学号：201910121213

“操作系统”课程设计

题 目： 线程安全型数据结构（单链表实现）

班 级： 网络191

姓 名： 徐艺芸

指导老师： 毕坤

完成日期： 2021年12月10日

上海海事大学信息工程学院

**2021 年 12 月**

**目 录**

[1.引言 1](#_Toc90070941)

[1.1动机 1](#_Toc90070942)

[1.2 要解决的问题 1](#_Toc90070943)

[2.系统设计 2](#_Toc90070944)

[2.1系统总体框架 2](#_Toc90070945)

[2.2 系统总体流程 3](#_Toc90070946)

[2.3 系统功能模块 4](#_Toc90070947)

[3.数据结构设计 4](#_Toc90070948)

[3.1单向链表数据结构 4](#_Toc90070949)

[3.2读者模型 11](#_Toc90070950)

[3.3写者模型 14](#_Toc90070951)

[4.关键技术与系统实现 15](#_Toc90070952)

[4.1初始化 15](#_Toc90070953)

[4.2选择菜单功能 15](#_Toc90070954)

[4.3读者写者模型构建 17](#_Toc90070955)

[4.4线程同步互斥的实现 26](#_Toc90070956)

[4.5验证数据结构的准确性 29](#_Toc90070957)

[4.6为线程传递多个参数的实现 31](#_Toc90070958)

[5.系统运行结果 33](#_Toc90070959)

[5.1运行环境 33](#_Toc90070960)

[5.2运行与测试结果分析 34](#_Toc90070961)

[6.调试和改进 37](#_Toc90070962)

[7.心得和结论 38](#_Toc90070963)

[8.结论和体会 38](#_Toc90070964)

[9.进一步改进方向 39](#_Toc90070965)

[主要参考文献 40](#_Toc90070966)

# 1.引言

链表作为经典的数据结构，在计算机各个领域中都有着广泛的应用，为了满足用户需求，各种计算机软件的开发都需要应用到链表这一数据结构，由于程序运行在操作系统中，为了让程序更好更快更高效的在操作系统中运行，必须引入同步与互斥机制，那么实现一个线程安全性的链表就显得尤为重要，针对该问题，本课程设计将着力设计一个基于c语言实现的线程安全性链表，以期用于实践生活中。

## 1.1动机

在计算机系统内部，经常发生多个线程同时访问一个数据结构的现象；多个线程会同时修改或读取一个数据结构的内容；多个线程同时读写会引入读写冲突问题，这些操作如果不经过特殊处理，很容易导致读写操作无效；基于上述现象，本课程设计的动机就是为了保证读写正确性， 让程序运行速度更快，输出结果所需时间更少，为将来的研究者提供参考。

所以，本课程设计旨在设计一个能够保证线程安全的链表，在对链表同时进行多次插入删除打印等操作时，相互之间不会产生冲突，保证程序的正常运行，结果正确。

## 1.2 要解决的问题

为实现保证线程安全的链表，本系统需要实现以下几个功能：

1. 用C语言设计链表这一数据结构，实现链表的插入、删除、打印等操作的函数，并保证其正确性。
2. 设计一个测试函数实现对定义好的链表数据结构进行健壮性测试，保证链表自身的准确性和稳定性，继而保证对后序实现链表的线程安全测试准确
3. 设计一个测试函数测试多个线程并发执行同时对其链表值修改时，不会产生冲突和错误。
4. 利用生产者消费者模型设计一个测试函数测试多个线程并发执行同时对其链表值修改和读操作时，不会产生冲突和错误。

# 2.系统设计

## 2.1系统总体框架

以下是关于该课程设计的总体思路，其设计思路如图2.1所示：

（1）先构造单向链表的数据结构，然后用随机生成函数随机生成数对构造的数据结构多次进行测试，通过打印链表实现对链表的可视化，更加直观的验证链表是否正确，实现功能函数是否正确，保证之后验证并发操作和互斥操作不会出现问题；

（2）其次创建两个线程分别对链表多次进行插入节点操作和多次删除插入操作，通过简单的互斥机制实现两个线程同时启用，相互不会发生冲突，实现互斥模型，主要验证方式是打印要插入节点数，删除节点数，删除失败节点数，和最后链表的节点数来验证其是否实现的互斥同步模型正确；

（3）最后设计一个生产者消费者模型，让对链表的插入删除操作作为写者，统计节点数量和查找节点作为读者，通过屏幕的输出分析其构造的模型是否正确。

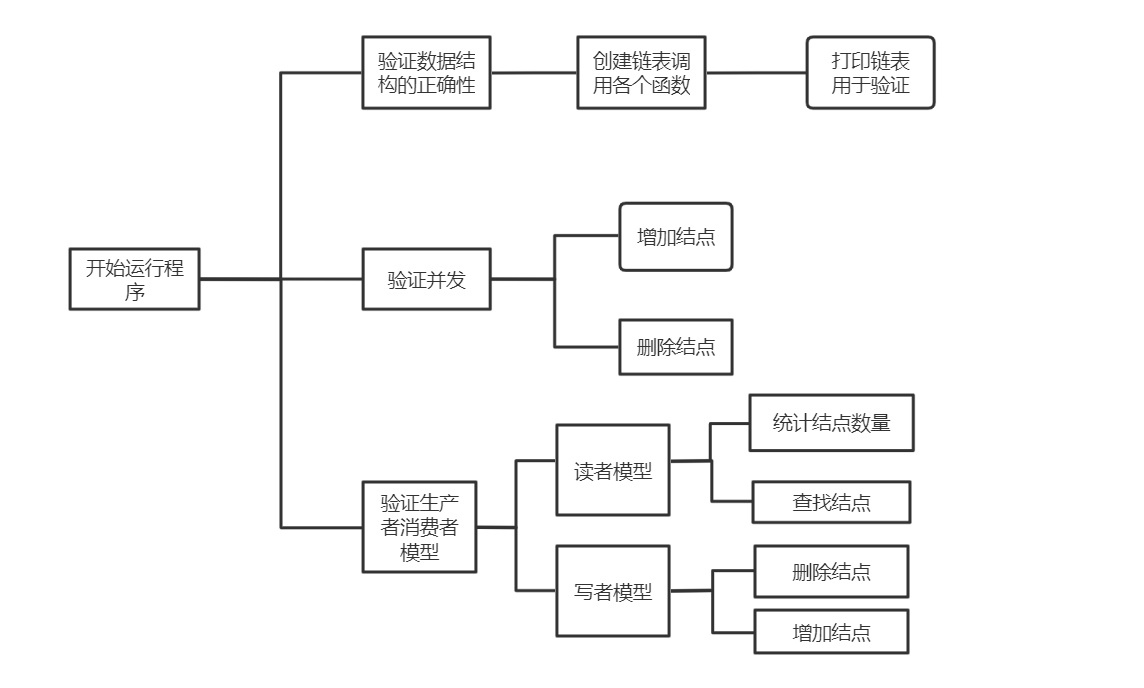


图2.1 系统框架图

## 2.2 系统总体流程

图2.2是总的系统流程图：其中设计了一个选择菜单，用户可以根据需求对其设计的程序进行测试，最后测试完毕后，程序返回主菜单，再次选择功能进行测试，验证程序功能的准确性；利用随机生成数对程序进行测试排除主观因素对实验结果造成的影响也排除其偶然性。

