实验 5. 请求页式存储管理的模拟

一、实验内容

熟悉虚拟存管理的各种页面置换算法, 并编写模拟程序实现请求页式存储管理的页面置换算法----FIFO, 测试分配不同数目物理页面时的缺页率, 并绘制“物理页面/缺页率”曲线图, 圈出工作点（Operating point）。

二、FIFO 算法伪代码描述

初始化:

```
    pageframe[total_pf]    // 创建一个大小为 total_pf 的页框数组, 存储
```

当前加载的页面

```
    rpoint <- 0           // 设置替换指针，指向要替换的页面的位置

    diseffect <- 0        // 初始化缺页次数为 0
```

对于 pageframe 中的每个位置 i:

```
    pageframe[i] <- -1    // 初始化页框数组，用-1 表示空
```

对于页面访问序列中的每个页面 page:

```
    如果 page 存在于 pageframe 中:

        // 页面已在内存中，不需要替换

        继续到下一个页面

    否则:

        // 页面不在内存中，发生缺页

        diseffect <- diseffect + 1  // 缺页次数加 1

        替换掉 pageframe 中位置为 rpoint 的页面

        pageframe[rpoint] <- page   // 将新页面加载到页框中

        rpoint <- (rpoint + 1) % total_pf // 移动替换指针到下一个位
```

置

计算总缺页率:

page_fault_rate <- diseffect /（页面访问序列的长度）

输出总缺页率

三、代码实现

全局变量和数据结构

```
struct PageInfo {
    vector<int> serial;  // 模拟的访问页面序列
    int diseffect;       // 缺页次数
    int total_pf;        // 分配的页框数
} pf_info;

vector<int> pageframe;  // 分配的页框
```

主函数

```
int main() {
    createps();
    cout << "Please input physical page number: ";
    cin >> pf_info.total_pf;
    cin.ignore();
    FIFO();
}
```

获得页面序列

```
void createps() {
    string line;
    cout << "请输入页面访问序列，以空格分隔，按回车结束：" << endl;
    getline(cin, line);
    stringstream ss(line);
    int page;
    while (ss >> page) {
        pf_info.serial.push_back(page);
    }
}
```

寻找页面

```
int findpage(int page) {
    for (size_t i = 0; i < pageframe.size(); i++) {
```

```
        if (pageframe[i] == page) {
            return 1;  // 页面存在于内存中
        }
    }
    return 0;  // 页面不存在，缺页
}
```

展示内存状态和缺页信息

```
void displayinfo(int id, int page, int pstate) {
    cout << "ID: " << setw(2) << id << " 访问 " << setw(2) << page
         << ". 内存状态:<";
    for (size_t i = 0; i < pageframe.size(); i++) {
        if (pageframe[i] != -1) {
            cout << setw(2) << pageframe[i];
        } else {
            cout << "  ";  // 空白占位
        }
        if (i < pageframe.size() - 1) {
            cout << ",";
        }
    }
    cout << ">";
    if (pstate == 0) {
        cout << " 缺页,";
    }
    cout << " 缺页率=" << fixed << setprecision(1)
         << (static_cast<double>(pf_info.diseffect) / (id + 1)) *
100.0 << "%"
         << endl;
}
```

FIFO 算法

```
void FIFO() {
    pf_info.diseffect = 0;
    pageframe.assign(pf_info.total_pf, -1);
    int rpoint = 0;

    cout << " ---FIFO Algorithm: " << pf_info.total_pf << "
frames---" << endl;

    for (size_t i = 0; i < pf_info.serial.size(); i++) {
        int page = pf_info.serial[i];
        int pstate = findpage(page);
```

```
        if (pstate == 0) {  // 若页不存在，则需装入页面
            pageframe[rpoint % pf_info.total_pf] = page;
            rpoint++;
            pf_info.diseffect++;  // 缺页次数加 1
        }

        displayinfo(i, page, pstate);  // 显示当前状态
    }

    cout << "总缺页数=" << pf_info.diseffect << "，总缺页率=" << fixed
         << setprecision(1)
         << (static_cast<double>(pf_info.diseffect) /
pf_info.serial.size()) *
                100.0
         << "%" << endl;
}
```

