Linux 上机作业“脚本程序设计”实验报告

实验一 生成 TCP 活动状况报告

一、 实验题目

编写 shell 脚本程序，每隔 1 分钟生成 1 行信息：

1. 当前时间；
2. 这一分钟内 TCP 发送了多少报文；
3. 接收了多少报文；
4. 收发报文总数；
5. 行尾给出符号+或-或空格（+表示这分钟收发报文数比上分钟多 10 包以上，差别在 10 包或以内用空格，否则用符号-）。

运行示例如下：

```
2020-04-17 00:02   345   314   659
2020-04-17 00:03  1252  1100  2352    +
2020-04-17 00:04   714   570  1284    -
2020-04-17 00:05   151   139   290    -
2020-04-17 00:06  1550  1097  2647    +
2020-04-17 00:07  1385   959  2344    -
2020-04-17 00:08     5     1     6    -
2020-04-17 00:09     5     1     6
2020-04-17 00:10   837   723  1560    +
2020-04-17 00:11    22    22    44    -
```

二、 实验分析

1. 观察实验给出的输出结果，第一行输出最后不需要添加符号，或添加空格。
2. 题目要求每分钟记录一次，并且行尾需要给出+或-或空格，因此程序主体应当为循环和分支结构。

三、 实验步骤

1. 使用 netstat --statistics | cat -n 命令查看需要的 TCP 活动状况，以及我们需要的收发情况，分别在第 49 行和第 50 行：

```
43  Tcp:
44      4556033 active connection openings
45      280844 passive connection openings
46      115946 failed connection attempts
47      79138 connection resets received
48      105 connections established
49      106492455 segments received
50      124894818 segments sent out
51      393829 segments retransmitted
52      945 bad segments received
53      204081 resets sent
54      InCsumErrors: 17
```

2. 将上述两行可以通过 awk 指令取出第一列的两个数字，作为变量保存（需要使用 NR 代表文本行号）；等待 60s 可以使用 sleep 命令；而时间信息可以通过 date 指令获得，date 指令后面添加以“+”为首的字符串可以控制输出时间的格式与题目要求一致：

b285@Ubuntu-bupt: ~/homework3

```
last_receive=$(netstat --statistics | awk 'NR==49 {print $1}')
last_send=$(netstat --statistics | awk 'NR==50 {print $1}')

sleep 60

time=$(date "+%F %H:%M")
now_receive=$(netstat --statistics | awk 'NR==49 {print $1}')
now_send=$(netstat --statistics | awk 'NR==50 {print $1}')
```

3. 获得了间隔一分钟的收发报文数量，对应的值相减可以得到这一分钟内的收发报文情况，而得到的两个结果相加可以得到这一分钟总的收发情况：

```
r=$(expr $now_receive - $last_receive)
s=$(expr $now_send - $last_send)
all=$(expr $r + $s)
echo $time $s $r $all
```

4. 这样就得到了这一分钟内的结果，使用 echo 将结果依次打印出来。由于

这是第一分钟内的情况，它最后没有“+”“-”或空格，所以单独分开来，接下来需要对每一分钟进行判断和记录，需要使用循环结构，循环内部的整体思路和第一分钟的思路大致相同，首先等待 60s 之前应该记录上一分钟的收发情况：

```
while true; do
    last_receive=$now_receive
    last_send=$now_send
    last_all=$all

    sleep 60
```

