参考代码三　地址转换

1.说明

本程序实现了以下地址转换模拟，即将将虚拟地址转换成物理地址：

1.可变分区存储管理技术的地址转换的模拟实现

2.分页存储管理的地址转换的模拟实现　//页内位移 12 位，设指令地址 15 位，故页号 3 位

3.分段存储管理的地址转换的模拟实现　//s,段号 3 位　;w,段内偏移 10 位;

4.段页式存储管理的地址转换的模拟实现 //缺段后没有把缺失的段调入

2. 虚拟地址的形成及地址转换

设计一个随机数发生程序。

功能：(1)根据给定的随机数取值范围产生一个随机数序列

(2)从产生的随机数序列中均匀选出 M 个随机数。

参考形式：RND（n0，n，M，RN）

随机数取值范围是 n0～n，从中选出 M 个随机数，选出的随机数存于数组 RN（1：N）中，N 应足够大。

2.1 可变分区存储管理技术的地址转换的模拟实现

重定位方式：动态重定位，即每次存储访问时进行

硬件支持：基地址寄存器 BR 与长度寄存器 LR

举例：

⑴给定 BR=1400，LR=2048；

⑵在一个取值范围为 0～2500 的随机数序列中，选出 20 个随机数作为虚地址；

⑶对于每一个虚地址，根据给定的 BR 和 LR 的值 ，计算相应的物理地址，并判断是否越界，若越界，打印出警告信息。

方法：

对于每一个虚地址，应与 LR 进行比较越界否，未越界，则物理地址=虚地址 + BR 的值 。

2.2 分页存储管理的地址转换的模拟实现

举例：

⑴设计算机系统中存储块的大小为 4K；

⑵建立两个作业的页表（用两个数组或一个数组）、作业表、主存分块表 MBT；

作业 1 页表

页号	块号
0	5
1	7

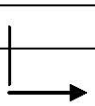
（页号可以缺省）

作业 2 页表

页号	块号
0	2
1	4
2	6

作业表：

作业号	页表长度	页表起始地址	表目状态
1	2		1
2	3		1

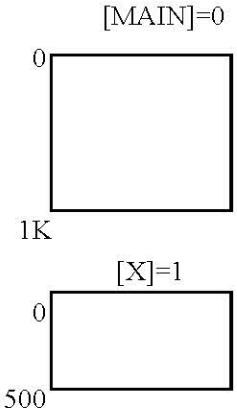


可用页表名代替

⑶用 0～8191 和 0～12287 两个随机数序列模拟作业 1 和作业 2 的地址空间，分别从两随机数序列中各取 20 个随机数表示指令中的虚拟地址；
⑷对于每个虚地址求对应的物理地址，打印出相应的页号、内存块号及相应的物理地址。

方法：
①将虚地址分解成页号 P 和页内位移 d
（可用数学公式或转换成二进制后用位操作取出表示页号的几位）
②越界判断（比较页号与页表长度）：若越界，则打印有关信息
③查页表中该页号所对应的内存块号
④计算物理地址：内存块号*4K+页内位移

2.3 分段存储管理的地址转换的模拟实现



段 表

段号	段长	内存始址
0	1K	6K
1	500B	4K

地址结构：

⑴建立段表、作业表、主存分块表 MBT
⑵从取值范围为 0～2047 的随机数序列中选出 20 个随机数，分别作为段[MAIN]和段[X]的虚地址。
⑶对每一个虚地址计算其代表的有效地址所对应的物理地址，并进行越界判断
⑷要求打印每个虚地址的所在段段号、段内偏移、内存始址及对应的物理地址。

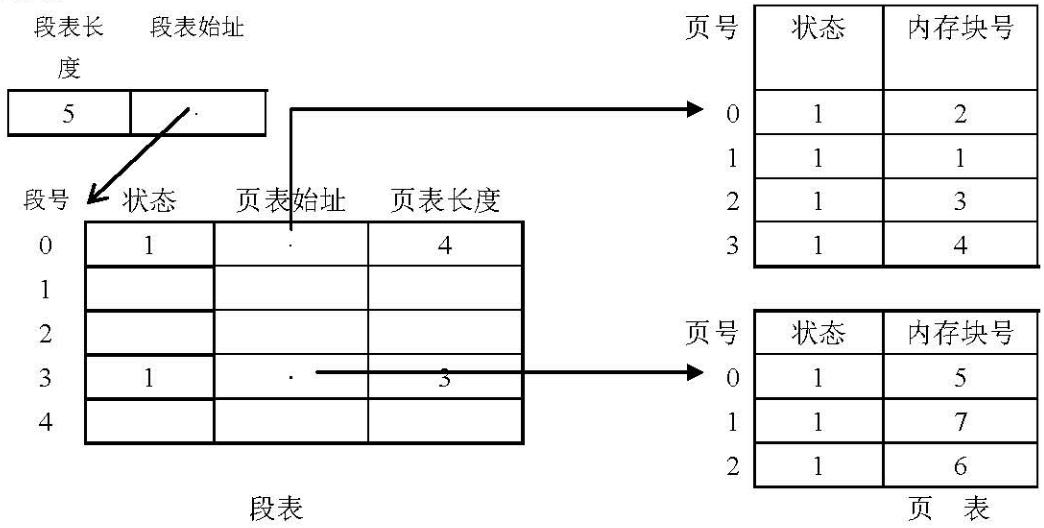
方法：
①对虚地址分解成段号和段内偏移；
②通过段表查出该段在内存的始址；
③物理地址=始址+段内偏移；
④越界判断：段号与段表长比较、段内偏移与段长比较。

