电子科技大学

实
验
报
告

课程：计算机操作系统

姓名：李果念

学号：2013040203030

页式存储逻辑地址到物理地址映射

条件：64 位地址空间
输入：
页记录大小（如 4Byte）
页大小（如 4KB）
页表级数（如，2 表示 2 级页表，n 表示 n 级页表）
逻辑地址（十六进制）
输出：物理地址（物理块号，块内偏移）
说明：页表随机产生，为便于验证可令逻辑页号 n 的物理块号为 n。

实验源代码：

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <malloc.h>

//__64
//unsigned short pagetable[256]\[256]\[256]\[256]\[1024];
//页记录大小(8 字节) 页大小 逻辑地址
//页大小/页记录大小=每页记录数
//页表级数=(64-log2 页大小)/log2 每页记录数（若有余数则+1）
//随机排列的序数表

struct LCG
{
unsigned multiple;//倍数
unsigned addition;//加数
}lcg[32];

struct LOG
{
    unsigned char bit;
    unsigned __int64 pwr;
}recsize,pagesize;

struct KILO
{
    unsigned char n;
    unsigned short b;//最多能取 512
    char prefix;
}simp;

struct PAGETABLE
{
    char mod;//最高级页表
    char lth;//中间级页表
    char grd;//中间级页表级数
    char ini;//页内地址长度
}pagetbl;

char hex[17]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};//逻辑地址（文字）
char num[17]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};//逻辑地址（数字）
char num_phy[17]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};//物理地址（数字）
char hex_phy[17]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};//物理地址（文字）
char addr[65];//逻辑地址（字节）
char addr_phy[65];//物理地址（字节）

void tonum_phy(void)
{
char i,j;
for (i=0;i<=15;i++)
for (j=0;j<=3;j++)
{
num_phy[i]+=addr_phy[4*i+j]<<(3-j);
}

}
```

```
void tohex_phy(void)
{
     char i;
     for (i=0;i<=15;i++)
     {
     if ((num_phy[i]>=0)&&(num_phy[i]<=9))
          hex_phy[i]=num_phy[i]+0x30;
     else hex_phy[i]=num_phy[i]-0x09+0x40;
     }

}

void fillLCG(void)//这个函数是用来构造线性同余随机数的。
{
lcg[0].multiple=0; lcg[0].addition=0;
lcg[1].multiple=1; lcg[1].addition=1;
lcg[2].multiple=3; lcg[2].addition=1;
lcg[3].multiple=5; lcg[3].addition=3;
lcg[4].multiple=9; lcg[4].addition=7;
lcg[5].multiple=19;    lcg[5].addition=13;
lcg[6].multiple=39;    lcg[6].addition=25;
lcg[7].multiple=79;    lcg[7].addition=49;
lcg[8].multiple=159;   lcg[8].addition=97;
lcg[9].multiple=317;   lcg[9].addition=195;
lcg[10].multiple=633;  lcg[10].addition=391;
lcg[11].multiple=1265; lcg[11].addition=783;
lcg[12].multiple=2531; lcg[12].addition=1565;
lcg[13].multiple=5063; lcg[13].addition=3129;
lcg[14].multiple=10125;lcg[14].addition=6259;
lcg[15].multiple=20251;lcg[15].addition=12517;
lcg[16].multiple=40503;lcg[16].addition=25033;
lcg[17].multiple=81007;lcg[17].addition=50065;
lcg[18].multiple=162013;    lcg[18].addition=100131;
lcg[19].multiple=324027;    lcg[19].addition=200261;
lcg[20].multiple=648055;    lcg[20].addition=400521;
lcg[21].multiple=1296111;   lcg[21].addition=801041;
lcg[22].multiple=2592223;   lcg[22].addition=1602081;
lcg[23].multiple=5184445;   lcg[23].addition=3204163;
lcg[24].multiple=10368889;  lcg[24].addition=6408327;
lcg[25].multiple=20737779;  lcg[25].addition=12816653;
lcg[26].multiple=41475559;  lcg[26].addition=25633305;
lcg[27].multiple=82951117;  lcg[27].addition=51266611;
lcg[28].multiple=165902235; lcg[28].addition=102533221;
lcg[29].multiple=331804471; lcg[29].addition=205066441;
lcg[30].multiple=663608943; lcg[30].addition=410132881;
lcg[31].multiple=1327217885;    lcg[31].addition=820265763;
}

unsigned LCGtreat(unsigned seed,unsigned char bits,unsigned char times)//这个函数是用来生成线
性同余随机数的。
{
unsigned char i=0;
unsigned __int64 sed=seed;
unsigned ret;
unsigned __int64 mul=lcg[bits].multiple;
unsigned __int64 add=lcg[bits].addition;
unsigned mod=1;
mod=1<<bits;
for (i=0;i<times;i++)
```

