Akshala Bhatnagar 2018012

OS Assignment - 1

1. What is your page metadata structure?
    a. The page metadata is the first 16 bytes of the page. It contains the bucket size for that particular page and the number of bytes which are available on that page.

2. How do you find that an object is large or small during myfree?
    a. We can determine whether the object is large or small depending upon the bucket size of the object. The bucket size is stored in the page metadata. The page metadata pointer would be the largest multiple of 4096 less than the object pointer. The bucket size is found from this. If the bucket size is more than 2048 then that means that this is a large page.

3. When do you free a page allocated for objects in buckets (lists)?
    a. When the number of bytes available for a page becomes 4080 then that means that page has become completely free.

4. How do you find the page metadata of the input object during myfree?
    a. The page metadata pointer would be the largest multiple of 4096 less than the object pointer.

```
void *page_ptr = get_page_ptr_for_allocated_mem(ptr);
   struct page_metadata *page_metadata_ptr = (struct page_metadata *)
(page_ptr);
void *get_page_ptr_for_allocated_mem(void * allocated_mem_ptr)
{
   long long int num_ptr = (long long int)allocated_mem_ptr;
   void *page_corresponding_to_ptr = (void *) ((num_ptr/4096) * 4096);
   return page_corresponding_to_ptr;
}
```

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

```
int delete_all_blocks_of_page(void *page_ptr)
{
   struct page_metadata *page_metadata_ptr = (struct page_metadata
*)page_ptr;
   int bucket_size = page_metadata_ptr->bucket_size;
   int num_blocks =
(page_metadata_ptr->number_of_bytes_available)/bucket_size;
   int bucket_id = get_bucket_index(bucket_size);
   printf("Deleting %d blocks for bucket_id %d for page_ptr
%p\n",num_blocks,bucket_id,page_ptr);

   void *node_ptr = page_ptr + PAGE_METADATA_SIZE;
   for (int block_num = 0; block_num < num_blocks; block_num++)
   {
        printf("Trying to delete block %p\n", node_ptr);
        if (delete_specific_node_from_linked_list(bucket_id, node_ptr)==1)
        {
                return 1;
        }
        node_ptr += bucket_size;
   }
   delete_page_from_pagelist(page_ptr);
   return 0;
}

int delete_page_from_pagelist(void *page_ptr)
{
   int pageFound = 0;
   for (int pageNum =0; pageNum < pageCount; pageNum++)
   {
```

```
        struct page_metadata *page_metadata_ptr = (struct page_metadata
*)pages[pageNum];
        if (pageFound==1)
        {
                pages[pageNum-1] = pages[pageNum];
        }
        if (page_metadata_ptr == (struct page_metadata *)page_ptr)
        {
                free_ram(page_ptr,
page_metadata_ptr->number_of_bytes_available +
PAGE_METADATA_SIZE);
                pageFound = 1;
        }
    }
    if (pageFound==1)
    {
        pageCount--;
        return 0;
    }
    else
    {
        return 1;  // error. Page not found
    }
}
```

6. Dump the output of the‘‘make test’’.

```
num replacements:1000000
   Elapsed (wall clock) time (h:mm:ss or m:ss): 1:35.91
   Maximum resident set size (kbytes): 330064
   Minor (reclaiming a frame) page faults: 834470
num replacements:2000000
   Elapsed (wall clock) time (h:mm:ss or m:ss): 2:53.55
   Maximum resident set size (kbytes): 350336
```

Minor (reclaiming a frame) page faults: 1591126
num replacements:3000000
Elapsed (wall clock) time (h:mm:ss or m:ss): 4:44.44
Maximum resident set size (kbytes): 367724
Minor (reclaiming a frame) page faults: 2347499
num replacements:4000000
Elapsed (wall clock) time (h:mm:ss or m:ss): 6:46.12
Maximum resident set size (kbytes): 385068