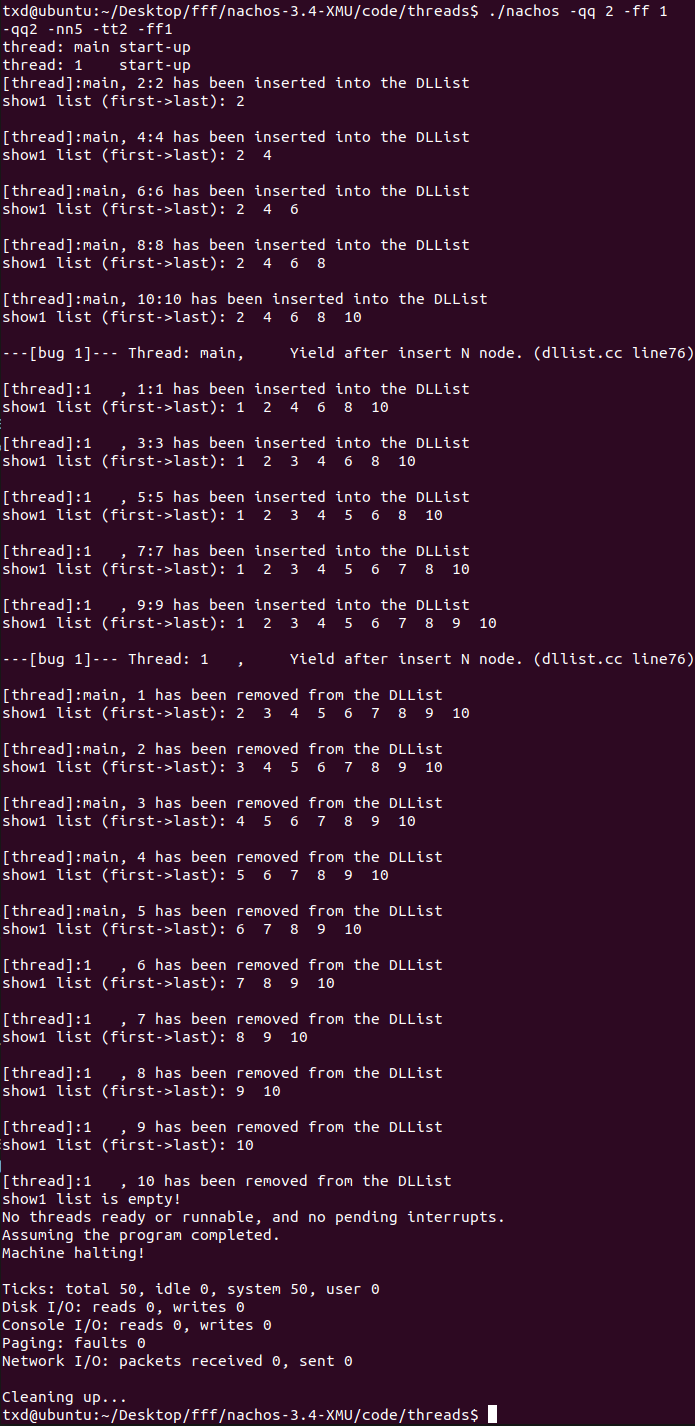
**bug1**

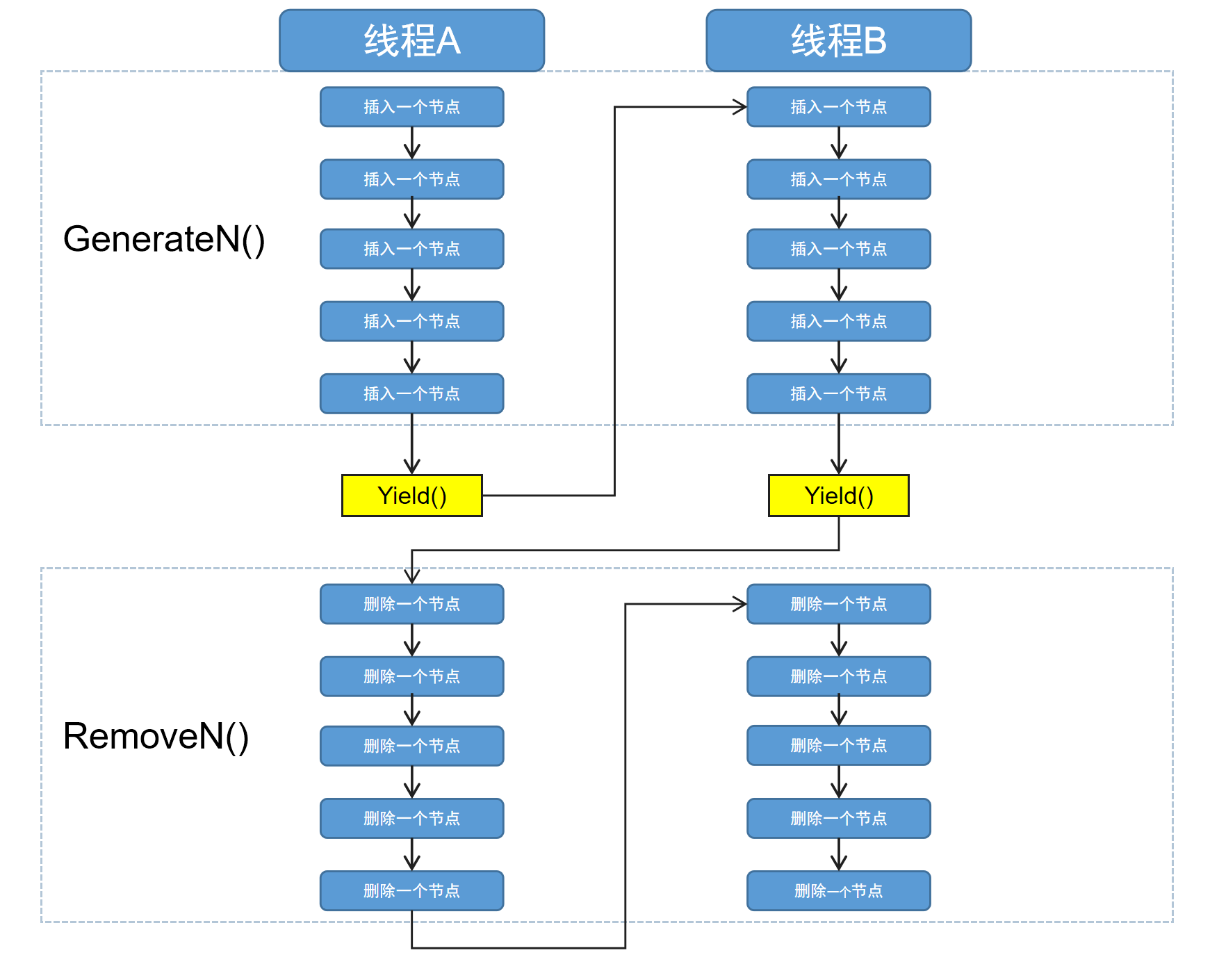
错误1.突出一个取出的数据可能是**乱序**的

比如存入的数据是1、2、3、4、5，取出的数据是5、6、7、8、9。



线程A输入偶数，输出12345

线程B输入奇数，输出678910



**bug2**

在删除first节点时，线程A使用remove程序先用A.element变量记录first地址，但还未执行删除操作时Yield()，此时，如果正好线程B也使用remove删除first节点，就会产生段错误。

错误1.段错误

因为线程B也删除first节点，用B.element记录下first地址，Yield()。线程A继续工作，成功删除first节点。之后轮到线程B工作时，它记录下的B.element此时已经是无意义的NULL，线程B试图访问NULL->next，报**段错误。**

