操作系统 project-8 实验报告

∗ 姓名: 管仁阳 学号:519021911058 邮箱: guanrenyang@sjtu.edu.cn

1 实验名称

Designing a Virtual Memory Manager

2 实验任务

编写 C 程序来实现将对于 65536Bytes 的虚拟内存空间中虚拟内存地址转化为物理内存地址。程序必须使用 TLB 和页表。程序需要将虚拟内存地址转化为物理内存地址并且输出对应位置上的值。此外，需要通过按需分页、管理 TLB 并实现页面替换算法来解决业面失效的问题。

3 预备知识

3.1 Virtual Memory：

虚拟内存是一种允许执行进程不完全存储在内存中的技术（进程看到的虚拟内存大于实际物理内存）。此方案的一个主要优点是程序可以大于物理内存。此外，虚拟内存将主内存分成一个很大的统一的存储阵列，由程序员从物理内存中查看。这项技术使程序员摆脱了对内存存储限制的担忧。虚拟内存还允许进程共享文件和库，并且实现共享内存。另外，它提供了一种有效的机制用于进程创建。但是，虚拟内存不容易实现，并且如果不小心使用它，可能会大大降低性能。

3.2 Demand Paging

按需分页的基本概念就是：当且仅当一个页一定需要被使用时才将其加载进入内存，否则就留在主存中。

3.3 Page Replacement

当一个页面失效出现失效 (需要使用的页并不在物理内存中而只在 back store 中)，就需要将页对应的页框加载进入内存。但是如果出现了页表已满的情况就需要考虑将以后页面从页表中替换出去。

3.4 TLB

TLB 是一个高速内存。TLB 中的每一个入口包含两个部分：1. tag 2. value 对于每一个虚拟内存地址，都将先在 TLB 中查找对应的页框号，若此页不再 TLB 中则再进入页表中查找。TLB 的入口数远小于页表，但是 TLB 的速度远大于内存。可以理解为 TLB 为页表的缓存。

