Operating Systems Assignment 4

Alejandro Vidal

Juan C Valladares

11/19/2017

In this assignment we were tasked with creating a memory allocator library at the user level. We began by creating two structs. One for global accesses, and the other is a structure to manage the the different blocks of memory as a linked list. We created a helper method for the policy, that would be called by mem_alloc(), called Set_Next_Free(). This would point to the next open block of memory depending on the policy. Then we have Mem_Init(), which initialized the memory. It first checked if the process had already initialized memory, if it had, it would print an error. If the size requested was below 0, it would return an error. Then it we should ensure the size was matching to the pagesizes, so we would add to the size to round up so that we could request from mmap by the page size. Finally we call mmap, and receive a pointer back. That pointer is the beginning of our structure that keeps tabs on the blocks.

Mem_Alloc allocates the first available block of the size according to the policy of choice. It then ensures the block pointers point correctly to the next open block. Mem_Free receives a pointer and frees the block in which that pointer is. It figures it out by going allocated block by allocated block checking if the pointer is in that block. Mem_IsValid does something similar to mem_free to find where the pointer is, and all it checks for if is the memory is allocated. Mem_GetSize does the same to traverse through the linked list looking for the pointer that is passed in. Then returns the size of the block in which the pointer is found.

Mem_GetFragmentation similarly traverses through the linked list of block headers, but it stores the largest of the free spaces, while also adding the size of the free spaces. It then finds the fragmentation and returns that.

The output of the testmem is interesting. First I tries to allocate memory without initializing. Then it initializes the memory, then it tries again to initialize memory, but it returns an error because it has already been initialized. Finally it allocates 64 bytes, it prints the memory address where that block begins and the fragmentation, which is 1 at this point. It does so for a while until it fails to allocate memory because it tries to allocate memory that is too big. Then test mem allocates some more memory before it begins to free space. You then start seeing fragmentation below 1. Then it also tries to free memory address that can't be freed because it's not in there, and it fails. Finally it starts allocating memory again, and we see the fragmentation to be more because we see a small fragmentation number.