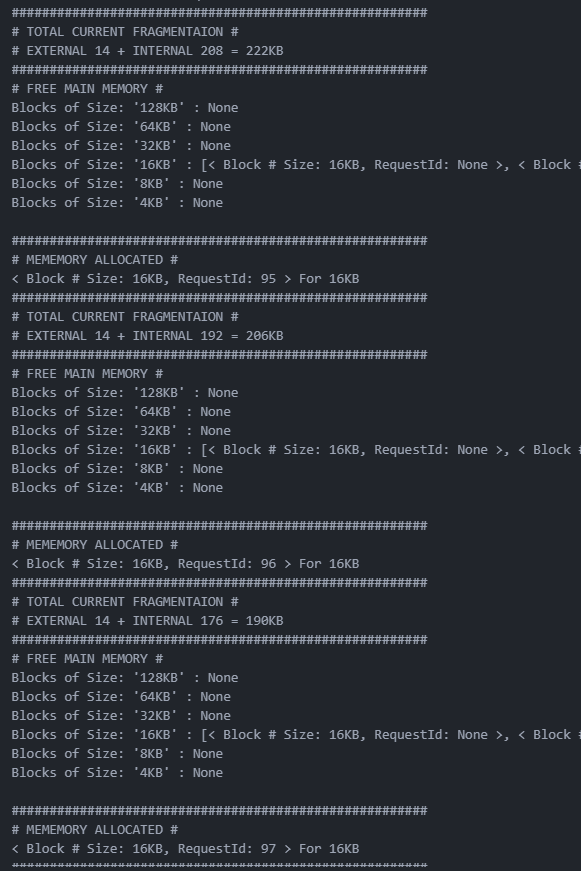
****

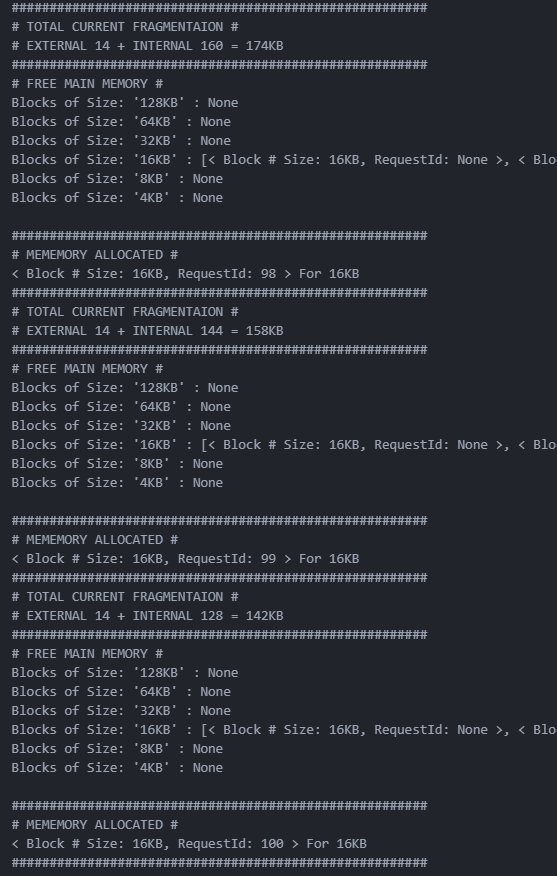
****

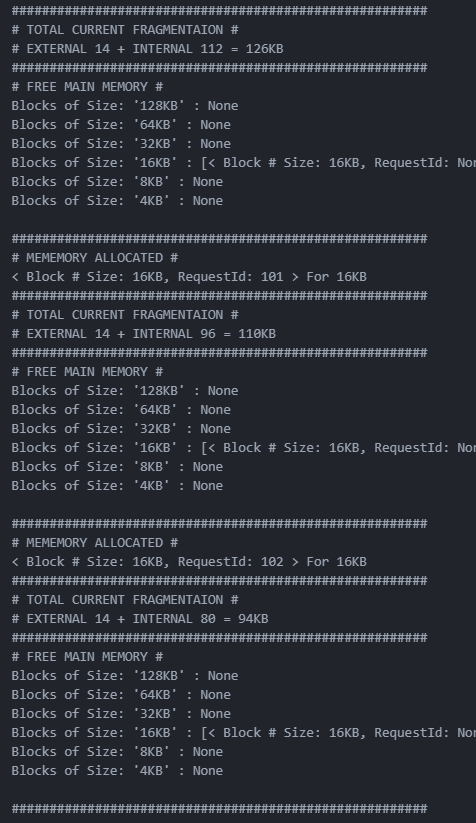
Replacement was not of the full block amount, so internal fragmentation went up up.

