Paging and Replacement

1. Introduction

In a computer operating system that uses paging for virtual memory management, **page replacement algorithms** decide which memory pages to page out, sometimes called swap out, or write to disk, when a page of memory needs to be allocated. Page replacement happens when a requested page is not in memory (page fault) and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.

2. Problem

It is required to simulate some of paging replacement algorithms. The required algorithms to be implemented are as follows:

- Optimal
- First In First Out (FIFO)
- Least Recently Used (LRU)
- Clock

3. Solution

3.1. Design

3.1.1. Reading the input

As specified in the problem, the input should contain cache size, selected replacement algorithm and the input stream (line separated). Reading the input would start by reading the cache size then allocating it, reading in the algorithm and finally getting the input stream until it's terminated by a -1.

3.1.2. Selecting the algorithm

The algorithm selection would occur right after reading the input by specifying an array of the available algorithms options and starting the selected algorithm.

3.1.3. Paging algorithms

Each algorithm has its own body but all algorithms share utility functions such as specifying whether the input already exists in cache, print output state, find next empty index … etc.

3.1.4. Printing the output

To print the output that is exactly the same as the given output would be possible by printing the constant parts in the main and printing the variable part inside each algorithm after each insertion.

3.2. Implementation

3.2.1. Reading the input

The `read_input()` function reads the input file, first thing it scans the cache size and stores it in the global variable `c_size` then initializes the cache with `init_cache()`, then it reads the algorithm and finally reads the input stream and sets the selected algorithm from its index in the array of `algorithms`.

```
/* GLOBAL VARIABLES */
int c_size, alg_option, s_size;
int *cache, *stream;
const char* algorithms[] =
{
    "OPTIMAL",
    "FIFO",
    "LRU",
    "CLOCK"
};
void init_cache(){
    cache = (int*) malloc(c_size * sizeof(int));
    for(int i = 0 ; i < c_size ; i++){
        cache[i] = -1;
    }
}
/* FILE IO */
void read_input()
{
    scanf("%d", &c_size);
    init_cache();

    char* alg;
    alg = (char*) malloc(10*sizeof(char));
    scanf("%s",alg);

    int i = 0;
    while(stream[i-1] != -1){
        scanf("%d", &stream[i++]);
    }
    s_size = i-1;
    for(int j = 0 ; j < 4 ; j++){
        if(strcmp(algorithms[j], alg) == 0){
            alg_option = j;
        }
    }
    if(alg_option == -1){
        puts("Please enter a valid algorithm!");
        exit(1);
    }
}
```

3.2.2. Selecting the algorithm

A very simple switch statement is used to select the desired algorithm from the `alg_option` variable.

```
switch(alg_option){
        case 0:
            optimal_replacement();
            break;
        case 1:
            fifo_replacement();
            break;
        case 2:
            lru_replacement();
            break;
        case 3:
            clock_replacement();
            break;
}
```

3.2.3. Paging algorithms

3.2.3.1. Optimal Paging Algorithm

The optimal paging algorithm depends on predicting the farthest incoming input page in the stream and ejecting it, to achieve this, I created the `ejection()` function that does this for me every time a page fault occurs. It scans the input stream and returns the index to replace.

```
/* OPTIMAL ALGORITHM */

int ejection(int index){
    int farthest = index, ejected = -1;

    for(int i = 0 ; i < c_size ; i++){
        int j;
        for(j = index ; j < s_size ; j++){
            if(stream[j] == cache[i]){
                if(j>farthest){
                    farthest = j;
                    ejected = i;
                }
                break;
            }
        }

        if(j == s_size){
            return i;
        }
    }
    return (ejected == -1) ? 0 : ejected;
}
```

