操作系统实验 5　页面置换算法

计算机 1202　张艺瀚
学号:20123852

June 23, 2015

1　实验题目

编程实现 FIFO 和 LRU 算法

2　实验目的

1. 进一步理解父子进程之间的关系
2. 理解内存页面调度的机理
3. 掌握页面置换算法的实现方法
4. 通过实验比较不同调度算法的优劣
5. 培养综合运用所学知识的能力

页面置换算法是虚拟存储管理实现的关键，通过本次试验理解内存页面调度的机制，在模拟实现 FIFO、LRU 等经典页面置换算法的基础上，比较各种置换算法的效率及优缺点，从而了解虚拟存储实现的过程。将不同的置换算法放在不同的子进程中加以模拟，培养综合运用所学知识的能力

3　实验内容及要求

1. 这是一个综合型实验，要求在掌握父子进程并发执行机制和内存页面置换算法的基础上，能综合运用这两方面的知识，自行编制程序
2. 程序涉及一个父进程和两个子进程。父进程使用 rand() 函数随机产生若干随机数，经过处理后，存于一数组 Acess_Series[] 中，作为内存页面访问的序列。两个子进程根据这个访问序列，分别采用 FIFO 和 LRU 两种不同的页面置换算法对内存页面进行调度。要求：

(a) i. 设缺页的次数为 diseffect。总的页面访问次数为 total_instruction。
ii. 缺页率 = disaffect/total_instruction
iii. 命中率 = 1- disaffect/total_instruction

(b) 将为进程分配的内存页面数 mframe 作为程序的参数，通过多次运行程序，说明 FIFO 算法存在的 Belady 现象。

4 程序流程图

见图 1

5 源程序

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <error.h>
#include <wait.h>
#include <unistd.h>

#include <algorithm>
#include <cassert>
#include <cmath>
#include <ctime>
#include <initializer_list>
#include <iostream>
#include <fstream>
#include <functional>
#include <set>
#include <sstream>
#include <vector>

using namespace std;

const int total_instruction = 10;
const int mem_frame_num = 4;
const int access_frame_num = 5;

```

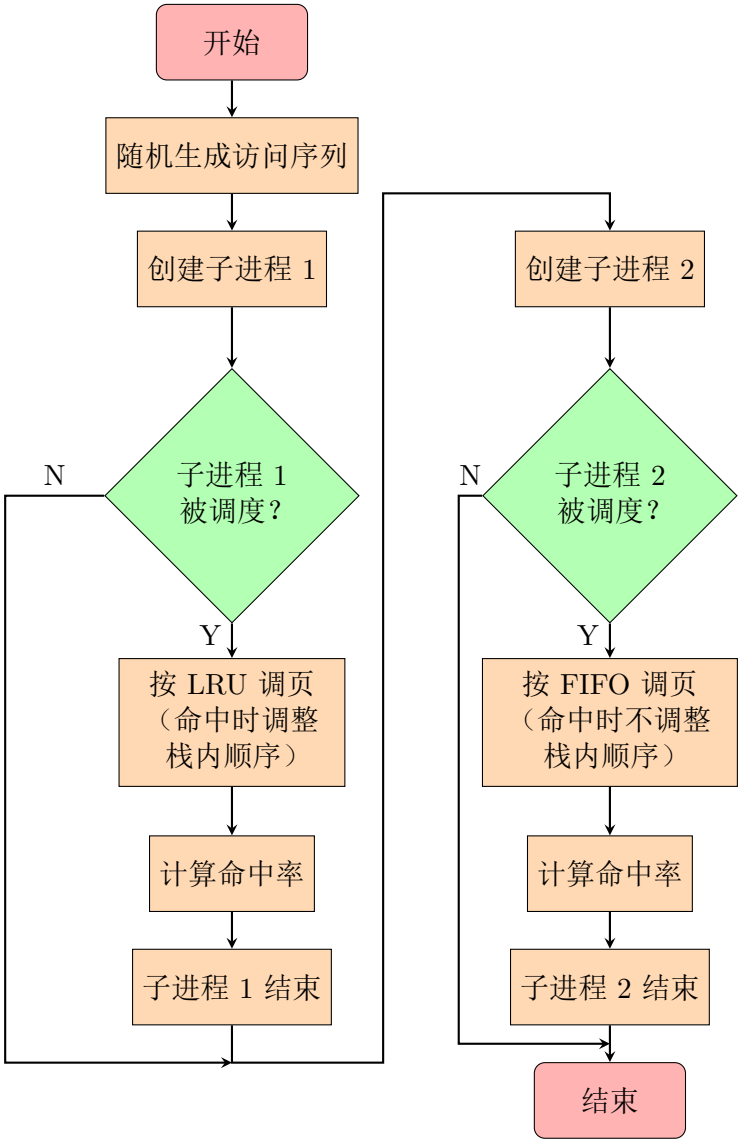
开始
随机生成访问序列
创建子进程 1
创建子进程 2
子进程 1
被调度？
子进程 2
被调度？
N
Y
N
Y
按 LRU 调页
（命中时调整
栈内顺序）
按 FIFO 调页
（命中时不调整
栈内顺序）
计算命中率
计算命中率
子进程 1 结束
子进程 2 结束
结束

Figure 1: 主过程