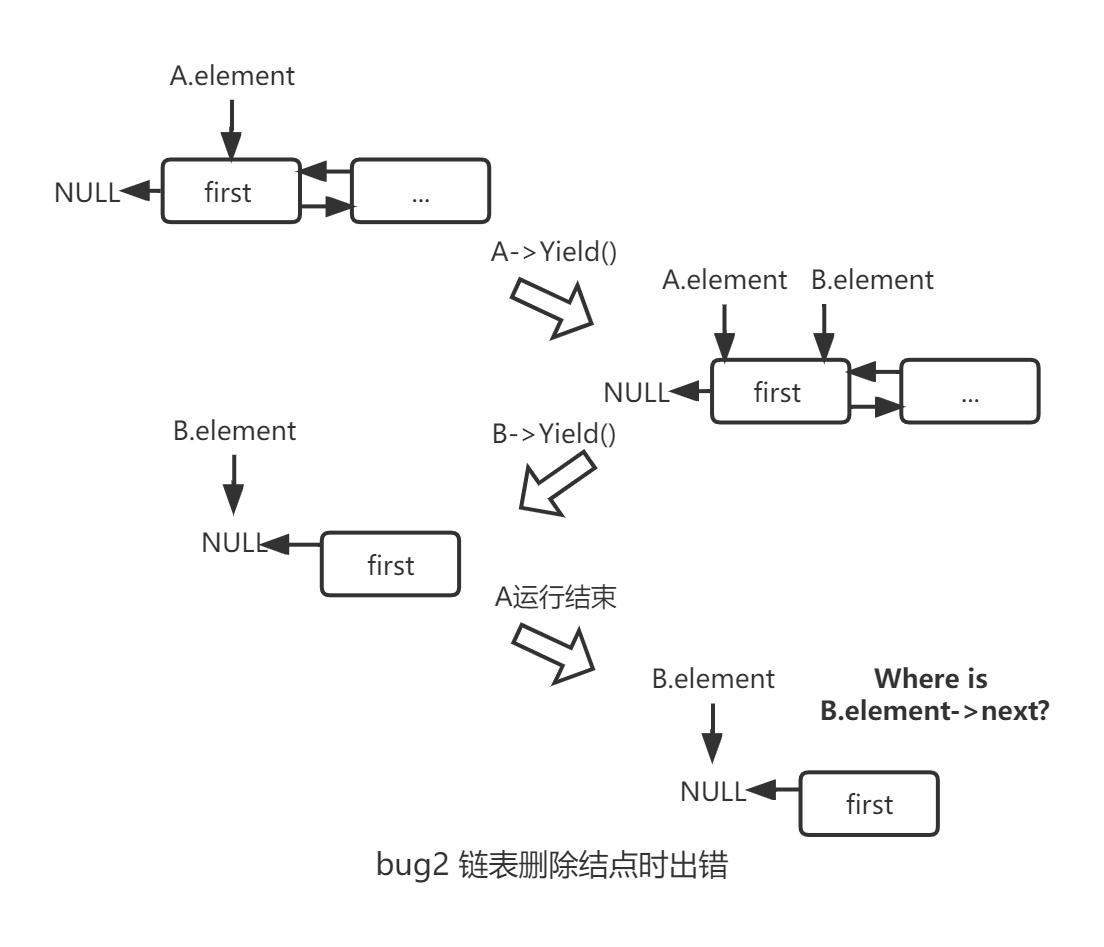
可使用序列：

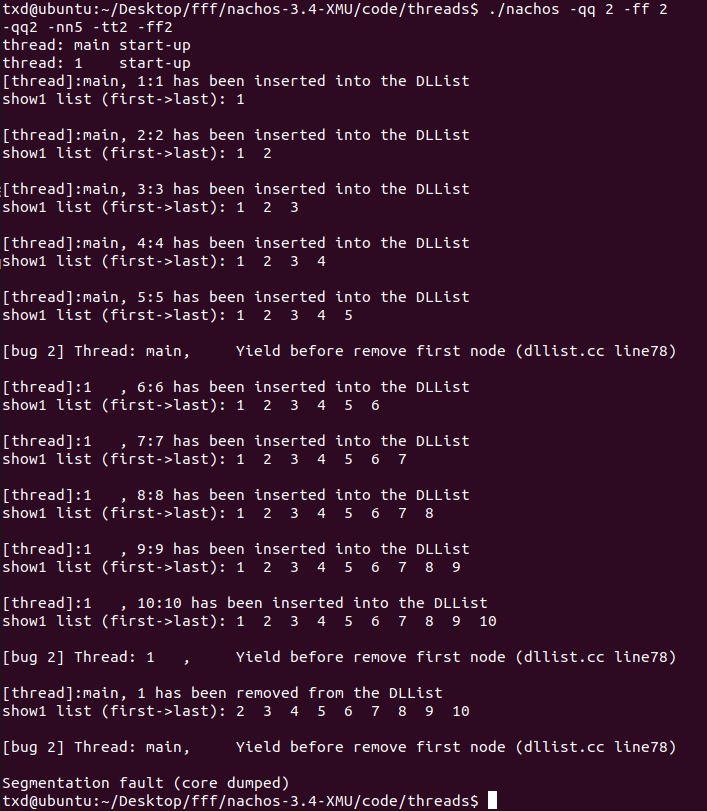
A：1、2、3、4、5

B：6、7、8、9、10

两个线程都要删“key=1”这个最前面的节点，

dllist-driver.cc中的删除函数RemoveN()，是从头删除N个函数，所以一旦以-ff 2系统运行，就一定会使用埋藏了bug的Remove函数。一定会报错。





