虚拟内存管理模拟程序实验报告

18364048 李尚林

1.原代码及注释：

#include <assert.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define TLB\_SIZE 16 //TLB有16个条目

#define PAGE\_SIZE 256 //页面大小为256

#define FRAME\_SIZE 256 //帧数为256

#define PHYSICAL\_MEMORY\_SIZE 65536 //物理内存为65536字节

//全局变量

int logicalAddress = 0; //逻辑地址

int offsetNumber = 0; //偏移量

int pageNumber = 0; //页码

int physicalAddress = 0; //物理地址

int Frame = 0; //帧码

int Value = 0; //数据

int Hit = 0; //TLB击中后用于记录帧码

int tlbIndex = 0; //TLB索引

int tlbSize = 0; //TLB大小

unsigned pageNumberMask = 65280; //页表掩码，65280相当于二进制数1111111100000000

unsigned offsetMask = 255; //偏移掩码，255相当于二进制数0000000011111111

int tlbHitCount = 0; //TLB击中数

float tlbHitRate = 0; //TLB命中率

int addressCount = 0; //地址数

int pageFaultCount = 0; //缺页数

float pageFaultRate = 0; //缺页率

struct tlbTable { //TLB结构体

unsigned int pageNum;

unsigned int frameNum;

};

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

//检查是否已将BACKING\_STORE.bin和addresses.txt传给main函数

if (argc != 3) {

printf("没有传入BACKING\_STORE.bin和addresses.txt");

exit(1);

}

FILE\* BACKINGSTORE = fopen(argv[1], "rb"); //打开文件BACKING\_STORE.bin

FILE\* addresses = fopen(argv[2], "r"); //打开文件addresses.txt

FILE\* Output = fopen("out.txt", "w"); //新建文件out.txt，用于存储输出结果

//声明物理内存数组、缓冲数组、索引

int physicalMemory[PHYSICAL\_MEMORY\_SIZE];

char Buffer[256];

int Index;

//声明并初始化页表数组

int pageTable[PAGE\_SIZE];

memset(pageTable, -1, 256 \* sizeof(int));

//声明并初始化TLB结构体

struct tlbTable tlb[TLB\_SIZE];

memset(pageTable, -1, 16 \* sizeof(char));

//开始从addresses.txt中循环读取数据

while (fscanf(addresses,"%d",&logicalAddress) == 1) { //取得一个逻辑地址

addressCount++; //地址数加1

// 利用位掩码和位移动提取页码和偏移量

pageNumber = logicalAddress & pageNumberMask; //按位与运算，逻辑地址高8位保留作为页码，低8位置零

pageNumber = pageNumber >> 8; //右移8位获取正确的页码

offsetNumber = logicalAddress & offsetMask; //按位与运算，逻辑地址低8位保留作为偏移量，高8位置零

Hit = -1;

// 遍历TLB，寻找相对应的页码

for (Index = 0; Index < tlbSize; Index++)

if (tlb[Index].pageNum == pageNumber) { //TLB中找到对应页码

Hit = tlb[Index].frameNum; //提取帧码

physicalAddress = Hit \* 256 + offsetNumber; //计算物理地址

}

//如果变量Hit已被更新，TLB命中，命中数加1

if (Hit != -1) tlbHitCount++;

// 如果TLB未命中，就从页表获取物理地址，采取FIFO策略（与TLB统一策略）实现页面置换

else if (pageTable[pageNumber] == -1) { //如果该页对应帧码的值为-1，说明发生缺页错误，需请求调页

fseek(BACKINGSTORE, pageNumber \* 256, SEEK\_SET); //根据缺页的页码，将文件指针指向后备存储文件中应的位置

fread(Buffer, sizeof(char), 256, BACKINGSTORE); //从后备存储文件中调页，记录在缓冲中

pageTable[pageNumber] = Frame; //将对应帧码存储到页表中

for (Index = 0; Index < 256; Index++) //将缓冲中的数据加载到物理内存

physicalMemory[Frame \* 256 + Index] = Buffer[Index];

