PROJECT 3

Github repository : https://github.com/moktayoglu/comp304-p3

Part 1

In this part of the project virtual memory and physical memory are taken with the same size as instructed.

While defining the PAGE MASK and OFFSET MASK, among 20 bits first 10 bits are allocated to PAGE MASK and last 10 bits are allocated to OFFSET MASK as below:

#define PAGE_MASK 0b11111111110000000000
#define OFFSET_MASK 0b00000000001111111111

Once PAGE MASK and OFFSET MASK are defined, page offsets and logical page numbers are calculated by using the operators of bit-masking and bit-shifting. The bitwise and operator silences the irrelevant bits, and for page number we right-shift by offset bits to get the last 10 bits.

int offset = (logical_address & OFFSET_MASK);
int logical_page = (logical_address & PAGE_MASK) >> OFFSET_BITS ;

- Searching in and Adding to TLB

In search_tlb method, we used the tlbentry struct for getting the current entry of tlb and returning the physical address of it accordingly. If the entry that we are looking for does not exist, then we returned -1 as instructed.

```
/* Returns the physical address from TLB or -1 if not present. */
int search_tlb(unsigned char logical_page) {
    /* TODO */
    struct tlbentry curr_entry;
    for(int i = 0; i<TLB_SIZE; i++){
      curr_entry = tlb[i];

      if(curr_entry.logical == logical_page){
        return curr_entry.physical;
      }

    }
    return -1;

}
```

In add_tlb method, we assigned logical and physical mappings to the TLB using the current index, and incremented the current index after the mapping. Since we are implementing FIFO on TLB of size 16, tlbindex is corrected using tlbindex % TLB_SIZE.

```
/* Adds the specified mapping to the TLB, replacing the oldest mapping (FIFO replacement). */
void add_to_tlb(unsigned char logical, unsigned char physical) {
    /* TODO */
    tlb[tlbindex].logical = logical;
    tlb[tlbindex].physical = physical;

    tlbindex += 1;

    //For circular replacement of first come, first out
    tlbindex = tlbindex % TLB_SIZE;
}
```

- FIFO replacement

We know that when physical_page is equal to -1, the physical memory has no match with a virtual memory. After incrementing the page faults, physical page is set to free page to select a new space for allocating that page. In page table that logical page is mapped to physical page and memcpy is used for copying from disk into memory, with equivalent page and frame range.

```
  // Page fault
  if (physical_page == -1)
  {

    page_faults++;

    physical_page = free_page;  //select new space to fill
    free_page++;
    free_page = free_page % PAGES;

    pagetable[logical_page] = physical_page;

    //copy backing into memory, with equivalent page and frame range
    memcpy(main_memory + physical_page * PAGE_SIZE, backing +  logical_page * PAGE_SIZE,
_SIZE);

  }
```

```
parallels@parallels-Parallels-Virtual-Platform:~/Downloads/project3/comp304-p3/comp304-p3$ gcc part1.c -o part1
parallels@parallels-Parallels-Virtual-Platform:~/Downloads/project3/comp304-p3/comp304-p3$ ./part1 BACKING_STORE.bin addresses.txt
Virtual address: 28928 Physical address: 256 Value: 0
Virtual address: 30711 Physical address: 2039 Value: -3
Virtual address: 8800 Physical address: 2656 Value: 0
Virtual address: 52335 Physical address: 3183 Value: 27
Virtual address: 38775 Physical address: 4983 Value: -35
Virtual address: 52704 Physical address: 3552 Value: 0
Virtual address: 24380 Physical address: 5948 Value: 0
Virtual address: 19602 Physical address: 6290 Value: 19
Virtual address: 57998 Physical address: 7822 Value: 56
Virtual address: 2919 Physical address: 9063 Value: -39
Virtual address: 8362 Physical address: 2218 Value: 8
Virtual address: 17884 Physical address: 9692 Value: 0
Virtual address: 45737 Physical address: 10921 Value: 0
Virtual address: 47894 Physical address: 12054 Value: 46
Virtual address: 59667 Physical address: 12563 Value: 68
Virtual address: 10385 Physical address: 13457 Value: 0
Virtual address: 52782 Physical address: 3630 Value: 51
Virtual address: 64416 Physical address: 15264 Value: 0
Virtual address: 40946 Physical address: 16370 Value: 39
Virtual address: 16778 Physical address: 16778 Value: 16
Virtual address: 27159 Physical address: 17943 Value: -123
```

```
Number of Translated Addresses = 1000
Page Faults = 421
Page Fault Rate = 0.421
TLB Hits = 164
TLB Hit Rate = 0.164
```

Part 2

In this part, we had physical memory size smaller than virtual memory. Thus the needed to have page-replacement policies implemented.