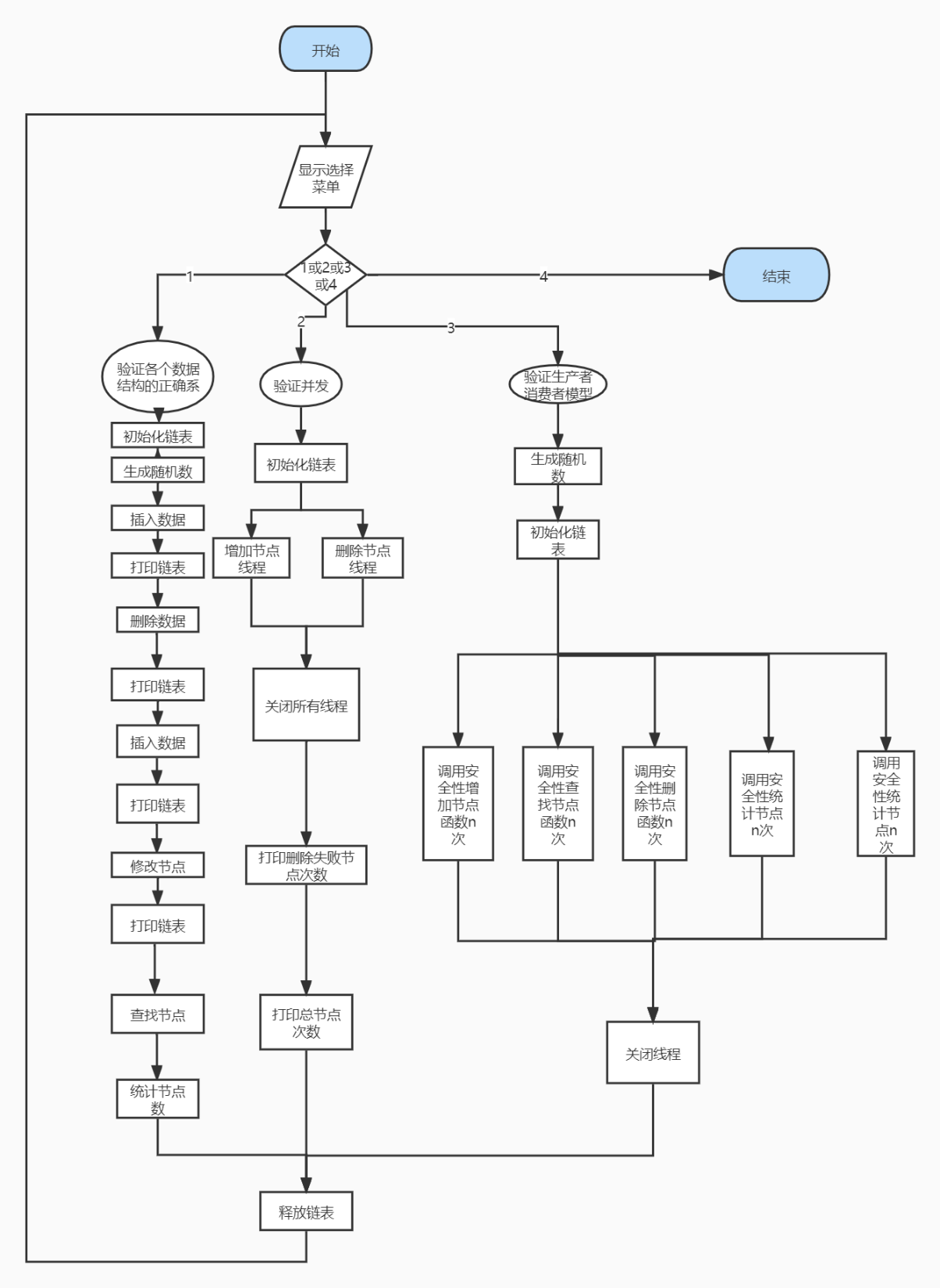


图2.2系统总体流程图

## 2.3 系统功能模块

（1）验证数据结构的正确性

该模块设计思路是构造一个测试函数，初始化链表并随机生成一个链表，分别调用链表的多个实现不同功能的数据结构，每次调用完函数过后打印链表，通过打印的链表验证构造的数据结构的正确性。

（2）验证并发互斥

构造一个测试函数，同时调用多次增加节点的线程和删除结点的线程，两个线程通过调用同一个互斥量对每一次删除操作或增加操作进行保护，防止冲突。

最后打印删除失败信息和节点数量信息验证其准确性。

（3）验证生产者消费者模型

构造一个测试函数，调用多个生产者消费者模型线程，执行完毕后打印其链表的情况验证其模型是否实现成功。

# 3.数据结构设计

本课程设计主要用到的数据结构是单向链表：主要构造的函数如下所示，在创建构造读者写者模型中，让增加删除节点的数据结构作为读者，查找统计节点的数据结构作为写者：

## 3.1单向链表数据结构

节点结构体信息

由于设计的是单链表，结点的数据结构较为简单，该数据结构包含两个变量，其中val为当前节点的值，数据类型为整数类型，next为当前数据结构类型的指针，即含义是当前节点的下一个节点的地址；

|  |
| --- |
| **typedef** struct MyLinkedList**{**  int val**;**//当前节点的值  struct MyLinkedList**\*** next**;**//下一个节点的地址  **}** MyLinkedList**;** |

(1)初始化链表

动态分配地址，创建一个头节点，由于初始化链表，头节点的下一个节点的地址为空；头节点只是用来表示链表，无含义也就没有值。

|  |
| --- |
| MyLinkedList**\*** myLinkedListCreate**()** **{**  MyLinkedList**\*** node **=** **(**MyLinkedList**\*)**malloc**(sizeof(**MyLinkedList**));**  node**->**next **=** **NULL;**  **return** node**;**  **}** |

(2)添加头节点

创建一个新的节点，动态分配地址给它，插入到链表的头部，函数的参数val为新节点的值，参数obj节点要添加到的链表。

|  |
| --- |
| void myLinkedListAddAtHead**(**MyLinkedList**\*** obj**,** int val**)** **{**  MyLinkedList**\*** head **=** **(**MyLinkedList**\*)**malloc**(sizeof(**MyLinkedList**));**  head**->**val **=** val**;**  head**->**next **=** obj**->**next**;**  obj**->**next **=** head**;**  **}** |

(3)插入节点

该函数有三个变量，其中obj为要插入的链表，index是要插入的位值的索引值，val为要插入节点的值。实现也非常简单，先判断索引值是否和法，不合法则返回，再判断索引值是否超过了该链表的长度，超过则返回，上述条件都不成立后，用一个for循环，从头节点出发找到索引值，修改索引值前一个节点下一个节点的地址，使其指向新增节点，新增节点的下一个节点的地址为其上一个节点原下一个节点的地址，插入进去即可。

|  |
| --- |
| void myLinkedListAddAtIndex**(**MyLinkedList**\*** obj**,** int index**,** int val**)** **{**  **if** **(**index**<**0**)return;**  MyLinkedList**\*** node **=** obj**;**  **for** **(**int i **=** 0**;** i**<**index**;** i**++){**  **if** **(**node**->**next **==** **NULL)return;**  **else** node **=** node**->**next**;**  **}**  MyLinkedList**\*** add **=** **(**MyLinkedList**\*)**malloc**(sizeof(**MyLinkedList**));**  add**->**val **=** val**;**  **if** **(**node**->**next **!=** **NULL)**add**->**next **=** node**->**next**;**  **else** add**->**next **=** **NULL;**  node**->**next **=** add**;**  **}** |