```
     {
     sed=(((sed*mul)%mod)+add)%mod;
     }
ret=(unsigned)sed;
return ret;
}

void getpagetable(void)
{
     pagetbl.ini=pagesize.bit;
     pagetbl.lth=pagetbl.ini-recsize.bit;
     pagetbl.mod=(64-pagetbl.ini)%pagetbl.lth;
     pagetbl.grd=(64-pagetbl.ini)/pagetbl.lth;
     if (pagetbl.mod==0)
     {
          pagetbl.mod=pagetbl.lth;
          pagetbl.grd--;
     }
}

void numtoaddr(void)
{
     char i=0;
     char j=0;
     for (i=0;i<=15;i++)
     {
          for (j=0;j<=3;j++)
          addr[4*i+j]=(num[i]&(1<<(3-j)))>>(3-j);
     }
}

unsigned __int64 addrtonum(char bgn, char p)//提取从 bgn 开始 p 位的数据
{
     char i=0;
     char t=bgn;
     unsigned __int64 r=0;
     unsigned __int64 reg=1;
     for (i=p-1;i>=0;i--)
     {
          if (addr[t])
          r+=reg<<i;
          t++;
     }
     return r;
}

void numtoaddr_phy(unsigned n, char bgn, char p)//用 n 填充从 bgn 开始 p 个字节的数组 addr_phy[]
{
char i=0;
char q=p-1;
for (i=0;i<p;i++)
{
q=p-1-i;
addr_phy[bgn+i]=(n&(1<<q))>>q;
}

}

char pref(unsigned char i)//求前缀
```

```
{
     switch (i)
     {
          case 1:return 'k';
          case 2:return 'M';
          case 3:return 'G';
          case 4:return 'T';
          case 5:return 'P';
          case 6:return 'E';
          default:return 'e';
     }
}

struct KILO kilo(unsigned char i)//求前缀
{
struct KILO p;
unsigned char j=i/10;
p.n=i%10;
p.b=1<<p.n;
p.prefix=pref(j);
return p;
}

void input(void)//输入逻辑地址
{
     char i;
     char j=15;
     char strl;
     gets(hex);
     gets(hex);//这里用 2 个 gets()是因为第一个 gets()不起作用
     strl=strlen(hex);
     if (strl<16)
     {
          for (i=strl-1;i>=0;i--)
          {
               hex[j]=hex[i];
               j--;
          }
          for (;j>=0;j--)
               hex[j]=0x30;
     }
}

void empty(void)
{
char i;
for (i=0;i<=15;i++)
     hex[i]=0;
}

char checkillegalhex(void)
{
     char i=0;
     char r=0;
     for (i=0;i<=15;i++)
     {
     if (!isxdigit(hex[i]))//ctype.h
     r++;
     else ;
```

```
    }
    return r;
}

void hextonum(void)
{
    char i=0;
    for (i=0;i<=15;i++)
    {
        if (isdigit(hex[i]))
            num[i]=hex[i]-0x30;
        else num[i]=(hex[i]%0x20)+0x09;
    }
}

void printhex(char p)
{
if ((p>=0)&&(p<=9))
printf("%d",p);
else printf("%c",p-0x09+0x40);
}

int main()
{
    char beg=0;
    char i=0;
    char confirm=0;
    char logiceqphysic=0;

    unsigned int* p=0;
    unsigned int* part1=0;
    unsigned __int64 part2=0;
    printf("64 位系统页表实验\n");
    do
    {
        printf("请输入页大小（以字节为单位取以 2 为底的对数）。\n");
        do
        {
            scanf("%u",&pagesize.bit);
            if (pagesize.bit<1)
                printf("页大小仅为 1 字节，因此页记录大小无法取值，请重新输入。\n");
            else if (pagesize.bit>=64)
                {
                    printf("页大小至少为 2 的 64 次方字节，逻辑地址就是物理地址。\n");
                    logiceqphysic=1;
                    break;
                }
            else break;
        }while (1);
        if (pagesize.bit<=63)
        {
            pagesize.pwr=1;
            pagesize.pwr=pagesize.pwr<<pagesize.bit;
            printf("你输入的页大小为%I64u 字节。\n",pagesize.pwr);
            if (pagesize.bit>=10)
            {
                simp=kilo(pagesize.bit);
                printf("即%u%c",simp.b,simp.prefix);
                printf("B。\n");
```

