操作系统 课后编程作业 2

171860607
白晋斌
邮箱 810594956@qq.com

目录

一、实验目的

基于 pthread 线程库实现一个多线程程序，用于统计一个英文文本文件（例如英文原版莎士比亚全集）中各英文单词出现的次数，最终输出出现次数最多的 10 个单词及其出现次数。实现中需要有多个读文件和分词线程，至少一个统计线程，并正确处理多线程间同步（信号量或管程）。

二、实验原理

1. 计算原理

首先通过参数获取用户想要的线程数，之后分 n 个线程打开文本读取文本并分词。这里的分工方式是主线程先试探出被读文本的长度 L，之后得到每个线程要读的长度 L/n，这样每个线程要读的部分就是 num*L/n ~ [num+1]*L/n，通过用 istream::seekg(int) 来定位文件指针。关于边界的情况，我们只需要“留右不留左”，即从当前开头的第一个空格开始读取，读到超出当前边界的第一个空格。

之后是分词，简单说直接空格分词，这里仅仅对词汇作简单处理（因为停用词库种类繁多，不够统一，而如果不去除无意义词的话其实排序结果意义不大，所以从简处理，重点放在多线程），如，",.;:'?!><+=|*&^%$#@\""词汇中出现这样的特殊字符我们将直接删除它。之后通过 map 的数据结构（这里最初尝试用自行实现的二叉树，后发现性能不如 stl 中的 map）将分词结果存于对应的线程 map 中。

我们通过 PV 操作来实现线程间的通信。每个线程读取分词完毕自己的部分后，便执行 V 操作 sem_post(&writee); 而最终的综合统计线程开始则是 P 操作 sem_wait(&writee); 此处 writee 初始值为线程数的相反数。这样当所有线程都执行完毕后，综合统计线程便可以将统计结果（n 个 map）综合成一个 vector 并排序，输出前十位即可。

2. 并行思想

在上述计算过程中可以并行化的一个地方便是文本 IO 与分词了，假设要读取的文本总字节为 K，开启 num_threads 个线程，这样每个线程平均的去读取 K/num_threads 个字节，最后将所有线程的分词结果求和并作统计，这样比串行的效率高很多 。

三、实验代码

见附件。

四、统计结果

我们以徐老师所给的莎士比亚全文作为数据来源。

分别分析线程数从 1 到 10 的程序运行时间与统计结论。

```
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 1
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:1  time:1.329778 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 2
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:2  time:0.753167 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 3
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:3  time:0.555544 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 4
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:4  time:0.476294 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 5
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:5  time:0.447599 s
```

```
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 6
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:6  time:0.429784 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 7
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:7  time:0.441404 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 8
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:8  time:0.420721 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 9
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:9  time:0.543412 s
parallels@parallels-Parallels-Virtual-Platform:~/Desktop$ ./code 10
the: 25245
I: 20640
and: 19733
to: 16893
of: 16627
a: 13614
my: 11415
in: 10567
you: 9540
is: 8287
pthreadnum:10  time:0.600301 s
```

可以得出，词频与线程数没有关系。

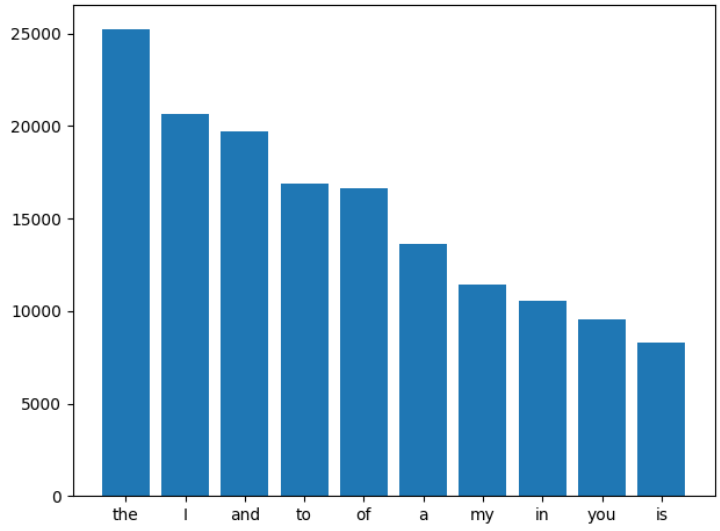
接着我们在 pycharm 中运行如下代码:

```
import matplotlib.pyplot as plt

plt.bar(range(len(num_list)), num_list, tick_label=name_list)

plt.show()
```

得到如下图片:



可以看出排名前十的单词词频分布情况。同时也发现，如果不去掉某些无意义词的话，单纯统计词频可能无法从文本中获取有效信息。

五、实验结果分析

1.实验结果

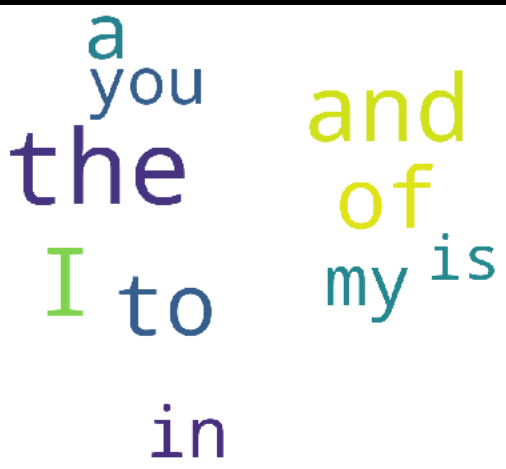
首先我们分析了一下程序运行时间，发现如果文本量太小的话，多线程反而没有优势，但是当文本量很多（如莎士比亚全文约有 96 万词），便基本符合线程数为硬件线程数时刚好程序运行时间最短。