四、运行结果

```
(base) nanmener@Haotians-MacBook-Pro output % ./"main"
请输入页面访问序列，以空格分隔，按回车结束：
1 4 7 9 0 3 3 5 2 8 1 6 5 7 2 4 6 0 9 8 2 1 7 3 6 5 4 0 8 9 1 3 5 2 6 7 4 8 0 9
Please input physical page number: 8
 ---FIFO Algorithm: 8 frames---
ID:  0 访问  1. 内存状态:< 1,  ,  ,  ,  ,  ,  ,  > 缺页，缺页率=100.0%
ID:  1 访问  4. 内存状态:< 1, 4,  ,  ,  ,  ,  ,  > 缺页，缺页率=100.0%
ID:  2 访问  7. 内存状态:< 1, 4, 7,  ,  ,  ,  ,  > 缺页，缺页率=100.0%
ID:  3 访问  9. 内存状态:< 1, 4, 7, 9,  ,  ,  ,  > 缺页，缺页率=100.0%
ID:  4 访问  0. 内存状态:< 1, 4, 7, 9, 0,  ,  ,  > 缺页，缺页率=100.0%
ID:  5 访问  3. 内存状态:< 1, 4, 7, 9, 0, 3,  ,  > 缺页，缺页率=100.0%
ID:  6 访问  3. 内存状态:< 1, 4, 7, 9, 0, 3,  ,  > 缺页率=85.7%
ID:  7 访问  5. 内存状态:< 1, 4, 7, 9, 0, 3, 5,  > 缺页，缺页率=87.5%
ID:  8 访问  2. 内存状态:< 1, 4, 7, 9, 0, 3, 5, 2> 缺页，缺页率=88.9%
ID:  9 访问  8. 内存状态:< 8, 4, 7, 9, 0, 3, 5, 2> 缺页，缺页率=90.0%
ID: 10 访问  1. 内存状态:< 8, 1, 7, 9, 0, 3, 5, 2> 缺页，缺页率=90.9%
ID: 11 访问  6. 内存状态:< 8, 1, 6, 9, 0, 3, 5, 2> 缺页，缺页率=91.7%
ID: 12 访问  5. 内存状态:< 8, 1, 6, 9, 0, 3, 5, 2> 缺页率=84.6%
ID: 13 访问  7. 内存状态:< 8, 1, 6, 7, 0, 3, 5, 2> 缺页，缺页率=85.7%
ID: 14 访问  2. 内存状态:< 8, 1, 6, 7, 0, 3, 5, 2> 缺页率=80.0%
ID: 15 访问  4. 内存状态:< 8, 1, 6, 7, 4, 3, 5, 2> 缺页，缺页率=81.2%
ID: 16 访问  6. 内存状态:< 8, 1, 6, 7, 4, 3, 5, 2> 缺页率=76.5%
ID: 17 访问  0. 内存状态:< 8, 1, 6, 7, 4, 0, 5, 2> 缺页，缺页率=77.8%
ID: 18 访问  9. 内存状态:< 8, 1, 6, 7, 4, 0, 9, 2> 缺页，缺页率=78.9%
ID: 19 访问  8. 内存状态:< 8, 1, 6, 7, 4, 0, 9, 2> 缺页率=75.0%
ID: 20 访问  2. 内存状态:< 8, 1, 6, 7, 4, 0, 9, 2> 缺页率=71.4%
```

```
ID: 22 访问  7. 内存状态:< 8, 1, 6, 7, 4, 0, 9, 2> 缺页率=65.2%
ID: 23 访问  3. 内存状态:< 8, 1, 6, 7, 4, 0, 9, 3> 缺页，缺页率=66.7%
ID: 24 访问  6. 内存状态:< 8, 1, 6, 7, 4, 0, 9, 3> 缺页率=64.0%
ID: 25 访问  5. 内存状态:< 5, 1, 6, 7, 4, 0, 9, 3> 缺页，缺页率=65.4%
ID: 26 访问  4. 内存状态:< 5, 1, 6, 7, 4, 0, 9, 3> 缺页率=63.0%
ID: 27 访问  0. 内存状态:< 5, 1, 6, 7, 4, 0, 9, 3> 缺页率=60.7%
ID: 28 访问  8. 内存状态:< 5, 8, 6, 7, 4, 0, 9, 3> 缺页，缺页率=62.1%
ID: 29 访问  9. 内存状态:< 5, 8, 6, 7, 4, 0, 9, 3> 缺页率=60.0%
ID: 30 访问  1. 内存状态:< 5, 8, 1, 7, 4, 0, 9, 3> 缺页，缺页率=61.3%
ID: 31 访问  3. 内存状态:< 5, 8, 1, 7, 4, 0, 9, 3> 缺页率=59.4%
ID: 32 访问  5. 内存状态:< 5, 8, 1, 7, 4, 0, 9, 3> 缺页率=57.6%
ID: 33 访问  2. 内存状态:< 5, 8, 1, 2, 4, 0, 9, 3> 缺页，缺页率=58.8%
ID: 34 访问  6. 内存状态:< 5, 8, 1, 2, 6, 0, 9, 3> 缺页，缺页率=60.0%
ID: 35 访问  7. 内存状态:< 5, 8, 1, 2, 6, 7, 9, 3> 缺页，缺页率=61.1%
ID: 36 访问  4. 内存状态:< 5, 8, 1, 2, 6, 7, 4, 3> 缺页，缺页率=62.2%
ID: 37 访问  8. 内存状态:< 5, 8, 1, 2, 6, 7, 4, 3> 缺页率=60.5%
ID: 38 访问  0. 内存状态:< 5, 8, 1, 2, 6, 7, 4, 0> 缺页，缺页率=61.5%
ID: 39 访问  9. 内存状态:< 9, 8, 1, 2, 6, 7, 4, 0> 缺页，缺页率=62.5%
总缺页数=25，总缺页率=62.5%
```

