实验代码：

```
#include <iostream>
#include <cstdio>
#include <time.h>
#include <cstdlib>
#include <vector>

#define MEM_PAHE_NUM 4
#define COMMD_NUM 320
#define PAGE_COMMD 10
#define MAX_FAR 1000000000

using namespace std;

struct MemoryCell
{
	int index;//页号
	int time;//时间戳
};

int commds[COMMD_NUM];	//存放的是每一条指令的页号
MemoryCell memory[MEM_PAHE_NUM];	//内存块
vector<int> order_commd;
vector<int> order_page;

void initPage();	//初始化页表
void createArray(); //生成序列
double FIFO();		//用 FIFO 置换算法
double LRU();		//用 LRU 置换算法
double OPT();		//用 OPT 置换算法

int main()
{
	initPage();
	createArray();
	double fifo = FIFO();
	double lru = LRU();
	double opt = OPT();
	printf("fifo = %f lru = %f opt = %f\n", fifo, lru, opt);
	return 0;
}

void initPage()
{
```

```
	//现在开始给每个页分配指令
	int page_index = 0;
	for (int i = 0; i < COMMD_NUM; i++)
	{
		commds[i] = page_index;
		if ((i + 1) % PAGE_COMMD == 0)
		{
			page_index++;
		}
	}

	//	for(int i=0;i<COMMD_NUM;i++)
	//	{
	//		printf("%d ",commds[i]);
	//		if((i+1) % PAGE_COMMD ==0)
	//		{
	//			printf("\n");
	//		}
	//	}
}

void createArray()
{
	srand((unsigned)time(NULL));
	int commd = 0;
	while (commd < COMMD_NUM)
	{
		//范围是[0,319]
		int m = rand() % COMMD_NUM;
		order_commd.push_back(m);
		commd++;
		if (commd >= COMMD_NUM)
			break;
		if (m == COMMD_NUM - 1)
			continue;
		order_commd.push_back(m + 1);
		commd++;
		if (commd >= COMMD_NUM)
			break;

		//范围是[0,m+1]
		m = rand() % (m + 2);
		order_commd.push_back(m);
		commd++;
```

```
            if (commd >= COMMD_NUM)
                break;
            if (m == COMMD_NUM - 1)
                continue;
            order_commd.push_back(m + 1);
            commd++;
            if (commd >= COMMD_NUM)
                break;

            //范围是[m+2,319]
            m = rand() % (COMMD_NUM - m - 2) + m + 2;
            order_commd.push_back(m);
            commd++;
            if (commd >= COMMD_NUM)
                break;
        }

        //将指令序列转化为页号序列
        for (int i = 0; i < order_commd.size(); i++)
        {
            order_page.push_back(commds[order_commd[i]]);
        }

        //      printf("order_commd_size = %d\n",order_commd.size());
        //      for(int i=0;i<order_commd.size();i++)
        //      {
        //          printf("%d ",order_commd[i]);
        //      }

        //      printf("order_page_size = %d\n",order_page.size());
        //      for(int i=0;i<order_page.size();i++)
        //      {
        //          printf("%d ",order_page[i]);
        //      }
}

double FIFO()
{
        int memory_page_num = 0;
        int err = 0;
        for (int i = 0; i < order_page.size(); i++)
        {
            bool flag = 0;
            for (int j = 0; j < memory_page_num; j++)
```

```
        {
            if (memory[j].index == order_page[i])
            {
                flag = true;
                break;
            }
        }

        if (!flag)     //没找到,发生缺页
        {
            err++;
            if (memory_page_num < MEM_PAHE_NUM)
            {
                memory[memory_page_num++].index = order_page[i];
            }
            else       //利用置换算法开始置换
            {
                for (int j = 1; j < memory_page_num; j++)
                {
                    memory[j - 1] = memory[j];//将第一个置换出去
                }
                memory[memory_page_num - 1].index = order_page[i];
            }
        }

        for (int j = 0; j < memory_page_num; j++)
        {
            printf("%d", memory[j].index);
            (memory[j].index == order_page[i]) ? printf("* ") : printf(" ");
        }
        if (!flag) printf("F");
        printf("\n");
    }
    printf("FIFO err =%d\n", err);
    return (double)err / COMMD_NUM;
}

double LRU()
{
    int memory_page_num = 0;
    int err = 0;
    for (int i = 0; i < order_page.size(); i++)
    {
        bool flag = 0;
```

```
        for (int j = 0; j < memory_page_num; j++)
        {
            if (memory[j].index == order_page[i])
            {
                flag = true;
                memory[j].time = i; //更新时间戳
                break;
            }
        }

        if (!flag)     //没找到,发生缺页
        {
            int minn = 0;//标记最小的
            err++;
            if (memory_page_num < MEM_PAHE_NUM)
            {
                memory[memory_page_num].time = i; //留下时间戳
                memory[memory_page_num++].index = order_page[i];
            }
            else       //利用置换算法开始置换
            {
                for (int j = 0; j < memory_page_num; j++)
                {
                    if (memory[minn].time > memory[j].time)
                    {
                        minn = j;
                    }
                }
                memory[minn].time = i;
                memory[minn].index = order_page[i];
            }
        }

        for (int j = 0; j < memory_page_num; j++)
        {
            printf("%d", memory[j].index);
            (memory[j].index == order_page[i]) ? printf("* ") : printf(" ");
        }
        if (!flag) printf("F");
        printf("\n");
    }
    printf("LRU err =%d\n", err);
    return (double)err / COMMD_NUM;
}
```

```
double OPT()
{
    int memory_page_num = 0;
    int err = 0;
    for (int i = 0; i < order_page.size(); i++)
    {
        bool flag = 0;
        for (int j = 0; j < memory_page_num; j++)
        {
            if (memory[j].index == order_page[i])
            {
                flag = true;
                break;
            }
        }

        if (!flag)     //没找到,发生缺页
        {
            err++;
            if (memory_page_num < MEM_PAHE_NUM)
            {
                memory[memory_page_num++].index = order_page[i];
            }
            else       //利用置换算法开始置换
            {
                int far[MEM_PAHE_NUM];//存储距离
                int furthest = 0;//找最远的位置；
                for (int j = 0; j < MEM_PAHE_NUM; j++) //初始化为最大值
                    far[j] = MAX_FAR;
                for (int j = 0; j < memory_page_num; j++)
                {
                    for (int k = i + 1; k < order_page.size(); k++)
                    {
                        if (memory[j].index == order_page[k]) //未来出现的位置
                            far[j] = k;
                    }
                }
                for (int j = 0; j < memory_page_num; j++) //找最远的位置
                    if (far[furthest] < far[j])
                        furthest = j;
                //找到之后，置换
                memory[furthest].index = order_page[i];
            }
```

```
            }

            for (int j = 0; j < memory_page_num; j++)
            {
                printf("%d", memory[j].index);
                (memory[j].index == order_page[i]) ? printf("* ") : printf(" ");
            }
            if (!flag) printf("F");
            printf("\n");
        }
        printf("OPT err =%d\n", err);
        return (double)err / COMMD_NUM;
}
```

实验结果：