Anunay Yadav
2018021

Design Documentation

Questions :

- **What is your page metadata structure ?**
  -> struct page_metadata{
  int allocation_size;
  long long free_bytes_available;
  } page_metadata;
  -> allocation size stores the bucket size to which this page belongs to.
  -> free_bytes_available stores the free bytes available in page.

- **How do you find that an object is large or small during myfree?**
  -> by getting the page metadata allocation size i can check wether this page belongs to large memory or small.

- **When do you free a page allocated for objects in buckets (lists)?**
  -> when whole page is free or when free bytes available in page metadata is 4096 for buckets. This is handled by myfree in memory.c .

- **How do you find the page metadata of the input object during myfree?**
  -> by setting the last 12 bits to 0 of the address as each page allocated is stored at a multiple of 4096 hence page metadata of a particular page must be at the starting of the memory address which has last 12 bits set to 0. Done by get_metadata_addr in memory.c

- **Paste your code corresponding to the removal of all objects on the page from the bucket (list), when a page is freed.**

```
-> node* temp = bucket[pow-BUCKET_START];
   node* prev = temp;
   int count = 0;
   if(temp == NULL) assert(l -> allocation_size == 4080 || l -> allocation_size == 2048); // check for trivial cases
   while(temp != NULL){ // traversing whole linked list
      if(l == ((page_metadata*) get_metadata_addr((void*) temp)) ) {
       count++;
       if(prev != temp){
              prev -> right = temp -> right; // remove from between
              temp = temp -> right;
          }
       else{
              bucket[pow-BUCKET_START] = temp -> right; //remove from start
              prev = temp -> right;
              temp = temp -> right;
          }
          continue;
```

```
            }
                prev = temp;
                temp = temp -> right;
            }
            int cnt = (l -> allocation_size != 4080);
            assert(count == ((l -> free_bytes_available) / (l -> allocation_size)) - cnt - 1);
    // check if whole page is removed or not
            free_ram(p,4096);
            return;
```

-> this code can also be found in myfree function :127 line in memory.c

- **Dump the output of the "make test"**

-> num replacements:1000000
```
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:05.06
        Maximum resident set size (kbytes): 326432
        Minor (reclaiming a frame) page faults: 833208
    num replacements:2000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:08.67
        Maximum resident set size (kbytes): 326976
        Minor (reclaiming a frame) page faults: 1585175
    num replacements:3000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:12.61
        Maximum resident set size (kbytes): 327260
        Minor (reclaiming a frame) page faults: 2337047
    num replacements:4000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:16.03
        Maximum resident set size (kbytes): 327232
        Minor (reclaiming a frame) page faults: 3088786
```

Design :

data container :

```
// page metadata data container
typedef struct page_metadata{
    int allocation_size;
    long long free_bytes_available;
} page_metadata;

// node which points to each section
typedef struct node{
    struct node* left;
    struct node* right;
} node;
```

Code Explaination :

- **mymalloc ->**

  1. first it checks whether size to be allocated is greater than 4080 or not if it is it allocates straight from the alloc from ram after making it a multiple of 4096.

2. if size is not greater than 4080 appropriate bucket size is found by just_larger(size)
3. page is requested from alloc from ram of PAGESIZE and metadata is added to those pages.
4. first section is ignored as the first section is used by page metadata.
5. all other sections are added to the bucket (linked list by push) by shifting pointer by bucket size.
6. first section is poped from bucket and given to user to use.

- **Myfree** ->
  1. it checks whether size of ptr is greater than 4080 by checking allocation size of page metadata and if it is it deallocates the whole page straight away using free_ram.
  2. updates page metadata.
  3. if whole page is free (all sections are not allocated) than traverse throigh the whole bucket and remove all those sections from bucket free the whole page from ram.
  4. if all the above conditions are not met than it means that whole page is not free hence add the section back to bucket to make it available for other user.

- **Push** -> stores left and right pointer of the sections . Adds nodes to bucket

- **get_metadata_addr** -> gets the address of page metadata of the section by setting the last 12 bits to 0.

- **pop** -> pops the first node from the linked list of bucket and returns.

- **just_larger** -> returns the appropriate bucket size for allocation for given size

- **update_dec** -> updates page metadata after allocating section to user