## 多线程与单线程比较

编写一个支持泛型的线程安全FIFO阻塞队列，采用生产和消费者模型，一个线程随机生成一个长度为12的包含字母和数字的字符串放入队列，另一个线程从队列读取字符并按字符顺序升序排序。

在同一个线程内随机生成一个长度为12的包含字母和数字的字符，对这个字符串进行升序排序。

## 程序代码说明：

1. 有些类存在严重安全问题以及使用不得当引发的功能问题，为了体现编码意识只是简单的空实现拷贝构造函数并设置成私有函数来规避，其他的一些默认生成函数没有重写，仅仅代表这个编码意识而已
2. 功能函数没有异常处理，仅仅考虑正常情况，
3. 单件实现没有加锁保证，全部依赖在主线程初始化

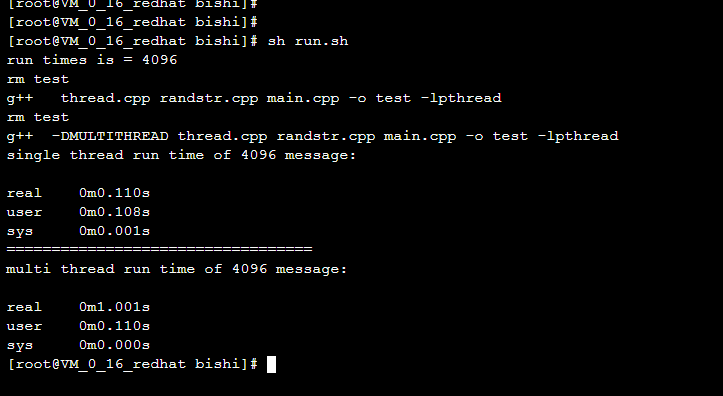
## 代码编译运行说明：

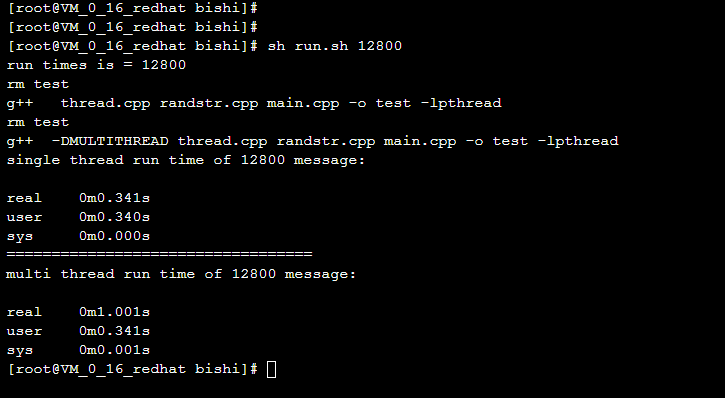
1. 文件默认是dos格式，将整个目录上传到linux服务器后，需要手动更改下文件格式，只要在此目录下执行以下命令：

dos2unix \*; chmod +x run.sh

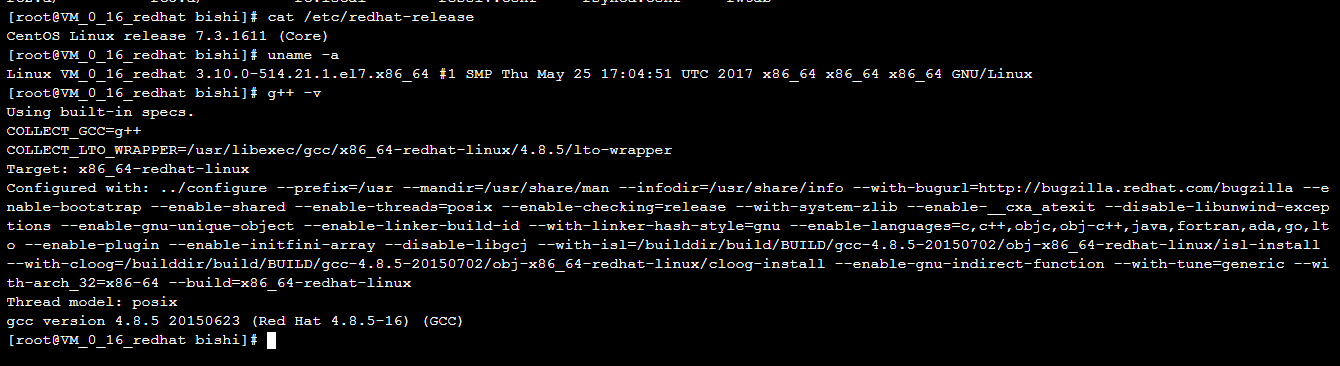
1. 运行run.sh进行编译运行验证；run.sh允许传入一个参数，指定产生的消息条数，不指定默认是4096，最终看单线程和多线程程序生产消费同样数目的消息所耗时间长短：

