实验总结

前言 :

关于实验总结和问题分析, 本人一直想写, 感谢老师给了我这个机会. 我单核和双核都花了很多时间去优化和修缮, 这方面还是有很多可以说的. 鉴于是双核版验收, 所以简化了单核的问题. 一下一共有

代码运行环境 :

DevC++编译器 , C语言 ( C99 ) , 没有用到任何C++语法

函数全部都是先声明后定义, 符合基本代码规范

程序设计思路 :

1. 文件处理

因为是处理dict文件, 所以有两种方法:

逐个**读取数据到本地**进行指令操作.

直接在文件中进行**文件指针修改**.

因为涉及到跳转指令, 所以在编写中, 我果断放弃了处理文件指针的方式 ( 因为跳转涉及的步数太多, 严重影响算法性能 ), 将文件数据存到本地之后再进行操作.

将文件数据存到本地, 又有两种方式:

开辟数组用**%s**读入

用字符型逐个**%c**读入.

因为scanf的效率较高, 又喜欢逐个控制, 所以选择了逐个读入字符型**%c**处理数据.

而这个也为我后面修改算法, 提高效率 “埋下了伏笔” .

2. 立即数计算

补码转换为源码, 是之前作业没有写过的问题.

不论先**取反**还是先**+1**, 都不碍事.

所以我是判断首位数字, 假如是负数,就记flag为0 ( 标志为负数 )

然后复制一下后面15个数据进行操作 ( 秦九韶算法 ) , 避免修改codeSegment.

但是之后在精简算法时, 我是在读入字符型就进行了十进制和补码转换处理, 使效率更高.

3. 跳转指令

单线程中最难的应该就是实现跳转指令, 我是一行一行处理文件的, 有二维数组便于查询和跳转, 只要在循环中控制行数, 在ip修改后修正行数 i 即可.

4, 高层算法设计

由于喜欢写子函数, 所以我都是把所有功能性的过程全部写成了子函数, 所以

**主函数** : 准备工作, 开线程, 结束线程, 结束工作

**线程函数** : 准备工作, 循环switch指令进行卖票, 结束线程

困难与解决

1. 代码风格

我写多核和单线程一样:

1. **完全避免全局变量的使用**

2. 并且尽量行数不超过**50**行

3. 列数不超过**80**字

4. 子函数分类明确

5. 所有需要子函数使用的参数全都是用**传参**解决

6. 配有**详细的注释**, 方便阅读和使用.

这个也为我后面将源码修改成项目也奠定了良好的基础.

2. 众多传参冗杂问题 :

不论是主函数传参到线程, 线程传参给子函数, 都有很多不同分类的数据, 且数据众多, 若逐个传参, 必定会造成函数冗杂和参数混乱难懂, 所以我在头文件的位置定义了三种结构体, 分别是 :

**1. struct Segment :**

**程序共用数据**, 相当于全局变量, 主函数创建, 主函数, 线程函数, 子函数共用. 封装互斥句柄, 同步锁线程信号, 文件1,2处理后数据空间指针, codeSegment, dataSegment.

**2. struct mtc :**

**线程单独寄存器**, 封装线程数据, 如 : id, ip, flag, ir, ax[8].

**3. struct number :**

**整合行指令处理数据**, 方便传参进入子函数. 有 : 1-8位转com, 9-12位转front, 13-16位转behind, 17-32位补码转im.

3. 线程互斥问题 :

**第一阶段 : 实现线程互斥**

如ppt上所给, 首先是创建线程, 并且两线程公用一个互斥句柄.

所以我先在主函数前面声明两个线程,

然后在主函数申请结构体空间 (包含互斥句柄), 和创建两个线程.

把**结构体指针**当作**无类型指针传入**线程使用.

只不过因为是**无类型指针**, 线程内要进行**强制转换**进行使用.

**第二阶段 : 出现 卖票id 与 票奇偶性 匹配出错问题**

因为我的程序比老师ppt上的程序复杂, 导致即使互斥也会因运行速度导致卖票顺序不互斥.

所以在单个互斥的基础上在锁线程指令前加了一道**同步锁**

这使卖票一定会按照**id奇偶奇偶**来卖, 而不会出现同一id卖票前后奇偶不同

**第三阶段 : 第一张票的随机性问题**

因为在线程中加了**同步锁**, 而一开始将同步锁就作用在了2线程上, 导致线程1永远先卖票.

这与初衷是相违背的.

所以我用**随机数**解决了这个问题 ( 时间种子计算 ), 在主函数随机锁住1或2, 随机先卖票.

**第四阶段 : 卖票顺序out出现问题**

虽然卖票的奇偶性已经没有问题, 但是经过分析dict得出, 卖票的输出是有以下过程 :

将寄存器1的数-1.