(4)删除节点

删除节点和插入节点的实现思路也是一样，先解释该函数参数的含义，obj为要修改的链表，index为要删除的链表的节点位置，先判断索引值是否和法，不合法则返回，再判断索引值是否超过了该链表的长度，超过则返回，从头节点出发找到索引值将对应的节点删除即可。

|  |
| --- |
| void myLinkedListDeleteAtIndex**(**MyLinkedList**\*** obj**,** int index**)** **{**  **if** **(**index**<**0**)return;**  MyLinkedList**\*** node **=** obj**;**  **for** **(**int i **=** 0**;** i**<**index**;** i**++){**  **if** **(**node**->**next **==** **NULL)return;**  **else** node **=** node**->**next**;**  **}**  **if** **(**node**->**next **==** **NULL)return;**  **else** node**->**next **=** node**->**next**->**next**;**  **}** |

（5）删除尾节点

就是把链表的最后一个节点给删除，实现方式就是把链表原最后一个节点的上一个节点的下一个节点的地址设为空。

|  |
| --- |
| void myLinkedListDeleteAtTail**(**MyLinkedList**\*** obj**)** **{**  MyLinkedList**\*** node **=** obj**;**  **if** **(**node**->**next **==** **NULL){** delete\_err\_count**++;return;** **}**//删除失败，记录删除失败的次数  **while** **(**node**->**next**->**next **!=** **NULL){**  node **=** node**->**next**;**  **}**  node**->**next **=** **NULL;**  **return;**  **}** |

（6）删除链表

用一个tmp节点类型的变量保存当前链表指向的节点下一个节点，释放该节点，然后让将新的当前节点的tmp,重复这样的操作指导，当前节点为空。

|  |
| --- |
| void myLinkedListFree**(**MyLinkedList**\*** obj**)** **{**  MyLinkedList**\*** node **=** obj**;**  MyLinkedList**\*** tmp**;**  **while** **(**node**){**  tmp **=** node**->**next**;**  free**(**node**);**  node **=** tmp**;}**  **}** |

（7）查找节点

先判断索引值是否和法，不合法则返回，再判断索引值是否超过了该链表的长度，超过则返回，用一个for循环从链表的头节点进行遍历，直到遍历到索引值次数为止，返回位于索引值节点的值。

|  |
| --- |
| int myLinkedListGet**(**MyLinkedList**\*** obj**,** int index**)** **{**  **if** **(**index**<**0**){**  **return** **false;**  **}**  MyLinkedList**\*** node **=** obj**;**  **for** **(**int i **=** 0**;** i **<=** index**;** i**++){**  **if** **(**node**->**next **==** **NULL){**  **return** **false;**  **}**  **else** node **=** node**->**next**;**  **}**  **return** node**->**val**;**  **}** |

（8）修改节点

先判断索引值是否和法，不合法则返回，再判断索引值是否超过了该链表的长度，超过则返回，用一个for循环从链表的头节点进行遍历，直到遍历到索引值次数为止，将索引值对应链表节点的信息改为想要该的信息即可。

|  |
| --- |
| void myLinkedListChange**(**MyLinkedList**\*** obj**,** int index**,** int val**){**  **if** **(**index**<**0**)return;**  MyLinkedList**\*** node **=** obj**->**next**;**  **for** **(**int i **=** 0**;** i**<**index**;** i**++){**  **if** **(**node **==** **NULL)return;**  **else** node **=** node**->**next**;**  **}**  node**->**val **=** val**;**  **}** |

（9）打印链表

用一个for循环从链表的第一个节点开始（注意不是头节点，头节点无值），每到一个节点打印其节点信息，再将当前节点换为原节点的下一个节点，直到节点的下一个节点的地址为空。

|  |
| --- |
| void printlist**(**MyLinkedList**\*** obj**)**  **{**  printf**(**"开始打印~~\n"**);**  **if** **(**obj**->**next **==** **NULL)**  **{**  printf**(**"hhh\t"**);**  **return;**  **}**  MyLinkedList**\*** cur **=** obj**->**next**;**  **while** **(**cur **!=** **NULL){**  printf**(**"%d\t"**,** cur**->**val**);**  cur **=** cur**->**next**;**  **}**  printf**(**"\n打印完毕\n"**);**  **return;**  **}** |

（10）统计当前链表节点数量

定义一个整数类型的变量统计节点数，初始值为0；用一个for循环从链表的第一个节点开始（注意不是头节点，头节点无值），每到一个节点，节点数加一，再将当前节点换为原节点的下一个节点，直到节点的下一个节点的地址为空。打印节点数。

|  |
| --- |
| void countnodes**(**MyLinkedList**\*** obj**)**  **{**  int count **=** 0**;**  MyLinkedList**\*** cur **=** obj**;**  **while** **(**cur**->**next **!=** **NULL){**  count**++;**  cur **=** cur**->**next**;**  **}**  g\_count**=**count**;**  printf**(**"【--count--】当前节点数量:%d\n"**,** count**);**  **}** |

## 3.2读者模型

（1）统计节点数量

要进行统计节点数量需要调用事先写好的统计节点的函数，由于统计节点操作为读者，在此操作之前，先增加读者数量，操作完成后，读者数量减一，用一个互斥量让读者数目改变时不会发生线程之间访问冲突，其中增加读者数量后要判断读者数量是否为一，如果为一就调用一个写者的互斥量让写者不能写操作，读者数量减少时判断读者数量是否为零，如果为零才释放写者的互斥量，让写者可以写。

|  |
| --- |
| void countnodes\_s**(**MyLinkedList**\*** obj**)** **{**  WaitForSingleObject**(**hMutex**,** **-**1**);**//使用互斥量，保证读者数量改变时互斥  readcount**++;**  **if** **(**readcount **==** 1**)**  **{**  WaitForSingleObject**(**RP\_Write**,** INFINITE**);**  **}**  ReleaseMutex**(**hMutex**);**//释放互斥信号  countnodes**(**obj**);**//读链表  WaitForSingleObject**(**hMutex**,** **-**1**);**  readcount**--;**  **if** **(**readcount **==** 0**)**  **{**  ReleaseSemaphore**(**RP\_Write**,** 1**,** **NULL);**//读者数量为零释放信号量，使得写者可以进行写操作  **}**  ReleaseMutex**(**hMutex**);** //释放互斥信号  **}** |

(2) 查找节点

要进行查找节点需要调用事先写好的查找节点的函数，由于统计节点操作为读者，在此操作之前，先增加读者数量，操作完成后，读者数量减一，用一个互斥量让读者数目改变时不会发生线程之间访问冲突，其中增加读者数量后要判断读者数量是否为一，如果为一就调用一个写者的互斥量让写者不能写操作，读者数量减少时判断读者数量是否为零，如果为零才释放写者的互斥量，让写者可以写。

|  |
| --- |
| void myLinkedListGet\_s**(**MyLinkedList**\*** obj**,** int index**){**  WaitForSingleObject**(**hMutex**,** **-**1**);**  readcount**++;**  **if** **(**readcount **==** 1**)**  **{**  WaitForSingleObject**(**RP\_Write**,** INFINITE**);**  **}**  ReleaseMutex**(**hMutex**);**  **if(**myLinkedListGet**(**obj**,** index**)){**  printf**(**"【--locate--】当前索引值为%d的值为%d\n"**,**index**,**myLinkedListGet**(**obj**,** index**));**  find\_count**++;**//统计查找成功次数  **}**  **else{**  printf**(**"【--locate--】查找失败~!\n"**);**  **}**  WaitForSingleObject**(**hMutex**,** **-**1**);**  readcount**--;**  **if** **(**readcount **==** 0**)**  **{**  ReleaseSemaphore**(**RP\_Write**,** 1**,** **NULL);**  **}**  ReleaseMutex**(**hMutex**);** //释放互斥信号  **}** |