The actual optimal replacement function:

```
void optimal_replacement(){
    int input;
    for(int i = 0 ; i < s_size ; i++){
        input = empty_index();
        fault_flag = 0;
        // if input == -1, no empty spaces in cache
        if(input == -1){
            if(!in_cache(stream[i])){
                faults++;
                fault_flag = 1;
                int replacement_index = ejection(i);
                cache[replacement_index] = stream[i];
            }
        }
        else{
            if(!in_cache(stream[i])){
                cache[input] = stream[i];
            }
        }
        print_state(stream[i]);
    }
}
```

3.2.3.2. FIFO Paging Algorithm

First-In-First-Out policy replaces the first inserted page, so, a pointer on the last inserted is saved to remove it whenever a page fault occurs.

```
/* FIFO ALGORITHM */
void fifo_replacement(){
    int input;
    int pointer = 0;
    for(int i = 0 ; i < s_size ; i++){
        input = empty_index();
        fault_flag = 0;
        // if input == -1, no empty spaces in cache
        if(input == -1){
            if(!in_cache(stream[i])){
                faults++;
                fault_flag = 1;
                cache[pointer] = stream[i];
                pointer = (pointer+1)%c_size;
            }
        }
        else{
            if(!in_cache(stream[i])){
                cache[input] = stream[i];
            }
        }
        print_state(stream[i]);}}
```

3.2.3.3. LRU Paging Algorithm

LRU or Least Recently Used algorithm replaces the least recently used page in the cache, which means we need to keep track of the usage history of each page, I chose to do that by the counter method i.e. I will store a value for each page inserted in usage array, when one insertion happens, the usage timer decreases by one and the least number is the least recently used page. The `LRU(int* used)` function returns the least recently used page by checking the array of usage and returning the index of the least value.

```
/* LRU ALGORITHM */

int LRU(int* used){
    int min = used[0];
    int min_index = 0;
    for(int i = 0 ; i < c_size ; i++){
        if(used[i] < min){
            min = used[i];
            min_index = i;
        }
    }
    return min_index;
}

void lru_replacement(){
    int input;
    int* used = (int*) malloc(c_size*sizeof(int));
    for(int i = 0 ; i < s_size ; i++){
        input = empty_index();
        fault_flag = 0;
        // if input == -1, no empty spaces in cache
        if(input == -1){
            if(!in_cache(stream[i])){
                faults++;
                fault_flag = 1;
                int replacement_index = LRU(used);
                cache[replacement_index] = stream[i];
                used[replacement_index] = INT_MAX;
            }
            else{
                for(int j = 0 ; j < c_size ; j++){
                    if(cache[j] == stream[i]){
                        used[j] = INT_MAX;
                        break;
                    }
                }
            }
        }
        else{
            if(!in_cache(stream[i])){
                cache[input] = stream[i];
                used[input] = INT_MAX;
            }
            else{
                for(int j = 0 ; j < c_size ; j++){
                    if(cache[j] == stream[i]){
                        used[j] = INT_MAX;
                        break;
                    }
                }
            }
        }
        for(int u = 0 ; u < c_size ; u++){
            used[u]--;
        }
        print_state(stream[i]);
    }
}
```

3.2.3.4. Clock Policy Algorithm

Clock policy algorithm replaces the least use value page in the cache, use_check() function does the clock policy by decreasing the use and iterating until it reaches a use of 0. It takes the use array and the current index to start iteration from there.

```
/* CLOCK ALGORITHM */
int use_check(int* use, int i){
    while(1){
        if(use[i] == 0)
            return i;
        else
            use[i]--;
        i = (i+1)%c_size;
    }
    return -1;
}
```

```c
void clock_replacement(){
    int input, pointer;
    int* use = (int*) malloc(c_size*sizeof(int));
    pointer = 0;

    for(int i = 0 ; i < s_size ; i++){
        input = empty_index();
        fault_flag = 0;
// if input == -1, no empty spaces in cache
        if(input == -1){
            if(!in_cache(stream[i])){
                faults++;
                fault_flag = 1;
                int replacement_index = use_check(use, pointer);
                cache[replacement_index] = stream[i];
                use[replacement_index] = 1;
                pointer = (pointer+1)%c_size;
            }
            else{
                for(int j = 0 ; j < c_size ; j++){
                    if(cache[j] == stream[i]){
                        use[j] = 1;
                        break;
                    }
                }
            }
        }
        else{
            if(!in_cache(stream[i])){
                cache[pointer] = stream[i];
                use[pointer] = 1;
                pointer = (pointer+1)%c_size;
            }
            else{
                for(int j = 0 ; j < c_size ; j++){
                    if(cache[j] == stream[i]){
                        use[j] = 1;
                        break;
                    }
                }
            }
        }
        print_state(stream[i]);
    }
}
```

