

《计算机系统工程》实验报告

- 姓名：王骏
- 学号：2202000 7104
- 日期：2025-03-11

✧ 实验内容

- 1 文件系统相关命令：
ls、cd、pwd、mkdir、touch、cp、mv、rm、cat、find、grep、
more、less、head、tail、ln、stat、file
- 2 管道：|
- 3 权限相关命令：chmod、chown、chgrp，特殊权限：suid、sgid、sbit
- 4 内存相关命令：free、top、ps、pmap、vmstat、sar，
- 5 内存信息：/proc/pid/maps、/proc/pid/smaps
- 6 网络相关命令：ifconfig、netstat、curl
- 7 shell脚本

✧ 实验目的

- 1 继续学习Linux操作系统shell中文件系统、管道、权限、设备、内存、网络相关的命令（程序）
- 2 思考这些程序背后调用的操作系统内核的系统调用、开发者考虑的问题
- 3 为能够在后续实验中进行实验环境部署打下基础

❖ 实验步骤

文件系统及管道

执行以下命令并说明其功能：ls /home | grep "^test"

- `ls /home`：列出 `/home` 目录下的所有文件和目录。
- `|`：管道符号，将 `ls` 的输出传递给 `grep`。
- `grep "^test"`：从 `ls` 的输出中筛选出以 `test` 开头的行。

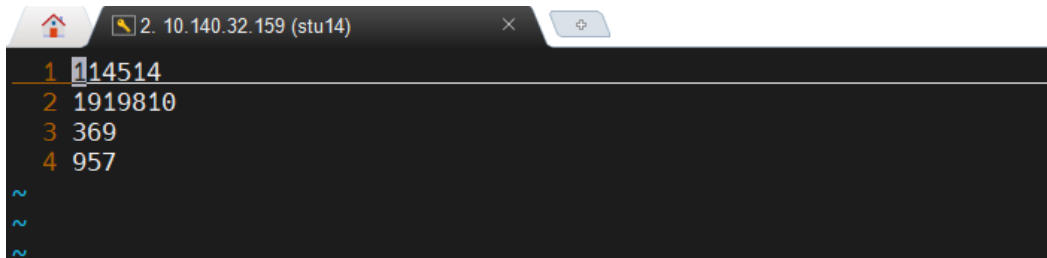
```
stu14@5aa3e8799848:~$ ls /home | grep "^test"
test00
```

这里只有一个文件 `test00` 是 `test` 开头的文件/目录

使用管道命令将"cat" 命令的输出传递给"head" 命令，显示某个文件的前4行内容。

- 随便建一个文件 `1.txt`，然后写四行内容

```
1 vim 1.txt
```



```
1 14514
2 1919810
3 369
4 957
~
~
~
```

- 打印文件

```
1 cat 1.txt | head -n 4
```

```
stu14@5aa3e8799848:~$ cat 1.txt | head -n 4
114514
1919810
369
957
```

- 可以看到输出了前四行内容

使用管道命令将/lib目录下的文件按字母顺序对文件名进行排序

```
1 ls /lib | sort
```

```
stu14@5aa3e8799848:~$ ls /lib | sort
apt
bfd-plugins
binfmt.d
compat-ld
cpp
dbus-1.0
debug
dpkg
environment.d
file
gcc
git-core
gnupg
gnupg2
gold-ld
groff
i386-linux-gnu
init
jvm
kernel
ld-linux.so.2
libsupp.a
locale
lsb
man-db
mime
modprobe.d
modules-load.d
networkd-dispatcher
openssh
os-release
pkgconfig
pkg-config.multiarch
python2.7
python3
```

思考：为什么两个命令之间不能使用变量或者内存直接传递数据

管道的设计原理

- 基于流的数据传递：管道是一种单向的、流式的数据传输机制。它只能传递字符流（文本数据），而不能直接传递复杂的变量或内存对象。
- 标准输入/输出的限制：管道只能操作标准输入和标准输出，而变量和内存数据通常存储在进程的私有内存空间中，无法直接通过管道传递。

变量和内存的作用域

- 进程隔离：每个命令在运行时都是一个独立的进程，进程之间是隔离的。一个进程的变量或内存数据无法直接访问另一个进程的内存空间。
- 作用域限制：变量通常只在当前 Shell 或进程中有效，无法跨越进程边界直接传递。