```
            }
        }
        else
            logiceqphysic=1;
        printf("确定吗？\n 输入 1 表示确定，输入 0 表示取消。\n");
        scanf("%d",&confirm);
    }while (confirm==0);
    do
    {
        printf("请输入页记录大小（以字节为单位取以 2 为底的对数）\n");
        do
        {
            scanf("%u",&recsize.bit);
            if (recsize.bit>=pagesize.bit)
                printf("此时一页最多只能存放一张页表，因为要求一页最少要存放两张页表，所以
请重新输入\n");
            else
            {
                recsize.pwr=1;
                recsize.pwr=recsize.pwr<<recsize.bit;
                break;
            }
        }while (1);

        printf("你输入的页记录大小为%I64u 字节。\n",recsize.pwr);
            if (recsize.bit>=10)
            {
                simp=kilo(recsize.bit);
                printf("即%u%c",simp.b,simp.prefix);
                printf("B。\n");
            }
        printf("确定吗？\n 输入 1 表示确定，输入 0 表示取消。\n");
        scanf("%d",&confirm);
    }while (confirm==0);
    do
    {
        printf("请输入 16 进制逻辑地址（不要输入 0x 或者 h，不区分大小写）\n");
        do
        {
        empty();
        input();
        if (checkillegalhex())
            printf("这不是一个合法的 16 进制逻辑地址，所以请重新输入\n");
        else break;
        }while (1);
        hextonum();
        printf("你输入的 16 进制逻辑地址是：\n");
        for (i=0;i<=15;i++)
            printhex(num[i]);
        printf("\n");
        printf("确定吗？\n 输入 1 表示确定，输入 0 表示取消。\n");
        scanf("%d",&confirm);
    }while (confirm==0);

    if (logiceqphysic==1)
        memcpy(addr_phy,addr,65);
    else //逻辑地址到物理地址查表
    {
    getpagetable();
```

```
    numtoaddr();
    memcpy(&addr_phy[64-pagesize.bit],&addr[64-pagesize.bit],pagesize.bit);
    beg=pagetbl.mod+pagetbl.lth*pagetbl.grd;
    part2=addrtonum(beg, 64-beg);
    part1=(unsigned int*)calloc(pagetbl.grd+1,4);
    p=part1;
    *p=(unsigned int)addrtonum(0,pagetbl.mod);
        if (pagetbl.grd)
        {
        p=part1+1;
        for (i=0;i<pagetbl.grd;i++)
            {
                *p=(unsigned int)addrtonum(pagetbl.mod+i*pagetbl.lth,pagetbl.lth);
                p++;
            }
        }
    fillLCG();
    p=part1;
    *p=LCGtreat(*p,pagetbl.mod,1);
    p++;
    for (i=1;i<=pagetbl.grd;i++)
        {
            *p=LCGtreat(*p,pagetbl.lth,i);
            /*
            void getpagetable(void)
            {
                pagetbl.ini=pagesize.bit;
                pagetbl.lth=pagetbl.ini-recsize.bit;
                pagetbl.mod=(64-pagetbl.ini)%pagetbl.lth;
                pagetbl.grd=(64-pagetbl.ini)/pagetbl.lth;
                if (pagetbl.mod==0)
                {
                    pagetbl.mod=pagetbl.lth;
                    pagetbl.grd--;
                }
            }
            */
            p++;
        }
    p=part1;
    numtoaddr_phy(*p,0,pagetbl.mod);
    p++;
        for (i=0;i<pagetbl.grd;i++)
            {
                numtoaddr_phy(*p,pagetbl.mod+i*pagetbl.lth,pagetbl.lth);
                p++;
            }
    }
        tonum_phy();
        tohex_phy();
printf("对应的物理地址是：\n");
for (i=0;i<=15;i++)
printhex(num_phy[i]);
printf("\n");
return 0;
}
```

运行结果：

64 位系统页表实验
请输入页大小（以字节为单位取以 2 为底的对数）。
14
你输入的页大小为 16384 字节。
即 16kB。
确定吗？
输入 1 表示确定，输入 0 表示取消。
1
请输入页记录大小（以字节为单位取以 2 为底的对数）
3
你输入的页记录大小为 8 字节。
确定吗？
输入 1 表示确定，输入 0 表示取消。
1
请输入 16 进制逻辑地址（不要输入 0x 或者 h，不区分大小写）
0a1b2c3d4e5f6789
你输入的 16 进制逻辑地址是：
0A1B2C3D4E5F6789
确定吗？
输入 1 表示确定，输入 0 表示取消。
1
对应的物理地址是：
9CF2B91768366789

结论：

将逻辑地址按位分段提取出来的数据实质上只是数组下标而已（例如当逻辑地址为 0A1B2C3D4E5F6789，页大小为 16kB，页记录大小为 8 字节时，数组下标（用十进制表示）分别是 2,1078,707,1703,381），查物理地址就是查数组里的元素（在这里是十进制的 690265205350617，十六进制的 273CAE45DA0D9）而已。