电子科技大学

计算机专业类课程

实验报告

课程名称：计算机操作系统

学　　院：计算机科学与工程学院

专　　业：计算机科学与技术

学生姓名：赵泳澎

学　　号：2013060108027

指导教师：薛瑞尼

日　　期：　　2016 年 6 月 6 日

实验一

一、实验名称

- 消费者和生产者问题

二、实验目的和要求

共享缓冲区中放置一个数字，取值范围为[0, 10]，初值为 0。生产者将此值加 1，消费者将此值减 1。

场景 1：

- 同一进程内启动一组生产者线程和一组消费者线程
- 缓冲区为本进程的全局变量

场景 2：

- 启动一组生产者进程和一组消费者进程
- 同一个数据文件为缓冲区

输入

- p：生产者数量
- c：消费者数量

输出

- 打印当前共享缓冲区中的数值，或者生产者消费者的状态

三、实验器材、设备、元器件

- 程序设计语言：C

四、设计思想和算法

资源是计算机中的稀缺个体，具有独占性（不可复用性），进程/线程是计算机中的独立个体，具有异步性（并发性），在计算机中，

各进程协作完成各种任务，并发控制就是如何协调进程/线程在推进中的制约关系。

进程间的制约关系分为直接制约和间接制约，其中，间接制约主要是资源共享互斥，直接制约主要是进程间合作同步。临界资源是一次仅允许一个进程访问的资源，临界区指进程中访问临界资源的代码段。

wait 操作中，只要信号量 S<=0，就会不断地测试。因此，该机制并未遵循“让权等待” 的准则，而是使进程处于“忙等”的状态。记录型信号量是不存在“忙等”现象的进程同步机制。除了需要一个用于代表资源数目的整型变量 value 外，再增加一个进程链表 L，用于链接所有等待该资源的进程，记录型信号量是由于采用了记录型的数据结构得名。