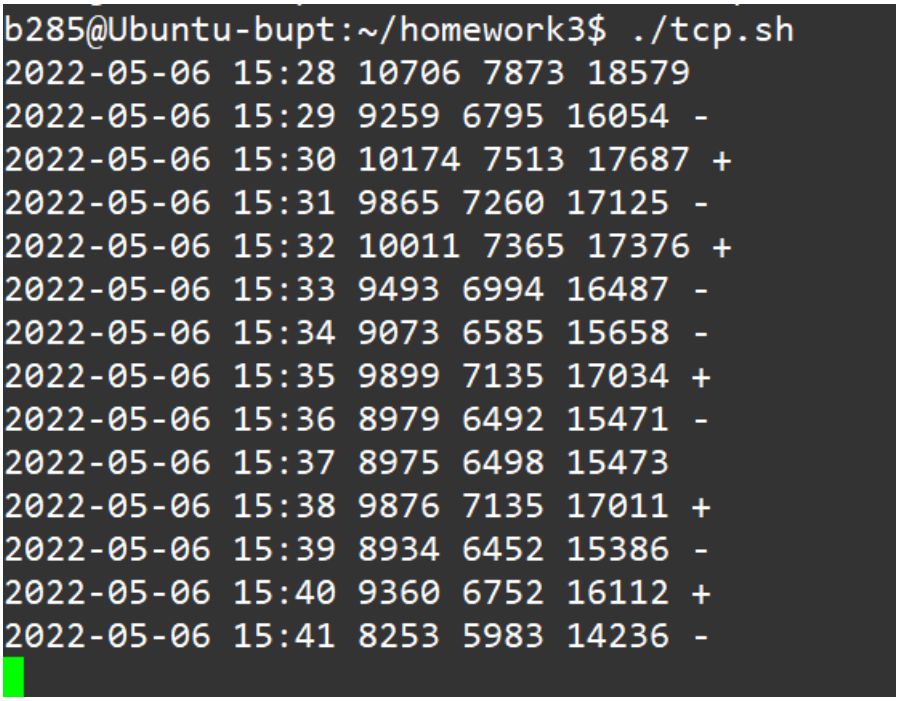
5. 等待过后需要对目前的收发报文情况进行记录，并与一分钟前的记录进行相减：

```
    time=$(date "+%F %H:%M")
    now_receive=$(netstat --statistics | awk 'NR==49 {print $1}')
    now_send=$(netstat --statistics | awk 'NR==50 {print $1}')
    r=$(expr $now_receive - $last_receive)
    s=$(expr $now_send - $last_send)
    all=$(expr $r + $s)
```

6. 最后的符号判断需要使用分支结构，只需要比较这一分钟的收发报文总数和上一分钟的即可，-gt 表示大于等于，-le 表示小于，最后使用 echo 输出结果：

```
    differ=$(expr $all - $last_all)
    if [ $differ -gt 10 ]; then
        flag='+'
    elif [ $differ -le 10 -a $differ -gt 0 ]; then
        flag=' '
    else
        flag='-'
    fi
    echo $time $s $r $all $flag
done
```

7. tcp.sh 文件的部分执行结果为：



四、 实验总结

本次实验主要让我实际运用了循环与分支结构，复习了 awk 指令的使用，以及脚本程序编写的一些规范，例如使用变量的时候需要在前面加上$，很多时候没有特别注意这么做而使程序崩溃，还有很多指令的规范书写，if 条件的书写需要在中括号内输入空格，这些细小的点如果一开始不注意后面很难发现错误，所以该加的符号一定要加上，在此基础上才能进而保证程序的逻辑正确。

五、 源代码

```
last_receive=$(netstat --statistics | awk 'NR==49 {print $1}')
last_send=$(netstat --statistics | awk 'NR==50 {print $1}')

sleep 60
# 第一分钟的输出
time=$(date "+%F %H:%M")
now_receive=$(netstat --statistics | awk 'NR==49 {print $1}')
now_send=$(netstat --statistics | awk 'NR==50 {print $1}')
r=$(expr $now_receive - $last_receive)
```

```
s=$(expr $now_send - $last_send)
all=$(expr $r + $s)
echo $time $s $r $all
# 之后每一分钟的输出
while true; do
    last_receive=$now_receive
    last_send=$now_send
    last_all=$all

    sleep 60

    time=$(date "+%F %H:%M")
    now_receive=$(netstat --statistics | awk 'NR==49 {print $1}')
    now_send=$(netstat --statistics | awk 'NR==50 {print $1}')
    r=$(expr $now_receive - $last_receive)
    s=$(expr $now_send - $last_send)
    all=$(expr $r + $s)
    differ=$(expr $all - $last_all)
    if [ $differ -gt 10 ]; then
        flag='+'
    elif [ $differ -le 10 -a $differ -gt 0 ]; then
        flag=' '
    else
        flag='-'
    fi
    echo $time $s $r $all $flag
done
```

