Linux 开发环境及应用实验报告

实验三：Shell 脚本程序设计

付容天

学号 2020211616

班级 2020211310

计算机学院（国家示范性软件学院）

2023 年 4 月 23 日

1 生成 TCP 活动状况报告

命令 netstat －statistics 可以列出 tcp 等协议的统计信息。编写 shell 脚本程序，每隔一分钟生成 1 行信息，包括：当前时间；这一分钟内 TCP 发送了多少报文；接收了多少报文；收发报文总数；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 包以内用空格，否则用符号-）。示例如下（来自实验指导 PPT）：

```
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     -
```

图 1: 生成 TCP 活动状况报告示例图

1.1 关键点设计思路

在本实验中，有许多需要设计的要点，现在简单介绍我对于这些关键点的设计方案，后面再结合具体代码给出更详细的说明：

（1） 当前时间获取：要求输出当前时间，当前时间可以通过 date 命令得到；

（2） TCP 收发报文数据获取：根据实验指导 PPT 内容，可以通过命令 netstat －statistics 获取，并且可以通过选项-tcp 指定筛选出 TCP 数据；

（3） TCP 收发报文总数：通过两次收发报文数作差即可；

（4） 收发报文数增减判断：记录上一分钟收发报文总数，作差，通过 shell 控制中的条件判断语句即可实现。

1.2 详细设计与实验结果

这部分我将详细介绍我的代码，并给出最后运行的实验结果。首先给出详细代码：

```
nowRecv=$(netstat --statistics -tcp | grep 'segments received' |
awk 'NR==1{print $1}')
nowSend=$(netstat --statistics -tcp | grep 'segments sent out' |
awk 'NR==1{print $1}')
msgNum=0
```

```
cnt=0
while true
do
        sleep 60
        nowTime=$(date +"%Y-%m-%d %H:%M")
        lastRecv=$nowRecv
        lastSend=$nowSend
        lastMsgNum=$msgNum
        nowRecv=$(netstat --statistics -tcp | grep 'segments
received' | awk 'NR==1{print $1}')
        nowSend=$(netstat --statistics -tcp | grep 'segments sent
out' | awk 'NR==1{print $1}')

        let cnt+=1
        recv=$(expr $nowRecv - $lastRecv)
        send=$(expr $nowSend - $lastSend)
        msgNum=$(expr $recv + $send)

        num=$(expr $msgNum - $lastMsgNum)
        if [ $num -gt 10 ]; then
         sign="+"
        elif [ $num -lt -10 ]; then
         sign="-"
        else
         sign=" "
        fi

        if [ $cnt -eq 1 ]; then
         sign=" "
        fi

        printf "%-s %-s %6s %6s %6s %6s\n" $nowTime $recv $send
$msgNum $sign
done
```

其中，部分变量含义：（1）nowRecv：当前收到的报文总数；（2）now-Send：当前发送的报文总数；（3）msgNum：一分钟内收发报文总数；（4）cnt：记录输出行数；（5）lastRecv：记录上一次的 nowRecv；（6）lastSend：记录上一次的 nowSend；（7）recv：当前一分钟内收到的报文总数；（8）send：当前一分钟内发送的报文总数；（9）sign：收发报文数增减变化符号。

为什么设计了 nowRecv/nowSend、lastRecv/lastSend 和 recv/send 三组变量？这是因为命令 netstat -statistics -tcp 获取的信息只包含了收发报文

的总数，需要我们自己作差才能得到一分钟内的收发报文总数，如下所示：

```
c1616@Ubuntu-bupt:~$ netstat --statistics -tcp
IcmpMsg:
    InType0: 235166
    InType3: 2110156
    InType8: 463249
    InType11: 1
    InType13: 2
    OutType0: 463249
    OutType3: 2110144
    OutType8: 470482
    OutType14: 2
Tcp:
    335237 active connection openings
    4385 passive connection openings
    9063 failed connection attempts
    8050 connection resets received
    22 connections established
    7099988 segments received
    7467365 segments sent out
    51235 segments retransmitted
    89 bad segments received
    18258 resets sent
```