4 实验内容

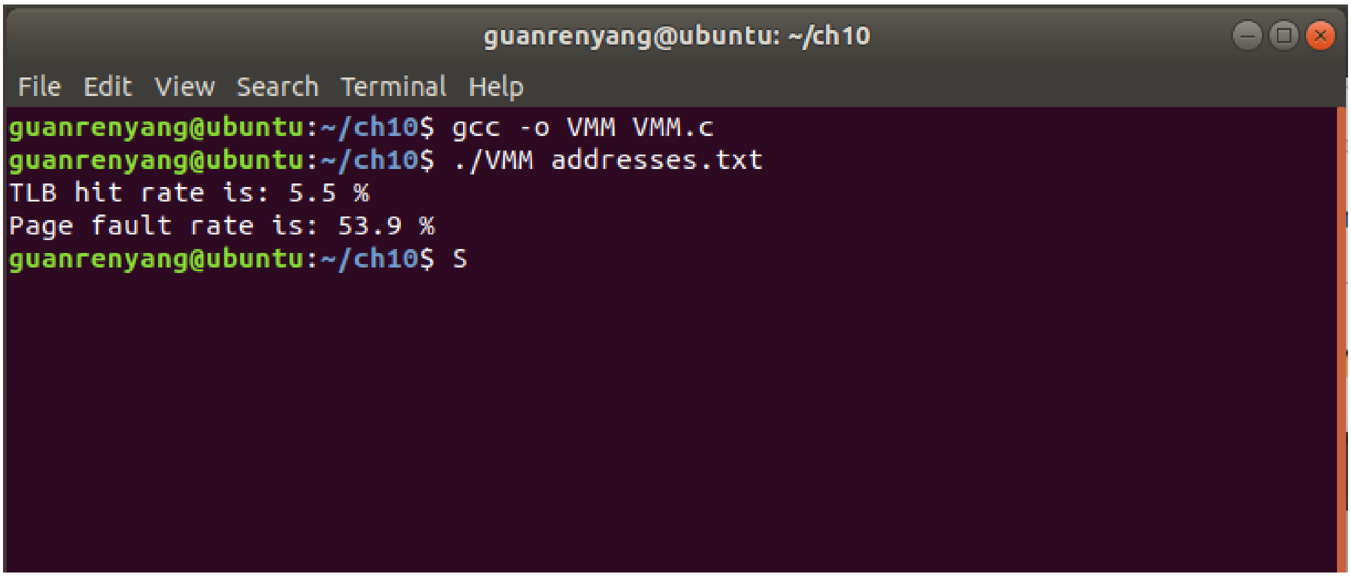
```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

# define PAGE_NUMBER 256
# define PAGE_SIZE 256
# define FRAME_NUMBER 128
# define FRAME_SIZE 256
# define TLB_SIZE 16

struct empty_frame {
        int frame_number;
        struct empty_frame *next;
};

struct empty_frame *head = NULL;
struct empty_frame *tail = NULL;

int page_table[PAGE_NUMBER];
int valid[PAGE_NUMBER];
int page_fault_count;

char memory[FRAME_NUMBER * FRAME_SIZE];
int LRU[FRAME_NUMBER];
char buf[FRAME_SIZE];
FILE *backing_store;

int TLB_page[TLB_SIZE], TLB_frame[TLB_SIZE];
int TLB_LRU[TLB_SIZE];
int TLB_hit;

void add_frame(int frame_number) {
        if (head == NULL && tail == NULL) {
                tail = (struct empty_frame *) malloc (sizeof(struct empty_fra
                tail -> frame_number = frame_number;
                tail -> next = NULL;
                head = tail;
        } else {
                tail -> next = (struct empty_frame *) malloc (sizeof(struct e
                tail -> next -> frame_number = frame_number;
                tail -> next -> next = NULL;
                tail = tail -> next;
        }
}
int get_frame() {
        if (head == NULL && tail == NULL) return -1;
        int frame_number;
        if (head == tail) {
                frame_number = head -> frame_number;
                free(head);
                head = tail = NULL;
                return frame_number;
        }
        struct empty_frame *tmp;
        frame_number = head -> frame_number;
```

```c
        tmp = head;
        head = head -> next;
        free(tmp);
        return frame_number;
}
void initialize_frame() {
        for (int i = 0; i < FRAME_NUMBER; ++i)
                add_frame(i);
}


void delete_TLB(int page_number, int frame_number);
void page_table_delete(int frame_number)
{
        int page_number = -1;
        for (int i = 0; i < PAGE_NUMBER; ++i)
                if(valid[i] && page_table[i] == frame_number) {
                        page_number = i;
                        break;
                }
        valid[page_number] = 0;
        delete_TLB(page_number, frame_number);
}
int add_page(int page_number) {
        fseek(backing_store, page_number * FRAME_SIZE, SEEK_SET);
  fread(buf, sizeof(char), FRAME_SIZE, backing_store);

        int frame_number = get_frame();
        if (frame_number == -1) {
                for (int i = 0; i < FRAME_NUMBER; ++i)
                        if (LRU[i] == FRAME_NUMBER) {
                                frame_number = i;
                                break;
                        }
                page_table_delete(frame_number);
        }

        for (int i = 0; i < FRAME_SIZE; ++i)
                memory[frame_number * FRAME_SIZE + i] = buf[i];
        for (int i = 0; i < FRAME_NUMBER; ++i)
                if (LRU[i] > 0) ++LRU[i];
        LRU[frame_number] = 1;
        return frame_number;
}

char access_memory(int frame_number, int offset) {
        char res = memory[frame_number * FRAME_SIZE + offset];
        for (int i = 0; i < FRAME_NUMBER; ++i)
                if (LRU[i] > 0 && LRU[i] < LRU[frame_number])
                        ++LRU[i];
        LRU[frame_number] = 1;
        return res;
}

int get_TLB_frame_num(int page_number) {
        int pos = -1;
        for (int i = 0; i < TLB_SIZE; ++i)
                if (TLB_LRU[i] > 0 && TLB_page[i] == page_number) {
                        pos = i;
                        break;
