OS Assignment 1

Design Document

Page Metadata Structure

I have used the following structure to save the metadata of the page in the first 16 bytes.

```
typedef struct md{
     size_t allocation_size;
     long long free_bytes_available;
}Metadata;
```

Here allocation size is the size of each node on the page. It takes 4 bytes on 32 bit system and 8 bytes in a 64 bit system.

The variable free_bytes_available stores the number of free bytes available on the page. It takes 8 bytes in both 32 bit and 64 bit systems.

Hence maximum size of datatype Metadata is 16 bytes.

Verifying that an object is large or small during myfree

The following logic is used to verify an object as large or small.

```
void *base = getbase(ptr);
     Metadata *meta = (Metadata*)base;
     if (meta->allocation_size>4080){
          free_ram(base,meta->allocation_size);
          return;
     }else{
          //manage small object
     }
}
```

Explanation: I calculate the base address of the page in which the node *ptr* resides using the *getbase* () function given in appendix.

We can access the metadata stored at the base address. Then using allocation_size stored in metadata structure we can determine as object as large or small.

When do you free a page allocated for objects in buckets

We again access the metadata using *getbase()* function. From the Metadata structure we check `free_bytes_available`, if `free_bytes_available==4080` then we traverse the list and remove all the Nodes residing on this page from the linked list.

After doing the above process we free the page.

Find the page metadata of the input object during myfree

We find the base address of the page in which the pointer resides using getbase function as defined below. In this function we find a number with first least significant 12 bits == 1. That number is 4095. Then we find a number with first 12 least significant bits == 0. We can do so by using a bit-wise not, '~'. Taking the bit-and with the address of the Node gives us the base of the node.

```
void *getbase(void *b){
     long max=PAGE_SIZE-1;
     max=~max;
     b=(void*)((long)b&max);
     return b;
}
```

By typecasting this address into (Metadata*) we can access the metadata of the page.

Code corresponding to the removal of all objects on the page from the bucket (list), when a page is freed

```
base->free_bytes_available+= base->allocation_size;
      if(base->free_bytes_available==4080){
            Node *current = buckets[bucket].current;
            while (current!=NULL)
            {
                  if (getbase((void*)current)==(void*)base)
                  {
                        if (current->prev!=NULL && current->next!=NULL){
                              current->prev->next=current->next;
                              current->next->prev=current->prev;
                        }else if(current->prev==NULL && current->next!=NULL){
                              current->next->prev=current->prev;
```

```
                }else if(current->prev!=NULL && current->next==NULL){
                      current->prev->next=current->next;
                }else{
                      current=NULL;
                }
          }
          if(current!=NULL){
                current=current->next;
                buckets[bucket].current=NULL;
          }
    }
    if (buckets[bucket].current!=NULL)
    {
          while(getbase((void*)buckets[bucket].current)==(void*)base){
                buckets[bucket].current=buckets[bucket].current->next;
                if (buckets[bucket].current==NULL)
                            break;
          }
    }
    free_ram((void*)base,PAGE_SIZE);
    return;
}
```

Output of the“make test”

Running in gdb debugger.

Command: gdb --args **make test**

```
      (gdb) run
Starting program: /usr/bin/make test
[Detaching after vfork from child process 12364]
[Detaching after vfork from child process 12367]
[Detaching after vfork from child process 12374]
[Detaching after vfork from child process 12377]
```

```
[Detaching after vfork from child process 12387]
num replacements:1000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:03.36
        Maximum resident set size (kbytes): 356052
        Minor (reclaiming a frame) page faults: 839839
num replacements:2000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:06.79
        Maximum resident set size (kbytes): 381684
        Minor (reclaiming a frame) page faults: 1616949
num replacements:3000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.52
        Maximum resident set size (kbytes): 403160
        Minor (reclaiming a frame) page faults: 2394611
num replacements:4000000
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:13.23
        Maximum resident set size (kbytes): 425376
        Minor (reclaiming a frame) page faults: 3172656
[Detaching after vfork from child process 12389]
[Inferior 1 (process 12360) exited normally]
```

Appendix

```c
void *getbase(void *b){
    long max=PAGE_SIZE-1;
    max=~max;
    b=(void*)((long)b&max);
    return b;
}
```

This function turns the last 12 bits of an address to 0. In this function we find a number with first least significant 12 bits == 1. That number is 4095. Then we find a number with first 12 least significant bits == 0. We can do so by using a bit-wise not, ‘~’. Taking the bit-and with the address of the Node gives us the base of the node.