## 3.3写者模型

（1）增加尾节点

写者模型相比读者模型更为简单，每要调用一个增加尾节点的函数之前先使用一个写者的互斥量，调用完成后释放写者的互斥量。

|  |
| --- |
| void myLinkedListAddAtTail\_s**(**MyLinkedList**\*** obj**,** int val**){**  WaitForSingleObject**(**RP\_Write**,** INFINITE**);**  myLinkedListAddAtTail**(**obj**,** val**);**  ReleaseSemaphore**(**RP\_Write**,** 1**,** **NULL);**  **}** |

（2）删除尾节点

同增加尾节点操作一样：每要调用一个删除尾节点的函数之前先使用一个写者的互斥量，调用完成后释放写者的互斥量。

|  |
| --- |
| void myLinkedListDeleteAtTail\_s**(**MyLinkedList**\*** obj**){**  WaitForSingleObject**(**RP\_Write**,** INFINITE**);**  myLinkedListDeleteAtTail**(**obj**);**  printf**(**"【！！！delete】正在删除节点\n"**);**  ReleaseSemaphore**(**RP\_Write**,** 1**,** **NULL);**  **}** |

# 4.关键技术与系统实现

## 4.1初始化

为了让测试的结果更加的有说服力和科学性，本程序用一个window自带库函数中的随机数生成函数代替人工输入数据，完成该系统自动化的实现，这里讲解一下该系统是怎实现它的。调用随机生成函数，生成随机数数组，for循环反复调用一个函数时用随机数进行条件判断；实现不用用户操作就能对程序中的函数自动进行测试，为了防止每次生成的随机数不一样，调用time()函数，部分代码如下：

（1）在调用函数之前，先引用其对应的头文件：

|  |
| --- |
| #include<stdlib.h>//头文件包含rand和srand函数，这些函数用来生成随机数  #include<time.h>//C/C++中的日期和时间头文件，用于需要时间方面的函数 |

（2）生成随机数：

调用time()函数让每次生成的随机数都不一样

|  |
| --- |
| srand**((**unsigned**)**time**(NULL));**//选取种子文件  int k**=**rand**()%**100000**\***100**;**//控制随机数的范围在10000000中 |

## 4.2选择菜单功能

通过设计一个选择菜单，用户可以更加方便的管理此程序，实现程序界面良好，操作简单的要求，让不懂计算机的人士也能轻松操作，也有利于后期维护管理其代码，其基本思路为先打印用户可以操作的功能信息，然后用一个do…while循环语句让用户输入指定指令，根据用户输入的值调用相应的测试函数。关键代码如下所示：

|  |
| --- |
| int main**()**  **{**  char ch**;**  **while** **(** **true** **)**  **{**  printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**  printf**(**" 1:验证各个数据结构的正确性\n"**)** **;**  printf**(**" 2:验证并发互斥\n"**)** **;**  printf**(**" 3:验证生产者消费者模型\n"**)** **;**  printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**  printf**(** "Enter your choice (1 or 2 or 3): "**);**  //如果输入信息不正确，继续输入  **do{**  ch **=** **(**char**)**\_getch**()** **;**  **}** **while(**ch **!=** '1' **&&** ch **!=** '2' **&&** ch **!=**'3'**);**  system**(**"cls"**)** **;** //清除控制台显示的信息  //选择1  **if(**ch **==** '1'**)**  **{**  printf**(**"------验证各个数据结构的正确性------\n"**);**  void TEST1**();**TEST1**();**  **}**  //选择2  **if(**ch **==** '2'**)**  **{**  printf**(**"2:验证并发互斥\n"**)** **;**  void TEST2**();**TEST2**();**  **}**  //选择3  **if(**ch **==** '3'**)**  **{**  printf**(**"3:验证生产者消费者模型\n"**)** **;**  void TEST3**();** TEST3**();**  **}**  **}**  **return** 0**;**  **}** |

## 4.3读者写者模型构建

为了便于理解其模型，现展示其伪代码实现方式，主要概述为每当读者者要读时，先增加读者数量，读操作完成后，读者数量减一，用一个互斥量让读者数目改变时不会发生线程之间访问冲突，其中增加读者数量后要判断读者数量是否为一，如果为一就调用一个写者的互斥量让写者不能写操作，读者数量减少时判断读者数量是否为零，如果为零才释放写者的互斥量，让写者可以写。写者要进行写时先使用一个写者的互斥量，调用完成后释放写者的互斥量。

|  |
| --- |
| semaphore rmutex=1, wmutex=1;  int readcount=0;  void reader(){  do{  wait(rmutex);  if(readcount == 0)wait(wmutex);  readcount++;  signal(rmutex);  ...  perform read operation;  ...  wait(rmutex);  readcount--;  if(readcount == 0)singnal(wmutex);  signal(rmutex);  }while(true);  }    void writer(){  do{  wait(wmutex);  perform write operation;  signal(wmutex);  }while(true); |

读者写者模型验证：

实现思路是对每一个进入系统的读者和写者分别创建一个线程，让每个线程多次调用一个读者函数语句，实现读者或者写者同时多次对链表进行访问和修改。

实现代码如下：

1. 首先分别构建读者和写者测试函数

下面增加节点为读者，为实现每次验证读者写者模型是否正确，将传进的多个参数用结构体封装，由于传来的随机数很大，即每次插入删除节点的次数很多，为了便于显示每进行安全性增加节点的操作是否有效进行，每50次在操作前后分别打印其要准备增加节点的信息和增加节点完成的信息。

|  |
| --- |
| //增加节点n次  unsigned \_\_stdcall threadTest3**(**void**\*** my\_para**)**  **{**  Arg **\***a**;**//为了让读者读的数随机，线程传的参数是一个结构体，即所有要传的参数打包在一个结构体中，实现多个参数的传递  a**=(**Arg**\*)**my\_para**;**//将无符号参数强制转换为结构体的格式  int T**=**a**->**num**;**//要增加节点的个数  MyLinkedList**\*** my\_list **=** a**->**my\_list**;**//要修改的链表  **for** **(**int t **=** 0**;** t**<**T**;** t**++)**  **{**  **if** **(**t **%** 50 **==** 0**)** **{**  printf**(**"【start\_ADD】第 %d 次准备增加结点了\n"**,** t **+** 1**);**  **}**  myLinkedListAddAtTail\_s**(**my\_list**,** t**);**  **if** **(**t **%** 50 **==** 0**)** **{**  printf**(**"【end\_ADD】第 %d 次增加结点已完成\n"**,** t **+** 1**);**  **}**  **}**  **return** 0**;**  **}** |