pageFaultCount++; //缺页数加1

Frame++; //物理内存帧码加1 ，指向下一个空闲帧

// TLB使用FIFO策略

if (tlbSize == 16) tlbSize--; //若TLB已满，TLB大小减1，队尾条目被移出队列

for (tlbIndex = tlbSize; tlbIndex > 0; tlbIndex--) { //TLB中所有条目后移

tlb[tlbIndex].pageNum = tlb[tlbIndex - 1].pageNum;

tlb[tlbIndex].frameNum = tlb[tlbIndex - 1].frameNum;

}

if (tlbSize <= 15) tlbSize++; //若TLB未满 ，TLB大小加1，将目前条目加入TLB队首

tlb[0].pageNum = pageNumber; //页码加入队首

tlb[0].frameNum = pageTable[pageNumber]; //帧码加入队首

physicalAddress = pageTable[pageNumber] \* 256 + offsetNumber; //取得物理地址，完成调页

}

else //没有缺页错误，则无需调页，直接获取帧码并计算物理地址

physicalAddress = pageTable[pageNumber] \* 256 + offsetNumber;

// 地址转换完成，从物理内存对应地址提取数据

Value = physicalMemory[physicalAddress];

// 将虚拟地址，物理地址和对应数据的值保存到out.txt中

fprintf(Output, "Virtual Address: %d Physical Address: %d Value: %d \n",

logicalAddress, physicalAddress, Value);

}

//计算缺页率和TLB命中率并打印到out.txt文件

pageFaultRate = pageFaultCount \* 1.0f / addressCount;

tlbHitRate = tlbHitCount \* 1.0f / addressCount;

fprintf(Output, "Page-fault rate: %f\n", pageFaultRate);

fprintf(Output, "TLB hit rate %f\n", tlbHitRate);

//关闭所有文件，程序结束

fclose(addresses);

fclose(BACKINGSTORE);

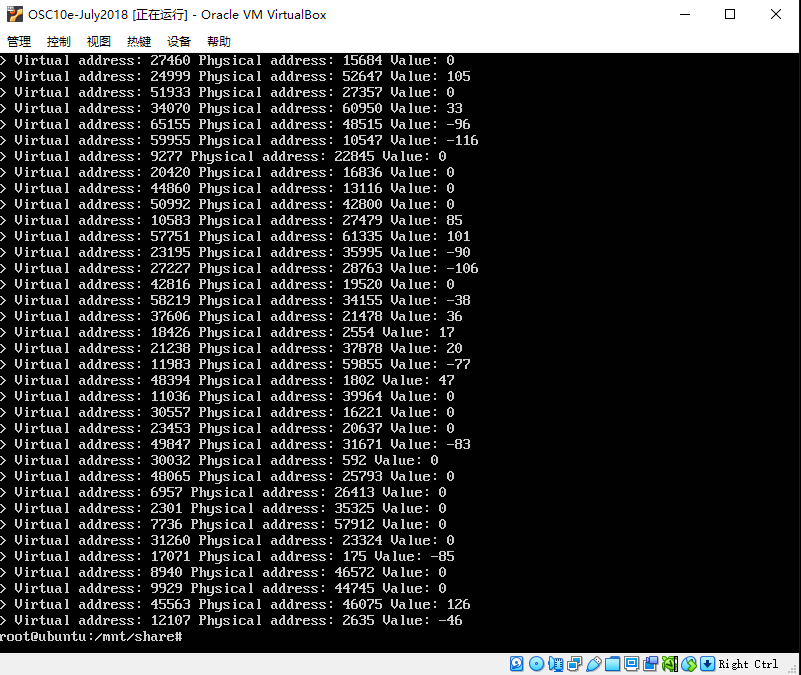
fclose(Output);

return 0;

}

2.实验结果

test.sh 脚本正常运行，结果正确



缺页率 0.244000

TLB命中率 0.055000