当记录型信号量的值大于 0 时，代表还有多少个资源可用，当他的值小于 0 时，他值的绝对值代表有多少个进程在等待资源。

信号量可以分成两种类型，一种信号量的值初始为 1，代表互斥信号量，资源只能有一个进程/线程访问；另一种信号量为资源信号量，代表共享个数。

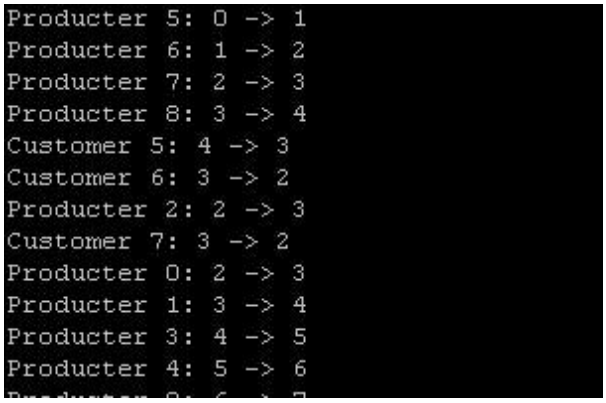
五、实验数据及结果分析

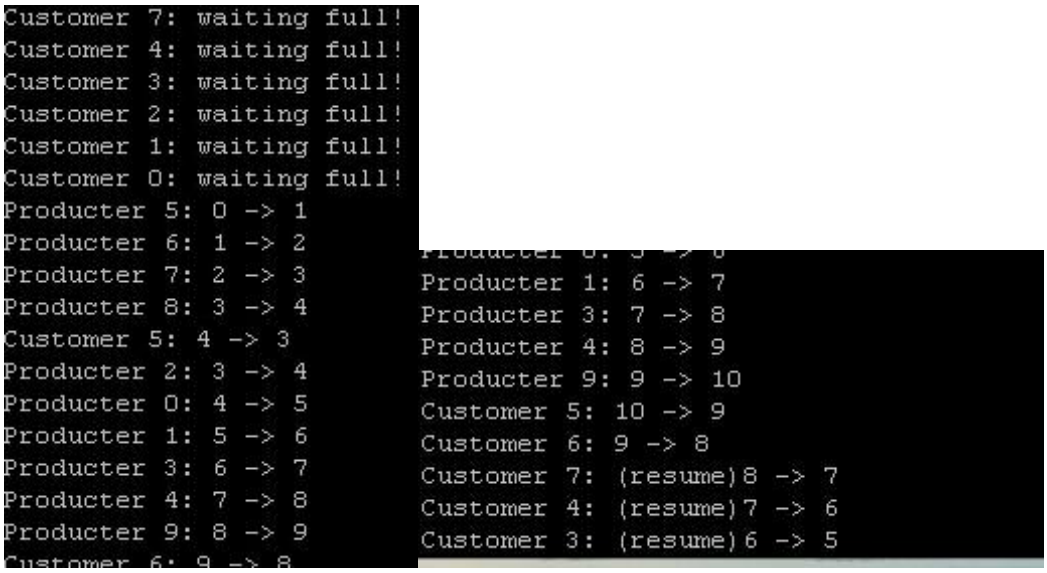
5.1 程序运行过程

用信号量解决生产者—消费者问题时，可以通过一个缓冲区(用整型来表示现有产品数量)把生产者和消费者联系起来。假定生产者和消费者的优先级是相同的，只要缓冲区未满，生产者就可以生产产品并将产品送入缓冲区。类似地，只要缓冲区未空，消费者就可以从缓冲区中去走产品并消费它。为了解决生产者/消费者问题，应该设置两个资源信号量,其中一个表示空缓冲区的数目,用 empty_sem 表示，其初始值为有界缓冲区的大小 ShareBuffer；另一个表示缓冲区中产品的数目，用 full_sem 表示，其初始值为 0。另外，由于有界缓冲区是一个临界资源，必须互斥使用，所以还需要再设置一个互斥信号量 mutex，起初值为 1。在生产者/消费者问题中，信号量实现两种功能。首先，它是生产产品和消费产品的计数器，计数器的初始值是可利用的资源数 目(有界缓冲区的长度)。其次，它是确保产品的生产者和消费者之间动作同步的同步器。生产者要生产一个产品时，首先对资源信号量 empty 和互斥信号量 mutex 进行 wait 操作，申请资源。如果可以通过的话，就生产一个产品，并把产品送入缓冲区然后对互斥信号量 mutex 和资源信号量 full 进行 signal 操作，释放资源。消费者要消费一个产品时，首先对资源信号量 empty 和互斥信号量 mutex 进行 wait 操作，申请资源。如果可以通过的话，就从缓冲区取出一个产品并消费掉。然后对互斥信号量 mutex 和资源信号量 empty 进行 signal 操作，释放资源。