实验结果中，两个线程都试图删除“1”节点。

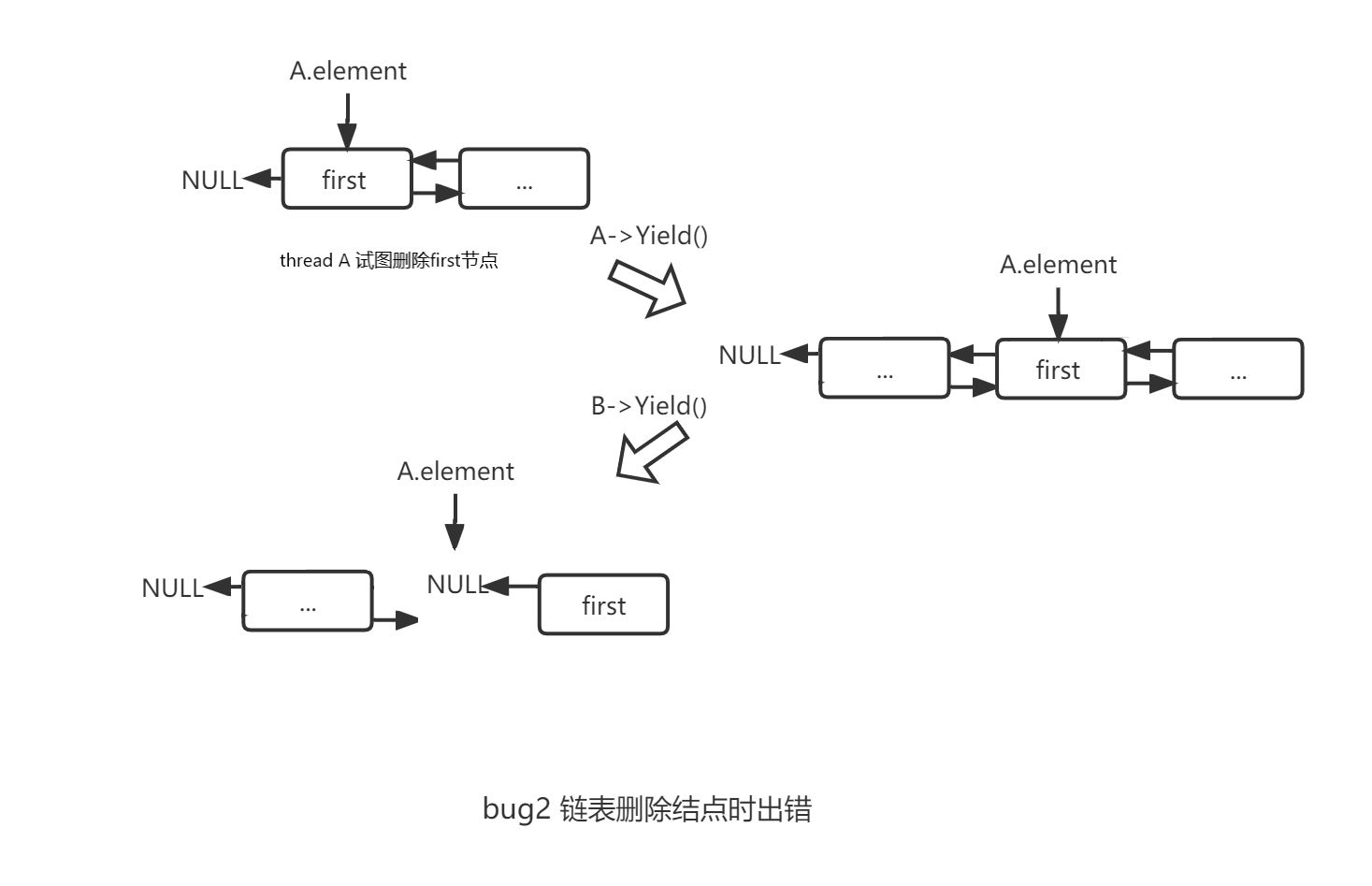
错误2.链表中间断

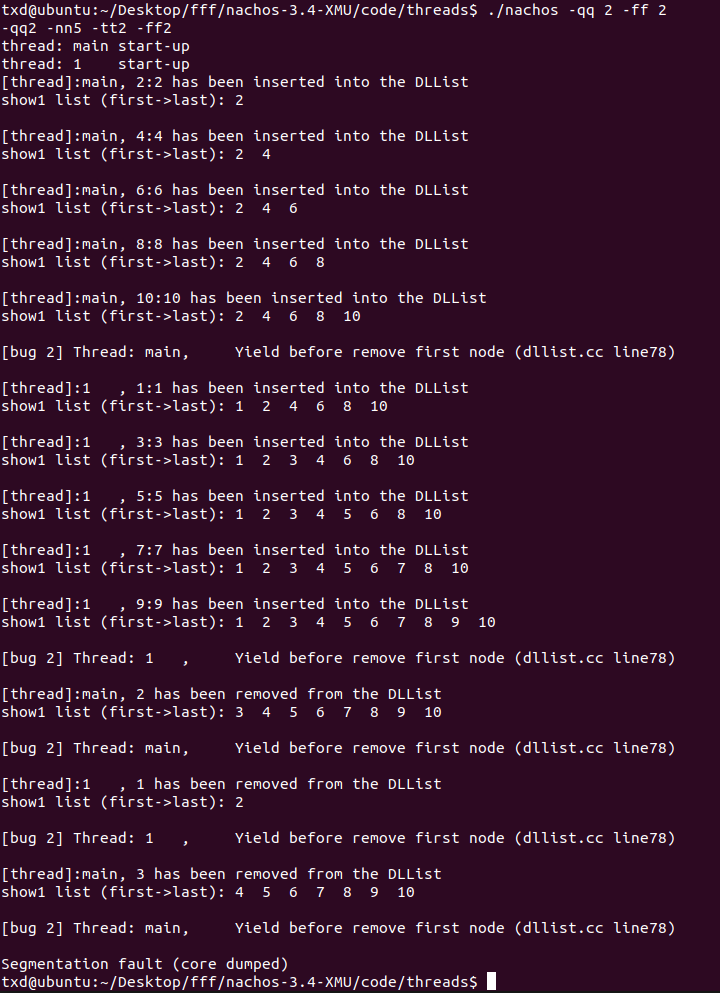
remove记录下要删除的节点的地址，刚要删，就Yield()，之后的线程往链表前插入了节点，之前要删的那个节点到了链表中端。remove恢复运行，把这个点删了，链表中间就断。

可使用序列：

A：小于等于10的偶数

B：小于等于10的奇数





在实验结果中，被删除的是“key=2”。它前面有1，后面有3，链表断。

**bug 3**

导致本来线程A要连续插3次，B再连续插3次。变成了各自交替插3次。导致删除的不是自己插入的。导致插入**延迟**，如下图，线程B在第6时就已经分配了新节点的空间，但是到第11时才成功修改next、prev，将节点插入链表中。

