

**实 验 报 告**

**（ 2019 / 2020 学年 第 1 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 操作系统B | | | | | |
| 实验名称 | 页面调度算法模拟 | | | | | |
| 实验时间 | 2020 | 年 | 05 | 月 | 20 | 日 |
| 指导单位 | 计算机学院计算机科学与技术系 | | | | | |
| 指导教师 | 段卫华 | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学生姓名 | | 刘炎 | | | 班级学号 | | B18031801 | |
| 学院(系) | | 计软网安学院 | | | 专 业 | | 软件工程嵌入式培养 | |
| **实验名称** | Linux系统及进程创建 | | | | | **指导教师** | | 段卫华 |
| **实验类型** | 验证 | | **实验学时** | 2 | | **实验时间** | | 2020.5.20 |
| 1. **实验目的和要求**   实验目的  页面调度算法主要有：FIFO，最近最少使用调度算法（LRU），最佳算法（OPT）。设计并编写程序模拟以上算法。  实验内容  1. 掌握 FIFO，LRU 和 OPT 三种算法的原理。  2. 选择合适的存储结构，分别实现这三种算法。给定一个页面请求序列，给定系统所分配 的物理块数后，给出分别采用这三种算法进行调度时产生的缺页次数以及缺页率。 | | | | | | | | |
| 二、**实验环境(实验设备)**  Windows10系统 + vs2010 | | | | | | | | |
| **三、实验原理及内容**   1. **FIFO算法**    1. **FIFO实现原理**   先进先出调度思想，在调页过程中需要遍历队列检查该页是否已存在。   * 1. **FIFO实现代码**   /\*  页面置换算法FIFO，先进先出  从外部的 txt 文件中读取请求序列  计算缺页次数和缺页率  \*/  #include<iostream>  #include<vector>  #include<algorithm>  #define MLim 50 //设置内存中页表长度上限  #define Mnum 100 //设置所要访问的页面的最大个数  using namespace std;  int sum = 0; //缺页次数  int t = -1; //顺序  class FIFO{  public:  FIFO(int page, int visit[], int nv):v(visit,visit + nv), time(nv,1000){  plen =page;  }    int isInPage(int x); // 所要访问的页面x是否在内存中  void PageFault(); // 缺页中断  private:  vector<int> v; // 所要访问的页面  vector<int> p; // 内存中页表  vector<int> time; // 内存中页面进入的时间，越小进入的越早  int plen; // 内存中页表长度  };  int FIFO::isInPage(int x){  vector<int>::iterator it = find(p.begin(), p.end(), x); //返回查找的地址  if(it != p.end()){  return \*it;  }  else{  return -1; //不在页表中  }  }  void FIFO::PageFault(){  int i,j;  for(i = 0; i < v.size(); i++){  if(isInPage(v[i]) == -1){  sum++;  if(p.size() < plen){ // 不在内存且页表有空  p.push\_back(v[i]);  time[v[i]] = ++t; //time下标为所要访问的按顺序的页面标号  }  else{ // 不在内存且页表满  vector<int>::iterator minvalue = min\_element(time.begin(), time.end());  // 求最小的时间值  for(j = 0; j < time.size(); j++){  if(time[j] == \*minvalue){  break;  }  } //求最小时间值对应的time下标，即为需要被替换的页面标号  vector<int>::iterator fir = find(p.begin(), p.end(), j);  \*fir = v[i]; // 替换页面标号  \*minvalue = 1000;  time[v[i]] = ++t;  }  }  }  cout << "缺页次数为：" << sum <<endl;  cout << "缺页率为：" << sum\*100.0/v.size() <<"%";  }  int main(){  int visit[12] = {2,3,2,1,5,2,4,5,3,2,5,2};  int page = 3;    FIFO f(3, visit, 12);  f.PageFault();  return 0;  }     * 1. **FIFO代码结果** | | | | | | | | |

1. **LRU算法**

**2.1 LRU实现原理**

最近最久未使用页面置换算法（least recently used）

**2.2 LRU实现代码**

/\*

缺页中断

最近最久未使用置换算法

\*/

#include<iostream>

#include<vector>

#include<algorithm>

#define MLim 50 //设置内存中页表长度上限

#define Mnum 100 //设置所要访问的页面的最大个数

using namespace std;

int sum = 0; //缺页次数

int t = -1; //顺序

class FIFO{

public:

FIFO(int page, int visit[], int nv):v(visit,visit + nv), time(nv,1000){

plen =page;

}

int isInPage(int x); // 所要访问的页面x是否在内存中

void PageFault(); // 缺页中断

private:

vector<int> v; // 所要访问的页面

vector<int> p; // 内存中页表

vector<int> time; // 内存中页面进入的时间，越小进入的越早

int plen; // 内存中页表长度

};

int FIFO::isInPage(int x){

vector<int>::iterator it = find(p.begin(), p.end(), x); //返回查找的地址

if(it != p.end()){

return \*it;