图 2：segments received 和 segments send out 两个字段给出了相关信息

相应地，想要从中提取出这两个字段，可以使用 grep 指令（如上述代码第 1 行和第 2 行所示）；在代码的主循环中，我设计了 sleep 60 来实现“每隔一分钟输出一次统计信息”的效果；在收发报文数量增减判断中，我使用 shell 中的控制逻辑“if-elif-else-fi”来实现符号（即上述代码中的变量 sign）的赋值，并在随后输出相应的统计信息；在输出的时候，我还设计了输出的格式化（详见代码）。

将上述代码编写进脚本文件 tcpLab.sh 中，使用 bash tcpLab.sh 命令执行，结果如下：

```
c1616@Ubuntu-bupt:~$ bash tcpLab.sh
2023-04-23 15:37   2913  12333  15246
2023-04-23 15:38    709    669   1378      -
2023-04-23 15:39    688    591   1279      -
2023-04-23 15:40    477    488    965      -
2023-04-23 15:41    812   1716   2528      +
2023-04-23 15:42    332    312    644      -
2023-04-23 15:43    480    449    929      +
2023-04-23 15:44   2903   4564   7467      +
2023-04-23 15:45   8577  13253  21830      +
2023-04-23 15:46   6137   9832  15969      -
2023-04-23 15:47    701    667   1368      -
2023-04-23 15:48   5031   9250  14281      +
2023-04-23 15:49   1309   1786   3095      -
2023-04-23 15:50    253    246    499      -
2023-04-23 15:51    187    191    378      -
2023-04-23 15:52    293    283    576      +
2023-04-23 15:53    470    403    873      +
2023-04-23 15:54    595    531   1126      +
2023-04-23 15:55    414    356    770      -
2023-04-23 15:56    283    237    520      -
```

图 3：生成 TCP 活动状况报告输出结果

1.3 实验问题与实验总结

在本次实验中，我也遇到了一些问题，现将问题与解决方法记录如下：

（1） TCP 消息提取问题：一开始我并不知道 netstat 命令可以增加-tcp 选项，这导致输出数据过多。我通过查阅资料，解决了这个问题；

（2） Windows 和 Linux 下文件换行符不同：这导致我在 Windows 环境下编写的脚本文件传到 Linux 系统上无法执行（报错 syntax error near unexpected token `elif'），通过查阅相关资料，我使用 notepad++更改了文件编码，解决了这个问题。

在本次实验中，我练习了 shell 脚本编程技术，掌握了 bash 脚本相关知识，并且熟练地使用之前所学内容（例如 grep）从而完成了综合的实验任务，同时了解了 Windows 和 Linux 系统对文件编码的差异。总的来说，收获颇丰！

2 下载 Bing 图库中的图片

访问 https://bing.ioliu.cn/?p=36 可以看到 Bing 图库第 36 页的内容，这个 Web 页有多个图片小样，将鼠标放到某个小样上，如右上角，可见中文说明信息“野花草甸上的一只欧亚雕鸮，德国莱茵兰-普法尔茨”和日期信息 2019-08-03，点击一下，此图片就可以下载。

本实验要求编写脚本程序 bing.sh，将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

本实验要求：

（1） 不重复下载已下载的图片：检查图片是否下载，如果已下载，则不再下载；支持批量任务在被中断后再次启动程序可以从中断处继续；

（2） 考虑并发：启动多个进程同时下载，可以加快任务完成的速度；

（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，进而判断 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。

2.1 关键点设计思路

在本实验中，有许多需要设计的要点，现在简单介绍我对于这些关键点的设计方案，后面再结合具体代码给出更详细的说明：

（1） 图片描述信息获取：我们需要使用图片描述信息对下载后的图片进行重命名，而图片的描述信息可以通过在 html 文件中检索相应字段来获取；

（2） 多线程下载实现思路：关键是要避免临时文件名重复，从而避免重复下载同一张图片。我的做法是在临时文件名中加上当前进程 ID 号（由 shell 中“$$”符号给出），并在下载时进行判断；