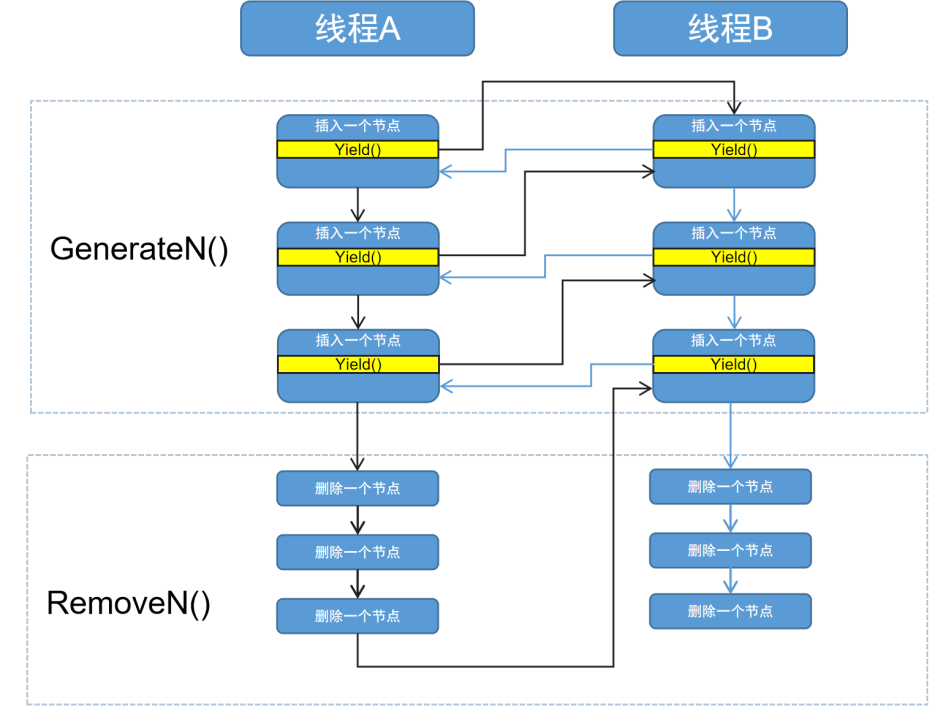
可使用序列：

A：1、2、3、4、5

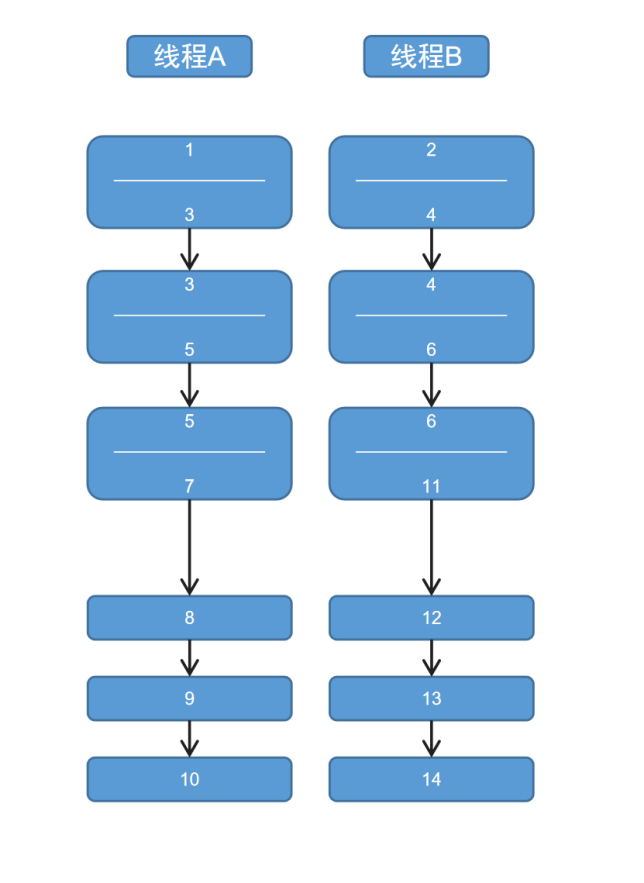
B：6、7、8、9、10

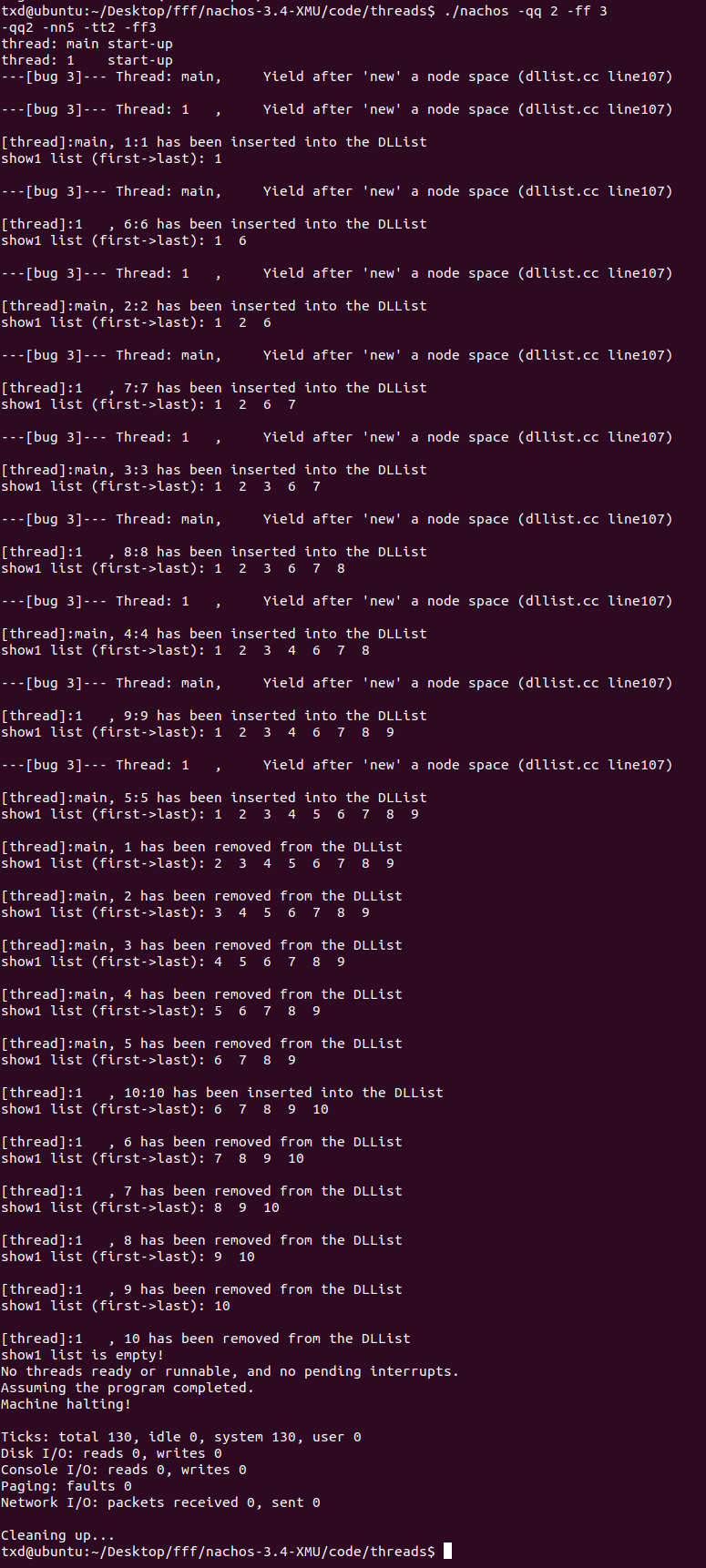
所有序列都会延迟

流程图：



时序图：





