题目1.

• 使用**http_load**命令来测试这两个模型下**Web**服务器的性能指标。并根据这些测试指标对比，分析 为什么这两种模型会产生不同的性能结果？

多进程

```
→ http_load-12mar2006 git:(main) ✗ ./http_load -parallel 10 -fe
tches 1000 urls.txt
1000 fetches, 10 max parallel, 260000 bytes, in 0.322542 seconds
260 mean bytes/connection
3100.37 fetches/sec, 806097 bytes/sec
msecs/connect: 0.056592 mean, 1.123 max, 0.017 min
msecs/first-response: 3.13809 mean, 13.886 max, 0.196 min
HTTP response codes:
  code 200 -- 1000
→ http_load-12mar2006 git:(main) ✗
```

多线程

```
oject/http_load/http_load-12mar2006
hatsunemiku@ubuntu:~/OS-project/http_load/http_load-12mar2006 64x24
→ ~ j 2006
/home/hatsunemiku/OS-project/http_load/http_load-12mar2006
→ http_load-12mar2006 git:(main) ✗ ./http_load -parallel 10 -fe
tches 1000 urls.txt
1000 fetches, 10 max parallel, 260000 bytes, in 0.252117 seconds
260 mean bytes/connection
3966.41 fetches/sec, 1.03127e+06 bytes/sec
msecs/connect: 0.161524 mean, 2.011 max, 0.017 min
msecs/first-response: 2.23772 mean, 32.27 max, 0.075 min
HTTP response codes:
  code 200 -- 1000
→ http_load-12mar2006 git:(main) ✗
```

在多线程模型和多进程模型下进行测试展现出不同的性能结果可能是因为：

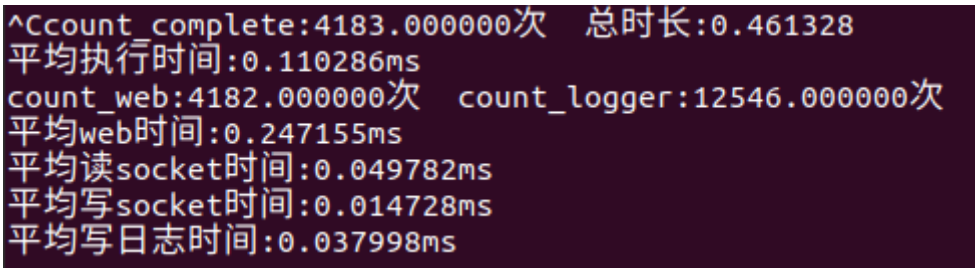
1. 并发处理能力：
    - 多线程模型： 共享同一进程的多个线程可以共享内存，因此它们之间的通信相对较为简便。然而，这也可能导致竞争条件（Race Conditions）等问题。
    - 多进程模型： 不同进程之间通常有独立的内存空间，需要通过进程间通信（IPC）来传递数据。这可能导致一些额外的开销。但是，由于每个进

程都是独立的，它们之间不存在共享内存的竞争条件，这可以降低一些同步问题。

2. 资源消耗：
    - 多线程模型： 线程相对较轻量，切换成本较低，但它们共享相同的地址空间和资源。这可能导致一些资源争用的问题，特别是在高并发情况下。
    - 多进程模型： 每个进程都有独立的地址空间，但进程切换成本通常较高。此外，每个进程都有自己的资源副本，这可能导致一些额外的内存消耗。
3. 系统调度：
    - 多线程模型： 线程由操作系统的线程调度器进行管理，切换通常较为迅速。
    - 多进程模型： 进程的切换可能涉及更多的上下文切换，这可能导致一些性能开销。

• 对模型中的**socket**数据读取、发送、网页文件读取和日志文件写入四个**I/O**操作分别计时， 并打印出每个进程或线程处理各项**I/O**计时的平均时间。

多线程



• 根据上面的计时数据结果，分析并说明模型中哪些**I/O**操作是最消耗时间的？

1. 写入日志文件的操作是最消耗时间的，其平均执行时间为0.37998ms。写入日志文件通常涉及磁盘I/O，而磁盘I/O相对于内存或网络I/O来说通常是较为耗时的操作。
2. 读取web文件的操作是次消耗时间的，其平均执行时间为0.247155ms。这可能是因为从网络读取文件会涉及到网络延迟和带宽限制，因此相对而言，这个操作可能需要更多的时间。