同增加节点一样的处理方式，在进行统计节点的前后打印其执行操作的情况

|  |
| --- |
| //统计节点n次  unsigned \_\_stdcall threadTest4**(**void**\*** my\_para**)**  **{**  Arg **\***a**;**  a**=(**Arg**\*)**my\_para**;**  int T**=**a**->**num**;**  MyLinkedList**\*** my\_list **=** a**->**my\_list**;**  **for** **(**int t **=** 0**;** t**<(**T**/**10**);** t**++)**  **{**  printf**(**"【count】申请读节点\n"**);**  countnodes\_s**(**my\_list**);**  printf**(**"【count】统计节点结束\n"**);**  **}**  **return** 0**;**  **}** |

再起一个统计节点n次线程，增加其程序的说服力。

|  |
| --- |
| //统计节点n次  unsigned \_\_stdcall threadTest5**(**void**\*** my\_para**)**  **{**  Arg **\***a**;**  a**=(**Arg**\*)**my\_para**;**  int T**=**a**->**num**;**  MyLinkedList**\*** my\_list **=** a**->**my\_list**;**  **for** **(**int t **=** 0**;** t **<** **(**T**/**10**);** t**++)**  **{**  printf**(**"【s\_count】申请读节点\n"**);**  countnodes\_s**(**my\_list**);**  printf**(**"【e\_count】统计节点结束\n"**);**  **}**  **return** 0**;**  **}** |

|  |
| --- |
| //查找节点n次  unsigned \_\_stdcall threadTest6**(**void**\*** my\_para**)**  **{**  Arg **\***a**;**  a**=(**Arg**\*)**my\_para**;**  int T**=**a**->**num**;**  MyLinkedList**\*** my\_list **=** a**->**my\_list**;**  **for** **(**int t **=** 0**;** t **<** **(**T**/**10**);** t**++)**  **{**  **if(**t**%**5**==**0**){**  printf**(**"【s\_locate】第%d次查找节点开始执行了！\n"**,**t**);**  **}**  myLinkedListGet\_s**(**my\_list**,** t**);**  Sleep**(**10**);**  **if(**t**%**5**==**0**){**  printf**(**"【e\_locate】第%d次查找执行结束了!\n"**,**t**);**  **}**  **}**  **return** 0**;**  **}** |

|  |
| --- |
| //删除节点n次  unsigned \_\_stdcall threadTest7**(**void**\*** my\_para**)**  **{**    Arg **\***a**;**  a**=(**Arg**\*)**my\_para**;**  int T**=**a**->**num**;**  MyLinkedList**\*** my\_list **=** a**->**my\_list**;**  **for** **(**int t **=** 0**;** t **<** T**;** t**++)**  **{**  printf**(**"【s\_delete】第%d次申请删除节点~~~\n"**,**t**+**1**);**  myLinkedListDeleteAtTail\_s**(**my\_list**);**  printf**(**"【e\_delete】第%d次删除节点成功！\n"**,**t**+**1**);**  **}**  **return** 0**;**  **}** |

2)构建好测试函数过后，就可用一个验证读者写者模型的主测试函数进行封装。

这个主测试函数的设计思想就是，先定义要用到的互斥量和信号量，再随机生成五个随机数用于传给不同的线程，打印计划增加节点数和计划删除节点数，这些值均为生成的随机数的值，启用5个线程，让他们并发执行，当他们全部执行结束后，打印删除失败的节点数，查找成功的节点数，理想节点数，当前节点数，如果当前节点数-删除失败节点数=理想节点数则证明且查找成功的次数小于设置的查找次数的值，则证明构造的读者写者模型准确。

|  |
| --- |
| void TEST3**()**  **{**  MyLinkedList**\*** my\_list **=** myLinkedListCreate**();**  HANDLE hThread**[**5**];**  hMutex **=** CreateMutex**(NULL,** FALSE**,** **NULL);**  RP\_Write **=** CreateSemaphore**(NULL,** 1**,** 1**,** **NULL);**    int k**[**5**];**  **for(**int i**=**0**;**i**<**5**;**i**++)**  **{**  srand**((**unsigned**)**time**(NULL));**//选取种子文件  k**[**i**]=**rand**()%**1000**;**//控制随机数的范围在1000内  **}**  printf**(**"计划增加节点%d\n"**,**k**[**0**]);**  printf**(**"计划删除节点%d\n"**,**k**[**4**]);**  Arg arg**[**5**];**  **for(**int j**=**0**;**j**<**5**;**j**++){**  arg**[**j**].**my\_list**=**my\_list**;**  arg**[**j**].**num**=**k**[**j**];**  **}**  hThread**[**0**]** **=** **(**HANDLE**)**\_beginthreadex**(NULL,** 0**,** threadTest3**,** **(**void**\*)&**arg**[**0**],** 0**,** **NULL);**//增加节点  hThread**[**1**]** **=** **(**HANDLE**)**\_beginthreadex**(NULL,** 0**,** threadTest4**,** **(**void**\*)&**arg**[**1**],** 0**,** **NULL);**//查看节点数量  hThread**[**2**]** **=** **(**HANDLE**)**\_beginthreadex**(NULL,** 0**,** threadTest5**,** **(**void**\*)&**arg**[**2**],** 0**,** **NULL);**//查看节点数量  hThread**[**3**]** **=** **(**HANDLE**)**\_beginthreadex**(NULL,** 0**,** threadTest6**,** **(**void**\*)&**arg**[**3**],** 0**,** **NULL);**//查找节点  hThread**[**4**]** **=** **(**HANDLE**)**\_beginthreadex**(NULL,** 0**,** threadTest7**,** **(**void**\*)&**arg**[**4**],** 0**,** **NULL);**//删除节点  WaitForSingleObject**(**hThread**[**0**],** **-**1**);**  WaitForSingleObject**(**hThread**[**1**],** **-**1**);**  WaitForSingleObject**(**hThread**[**2**],** **-**1**);**  WaitForSingleObject**(**hThread**[**3**],** **-**1**);**  WaitForSingleObject**(**hThread**[**4**],** **-**1**);**  CloseHandle**(**hThread**[**0**]);**  CloseHandle**(**hThread**[**1**]);**  CloseHandle**(**hThread**[**2**]);**  CloseHandle**(**hThread**[**3**]);**  CloseHandle**(**hThread**[**4**]);**  CloseHandle**(**hMutex**);**  CloseHandle**(**RP\_Write**);**  printf**(**"\*\*\*\*\*delete\_err:%d\*\*\*\*\*\*\*\*\n"**,**delete\_err\_count**);**  printf**(**"\*\*\*\*\*findsuccess:%d\*\*\*\*\*\*\*\*\n"**,**find\_count**);**  countnodes**(**my\_list**);**  printf**(**"理想节点数量：%d\n"**,(**k**[**4**]-**k**[**0**]));**  myLinkedListFree**(**my\_list**);**  delete\_err\_count**=**0**;**  **}** |

## 4.4线程同步互斥的实现

伪代码实现思路为在每个线程进行要对临界资源进行访问时，用互斥量对临界资源锁住，使其只能由它操作，操作完成后释放互斥量，其他线程才可以动它。

|  |
| --- |
| semaphore mutex = 1;  Pa(){  while(1){  wait(mutex)；  临界区；  signal(mutex);  剩余区；  }  }    Pb(){  while(1){  wait(mutex)；  临界区；  signal(mutex);  剩余区；  }  } |