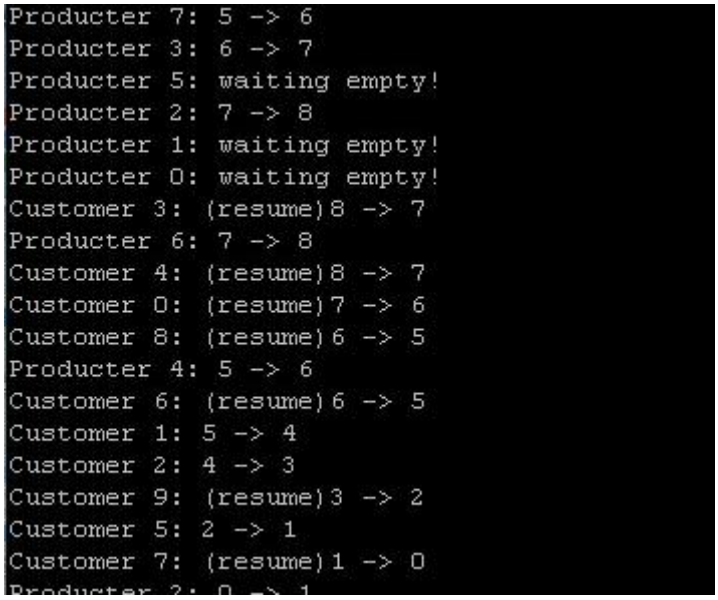
5.2 实验结果

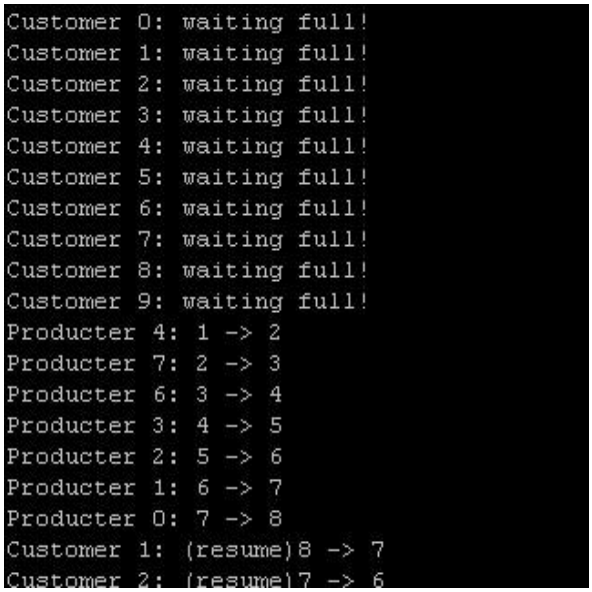
5.2.1 场景一：多线程


Producter 5: 0 -> 1
Producter 6: 1 -> 2
Producter 7: 2 -> 3
Producter 8: 3 -> 4
Customer 5: 4 -> 3
Customer 6: 3 -> 2
Producter 2: 2 -> 3
Customer 7: 3 -> 2
Producter 0: 2 -> 3
Producter 1: 3 -> 4
Producter 3: 4 -> 5
Producter 4: 5 -> 6


Customer 7: waiting full!
Customer 4: waiting full!
Customer 3: waiting full!
Customer 2: waiting full!
Customer 1: waiting full!
Customer 0: waiting full!
Producter 5: 0 -> 1
Producter 6: 1 -> 2
Producter 7: 2 -> 3
Producter 8: 3 -> 4
Customer 5: 4 -> 3
Producter 2: 3 -> 4
Producter 0: 4 -> 5
Producter 1: 5 -> 6
Producter 3: 6 -> 7
Producter 4: 7 -> 8
Producter 9: 8 -> 9
Producter 1: 6 -> 7
Producter 3: 7 -> 8
Producter 4: 8 -> 9
Producter 9: 9 -> 10
Customer 5: 10 -> 9
Customer 6: 9 -> 8
Customer 7: (resume)8 -> 7
Customer 4: (resume)7 -> 6
Customer 3: (resume)6 -> 5

5.2.2 场景二：多进程


Producter 7: 5 -> 6
Producter 3: 6 -> 7
Producter 5: waiting empty!
Producter 2: 7 -> 8
Producter 1: waiting empty!
Producter 0: waiting empty!
Customer 3: (resume)8 -> 7
Producter 6: 7 -> 8
Customer 4: (resume)8 -> 7
Customer 0: (resume)7 -> 6
Customer 8: (resume)6 -> 5
Producter 4: 5 -> 6
Customer 6: (resume)6 -> 5
Customer 1: 5 -> 4
Customer 2: 4 -> 3
Customer 9: (resume)3 -> 2
Customer 5: 2 -> 1
Customer 7: (resume)1 -> 0



六、实验结论、心得体会和改进建议

在操作系统中对进程控制最重要的一点就是进程的同步与互斥，操作系统实现的信号量机制可以有效的保证进程推进的顺序性。

。

实验二

一、实验名称

- 银行家算法

二、实验目的和要求

输入

- p: 进程数量
- r：资源数量
- 各进程的 max, allocation

输出

- 若产生死锁，打印提示：死锁状态。
- 否则，给出一种调度顺序

三、实验器材、设备、元器件

- 程序设计语言：C

四、设计思想和算法

4.1 银行家算法原理

我们可以把操作系统看作是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。

为保证资金的安全，银行家规定：

(1) 当一个顾客对资金的最大需求量不超过银行家现有的资金时就可接纳该顾客；

(2) 顾客可以分期贷款，但贷款的总数不能超过最大需求量；

(3) 当银行家现有的资金不能满足顾客尚需的贷款数额时，对顾客的贷款可推迟支付，但总能使顾客在有限的时间里得到贷款；

(4) 当顾客得到所需的全部资金后，一定能在有限的时间里归还所有的资金.