使用man学习chown和chgrp的功能及用法

```
1 chown user1 1.txt
2 chown user1:group1 1.txt
3 chown -R stu14 data/
4 chown --reference=1.txt 2.txt
5 chgrp group1 1.txt
```

在桌面下创建一个目录**data**，使用**chmod**命令将目录**data**设置为**STICKYBIT**权限，从而使得在该目录下创建的文件只能被其所有者和超级用户删除。说明：该权限位什么含义？

```
1 mkdir data
2 chmod +t data/
```

- STICKY BIT 是一种特殊的目录权限，用于限制文件的删除和重命名操作。
- 只有文件的所有者、目录的所有者或超级用户才能删除或重命名设置了STICKY BIT 的目录下的文件。

使用**chown**命令将文件"**file.txt**"的所有者更改为**root**。

```
1 sudo chown -R root data/
```

使用**chgrp**命令递归地将目录及其所有子目录下的文件和目录的用户组设置为目标用户组

```
1 grep dev data/
```

```
stu14@5aa3e8799848:~$ sudo chown -R root file.txt
[sudo] password for stu14:
stu14 is not in the sudoers file. This incident will be reported.
stu14@5aa3e8799848:~$ sudo chown -R root 1.txt
[sudo] password for stu14:
stu14 is not in the sudoers file. This incident will be reported.
```

使用free命令查看系统的内存使用情况，并解释其中的列名称和含义

```
stu14@5aa3e8799848:~$ free
              total        used         free       shared    buff/cache   available
Mem:          32860596       3040244       20838940        26728       8981412       29319268
Swap:          8388604         39368         8349236
```

- 1 total：系统可用的总物理内存量。
- 2 used：当前已使用的物理内存量。
- 3 free：当前未使用的交换空间大小。
- 4 shared：被多个进程共享的物理内存量。
- 5 buff/cache：这部分内存可以被回收，用于满足应用程序的需求。
- 6 available：系统当前可用的内存量。

使用top命令查看系统的内存占用情况，并解释其中的内存使用率显示

```
stu14@5aa3e8799848:~$ top
top - 19:47:37 up 109 days, 20:09, 1 user, load average: 0.18, 0.21, 0.25
Tasks: 11 total, 1 running, 10 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.4 us, 0.8 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 32090.4 total, 20309.0 free, 2966.0 used, 8815.5 buff/cache
MiB Swap: 8192.0 total, 8153.6 free, 38.4 used. 28635.1 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	6980	3448	3184	S	0.0	0.0	0:04.82	init.sh
34	root	20	0	12196	4164	3236	S	0.0	0.0	0:00.00	sshd
38	root	20	0	6824	2436	2180	S	0.0	0.0	0:00.46	cron
13895	root	20	0	13164	8332	7184	S	0.0	0.0	0:00.04	sshd
13904	root	20	0	13164	8252	7100	S	0.0	0.0	0:00.03	sshd
13907	stu14	20	0	13720	6412	5044	S	0.0	0.0	0:00.26	sshd
13917	stu14	20	0	13404	5176	4004	S	0.0	0.0	0:00.03	sshd
13918	stu14	20	0	5896	4220	3768	S	0.0	0.0	0:00.02	sftp-server
13920	stu14	20	0	7244	3920	3348	S	0.0	0.0	0:00.08	bash
14875	root	20	0	5484	508	448	S	0.0	0.0	0:00.00	sleep
14876	stu14	20	0	9072	3600	3092	R	0.0	0.0	0:00.01	top

内存使用率全是0.0，估计是太小了，显示的尾数显示不了

使用ps命令查看当前系统中内存占用最高的进程，并显示其进程ID和内存占用

```
1 ps aux --sort=-%mem | head -n 5
```

```
stu14@5aa3e8799848:~$ ps aux --sort=-%mem | head -n 5
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      13895  0.0  0.0  13164  8332 ?        Ss   18:35   0:00 sshd: stu14 [priv]
root      13904  0.0  0.0  13164  8252 ?        Ss   18:35   0:00 sshd: stu14 [priv]
stu14     13907  0.0  0.0  13720  6412 ?        S    18:35   0:00 sshd: stu14@pts/0
stu14     13917  0.0  0.0  13404  5176 ?        S    18:35   0:00 sshd: stu14@notty
```

- PID 13895
- %MEM = 0.0% 应该是太小了，后面小数位没显示

使用**pmap**命令查看指定进程的内存映射情况，并解释输出中的地址范围和映射类型