实验二 下载 bing 图库中图片

一、 实验题目

编写脚本程序 bing.sh，将 https://bing.ioliu.cn/?p=36 图库中照片全部下载下来存放到本地 bing 目录，上面 URL 中 p=36 可以换成 p=126 可访问 126 号页面，每页有 12 个图，每个图的日期，中文说明信息和下载地址及文

件名 html 文件中可提取。要求下载后的文件命名为“日期 说明.jpg”例如：2019-08-03 野花草甸上的一只欧亚雕鸮，德国莱茵兰-普法尔茨.jpg

二、 实验要求

1. 不重复下载已下载的图片
2. 考虑并发
3. 考虑下载文件出现故障的情况

   如果一个图片有 5MB，接收 1.5MB 后网络断开，则残存一个不完整的图片文件。避免这种现象发生的一种方法是 wget 下载时使用一个临时文件名。判断 wget 是否成功，若成功则将文件改名为正式名称；若失败，删除临时文件。临时文件名的选取要考虑的并发问题，至少不可以程序中写死一个文件名导致两进程因使用相同的临时文件名而失败。

三、 实验步骤

1. 首先需要设置抓取的页面范围，同时获取参数，如果参数数量为 3，那么需根据给出的参数来确定页面范围：

```
start_page=1
end_page=188
proc=1
if [ $# = 3 ]; then
    start_page=$1
    end_page=$2
    proc=$3
fi
```

2. 根据需要，之后要创建文件夹和文件共四个，分别是 bing 文件夹，page.txt、image.txt、loaded_image.txt 三个文件；

   其中：

   bing 文件夹用来存放下载的图片；

   page.txt 文件用来存放已经加载过的页面；

   image.txt 用来存放需要下载的图片信息；

   loaded_image.txt 用来存放已经下载过的图片信息。

```
folder="bing"
if [ ! -d $folder ]; then
    mkdir $folder
fi

path="page.txt"
if [ ! -f $path ]; then
    touch $path
fi

path="image.txt"
if [ ! -f $path ]; then
    touch $path
fi

path="loaded_image.txt"
if [ ! -f $path ]; then
    touch $path
fi
```

3. 首先需要进行页面的抓取，遍历整个需要抓取的页面范围，由于已经抓取的页面号记录在 page.txt 中，因此需要找到 page.txt 中没有的页面号进行抓取即可（如果 loaded_page=0 说明没有抓取页面 page）：

```
    for page in $(seq $start_page $end_page)
    do
        loaded_page=0
        for i in $(cat page.txt)
        do
            [ $page = $i ] && loaded_page=1
        done
```

4. 如果 wget 抓取成功则需要对整个页面进行正则处理，取出我们需要的信息保存在 image.txt 中，抓取失败则删除刚刚的临时文件。

```
        if [ $loaded_page -eq 0 ]; then
            wget https://bing.ioliu.cn/?p=$page -O $page.txt
            ret=$?
            if [ $ret -eq 0 ]; then
                echo $page >> page.txt
                cat $page.txt | sed 's/<div class="card progressive">/\n/g' | sed 's/<p class="view">/\n/g'
 | sed 's/src="/\n/g' | sed 's/(.*)//g' | awk '/^http:/ {print}' | sed 's/<[^<>]*>/ /g' | sed 's/">//g' >>
image.txt
            fi
            rm $page.txt
        fi
    done
```

```
<div class="item">
  <div class="card progressive">
    <img class="progressive__img progressive--not-loaded" data-progressive="http://h2.ioliu.cn/bing/CapePerpetua_ZH-CN4150223705_640x480.jpg?imageslim"
    src="http://h2.ioliu.cn/bing/CapePerpetua_ZH-CN4150223705_640x480.jpg?imageslim"> 溢出
    <a class="mark" href="/photo/CapePerpetua_ZH-CN4150223705?force=home_36"></a>
    <div class="description">
      <h3> 溢出
        俄勒冈海岸佩笆角的雷神之井 (© Cavan Images/Offset by Shutterstock)
      </h3>
      <p class="calendar">
        <i class="icon icon-calendar"></i> 溢出
        <em class="t">2021-03-11</em> 溢出
      </p>
      <p class="view">…</p>
    </div>
    <div class="options">…</div>
  </div>
```

