北京郵電大學

Beijing University of Posts and Telecommunications

操作系统
实验报告

[内存管理实验]

学院：计算机学院
2020211376 马天成
2022 年 12 月 3 日

北京邮电大学《操作系统》课程实验报告

<table>
<tr><td>实验
名称</td><td colspan="2">内存管理实验</td><td>学
院</td><td>计算机</td><td>指导教师</td><td>李文生</td></tr>
<tr><td>班 级</td><td>班内序
号</td><td colspan="2">学 号</td><td>学生姓名</td><td colspan="2">成绩</td></tr>
<tr><td>2020211305</td><td>2</td><td colspan="2">2020211376</td><td>马天成</td><td colspan="2"></td></tr>
<tr><td>实
验
内
容</td><td colspan="6"></td></tr>
<tr><td>学生实验报
告</td><td colspan="6">(详见“实验报告和源程序”册)</td></tr>
<tr><td>课
程
设
计
成
绩
评
定</td><td colspan="6">评语:

成绩:

指导教师签名:

年 月 日</td></tr>
</table>

注：评语要体现每个学生的工作情况，可以加页。

目录

1. 实验目的和要求

1. 生成内存访问串

首先用 srand()和 rand()函数定义和产生指令地址序列，然后将指令地址序列 变换成相应的页地址流。 比如：通过随机数产生一个内存地址，共 100 个地址，地址按下述原则生成：

（1）70%的指令是顺序执行的

（2）10%的指令是均匀分布在前地址部分

（3）20%的指令是均匀分布在后地址部分

具体的实施方法是：

（a）从地址 0 开始;

（b）若当前指令地址为 m，按上面的概率确定要执行的下一条指令地址，分别为顺序、在前和在后： 顺序执行： 地址为 m+1 的指令; 在前地址：[0，m-1]中依前面说明的概率随机选取地址; 在后地址：[m+1，99]中依前面说明的概率随机选取地址;

（c）重复（b）直至生成 100 个指令地址。 假设每个页面可以存放 10（可以自己定义）条指令，将指令地址映射到页 面，生成内存访问串。

2. 设计算法，计算访问缺页率并对算法的性能加以比较

（1）最优置换算法（Optimal）

（2）最近最少使用（Least Recently Used）

（3）先进先出法（Fisrt In First Out）

其中，缺页率= 页面失效次数/ 页地址流长度 要求： 分析在同样的内存访问串上执行，分配的物理内存块数量和缺页率之间的关 系; 并在同样情况下，对不同置换算法的缺页率比较

2. 实验内容

2.1 准备工作

系统是 windows2021a
开发工具 VS2022

2.2 生成地址序列

```
// generate address depend on m
int generateAddress(int m) {
    int tmp = rand() % 10;
    // +1
    if (tmp < 7) {
        if (m == 99)
            return 99;
        return m + 1;
    }
    // 后地址
    else if (tmp == 9) {
        if (m == 99)
            return 99;
        return (rand() % (99 - m)) + m + 1;
    }
    // 前地址
    else {
        if (m == 0)
            return 0;
        return rand() % m;
    }
}
```

通过生成随机的 0-9.然后进行判断返回值。

1. 如果随机数为 0-6
   m 为 99 则返回 99
2. 如果随机数为 78
   m 为 99 返回 99
   m 小于 99 返回后地址
3. 如果随机数为 9
   m 为 0 返回 0
   m 大于 0 返回前地址

2.3 三种替换算法设计

2.3.1 Optimal

最佳(Optimal)置换算法是指，其所选择的被淘汰页面，将是以后永不使用的，或许是在最长(未来)时间内不再被访问的页面。采用最佳置换算法，通常可保证获得最低的缺页率。但由于人们目前还无法预知一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但可以利用该算法去评价其它算法。

```cpp
void Optimal(int cmds[], int pageSize) {
    set<int> page;
    int calls = 0;
    bool full = false;
    for (int i = 0; i < 100;i++) {
        int val = cmds[i];
        // not full
        if (!full) {
            if (page.find(val) == page.end()) {
                page.insert(val);
                calls++;
            }
            if (page.size() == pageSize)
                full = true;
        }
        // full
        else {
            if (page.find(val) == page.end()) {
                set<int> dismatch = page;
                for (int j = i+1; j < 100; j++) {
                    auto it = dismatch.find(cmds[j]);
                    if (it != dismatch.end())
                        dismatch.erase(cmds[j]);
                    if (dismatch.size() == 1)
                        break;
                }
                page.erase(*dismatch.begin());
                page.insert(val);
                calls++;
            }
        }
    }
    cout <<"Missing page rate: "<<calls<<"%\n";
}
```

