Design Doc 3

John Carlyle, Andrew Edwards,
Morgan McDermott, and Kaya Zekioglu

May 21, 2014

1 General Description

This program provides extended versions of `malloc` and `free` that check for common allocations problems and mistakes, as well as a function that prints out all currently allocated memory and some additional statistics. The memory management suite will install this latter function as an exit handler to report memory leaks.

2 Usage

To make use of our augmented `malloc` and `free` functions, a user should include the `slug_mem.h` file in their program. This file provides a handful of macros that replace the standard `malloc` and `free` with `slug_malloc` and `slug_free`. In addition to including our header they must compile and link `slug_mem.c` to their executable to provide the three functions `slug_malloc`, `slug_free`, and `slug_memstats`. This design is simple and portable.

3 Design

Here we explain the design of our memory management suite.

3.1 Allocating Memory

Allocating memory is handled by the `slug_malloc` function, which replaces the `malloc` function in any program that includes the `slug_mem.h` header file. This function takes a request for memory to be allocated, and vets the request to make sure it is viable. If the request doesn't violate any restrictions (such as $\leq$ 128MB) the memory is allocated and a linked list is updated to remember that that block is currently in use. The linked list has one node per allocation, the node stores the address and size of the block, where in the code it was requested, and the approximate time of the request.

The implementation of the system is an extremely simple doubly linked list. When a new object is added, it is pushed onto the tail of the linked list and will remain there until it is freed.

3.2 Freeing Memory

Freeing memory is handled by the `slug_free` function which replaces the standard `free` function. This operation is fairly simple. It takes in the memory address to be freed and checks against the linked list maintained by the `slug_malloc` function. If the address is found somewhere in the linked list then we have received a valid block of memory to free. We can splice the node out of the linked list, update a few variables for statistics calculations, and finally call the system `free` function.

If the block is not in our linked list then the user has requested an invalid address to be freed. This error is reported by `perror` and the program is terminated.

3.3 Leak Detector and Memory Statistics

When a call to `slug_malloc` is made for the first time during the execution of a program, `slug_memstats` is registered as an exit handler. This function checks over the linked list that has been maintained by `slug_malloc` and `slug_free` looking for entries. Any entry within the list represents a block of memory that has been leaked, as the program is terminating. For every element remaining in the list we print out all of the information about that block of memory. This way the user can sift through the data and figure out where their memory mismanagement is happening. Once at the end of all of the unfreed blocks the mean block size and stdev is printed.

This function could also be called manually to hunt for memory that should have been freed at some point before the termination of the program.

3.4 Internal Data Structure

We elected to use a doubly linked list data structure to store information about allocated memory. The pros of this design choice include simplicity in implementation and in maintenance. The only con is that it is not as fast as some other storage data structures, such as red-black trees. Insertion and deletion is wholly contained within `slug_malloc` and `slug_free`.

Each node contains the following information relating to an allocated region of memory: the start address, the total length in bytes, a timestamp of when it was created, the file name and line number the `malloc` call came from, and pointers to the previous and next nodes. The list structure has pointers to the head and tail nodes, along with maintaining information on the total number of allocations made, the current number of active allocations, the current amount of active memory allocated in bytes the average memory block size, and an

auxilary variable to later compute the standard deviation of the memory block size.