（3） 避免重复下载：通过 shell 的 if 语句检查是否存在相应的图片文件（if [! -f “$img”]）来实现已下载文件的检测；

（4） 考虑下载文件出现故障的情况：使用实验指导 PPT 上介绍的方法，引入一个临时文件，并在下载完成后再转移到目文件夹并删除临时文件。

2.2 详细设计与实验结果

首先看如何从 html 中提取出图片的描述信息，利用 Chrome 中的网页源码查看工具，可以定位到图片的描述信息如下图所示：

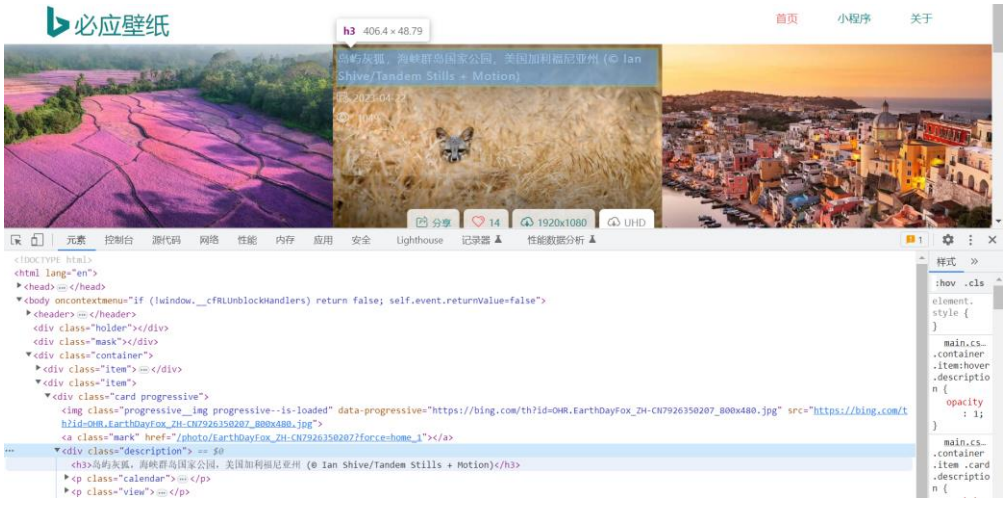

图 4：在 html 中定位图片描述信息

之后看如何下载图片，同样使用 Chrome 中的网页源码查看工具，发现图片的下载链接在 href 中给出，如下图所示：

图 5：在 html 中定位图片下载链接

接下来我将结合具体的代码进行分析，完整源代码如下：

```
#!/bin/bash
START=1
END=100

if [ ! -d bing ]; then
    mkdir bing
fi

if [ ! -d html ]; then
    mkdir html
fi

if [ ! -d temp ]; then
    mkdir temp
fi

for i in $(seq $START $END)
do
    if [ ! -f html/$i/html ]; then
        wget -O html/$i.html https://bing.ioliu.cn/?p=$i
    fi

    if [ ! -f temp/$i.dat ]; then
        cat html/$i.html | \
            sed -e 's/<div class="item">/\n/g' | \
            grep "<div class=\"card progressive\"" | \
            sed -e 's#.*\"mark\"
href=\"\(.*\)\".*\"description\"><h3>\(.*\)(©.*\([0-9]\{4\}\-[0-
9]\{2\}-[0-9]\{2\}\)<.*#\1$\2$\3#g' \
            -e 's#home_\([0-9]*\)#download#g' \
            -e 's# $2#$2#g' > temp/$i.dat
    fi
done
```

```
for i in $(seq $START $END)
do
    while read line
    do
        eval $(echo "$line" | awk -F "[$]"
'{printf("url=https://bing.ioliu.cn%s;name=\"%s\";date=%s",$1,$2,$3
)}')
        img="bing/$date $name.jpg"
        if [ ! -f "$img" ]; then
            tmp="img.tmp$$"
            wget -O "$tmp" $url
            if [ $? = 0 -a ! -f "$img" ]; then
                mv "$tmp" "$img"
            else
                rm "$tmp"
            fi
        fi
    done < temp/$i.dat
done
```