......

将寄存器1的数传入寄存器5所指的空间.

......

将寄存器1的数输出.

这就导致了**卖票和输出不是同一个时间段**. 可能先卖票而另一个线程块先输出, 而出现先打印卖98张后打印卖99张.

所以我投机取巧, 将线程的同步锁返还控制权写在了输出指令后, 所以**在卖票和输出的一整个过程中只有一个线程运行**. 虽然用时可能多了那么一两毫秒, 但是稳定性永远强于功能性.

4. 项目问题 :

一开始的时候我写的是源码, 写完之后想试试项目就写了.

我没有遇到问题, 因为我写的代码**没有全局变量**,

所以把**声明和结构体定义分到.h**, **其他分一些.c**就结束了.

5. 运行时间问题 :

与同学的运行结果进行比较来看, 我的**运行时间**略长.

初版程序运行时间在电脑插电情况下是约为**3秒**.

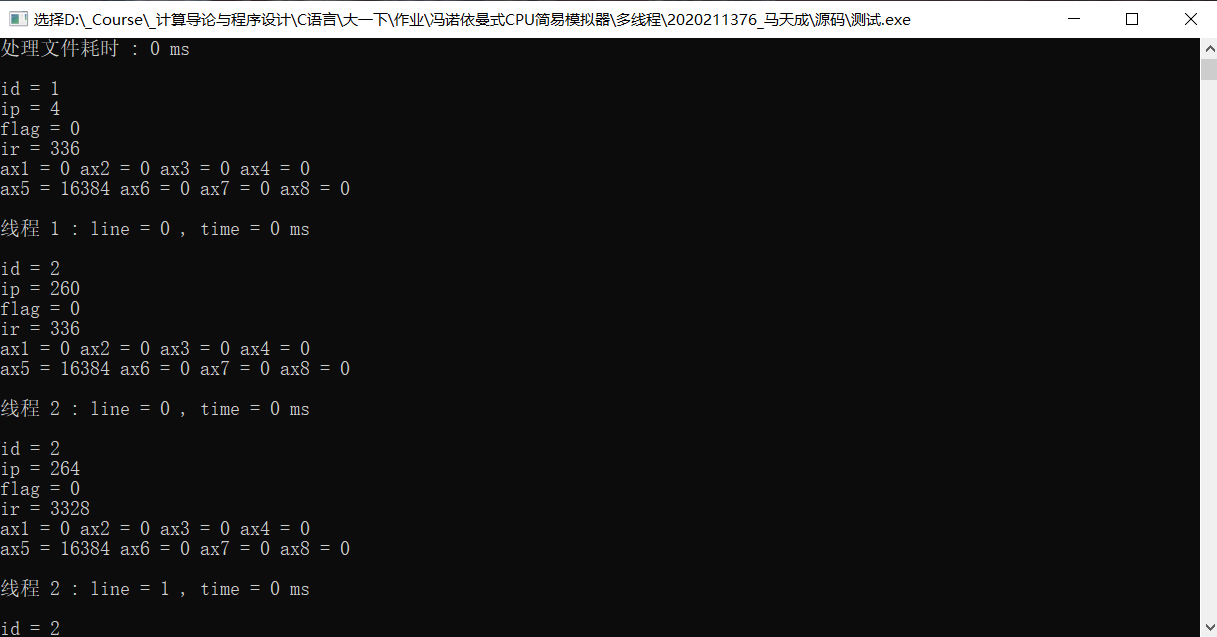
我一开始认为是我的算法问题, 如下 :

用**结构体**, 导致搜索数据的时候耗费了较多时间;

将文件读入本地二维数组再进行**二次处理**.

我经过修改, 将**结构体改成了全局变量**, 将**文件处理变成了单次处理**, 大大提高了效率.

但是, 时间并没有变短.

经过测试发现, 不论哪种写法, 都在**1毫秒**内完成了文件处理

再次分析, 原来是每次寄存器的**printf**函数花费了巨量时间

所以我将每次输出只用一个**printf**函数, 成功将时间减小到了1秒左右.

心得体会 :

1. 之后把printf的源码扒开来看了一遍 ( 确实难读 ) , 但很有收获.

**看源码**很重要, 特别是以后搞框架, 遇到bug那就根本不会了.

2. **高层算法设计**要精简, 避免多次修改浪费时间.

3. 结构体是个好东西, 修改代码, 增添参数时很好用.

4. 要养成**高效严谨的逻辑习惯**, 也要学会**测试数据**和逐块分析**debug**.

这是成为一个合格的程序员的关键 !

程序一定要bug极少, 因为你永远不会知道用户会怎样使用你的程序.

**2020211376马天成**