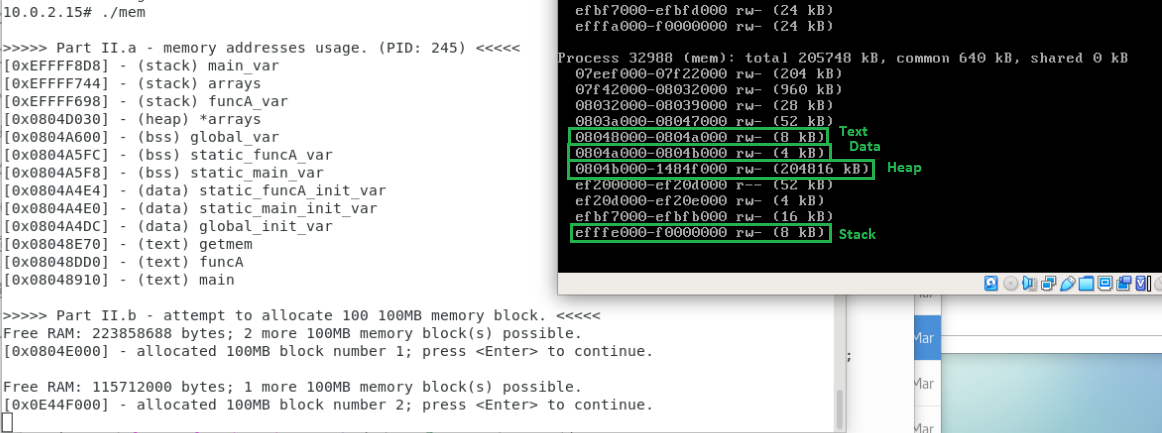
**Problem 3 Part II (Zavvio Mok)**

**1a.**

No matter how many concurrent sessions of “mem” processes started from the same executable (e.g. starting background process via “~/mem &”), they all print out identical memory addresses for most (if not all) elements. For example, the main() function (0x08048910) is in the Text area at 0x08048000; the initialized/uninitialized global variables like global\_init\_var (0x0804A4DC) is in the Data area at 0x0804A000; dynamically allocated memory like \*array (0x0804D030) is in the Heap area immediately after the end of Data area (0x0804B000); and local variables like funcA\_var (0xEFFFF698) are in the Stack area growing downward from 0xF0000000 to 0xEFFFE000.

The ability for different processes to reuse the exact same addresses demonstrated the use of virtual address; such that each process can independently utilize the full range of memory space (~0x08048000-0xF0000000), isolated from other processes. This is realized via the page table mechanism (and maybe MMU) which map the virtual address into physical address (if the page is selected to be put into RAM). Moreover, the user program itself is not aware of the existence of actual physical memory; in its point of view, it has the entire ~32-bit range memory space all to itself.



**1b.**

No, Minix does not have a non-volatile “swap area”; instead, it keeps all processes (referenced pages) on the RAM. The following screenshot captures the “mem” program on a VM with 256MB RAM, and it first allocates 100MB dynamic memory when started, adding 100MB more each time Enter key is pressed. With 256MB, the program runs fine for the first two rounds of 100MB memory allocation but it instantly crashed in attempt to allocate the third 100MB block; likewise, with two 100MB blocks on a first instance of “mem” process, starting a second “mem” process would instantly crash in attempt to allocate a first block of 100MB.

The program crashes because the OS (without swap area) ran out of physical memory. If swap area was available, the OS would page-out one of the existing page currently resides in the RAM onto the non-volatile Swap area, then bring in a new page corresponding to the newly referenced memory. When a new process is started, or for new dynamic memory allocation, they would first be placed (as pages) into the Swap are, and then page-in onto the RAM when (immediately) referenced. With that, the program would not crash simply because physical memory (RAM) ran out.