```
28 template<typename T>
29 void disp(const vector<T>& vec){
30   for_each(vec.begin(), vec.end(),
31     [](const T& t){
32       cout << t << " ";
33     });
34 }
35
36 int main(int argc, char** argv){
37   vector<int> access_series(total_instruction);
38   srand(time(0));
39   for(int i = 0; i < access_series.size(); ++i){
40     access_series[i] = rand() % access_frame_num + 1;
41   }
42   cout << "access_series: ";
43   disp(access_series);
44   cout << endl;
45   vector<int> fifo, lru;
46   int p1, p2;
47   while((p1 = fork()) == -1);
48   if(p1 == 0){
49     int disaffect = 0;
50     double hit = 0;
51     for(int i = 0; i < access_series.size(); ++i){
52       cout << "lru\taccess frame " << access_series[i] << ";\t";
53       auto pos = find(lru.begin(), lru.end(), access_series[i]);
54       if(pos == lru.end()){
55         if(lru.size() >= mem_frame_num){
56           lru.erase(lru.begin());
57         }
58         lru.push_back(access_series[i]);
59         ++disaffect;
60         cout << "disaffect = " << disaffect << "\t";
61         disp(lru); cout << endl;
62       }else{
63         lru.erase(pos);
64         lru.push_back(access_series[i]);
65         cout << "\t\t";
66         disp(lru);
67         cout << endl;
68       }
```