操作系统按照银行家制定的规则为进程分配资源，当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。当进程在执行中继续申请资源时，先测试该进程本次申请的资源

数是否超过了该资源所剩余的总量。若超过则拒绝分配资源，若能满足 则按当前的申请量分配资源，否则也要推迟分配。

4.2 程序设计流程图

开始

输入进程数 n

输入资源总量 m

输入每个进程所需的资源最大量 max

输入每个资源已分配的资源总数 All

各种可利用资源的数量 Avai

系统是否有足够的资源分配给进程 fi=0（否）

need[i][i]<work&&fi[i]=0

work+=All[i]

fi[i]=1

所有进程 fi[i]=1

系统不安全

输出安全序列

request[i]<need[i]

request[i]<Avai[i]

Avai[i]=request[i]

all[1]+=request[i]

need[i]-=request[i]

判断安全性

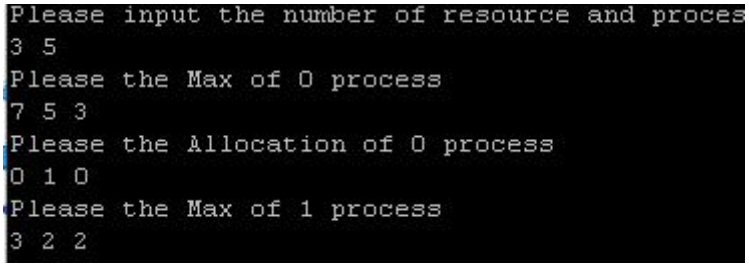
五、实验数据和结果分析

以旧版教材 97 页例子进行测试：

在 T0 时刻

	MAX			Allocation		
	A	B	C	A	B	C
P0	7	5	3	0	1	0
P1	3	2	2	2	0	0
P2	9	0	2	3	0	2
P3	2	2	2	2	1	1
P4	4	3	3	0	0	2