最后在 image.txt 中应该是如下效果：

5. 下载图片时采用并发的概念使用如下结构：

```
for num in $(seq 1 $proc)
do
{}&
done
```

这样每一个循环实际上都是在后台进行，因此达到了并发的目的，下载图片的程序需要写在{}内部。

6. 之后需要对 image.txt 文件中的每一个图片进行下载，和下载 html 一样，遍历 image.txt 时需要对 loaded_image.txt 文件进行查看，找到 loaded_image.txt 中没有的图片进行下载，如果下载成功则将这个图片记录写入 loaded_image.txt 中，并将临时文件改名，如果下载失败则删除临时文件：

判断图片是否已经下载：

```
for num in $(seq 1 $proc)
do
{
    IFS=$'\n'
    for i in $(cat image.txt)
    do
        flag=0
        for j in $(cat loaded_image.txt)
        do
            [ $(echo $i | awk '{print $2}') = $(echo $j | awk '{print $2}') ] && flag=1
        done
```

接下来为了确保每一个进程下载不同的图片，故需要给每一个进程分配其需要下载哪一行的图片，这里采用“进程号+n*进程数”的规则，如果是一号线程，且一共有三个线程，那么一号线程需要下载第 1，4，7，……行的图片。

```
        load=$(echo $i | awk '{print $2}')
        load=$(awk '{if($0~"'$load'") {print NR}}' image.txt)
        x=$(expr $load - $num)
        no=$(expr $x % $proc)
```

接下来只有满足行号的要求且图片未被下载过才可以下载图片（需要判断 wget 是否成功）：

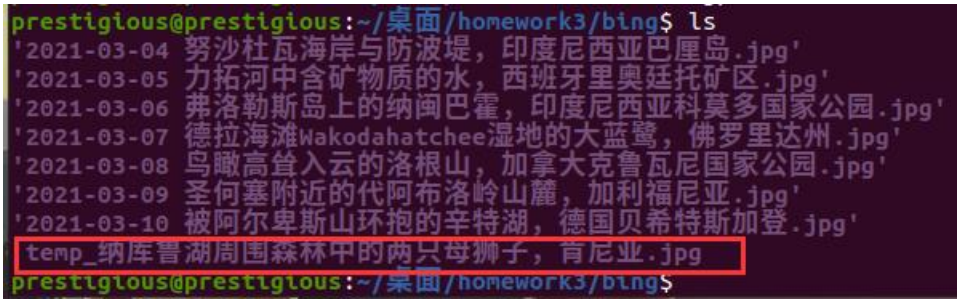
```
        if [ $flag -eq 0 -a $no -eq 0 ]; then
            image_path=$(echo $i | awk '{print $1}')
            image_name=$(echo $i | awk '{print $2}')
            image_time=$(echo $i | awk '{print $3}')
            echo -e "\n$num is loading"
            wget $image_path -O "$folder/temp_$image_name.jpg"
            ret=$?
            if [ $ret -eq 0 ]; then
                echo $i >> loaded_image.txt
                mv "$folder/temp_$image_name.jpg" "$folder/$image_time $image_name.jpg"
            else
                rm "$folder/temp_$image_name.jpg"
            fi
        fi
    done
} &
done
```