```
    }
    hit = 1 - disaffect * 1.0 / total_instruction;
    cout << "lru\thit = " << hit << endl;
    exit(0);
  }
  while((p2 = fork()) == -1);
  if(p2 == 0){
    int disaffect = 0;
    double hit = 0;
    for(int i = 0; i < access_series.size(); ++i){
      cout << "fifo\taccess frame " << access_series[i] << ";\t";
      auto pos = find(fifo.begin(), fifo.end(), access_series[i])
   ;
      if(pos == fifo.end()){
        if(fifo.size() >= mem_frame_num){
          fifo.erase(fifo.begin());
        }
        fifo.push_back(access_series[i]);
        ++disaffect;
        cout << "disaffect = " << disaffect << "\t";
        disp(fifo); cout << endl;
      }else{
        cout << "\t\t";
        disp(fifo);
        cout << endl;
      }
    }
    hit = 1 - disaffect * 1.0 / total_instruction;
    cout << "fifo\thit = " << hit << endl;
    exit(0);
  }
  return 0;
}
```

Listing 1: 代码清单

6 运行结果及其说明

访问页	LRU	命中	FIFO	命中
5	5	×	5	×
1	5 1	×	5 1	×
5	1 5	✓	5 1	✓
2	1 5 2	×	5 1 2	✓
1	5 2 1	✓	5 1 2	✓
5	2 1 5	✓	5 1 2	✓
1	2 5 1	✓	5 1 2	✓
4	2 5 1 4	×	5 1 2 4	×
1	2 5 4 1	✓	5 1 2 4	✓
4	2 5 1 4	✓	5 1 2 4	✓

7 回答问题

7.1 父进程空间与子进程空间的关系。

使用 fork 创建子进程时，逻辑上，除了 pid 和父进程的一些不允许共享的数据外，子进程的 pcb 几乎是父进程 pcb 的精确复制，它们具有相同的代码、数据、现场、资源等等；物理上，并不会为子进程另外开辟一段内存空间将父进程复制一份，而是二者通过一定的控制共享同一个进程空间。父子进程和系统中的其他进程并发执行，调度顺序由操作系统调度程序决定，不受用户干预，可以通过 fork 的返回值区分它们，子进程从 fork 语句的下一条开始指令行开始执行。

7.2 通过完成实验，根据你的体会，阐述虚拟存储器的原理。

1. 部分调入：程序装入时，不必一次性全部装入内存，而是允许部分装入就开始执行；
2. 现用现调：程序执行过程中如果访问的指令（数据）不在内存，产生缺页（段）中断，将需要的页（段）调入内存；
3. 不够换出：内存紧张时，将一些页（段）换出内存，从而装入需要调入的页（段）。

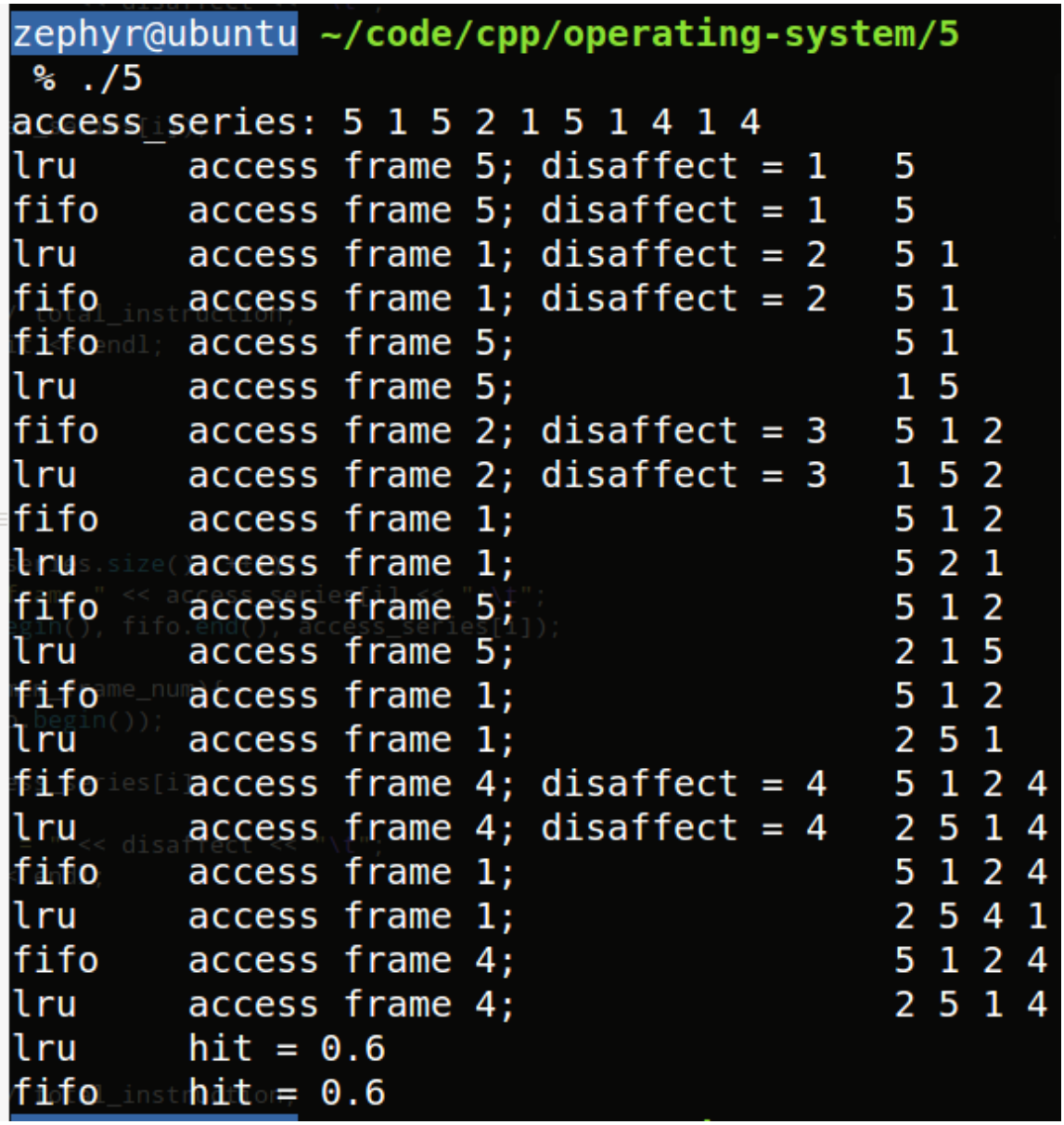

Figure 2: 运行结果

7.3 写出 FIFO 算法中出现 Belady 现象的内存页面访问序列。

访问页	FIFO1	命中	FIFO2	命中
1	1	×	1	×
2	2 1	×	2 1	×
3	3 2 1	×	3 2 1	×
4	4 3 2	×	4 3 2 1	×
1	1 4 3	×	4 3 2 1	✓
2	2 1 4	×	4 3 2 1	✓
5	5 2 1	×	5 4 3 2	×
1	5 2 1	✓	1 5 4 3	×
2	5 2 1	✓	2 1 5 4	×
3	3 5 2	×	3 2 1 5	×
4	4 3 5	×	4 3 2 1	×
5	4 3 5	✓	5 4 3 2	×