                }

        if (pos == -1) return -1;
        ++TLB_hit;
        for (int i = 0; i < TLB_SIZE; ++i)
                if (TLB_LRU[i] > 0 && TLB_LRU[i] < TLB_LRU[pos])
                        ++TLB_LRU[i];
        TLB_LRU[pos] = 1;
        return TLB_frame[pos];
}

void update_TLB(int page_number, int frame_number) {
        int pos = -1;
        for (int i = 0; i < TLB_SIZE; ++i)
                if(TLB_LRU[i] == 0) {
                        pos = i;
                        break;
                }
        if (pos == -1) {
                for (int i = 0; i < TLB_SIZE; ++i)
                        if(TLB_LRU[i] == TLB_SIZE) {
                                pos = i;
                                break;
                        }
        }

        TLB_page[pos] = page_number;
        TLB_frame[pos] = frame_number;
        for (int i = 0; i < TLB_SIZE; ++i)
                if (TLB_LRU[i] > 0) ++TLB_LRU[i];
        TLB_LRU[pos] = 1;
}
void delete_TLB(int page_number, int frame_number) {
        int pos = -1;
        for (int i = 0; i < TLB_SIZE; ++i)
                if(TLB_LRU[i] && TLB_page[i] == page_number && TLB_frame[i] =
                        pos = i;
                        break;
                }
        if (pos == -1) return;
```

```c
        for (int i = 0; i < TLB_SIZE; ++i)
                if (TLB_LRU[i] > TLB_LRU[pos])  TLB_LRU[i]--;
        TLB_LRU[pos] = 0;
}

void initialize() {
        page_fault_count = 0;
        for (int i = 0; i < PAGE_NUMBER; ++i) {
                page_table[i] = 0;
                valid[i] = 0;
        }

        TLB_hit = 0;
        for (int i = 0; i < TLB_SIZE; ++i) {
                TLB_page[i] = 0;
                TLB_frame[i] = 0;
                TLB_LRU[i] = 0;
        }

        backing_store = fopen("BACKING_STORE.bin", "rb");
        initialize_frame();
        for (int i = 0; i < FRAME_NUMBER; ++i)
                LRU[i] = 0;
}

int get_frame_number(int page_number) {
        if (page_number < 0 || page_number >= PAGE_NUMBER) return -1;

        int TLB_res = get_TLB_frame_num(page_number);
        if (TLB_res != -1) return TLB_res;

        if (valid[page_number] == 1) {
                update_TLB(page_number, page_table[page_number]);
                return page_table[page_number];
        } else {
                ++page_fault_count;
                page_table[page_number] = add_page(page_number);
                valid[page_number] = 1;
                update_TLB(page_number, page_table[page_number]);
                return page_table[page_number];
        }
}

int main(int argc, char *argv[]) {

        FILE *input_file = fopen(argv[1], "r");
        FILE *output_file = fopen("output.txt", "w");

        initialize();

        int address, page_number, offset, frame_number, result, count = 0;
        while(~fscanf(input_file, "%d", &address)) {
                ++count;
                address = address & 0x0000ffff;
                offset = address & 0x000000ff;
                page_number = (address >> 8) & 0x000000ff;
                frame_number = get_frame_number(page_number);
                result = (int) access_memory(frame_number, offset);
                fprintf(output_file, "Virtual address: %d Physical address: %
        }

        double TLB_hit_rate=100.0 * TLB_hit / count;
        double page_fault_rate=100.0 * page_fault_count / count;
        fprintf(stdout, "TLB hit rate is: %.1f %%\nPage fault rate is: %.1f %

        return 0;
}
```

