

# 实现

本次页面置换算法项目采用C++语言编写并在Visual Studio 2017上开发，图形界面基于Qt平台。本程序的基本运行机制如下：

- 首先设置相关参数，例如指令数量、页面大小等
- 然后产生指令的随机访问序列，该序列会在指令序列框中显示
- 最后可单步执行指令，也可全部执行完余下的所有指令
- 页面调度情况会在日志框中显示
- 每次重新产生指令序列都会刷新日志框
- 相应的页错误次数和页错误率会在右下角显示

# 界面

本程序界面主要分为控制部分和显示部分。控制部分用于设置调度算法运行环境的相关参数，显示部分用于显示指令访问序列以及访问每条指令时的相关日志。



## 1.控制模块

请选择置换算法

☒ FIFO ☐ LRU

320 选择指令条数

10 选择页面大小

4 选择分配的内存块数量

产生指令访问序列

单指令执行 全部执行

缺页数量: 155 缺页率: 48.4%

这里可以设置页面置换算法，指令条数，页面大小以及分配的内存块数量，以观察这些参数对缺页率的影响。

## 2.显示模块

指令序列			页面调度日志
序号	逻辑地址		
0	247	^	Current frame is 24, visiting main memory address 127, no page error
1	248		Current frame is 24, visiting main memory address 128, no page error
2	70		Current frame is 7, visiting main memory address 0, no page error
3	71		Current frame is 7, visiting main memory address 1, no page error
4	199		Current frame is 19, visiting main memory address 259, no page error
5	200		Current frame is 20, visiting main memory address 90, no page error
6	312		Current frame is 31, visiting main memory address 233, page error happend, replace frame 24 with frame 31
7	313		Current frame is 13, visiting main memory address 154, page error happend, replace frame 7 with frame 13
8	134		Current frame is 13, visiting main memory address 155, no page error
9	135		Current frame is 31, visiting main memory address 237, no page error
10	317		Current frame is 31, visiting main memory address 238, no page error
11	318		Current frame is 30, visiting main memory address 168, page error happend, replace frame 19 with frame 30
12	308		Current frame is 30, visiting main memory address 169, no page error
13	309		Current frame is 9, visiting main memory address 299, page error happend, replace frame 20 with frame 9
14	99		Current frame is 10, visiting main memory address 240, page error happend, replace frame 31 with frame 10
15	100		Current frame is 22, visiting main memory address 51, page error happend, replace frame 13 with frame 22
16	221		Current frame is 22, visiting main memory address 52, no page error
17	222		Current frame is 9, visiting main memory address 291, no page error
18	91		Current frame is 9, visiting main memory address 292, no page error
19	92		Current frame is 4, visiting main memory address 268, page error happend, replace frame 30 with frame 4
20	48		Current frame is 4, visiting main memory address 269, no page error
21	49	v	Current frame is 9, visiting main memory address 294, no page error

这里，访问每条指令时都会产生一条日志消息，以表示当前访问的逻辑块地址、主存地址、有无页错误发生以及发生页错误后采取的措施。

# 算法

本程序提供的页面置换算法有FIFO和LRU。FIFO和LRU均采用C++中的std::list数据结构实现，具体实现如下：

## 1.FIFO

```
//单步执行FIFO
void PRA::singleFIFO() {
    int currentInsAddr = insSeq[currentInsIndex];
    int pageIndex = getPageIndex(currentInsAddr, pageSize);

    if (pageTable[pageIndex].validation) { //有效则直接访问
        emit(status(getPhysicalAddr(currentInsAddr, pageTable, pageSize), false, pageError,
pageIndex));
    }
    else if (!isFull) { //内存列表没空时加到空位置
        auto emptyAddr = getFrameIter(frameQueue, -1);
        if (emptyAddr != frameQueue.end()) {
            *emptyAddr = pageIndex;
            pageTable[pageIndex].validation = true;
            pageError++;
            emit(status(getPhysicalAddr(currentInsAddr, pageTable, pageSize), false,
pageIndex, pageIndex));
        }
        else
            isFull = true;
    }
    else { //置换队列头块
        pageError++;
        int frontAddr = frameQueue.front();
        if (frontAddr >= 0 && pageTable[frontAddr].validation)
            pageTable[frontAddr].validation = false;
        frameQueue.pop_front();
        frameQueue.push_back(pageIndex);
        pageTable[pageIndex].validation = true;
        emit(status(getPhysicalAddr(currentInsAddr, pageTable, pageSize), true, pageError,
pageIndex, frontAddr));
    }
    currentInsIndex++;
}
```