}

else{

return -1; //不在页表中

}

}

void FIFO::PageFault(){

int i,j;

for(i = 0; i < v.size(); i++){

if(isInPage(v[i]) == -1){

sum++;

if(p.size() < plen){ // 不在内存且页表有空

p.push\_back(v[i]);

time[v[i]] = ++t; //time下标为所要访问的按顺序的页面标号

}

else{ // 不在内存且页表满

vector<int>::iterator minvalue = min\_element(time.begin(), time.end());

// 求最小的时间值

for(j = 0; j < time.size(); j++){

if(time[j] == \*minvalue){

break;

}

} //求最小时间值对应的time下标，即为需要被替换的页面标号

vector<int>::iterator fir = find(p.begin(), p.end(), j);

\*fir = v[i]; // 替换页面标号

\*minvalue = 1000;

time[v[i]] = ++t;

}

}

else{ //在内存中

time[v[i]] = ++t;

}

}

cout << "缺页次数为：" << sum <<endl;

cout << "缺页率为：" << sum\*100.0/v.size() <<"%";

}

int main(){

int visit[12] = {2,3,2,1,5,2,4,5,3,2,5,2};

int page = 3;

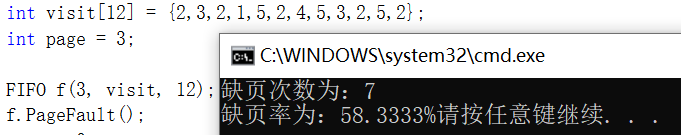
FIFO f(3, visit, 12);

f.PageFault();

return 0;

}

**2.3 LRU代码结果**



1. **OPT算法**

**3.1 OPT实现原理**

置换以后不再被访问，或者在将来最迟才回被访问的页面，缺页中断率最低

**3.2 OPT实现代码**

/\*

Optimal最佳置换算法（理论）

置换以后不再被访问，或者在将来最迟才回被访问的页面，缺页中断率最低

\*/

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int sum = 0; //缺页次数

class OPT{

public:

OPT(int page, int visit[], int nv):v(visit, visit+nv){

plen = page;

}

int isInPage( int); //是否在内存页表中，在返回页表中的页号,否则-1

void PageFault(); //缺页中断

private:

vector<int>p; // 内存中的页表

vector<int>v; // 所要访问的页面

int plen; // 内存中页表长度

};

int OPT::isInPage(int x){

vector<int>::iterator it = find(p.begin(), p.end(), x);

if(it != p.end()){

return \*it;

}

else{

return -1;

}

}

void OPT::PageFault(){

int i, j;

for(i = 0; i < v.size(); i++){

if(isInPage(v[i]) == -1){ //不在页表中

sum++;

if(p.size() < plen){ //页表为空

p.push\_back(v[i]);

}

else{ //页表满

int index; //被置换的页号

vector<int>::iterator far = v.begin()+i;

for(j = 0; j < p.size(); j++){

vector<int>::iterator temp = find(v.begin()+i, v.end(), p[j]);

//从下一个开始遍历，查找最远访问的页面

if(far < temp ){

far = temp;

index = j;

}

}

p[index] = v[i];

}

}

}

cout << "缺页次数为：" << sum <<endl;

cout << "缺页率为：" << sum\*100.0/v.size() <<"%";

}

int main(){

int visit[12] = {2,3,2,1,5,2,4,5,3,2,5,2};

int page = 3;

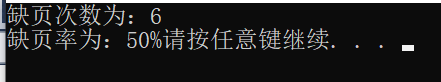
OPT f(3, visit, 12);

f.PageFault();

return 0;

}

**3.3 OPT代码结果**

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  这是操作系统B的第三个实验，通过实验模拟页面置换算法，从而更加熟悉了页面置换的过程和缺页率的计算。同时对于vector使用更加了解。  **问题及其解决：**  **1、**错误：  C++中 类成员函数声明中，vector成员的初始化方式出现问题  修改：  初始化列表的方式在构造函数定义时给成员变量赋值。  FIFO(int page, int visit[], int nv):p(page),v(visit,visit + nv){}  2、错误：    vector<int>::iterator start = v.begin() + 1;  //从下一个开始遍历，查找最远访问的页面  vector<int>::iterator far = find(start+i, v.end(), j);  原因：  Start+i 越界了。  意见与建议： | | | | | |
| **五、指导教师评语** | | | | | |
| **成 绩** |  | **批阅人** |  | **日 期** | 2019.09.19 |