COMP304 PROJECT3 REPORT

The functions that we have implemented and what do they aim?

The outputs are given at the end of the report

PART-I

```
62 /* Adds the specified mapping to the TLB, replacing the oldest mapping
63 (FIFO replacement). */
64 void add_to_tlb(unsigned char logical, unsigned char physical) {
65     /* TODO */
66  int counter = tlbindex % TLB_SIZE;
67  tlb[counter].logical == logical;
68  tlb[counter].physical == physical;
69  tlbindex++;
70
71 }
```

This function is used to populate the translation lookaside buffer with a mapping from a logical page number to a physical page number .The function accepts two parameters: a logical page number and a physical page number . These values are stored in the TLB using FIFO replacement. When the TLB becomes full, the oldest mapping is replaced with the new mapping. To do this, the function keeps a counter (tlbindex) that is incremented whenever a new mapping is added to the TLB. The counter determines the index of the TLB entry that has to be changed. The counter is also used to build a circular buffer, such that when the TLB is full, the next item is inserted at the beginning.

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

This function looks for a mapping from a logical page number to a physical page number in the translation lookaside buffer. It accepts a single input, the logical page number, and returns the physical page number corresponding to it if it is located in the TLB. The function returns -1 if the mapping cannot be found. The function searches the TLB by iterating over all of the entries and determining if the current entry's logical page number matches the input. If there is a match, the method returns the physical page number. If there is no match, the method returns -1.

```
   /* TODO
   / Calculate the page offset and logical page number from
ogical_address */
   int offset = (logical_address) & (OFFSET_MASK);
   int logical_page = (logical_address >> OFFSET_BITS) & (PAGE_MASK);
   ///////
```

Offset is extracted from

logical_address using a bitmask with 1s in the least significant bits.

Logical_page is extracted from logical_address using a bit shift and a bitmask with 1s in the most significant bits.

```
123       // Page fault
124       if (physical_page == -1) {
125         /* TODO */
126         page_faults++;
127         memcpy(main_memory + free_page*PAGE_SIZE, backing + logical_page *
    PAGE_SIZE, PAGE_SIZE);
128         physical_page = free_page;
129         pagetable[logical_page] = physical_page;
130
131       }
132       add_to_tlb(logical_page, physical_page);
```

It determines whether or not the physical page number matching to the provided logical page number is present in the TLB or page table. If this is the case, the code increments a page faults counter and moves the data from a backup store into main memory at the next available free page. The physical page number is allocated to the logical page's physical page number in the page table, and the pair (logical page number, physical page number) is stored in the TLB.

PART-II

-Least Recently Used:The lru_element() method implements the Least Recently Used (LRU) policy using a page replacement mechanism. This indicates that the system chooses the least recently viewed page to be replaced with a new one. The function accepts two parameters:

logical page: The logical page number of the currently visited page.

pageFault: The total number of page faults encountered thus far.

It begins by determining if the number of page faults is fewer than or equal to the number of page frames. If this is the case, there is still room in the page table for the new page, and the function may simply add it without changing any current pages. The function adds the new mapping from logical page to physical page to the page table and returns the physical page number of the new page. If the number of page faults exceeds the number of page frames, there is no more room in the page table and a page replacement is required and scans through the page table and finds the page with the oldest last access time. This is done using the last_access_time array, which keeps track of the time that each page was last accessed. The function then empties the physical page corresponding to the least recently used page.

-FIFO:The loop searches for the virtual page that is currently mapped to the physical page specified by the free_page argument. When this page is found, the function sets the corresponding entry in the page table to -1, indicating that the virtual page is no longer mapped to any physical page. The loop then breaks out of the loop, because the page has been found and the replacement has been done.

-MAIN:As we have two different types of algorithms FIFO&LRU , we passed the string value of argument and convert the fourth element of it to decide users' choice of algorithm.Then conditions take the value and implement the corresponding policy of replacement . Once we have executed our if conditions, we copy a page of data from backing store into main memory and keep updating a page table to reflect the mapping between the logical page and the physical page.

In this case, it is copying PAGE_SIZE bytes of data from the backing store, starting at the location backing + logical_page * PAGE_SIZE, to the main memory, starting at the location main_memory + free_page * PAGE_SIZE. The free_page variable is probably an integer representing an available page of main memory where the data can be stored.After the data has been copied, the physical page number (free_page) is stored in the page table for the logical page (logical_page). This updates the page table to reflect the mapping between the logical page and the physical page in main memory.

```
efe@efe:~/Desktop/lewisHam$ nano part1.c
efe@efe:~/Desktop/lewisHam$ gcc part1.c -o part1
efe@efe:~/Desktop/lewisHam$ ./part1 backingstore.bin addresses.txt
Virtual address: 28928 Physical address: 256 Value: 0
Virtual address: 30711 Physical address: 1015 Value: -3
Virtual address: 8800 Physical address: 608 Value: 0
```

```
Number of Translated Addresses = 1000
Page Faults = 436
Page Fault Rate = 0.436
TLB Hits = 7
TLB Hit Rate = 0.007
```

```
efe@efe:~/Desktop/lewisHam$ ./efe backingstore.bin addresses.txt -p 1
choice:1
Virtual address: 28928 Physical address: 256 Value: 0
Virtual address: 30711 Physical address: 1015 Value: -3
```

```
Number of Translated Addresses = 1000
Page Faults = 422
Checker = 224
Page Fault Rate = 0.422
TLB Hits = 164
```

```
efe@efe:~/Desktop/lewisHam$ ./efe backingstore.bin addresses.txt -p 2
choice:2
Virtual address: 28928 Physical address: 256 Value: 0
Virtual address: 30711 Physical address: 1015 Value: -3
Virtual address: 8800 Physical address: 608 Value: 0
```

```
Number of Translated Addresses = 1000
Page Faults = 421
Checker = 225
Page Fault Rate = 0.421
TLB Hits = 164
TLB Hit Rate = 0.164
```