**被测软件执行结果分析报告**

## 功能分析

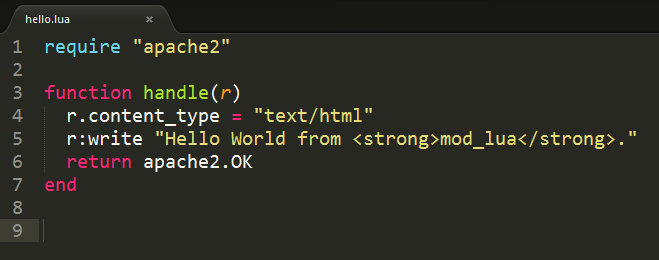
##### 搭建动态网页功能测试

在该测试实验中，我们主要是利用apache这个web服务器的mod\_lua模块，这个模块提供了对lua的支持。

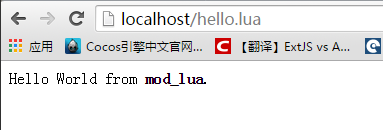
首先要下载apache2.4版本，并进行安装。因为以前的版本并不支持mod\_lua模块。

安装之后，打开apache的配置文件httpd.conf，然后打开mod\_lua这个模块。

然后在htdocs这个文件下编写一个hello.lua文件，代码如下：



实验结果：在浏览器中输入localhost/hello.lua，然后浏览器的显示结果如下：



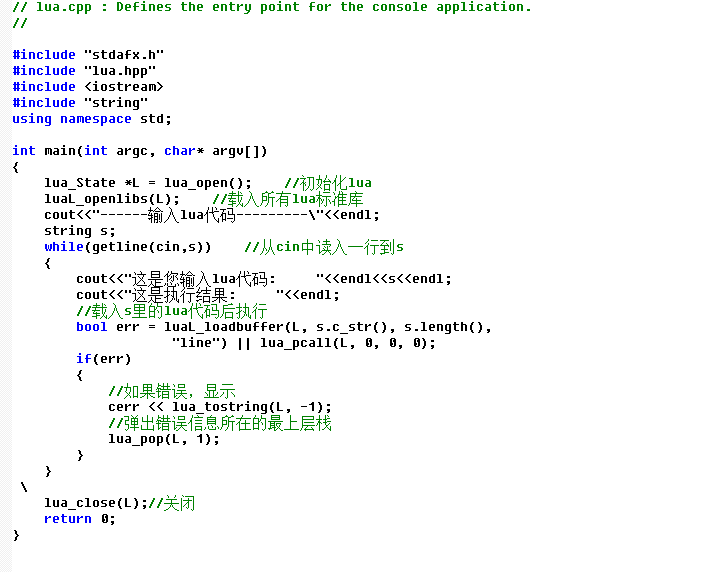
该测试实验通过。

###### C++调用lua功能测试

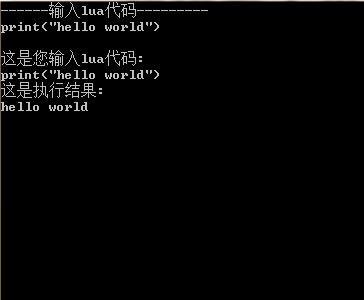
在该测试实验中，我们主要是在vc6.0中调用lua，通过lua的标准库。

首先要安装VC6.0,然后下载Lua,然后将lua的扩展库路径放入环境变量里。

在VC6.0中编写代码，代码如下:



然后对该代码进行编译执行，执行结果如下：

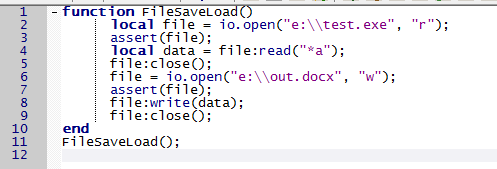


实验结果：该测试完美通过。

###### 文件复制功能测试

在该实验中，我们需要首先搭建lua代码的运行环境，安装Lua的windows版本安装包即可，该测试实验中，我们安装的是官方提供的最新的安装包。

Lua拷贝文件的代码并不复杂，代码贴图如下：



在该实验中，我们测试了从1K到100M的纯文本文件复制（包括扩展名为lua和txt的文件），发现Lua都能将文本内容准确复制到新文件中去。

但是在测试一些视频文件，可执行文件，word文档等其他类型的文件时，我们发现Lua的文件复制程序并不能正常复制，而是只复制1K左右的内容过去。

于是我们希望得出无法拷贝的原因，我们将上述不能复制的文件扩展名做了修改，改为文本文件的扩展名，发现依然不能正常拷贝；而我们将能复制的纯文本文件的扩展名更改为不能复制的文件的扩展名时，发现该文件依然能正常拷贝，因此我们得出结论，文件拷贝不正常的原因不与扩展名有关。