```

stu14@5aa3e8799848:~$ pmap -x 13920
13920:  -bash
Address      Kbytes      RSS      Dirty Mode  Mapping
000055b11146a000    180      180        0 r---- bash
000055b111497000    708      708        0 r-x-- bash
000055b111548000    220     152        0 r---- bash
000055b11157f000     16      16       16 r---- bash
000055b111583000     36      36      36 rw--- bash
000055b11158c000     40      28      28 rw--- [ anon ]
000055b1262cd000   396     312     312 rw--- [ anon ]
00007f8da85f0000     12      12        0 r---- libnss_files-2.31.so
00007f8da85f3000     28      28        0 r-x-- libnss_files-2.31.so
00007f8da85fa000      8       8        0 r---- libnss_files-2.31.so
00007f8da85fc000      4       4        4 r---- libnss_files-2.31.so
00007f8da85fd000      4       4        4 rw--- libnss_files-2.31.so
00007f8da85fe000     24       0        0 rw--- [ anon ]
00007f8da860c000  2968     384        0 r---- locale-archive
00007f8da88f2000     12       8        8 rw--- [ anon ]
00007f8da88f5000    136     136        0 r---- libc-2.31.so
00007f8da8917000   1504    1164        0 r-x-- libc-2.31.so
00007f8da8a8f000    312     184        0 r---- libc-2.31.so
00007f8da8add000     16      16       16 r---- libc-2.31.so
00007f8da8ae1000      8       8        8 rw--- libc-2.31.so
00007f8da8ae3000     16      16       16 rw--- [ anon ]
00007f8da8ae7000      4       4        0 r---- libdl-2.31.so
00007f8da8ae8000      8       8        0 r-x-- libdl-2.31.so
00007f8da8aea000      4       0        0 r---- libdl-2.31.so
00007f8da8aeb000      4       4        4 r---- libdl-2.31.so
00007f8da8aec000      4       4        4 rw--- libdl-2.31.so
00007f8da8aed000     56      56        0 r---- libtinfo.so.6.2
00007f8da8afb000     60      60        0 r-x-- libtinfo.so.6.2

00007f8da8b0a000     56      56        0 r---- libtinfo.so.6.2
00007f8da8b18000     16      16       16 r---- libtinfo.so.6.2
00007f8da8b1c000      4       4        4 rw--- libtinfo.so.6.2
00007f8da8b1d000      8       8        8 rw--- [ anon ]
00007f8da8b20000     28      28        0 r--s- gconv-modules.cache
00007f8da8b27000      4       4        0 r---- ld-2.31.so
00007f8da8b28000    140     140        0 r-x-- ld-2.31.so
00007f8da8b4b000     32      32        0 r---- ld-2.31.so
00007f8da8b54000      4       4        4 r---- ld-2.31.so
00007f8da8b55000      4       4        4 rw--- ld-2.31.so
00007f8da8b56000      4       4        4 rw--- [ anon ]
00007ffeb0d2a000    132     108     108 rw--- [ stack ]
00007ffeb0df8000     16       0        0 r---- [ anon ]
00007ffeb0dfc000      8       4        0 r-x-- [ anon ]
fffffffffff60000      4       0        0 --x-- [ anon ]

-----
total kB          7248    3952    604

```

- 地址范围: 000055b11146a000 - ffffffffffff600000

- 映射类型：
 - ① `bash` : Bash 进程的可执行文件。
 - ② `anon` : 匿名内存映射。
 - ③ `libnss_files-2.31.so` : GNU C 库 (glibc) 中的 `libnss_files` 共享库。
 - ④ `locale_archive` : 系统本地化 (Locale) 数据库文件。
 - ⑤ `libdl-2.31-so` : GNU C 库 (glibc) 中的 `libdl` 共享库。
 - ⑥ `libtinfo.so.6.2` : 终端信息库 (`libtinfo`) 的共享库。

使用 `vmstat` 命令实时监控系统的虚拟内存使用情况，并解释输出中的各列含义