此时的 available 为（3,3,2）



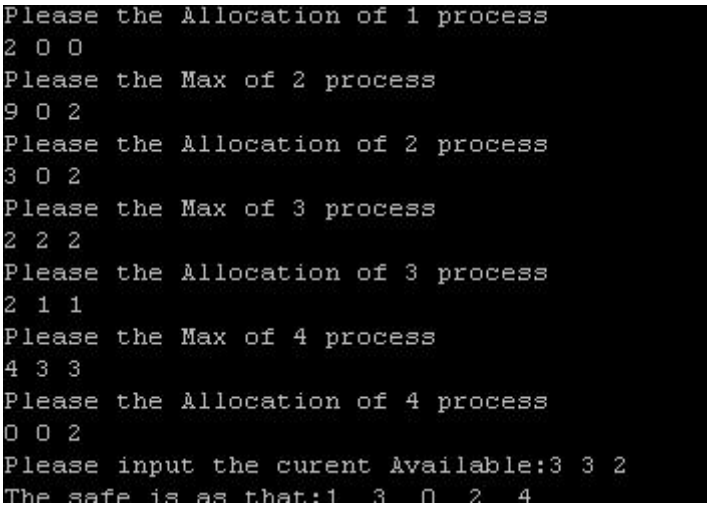


图 1：输入初始化状态图，并判断是否安全

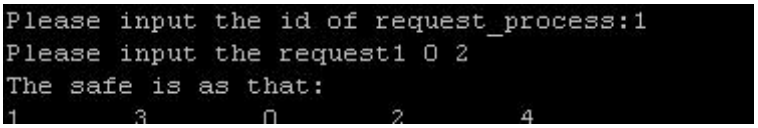


图 2：进程 P1 请求资源（1,0,2），检查知安全，输出安全序列，分配

六、心得体会与改进建议

通过本次实验加深了我对银行家算法的理解，感觉银行家算法可以应用在更多的地方，我们现在学的东西都很基础，当我们有能力做的更多的时候，可以更好地运用我们学过的知识。

实验三

一、 实验名称：多级页表分配

二、 实验内容和目的：

条件：64 位地址空间

输入：

- 页记录大小（如 4Byte）
- 页大小（如 4KB）
- 逻辑地址（十六进制）

输出：物理地址（物理块号，块内偏移）

说明： 1. 自动计算页表级数； 2. 页表随机产生，为便于验证可令最后一级逻辑页号 n 的物理块号为 n。

本次试验的主要目的是通过编程梳理页表，特别是多级页表的建立过程，求解在多级页表中如何求页面大小，页表大小。建立内存管理的基本框架。

三、 实验器材、设备、元器件

- 程序设计语言：C

四、 实验原理

页式存储方式是为了提高存储器空间利用率应运而生的。传统的存储方式会产生大量的内部碎片或者外部碎片，而使用页式存储，不会产生外部碎片，又因为每一个页都比较小，所以产生的内部碎片也不大。

页式存储的基本思想是是将一个进程的逻辑地址空间分成若干

个大小相等的片，称为页，并为各页加以编号，从 0 开始，如第 0 页、第 1 页等。相应地，也把内存空间分成与页面相同大小的若干个存储块，称为物理页号，也同样为它们加以编号。在为进程分配内存时，以页为最小分配单位，即每一个进程都至少分配一个完整的页面，如果一个页面有空余空间没有分配，也不对此页面进行拆分。页表小一方面可使内存碎片减小，减少了内存碎片的总空间，有利于提高内存利用率，但另一方面也会使每个进程占用较多的页面，从而导致进程的页表过长，占用大量内存；此外，还会降低页面换进换出的效率。页表大可以减少页表的长度，提高页面换进换出的速度，但却又会使页内碎片增大。为了兼顾性能与利用率，页面的大小通常为 512 B～8 KB，并且为 2 的 n 次幂；

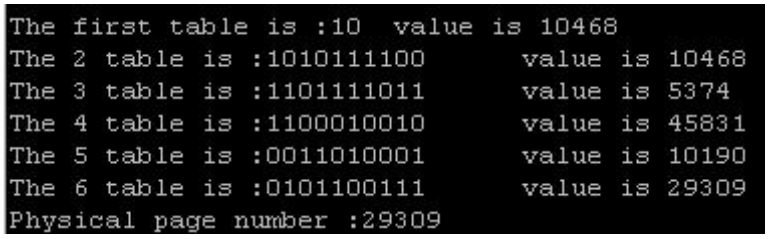
一般来说，页表项有一定的大小，并且页表项也是存储在页表中，随着计算机位数的增加，一个单独的页面不能存储下所有的页表项。举个例子：一个 32 位的存储器，页面大小 4kb，页表项 4kb，可以算出每一个页表可以存储 2 的 10 次方个页表项，排除 12 位的页内地址，系统总共需要 2 的 20 次方的页表项来存储所有页号。但是一个页表只可以存储 2 的 10 次方个页表项。系统很难找到一块足够大的内存空间存储所有页表项，于是我们采用离散分配方式来解决难以找到一块连续的大内存空间的问题，这就是所谓的二级页表和多级页表。对于要求连续的内存空间来存放页表的问题，可利用将页表进行分页，并离散地将各个页面分别存放在不同的物理块中的办法来加以解决，同样也要为离散分配的页表再建立一张页表，称为外层页表(Outer

Page Table)，在每个页表项中记录了页表页面的物理块号。

在计算机中，多级页表的逻辑地址和物理地址的转换一般使用硬件电路完成。

五、 实验数据及结果分析

程序运行需输入页面大小，页表项大小和一个 16 进制的地址，在本次测试中分别为 4B,4096B。4096=2^12，则页内地址为 12 位。4096/4=1024=2^10，可知每个页面中可存储 1024 个页表项，即除最外层页表外，每级表号都是 10 位。64-12-10*5=2，则在本次测试中为 6 级页表，第 1 级页表号 2 位，其他页表号 10 位，页内偏移地址 12 位。试验运行截图如下



