LRU实验报告

57119101 王晨阳

2021年6月8日

实验目的

通过实验，理解LRU页面置换算法的算法思想及其实现方法，比较各种实现算法的复杂度和实现难度，体会LRU算法与各种近似算法间的区别，并进而加深对虚拟内存概念的理解

实验步骤

程序

```
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
using namespace std;

int Cache_Cap; //capacity
int Number_of_Pages;
int Test_Time;

//counter
struct Unit1
{
    Unit1(int v, int c)
    {
        val = v;
        cnt = c;
    }
    int val;
    int cnt;
};

vector<Unit1> Cache1;

void Vis1(int Val, int &Pos, bool &Find)
```

```
{
    int MaxPos = -1;
    int MaxCnt = -1;
    Find = false;
    for (int i = 0; i < Cache1.size(); i++)
    {
        if (Cache1[i].cnt > MaxCnt)
        {
            MaxCnt = Cache1[i].cnt;
            MaxPos = i;
        }
        Cache1[i].cnt++;
        if (Cache1[i].val == Val)
        {
            Find = true;
            Pos = i;
            Cache1[Pos].cnt = 0;
        }
    }
    if (!Find)
    {
        if (Cache1.size() < Cache_Cap)
        {
            Cache1.push_back(Unit1(Val, 0));
            Pos = Cache1.size() - 1;
            Cache1[Pos].cnt = 0;
        }
        else
        {
            Cache1[MaxPos].val = Val;
            Cache1[MaxPos].cnt = 0;
            Pos = MaxPos;
        }
    }
}

//stack
struct Unit2
{
    Unit2(int v)
    {
        val = v;
    }
    int val;
    int pre, nxt;
};

vector<Unit2> Cache2;

int top = 0, bot = 0;

void Vis2(int Val, int &Pos, bool &Find)
{
    Find = false;
    for (int i = 0; i < Cache2.size(); i++)
    {
        if (Cache2[i].val == Val)
        {
```

```
                Find = true;
                Pos = i;
                if (bot == Pos)
                {
                    bot = Cache2[Pos].pre;
                    Cache2[top].pre = Pos;
                    Cache2[Pos].nxt = top;
                    top = Pos;
                }
                else if (top != Pos)
                {
                    Cache2[Cache2[Pos].pre].nxt = Cache2[Pos].nxt;
                    Cache2[Cache2[Pos].nxt].pre = Cache2[Pos].pre;
                    Cache2[top].pre = Pos;
                    Cache2[Pos].nxt = top;
                    top = Pos;
                }
            }
        }
        if (!Find)
        {
            if (Cache2.size() < Cache_Cap)
            {
                Cache2.push_back(Unit2(Val));
                Pos = Cache2.size() - 1;
                Cache2[top].pre = Pos;
                Cache2[Pos].nxt = top;
                top = Pos;
            }
            else
            {
                Pos = 1;
                bot = Cache2[Pos].pre;
                Cache2[top].pre = Pos;
                Cache2[Pos].nxt = top;
                top = Pos;
                Cache2[Pos].val = Val;
            }
        }
    }

    //Additional-Reference-Bits
    struct Unit3
    {
        Unit3(int v, int b)
        {
            val = v;
            bit = b;
        }
        int val;
        unsigned char bit;
    };

    vector<Unit3> Cache3;

    void Vis3(int Val, int &Pos, bool &Find)
    {
        int MinPos = -1;
```

```
    unsigned char MinBit = 0xFF;
    Find = false;
    for (int i = 0; i < Cache3.size(); i++)
    {
        Cache3[Pos].bit <<= 1;
        if (Cache3[i].val == Val)
        {
            Find = true;
            Pos = i;
            Cache3[Pos].bit += (unsigned char)(1);
        }
        else if (Cache3[i].bit < MinBit)
        {
            MinBit = Cache3[i].bit;
            MinPos = i;
        }
    }
    if (!Find)
    {
        if (Cache3.size() < Cache_Cap)
        {
            Cache3.push_back(Unit3(Val, 0xFF));
            Pos = Cache3.size() - 1;
            Cache3[Pos].bit = 0xFF;
        }
        else
        {
            Cache3[MinPos].val = Val;
            Cache3[MinPos].bit = 0xFF;
            Pos = MinPos;
        }
    }
}

//Second chance
struct Unit4
{
    Unit4(int v, int r)
    {
        val = v;
        ref = r;
    }
    int val;
    bool ref;
};

vector<Unit4> Cache4;

int ptr = 0;

void Vis4(int Val, int &Pos, bool &Find)
{
    Find = false;
    for (int i = 0; i < Cache4.size(); i++)
    {
        if (Cache4[i].val == Val)
        {
            Find = true;
```

```
            Pos = i;
            return;
        }
    }
    if (ptr >= Cache4.size())
    {
        Cache4.push_back(Unit4(Val, 1));
        ptr = (ptr + 1) % Cache_Cap;
    }
    else
    {
        while (Cache4[ptr].ref == 1)
        {
            Cache4[ptr].ref = 0;
            ptr = (ptr + 1) % Cache_Cap;
        }
        Cache4[ptr].val = Val;
        Cache4[ptr].ref = 1;
    }
}

int main()
{
    cout << "Input the capacity of the cache: ";
    cin >> Cache_Cap;
    cout << "Input the total number of different pages: ";
    cin >> Number_of_Pages;
    cout << "Input the time of tests: ";
    cin >> Test_Time;

    int WrongPage1 = 0, WrongPage2 = 0, WrongPage3 = 0, WrongPage4 = 0;

    srand(time(0));
    for (int i = 1; i <= Test_Time; i++)
    {
        int Pos;
        bool Find;
        int Num = rand() % Number_of_Pages;
        Vis1(Num, Pos, Find);
        if (!Find)
            WrongPage1++;
        Vis2(Num, Pos, Find);
        if (!Find)
            WrongPage2++;
        Vis3(Num, Pos, Find);
        if (!Find)
            WrongPage3++;
        Vis4(Num, Pos, Find);
        if (!Find)
            WrongPage4++;
    }
    cout<<endl;
    cout << "Counter Implementation:              " << WrongPage1 << " pages not found." << endl;
    cout << "Stack Implementation:                " << WrongPage2 << " pages not found." << endl;
    cout << "Additional-Reference-Bits Algorithm: " << WrongPage3 << " pages not found." << endl;
```

```
    cout << "Second chance Algorithm:            " << WrongPage4 << " pages not found." << endl;
    cout<<endl;
}
```