5 实验结果

实验结果如图 2，数据正确性可见图 1



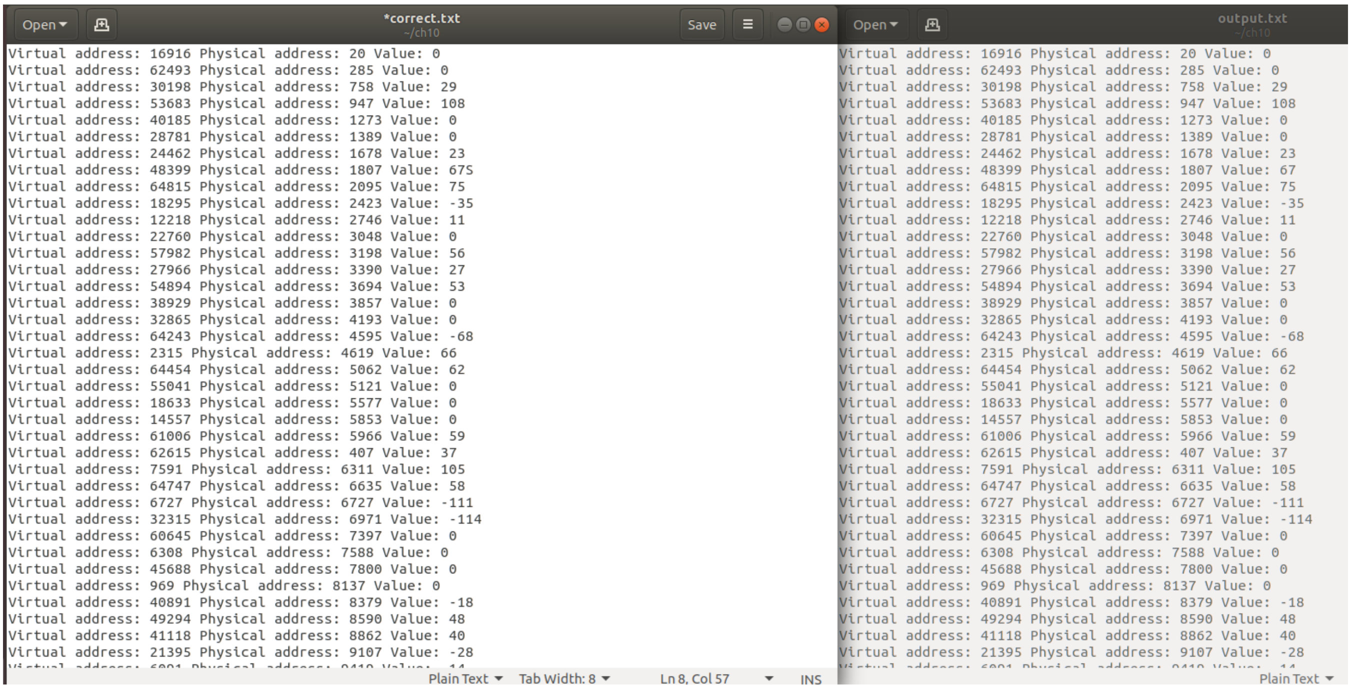
guanrenyang@ubuntu: ~/ch10

```
guanrenyang@ubuntu:~/ch10$ gcc -o VMM VMM.c
guanrenyang@ubuntu:~/ch10$ ./VMM addresses.txt
TLB hit rate is: 5.5 %
Page fault rate is: 53.9 %
guanrenyang@ubuntu:~/ch10$ S
```

