Julius Cassin
System Programming: PA3 ReadMe

Heap Memory Block

Memory allocated by myMalloc is stored in a continuous memory block. This memory block is initialized by declaring a char array in memory_alloc.c. The size of the block is declared to be 32760 bytes. This memory block is divided into blocks with each block containing a memEntry struct and actual data that follows the memEntry struct. The memEntry struct contains meta data about that memory block. Each memEntry struct has a prev and succ field. This fields are pointers to the next and prev memEntry structs. Other meta data contained within the struct includes if it is free to use, the size of the data following the struct, and a recognition pattern to prevent the user from freeing invalid pointers. The first memEntry struct's prev points to NULL. The last memEntry struct's succ points to NULL.

myMalloc Function

The myMalloc function finds an appropriate memory block within the heap that can store the users data. It then returns a pointer to where data can be stored within that memory block. The heap is initialized on the users first call to malloc. On initialization there is only one memory block that takes up the entire heap. Pointers returned by myMalloc may or may not contain previous “garbage” data.

The myMalloc function does a linear search to find an appropriate memory block. The linear search in the worst case must traverse the entire list. The average case takes (n+1)/2 time where n relates to the number of memory blocks. The first free memory block that is larger than the users request but smaller than the users request + a memEntry Struct + fudgeFactor OR the first memory block that is greater than the previous is chosen as the appropriate block. In the first case the memory block found is just returned to the user. In the second case the block is subdivided into two memory blocks and the first one is returned to the user. The fudge factor is to ensure that no block is subdivided with a second block that is too small to be useful. In this implementation it was chosen to arbitrarily be 20 but it can be changed by changing the macro fudgeFactor.

myFree Function

The myFree takes in a pointer from myMalloc and frees the memory block associated with that pointer. This is done in O(1) time. The four major cases for freeing a valid pointer are the following. The memory block associated with that pointer is free and the prev and succ blocks are free. All three blocks are merged. The memory block associated with that pointer is free and the prev block is free. The succ block is not free. Those free blocks are merged. The memory block associated with that pointer is free and the succ block is free. The prev block is not free. Those free blocks are merged. Neither the succ nor the prev block is free. Only the pointer block is freed. Merging is done by eliminating pointers to unnecessary blocks and reassigning the pointers to valid blocks.

Error Checking

The myFree function does not allow the user to free pointers that point to a freed block, that are not in the heap, and that do not point to the correct location in the heap. This is done by use of a recognition code contained in the memEntry struct. The reocgnition code is changed to a non valid code whenever a block is freed to prevent double freeing. If a pointer is not associated with the correct recognition code an error is returned to the user and nothing is freed. The recognition code in this implementation is 0xAAAAAAAA.

Testing

A printHeap function was used to test and error check. See hwextra-testcases.txt for test cases

used to test code.