系统代码实现：

写一个多次在链表的头部增加或者删除节点的线程，每次进行多次插入或者删除之前，随机生成一个数打印要增加或者删除节点的个数，在每次进行插入或者删除节点之时用上述思想用一个互斥量对其操作进行保护。

|  |
| --- |
| //多次在头部增加节点  unsigned \_\_stdcall threadTest1**(**void**\*** my\_para**)**  **{**  MyLinkedList**\*** my\_list **=** **(**MyLinkedList**\*)**my\_para**;**  srand**((**unsigned**)**time**(NULL));**//选取种子文件  int k**=**rand**()%**100000**\***100**;**//控制随机数的范围在10000000中  printf**(**"要增加的节点数量:%d\n"**,**k**);**  printf**(**"正在增加节点处理中，请稍等。。。。\n"**);**  **for** **(**int i **=** 0**;** i**<**k**;** i**++)**  **{**  WaitForSingleObject**(**hMutex**,** **-**1**);**  myLinkedListAddAtHead**(**my\_list**,** i**);**  ReleaseMutex**(**hMutex**);**  **}**  **return** 0**;**  **}** |

|  |
| --- |
| //多次删除尾节点  unsigned \_\_stdcall threadTest2**(**void**\*** my\_para**)**  **{**  MyLinkedList**\*** my\_list **=** **(**MyLinkedList**\*)**my\_para**;**  srand**((**unsigned**)**time**(NULL));**//选取种子文件  int s**=**rand**()%**100000**\***100**;**//%100控制随机数的范围  printf**(**"要删除的节点个数：%d\n"**,**s**);**  printf**(**"正在删除节点，请稍等。。。。\n"**);**  **for** **(**int j **=** 0**;** j**<**s**;** j**++)**  **{**  WaitForSingleObject**(**hMutex**,** **-**1**);**  myLinkedListDeleteAtTail**(**my\_list**);**//删除尾节点  ReleaseMutex**(**hMutex**);**  **}**  **return** 0**;**  **}** |

构建完成后就让两个线程同时在测试函数中运行，两个线程完成后，打印当前链表的节点个数，如果当前节点数减去删除失败节点数等于计划增加节点数减去计划删除节点数，即可验证两个线程互斥并发。

|  |
| --- |
| void TEST2**()**  **{**  MyLinkedList**\*** my\_list **=** myLinkedListCreate**();**  HANDLE hThread**[**2**];**  hMutex **=** CreateMutex**(NULL,** FALSE**,** **NULL);**  hThread**[**0**]=(**HANDLE**)**\_beginthreadex**(NULL,**0**,**threadTest1**,**my\_list**,**0**,NULL);**  hThread**[**1**]=(**HANDLE**)**\_beginthreadex**(NULL,**0**,**threadTest2**,**my\_list**,**0**,NULL);**  WaitForSingleObject**(**hThread**[**0**],-**1**);**  WaitForSingleObject**(**hThread**[**1**],-**1**);**  CloseHandle**(**hThread**[**0**]);**  CloseHandle**(**hThread**[**1**]);**  CloseHandle**(**hMutex**);**  printf**(**"删除失败次数:%d\n"**,**delete\_err\_count**);**  countnodes**(**my\_list**);**  myLinkedListFree**(**my\_list**);**  delete\_err\_count**=**0**;**  **}** |

## 4.5验证数据结构的准确性

实现思路是每调用一个数据结构对链表进行访问或者修改后就对链表进行打印，由于篇幅限制，且大部分代码实现思路一致，这里列出部分实现代码，该代码的含义时，先创建一个链表，生成五个随机数并打印，打印要插入节点的次数，执行操作，并打印链表每个节点的信息，可以验证其定义好的函数是否正确。

|  |
| --- |
| void TEST1**()** **{**  MyLinkedList**\*** my\_list **=** myLinkedListCreate**();**//创建链表  int n**[**5**]={**0**};**//生成五个随机数  **for(**int a **=**0**;**a**<**5**;**a**++){**  n**[**a**]=**rand**()%**100**;**  printf**(**"%d\t"**,**n**[**a**]);**  **}**//生成随机数    //验证头部插入函数的正确性  printf**(**"在头部插入节点%d次.......\n"**,**n**[**4**]+**10**);**  **for** **(**int i **=** 0**;** i **<**n**[**4**]+**10 **;** i**++)**  **{**  myLinkedListAddAtHead**(**my\_list**,** i**+**1**);**  **}**  //打印链表  printlist**(**my\_list**);**  **。。。。。。**  //统计节点数  countnodes**(**my\_list**);**  myLinkedListFree**(**my\_list**);**  **}** |

## 4.6为线程传递多个参数的实现

为了实现验证读者写者模型时，每次测试时，多个读者同时写或者多个读者同时读的次数都不样，需要将生成的随机数传递给其相应线程当中去，因此在个线程传参时不仅要传递链表的指针还要传递随机数，那么就必须对多个参数用一个结构体封装，传递参数时化为无符号型，传到线程当中去后在再将其强制转换成结构体。

下面请看代码实现：

1）先在test3（）函数（验证读者写者模型主函数）中生成多个随机数，生成的多个数用一个数组来表示

|  |
| --- |
| int k**[**5**];**//随机数数组  **for(**int i**=**0**;**i**<**5**;**i**++)**  **{**  srand**((**unsigned**)**time**(NULL));**//选取种子文件  k**[**i**]=**rand**()%**1000**;**//控制随机数的范围在1000内  **}** |

2）创建一个参数结构体，专门用来对线程的传参

|  |
| --- |
| struct Arg  **{**  MyLinkedList**\*** my\_list**;//链表指针**  int num**;//随机数**  **};** |

3）创建一个结构体数组，给每一个结构体赋予不同的随机数

|  |
| --- |
| Arg arg**[**5**];**  **for(**int j**=**0**;**j**<**5**;**j**++){**  arg**[**j**].**my\_list**=**my\_list**;**  arg**[**j**].**num**=**k**[**j**];**  **}** |

4）将结构体参数传递给线程

|  |
| --- |
| hThread**[**0**]** **=** **(**HANDLE**)**\_beginthreadex**(NULL,** 0**,** threadTest3**,** **(**void**\*)&**arg**[**0**],** 0**,** **NULL);**//增加节点 |

5）线程对传来的结构体参数进行处理和调用

|  |
| --- |
| //增加节点n次  unsigned \_\_stdcall threadTest3**(**void**\*** my\_para**)**  **{**  Arg **\***a**;**//为了让读者读的数随机，线程传的参数是一个结构体，即所有要传的参数打包在一个结构体中，实现多个参数的传递  a**=(**Arg**\*)**my\_para**;**//将无符号参数强制转换为结构体的格式  int T**=**a**->**num**;**//要增加节点的个数  MyLinkedList**\*** my\_list **=** a**->**my\_list**;**//要修改的链表  **for** **(**int t **=** 0**;** t**<**T**;** t**++)**  **{**  **if** **(**t **%** 50 **==** 0**)** **{**//没  printf**(**"【start\_ADD】第 %d 次准备增加结点了\n"**,** t **+** 1**);**  **}**  myLinkedListAddAtTail\_s**(**my\_list**,** t**);**  **if** **(**t **%** 50 **==** 0**)** **{**  printf**(**"【end\_ADD】第 %d 次增加结点已完成\n"**,** t **+** 1**);**  **}**  **}**  **return** 0**;**  **}** |

