**北 京 林 业 大 学**

**2020 学年— 2021 学年第1 学期 操作系统 实验报告书**

专 业：计算机科学与技术 班 级：计算机191

姓 名：高蔚轩 学 号：191002418

实验地点：信息中心N07机房 辅导教师： 田萱

实验环境： C++语言编译器或Java语言编译器

# 一、实验目的

1、理解分页式存储管理和页表结构；

2、理解请求式分页存储管理的页面置换算法；

# 二、实验内容

1、初始化物理内存空间布局

设内存大小为4MB，页面大小为4KB。

2、为进程分割页面，随机存储到未用页框中，并构建其页表。

由用户创建进程，输入进程名称和所需存储大小，随机产生该进程的一个页面访问序列。

若内存区域有空闲，将内存中未占用页框随机分配给该进程存储使用，并构建其页表，随机设定其状态位（1表示在内存，0表示在外存）、访问位（状态位=1时，访问位为>0的整数，表示访问次数）和修改位（状态位=1时， 修改位=0表示未修改，修改位=1表示已修改）的值，并设置一个时间字段记录将该页面加载入内存的时刻。输出进程名称，页面号和分配的页块号。

若内存空间已满，分别实现LRU、FIFO和OPT算法进行页面置换。**输出页面置换的过程，以及该进程最后分配的页面号和页块号**。

# 三、实验过程

1、首先，设计一个页表项结构体，按照实验要求，其中包含6个数据域。

typedef struct Block

{

int pageID; //页面号

int chunkID; //页块号

bool state; //状态位

int visited; //访问位

bool changed; //修改位

long long time; //载入内存的时刻

} Block;

将time的数据类型设置为long long是为了与时间函数相匹配，后续会进行详细说明。

1. 用页表项构成页表

typedef struct PageTable

{

int progessID;

Block \*pages;

int length;

} PageTable;

PageTable pagetable[10];

1. 模拟物理内存空间，根据实验要求设内存大小为4MB，页面大小为4KB。通过计算可得：4MB/4KB=1024，一共1024个页面。

//内存,1为占用，0为未占用

int Memory[1024];

1. 为了记录进入时间，采用函数gettimeofday(&tv, NULL)，记录当前时间和程序开始时间，返回一个以毫秒为单位的值。为了匹配gettimeofday的单位，数据类型为long long。
2. 页面置换算法void load(PageTable &pt, int id, int l, int \*ra)