**BUG 4**

程序对空链表插入时会触发这个bug。但是这个bug是个地雷，你不去动他，就能相安无事。如果这个程序的随机数恰好没有一次是向链尾插入(要修改last)，这个bug就永远不会触发。

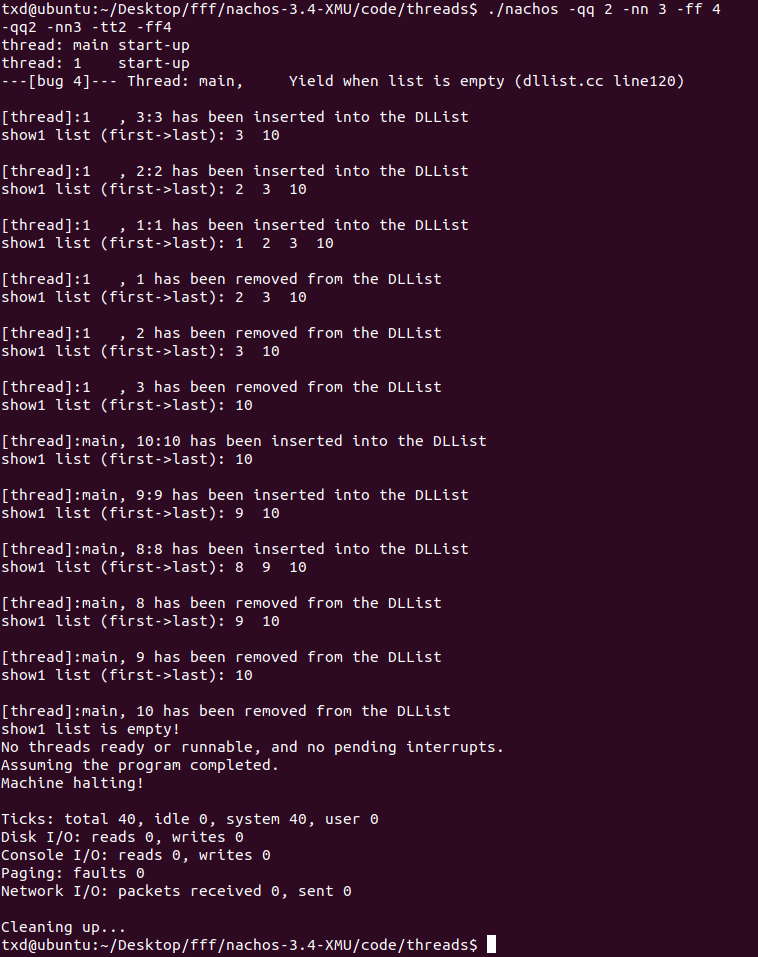
(插入链尾，会使用这个语句：last->next=element)

