CS356 Project 8

Name : HaotianXue

Student ID : 518021910506

Email : xavihart@sjtu.edu.cn

Introduction

Purpose of this Project

This project encourage us to simulate the key operations of the virtual memory using paging technique, we should implement the TLB, page table and framed memory, also we are encouraged to use demanding paging and we should calculate the page fault rate and TLB hit rate.

Our Work

The source data include the `addresses.txt` and `BACKING_STORE.bin`, the checking data `correct.txt`. They are all from the source data offered by the book from https://github.com/greggagne/osc10e/tree/master/ch10. We implement the virtual memory simulator in C in Linux OS Ubuntu 18.04, we check our result by checking our output(including physical address and fetched data) with `corrent.txt` and we also give the hit rate and page fault rate as a result. We use LRU as the replacement strategy for both TLB and frame.

General Ideas

Basic Structures

First we define some structures in C including TLB struct, page table struct and memory struct. To do LRU, we set a `latest_used` to note down the latest using time of TLB or frame.

```
typedef struct TLB{
    int latest_used;
    int page_number;
    int frame_number;
}tlb_t;

typedef struct PAGE_TABLE{
    int valid;
    int frame_number;
}page_table_t;

typedef struct memory{
    int latest_used;
    char data[FRAME_SIZE];
}mem_t;
```

Important Constant Values

We have some important constant values for this work, and they are:

```
#define TLB_SIZE 16
#define FRAME_NUMBER 256
#define FRAME_SIZE 256
#define PAGE_TABLE_SIZE 256
```

The `FRAME_NUMER` will be set to 256 and 128 respectively, the later means the frame number is smaller than the page number.

Key Functions

Key functions include initiation function, LRU for TLB and LRU for frame:

```
void init();
void close_file();
void LRU_tlb(int, int);
void LRU_mem(int, int);
```

The `init()` function include the initialization of TLB, page table ,memory and basic values like timer and counter:

```
void init(){
    for(int i = 0;i < TLB_SIZE;++i){
        tlb[i].frame_number = -1;
        tlb[i].latest_used = -1;
        tlb[i].latest_used = 0;
    }
```

```
    for(int i = 0;i < PAGE_TABLE_SIZE;++i){
        page_table[i].frame_number = -1;
        page_table[i].valid = 0;
    }
    for(int i = 0;i < FRAME_NUMBER;++i){
        mem[i].latest_used = 0;
    }
    time = tlb_hit_number = page_fault_number = count = 0;
    printf("space initialized----\n");
}
```

LRU part is based on the value of `latest_used`, we choose the minimal number of latest using time and replace the corresponding part, and the implementation of them is alike:

```
void LRU_tlb(int page_id, int frame_id){
    int min_time=1e9, min_index=0;
    for(int i = 0;i < TLB_SIZE;++i){
        if(tlb[i].latest_used < min_time){
            min_time = tlb[i].latest_used;
            min_index = i;
        }
    }
    tlb[min_index].frame_number = frame_id;
    tlb[min_index].page_number = page_id;
    tlb[min_index].latest_used = time;
    return;
}

void LRU_mem(int frame_number, int page_number){
    mem[frame_number].latest_used = time;
    fseek(bin, page_number * FRAME_SIZE, SEEK_SET);
    fread(mem[frame_number].data, sizeof(char),
FRAME_SIZE, bin);
    return;
}
```

Also we generalize the process of closing files:

```
void close_file(){
    fclose(bin);
    fclose(addr);
    fclose(out);
    return;
}
```

Detailed Implementation

Each time we read from the `addresses.txt` and calculate the offset as well as the page number. Then we check if the TLB hits, TLB not hit but page table hit, page fault in in order. In each loop we should update the timer, counter, TLB hit time and page fault rate.