四、 实验结果

1. 使用./bing.sh 36 36 1 命令，表示启动 1 个进程下载第 36 页的图片。下载至一半停止：

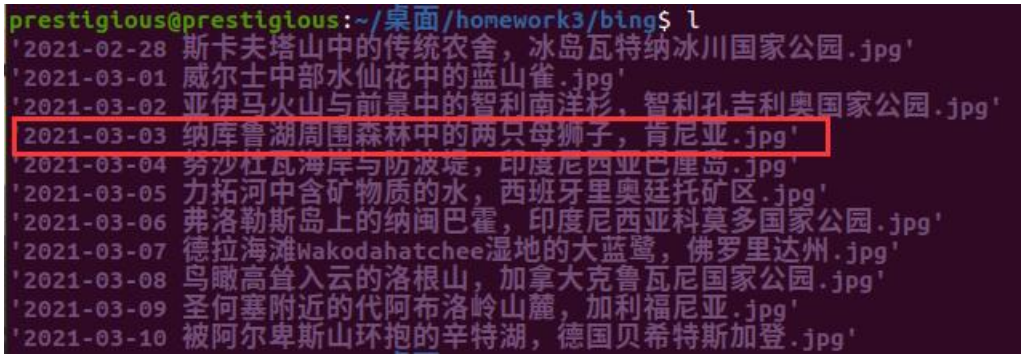
```
正在连接 h2.ioliu.cn (h2.ioliu.cn)|121.51.68.173|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 65167 (64K) [image/jpeg]
正在保存至：“bing/temp_努沙杜瓦海岸与防波堤，印度尼西亚巴厘岛.jpg”

bing/temp_努沙杜瓦  100%[===================>]  63.64K   227KB/s    用时 0.3s

2022-05-06 21:22:36 (227 KB/s) - 已保存 “bing/temp_努沙杜瓦海岸与防波堤，印度尼西亚巴厘岛.jpg” [65167/65167])


1 is loading
--2022-05-06 21:22:36--  http://h2.ioliu.cn/bing/WWDLions_ZH-CN3506997987_640x480.jpg?imageslim
正在解析主机 h2.ioliu.cn (h2.ioliu.cn)... 121.51.68.173, 121.51.52.73
正在连接 h2.ioliu.cn (h2.ioliu.cn)|121.51.68.173|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 56011 (55K) [image/jpeg]
正在保存至：“bing/temp_纳库鲁湖周围森林中的两只母狮子，肯尼亚.jpg”

         bing/temp_  77%[==============>      ]  42.59K   201KB/s              ^Z
[7]+  已停止               ./bing.sh 36 36 1
```

可以看到 bing 文件夹中存在临时文件“temp_”：

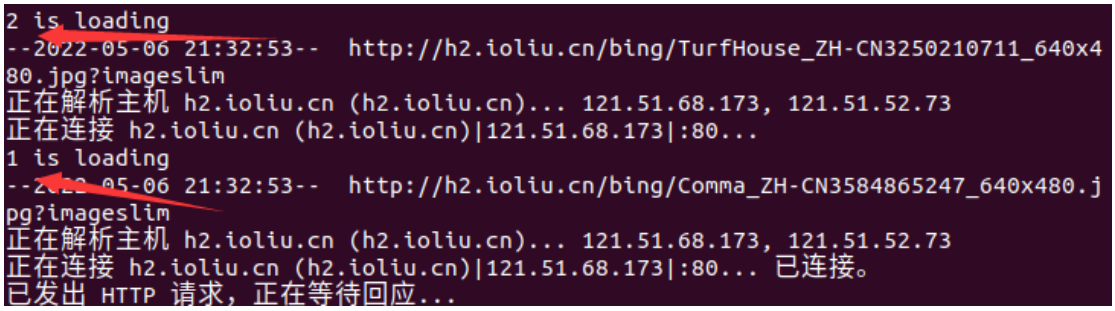


当我们再次使用./bing.sh 36 36 1时：

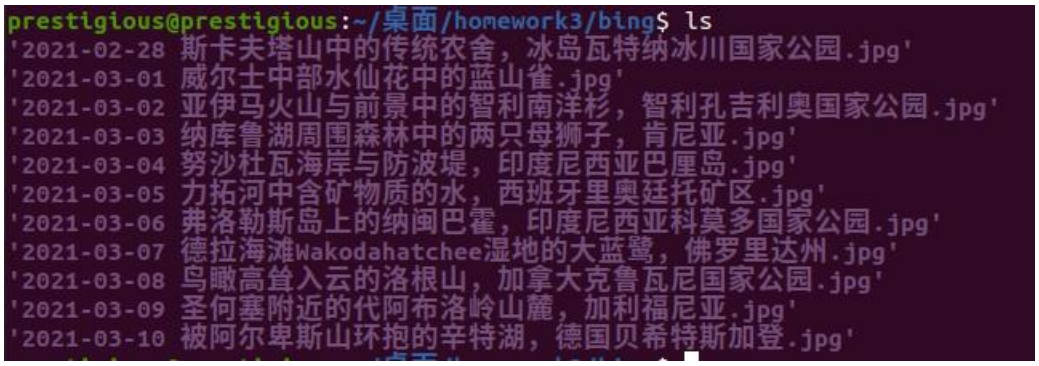
可以看到从这个临时文件开始下载之后的图片了，并且执行完毕后临时文件不见了，只剩下完整的图片了：