• 思考一下，怎么修改线程模型，才能提高线程的并发性能？

为了提高多线程Web服务器的并发性能，可以考虑以下修改：

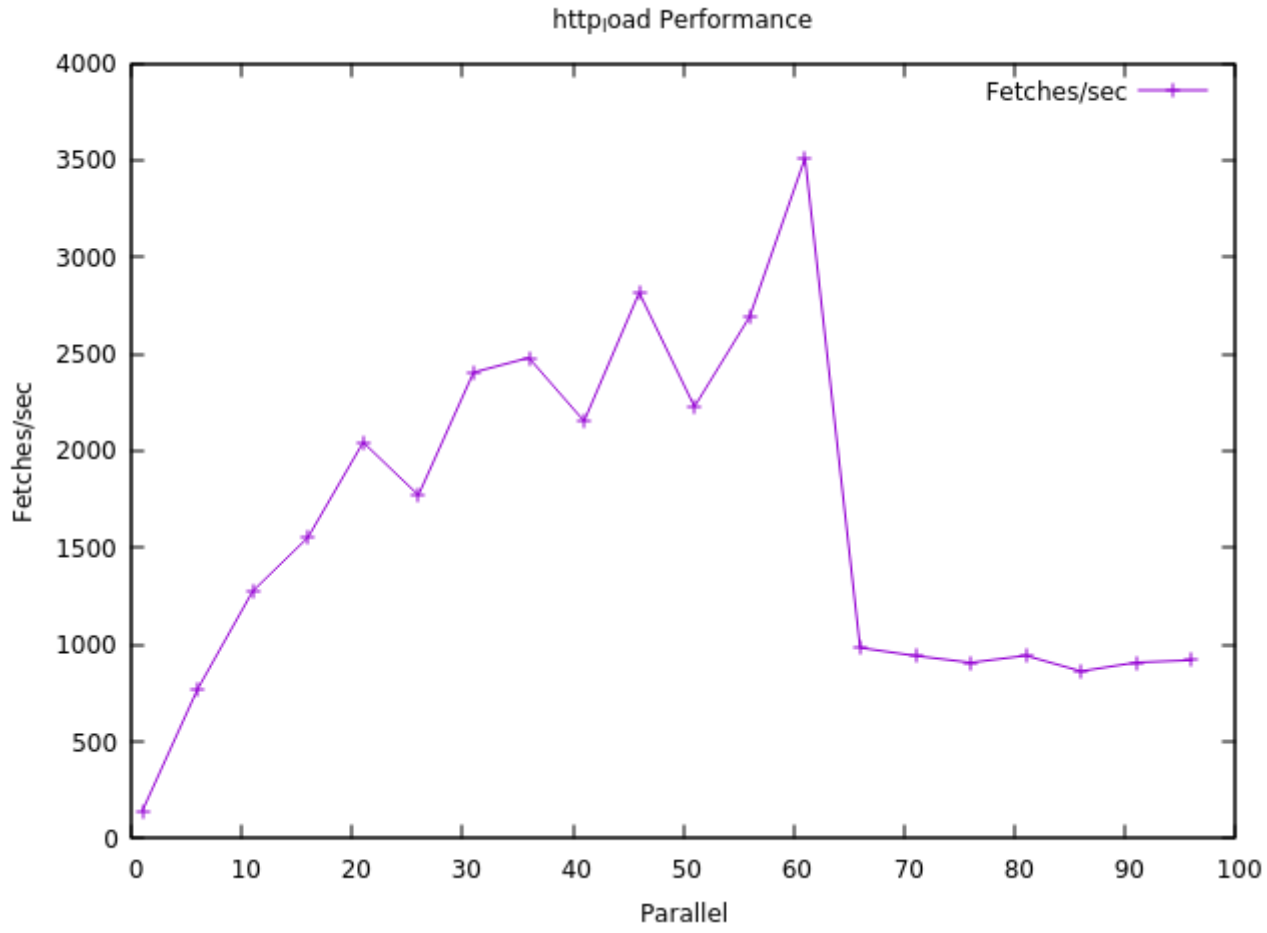
1. 线程池：
   不要为每个传入请求创建一个新线程，考虑使用线程池。线程的创建和销毁可能很昂贵，维护一个线程池可以减少这种开销。使用队列来管理传入请求，并将它们分配给池中可用的线程。

2. 优化文件**I/O**：

   `web`函数中的文件I/O操作可能成为瓶颈。考虑使用异步I/O或非阻塞I/O来处理多个文件读取请求。实现缓存机制，将经常请求的文件存储在内存中。这可以显著减少对磁盘I/O的需求，提高对常用资源的响应时间。

题目**2.** 调整**http_load**命令参数，增加其并发访问线程数量，会发现随着并发访问数量在达到一定 数量后，再增多会导致多线程**Web**服务进程的性能出现下降的现象。试分析产生上述现象的原因是 什么？

```
#!/bin/bash

# 设置 http_load 命令路径
HTTP_LOAD_CMD="./http_load"

# 设置输出文件名
OUTPUT_FILE="results.csv"

# 遍历并发数量范围，步长为5
for parallel in {1..100..5}
do
    # 运行 http_load 命令，获取结果
    output=$($HTTP_LOAD_CMD -parallel $parallel -fetches 1000 urls.txt)

    # 使用正则表达式提取 fetches/sec 的整数部分
    fetches_sec=$(echo "$output" | grep -oE "([0-9]+[.])?[0-9]+ fetches/sec" | cut -d ' ' -f 1)

    # 输出结果到 CSV 文件
    echo "$parallel,$fetches_sec" >> $OUTPUT_FILE
done

# 绘制折线图表
gnuplot -persist <<-EOF
    set title "http_load Performance"
    set xlabel "Parallel"
    set ylabel "Fetches/sec"
    set datafile separator ","
    plot "$OUTPUT_FILE" using 1:2 with linespoints title "Fetches/sec"
EOF
```



产生上述现象的原因是可能是：

1. 上下文切换开销：
   随着线程数量的增加，操作系统需要更频繁地进行上下文切换，将CPU从一个线程切换到另一个线程。过多的上下文切换会导致性能下降，因为切换本身也需要时间。
2. 资源竞争：
   系统中其他资源的竞争，例如网络资源、文件描述符等，也可能成为性能瓶颈。如果系统中存在瓶颈资源，增加并发线程数量可能会加剧竞争，导致性能下降。
3. 线程管理开销：
   线程的创建、销毁和管理也会带来一定的开销。如果频繁地创建和销毁线程，或者线程管理机制复杂，可能会降低性能。