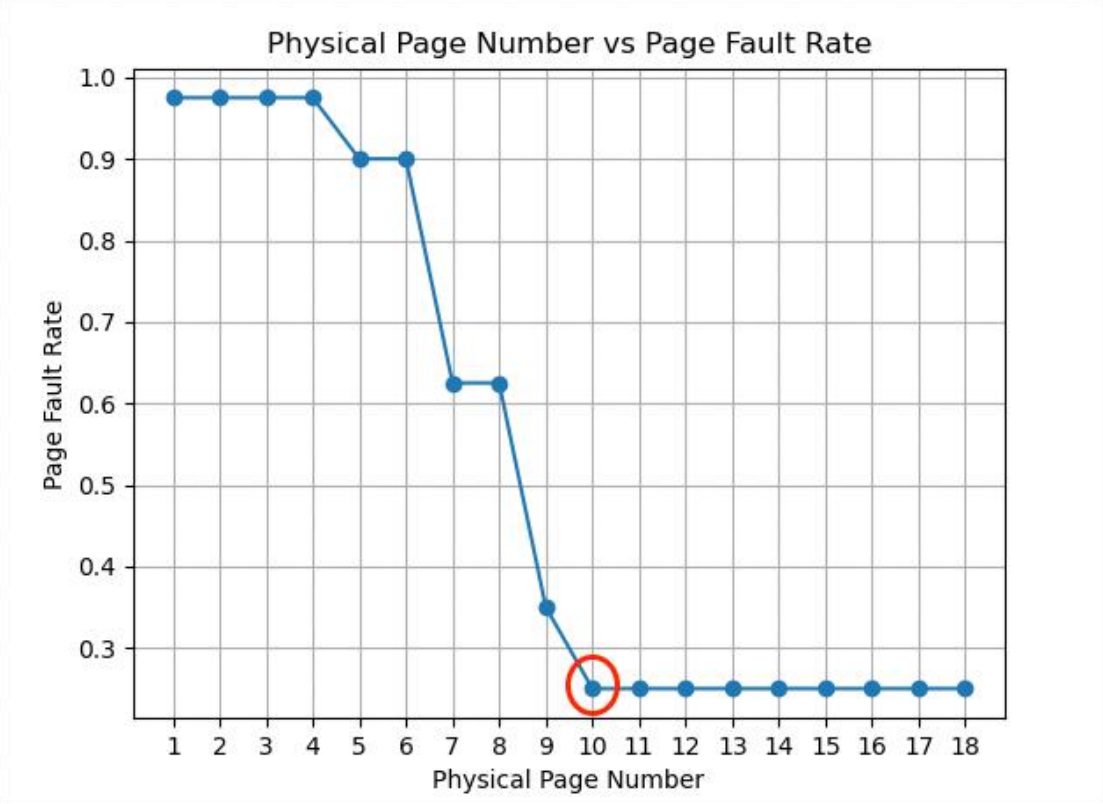
五、物理页面/缺页率 曲线图

序列 1

原始序列

```
1, 4, 7, 9, 0, 3, 3, 5, 2, 8, 1, 6, 5, 7,
2, 4, 6, 0, 9, 8, 2, 1, 7, 3, 6, 5, 4, 0,
8, 9, 1, 3, 5, 2, 6, 7, 4, 8, 0, 9
```