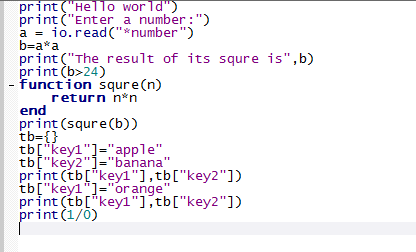
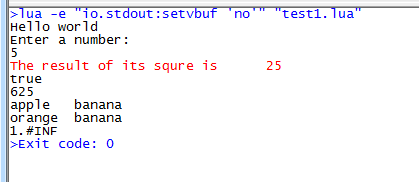
比较可能的原因是纯文本文件和非纯文本文件里的编码规则不一样，所以Lua在读取到内存的过程中出了问题，所以我们查询了Lua内存编码使用规则。在官方的说明书中，我们查到，Lua的string标准库只能处理单字节，但是多字节的汉字读取到内存里还是原来的单字节排序，这应该是没有问题的。所以编码应该不是这里的主要问题。其实如果编码是一个问题的话，那么我们在测试文本文件过程中，有一些汉字是多字节表示的，也可能会遇到问题。

查询file:read函数的使用说明中，我们可以看到该函数是读取到文件结尾才截止的。但是Lua的string库中说明了，string是带有结束符的，结束符是0，而二进制文件中，是有很多内容用0填充的，而一些像图片和视频的字符编码中，前面的一些字节是说明文件的一些基本内容的，可能会产生0字符，而且后面的数据中还有RGB信息，更可能产生0字符。

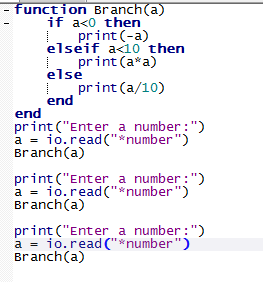
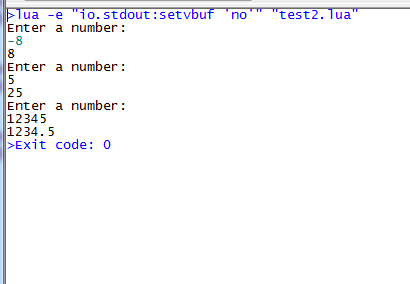
这里我们给出Lua文件拷贝的一个小建议，就是尽量只拷贝文本文件。实际上二进制文件和一些其他非文件文件往往都有自己的格式，在知道这些格式后我们在拷贝时可以用非字符串的方法来处理，这样一样能实现文件拷贝，当然可能不仅仅是拷贝这么简单的事了，还可以利用读取的数据做一些基本操作等。

###### 只运行一次的小程序功能测试

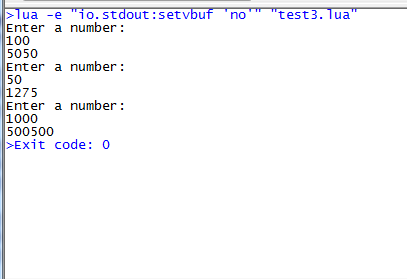
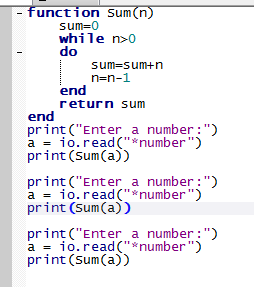
我们准备编写4类小程序运行，一种是顺序结构的小程序，包含基本的语句：输入，输出，赋值，多种类型的数据，其中Lua的主要数据包括数值，字符串，布尔值，函数值，表，一个特殊值nil。第二种是分支执行语句，程序做的内容为读入一个数值，根据值的不同范围做不同操作。第三种是循环语句，用来就1到n的和。第四种则是递归语句，求n的阶乘。下面我们给出测试结果。