```
stu14@5aa3e8799848:~$ vmstat
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi   bo    in   cs  us  sy  id  wa  st
 0  0   39368 2076772 1313908 7721620    0    0     0    0    3    0   0   0   0 100   0  0
```

procs

- ① `r`:
 - 含义：等待运行的进程数（运行队列长度）。
 - 说明：如果该值持续较高，说明 CPU 可能成为瓶颈。
- ② `b`:
 - 含义：处于不可中断睡眠状态的进程数（通常为等待 I/O 的进程）。
 - 说明：如果该值较高，说明系统可能遇到 I/O 瓶颈。

memory

- ① `swpd`:
 - 含义：使用的交换分区大小（以 KB 为单位）。
 - 说明：如果该值较高，说明物理内存可能不足。
- ② `free`:
 - 含义：空闲的物理内存大小（以 KB 为单位）。
 - 说明：如果该值较低，说明物理内存可能不足。
- ③ `buff`:

- 含义：用于缓冲区的内存大小（以 KB 为单位）。
- 说明：缓冲区用于存储文件系统元数据。

4 **cache:**

- 含义：用于缓存的内存大小（以 KB 为单位）。
- 说明：缓存用于存储从磁盘读取的文件数据。

swap

1 **si:**

- 含义：从交换分区读取到内存的数据量（以 KB/s 为单位）。
- 说明：如果该值较高，说明物理内存可能不足。

2 **so:**

- 含义：从内存写入到交换分区的数据量（以 KB/s 为单位）。
- 说明：如果该值较高，说明物理内存可能不足。

io

1 **bi:**

- 含义：从块设备读取的数据量（以 KB/s 为单位）。
- 说明：块设备通常是磁盘。

2 **bo:**

- 含义：写入块设备的数据量（以 KB/s 为单位）。
- 说明：块设备通常是磁盘。

system

1 **in:**

- 含义：每秒的中断次数。
- 说明：包括时钟中断和硬件中断。

2 **cs:**

- 含义：每秒的上下文切换次数。
- 说明：上下文切换是 CPU 从一个进程切换到另一个进程的过程。

cpu

1 us:

- 含义：用户空间占用 CPU 时间的百分比。
- 说明：用于运行用户进程。

2 sy:

- 含义：内核空间占用 CPU 时间的百分比。
- 说明：用于运行内核进程。

3 id:

- 含义：空闲 CPU 时间的百分比。
- 说明：CPU 未执行任何任务。

4 wa:

- 含义：等待 I/O 操作占用 CPU 时间的百分比。
- 说明：如果该值较高，说明系统可能遇到 I/O 瓶颈。

5 st:

- 含义：虚拟机偷取 CPU 时间的百分比。
- 说明：仅在虚拟化环境中有效，表示虚拟机等待物理 CPU 的时间。

使用sar命令查看系统的内存利用率，并解释输出中的各列含义

实验系统没有sudo权限，下载不了sar指令对应包

```
wj@wj-VMware-Virtual-Platform:~$ sar -r 1 5
Linux 6.11.0-13-generic (wj-VMware-Virtual-Platform)      2025年03月06日   _x86_64_(2 CPU)