可使用序列：

不报错：

A：10，9，8

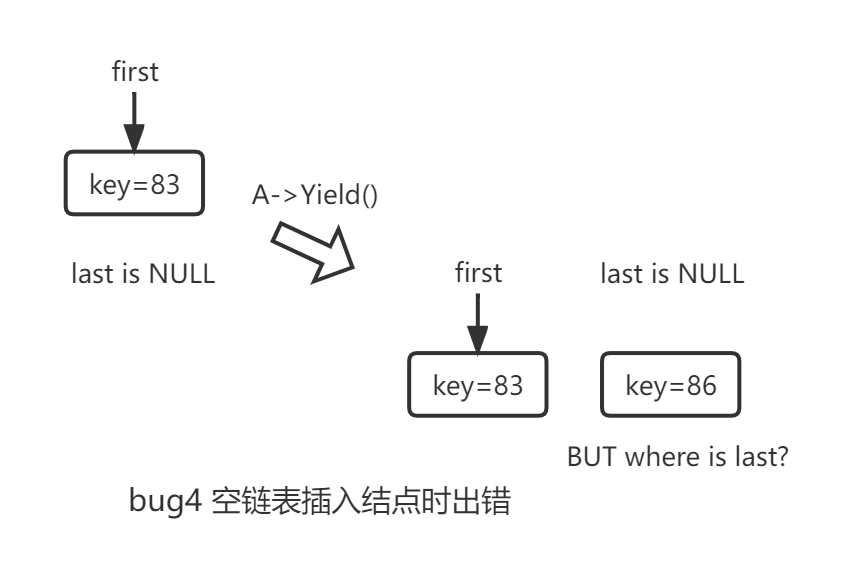
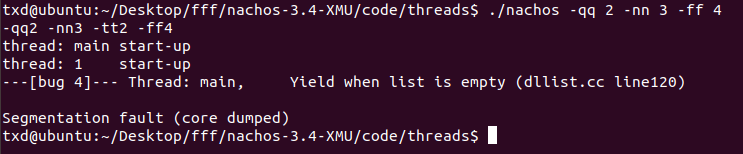
B：3，2，1