The `main` function of the program works like this:

```
int main(int argc, char*argv[]){
    assert(argc == 2);
    init();
    addr = fopen(argv[1], "r");
    out = fopen("out.txt", "w");
    bin = fopen("BACKING_STORE.bin", "rb");
    fscanf(addr, "%d", &address);
    while(!feof(addr)){
        time++;
        count++;
        int offset = address & 0x000000ff;
        int page_number = (address >> 8) & 0x000000ff;
        int in_tlb = 0;
        int page_fault = 1;
        int frame_id;
        for(int i = 0;i < TLB_SIZE;++i){
            if(page_number == tlb[i].page_number){
                tlb_hit_number ++;
                in_tlb = 1;
                page_fault = 0;
                frame_id = tlb[i].frame_number;
                mem[frame_id].latest_used = time;
                tlb[i].latest_used = time;
                LRU_tlb(page_number, frame_id);
                break;
            }
        }


        if(in_tlb == 0 && page_table[page_number].valid){
            page_fault = 0;
            frame_id =
page_table[page_number].frame_number;
            mem[frame_id].latest_used = time;
            LRU_tlb(page_number, frame_id);
        }


        if(page_fault){
            page_fault_number ++;
            page_table[page_number].valid = 1;
            //printf("%d\n", page_number);
            int min_time=1e9, min_index=0;
            for(int i = 0;i < FRAME_NUMBER;++i){
                if(mem[i].latest_used < min_time){
```

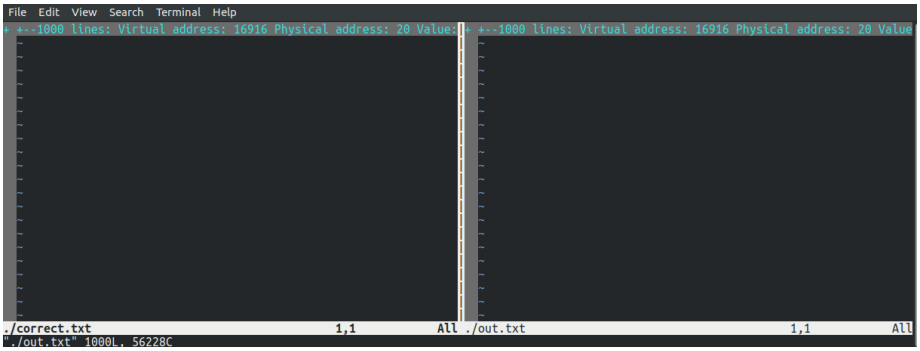
```
                    min_index = i;
                    min_time = mem[i].latest_used;
                }
            }
            for(int i = 0;i < PAGE_TABLE_SIZE;++i){
                if(page_table[i].frame_number == min_index
&& page_table[i].valid){
                    page_table[i].valid = 0;
                    break;
                }
            }
            LRU_mem(min_index, page_number);
            frame_id = min_index;
            page_table[page_number].frame_number =
frame_id;
            LRU_tlb(page_number,frame_id);

        }
        //printf("file opened---\n");
        //printf("%d %d", frame_id, offset);
        int data = mem[frame_id].data[offset];

        fprintf(out, "Virtual address: %d Physical
address: %d Value: %d\n", address, frame_id * FRAME_SIZE +
offset, data);
        fscanf(addr, "%d", &address);
    }

    double tlb_hit_rate = (double)tlb_hit_number / count;
    double page_fault_rate = (double)page_fault_number /
count;
    printf("TLB hit rate: [%d], page fault rate : [%f]\n",
tlb_hit_number, page_fault_rate);
    close_file();
    return 0;
}
```

Test and Evaluation

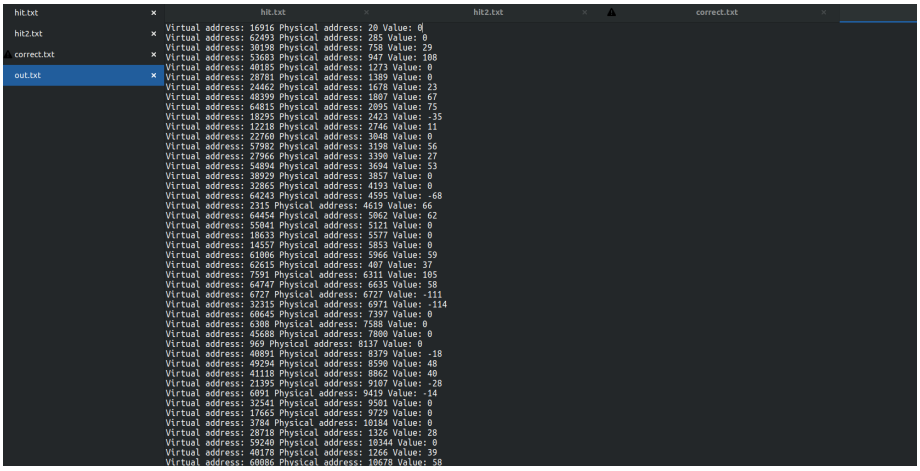
Test for the simulation includes two part:

- Check if the output is the same as that in the `correct.txt`
- Calculate the page fault rate and the TLB hit rate

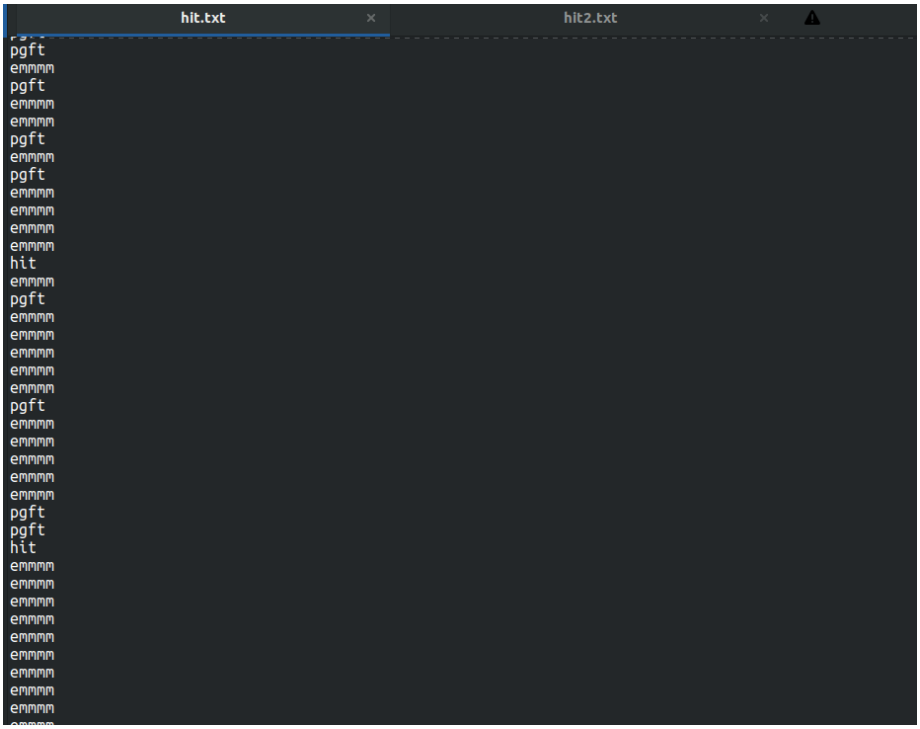
Since we output the result in `out.txt` , then we can use `vimdiff out.txt correct.txt` to check if the two files are same(Nothing shows on the terminal means they are the same)



The `out.txt` is as follows:

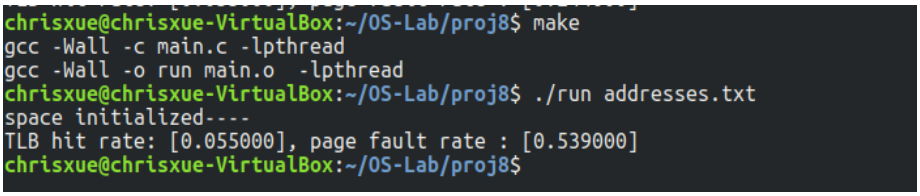


We also output the status like TLB hit, page fault and the other in the `hit.txt` :

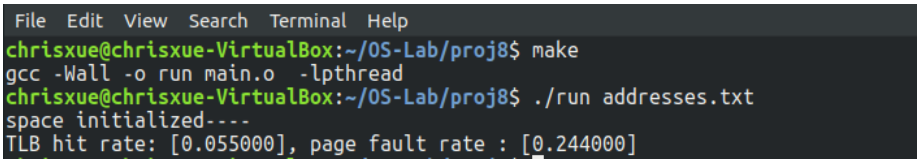


Finally we calculate the page fault rate and TLB hit rate:

For `FRAME_NUMER=128`:



For `FRAME_NUMER=256`:



Conclusion

This project helps me have a better understanding of how the demand paging and the LRU works in the virtual memory, also it helps me learn how to program in C in Linux better.

Thanks for the instructions and useful help offered by Prof. Wu and all the TAs!