2. 使用./bing.sh 36 36 3表示启动三个进程。



可以同时看到三个进程分别下载图片，输出不规律说明它们其实是同时在进行的，最后 bing 文件夹内的图片也和之前相同：



五、 实验总结

本次实验总体来说考察的时脚本程序逻辑结构的编写，而写作规范已经成为必备技能，总体来说包括对于变量和指令的灵活调用，对文件的读写等

等，训练了抓取网页数据的一些技能，可以说收获很大。

第一天还能正常访问图片页面，第二天写报告的时候就不能打开这个网页了，但不是被禁了，几个小时可能能打开一次，不知道之后会不会好转。

六、 源代码

```
start_page=1
end_page=188
proc=1
if [ $# = 3 ]; then
    start_page=$1
    end_page=$2
    proc=$3
fi
# 保存图片
folder="bing"
if [ ! -d $folder ]; then
    mkdir $folder
fi
# 保存已经加载的页号
path="page.txt"
if [ ! -f $path ]; then
    touch $path
fi
# 保存需要下载的图片
path="image.txt"
if [ ! -f $path ]; then
    touch $path
fi
# 保存已经下载的图片
path="loaded_image.txt"
if [ ! -f $path ]; then
    touch $path
fi
# 下载每一个页面并提取图片信息
for page in $(seq $start_page $end_page)
do
```

```
    loaded_page=0
    for i in $(cat page.txt)
    do
        [ $page = $i ] && loaded_page=1
    done
    if [ $loaded_page -eq 0 ]; then
        wget https://bing.ioliu.cn/?p=$page -O $page.txt
        ret=$?
        if [ $ret -eq 0 ]; then
            echo $page >> page.txt
            cat $page.txt | sed 's/<div class="card progressive">/\n/g' | sed 's/<p class="view">/\n/g' | sed 's/src="/\n/g' | sed 's/(.*)//g' | awk '/^http:/{print}' | sed 's/<[^<>]*>/ /g' | sed 's/">//g' >> image.txt
        fi
        rm $page.txt
    fi
done
# 启动线程，并下载图片
for num in $(seq 1 $proc)
do
{
    IFS=$'\n'
    for i in $(cat image.txt)
    do
        flag=0
        for j in $(cat loaded_image.txt)
        do
            [ $(echo $i | awk '{print $2}') = $(echo $j | awk '{print $2}') ] && flag=1
        done
        # 需要判断某行的图片是不是该线程需要下载的
        load=$(echo $i | awk '{print $2}')
        load=$(awk '{if($0~"'$load'") {print NR}}' image.txt)
        x=$(expr $load - $num)
        no=$(expr $x % $proc)
        if [ $flag -eq 0 -a $no -eq 0 ]; then
            image_path=$(echo $i | awk '{print $1}')
            image_name=$(echo $i | awk '{print $2}')
```

```
            image_time=$(echo $i | awk '{print $3}')
            echo -e "\n$num is loading"
            wget $image_path -O "$folder/temp_$image_name.jpg"
            ret=$?
            if [ $ret -eq 0 ]; then
                echo $i >> loaded_image.txt
                mv       "$folder/temp_$image_name.jpg"       "$folder/$image_time
    $image_name.jpg"
            else
                rm "$folder/temp_$image_name.jpg"
            fi
        fi
    done
} &
done

wait
rm page.txt
rm image.txt
rm loaded_image.txt
```