因为停用词库种类繁多，不够统一，因此我们并没有去除一些无意义代词连词，所以发现最后排名前十的词如下表所示。

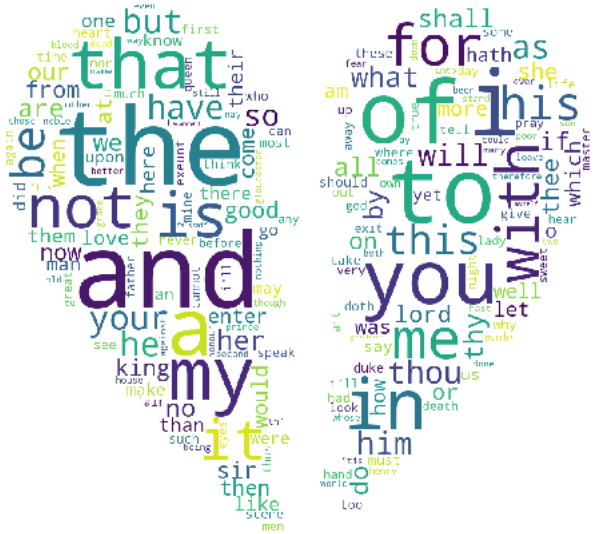
```
1 the 25245
2 I 20640
3 and 19733
4 to 16893
5 of 16627
6 a 13614
7 my 11415
8 in 10567
9 you 9540
10 is 8287
```

2.结果分析

虽然词汇没有意义，但我们仍然可以顺手做一个分词。



发现只取十个词不好看，我们多选一点词来，以心碎图为背景（符合莎士比亚悲剧色彩）。



3.总结概括

总结概括有以下三点：

1. 对比分析乔布斯演讲稿和莎士比亚全文，我们发现文本越长，创建线程所需的时间对总时间的影响越小，从而多线程的优势越发明显。并且如果需要进行大量相同类型的计算（如统计分析），则只需按核的数量创建线程就能充分利用系统资源了。

2. 如果不去掉某些无意义词的话，单纯统计词频可能无法从文本中获取有效信息。

3. PV 操作在 c 语言中的代码形式为 sem_wait(&writee);和 sem_post(&writee) ; 此外，关于 pthread 库的相关操作，其实 c++中还有一个 thread 库封装更为高级、便捷。但本次实验学习考虑，依旧使用 pthread 库。

六、代码优化

1.数据结构

早期自行实现二叉树对数据进行统计分类，后发现在 windows 平台下效率极低（统计莎士比亚约 1 分钟），后转向 map，成功将统计时间压缩到 2 秒内。

```
//早期代码：进行中序遍历查找并写入文件
void InOrderAndPrint(BiTree T, FILE* pf)
{
  if (T != NULL)
  {
    InOrderAndPrint(T->lChild, pf);
    fprintf(pf, "出现的词汇：%-30s 频率:%-9d\t\n", T->data, T->count);
    printf("出现的词汇：%-30s 频率:%-9d\t\n", T->data, T->count);
    InOrderAndPrint(T->rChild, pf);
  }
}
```

2.累加计数

```
pthread_mutex_lock(&mutex);
//map operation
pthread_mutex_unlock(&mutex);
```

如上图所示，如果仅仅维护同一张 map 数据结构时，要不断的 lock 和 unlock，大大拖慢了并行速度，在这里我们选择每个线程内部维护自己的 map，最终线程结束后，由一个统计线程综合前面所有的 map，这样可以大大提高效率。

3.clock

Linux 环境下似乎 clock 函数调用有问题，在这里我们改用 gettimeofday()函 数。效果似乎还可以。

六、心得体会

1. undefined reference to 'pthread_create'

Linux 下 undefined reference to ‘pthread_create’问题解决。

在使用 Linux 线程模块时，使用 pthread_create 函数。编译命令为 gcc main.c -o test 时，会出现如下错误。

```
/tmp/ccIvH3bU.o: In function `main':
main.c:(.text+0x81): undefined reference to `pthread_create'
collect2: error: ld returned 1 exit status
```

问题的原因：pthread 不是 linux 下的默认的库，也就是在链接的时候，无法找到 p hread 库中该函数的入口地址，于是链接会失败。

解决：在 gcc 编译的时候，附加要加 -lpthread 参数即可解决。

试用如下命令即可编译通过

```
gcc main.c -o test -lpthread
```

2.感悟总结

通过这次实验我学会了很多内容，以前虽然学习过关于信号量、管程的概念，但是并没有真正的使用信号量编写实际代码，统计词频的思想很简单，以前在自己选修的人工智能程序设计课上也通过 python 实践过，但是通过 C 语言实现统计词频的程序（而且还是多线程），当运行产生正确的输出结果时，心里十分高兴。随后让我绝望的是，由于没有添加停用词，导致词频最高的十个词都是些没有意义的词，这与统计词频的目的矛盾，此外，由于最开始使用二叉树实现统计，效率不高，通过修改代码，改成 map 数据结构，并行程序快的多了。通过编写这个小的多线程程序，对计算机的信号量（PV 操作）、管程有了更加 深刻的认识。通过对实验分词结果以及运行时间的分析，更加清晰的认识到了信号量与管程的重要性。