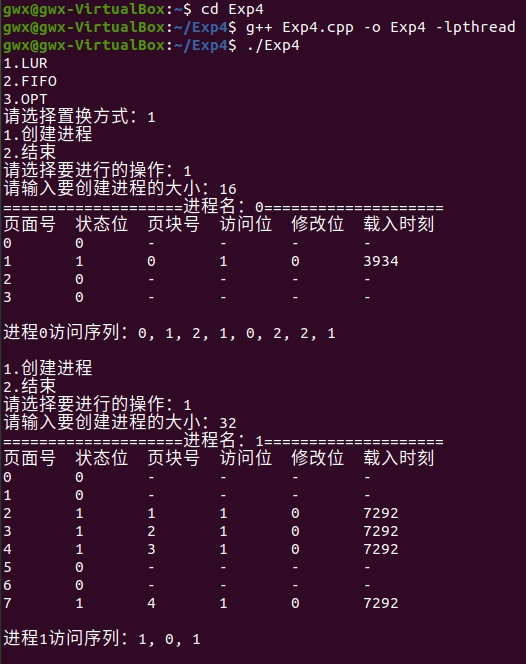
当内存未满时，直接调入。

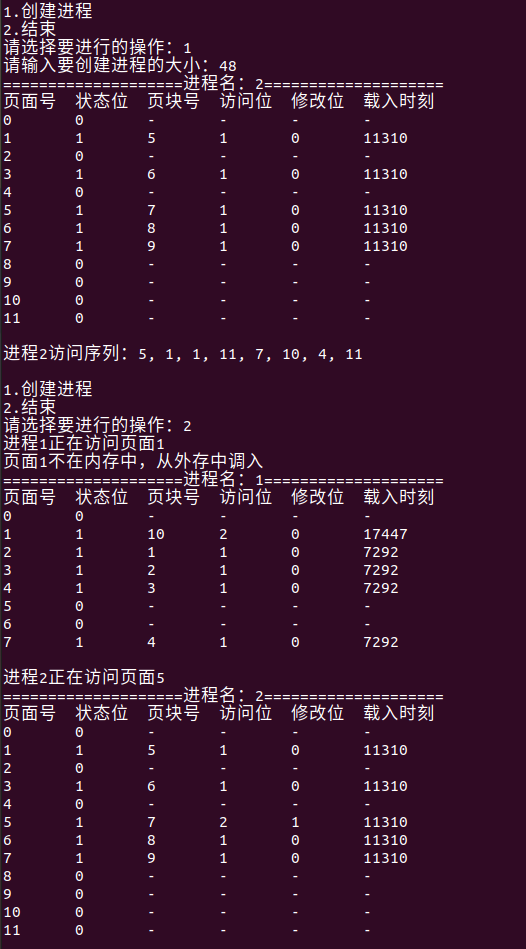
当内存已满时，采取三种页面置换算法。

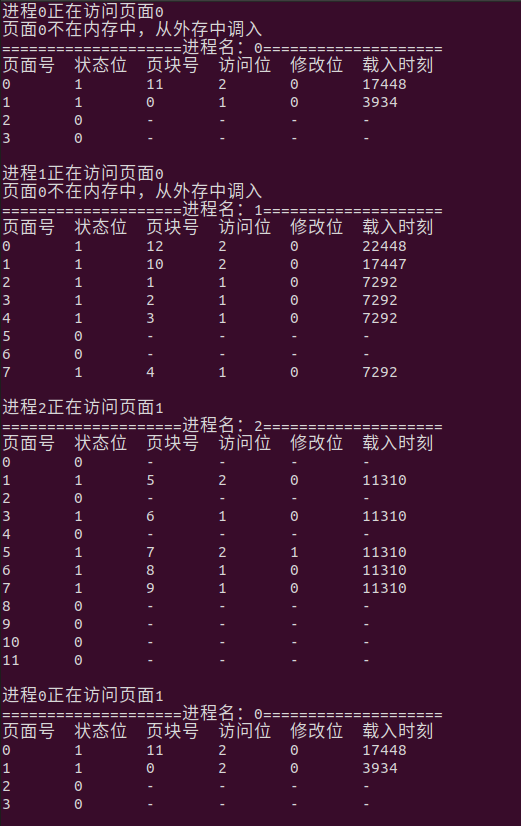
1. LRU：为了方便考虑，此处选择了更简单的软件实现算法，用访问次数代替访问时间。为每个页面建立一个计数器，当访问了某页后，此页对应的计数器增加1.淘汰时，系统查看计数器，选择值最小的计数器对应的页面淘汰。
2. FIFO：直接比较进入内存的时间，选择值最小的对应页面淘汰。
3. OPT：比较整个访问序列，淘汰距离下次使用时间最远的页面或不再使用的页面。用变量val评估每个页面，选择val最大的页面淘汰。从后往前扫描访问序列，val的记录本页面在访问序列下次使用的时间，如果不再使用则为初始值访问序列长度l。
4. 访问函数（多线程）void \*access\_fun(void \*arg)