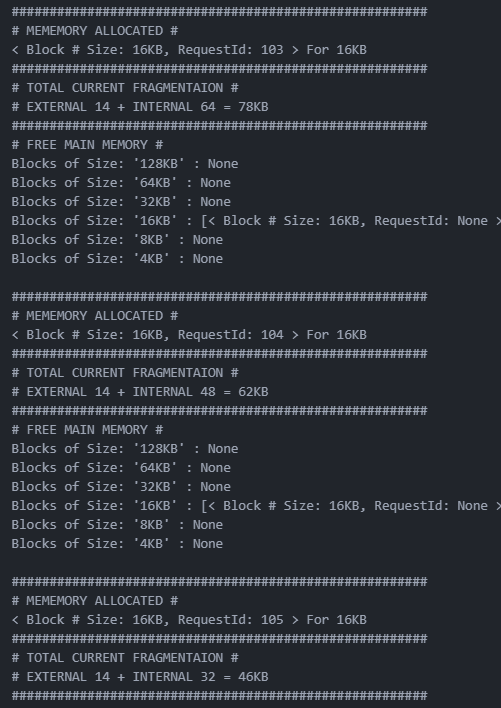
Will be a replacement if no blocks available and no blocks above available to split.

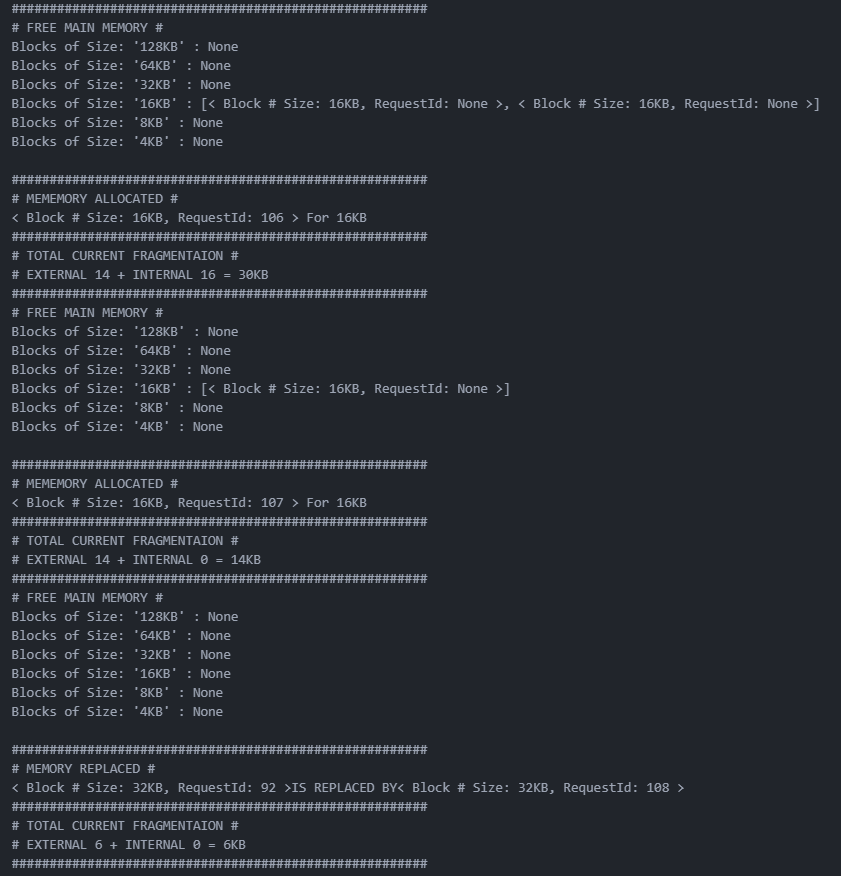
**NEXT NOTE ON PAGE 58**

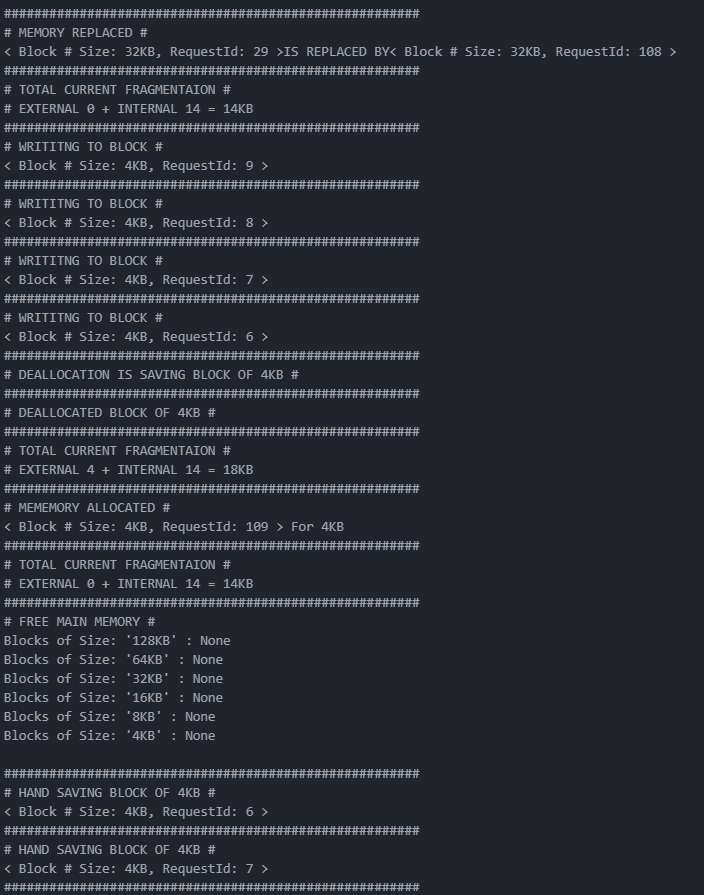
****

****

****

****

****

****

External fragmentation increased due to deallocation.

Replacement clock hand saving after faults

Example of a deAllocation with a save