六、 心得体会与改进建议

在一开始做实验的时候我试着构建了下页表，结果发现 64 位地址，页面大小 4096，页记录大小 4B，这种情况下需要 6 级页表，而且单单一级页表就需要 1.099*10^12 个所需内存过大。后来查资料得知现有的 64 位操作系统内存可直接寻址的存储器空间减小为 45 位，这样就只需要三级页表就可以实现。

实验四

一、 **实验名称：**混合索引逻辑地址到物理地址映射

二、 **实验内容和目的：**

条件：自定义混合索引 inode 结构

i. 必须包括一次，二次，和三次间接块

ii. 逻辑块 n 对应物理块 n

输入：文件逻辑地址

输出

i. 输出 inode 详细信息（间接块不展开）

ii. 物理地址（物理块号，块内偏移）

本实验目的是通过编程梳理混合索引的知识，掌握多级索引求物理块号的计算方法，同时，模拟 linux 操作系统 inode 的实现，更加深入了解 linux 文件系统的原理和实现。

三、 **实验器材、设备、元器件**

- 程序设计语言：C

四、 **实验原理：**

磁盘管理方式有四种实现：

- 第一种为顺序结构：具体的实现为把每一个文件的文件信息存放在若干连续的物理块中，这是最简单的磁盘利用方式，它的优点是简单，支持顺序和随机存取，寻道次数少，但是由于是文件信息连续存储，所以不利于文件修改，每一次修改都意味着修改单

元其后所有文件块都要顺序移动，造成时间浪费，他还会导致磁盘碎片，降低外存空间利用率。一般这种实现都会采用紧凑技术，即过一段时间将所有的文件向前移动，将空余块填满以减少磁盘空间外部碎片。

- 第二种方式为连接结构，它的特点是文件信息存放在若干不连续的物理块中，各块之间通过指针连接。每一个物理块的结尾都有一个隐式的指针，指向下一个物理块的位置。在整个磁盘中还有一个 FCB，维持着所有文件的起始指针和一些其他信息。连接实现大大的提高了磁盘空间利用率，也更加有利于文件的动态变化。但是连接结构不支持随机访问，在寻到上会话更多的时间。
- 第三个实现是索引结构，特点是文件信息存放在若干不连续物理块中，每个文件关联一个索引表，记录块号，索引表就是磁盘块地址数组，第 i 个条目指向文件的第 i 块。在磁盘中有一个文件分配表，记录了文件名和索引块号，当进行文件的读取时，只需通过文件名就可以知道文件在哪一个索引中，索引存储着文件真实存放在哪些物理块中，通过索引的信息，就可以计算出文件的真实物理地址（物理块号，块内偏移）。索引结构在随机存取性能上强于连接结构，同时也满足了文件动态增长、插入删除要求。缺点是较多的寻道次数和时间，并且索引表本身也带来了一定的系统开销。