代码一开始的 START 和 END 是网页图片下载的起始页码和终止页码。

我为这个脚本程序设计了三个处理文件夹，如下所示：

（1） bing 文件夹：存放最终处理完毕的图片（合理命名的 jpg 形式图片）；

（2） html 文件夹：存放网页每一页对应的 html 文件，供程序运行使用；

（3） tmp 文件夹：存放从网页每一页的 html 文件中抽取出的图片相关信息。

在代码的一开始，首先判断这三个文件夹是否创建，如果没有创建则首先创建这三个文件夹。

接着执行循环，每次下载一页的 html，并存放在 html 文件夹中。这里也设计了重复性判断（if [! -f html/$i.html]），如果当前页面已经下载过了则不再下载，这是为了多线程执行考虑。并且，在每一次下载完对应页面的 html 文件之后，使用 sed 和 grep 等方法提取出图片的描述信息（去除信息中不必要的部分），存入 dat 文件中。

然后执行循环，每次循环处理对应页面的 html 并完成图片下载、图片重命名和图片移动等操作，具体流程如下：

（1） 使用 eval 指令执行需要多次替换的指令，打印出图片信息；

（2） 对待下载图片进行命名，命名结果为 img（时间+描述信息）；

（3） 检查该图片是否已经下载，这是为了多线程考虑。如果没有下载，则使

用 wget 命令下载图片，临时图片文件的命名为 img.tmp$$，其中“$$”表示当前进程的 ID，这也是为了多线程而设计的，避免出现文件名冲突；

（4） 下载完成后再次判断图片是否重复，若不重复则将下载文件剪切到目标文件夹 bing 中；如果重复则不进行剪切操作，并删除对应的图片。

使用命令 bing.sh 运行该脚本文件（单线程），输出结果如下：

```
c1616@Ubuntu-bupt:~/bingLab$ cd bing
c1616@Ubuntu-bupt:~/bingLab/bing$ ls
'2023-04-11 哥伦比亚河峡谷，俄勒冈州，美国.jpg'    '2023-04-18 马丘比丘，秘鲁.jpg'
'2023-04-12 从国际空间站拍摄的地球.jpg'            '2023-04-19 褐头凤鹛.jpg'
'2023-04-13 斯诺登尼亚国家公园，威尔士，英国.jpg'  '2023-04-20 克雷斯特德比特山上方的月食，科罗拉多州，美国.jpg'
'2023-04-14 红海星，地中海.jpg'                    '2023-04-21 普罗奇达岛，意大利.jpg'
'2023-04-15 从纳哈加尔城堡鸟瞰斋浦尔，印度.jpg'    '2023-04-22 岛屿灰狐，海峡群岛国家公园，美国加利福尼亚州.jpg'
'2023-04-16 阿德莱德国际风筝节，澳大利亚.jpg'      '2023-04-23 日出时分薄雾笼罩下的薰衣草田，印度.jpg'
'2023-04-17 布列塔尼的小米努灯塔，法国.jpg'        '2023-04-24 巴伐利亚森林酒窖，德国.jpg'
c1616@Ubuntu-bupt:~/bingLab/bing$
```