报错：

A：1，2

这就够了，会报错，段错误，因为last指针为NULL



**BUG 5**

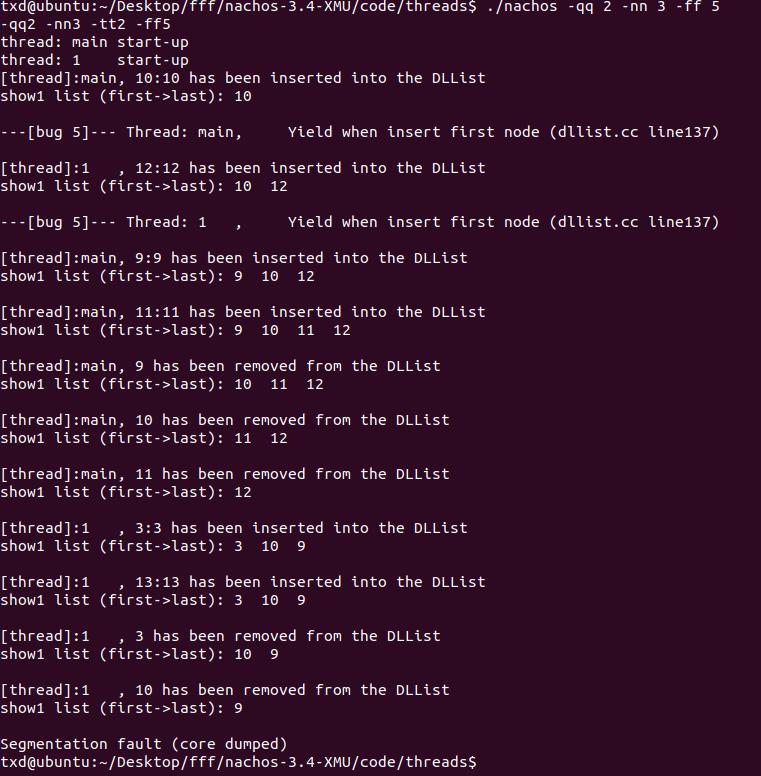
修改插入链表头的函数。导致链表中一个节点被屏蔽在整个链表外。

使用展示链表所有节点的函数无法显示它，最后函数删除节点时找不到它，就会报错。

可使用序列：

A：10，9，11

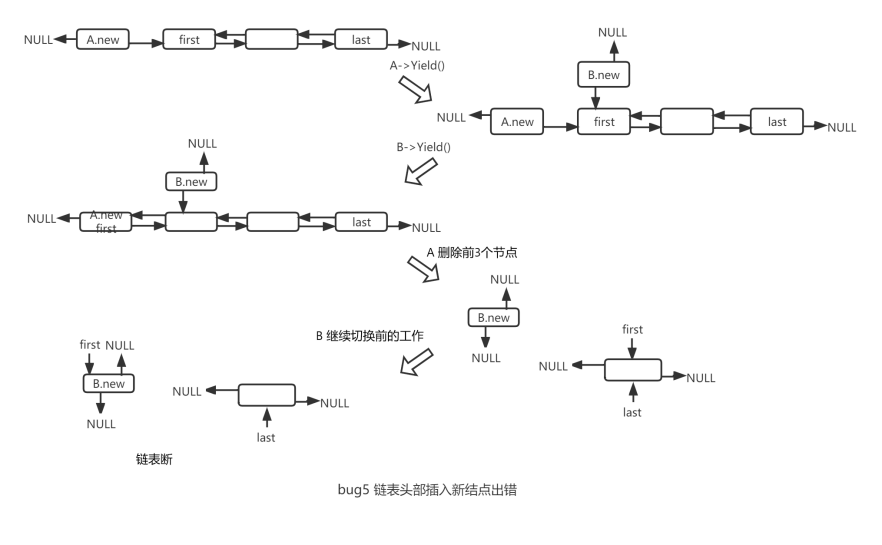
B：12，3，13



第一次序列显示3、10、9的时候已经出错了，因为3原本是插在10上的，线程A删除节点10的时候没有把内存空间重制为0，导致那个地址还是10，但是其他数据已经错乱了。