这里就是向后看。然后直接往后找所有的东西，将页面最迟用到的替换掉。

2.3.2 LRU

LRU 是 Least Recently Used 的缩写，即最近最少使用，是一种常用的页面置换算法，选择最近最久未使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间 t，当须淘汰一个页面时，选择现有页面中其 t 值最大的，即最近最少使用的页面予以淘汰。

```
97  void LRU(int cmds[], int pageSize) {
98      set<int> page;
99      int calls = 0;
100     bool full = false;
101     for (int i=0; i < 100; i++) {
102         int val = cmds[i];
103         // not full
104         if (!full) {
105             if (page.find(val) == page.end()) {
106                 page.insert(val);
107                 calls++;
108             }
109             if (page.size() == pageSize)
110                 full = true;
111         }
112         else {
113             if (page.find(val) == page.end()) {
114                 set<int> dismatch = page;
115                 for (int j=i-1; j >=0; j--) {
116                     auto it = dismatch.find(cmds[j]);
117                     if (it != dismatch.end())
118                         dismatch.erase(cmds[j]);
119                     if (dismatch.size() == 1)
120                         break;
121                 }
122                 page.erase(*dismatch.begin());
123                 page.insert(val);
124                 calls++;
125             }
126         }
127     }
128     cout << "Missing page rate: " << calls << "%\n";
129 }
```

2.3.3 FIFO

FIFO 是英文 First In First Out 的缩写，是一种先进先出的数据缓存器，他与普通存储器的区别是没有外部读写地址线，这样使用起来非常简单，但缺点就是只能顺序写入数据，顺序的读出数据，其数据地址由内部读写指针自动加 1 完成，不能像普通存储器那样可以由地址线决定读取或写入某个指定的地址。

```
131 void FIFO(int cmds[], int pageSize) {
132     vector<int> page;
133     int calls = 0;
134     bool full = false;
135     for (int i = 0;i < 100;i++) {
136         int val = cmds[i];
137         // not full
138         if (!full) {
139             if (find(page.begin(), page.end(),val) == page.end()) {
140                 page.push_back(val);
141                 calls++;
142             }
143             if (page.size() == pageSize)
144                 full = true;
145         }
146         else {
147             if (find(page.begin(), page.end(), val) == page.end()) {
148                 page.erase(page.begin());
149                 page.push_back(val);
150                 calls++;
151             }
152         }
153     }
154     cout << "Missing page rate: " << calls << "%\n";
155 }
```

2.4 实验分析

5&10

```
Please input the page size.
5
This is cmds:
 1 0 1 46 10 11 12 13 14 15 1 2 3 4 0 1 2 3 4 5 6 7 8 7 5 6 3 4 3 4 5 6 7 8 9 10 11 12 13 14 11 12 13 4 83 84 98 99 99 9
9 99 10 11 12 13 3 4 5 30 31 32 33 34 11 12 13 14 15 16 17 18 19 15 30 31 24 18 36 37 38 51 52 53 27 28 29 30 31 32 33 3
4 97 98 48 49 50 51 52 53 17

This is Optimal:
Missing page rate: 63%
This is Least Recently Used:
Missing page rate: 80%
This is Fisrt In First Out:
Missing page rate: 82%

Please input the page size.
10
This is cmds:
 1 2 3 4 5 6 7 8 9 5 2 1 2 3 4 9 2 0 1 2 3 4 0 1 2 3 4 79 60 61 62 63 64 65 66 67 68 69 70 71 53 51 52 53 54 58 56 39 23
 24 19 8 9 7 8 72 73 74 75 83 84 63 64 20 79 80 81 82 83 84 85 74 75 0 0 0 1 2 1 24 32 33 34 18 5 6 2 3 4 0 1 2 3 4 5 1
2 3 4 5

This is Optimal:
Missing page rate: 53%
This is Least Recently Used:
Missing page rate: 66%
This is Fisrt In First Out:
Missing page rate: 67%
```