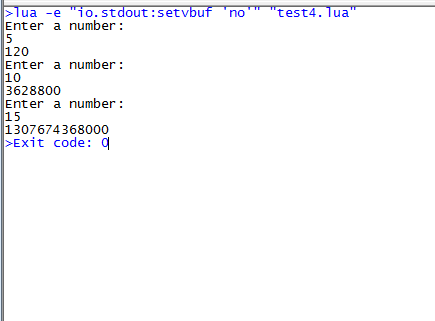
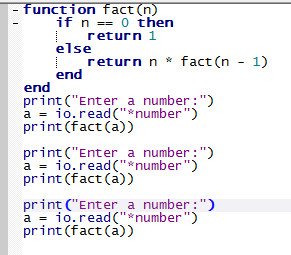
顺序执行程序和输出结果



分支程序及输出结果



循环程序及输出结果



递归程序及输出结果

上述四类基本程序中，尽管内容简单，但是我们各个方面都有测试到了，我们认为Lua确实提供了基本的功能支持，实际上在网上我们可以找到Lua脚本运行的bug，但这种往往是在非常庞大的脚本或者非常复杂的脚本中，我们很难理解这些脚本做了些什么，也无法具体的做一个测试，所以我们决定放弃寻找在大脚本中的bug。

### 性能测试报告

###### Table性能的测试：

当需要创建非常多的小size表时，应该预先填充好表的大小。

问题产生：当我们把一个新键值赋给表时，若数组和哈希表已经满了，则会触发一个再哈希(rehash)。再哈希的代价是高昂的。首先会在内存中分配一个新的长度的数组，然后将所有记录再全部哈希一遍，将原来的记录转移到新数组中。新哈希表的长度是最接近于所有元素数目的2的乘方。

问题解决：在非常多的小标创建的时候，预先填充以避免rehash。

实验效果输出：不填充表格时间为1.157567s 填充表格0.646453s。

测试代码：

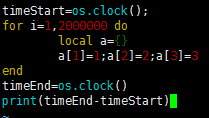
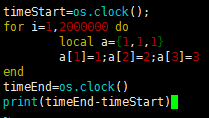
 

图1：不填充表格代码 图2：填充表格代码

###### 字符串性能测试：

大字符串连接中，应该避免..。应用table来模拟buffer，然后concat得到最终字符串。

问题产生：所有的字符串变量，只保存字符串引用，而不保存它的buffer。

问题解决：应用table来模拟buffer，然后concat得到最终字符串。

测试结果：通过..进行连接时间消耗:6.649481s 通过table缓存时间消耗0.07178。

测试代码：

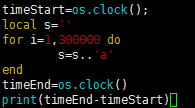
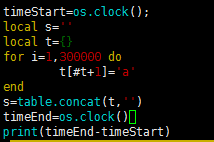
 

图3：使用字符串连接符号 图4：通过table缓存进行连接

###### WEB应用的性能测试：

ngx\_lua是基于nginx的扩展 nginx已经有无数家验证过，是个可靠的WEB应用。而且ngx\_lua 已经有很多广泛的应用。那么与其他脚本语言php, lua, python 和 node.js在性能上有何优势。ngx\_lua模型也是单线程的异步的事件驱动的，工作原理和nodejs相同，代码甚至比nodejs的异步回调更好写一些。通过并发我们对三个脚本语言进行了测试。

性能测试，100并发

php:17400

nodejs:31197

ngx\_lua:32628

在CPU，内存和Net的对比结果：



存在的缺点：lua确实是很快，不过也有缺点就是lua的文件处理很弱, 各种库非常的少。

测试结果：nodejs占用CPU比lua高一些，nodejs与ngx\_lua二者占用内存不变(在处理同一业务逻辑时)，php占用内存不稳定（可能跟多线程有关，连接数越多，内存占用越多）并发请求数lua/nodejs相当，均比php快一倍。资源占用上lua最少，nodejs次之，二者对内存并不贪婪，php占用最多且不稳定。