图 6：单线程下载结果

使用选项 nproc=5 启动五个线程进行下载，结果如下：

```
c1616@Ubuntu-bupt:~/bingLab$ cd bing
c1616@Ubuntu-bupt:~/bingLab/bing$ ls
'2023-03-21 彩色粉笔.jpg'                          '2023-04-09 复活节彩蛋.jpg'
'2023-03-23 杜费里峡谷，上萨瓦省，法国.jpg'        '2023-04-10 安博塞利国家公园的大象，肯尼亚.jpg'
'2023-03-24 盛开的野蒜，海尼希国家公园，德国.jpg'  '2023-04-11 哥伦比亚河峡谷，俄勒冈州，美国.jpg'
'2023-03-25 塞西尔布鲁尔楼梯，伦敦，英国.jpg'      '2023-04-12 从国际空间站拍摄的地球.jpg'
'2023-03-26 安沙波利哥沙漠州立公园的野花，加利福尼亚州，美国.jpg'  '2023-04-13 斯诺登尼亚国家公园，威尔士，英国.jpg'
'2023-03-27 云层中的纽约市天际线.jpg'              '2023-04-14 红海星，地中海.jpg'
'2023-03-28 意大利三峰山上空的银河.jpg'            '2023-04-15 从纳哈加尔城堡鸟瞰斋浦尔，印度.jpg'
'2023-03-29 两只海牛，佛罗里达州的水晶河，美国.jpg'  '2023-04-16 阿德莱德国际风筝节，澳大利亚.jpg'
'2023-03-30 孔雀羽毛.jpg'                          '2023-04-17 布列塔尼的小米努灯塔，法国.jpg'
'2023-03-31 斯太尔河，奥地利.jpg'                  '2023-04-18 马丘比丘，秘鲁.jpg'
'2023-04-02 爪哇岛东部的婆罗摩火山，印度尼西亚.jpg'  '2023-04-19 褐头凤鹛.jpg'
'2023-04-03 大岛上的霍璃璃国家历史公园，夏威夷.jpg'  '2023-04-20 克雷斯特德比特山上方的月食，科罗拉多州，美国.jpg'
'2023-04-05 杭州西湖水墨意境般的风景，浙江省，中国.jpg'  '2023-04-21 普罗奇达岛，意大利.jpg'
'2023-04-06 月亮升起，图森，亚利桑那州，美国.jpg'  '2023-04-22 岛屿灰狐，海峡群岛国家公园，美国加利福尼亚州.jpg'
'2023-04-07 欧亚河狸宝宝，芬兰.jpg'                '2023-04-23 日出时分薄雾笼罩下的薰衣草田，印度.jpg'
'2023-04-08 巨人之路，北爱尔兰，英国.jpg'           '2023-04-24 巴伐利亚森林酒窖，德国.jpg'
c1616@Ubuntu-bupt:~/bingLab/bing$
```

图 7：多线程下载结果

并且，可以发现对之前已经下载过的文件不会再次下载，满足实验要求。

在 secureCRT 软件中使用 sz 命令，可以方便地将下载的图片保存到本地，结果如下图所示：

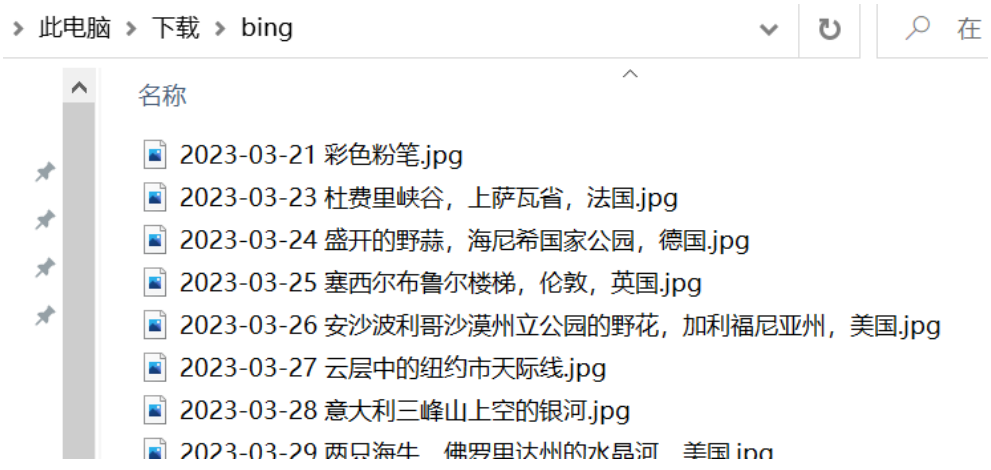

图 8：保存到本地的结果

2.3 实验问题与实验总结

在本次实验中，我也遇到了一些问题，现将问题与解决方法记录如下：

（1） 关于 url 复制错误：我在实验中发现，如果复制粘贴网站的 url 地址，则会出现编码问题，即使显示效果是一样的，其具体编码也会不一样（尤其是当 url 中有“？”等特殊字符时），导致在执行指令的过程中参数判断错误。解决方案有两种：更改字符集编码或手动输入 url 地址；

（2） eval 指令的使用：有些指令仅仅经过一次替换是不够的，这时就需要使用 eval 指令。

总的来说，在本次实验中，我通过分析和编写 bing.sh 脚本文件，实现了从指定网站上抓取图片并进行重命名和移动的工作，练习了 sed、grep 等指令的使用，加深了对脚本程序的理解，本次实验我收获满满。