NAME: Bereket Assefa
ID: 20170844

EE488A HW-2

- there are two files : tc_malloc.h and tc_malloc.c

tc_malloc.h

In tc_malloc.h there is the declaration of different Apis that implement page cache, central cache ,thread cache and main api. In addition there is also an implementation of spanlist,freelist and different mappings used in tc_malloc.c. tc_malloc.h also includes the data structure declaration of page heap data structure,central cache data structure and thread cache data structure.

```
Void spanlist_push(span_t * head,span_t span);
- this method inserts span to some spanlist at the front.

Void spanlist_pop(span_t *head,span_t span);
- this method pops span from the spanlist(span may not be at the
front) if available.
- Since spanlist is a doubly linked list that is possible.

Void freelist_push(span_t span, freelist ptr)
- inserts ptr to freelist(note that ptr is a freelist) at the
front.

freelist freelist_pop(size_t * len,freelist * head)
- pops the front element of the freelist from the freelist if it
exits and returns it.


Int index_to_size(int index)
- this method changes the given index to size and returns it.
This method is useful to map between central cache or thread
cache index to object size.
```

Int get_index(size_t size)
– this method changes the given size into an index and returns it. This method is useful to map a given size to it’s corresponding central cache or thread cache index.

tc_malloc.c

tc_malloc.c implements the below apis which is declared in tc_malloc.h.

```
/*==================== PAGE CACHE API
================================== */
void fetch_from_system();
span_t fetch_span(size_t npage);
void return_span_to_pgcache(span_t span_ptr);
/
*================================================================
=== */


/*==================== CENTRAL CACHE API
============================== */
span_t get_span_central(size_t size);
int fetch_objs(freelist *start, freelist *end, size_t num, size_t bytes);
void return_objs_to_centralcache(freelist ptr, size_t nbytes);
/
*================================================================
=====*/


/*==================== THREAD CACHE
API================================*/
void insert_to_threadcache(freelist objs, size_t nbytes);
void fetch_from_centralcache(int index);
void * allocate(size_t nbytes);
void deallocate(void * alloc_ptr,size_t nbytes);
/
*================================================================
======*/
```

```
/*================== INITIALIZATION
=====================================*/
void init_globals();
/
*=================================================================
=======*/



/*==================  MAIN API
============================================*/
void *tc_central_init();
void *tc_thread_init();
void *tc_malloc(size_t size);
void tc_free(void *ptr);
/
*=================================================================
=======*/
```

I implemented level by level for ease of debugging. I started from page cache API then I went on to central cache API then finally threadcache API and MAIN api.

Pagecache api

```
Void fetch_from_system();
- allocates huge chunk of memory(128 MB) and inserts it in its
appropriate page cache entry and updates the span.


span_t fetch_span(size_t npage);
- this method tries to fetch span from the page cache or it will
use slab allocation to apply a very huge pagesize such as
(pagesize > 256) and return the span. If there is no span of the
corresponding number of page then the function calls
fetch_from_system() then recursively call itself.

Void return_span_to_pgcache(span_t span_ptr);
- this method returns span_ptr to page cache and will coalesce
with neighbor spans. We can find out neighbor spans from pageId
info in the span structure. I implemented pageId relative to
initial heap address.initial heap address is the start of pageId
0. Start pointers are also stored in the span structure.
```

Centralcache api

```
span_t get_span_central(size_t size);
```

- this method fetches span from page heap and is called by the central cache for each size classes and will make objects from the span. The amount of objects will depend on the size class. If there is already a span in the central cache before request it will return the available span.

Int fetch_objs(freelist *start,freelist * end,size_t num,size_t bytes);
- this method fetches objects from central cache and is called by the each threadcache. Fetches min(num,available objs). If all objects are referenced by threadcaches then it will be inserted to idle field of central cache. It will return the number of fetched objs.

Void return_objs_to_centralcache(freelist ptr,size_t nbytes);
- this method returns objects that are pointed by ptr freelist and sized nbytes to the central cache. If the span in the central cache was in idle then insert it into the non-idle list.

Threadcache api
Void insert_to_threadcache(freelist objs,size_t nbytes);
- inserts objs to threadcache entry and uses nbytes to find the index of the threadcache.

Void fetch_from_centralcache(int index);
- this method calls fetch_objs then calls insert_to_threadcache but will find the appropriate size to fetch for each size class and will change the freelist max length field appropriately to maximize efficiency. For smaller sized objects max length is doubled if max length is the bottleneck compared to maximum fetch size.

Void * allocate(size_t nbytes);
- this function will make sure the threadcache entry corresponding to nbytes has some objects then it will pop one object from front and return it. This function is used only for small object allocation.

Void deallocate(void * alloc_ptr,size_t nbytes);
- this function returns alloc_ptr to the threadcache entry corresponding to nbytes. If the threadcache entry is full then it will return all objects to centralcache.

Main api
Void * tc_central_init();
-this function initializes all global data structures and

variables such as locks. It will also fetch spans for the central cache. It will return pagecache list.

Void * tc_thread_init();
- this method initializes the threadcache and fetches objects depending on the size class and returns the threadcache list.
Void *tc_malloc(size_t size);
– this method returns an object corresponding to size. It will use allocate function for small object or use fetch_span for large objects
Void tc_free(void *ptr);
–this function frees the ptr and inserts it to the threadcache. For small objects it will call deallocate or will return it to page cache.