Programming Assignment 5: Linux Slab Allocator

Krishna Chaitanya CS16BTECH11011
Nithin CS16BTECH11005

The Libmymem library

libmymem.hpp:

This is the header file of the libmymem library and contains the function declarations along with the data structures required used for mymalloc() and myfree()

***Data Structures*:**

Slab:

This represents the metadata for the whole slab. Including pointers to the bucket to which it belongs and the first object in the slab.

It also keeps track of total number of objects and free objects in the slab. The boolean array bitmap keeps track which objects are free and which are not. It also contains a pointer to next slab in the bucket (NULL if it is the last slab).

Bucket:

This represents a bucket which contains (in this case points to) slabs containing objects of same size.

The objSize variable stores the size of objects that slabs in this bucket contain. firstSlab points to the first slab in this bucket. And mutex m_mutex is used to make operations thread safe.

***Object*:**

This represents the metadata for the objects that the slab contains. Each object in the slab has a corresponding Object structure. It contains a pointer to the slab to which it belongs along with pointers to the next object in the slab and the memory of the object it corresponds to.

***HASH_TABLE[]*:**

This is the array of all the buckets that the slab allocator uses.

libmymem.cpp:

The libmymem.cpp contains the definition of mymalloc() and myfree() functions along with the functions required for the slab allocator.

Functions:

Slab initializeSlab(Bucket*)*:

This function creates a slab and return a pointer to the newly created slab. To create a slab first a contiguous memory of size 64KB is created using the mmap function . The metadata for this slab (contained in the Slab data structure) is present at start of this memory. Then all the variables in the Slab are initialized (like totobj, freeobj, nextSlab, bucket, objPtr, bitmap etc.).

The rest of the memory in the slab is divided into objects of size corresponding to the parent bucket of this slab

void initializeBucket(int, Bucket)*:

This function initializes object size and pointer to the first slab of this bucket (by making use of the initializeSlab function).

void createSlabAllocator():

This function initializes all the buckets in the HASH_TABLE[]. And this function is called the first time mymalloc() is used to initialize all the data required for the slab allocator to function.

void mymalloc(unsigned)*:

First we check if the size request can be satisfied. Then the slab allocator is initialized(if it hasn't been already). Next we calculate to which bucket the request has to be sent based on the size requested.

In this bucket we check for a slab with free objects (by using freeobj and bitmap) and then return a pointer to the memory (of best-fit size) that the user can use. The object is marked as not free in the bitmap an the number of free objects in the slab is reduced by 1.

This is made thread safe and re-entrant by using a mutex to lock all operations to a slab when it already servicing a request (or freeing memory).

Description:

The mymalloc() function allocates size bytes and returns a pointer to the allocated memory. The memory is not initialized. If size is 0, then mymalloc() returns NULL.

Return Value:

The mymalloc() function returns a pointer to the allocated memory, which is suitably aligned for any built-in type.

On error, this function returns NULL. NULL may also be returned by a successful call to malloc() with a size of zero

Memory allocation is not guaranteed to succeed, and may instead return NULL. Using the returned value, without checking if the allocation is successful, invokes undefined behavior. This usually leads to crash (due to the resulting segmentation fault on the null pointer dereference), but there is no guarantee that a crash will happen so relying on that can also lead to problems.

Attributes:

Thread safety

void myfree(void) :*

First we check whether the memory requested to be freed is not NULL (giving pointer to memory that was not allocated by the slab allocator will result in undefined behaviour).

Then we go the Object structure corresponding to the memory location and then go the parent slab. From here we search the required memory and mark it as free in the bitmap and increase number of free objects in the slab.

This is made thread safe and re-entrant just like mymalloc() by locking operations to the particular bucket while memory is being freed**.**

Description:

The myfree() function frees the memory space pointed to by ptr, which must have been returned by a previous call to mymalloc(). Otherwise, or if myfree(ptr) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed.

Return Value:

The myfree() function returns no value.

Attributes:

Thread safety

To avoid corruption in multithreaded applications, mutexes are used to protect the memory-management data structures used by these functions. In a multithreaded application in which threads simultaneously allocate and free memory, there could be contention for these mutexes.