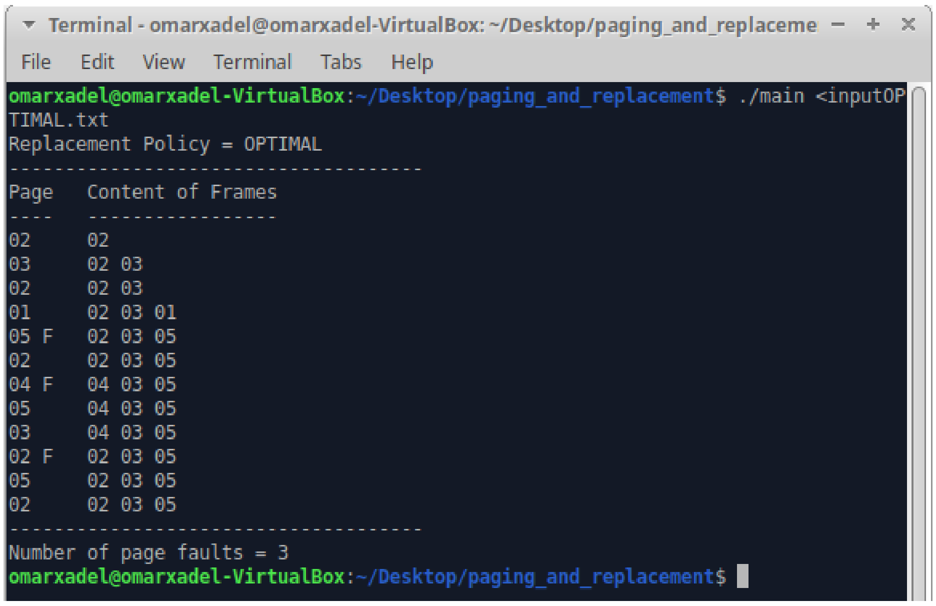
3.2.4. Printing the output

```
void print_state(int input){

    if(fault_flag){
        if(input/10 > 0)
            printf("%d F    ", input);
        else
            printf("0%d F    ", input);
    }
    else{
        if(input/10 > 0)
            printf("%d      ", input);
        else
            printf("0%d      ", input);
    }
    for(int i = 0 ; i < c_size ; i++){
        if(cache[i]!=-1){
            if(input/10 > 0)
                printf("%d ", cache[i]);
            else
                printf("0%d ", cache[i]);
        }
    }
    printf("\n");
}
int main()
{
    init();
    read_input();
    printf("Replacement Policy = %s\n-------------------------------------\nPage   Content
 of Frames\n----   -----------------\n", algorithms[alg_option]);

    switch(alg_option){
        case 0:
            optimal_replacement();
            break;
        case 1:
            fifo_replacement();
            break;
        case 2:
            lru_replacement();
            break;
        case 3:
            clock_replacement();
            break;
    }
    printf("-------------------------------------\nNumber of page faults = %d",faults);
    return 0;
}
```