10后面应该指11，这里却显示9。（如果系统把10节点回收时，重制其内存中存储的数据为NULL，那么就会报段错误）。第一次序列显示1、10、9时链表已经错乱了（断裂了）

3是first，12是last，中间没有链接。

所以下面这个链表图不是画错了，是画出了前半部份，没画出最后的结果。



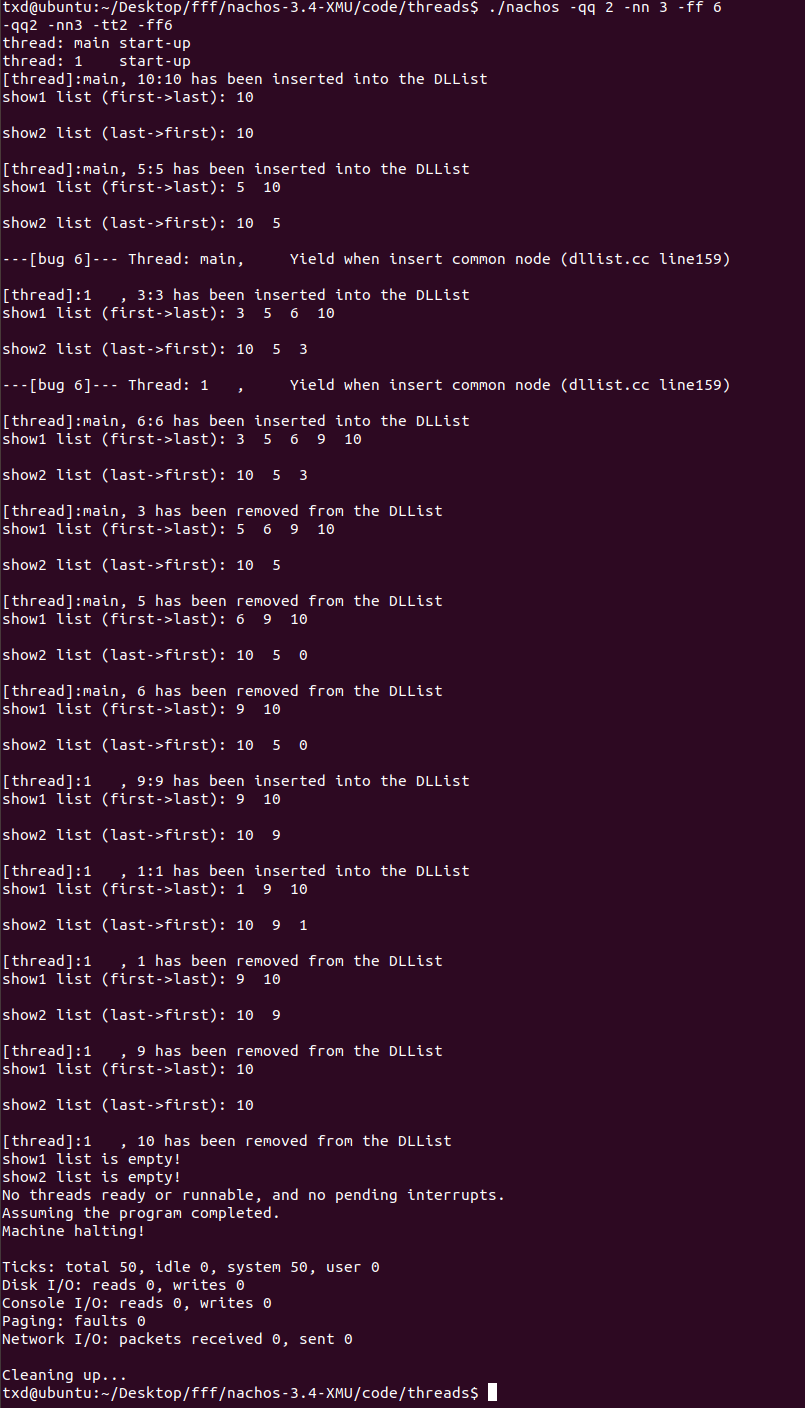
**BUG 6**

修改插入链表中的函数。导致链表“从first到last遍历”出来的，跟“从last到first遍历”出来的不一样

可使用序列：

A：10，5，6

B：3，9，1



图中的A.new是6，b.new是9