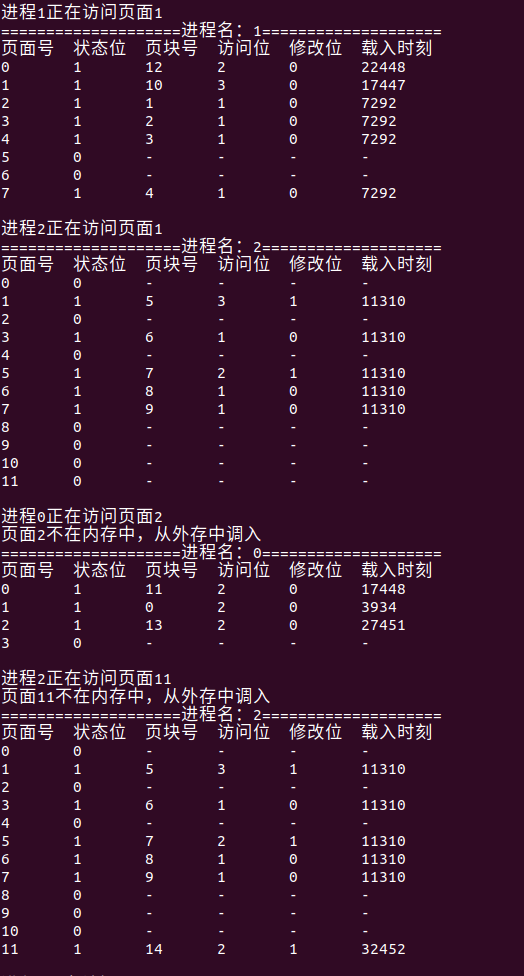
设置一个互斥变量rw，限制访问内存。每次访问先查询内存，如果不存在则从外存调入，每次都打印页表。