运行结果

EXAMPLE

```
Input the capacity of the cache: 4
Input the total number of different pages: 5
Input the time of tests: 1000000

Counter Implementation:                199380 pages not found.
Stack Implementation:                  199651 pages not found.
Additional-Reference-Bits Algorithm: 199573 pages not found.
Second chance Algorithm:               199651 pages not found.
```

多次尝试后得出结论：若 Cache 大小为 n，共有 p 个不同的页，则在 *完全随机* 的状况下命中略高于 $\frac{c}{p}$，且四种算法命中率相近。考虑到现实情况会有大量的反复使用，命中率会远高于随机状况。

算法分析

若 Cache 大小为 n，在不使用哈希表优化的情况下，四种算法的时间复杂度均为 $\mathrm{O}(n)$（因为查找复杂度为 $\mathrm{O}(n)$）。

若使用哈希表优化查找（使查找时间复杂度为 $\mathrm{O}(1)$），则算法时间复杂度如下

算法	时间复杂度
Counter Implementation	$\mathrm{O}(n)$
Stack Implementation	$\mathrm{O}(1)$
Additional-Reference-Bits Algorithm	$\mathrm{O}(n)$
Second chance Algorithm	最优状况下 $\mathrm{O}(1)$，最坏状况下 $\mathrm{O}(n)$

综上，若要求效率最高，应当使用哈希表优化的栈实现 LRU。

但考虑到硬件不足以支持较高的空间复杂度和复杂的数据结构，Counter Implementation 和 Stack Implementation 较难实现；

而 Additional-Reference-Bits Algorithm 和 Second chance Algorithm 区别不大。

实验体会

通过本次实验，掌握了LRU的四种算法及其实现，了解了四种算法的优劣，加深了对操作系统概念的理解。提高了动手能力，解决问题的能力得到强化。