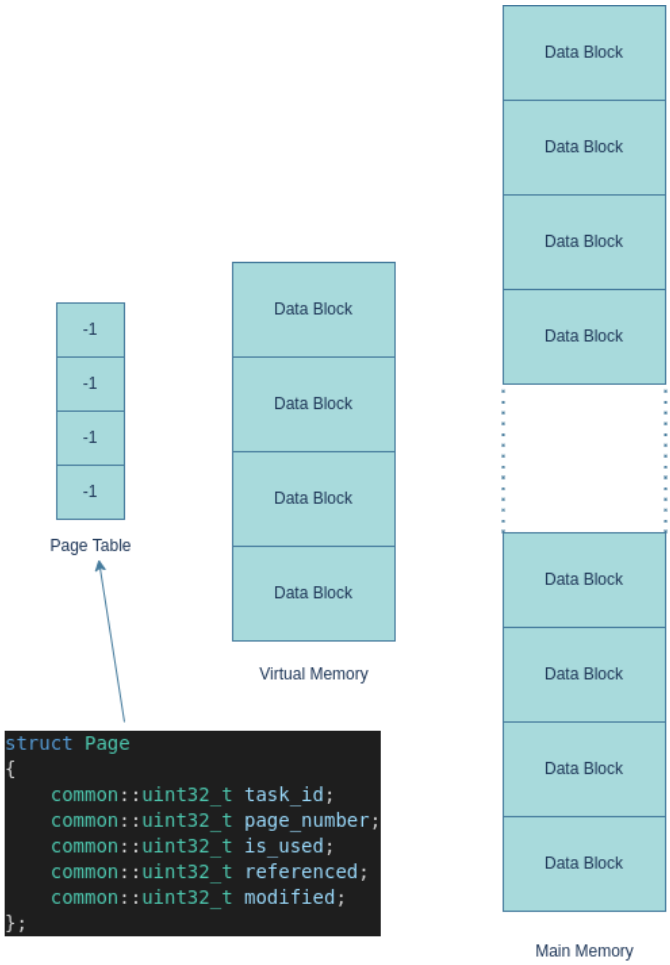
Gebze Technical University - Computer Engineering Department

CSE 312 – 2022 Operating System Course

Muhammed Bedir ULUCAY 1901042697

Problem:

Desing a memory management algorithm with paging and virtual memory and main memory. I implemented FIFO and the Second Change FIFO algorithm for replacing pages between virtual memory and main memory. We are creating need space for virtual memory, main memory, and page table.



Initialization of the memory management structure

Solution:

```
uint32_t data_size = 128;

uint32_t page_size = 16 * sizeof(uint32_t);
uint32_t vm_page_count = 4;
uint32_t mem_page_count = 16;

uint32_t vm_page_size = vm_page_count * page_size;
uint32_t mem_page_size = mem_page_count * page_size;

uint32_t page_table[vm_page_count];
uint32_t vm[vm_page_size];
uint32_t mem[mem_page_size];
```

Initial values of structure space

Then store the data in the main memory.

```
uint32_t data[data_size] = {
51, 93, 98, 7, 28, 94, 38, 25, 29, 29, 94, 29, 9, 42, 36, 19,
57, 75, 31, 90, 57, 89, 25, 43, 48, 61, 29, 16, 79, 27, 28, 30,
21, 26, 90, 1, 20, 28, 26, 49, 9, 72, 30, 18, 67, 66, 90, 24,
93, 73, 15, 50, 62, 92, 46, 63, 6, 75, 79, 85, 55, 7, 68, 76,
85, 58, 77, 5, 38, 56, 6, 47, 80, 36, 66, 47, 54, 8, 72, 0,
81, 39, 2, 96, 31, 48, 59, 37, 76, 90, 75, 31, 97, 43, 59, 82,
1, 88, 39, 91, 44, 97, 38, 25, 33, 56, 24, 39, 64, 48, 39, 98,
87, 94, 94, 71, 42, 5, 8, 18, 95, 35, 1, 44, 78, 60, 78, 31,
};

uint32_t data_page_count = sizeof(data) / page_size;

for(uint32_t i=0; i<data_size; ++i){
    mem[i] = data[i];
}
```

Then the algorithms start running as an example explain of Bubble sort with FIFO algorithm.

```
for(int i = 0; i < data_size; ++i){
    for(int j = 0; j < data_size - i; ++j){

        int page_number_left = j / page_size;
        int page_number_right = (j + 1) / page_size;
```

We are starting with two loops and find the page index of getting wanted values using division by page size.

Then we need to check this page values are in the page table. If they are in the page table that means we make a hit otherwise there is a miss.

```
int ret_l = in_page_table_fifo(vm_pages, vm_page_count, page_number_left);
int ret_r = in_page_table_fifo(vm_pages, vm_page_count, page_number_right);
* Uncommitted changes
```

Checks for both value and loop in the page table.

```
    You, 5 days ago • initial commit …
int in_page_table_fifo(Page vm_pages[], uint32_t len, uint32_t page_number){

    for(uint32_t i=0; i<len; i++){
        if(vm_pages[i].page_number == page_number)
            return i;
    }

    return -1;
}
```

If the page table has the value returns the index of the page. Otherwise return –1 to detect page fault and swap page. This is for the FIFO. In the second change FIFO, we check repeatedly until we find a referenced bit is equal to 0.

```
if(ret_l < 0){
    ret_l = page_number_left;
    vm_pages[next].page_number = page_number_left;
    vm_pages[next].is_used = 1;
    vm_pages[next].referenced = 1;      You, 1 second ago • Uncommitted changes
    vm_pages[next].modified = 1;
    next = (next + 1) % vm_page_count;

    for(int k=0; k<page_size; ++k){
        vm[page_number_left * page_size + k] = mem[page_number_left * page_size + k];
    }
    miss++;
}
```

Setting the page table values and getting these values in to the virtual memory. And increase the miss with 1. Same for the second value.

```
if(ret_l >= 0){
    vm_pages[ret_l].referenced = 1;
}

if(ret_r >= 0){
    vm_pages[ret_r].referenced = 1;
}
```

If there is no miss, we make a hit then we make the reference bit of the page value with 1.

```
if(vm[ret_l * page_size + j] > vm[ret_r * page_size + j + 1]){
    uint32_t tmp = vm[ret_l * page_size + j + 1];
    vm[ret_l * page_size + j] = vm[ret_r * page_size + j];
    vm[ret_r * page_size + j] = tmp;
}
```

Then we are making comparisons between values to swapping.

```
for(int k=0; k<page_size && vm_pages[ret_l].modified == 1; ++k){
    mem[page_number_left * page_size + k] = vm[ret_l * page_size + k];
}      You, 1 hour ago • last …

for(int k=0; k<page_size && vm_pages[ret_r].modified == 1; ++k){
    mem[page_number_right * page_size + k] = vm[ret_r * page_size + k];
}

vm_pages[ret_l].modified = 0;
vm_pages[ret_r].modified = 0;
```

Then we rewrite the main memory of those pages to not lose any data. I may do this after the inner loop did, but I want to be grantees that do not lose data.