2.4 段页式存储管理的地址转换的模拟实现

地址结构：

15 12	11 10	9 0
段号 S	段内页号 P	段内偏移 W

寄存器：



⑴建立相应表格：段表和页表、作业表、主存分块表 MBT

⑵从取值范围为 0～16383 的随机数序列中选出 20 个随机数作为虚地址

⑶计算出每个虚地址所对应的段号、段内页号、页内偏移及物理地址

3.程序及运行结果

(1)程序清单　　(C++语言编写，在 VC 6.0 环境下运行)

```
//*******************************************************************************
//********************           参考代码四  地址转换            **********************
//********************        主程序文件 Address_Convert.cpp       **********************
//********************              版权所有: 陈礼青              **********************
//*******************************************************************************

#include <stdlib.h>
#include <time.h>
#include <iostream>
using namespace std;


////////////////////////////////////////////////////////////////////////
const int rand_len=100;
const int M=20;
const int k=1024;
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////                    随机数发生程序                    //////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////
void RND(const int n0,const int n,const int M,int* const RN)
{
	int temp[rand_len];
	srand( (unsigned)time( NULL ) );

	for(int i=0;i<rand_len;i++)
		temp[i]=rand()%(n-n0+1);

	int index;
	for(i=0;i<M;i++)
	{
			index=rand()%rand_len;
			RN[i]=temp[index];
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////          1.可变分区存储管理的地址转换的模拟实现               //////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void SubArea_varible()
{
	int BR=1400,LR=2048;
	int RN[M];

	cout<<"1.可变分区存储管理的地址转换的模拟实现:\n";

	RND(0,2500,M,RN);

	//硬件支持：基地址寄存器 BR 与长度寄存器 LR
	cout<<"BR=1400,LR=2048\n";
	cout<<"访问序号  虚地址\t 物理地址  越界判断\n";
	for(int i=0;i<M;i++)
	{
		cout<<i<<"\t"<<RN[i]<<"\t"<<RN[i]+BR;
		if(RN[i]>=LR)
			cout<<"\t!!!越界";          //物理地址=虚地址  + BR 的值
		cout<<endl;
	}
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////          2.分页存储管理的地址转换的模拟实现                    //////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Pagination()
{
    //  ⑪建立两个作业的页表(用两个数组或一个数组)
    int PT1[2]={5,7};                   //页表 1,<<4096
    int PT2[3]={2,4,6};                 //页表 2
    int PTL1=2,PTL2=3;
    int RN1[M],RN2[M],temp;

    cout<<"2.分页存储管理的地址转换的模拟实现:\n";

    //  ⑫用 0～8191 和 0～12287 两个随机数序列模拟作业 1 和作业 2 的地址空间，分别从两随机
    //数序列中各取 20 个随机数表示指令中的虚拟地址；
    RND(0,8191,M,RN1);

    short p,d;                          //  ⑬将虚地址分解成页号 P 和页内位移 d
                                        //页内位移 12 位，设指令地址 15 位，故页号 3 位
    cout<<"作业一的地址转换:\n";
    cout<<"访问序号  虚地址\t 物理地址  页号\t 页内位移  块号\t 越界判断信息\n";
    //  ⑭对于每个虚地址求对应的物理地址，打印出相应的页号、内存块号及相应的物理地址。

    for(int i=0;i<M;i++)
    {
        temp=RN1[i]&0x7fff;
        p=temp>>12;
        d=RN1[i]&0x0fff;

        cout<<i<<"\t"<<RN1[i]<<"\t";
        if(p>=PTL1)
            cout<<"\t\t\t\t\t\t!!!越界";     //  ⑮界判断（比较页号与页表长度）：若越界，则打印有关信息
        else                                //  ⑯算物理地址：内存块号*4K+页内位移
            cout<<PT1[p]*4*k+d<<"\t"<<p<<"\t"<<d<<"\t"<<PT1[p];//     //  ⑰页表中该页号所对应的内存块号
        cout<<endl;
    }
    getchar();

    RND(0,12287,M,RN2);
```

```
    cout<<"作业二的地址转换:\n";
    cout<<"访问序号 虚地址\t 物理地址 页号\t 页内位移 块号\t 越界判断信息\n";
    for(i=0;i<M;i++)
    {
        temp=RN2[i];
        p=temp>>12;
