



**《计算机操作系统》**  
**——第五次实验报告**

**学 号： 15281106**

**姓 名： 徐开元**

**专 业： 计算机科学与技术**

**学 院： 计算机与信息技术学院**

**提交日期： 2018 年 5 月 22 日**

**指导老师： 黄华**

《计算机操作系统》 .....	1
——第五次实验报告.....	1
一、 实验内容： .....	2
二、 程序描述.....	3
2.1 头文件 ( function.h/cpp ) .....	3
2.2 主程序逻辑.....	3
三、 实验心得.....	8

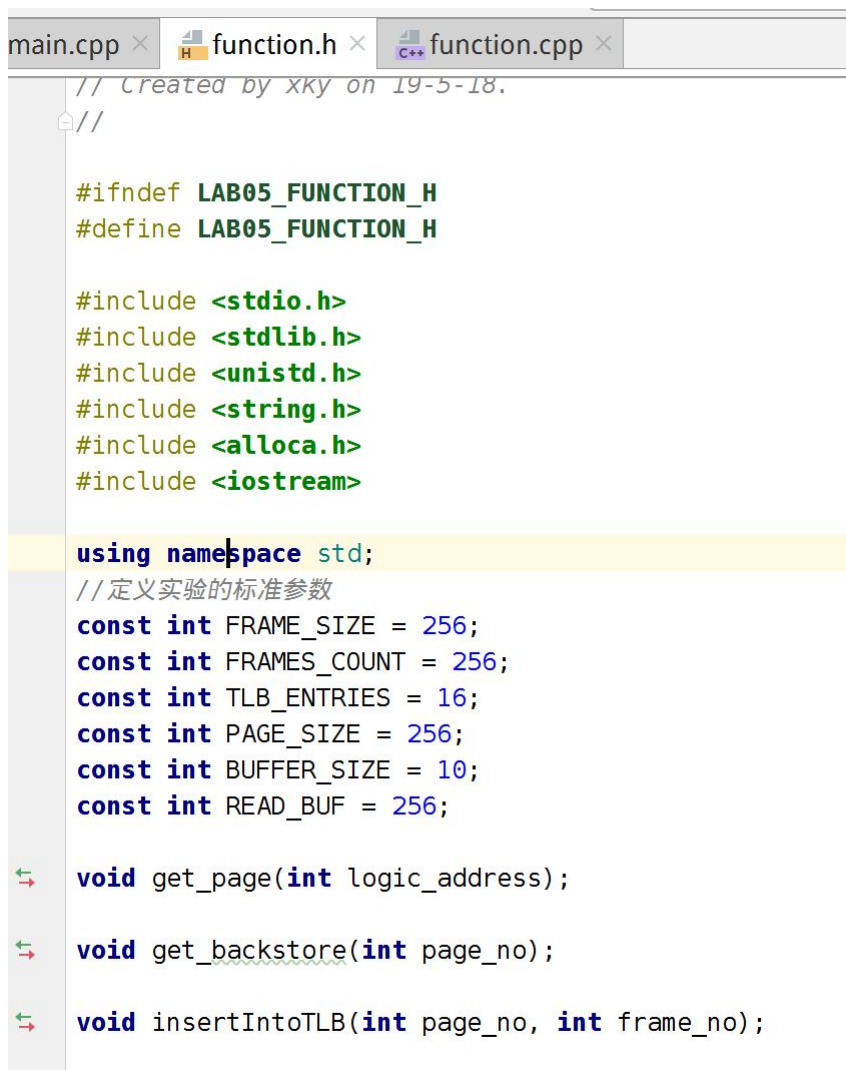
## 一、 实验内容：

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size  $2^{16} = 65,536$  bytes. Your program will read from a file containing logical addresses and, using a TLB and a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging, managing a TLB , and implementing a page-replacement algorithm.

## 二、程序描述

### 2.1 头文件 (function.h/cpp)

引用了工程里常用的库文件，定义了一些实验中的标准参数，比如 frame，表，TLB 的大小，以及读取 backing store.bin 二进制文件、查找表，插入缓存 TLB 的函数。



```
// Created by xky on 19-5-18.
//

#ifndef LAB05_FUNCTION_H
#define LAB05_FUNCTION_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <alloca.h>
#include <iostream>

using namespace std;
// 定义实验的标准参数
const int FRAME_SIZE = 256;
const int FRAMES_COUNT = 256;
const int TLB_ENTRIES = 16;
const int PAGE_SIZE = 256;
const int BUFFER_SIZE = 10;
const int READ_BUF = 256;

void get_page(int logic_address);

void get_backstore(int page_no);

void insertIntoTLB(int page_no, int frame_no);
```