1.1. Sample Runs

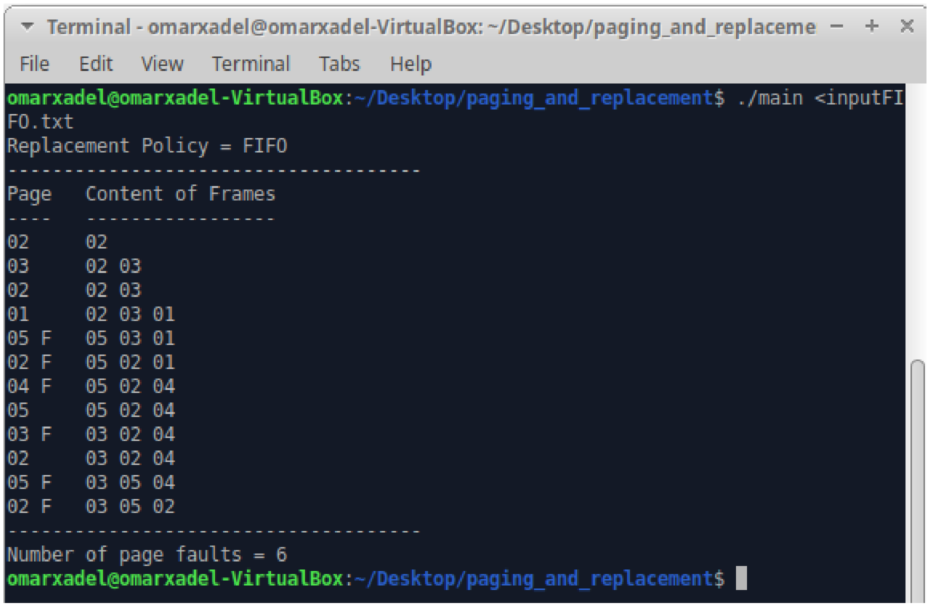
1.1.1. Runs

1.1.1.1. OPTIMAL


Terminal - omarxadel@omarxadel-VirtualBox: ~/Desktop/paging_and_replaceme
File Edit View Terminal Tabs Help

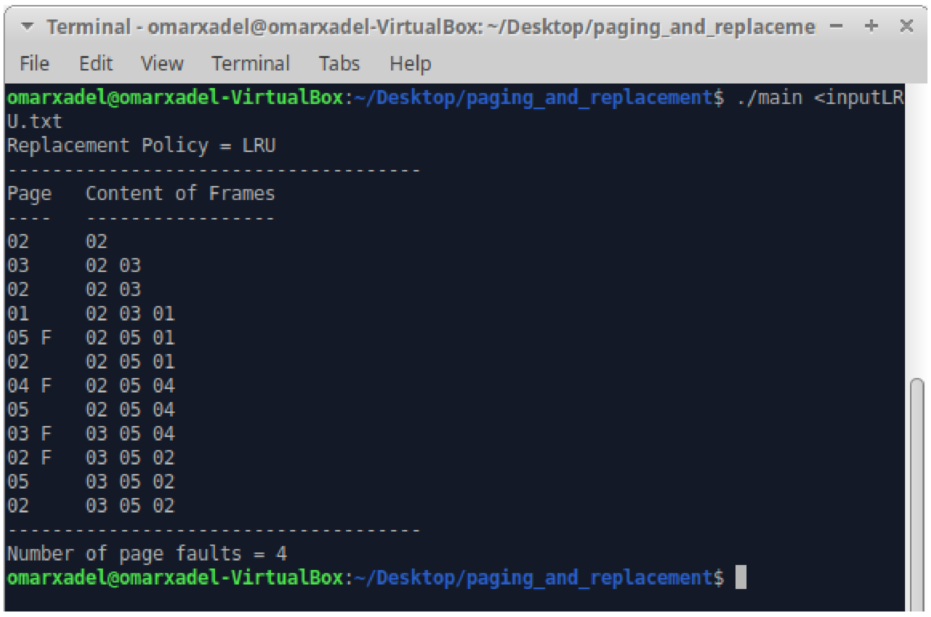
```
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ ./main <inputOP
TIMAL.txt
Replacement Policy = OPTIMAL
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   02 03 05
02     02 03 05
04 F   04 03 05
05     04 03 05
03     04 03 05
02 F   02 03 05
05     02 03 05
02     02 03 05
-------------------------------------
Number of page faults = 3
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$
```