图 1: Result

6 总结与思考

通过此次实验我实现了虚拟内存管理器，它将一个逻辑地址转变为物理地址并且访问对应物理地址，在这之中我还使用到了按需分页、页面替换、快表等技术。
这是一个综合性很强的实验，帮助我深入理解了现代计算机操作系统的页式内存组织方式。我认识到了将虚拟内存与物理内存区分的重要性。同时我还深入理解了 LRU 替换算法的与缓存的机制。



```
Virtual address: 16916 Physical address: 20 Value: 0
Virtual address: 62493 Physical address: 285 Value: 0
Virtual address: 30198 Physical address: 758 Value: 29
Virtual address: 53683 Physical address: 947 Value: 108
Virtual address: 40185 Physical address: 1273 Value: 0
Virtual address: 28781 Physical address: 1389 Value: 0
Virtual address: 24462 Physical address: 1678 Value: 23
Virtual address: 48399 Physical address: 1807 Value: 67S
Virtual address: 64815 Physical address: 2095 Value: 75
Virtual address: 18295 Physical address: 2423 Value: -35
Virtual address: 12218 Physical address: 2746 Value: 11
Virtual address: 22760 Physical address: 3048 Value: 0
Virtual address: 57982 Physical address: 3198 Value: 56
Virtual address: 27966 Physical address: 3390 Value: 27
Virtual address: 54894 Physical address: 3694 Value: 53
Virtual address: 38929 Physical address: 3857 Value: 0
Virtual address: 32865 Physical address: 4193 Value: 0
Virtual address: 64243 Physical address: 4595 Value: -68
Virtual address: 2315 Physical address: 4619 Value: 66
Virtual address: 64454 Physical address: 5062 Value: 62
Virtual address: 55041 Physical address: 5121 Value: 0
Virtual address: 18633 Physical address: 5577 Value: 0
Virtual address: 14557 Physical address: 5853 Value: 0
Virtual address: 61006 Physical address: 5966 Value: 59
Virtual address: 62615 Physical address: 407 Value: 37
Virtual address: 7591 Physical address: 6311 Value: 105
Virtual address: 64747 Physical address: 6635 Value: 58
Virtual address: 6727 Physical address: 6727 Value: -111
Virtual address: 32315 Physical address: 6971 Value: -114
Virtual address: 60645 Physical address: 7397 Value: 0
Virtual address: 6308 Physical address: 7588 Value: 0
Virtual address: 45688 Physical address: 7800 Value: 0
Virtual address: 969 Physical address: 8137 Value: 0
Virtual address: 40891 Physical address: 8379 Value: -18
Virtual address: 49294 Physical address: 8590 Value: 48
Virtual address: 41118 Physical address: 8862 Value: 40
Virtual address: 21395 Physical address: 9107 Value: -28
```

```
Virtual address: 16916 Physical address: 20 Value: 0
Virtual address: 62493 Physical address: 285 Value: 0
Virtual address: 30198 Physical address: 758 Value: 29
Virtual address: 53683 Physical address: 947 Value: 108
Virtual address: 40185 Physical address: 1273 Value: 0
Virtual address: 28781 Physical address: 1389 Value: 0
Virtual address: 24462 Physical address: 1678 Value: 23
Virtual address: 48399 Physical address: 1807 Value: 67
Virtual address: 64815 Physical address: 2095 Value: 75
Virtual address: 18295 Physical address: 2423 Value: -35
Virtual address: 12218 Physical address: 2746 Value: 11
Virtual address: 22760 Physical address: 3048 Value: 0
Virtual address: 57982 Physical address: 3198 Value: 56
Virtual address: 27966 Physical address: 3390 Value: 27
Virtual address: 54894 Physical address: 3694 Value: 53
Virtual address: 38929 Physical address: 3857 Value: 0
Virtual address: 32865 Physical address: 4193 Value: 0
Virtual address: 64243 Physical address: 4595 Value: -68
Virtual address: 2315 Physical address: 4619 Value: 66
Virtual address: 64454 Physical address: 5062 Value: 62
Virtual address: 55041 Physical address: 5121 Value: 0
Virtual address: 18633 Physical address: 5577 Value: 0
Virtual address: 14557 Physical address: 5853 Value: 0
Virtual address: 61006 Physical address: 5966 Value: 59
Virtual address: 62615 Physical address: 407 Value: 37
Virtual address: 7591 Physical address: 6311 Value: 105
Virtual address: 64747 Physical address: 6635 Value: 58
Virtual address: 6727 Physical address: 6727 Value: -111
Virtual address: 32315 Physical address: 6971 Value: -114
Virtual address: 60645 Physical address: 7397 Value: 0
Virtual address: 6308 Physical address: 7588 Value: 0
Virtual address: 45688 Physical address: 7800 Value: 0
Virtual address: 969 Physical address: 8137 Value: 0
Virtual address: 40891 Physical address: 8379 Value: -18
Virtual address: 49294 Physical address: 8590 Value: 48
Virtual address: 41118 Physical address: 8862 Value: 40
Virtual address: 21395 Physical address: 9107 Value: -28
```

图 2: Result