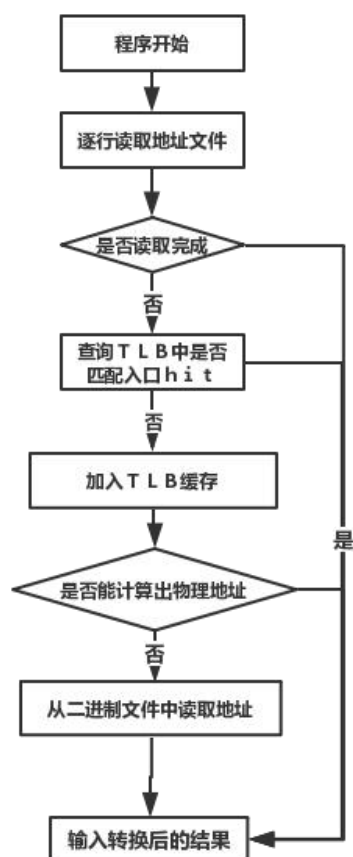
### 2.2 主程序逻辑

基本思路：

- 1、运行程序，逐行读取 addresses.txt，获取需要转换的逻辑地址。

- 2、获取到的逻辑地址，并转为 int 型。通过按位与运算的得到页序和偏移量。
- 3、先去 TLB 表中查找是否缓存了要查询的相应页表数据，如果有就取出，并且 hit\_count 加 1。
- 4、如果 TLB 中找不到，就说明没有进行缓存，因此加入缓存中，使用了相对简单实现的 F I F O 置换策略。
- 5、访问页表，取出虚拟页对应的物理页，再访问实际地址，计算得到物理地址。如果找不到就报 page\_fault 缺页中断加 1。
- 6、还是找不到的话就从 backingstore 二进制文件中读取相应的地址。
- 7、打印出转换得到的物理地址。

主程序流程图：



```
int main() {
    while (fgets(address, BUFFER_SIZE, input_file) != nullptr) {
        // 字节转32位数字.
        logic_address = atoi(address);
        get_page(logic_address);
        trans_count++;
        // exit(0);
    }
    double page_fault_ratio = (double) page_faults_count / (double) trans_count;
    double TLB_hit_ratio = (double) TLB_hit_count / (double) trans_count;
    cout << "转换的地址次数：" << trans_count << endl;
    cout << "缺页错误的次数：" << page_faults_count << endl;
    cout << "缺页错误率：" << page_fault_ratio << endl;
    cout << "TLB命中次数：" << TLB_hit_count << endl;
    cout << "TLB命中率：" << TLB_hit_ratio << endl;
    return 0;
}
```

查询表数据函数，执行主要逻辑。

```
// 通过按位与运算得到页序和偏移量，页序取后八位。
int page_no = ((add & 0xFFFF) >> 8);
int offset = (add & 0xFF);
int frame_no = -1;
int i;
// 先查找TLB中是否缓存了该页，如果有就取出，并且hit+1;
for (i = 0; i < TLB_ENTRIES; i++) {
    if (TLB_num_arr[i] == page_no) {
        frame_no = TLB_frame_arr[i];
        TLB_hit_count++;
    }
}
```

如果TLB没有hit直接拿到页表数据，那么就通过两次访存获取表页数据。如果还是找不到就去backing\_store二进制文件中读取页表数据。

```

// 如果没有就开始去 t a b l e 中查找 .
if (frame_no == -1) {
    for (i = 0; i < first_free_num; i++) {
        if (tables_num_arr[i] == page_no) {
            frame_no = tables_frame_arr[i];
        }
    }
    if (frame_no == -1) { // 如果找不到就缺页错误就加 1
        get_backstore(page_no);
        page_faults_count++;
        frame_no = first_free_frame - 1;
    }
}
}

```

使用 c 语言自带的 `f s e e k` 函数，通过传入的页表号，定位到页表位置。获取到页表信息，并装入 `phy_mem_matrix` 二维矩阵变量中。

```

// 产生缺页错误就从 b a c k . b i n 中获取匹配的页表 . 填充这一页的所有 frame;
void get_backstore(int page_no) {
    // 使用 c 语言自带的 f s e e k 函数，定位到页表位置 .
    int seek_status = fseek(backing_store, page_no * READ_BUF, SEEK_SET);
    int value = fread(buffer, sizeof(char), READ_BUF, backing_store);
    int i;
    // cout<<page_no;
    for (i = 0; i < READ_BUF; i++) {
        phy_mem_matrix[first_free_frame][i] = buffer[i];
    }
    // cout<<phy_mem_matrix[0][20];
    tables_num_arr[first_free_num] = page_no;
    tables_frame_arr[first_free_num] = first_free_frame;
    first_free_frame++;
    first_free_num++;
}