1.1.1.2. FIFO


Terminal - omarxadel@omarxadel-VirtualBox: ~/Desktop/paging_and_replaceme
File Edit View Terminal Tabs Help

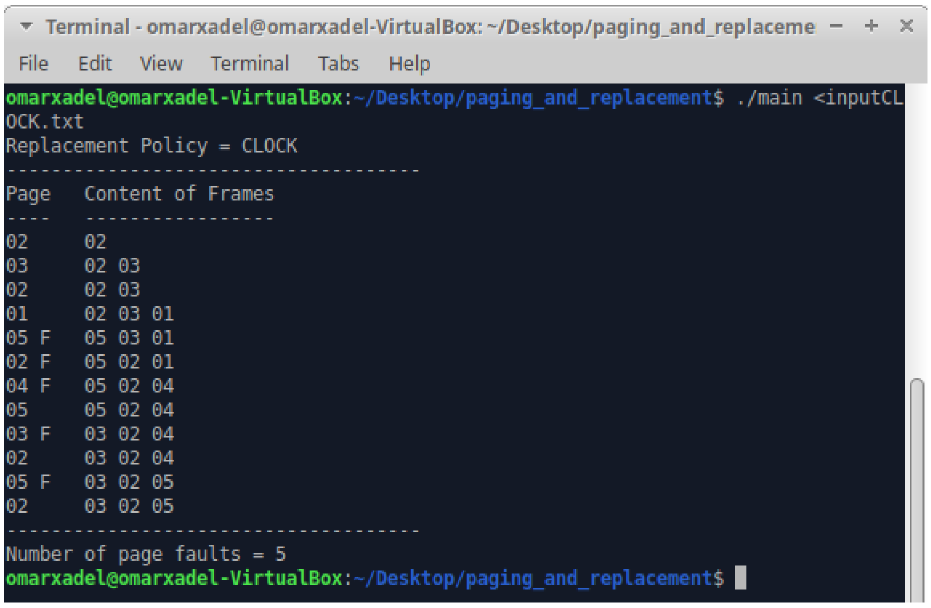
```
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ ./main <inputFI
FO.txt
Replacement Policy = FIFO
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   05 03 01
02 F   05 02 01
04 F   05 02 04
05     05 02 04
03 F   03 02 04
02     03 02 04
05 F   03 05 04
02 F   03 05 02
-------------------------------------
Number of page faults = 6
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$
```

1.1.1.3. LRU


Terminal - omarxadel@omarxadel-VirtualBox: ~/Desktop/paging_and_replaceme
File Edit View Terminal Tabs Help

```
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ ./main <inputLR
U.txt
Replacement Policy = LRU
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   02 05 01
02     02 05 01
04 F   02 05 04
05     02 05 04
03 F   03 05 04
02 F   03 05 02
05     03 05 02
02     03 05 02
-------------------------------------
Number of page faults = 4
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$
```

