CS 111 - Project 3

Design Document

5.20.14

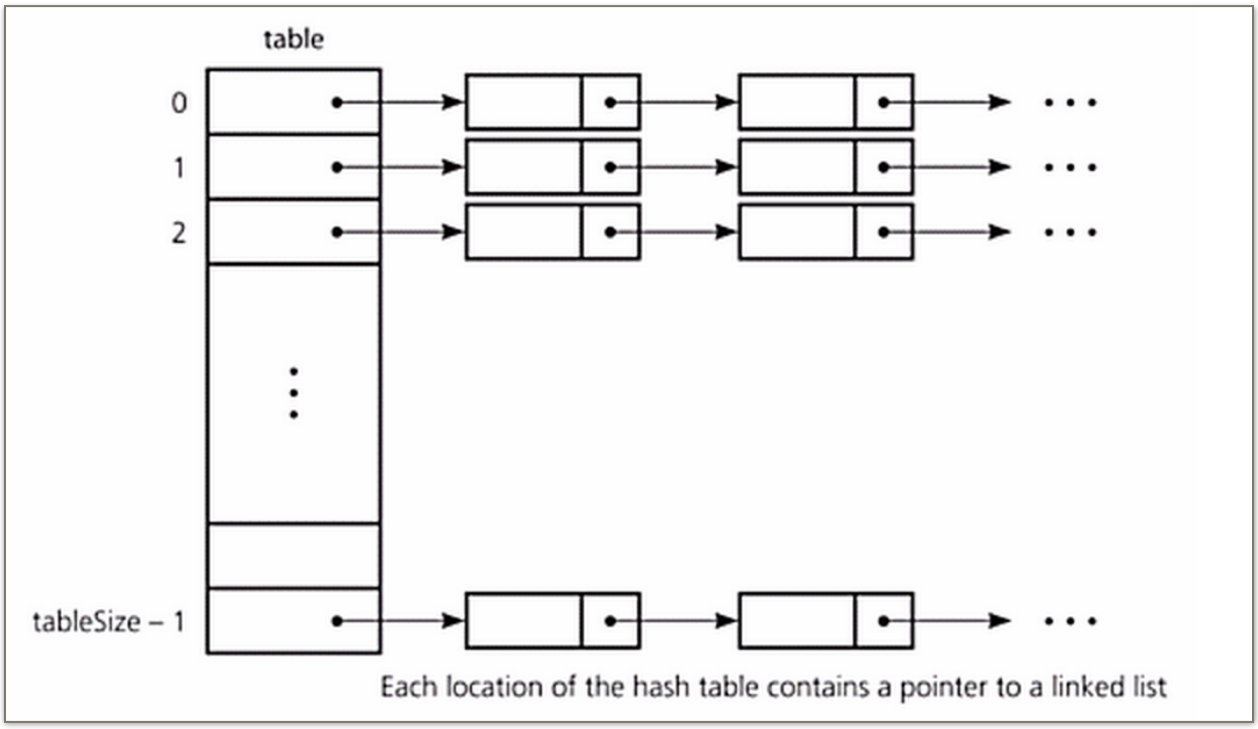
Greg Arnheiter
Ian Hamilton
Justin Kwok
Benjamin Lieu
Yugraj Singh

Purpose - This project overviews the design and implementation of memory management functions using `malloc()` and `free()`. The goal is to wrap the `malloc()` and `free()` to provide additional error checking and also to supply `slug_memstat()` that will dump additional info pertaining to all memory mappings for allocated and deallocated memory.

Part 1 - Memory Data Structures and Helper Functions

In order to implement new versions of `malloc()` and `free()` there needs be a data structure to maintain all info pertaining to the (de)allocated memory including: the size of allocated memory, the timestamp of creation, addresses of start of allocated memory, file and line of calling program, total number of allocations, number of active allocations, the mean and standard deviation of all allocations and the total amount of memory allocated.

A hash table can be used to keep all data, using chaining for collision detection. Each chunk of memory allocated using `slug_malloc()` will be entered by hashing the memory address with the size of the table (61 in our case). Each node in the list that is created (because of chaining) contains the info pertaining to that allocation, which will be printed by `slug_memstat()`. The result is a hash table that contains one node at a given index for each allocated block of memory, where each node contains a struct with data for its allocation.



Part 1 - Hash Table with Lists for Allocated Memory

Part 2 - Implementing `slug_malloc()`

Our implementation of `slug_malloc()` provides additional error checking before using regular calls to `malloc()` that a programmer might overlook when directly modifying memory allocation in a program. The data structures used for error checking is contained within `slug_malloc()` and is created and initialized the first time it's called. Our allocation function provides the following services and error prevention/detection mechanisms:

1. Service - Inserting nodes into the hash table and appending them to the list.
2. Error Check - Is memory region too big (> 128MiB)
   If Yes, Return Error and Exit. If No, continue normally.
3. Error Check - Is memory size requested equal to zero
   If yes, Return Error and DO NOT Exit. If No, continue normally.

Part 3 - Implementing `slug_free()`

Our implementation of `slug_free()` provides additional error checking before using regular calls to `free()` that a programmer might overlook when directly modifying memory allocation in a program. Our deallocation function provides the following services and error prevention/detection mechanisms:

1. Service - Removing nodes from link list and their entries from the hash table.
2. Error Check - Is *address* the start of valid memory region.
3. Error Check - Is *address* invalid or within a valid region but not the start.
4. Error Check - Detects attempts to free memory that is already deallocated.
5. Error Handle - Exiting with memory still allocated is detected, and memory is freed before exit.

Part 4 - Implementing `slug_memstat()`

Our implementation of `slug_memstat()` provides an information service for all allocated memory. This function traverses the data structures used for `slug_free()` and `slug_malloc()` and prints the struct info for each node in the list including: Size of allocation, timestamp of when the allocation took place, the address of the allocation, and the file and line number of the calling application. In addition to the node information, `slug_memstat()` produces records for the total number of allocations, how many allocations are still active, total amount of memory allocated, and the mean and standard deviation of the sizes of all allocations.

Part 5 - Implementing Leak Detection

Upon exit of the program, `slug_memstat()` will be called to report any allocations that have not been freed at the time of exit. The program should print all info produced by `slug_memstat()` and subsequently free all remaining nodes before exiting. The program frees all memory allocated by the user as well as any memory allocated by test programs and clears their entries from all data structures (list and table). The program should also detect whether or not an exit handler has been installed, and install it if necessary.

Part 6 - Implementing Header

This program needs to be compiled using the special header `slug_header.h` that we included with the project code. This header will replace previous calls to `malloc()` and `free()` with `slug_malloc()` and `slug_free()`. Also, the header is responsible for remembering the file and line number of the calling program.