```

更新 TLB 表，采用 FIFO 先进先出的策略。如果表满的话，在表中最早添加的就会被推出去。

```

//采用的是 F I F O 置换策略
void insertIntoTLB(int page_no, int frame_no) {
    int i;
    // 如果已经在 T L B 中存在则退出
    for (i = 0; i < TLB_entries; i++) {
        if (TLB_num_arr[i] == page_no) {
            break;
        }
    }
    // 如果表中有就更新
    if (i == TLB_entries) {
        if (TLB_entries < TLB_ENTRIES) { // 如果有空间则加入
            TLB_num_arr[TLB_entries] = page_no;
            TLB_frame_arr[TLB_entries] = frame_no;
        } else {
            for (i = 0; i < TLB_ENTRIES - 1; i++) {
                TLB_num_arr[i] = TLB_num_arr[i + 1];
                TLB_frame_arr[i] = TLB_frame_arr[i + 1];
            }
            TLB_num_arr[TLB_entries - 1] = page_no;
            TLB_frame_arr[TLB_entries - 1] = frame_no;
        }
    }
    // 如果没有就置换添加
    else {
        for (i = i; i < TLB_entries - 1; i++) {
            TLB_num_arr[i] = TLB_num_arr[i + 1];
            TLB frame arr[i] = TLB frame arr[i + 1];
        }
    }
}

```

实验最后结果，输出部分的结果与 correct.txt 文件中的值相同。

页号：66	偏移量：20	逻辑地址：16916	物理地址：20	值：0
页号：244	偏移量：29	逻辑地址：62493	物理地址：285	值：0
页号：117	偏移量：246	逻辑地址：30198	物理地址：758	值：29
页号：209	偏移量：179	逻辑地址：53683	物理地址：947	值：108
页号：156	偏移量：249	逻辑地址：40185	物理地址：1273	值：0
页号：112	偏移量：109	逻辑地址：28781	物理地址：1389	值：0
页号：95	偏移量：142	逻辑地址：24462	物理地址：1678	值：23
页号：189	偏移量：15	逻辑地址：48399	物理地址：1807	值：67
页号：253	偏移量：47	逻辑地址：64815	物理地址：2095	值：75
页号：71	偏移量：119	逻辑地址：18295	物理地址：2423	值：-35
页号：47	偏移量：186	逻辑地址：12218	物理地址：2746	值：11
页号：88	偏移量：232	逻辑地址：22760	物理地址：3048	值：0
页号：226	偏移量：126	逻辑地址：57982	物理地址：3198	值：56
页号：109	偏移量：62	逻辑地址：27966	物理地址：3390	值：27



```

页号：187    偏移量：193    逻辑地址：48065    物理地址：25793    值：0
页号：27    偏移量：45     逻辑地址：6957     物理地址：26413    值：0
页号：8     偏移量：253    逻辑地址：2301     物理地址：35325    值：0
页号：30    偏移量：56     逻辑地址：7736     物理地址：57912    值：0
页号：122    偏移量：28     逻辑地址：31260    物理地址：23324    值：0
页号：66    偏移量：175    逻辑地址：17071    物理地址：175      值：-85
页号：34    偏移量：236    逻辑地址：8940     物理地址：46572    值：0
页号：38    偏移量：201    逻辑地址：9929     物理地址：44745    值：0
页号：177    偏移量：251    逻辑地址：45563    物理地址：46075    值：126
页号：47    偏移量：75     逻辑地址：12107    物理地址：2635     值：-46
转换的地址次数：1000
缺页终端的次数：244
缺页终端率：0.244
TLB命中次数：55
TLB命中率：0.055
    
```

### 三、实验心得

本次实验让我获益匪浅，对于程序的结果我也有很多疑问。

1、为什么会发生缺页中断，如果他是对于操作系统是不好的，那么为什么还总是频繁发生。

缺页中断就是要访问的页不在主存，需要操作系统将其调入主存后再进行访问。但他也不是完全不好的，操作系统可以利用这个情况去增加虚拟内存来增加可用的内存。

2、为什么 T L B 缓存比较简单的思路，却被广泛的使用。因为一个大程序可能会有产生上万的查表访存的操作，而 T L B 能够在每次查表都减少一次内存访问，并且还能继续通过算法对于那些频繁访问的内存地址进行缓存，进一步增加内存地址的转换速度。并且自身占用空间很小，用了小空间来换取了很多的时间，因此被广泛使用。

3、为什么要使用逻辑地址，如果不使用，就不用花时间在转换上了。其实想想还是比较简单，因为逻辑地址是给操作系统之上的软件看的。他们不需要知道硬件是怎么设计的，不同的硬件可能有不同的物理地址，操作系统只需要一个虚拟的地址去操作就可以了。使用逻辑地址就可以让同样的软件应用于不同的硬件上。经过这次实验我深刻的体会到了内存的作用已经操作系统对于内存的完美掌控，提高了自己的动手能力，并且也加深了自己的对于概念算法的理解。