在发生页错误时，FIFO总是替换队列头的块，以实现“先进先出”。

## 2.LRU

```
//单步执行LRU
void PRA::singleLRU() {
    int currentInsAddr = insSeq[currentInsIndex];
```

```

int pageIndex = getPageIndex(currentInsAddr, pageSize);

if (pageTable[pageIndex].validation) { //有效则直接访问，并将被访问的块提到队前
    cout << getPhysicalAddr(currentInsAddr, pageTable, pageSize) << endl;
    auto frameIter = getFrameIter(frameQueue, pageIndex);
    if (frameIter != frameQueue.end()) {
        frameQueue.erase(frameIter);
        frameQueue.push_front(pageIndex);
    }
    emit(status(getPhysicalAddr(currentInsAddr, pageTable, pageSize), false, pageError,
pageIndex));
}
else if (!isFull) { //内存列表没空时加到空位置
    auto emptyAddr = getFrameIter(frameQueue, -1);
    if (emptyAddr != frameQueue.end()) {
        *emptyAddr = pageIndex;
        pageTable[pageIndex].validation = true;
        pageError++;
        emit(status(getPhysicalAddr(currentInsAddr, pageTable, pageSize), false,
pageError, pageIndex));
    }
    else
        isFull = true;
}
else { //置换队列尾块
    pageError++;
    int backAddr = frameQueue.back();
    if (backAddr >= 0 && pageTable[backAddr].validation)
        pageTable[backAddr].validation = false;
    frameQueue.pop_back();
    frameQueue.push_back(pageIndex);
    pageTable[pageIndex].validation = true;
    emit(status(getPhysicalAddr(currentInsAddr, pageTable, pageSize), true, pageError,
pageIndex, backAddr));
}
currentInsIndex++;
}

```

在发生页错误时，LRU总是替换队列尾的块，也即“最近最不经常访问”块

## 性能

### 1.FIFO

- 不命中时，FIFO只需将内存列表头的块替换掉，执行两个操作。
- 命中时，FIFO直接访问，不操作。

若指令数量为n，额外时间复杂度为O(n)

### 2.LRU

- 不命中时，LRU只需将内存列表尾的块替换掉，执行两个操作。
- 命中时，LRU直接访问，并将被访问的块提到队头，这里执行查找、删除和插入三个个操作

若指令数量为n，缺页率为r，则额外时间复杂度为:

$$O(n(1 - r) + n^2r/2)$$

## 结果分析

序号	指令条数	页面大小	内存块数量	FFIFO缺页率	LRU缺页率
0	320	10	4	45.3%	47.8%
1	480	10	4	47.2%	48.1%
2	640	10	4	46.2%	47.5%
3	800	10	4	46.0%	47.5%
4	320	10	4	46.4%	46.6%
5	1120	10	4	46.1%	46.1%
6	320	15	4	40.9%	40.6%
7	320	20	4	36.8%	39.6%
8	320	25	4	32.5%	35.0%
9	320	30	4	25.9%	29.0%
10	320	35	4	22.1%	23.4%
11	320	10	8	36.5%	40.9%
12	320	10	12	29.6%	30.1%
13	320	10	16	24.6%	22.1%
14	320	10	20	17.5%	12.5%
15	320	10	24	11.2%	9.06%

### 一些结论

- 从表中可以看出，在基准情况下指令数量对缺页率几乎没有什么影响。
- 页面大小和内存块数量对缺页率有很大影响。增加页面大小或增加分配的内存块数量都能有效减少缺页率。
- 表中没有体现出LRU相对于FIFO的优越性，可能与指令访问序列有关系，在真实情况下，LRU应该比FIFO性能要好。

- 这里FIFO没有表现出Belady异常，可能与指令访问序列和内存块设置有关。