Project 3: Allocating Memory

Design Document

Matt Caruano, Taylor Caswell, Eric Evans, Mesuilame Mataitoga, Radhika Mitra

Purpose

To write special versions of malloc() and free() that check for common allocation problems and mistakes in minix.

Available Resources:

We used the following files and made changes to them:

1. memory.c
   We used the **struct timeval** for accessing timestamp value
2. memory.h

We also used an existing ADT(Matt Caruano) created of a linked list as included in our project:

1. list.c
   In list.c there is a typedef struct Node in which we added new data fields (explained in Design Plans)
2. list.h
   Added the function prototypes of the newly added functions (explained in Design Plans)

Design Plans

- In order to meet the requirements, modification had to be made to the existing linked list.
    1. Added new data fields in structure of linked list (list.c) :
       typedef struct Node {
       /*other data also included apart from memory related data*/
       int allocated_size - space allocated with malloc
       struct timeval tv -time of the allocation
       void* address -address of the beginning of allocated region
       char* file_and_line; -file name and line number of malloc call
       short in_use; -if in use 1, if not in use 0
       }NodeType

2. Added three function: *insertNewMemoryRecord()*, *listPrintMemstats()* *isAllocated()*

insertNewMemoryRecord() is a variation on *insertAfterLast()*, made to fit the updated data structure.

listPrintMemstats() prints out the information (allocation details) for *slug_memstats()* as per the specs

isAllocated() checks to see if the address `slug_free()` is trying to free is allocated, or no memory has been allocated or if allocated memory is trying to be free when it has already been freed earlier and returns a integer value for each case.

- Creating *void *slug_malloc (size_t size, char *WHERE)*

  - set slug_memstats() to run after program completion using atexit() if it hasn't been installed yet
  - handle timestamp using **struct timeval**
  - handle when size is less than 0 with stderr response
  - handle when size is too large, and program exits
  - location of original malloc() call using char * WHERE
  - length (size)
  - function call to linked list *insertNewMemoryRecord(list, address, WHERE, size)* to add an entry to our list for this allocation
  - returns address

- Creating *void slug_free (void *addr, char *WHERE)*

  - See if address is allocated by calling `isAllocated(ListRef L, void* address)` then calls free if it is allocated otherwise prints error statement indicating a possible cause, such as no such allocation, double frees, or attempts to free using a address inside an allocated chunk..

- Creating *void slug_memstats (void)*

  - prints out the following list allocation details (stats) by calling *void listPrintMemstats(ListRef L)* :

    for each of the entries:

    - `Time of allocation`

- Address of allocation
- File and line number

A summary of all the allocations:

- Total number of allocations
- Number of allocations still in memory
- Total memory in use by the allocations still in memory

Testing

1. Show a test program that correctly uses allocation in a nontrivial way behaves correctly
   Test 1 - allocate memory by calling slug_malloc() of size 10 * counter of a finite loop, use the allocated space to store then print out a character.

2. Show that after allocating and deallocating memory, trying to deallocate an invalid address is immediately detected.
   Test 2 - allocate and free multiple block in a loop, then called free() on an invalid address

3. Show that after allocating and deallocating memory, trying to deallocate an already freed region is immediately detected.
   Test 3 - allocate and free multiple block in a loop, then call free() on the last freed block from the loop

4. Show that after allocating and deallocating memory, trying to deallocate a valid region by passing in a pointer inside the region is immediately detected.
   Test 4 - allocate and free multiple block in a loop, allocated a new block, then add one to the pointer to that block, then try to free() using that pointer.

5. Show that allocating memory and then exiting triggers the leak detector and shows where the leak occurred.
   Test 5 - allocate multiple blocks in a loop, do not free them.