曲线图

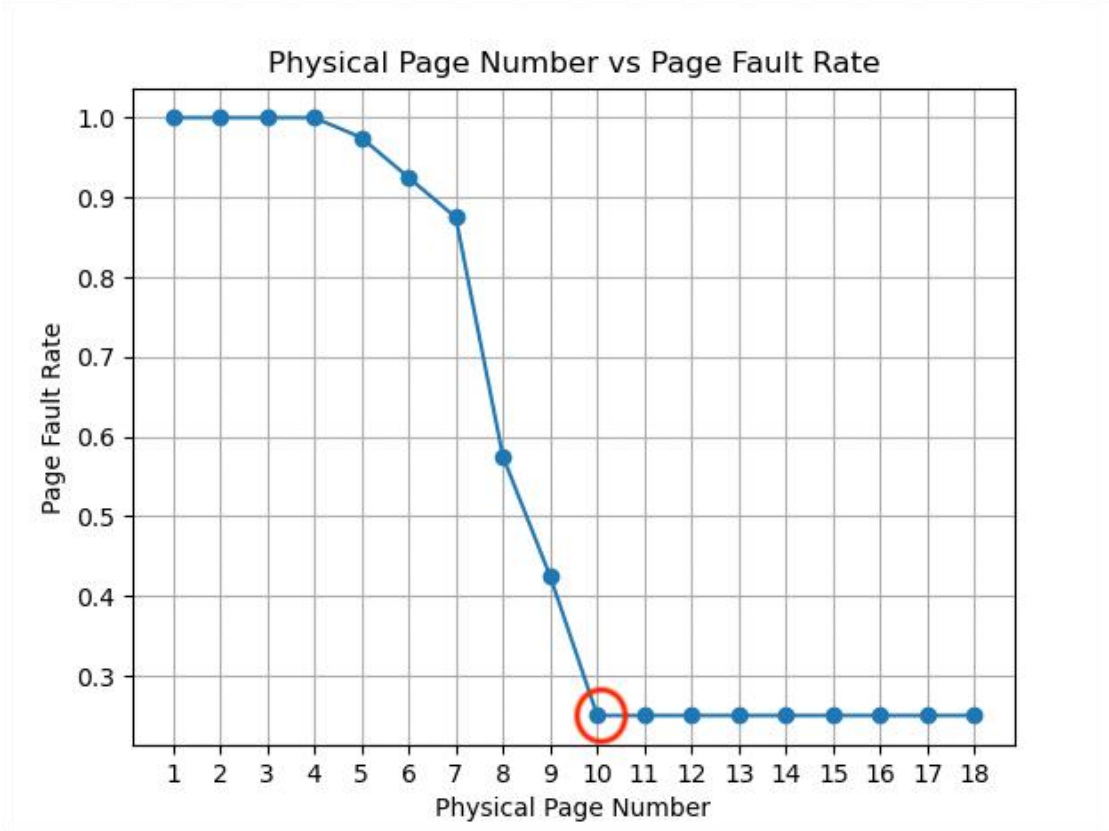


序列 2

原始序列

```
8, 2, 4, 7, 1, 0, 3, 5, 9, 6, 2, 7, 8, 4,
1, 3, 0, 5, 9, 6, 2, 7, 4, 8, 1, 0, 3, 5,
9, 6, 2, 7, 4, 8, 1, 0, 3, 5, 9, 6
```

曲线图



结论

页面置换算法对系统性能有显著影响：通过实验模拟，我们可以看到不同物理页面数下 FIFO 算法的缺页率，从而体会到页面置换算法以及物理页面分配数量对系统性能的影响。

缺页率与物理页面数的关系：实验中绘制的“物理页面/缺页率”曲线图清晰地展示了随着物理页面数的增加，缺页率逐渐降低。这符合理论预期，因为更多的物理页面意味着可以容纳更多的虚拟页面，从而减少需要替换的次数。

工作点的识别：工作点是指在缺页率曲线上，增加页框数带来的缺页率下降开始变得不明显的点。这个点可能是资源和性能之间的一个折中选择，实验帮助我们学习如何寻找和理解这一概念。