Example of a write

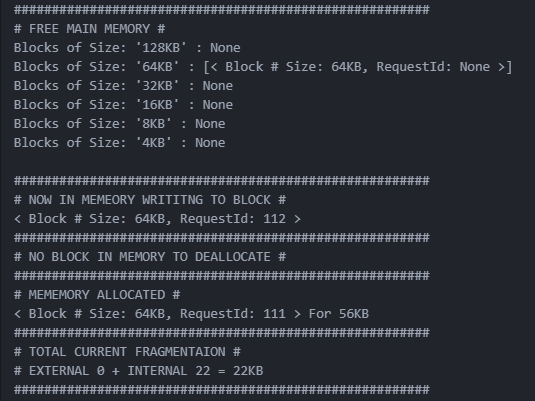
Example of memory full replacement

****

Example of a pair

Example of a normal deallocation

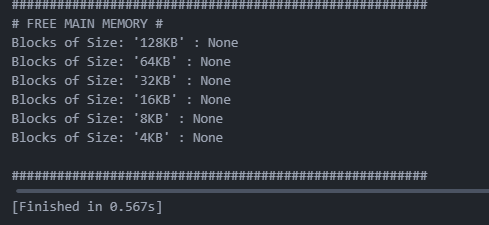
Example where no block available of required size for replacement.

****

The deferred request is allocated

deallocation but its already not in Memory

Example of a write that is not already in Memory

****