        d=RN2[i]&0x0fff;
        cout<<i<<"\t"<<RN2[i]<<"\t";
        if(p>=PTL2)
            cout<<"\t\t\t\t\t\t!!!越界";
        else
            cout<<PT2[p]*4*k+d<<"\t"<<p<<"\t"<<d<<"\t"<<PT2[p];
        cout<<endl;
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////        3. 分段存储管理的地址转换的模拟实现                    //////////
////////////////////////////////////////////////////////////////////////////////////////////////////
void Segment()
{
    int RN[M],temp;
    struct sec
    {
        int sec_len;                    //段长
        int sec_sta;                    //内存始址
    }seg[2]={{1024,6},{500,4}};         //段表

    cout<<"3. 分段存储管理的地址转换的模拟实现:\n";

    //  (从)取值范围为 0～2047 的随机数序列中选出 20 个随机数，分别作为段[MAIN]和段[X]的虚地址。
    RND(0,2047,M,RN);

    short s,w; //s,段号 3 位; w,段内偏移 10 位; >=1024,<1024
    //  要求打印每个虚地址的所在段段号、段内偏移、内存始址及对应的物理地址。
    cout<<"访问序号 虚地址 物理地址 段号\t 段内偏移 内存始址\t 越界判断信息\n";
    for(int i=0;i<M;i++)
    {
        temp=RN[i]&0x1c00;
        s=temp>>10;
        w=RN[i]&0x03ff;

        cout<<i<<"\t"<<RN[i]<<"\t";
```

```
            cout<<" "<<seg[s].sec_sta*k+w<<"\t";        //物理地址=始址+段内偏移；
            cout<<s<<"\t"<<w<<"\t"<<seg[s].sec_sta<<"k";

            if(s>=2) cout<<"\t\t!!!段号越界";
            //  ④越界判断：段号与段表长比较、段内偏移与段长比较。
            if(w>=seg[s].sec_len)    cout<<"\t\t!!!偏移越界";
            cout<<endl;
        }
}

class sec
{
public:
        bool status;
        int *pts;
        unsigned ptl;
        sec()
        {
            status=false;
            pts=0;
            ptl=0;
        }
};

/////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////          4.段页式存储管理的地址转换的模拟实现              /////////
/////////////////////////////////////////////////////////////////////////////////////////////////////////
void Seg_Pag()
{
        short s,p,w;     int temp,RN[M];
        int PT0[4]={2,1,3,4};                    //页表 1,<<4096
        int PT1[4]={0};
        int PT3[3]={5,7,6};                      //页表 2
        sec seg[5];

        seg[0].status=true;seg[0].pts=PT0;seg[0].ptl=4;
        seg[1].status=true;seg[1].pts=PT1;seg[1].ptl=1;
        seg[3].status=true;seg[3].pts=PT3;seg[3].ptl=3;
        int seg_len=5; //寄存器：
                    //  ⑸从取值范围为 0～16383 的随机数序列中选出 20 个随机数作为虚地址
        cout<<"4.段页式存储管理的地址转换的模拟实现:\n";

        RND(0,16383,M,RN);
```

```
	// ⑶算出每个虚地址所对应的段号、段内页号、页内偏移及物理地址
	cout<<"访问序号 虚地址   段号\t 段内页号 页内偏移 内存块号 物理地址   越界判断
信息\n";
	for(int i=0;i<M;i++)
	{
		temp=RN[i];
		s=temp>>12;
		temp=RN[i];
		p=temp&0x0c00;
		p=p>>10;
		w=RN[i]&0x03ff;

		cout<<i<<"\t"<<RN[i]<<"\t";
		cout<<s<<"\t"<<p<<"\t"<<w;

		if(s>=seg_len) cout<<"\t\t\t      段号越界!!!";
		else
			if(seg[s].status==false)
				cout<<"\t\t\t      缺段中断!!!";
			else
				if(p>=seg[s].ptl)
					cout<<"\t\t\t      页号越界!!!";
				else
					cout<<"\t    "<<seg[s].pts[p]<<"\t    "<<seg[s].pts[p]*k+w;
		cout<<endl;
	}
}

//*******************************************************************
//***************                    主函数                 *************
//*******************************************************************
void main()
{
	SubArea_varible(); getchar();
	Pagination();          getchar();
	Segment();             getchar();
	Seg_Pag();
}
```