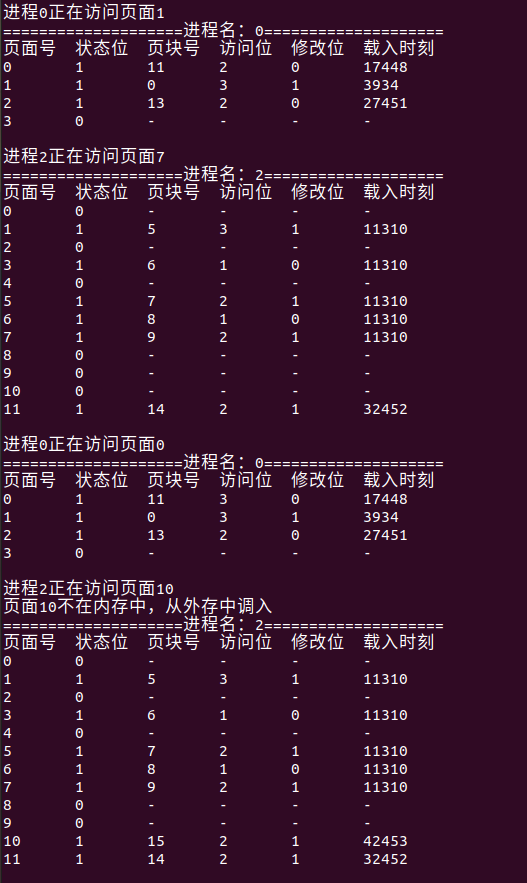
1. 实验结果

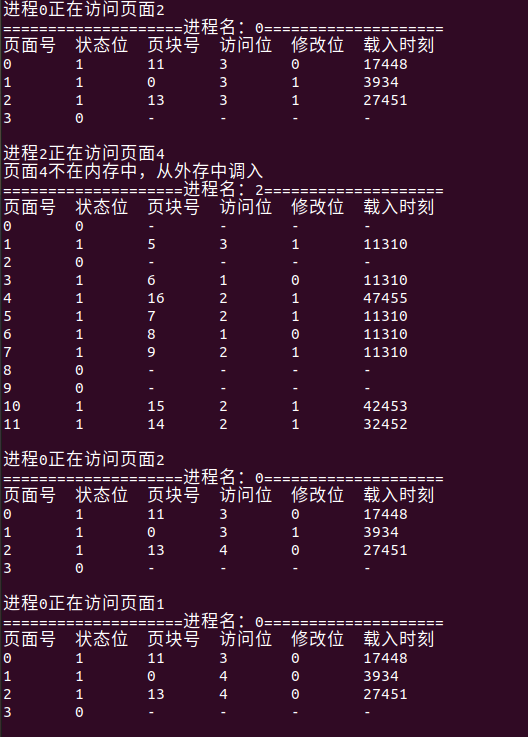








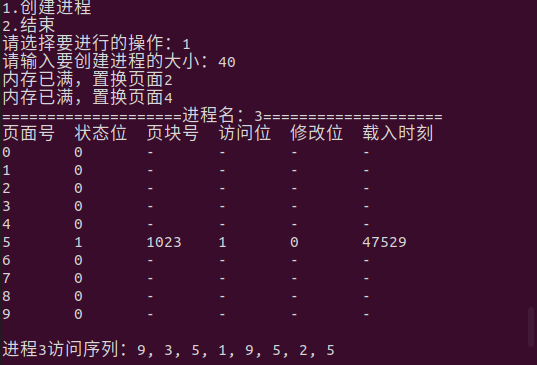


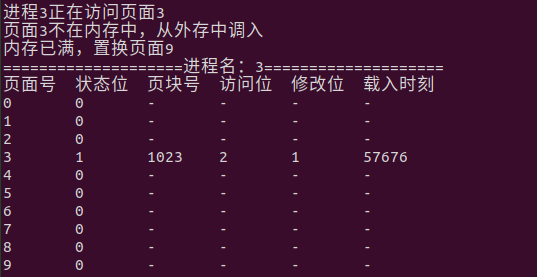




以上为一次完整的程序运行过程。

由于内存为较大，申请较小的进程时无法测试到页面置换，所以此处开辟一个较大的内存空间，测试结果不完全展示。





由于创建进程3时内存中只剩下一个页面的空间，所以进程3每次进行访问时都要重新调入页面。

# 心得体会

通过本次实验，我系统性的了解了三种页面置换算法和虚拟存储管理的分页机制，同时还熟悉了pthread.h库的相关应用，回顾了如何创建线程。最后我还学会了如何记录程序运行时间，并且更加直观地将其展现出来。

# 附录 实验代码

**Exp4.cpp**

#include <iostream>

#include <cstdlib>

#include <sys/time.h>

#include <pthread.h>

#include <unistd.h>

#include <math.h>

using namespace std;

#define random(a, b) (rand() % (b - a) + a)

enum Replacement

{

LRU,

FIFO,

OPT

};

const int maxm = 5, maxn = 10;

int r; //置换算法

int ra[maxm][maxn]; //最多生成10个访问

int l[maxm];

long long start;

pthread\_mutex\_t rw; //互斥访问内存

//页表项

typedef struct Block

{

int pageID; //页面号

int chunkID; //页块号

bool state; //状态位

int visited; //访问位

bool changed; //修改位

long long time; //载入内存的时刻

} Block;

//页表

typedef struct PageTable

{

int progessID;

Block \*pages;

int length;

} PageTable;

PageTable pagetable[maxm];

//内存,1为占用，0为未占用

//4MB/4KB=1024

int Memory[1024];

int getChunkID()

{

for (int i = 0; i < 1024; i++)

{

if (!Memory[i])

return i;

}

return -1;

}

//单位：毫秒

long long getTime()

{

timeval tv;

gettimeofday(&tv, NULL);

long long now = tv.tv\_sec \* 1000 \* 1000 + tv.tv\_usec;

return (now - start) / 1000;

}

void load(PageTable &pt, int id, int l, int \*ra)

{

int c = getChunkID();

if (c != -1)

{

pt.pages[id].chunkID = c;

Memory[c] = 1;

}

else

{

int t = 0, i = 0, maxval = l;

for (; i < pt.length; i++)

{

if (pt.pages[i].state)

{

t = i;

break;

}

}

switch (r)

{

case LRU:

//选择计数器最小的淘汰

for (; i < pt.length; i++)

{

if (pt.pages[i].state == 1 && (pt.pages[i].visited < pt.pages[t].visited))

t = i;

}

break;

case FIFO:

//选择最先进入主存的

for (; i < pt.length; i++)

{

if (pt.pages[i].state == 1 && (pt.pages[i].time < pt.pages[t].time))

t = i;

}

break;

case OPT:

//选择val最大的淘汰

for (int j = l - 1; j >= 0; j++)

{

if (ra[j] == t)

maxval = j;

}

for (; i < pt.length; i++)

{

if (maxval == l)

{

t = i;

break;

}

int val = l;

for (int j = l - 1; j >= 0; j++)

{

if (ra[j] == i)

val = j;

}

if (val > maxval)