10&20

```
Please input the page size.
10
This is cmds:
 1 2 3 4 5 6 7 8 9 5 2 1 2 3 4 9 2 0 1 2 3 4 0 1 2 3 4 79 60 61 62 63 64 65 66 67 68 69 70 71 53 51 52 53 54 58 56 39 23
 24 19 8 9 7 8 72 73 74 75 83 84 63 64 20 79 80 81 82 83 84 85 74 75 0 0 0 1 2 1 24 32 33 34 18 5 6 2 3 4 0 1 2 3 4 5 1
2 3 4 5

This is Optimal:
Missing page rate: 53%
This is Least Recently Used:
Missing page rate: 66%
This is Fisrt In First Out:
Missing page rate: 67%

Please input the page size.
20
This is cmds:
 46 47 48 49 32 33 34 82 83 84 90 91 92 93 94 96 57 67 68 69 70 71 72 34 25 26 27 94 95 96 98 99 99 99 99 99 99 99 99 99
 90 94 96 49 87 88 95 96 64 65 66 46 47 48 49 50 51 52 53 54 55 56 57 58 59 3 4 5 6 7 0 1 2 3 4 5 99 99 99 99 99 99 99 9
9 99 99 99 99 99 99 99 99 99 99 99 99 59 52 7 8 9

This is Optimal:
Missing page rate: 53%
This is Least Recently Used:
Missing page rate: 59%
This is Fisrt In First Out:
Missing page rate: 59%
```

30&40

```
Please input the page size.
30
This is cmds:
 0 1 2 3 4 5 6 29 30 7 1 2 3 4 0 1 2 3 4 2 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 79 80 81 82 83 84
 85 86 87 88 69 70 71 72 73 74 75 76 77 45 46 47 33 34 35 9 32 33 7 8 9 4 5 6 1 0 62 63 64 9 10 11 8 7 8 9 10 11 12 1 2
3 4 5 6 7 8 9 10 11

This is Optimal:
Missing page rate: 48%
This is Least Recently Used:
Missing page rate: 56%
This is Fisrt In First Out:
Missing page rate: 56%

Please input the page size.
40
This is cmds:
 1 0 1 2 3 1 2 3 4 30 31 1 2 1 2 1 2 3 2 0 1 0 1 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 6 3 4 11 12 13 11 7 8 9 10 32 33
56 57 58 59 60 61 62 63 64 33 34 35 33 34 35 28 29 19 20 21 22 23 24 75 76 77 78 79 80 81 82 57 58 59 60 14 15 16 17 18
19 16 17 18 19 20 21 22

This is Optimal:
Missing page rate: 50%
This is Least Recently Used:
Missing page rate: 52%
This is Fisrt In First Out:
Missing page rate: 50%
```

50&60

```
Please input the page size.
50
This is cmds:
 1 2 0 0 1 2 3 4 5 6 7 8 2 0 1 2 69 70 55 56 13 14 15 16 17 18 19 20 15 36 37 38 39 40 98 88 37 38 39 32 33 56 57 58 59
37 38 39 40 41 42 43 82 65 66 67 68 14 5 6 7 8 9 10 11 12 13 22 23 24 88 89 90 52 53 54 55 95 96 15 16 11 12 13 14 15 16
 17 1 2 20 86 87 88 89 16 17 18 19 20

This is Optimal:
Missing page rate: 57%
This is Least Recently Used:
Missing page rate: 61%
This is Fisrt In First Out:
Missing page rate: 59%

Please input the page size.
60
This is cmds:
 1 2 0 1 2 3 4 0 57 58 25 13 14 9 10 64 14 0 1 0 0 1 2 42 43 44 7 8 9 10 11 12 1 54 51 53 54 55 56 22 23 24 25 26 33 34
35 36 37 38 39 40 41 21 22 23 24 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 14 15 16 17 18 19 20 21 7 8 9 10 11 12 13
14 15 2 3 4 5 6 7 8 4 1 20 21

This is Optimal:
Missing page rate: 51%
This is Least Recently Used:
Missing page rate: 51%
This is Fisrt In First Out:
Missing page rate: 51%

Please input the page size.
```

- 缺页率与内存大小

  非常显然，内存空间增大 2，能存的更多，命中率更好，缺页率更低。

- 缺页率比较

  在前面参考也说了，最佳的无法实现，只能作为评判标准来进行算法的优劣评判。在内存空间很小的时候，后面两种算法显然比最好的差很多。但是当内存空间达到三十多四十多，其实差距就不大了。一般在五十六十的时候三者一样。而在此之前，后两种算法不分伯仲。