20时28分12秒 kbmemfree  kbavail  kbmempd  %mempd  kbbuffers  kbcached  kbcommit  %commit  kbactive  kbinact  kbdirty
20时28分13秒   1618048    2209128    766004     22.30     38992    764524   4117252     58.08   1064264   208508     208
20时28分14秒   1618048    2209128    766004     22.30     38992    764524   4117392     58.08   1064360   208508     208
20时28分15秒   1618048    2209128    766004     22.30     38992    764524   4117392     58.08   1064360   208508     208
20时28分16秒   1618048    2209128    766004     22.30     38992    764524   4117392     58.08   1064360   208508     208
20时28分17秒   1618048    2209128    766004     22.30     38992    764524   4117392     58.08   1064360   208508     208
平均时间:    1618048    2209128    766004     22.30     38992    764524   4117364     58.08   1064341   208508     208
```

内存使用列

1 kbmemfree:

- 含义：空闲的物理内存大小（以 KB 为单位）。
- 说明：如果该值较低，说明物理内存可能不足。

2 kbmempd:

- 含义：已使用的物理内存大小（以 KB 为单位）。
- 说明：包括应用程序和内核占用的内存。

3 **%memused:**

- 含义：物理内存使用率（百分比）。
- 说明：如果该值较高，说明物理内存可能不足。

4 **kbbuffers:**

- 含义：用于缓冲区的内存大小（以 KB 为单位）。
- 说明：缓冲区用于存储文件系统元数据。

5 **kbcached:**

- 含义：用于缓存的内存大小（以 KB 为单位）。
- 说明：缓存用于存储从磁盘读取的文件数据。

内存承诺列

1 **kbcommit:**

- 含义：当前工作负载需要的内存总量（以 KB 为单位）。
- 说明：包括物理内存和交换分区。

2 **%commit:**

- 含义：当前工作负载需要的内存占总内存的百分比。
- 说明：如果该值较高，说明系统可能需要更多内存。

活动内存列

1 **kbactive:**

- 含义：活跃的内存大小（以 KB 为单位）。
- 说明：活跃内存是正在使用的内存。

2 **kbinact:**

- 含义：非活跃的内存大小（以 KB 为单位）。
- 说明：非活跃内存是未使用但尚未释放的内存。

3 **kbdirty:**

- 含义：脏页大小（已被修改但未写回磁盘的内存大小）。
- 说明：如果该值较高，说明系统可能需要更多的磁盘 I/O。

使用netstat命令查看当前系统的网络连接状态，并且只显示TCP连接

```
stu14@5aa3e8799848:~$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:39025         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh            0.0.0.0:*               LISTEN
tcp        0      0 5aa3e8799848:ssh      10.142.220.176:54538     ESTABLISHED
tcp        0      0 5aa3e8799848:ssh      10.142.220.176:54509     ESTABLISHED
tcp6       0      0 [::]:ssh               [::]:*                  LISTEN
```

自学常用的binutils工具

```
1  ld -o output_file input_file1.o input_file2.o
2  as -o output_file.o input_file.s
3  objdump -d binary_file
4  nm binary_file
5  strip binary_file
6  readelf -h binary_file
7  ar rcs libname.a file1.o file2.o
8  ranlib libname.a
```

shell为何知道用sh来解析这个文件？

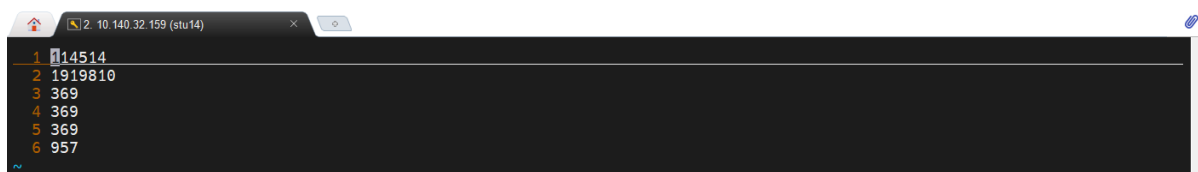
在脚本文件的第一行，会包含一个 **Shebang** 行，用于指定解释器，或者根据拓展名也可以知道用sh来解析

```
1  #!/bin/sh
```

编写一个Shell脚本，统计一个文件中某个关键词出现的次数

```
1  #!/bin/bash
2
3  if [ $# -ne 2 ]; then
4      echo "用法: $0 <文件名> <关键词>"
5      exit 1
6  fi
7
8  file_name=$1
9  keyword=$2
10
11 count=$(grep -o -w "$keyword" "$file_name" | wc -l)
12
13 echo "文件 '$file_name' 中关键词 '$keyword' 出现的次数: $count"
```

编写一个txt文件，写入若干内容，注意此时关键字为一个字符串，如114514是一个关键字。



执行shell脚本，参数为1.txt 369,表示从1.txt 找到关键字369，以及统计数量

```
stu14@5aa3e8799848:~$ ./count_keywords.sh 1.txt 369
文件 '1.txt' 中关键词 '369' 出现的次数: 3
```

分析程序