{

t = i;

maxval = val;

}

}

break;

default:

break;

}

cout << "内存已满，置换页面" << t << endl;

pt.pages[t].state = 0;

pt.pages[id].chunkID = pt.pages[t].chunkID;

}

pt.pages[id].state = 1;

pt.pages[id].time = getTime();

pt.pages[id].visited = 1;

pt.pages[id].changed = 0;

}

void visit(PageTable &pt, int id, int l, int \*ra)

{

if (!pt.pages[id].state)

{

cout << "页面" << id << "不在内存中，从外存中调入" << endl;

load(pt, id, l, ra);

}

pt.pages[id].visited++;

pt.pages[id].changed = random(0, 2);

}

bool isFull()

{

if (getChunkID() == -1)

{

cout << "内存已满，无法创建进程。" << endl;

return 1;

}

return 0;

}

//创建

void create(int id, int size, PageTable &pt)

{

pt.length = ceil(size / 4.0);

pt.progessID = id;

pt.pages = new Block[pt.length];

for (int i = 0; i < pt.length; i++)

{

pt.pages[i].pageID = i;

if (random(0, 2))

load(pt, i, l[id], ra[id]);

}

}

void print(PageTable pt)

{

cout << "====================进程名：" << pt.progessID << "====================" << endl;

cout << "页面号"

<< "\t状态位"

<< "\t页块号"

<< "\t访问位"

<< "\t修改位"

<< "\t载入时刻" << endl;

for (int i = 0; i < pt.length; i++)

{

if (pt.pages[i].state)

cout << pt.pages[i].pageID << "\t" << pt.pages[i].state << "\t" << pt.pages[i].chunkID << "\t" << pt.pages[i].visited << "\t" << pt.pages[i].changed << "\t" << pt.pages[i].time << endl;

else

cout << pt.pages[i].pageID << "\t" << pt.pages[i].state << "\t-\t-\t-\t-" << endl;

}

cout << endl;

}

//生成随即访问序列

void randomAccess(int t)

{

l[t] = random(1, 10);

for (int i = 0; i < l[t]; i++)

ra[t][i] = random(0, pagetable[t].length);

}

void \*access\_fun(void \*arg)

{

long in = (long)arg;

for (int i = 0; i < l[in]; i++)

{

sleep(5);

pthread\_mutex\_lock(&rw);

cout << "进程" << in << "正在访问页面" << ra[in][i] << endl;

visit(pagetable[in], ra[in][i], l[in], ra[in]);

print(pagetable[in]);

pthread\_mutex\_unlock(&rw);

}

return NULL;

}

int main()

{

srand((unsigned)time(0));

timeval tv;

gettimeofday(&tv, NULL);

start = tv.tv\_sec \* 1000 \* 1000 + tv.tv\_usec;

pthread\_t pro[maxm];

cout << "1.LUR" << endl;

cout << "2.FIFO" << endl;

cout << "3.OPT" << endl;

cout << "请选择置换方式：";

cin >> r;

if (r != 1 && r != 2 && r != 3)

cout << "输入错误！" << endl;

pthread\_mutex\_init(&rw, NULL);

int size, option;

int t = 0;

while (t < maxm && !isFull())

{

cout << "1.创建进程" << endl;

cout << "2.结束" << endl;

cout << "请选择要进行的操作：";

cin >> option;

if (option == 2)

break;

else if (option == 1)

{

cout << "请输入要创建进程的大小：";

cin >> size;

create(t, size, pagetable[t]);

print(pagetable[t]);

randomAccess(t);

cout << "进程" << t << "访问序列：";

for (int i = 0; i < l[t] - 1; i++)

cout << ra[t][i] << ", ";

cout << ra[t][l[t] - 1] << endl;

cout << endl;

t++;

}

else

cout << "输入错误！" << endl;

}

for (long i = 0; i < t; i++)

{

int ret = pthread\_create(&pro[i], NULL, access\_fun, (void \*)i);

if (ret != 0)

{

printf("error");

exit(0);

}

}

for (int i = 0; i < t; i++)

{

pthread\_join(pro[i], NULL);

}

}