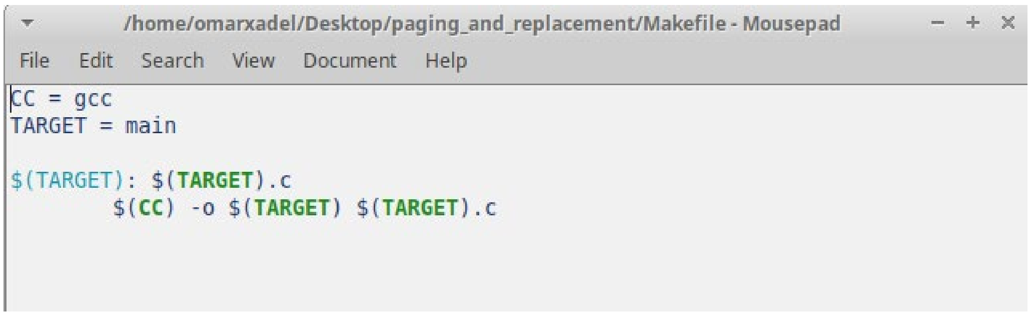
1.1.1.4. CLOCK


Terminal - omarxadel@omarxadel-VirtualBox: ~/Desktop/paging_and_replaceme
File Edit View Terminal Tabs Help

```
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ ./main <inputCL
OCK.txt
Replacement Policy = CLOCK
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   05 03 01
02 F   05 02 01
04 F   05 02 04
05     05 02 04
03 F   03 02 04
02     03 02 04
05 F   03 02 05
02     03 02 05
-------------------------------------
Number of page faults = 5
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$
```

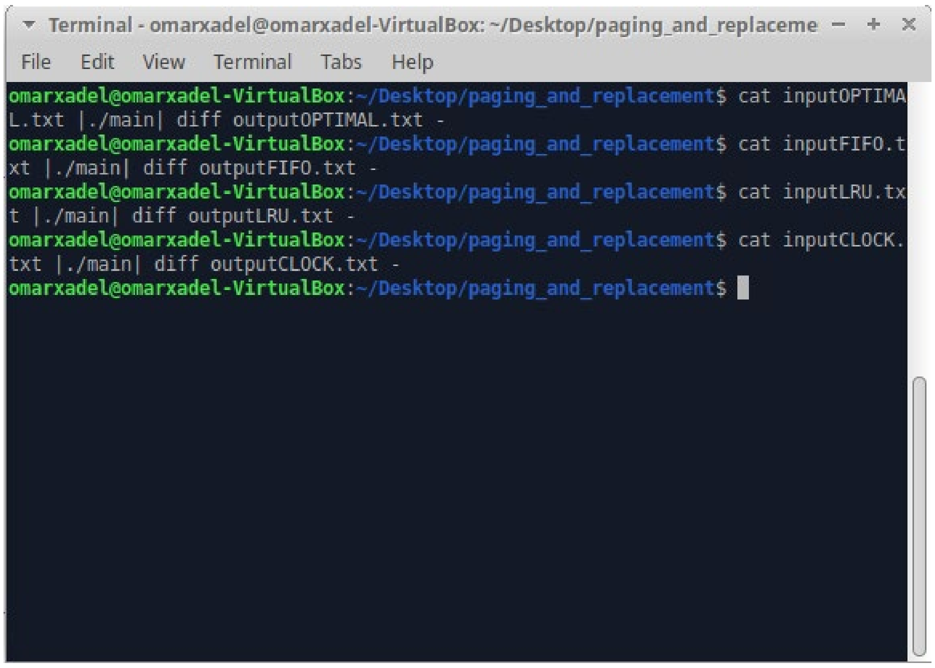
1.1.2. Makefile

1.1.2.1. Makefile File


/home/omarxadel/Desktop/paging_and_replacement/Makefile - Mousepad
File Edit Search View Document Help

```
CC = gcc
TARGET = main

$(TARGET): $(TARGET).c
	$(CC) -o $(TARGET) $(TARGET).c
```

1.1.2.2. File difference run


Terminal - omarxadel@omarxadel-VirtualBox: ~/Desktop/paging_and_replaceme
File Edit View Terminal Tabs Help

```
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ cat inputOPTIMA
L.txt |./main| diff outputOPTIMAL.txt -
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ cat inputFIFO.t
xt |./main| diff outputFIFO.txt -
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ cat inputLRU.tx
t |./main| diff outputLRU.txt -
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ cat inputCLOCK.
txt |./main| diff outputCLOCK.txt -
omarxadel@omarxadel-VirtualBox:~/Desktop/paging_and_replacement$ 
```