- **FIFO**
  This implementation is almost the same as the previous FIFO implementation, here first the free pages are filled, and once all are full the FIFO is done. Again by taking the modulo the increment circles thus always keeps track of the latest added frame. After, assigning the next available FIFO slot, the code checks if this was an already filed entry in the pagee table, if so deletes it and increments the FIFO frame index for the upcoming iterations.

```
if (isLRU==0){//FIFO
        if(filled_page < FRAMES) {
                physical_page = filled_page;
                filled_page++;

        }else{
        physical_page = curr_fifo_frame;


        for(int i=0; i<PAGES; i++) {
        //remove from the current table if we chose a victim
          if(pagetable[i] == curr_fifo_frame) {
                pagetable[i] = -1;

          }
        }

        curr_fifo_frame++;
        curr_fifo_frame = curr_fifo_frame % FRAMES;

        }

}else{ //LRU
```

Output:

```
Virtual address: 667527 Physical address: 188295 Value: -31
Virtual address: 347755 Physical address: 189035 Value: -102
Number of Translated Addresses = 1000
Page Faults = 441
Page Fault Rate = 0.441
TLB Hits = 164
TLB Hit Rate = 0.164
```

- **LRU**

  Next task was implementing Least Recently Used algorithm. Here the new frames overwrites the frames that were not been accessed or entried for the longest time among the other frames. For this implementation we had an array named LRU_CLOCK_COUNTER with the size of number of frames (256). Thus, each of the indices were keeping track of the latest time the frame had an accession/entry. This time was kept by keeping a clock variable that increments in each iteration.

  Again first all the free slots are filled, than LRU runs. Here since we didnt wanted import math module for infinity, we picked a very large number for min, and 0 for initial lru picked frame. Then we check for the minimum index (thus frame no), which tells that an activity on it happened much long ago. We kee track of this frame, than remove this victim from the page table so that it cannot be accessed in the next iteration as we know will insert our new frame. Finally since this is an entry activity we update the clock counter for this newly added frame as well.

```
}else{ //LRU

        //first we fill all available slots
        if(filled_page < FRAMES) {
                physical_page = filled_page;
                filled_page++;


        }else{

           int min = 999999999;
           int lru_frame = 0;
           for (int i=0; i<FRAMES; i++){
                 if (LRU_CLOCK_COUNTER[i] < min){
                        min = LRU_CLOCK_COUNTER[i];
                        lru_frame = i;
                }
            }

            for(int i=0; i<PAGES; i++) {
            //remove from the current table if we chose a victim
                if(pagetable[i] == lru_frame) {
                        pagetable[i] = -1;
                 }
             }
             physical_page = lru_frame;

        }
        LRU_CLOCK_COUNTER[physical_page] = clock;

}
```

Output:

```
Virtual address: 28928 Physical address: 256 Value: 0
Virtual address: 30711 Physical address: 2039 Value: -3
Virtual address: 8800 Physical address: 2656 Value: 0
Virtual address: 52335 Physical address: 3183 Value: 27
Virtual address: 38775 Physical address: 4983 Value: -35
Virtual address: 52704 Physical address: 3552 Value: 0
Virtual address: 24380 Physical address: 5948 Value: 0
Virtual address: 19602 Physical address: 6290 Value: 19
Virtual address: 57998 Physical address: 7822 Value: 56
Virtual address: 2919 Physical address: 9063 Value: -39
Virtual address: 8362 Physical address: 2218 Value: 8
Virtual address: 17884 Physical address: 9692 Value: 0
Virtual address: 45737 Physical address: 10921 Value: 0
```

```
Virtual address: 65714 Physical address: 186546 Value: 0
Virtual address: 667527 Physical address: 190343 Value: -31
Virtual address: 347755 Physical address: 191083 Value: -102
Number of Translated Addresses = 1000
Page Faults = 443
Page Fault Rate = 0.443
TLB Hits = 164
TLB Hit Rate = 0.164
parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Parallels
```