# 5.系统运行结果

## 5.1运行环境

|  |  |
| --- | --- |
| 编程工具 | Visual Studio 2019 |
| 编程语言 | C语言 |
| 演示系统 | Windows·10 家庭版 |
| 电脑配置 |  |

## 5.2运行与测试结果分析

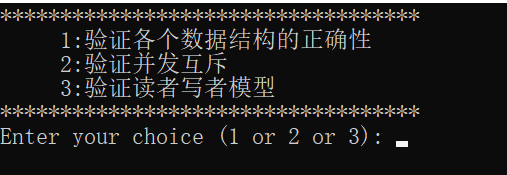
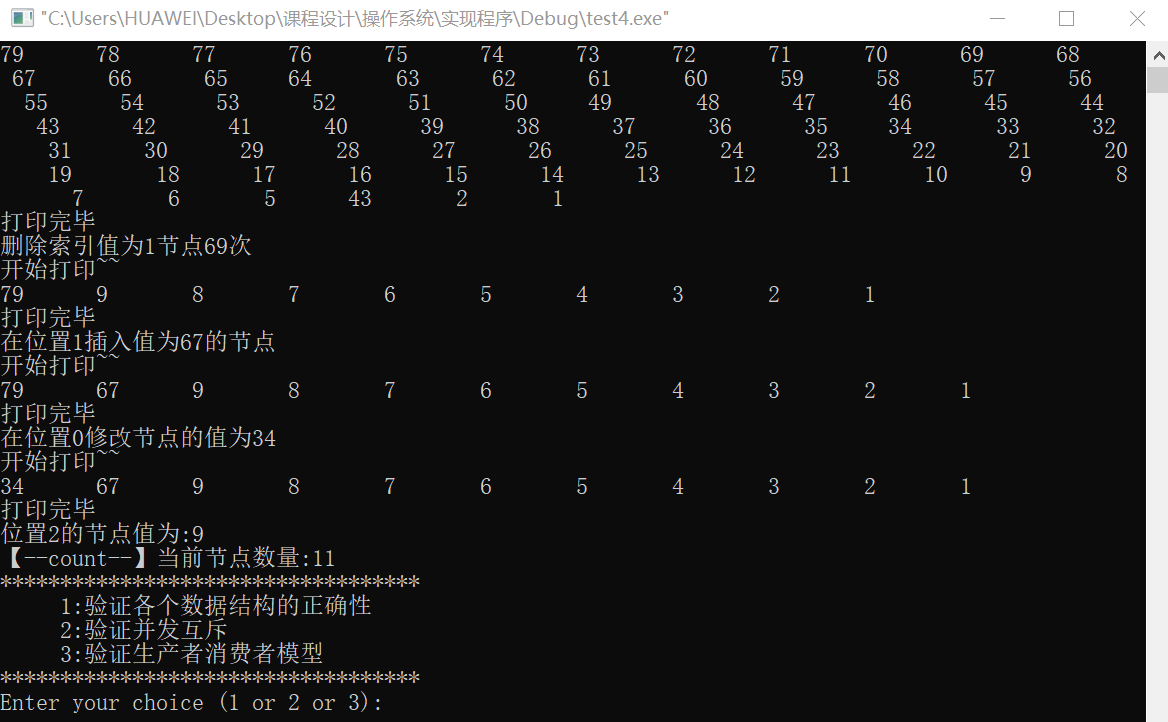
1. 主窗口