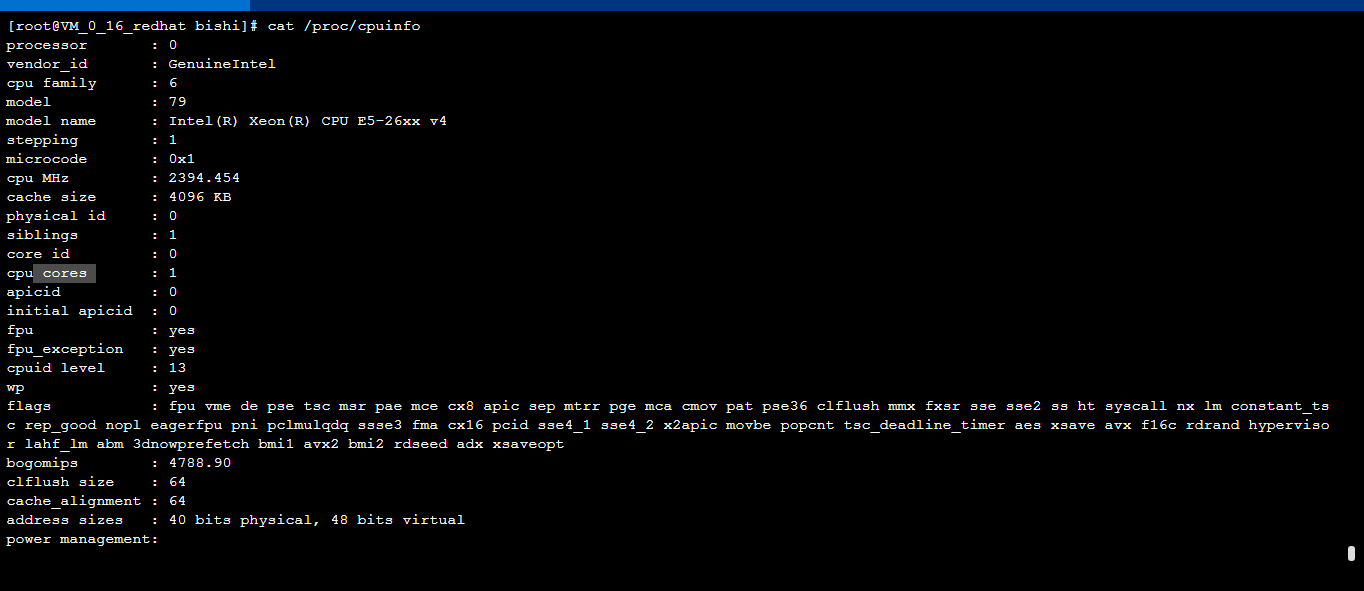
（因条件所限，编译运行都在单cpu单核上进行）

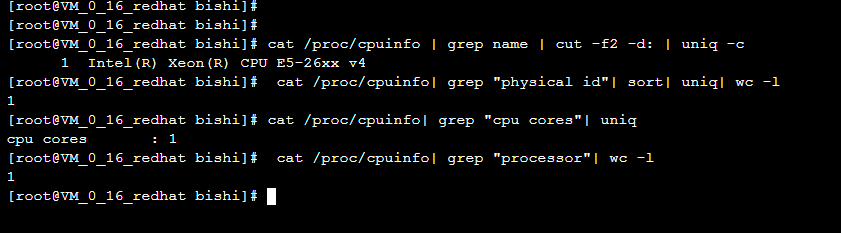




自编译运行环境单cpu：







## 测试说明：

**结果（根据经验及理论知识+单cpu多线程的验证）**：

从单cpu测试情况来看多线比单线程所耗时间要长，结合理论知识可以得出：

在生产消费相同数目的消息，单线程执行时间更短

**分析**：

1. 从代码层面，多线程多了锁机制其他跟单线程代码一致排除掉因代码原因导致的性能差异
2. 单线程的程序运行在用户态上，没有使用锁机制，没有在用户态和内核态切换导致性能；而多线程因为使用了锁机制（互斥量，条件变量）一个导致在用户态和内核态频繁切换导致系统开销增大，另一个生产消费者因为竞争锁而导致没有充分利用多核，即同一时刻只有生产者或者消费者线程在运行，并没有充分的利用多核资源，当然这里的锁机制有提升空间，可以使用volatile来模拟锁机制
3. 为什么多线程效率并没有提升，为啥还有很多场景下在使用，使用生产者消费者模型主要是为了架构解耦，为了使服务更聚焦，功能更易拆分。
4. 多线程使用的场景：个人认为不外乎两个：一个是一个job可以真正拆分程几个独立的子任务不相互依赖能并行执行，增大吞吐量提升效率；另一个就是增加用户体验，某个job很耗时开启线程单独执行。