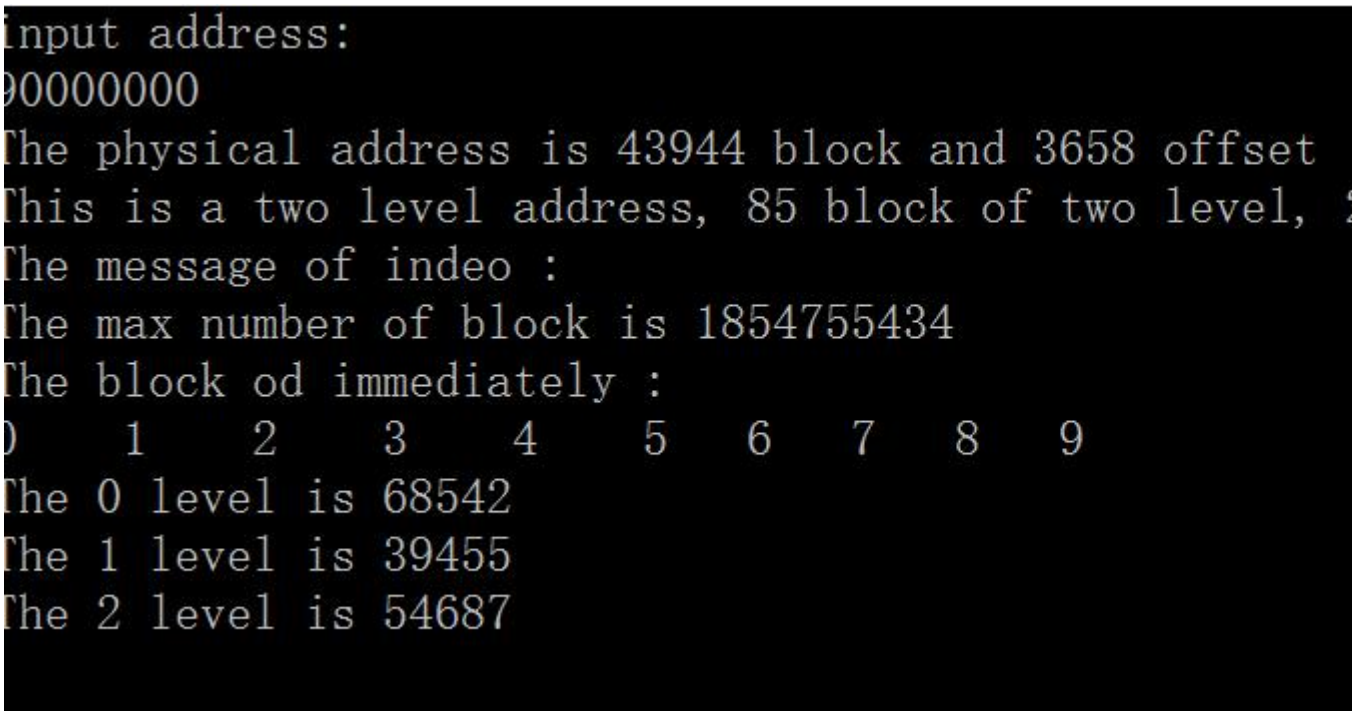
大多数的操作系统都采用了索引表结构的存储方案，但是随着文件越来越大，普通的索引结构已经不能支持大文件的传送，因为可能

一个物理块中保存的块号有限，当文件很大时，一个盘块就不能装下所有的分区。处理这种问题的方式一般是使用多个索引结点进行存储。可以使用连接模式：一个盘块一个索引表，多个索引表链接起来。也可以使用多级模式：将所有索引表的地址放在另一索引表中；或者采用混合模式：混合使用多种模式。Linux 文件系统就采用了混合模式，既采用了直接地址，也采用了多级索引。Linux 文件系统的 inode 节点结构包括了 13 个索引项，每项 2 个字节，前 10 项为直接块，第 11 项指向一个物理块，该块中最多可放 256 个文件物理块的块号（一次间接寻址），第 12 和第 13 项作为二次和三次间接寻址。假设块大小为 1kb，那么 linux 文件系统文件最大寻址空间可以达到 16GB 左右。

五、 实验数据及结果分析：

本次试验中采用 64 位地址，节点包括的信息有：文件最多拥有块的数量，然后还有十个直接块，一个一次间接块，一个二次间接块，一个三次间接块。在设计时盘块大小设计成 4096B 大小，每一个盘块记录设计成 4B 大小，即在一个盘块中可以存储 1024，即 2 的 10 次方个盘块号，使得本文件系统可以支持最大 10+2^10+2^20+2^30 个块，即文件最大(10+2^10+2^20+2^30)*4096B。

试验运行截图如下：



测试中输入的地址为 90000000，运算可知这个在整个文件的第 43944 块中存放，块内偏移地址为 3658。这是存放在二次间接块中的，是其中的第 85 个一次间接块块，是这个一次间接块中的第 202 块。根据输出的信息可知，这个文件最多可以拥有 1854755434 个块。直接块对应的物理块号分别是 0,1,2,3,4,5,6,7,8,9，一次间接块对应物理块号 68542，二次间接块对应物理块号 39455，三次间接块对应物理块号 54687。

六、 实验结论、心得体会和改进建议：

混合索引是指将多种索引分配方式结合而形成的一种分配方案，其中既包括了直接索引，有包括了一，二三及索引。