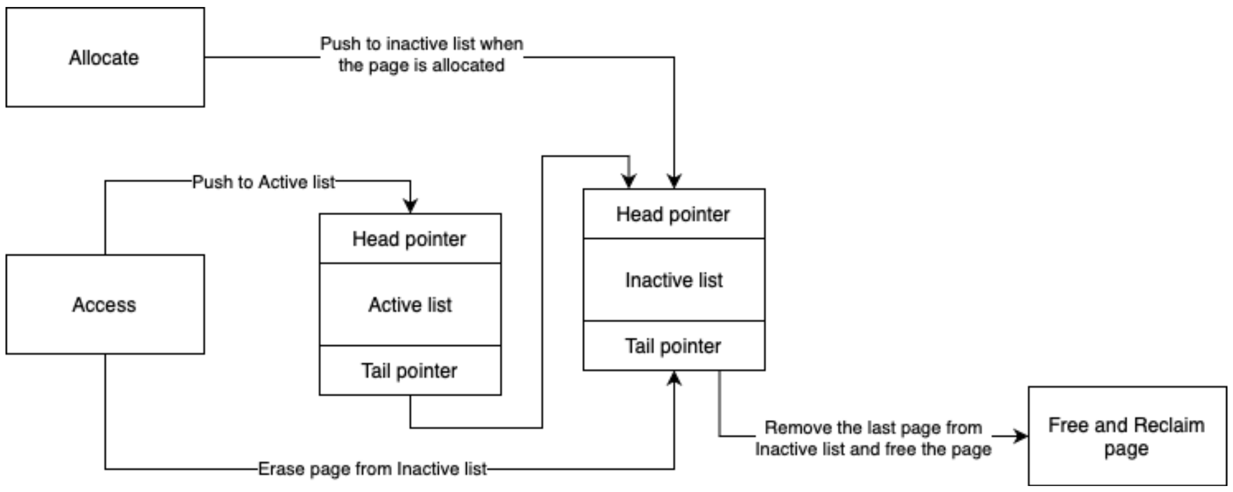
Assignment 4

Rohan Juneja (A0228349W)

Q1.


Allocate
Push to inactive list when the page is allocated
Push to Active list
Access
Head pointer
Active list
Tail pointer
Head pointer
Inactive list
Tail pointer
Remove the last page from Inactive list and free the page
Free and Reclaim page
Erase page from Inactive list

Buddy algorithm:

The buddy algorithm is initialized with an array which maintain 10 buckets, from order 0 to 9. Initially, one single chunk of 512 (2^9) pages are initialized at order 9.

- Allocate:
  - As soon as the allocate function is called, it calculates the ceil of log2 of the number of pages required to be allocated. If it finds the same order, the chunk of pages is sent. And the remaining set of unallocated pages are pushed into buckets of those respective orders, after dividing them into powers of 2.
  - Example, if 10 pages are required to be allocated. Chunk of pages in buckets with order 4 (i.e., 2^4 = 16 pages) are sent. Since only 10 pages are required, the remaining 6 pages are divided into powers of 2 (i.e., 2^2 = 4 and 2^1 = 2 pages) and pushed into buckets with order 2 and 1.
  - Though, if we don't have pages in that respective order (i.e., 4 in the example mentioned above). Pages from the upper order are split recursively till it reaches the specific order, and then gets allocated.
  - Also, if there are no pages in the buckets with upper order. The initial request of 10 is split recursively in orders of 2 and send requests to buckets of lower order, till it gets allocated.
  - Example, if there are no pages in buckets of upper order. The request of 10 is split into 2^3 = 8 and 2^1 = 2. Further if even 8 pages can't be allocated, the request is divided into 2^2 = 4, 2^2 = 4 and 2^1 = 2 pages. But even after recursively splitting it to 1, if no pages are allocated, the code throws an error. Though, such a case doesn't arrive in the example input file.
  - Also, as soon as the pages are allocated. They are pushed into the inactive list, so that the non-accessed pages also get a chance to get reclaimed.

- Free:
    - Initially, it checks if the page is already free, the algorithm simply returns.
    - The free algorithm iterates through both active and inactive list, and frees the page.
    - Then the page is pushed to bucket with order 0, which represents size of 2^0 = 1 page.
    - And the reclaim function is called, as explained below.

- Reclaim:
    - The reclaim algorithm gets the index of the buddy of the page (or set of pages) with order n. It then checks if the buddy is free, and if the order of buddy is equal to the given page.
    - If it is equal, both the buddy and page (or set of pages of order n) are removed from the buckets of order n. Then the size of the buddy are incremented by 1 to n+1. And is pushed to the buckets of order n+1.
    - And then the same reclaim algorithm is applied to this set of pages of order n+1.

LRU list:
Two queues, active and inactive are maintained with 250 pages as per the assignment. The aim of active list is to maintain the working set of all processes. And inactive list to contain reclaim candidates.

- Access:
    - Initial step is to check if the accessed page has already been freed, it gets allocated again. And then the algorithm checks for second condition, as explained below.
    - If the pages are neither in inactive nor active list, they are pushed into the inactive list. Then the algorithm checks the size of list, if it's more than 250, it frees the page in FIFO fashion.
    - If the pages are already present in the inactive list, they are removed from it and are pushed to the active list. Then similar to inactive list, it checks the size and remove the last page to inactive list.

- Eviction:
    - If the size of the inactive list grows more than 250, it simply calls the free function, explained in the buddy allocator above, on the last page of the inactive list.

Section B
Q1.
Address Calculation for Page$_0$
Level1: Address[5:0] = 4_{10} (As this is 4th index of the array)
Level2: Address[11:6] = 0_{10} (Similarly, this is the 0th index of the array at second level)
Level3: Address[17:12] = 0_{10} (This is also 0th index of the array at the top third level)
Address for Page$_0$ = 4_{10}

Address Calculation for Page$_1$
Level1: Address[5:0] = 1_{10}
Level2: Address[11:6] = 63_{10}
Level3: Address[17:12] = 2_{10}
Address for Page$_1$ = $12{,}225_{10}$

Address Calculation for Page$_2$
Level1: Address[5:0] = 1_{10}
Level2: Address[11:6] = 62_{10}
Level3: Address[17:12] = 63_{10}
Address for Page$_1$ = $262{,}017_{10}$

Q2.
Spinlock using CAS function.

```
void spinlock_init(int* lock) {
        *lock = 0;
}
// As out of order processors might execute load and store instructions in a different order,
// memory fence makes sure that code is executed as intended.

void spinlock_lock(int* lock) {
        while(compare_and_swap(lock, 0, 1) != 0) {}          // 0 means the thread acquired
                                                             // the lock
        __asm__ ("mfence");
}

void spinlock_unlock(int* lock) {
        __asm__ ("mfence");
        *lock = 0;
}
```