```
#include <stdio.h>
#include <stdlib.h>
#include "vm.h"
#include "API.h"
#include "list.h"
struct Node * PFN_queue = NULL;
int p = 0;
int fifo()
{
            int PFN;
            // Head of PFN_queue is used
            PFN = PFN_queue->data;
            // Remove head of PFN queue and rotates PFN to tail
            PFN_queue = list_remove_head(PFN_queue);
            PFN_queue = list_insert_tail(PFN_queue, PFN);

            return PFN;
}

int lru()
{
            int PFN;
            // Head of PFN_queue is used
            PFN = PFN_queue->data;
            // Remove head of PFN queue and rotates PFN to tail
            PFN_queue = list_remove_head(PFN_queue);
            PFN_queue = list_insert_tail(PFN_queue, PFN);

            return PFN;
}

int clock()
{
      int PFN = p % MAX_PFN;

      while (true) {

            if (clockArray[PFN] == 0) {
                  p++;
                  return PFN;
            }

            else if (clockArray[PFN] == 1)
            {
                  clockArray[PFN] = 0;
                  p++;
            }

      }
}

/*=========================================================================*/

int find_replacement()
{
            int PFN;
            if(replacementPolicy == ZERO)  PFN = 0;
            else if(replacementPolicy == FIFO)  PFN = fifo();
            else if(replacementPolicy == LRU) PFN = lru();
```

```
            else if(replacementPolicy == CLOCK) PFN = clock();

            return PFN;
}

int pagefault_handler(int pid, int VPN, char reqType)
{
            int PFN;
            PFN = get_freeframe();
            if (PFN >= 0) {
                  PFN_queue = list_insert_tail(PFN_queue, PFN); // Adds PFN to
doubly linked list
            }
            else if(PFN < 0) {
                        PFN = find_replacement();
                        /* ---- */
        IPTE victim = read_IPTE(PFN);
        if(read_PTE(victim.pid, victim.VPN).dirty){
          swap_out(victim.pid, victim. VPN, PFN);
        }
        PTE victim_pte;
        victim_pte.valid = false;
        write_PTE(victim.pid, victim.VPN, victim_pte);
            }
            // New page table entry for logical memory being moved, sets valid to
true and defines PFN
            PTE new_pte;
            new_pte.valid = true;
            new_pte.PFN = PFN;
            new_pte.dirty = false;

            IPTE new_ipte;
            new_ipte.pid = pid;
            new_ipte.VPN = VPN;

            // If it's a write operation then the Page Table Entry is dirty
            if (reqType == 'W') {
                  new_pte.dirty = true;
            }

            write_IPTE(PFN, new_ipte);
            write_PTE(pid, VPN, new_pte);
            swap_in(pid, VPN, PFN);

   return PFN;
}

int get_PFN(int pid, int VPN, char reqType)
{
            /* Read page table entry for (pid, VPN) */
            PTE pte = read_PTE(pid, VPN);

            /* if PTE is valid, it is a page hit. Return physical frame number
(PFN) */
            if(pte.valid) {
            /* Mark the page dirty, if it is a write request */
                        if(reqType == 'W') {
                                    pte.dirty = true;
                                    write_PTE(pid, VPN, pte);
```

```
                        }
                        if (replacementPolicy == LRU) {
                    // Removes PFN from queue then places it at tail since it's
interacted with
                                PFN_queue = list_remove(PFN_queue, pte.PFN);
                                PFN_queue = list_insert_tail(PFN_queue,
pte.PFN);
                        }
                        else if (replacementPolicy == CLOCK) {
                                if (clockArray[pte.PFN] == 0) {
                                      clockArray[pte.PFN]++;
                                }
                        }
                        return pte.PFN;
            }
            /* PageFault, if the PTE is invalid. Return -1 */
            return -1;
}

int MMU(int pid, int VPN, char reqType, bool *hit)
{
            int PFN;


            /* calculate VPN and offset
            VPN = ...
            offset = ...
            */

            // read page table to get Physical Frame Number (PFN)
            PFN = get_PFN(pid, VPN, reqType);
            if(PFN >= 0) { // page hit
                        stats.hitCount++;
                        *hit = true;
        return PFN;
            }
            stats.missCount++;
            *hit = false;
            PFN = pagefault_handler(pid, VPN, reqType);

            return PFN;
}
```