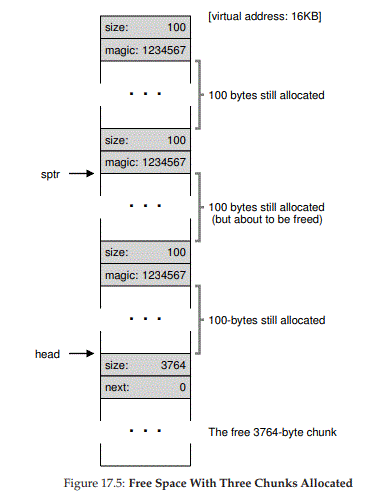
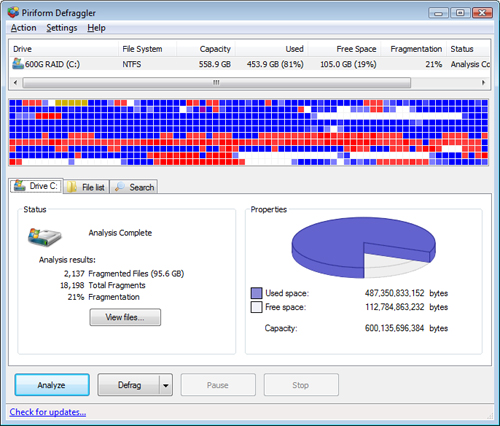
According to what we learned this week from our textbook (Arpaci-Dusseau & Arpaci-Dusseau, 2012), after a heap is created by a program, as when using mmap(), then the user can make malloc() calls to request memory spaces for objects. If there is contiguous free memory large enough to accommodate the requested memory space, the operating system (with the help of hardware), will provide the user’s requested virtual address space with a physical address space.

The problem with memory allocation as described above is two-fold. First, there is the issue of internal fragmentation. When memory is allocated, it is often the case that an address space will be provided which is larger than the requested space. The second issue, is that of external fragmentation. When memory is allocated for a program, it is usually segmented into blocks. On the one hand, memory could be divided into 3 segments (for example, the code, stack, and heap). However, program memory could be logically divided into many more segments than 3. Everytime a program makes a malloc() request, a separate memory space will be allocated for whatever object the space has been requested for. In so doing, “free space gets chopped into little pieces of different sizes and is thus fragmented” (Arpaci-Dusseau & Arpaci-Dusseau, 2012). This external fragmentation is a problem because later malloc() requests may not be able to return an address space if there is no contiguous space big enough to satisfy the memory request, even when there might have been enough free space if it were contiguous.



(Arpaci-Dusseau & Arpaci-Dusseau, 2012)

Figure 17.5 from the textbook shows, how 3 allocated blocks each roughly 100 bytes long could be arranged in memory. The middle of these, assigned to pointer sptr, is soon to be deallocated through a free() call. When it is freed the 100 or so bytes might remain sandwiched between two other memory blocks unless counter measures are taken. Assuming that this is happening at multiple locations in memory, soon memory will look like swiss cheese. This “big mess” becomes a problem when programs are looking to have large address spaces allocated, but all of the free space is now peppered around in memory. To demonstrate this, the image below provides a visual representation of free space (white) dispersed throughout allocated space (blue). If the white spaces were combined, there could be well over an entire row of free space, but since it isn’t contiguous, the OS would not be able to allocate all of this memory to a process. Granted, this image refers to OS fragmentation on disk space, but the general concept is comparable.



(Jay, 2010)

One argument is that external fragmentation is no longer a problem because memory is no longer measured by the gigabyte, but by the petabyte and exabyte. Yes, storage volumes are larger, but this only exacerbates the problem. “With the larger volumes, more and more data is getting stored on them and, [...] results in even more fragmentation which means more performance degradation” (Quan, 2012).

What I find most telling about the external fragmentation issue is that there are a plethora of defragmentation tools available that try to remedy this problem. According to Arpaci-Dusseau & Arpaci-Dusseau (2012), there is no best solution/algorithm for managing memory and keep it from becoming fragmented; and that all we can try to do is to prevent fragmentation from happening in the first place. Yet, this is unrealistic. Even on a “fresh install of Windows 7 on a newly formatted large volume [...and an installation…] of the latest service packs and [...there will be substantial fragmentation] (Quan, 2012).

References

Arpaci-Dusseau, R. & Arpaci-Dusseau, A. (2012). *Operating Systems: Three Easy Pieces. Madison, WI: University of Wisconsin-Madison.* Retrieved from <http://pages.cs.wisc.edu/~remzi/OSTEP//>

Jay, A. (2010). Six Powerful Free Disk Defragmentation Tool. *SmashingApps*. Retrieved from <http://www.smashingapps.com/2010/02/04/six-powerful-free-disk-defragmentation-tool.html>

Quan, G. (2012). Windows Defragmenter: Not Good Enough. *Condusiv Technologies Corporation.* Retrieved from <https://www.techrepublic.com/resource-library/whitepapers/windows-defragmenter-not-good-enough/post/?regId=&asset=4173227&skipAutoLoad=1>