图4.4.1 主窗口图

2、键入1，验证数据结构正确性

测试用例1

图4.4.2 验证数据结构正确性示意图

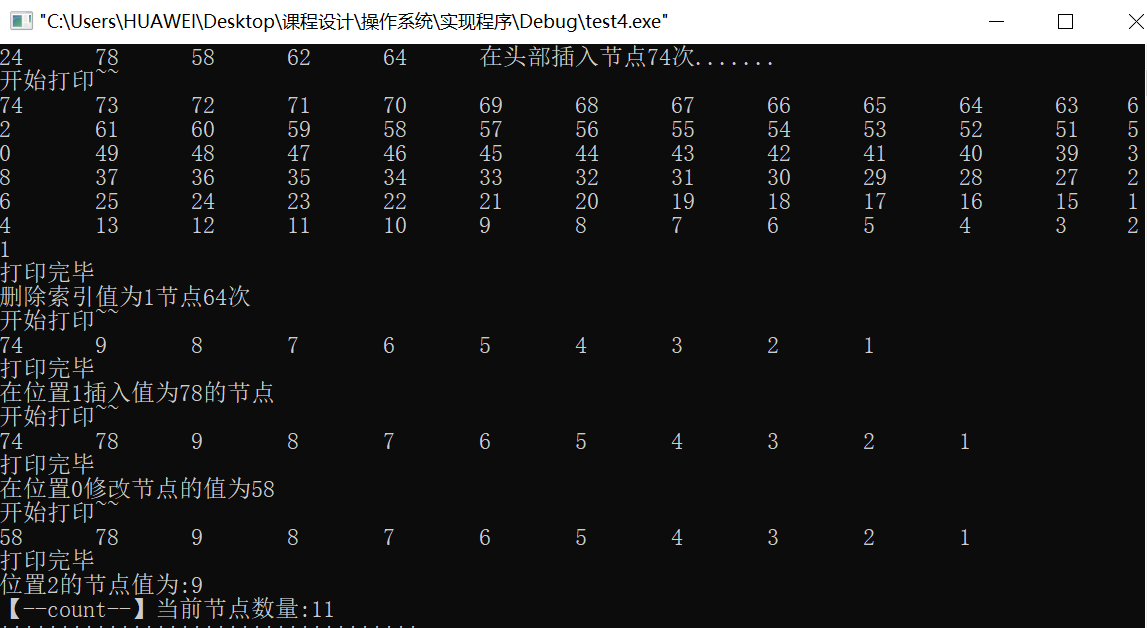
测试用例2

图4.4.3 验证数据结构正确性示意图

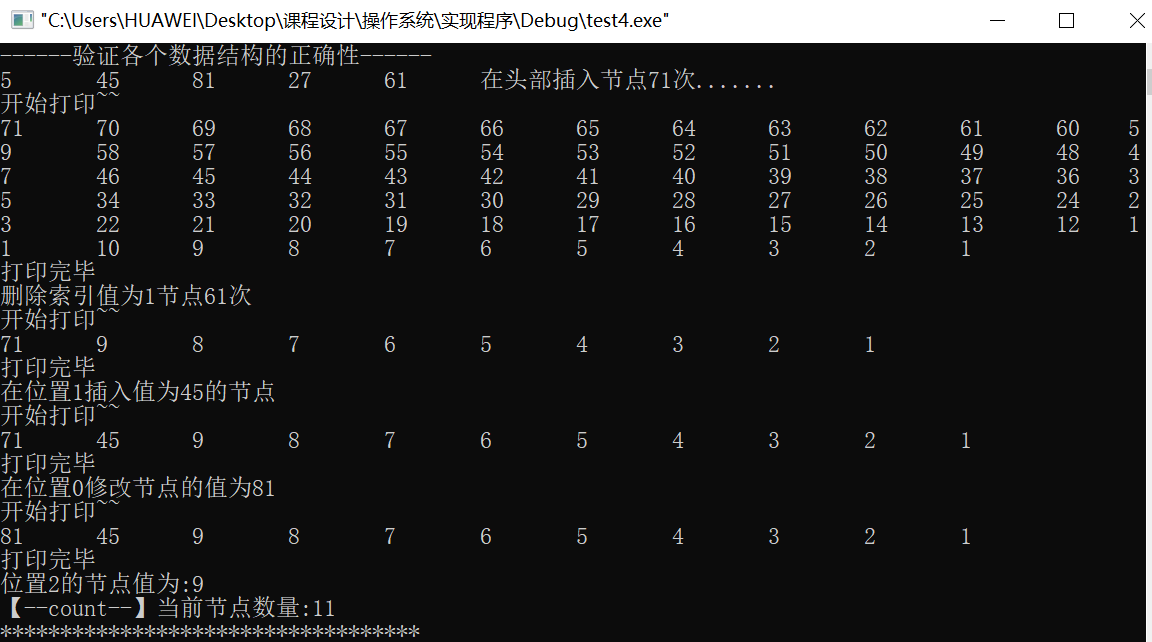
测试用例3

图4.4.4 验证数据结构正确性示意图

这里解释一下图4.4.5的含义，首先看到的五个数是随机生成的，调用第五个随机数加上10其值为插入节点的次数，这里要插入的节点数是71次，可以看出打印的链表有71个节点，插入节点的函数正确；删除节点61次，打印的链表的节点数正好为开始的71次减去61次的值，删除节点函数正确；在位置一中插入值为45的节点，打印的链表正好在位置插入了一个值为45的节点，插入节点函数正确，在位置0修改其值为81，位置为2的节点为9，当前节点数量有11，有打印的链表所示，修改节点函数正确，查找节点函数正确，统计节点数量正确。

3、键入2，验证并发互斥

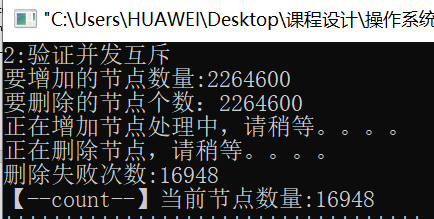


图4.4.5 验证并发互斥测试图1

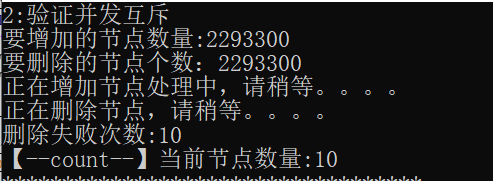


图4.4.6 验证并发互斥测试图2

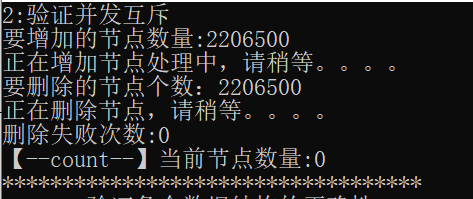


图4.4.7 验证并发互斥测试图3

这里解释一下上述图的含义，以图4.4.7为例，要增加的节点数为2206500 次，删除节点数为2206500次，删除失败次数为0次，因当前当前节点数减去删除失败节点数等于计划增加节点数减去计划删除节点数（0-0=2206500-2206500），并发互斥正确。

4、键入3，验证读者写者模型

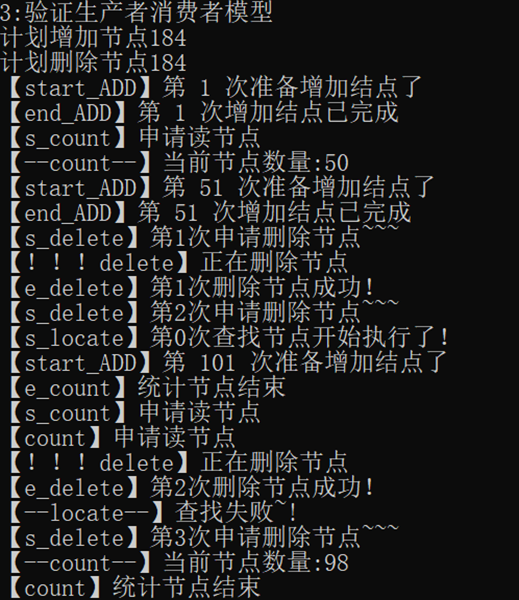


图4.4.8验证读者写者模型测试图1

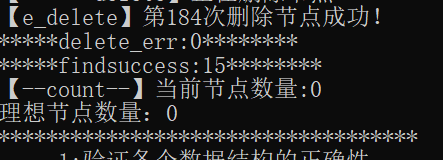


图4.4.9验证读者写者模型测试图2

由图4.4.8和图4.4.9所示，增加节点，删除节点，查找节点，统计节点，操作同时进行，但是最后的结果理想节点0等于当前节点数量0减去删除失败的个数0，说明读者写者模型是正确的。

# 6.调试和改进

在调试过程中，验证并发互斥的情况，为了让数据更加准确，特设置系统删除或者增加节点多次，但是由于程序会进行大量的读写操作，导致出实验结果很慢，印象实验进度和效率，以期在算法中得以优化。验证读者写者模型实验打印的结果不是很直观，需要人工花一点时间对数据处理和分析，改进方向就是如何打印出更加直观的实验结构，证明实验研究对象。

# 7.心得和结论

由于时间的限制，以及本人精力的有限，本次课程设计依旧存在很多的不足，还有很多功能有待完善和改进，但此次课程设计实现过程非常的难忘，当第一次构建的数据结构能经过简单的测试运行起来的时候，非常的兴奋和激动，紧接着立刻用老师实验课讲的知识实现互斥并发，刚开始实现起来非常的快，但是由于对课程设计的要求没有彻底弄懂，在实现读者写者模型的时候就遇到了瓶颈，不知道如何实现它以及如何验证它，好在毕坤老师的耐心指导下，以及老师精心提供的参考资料支持下，自己便开始有了思路，一点点的去实现它，在次期间也学到了很多编程上的知识和小技巧，学会了如何传递在一个线程中同时传递多个参数，如何充分利用好库函数实现想要的功能，如何在网上检索有用的信息帮助自己理解和思考等等，学习的道理不是一番风顺的，学到了就值得，引用linux创始人的话作为此次课程设计的结束语吧“Talk is cheap, Show me the code! ”

# 8.结论和体会

1、熟悉巩固了数据结构的相关知识

2、对操作系统中并发互斥中有了更为深刻的理解

3、掌握了读者写者模型的精髓，在实现过程中一定要老牢记其前提，多个读者可以同时读，读者读时写者不能写，写者写时读者不能读。

4、在做课程设计的时候一定要有耐性，细心，有计划有方向的去实现，切记走马观花，也不能钻死脑经，要不懂就问，充分利用身边的学习资源，在这里非常感谢毕坤老师的指导和点播，如果没有老师的耐心指导，本次课程设计也不能向现在这样如期完成，

5、，善于利用老师给的学习资料认真揣摩研究。

6、细节决定成败，但要大胆写代码，根据调试信心充分对代码进行优化和排除错误。

# 9.进一步改进方向

将单向链表的读者写者模型中利用到实际案例中，如分块进行文件的传输或者利用到游戏当中。

# 主要参考文献

【1】严蔚敏. 数据结构C语言版[M]. 第二版. 清华大学出版社, 2012.

【2】汤小丹. 计算机操作系统[M]. 第四版. 西安电子科技大学出版社, 2014.