```
1  #!/bin/bash
2  echo "100" > A.txt
3  echo "100" > B.txt
4  transfer() {
5      b=$(tail -n 1 $2.txt)
6      a=$(tail -n 1 $1.txt)
7      ((a -= $3))
8      ((b += $3))
9      echo "$a" >> $1.txt
10     echo "$b" >> $2.txt
11     echo "$a + $b = $((a+b))"
12 }
13 for ((i=0; i<7; i++)); do
14     transfer A B 1 $i &
15     transfer B A 1 $i &
```

```
16 done
17 wait
```

```
stu14@5aa3e8799848:~$ chmod 777 1.sh
stu14@5aa3e8799848:~$ ./1.sh
99 + 101 = 200
99 + 101 = 200
100 + 101 = 201
99 + 101 = 200
99 + 101 = 200
100 + 102 = 202
100 + 101 = 201
100 + 100 = 200
100 + 102 = 202
99 + 101 = 200
99 + 101 = 200
100 + 102 = 202
100 + 100 = 200
101 + 100 = 201
```

① 从这个程序的逻辑来看，不变量是什么？

A，B账户余额总和是不变量，每次转移的金额是固定的（1），且从一个账户扣除的金额会加到另一个账户中。

② 从这个程序的执行来看，结果是否正确？

- 使用了 `&` 将 `transfer` 函数放入后台执行，这会导致多个 `transfer` 进程同时读取和写入 `A.txt` 和 `B.txt`。
- 由于文件读写没有同步机制，会导致数据不一致。

① 在自己的shell环境中执行脚本并观察输出，尝试修改并调试程序

```
1  #!/bin/bash
2  # flock.sh
3  echo "100" > A.txt
4  echo "100" > B.txt
5  # 转账函数
6  transfer() {
7  echo "$4: $1 向 $2 支付 $3."
8  exec {lockA} <>A.lock
9  exec {lockB} <>B.lock
```

```

10 # 获取文件锁
11 flock $lockA
12 flock $lockB
13 # 执行操作
14 a=$(<$1.txt)
15 b=$(<$2.txt)
16 ((a -= $3))
17 ((b += $3))
18 echo "$a" > $1.txt
19 echo "$b" > $2.txt
20 # 释放文件锁
21 flock -u $lockA
22 flock -u $lockB
23 }
24 for ((i=0; i<10; i++)); do
25 transfer A B 1 $i &
26 transfer B A 1 $i &
27 echo ":: A 余额: $(<A.txt); B 余额: $(<B.txt)"
28 done
29 wait

```

```
stu14@5aa3e8799848:~$ chmod 777 2.sh
```

```
stu14@5aa3e8799848:~$ ./2.sh
```

```

0: A 向 B 支付 1.
0: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100
1: A 向 B 支付 1.
1: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100
2: B 向 A 支付 1.
2: A 向 B 支付 1.
:: A 余额: 100; B 余额: 100
3: B 向 A 支付 1.
3: A 向 B 支付 1.
:: A 余额: 100; B 余额: 101
4: B 向 A 支付 1.
4: A 向 B 支付 1.
:: A 余额: 99; B 余额: 101
5: A 向 B 支付 1.
5: B 向 A 支付 1.
:: A 余额: 99; B 余额: 101
6: A 向 B 支付 1.
6: B 向 A 支付 1.
:: A 余额: 99; B 余额: 101
7: A 向 B 支付 1.
7: B 向 A 支付 1.
:: A 余额: 99; B 余额: 100
8: A 向 B 支付 1.
8: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100
9: A 向 B 支付 1.
9: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100

```

2 如何比较脚本执行时间？提示：time指令

第一种方法：

```
stu14@5aa3e8799848:~$ time ./1.sh
99 + 101 = 200
98 + 101 = 199
100 + 101 = 201
97 + 101 = 198
100 + 101 = 201
96 + 102 = 198
99 + 101 = 200
100 + 102 = 202
98 + 98 = 196
98 + 99 = 197
98 + 101 = 199
100 + 100 = 200
99 + 99 = 198
98 + 100 = 198

real    0m0.020s
user    0m0.052s
sys     0m0.064s
```

第二种方法：

```

stu14@5aa3e8799848:~$ time ./2.sh
0: A 向 B 支付 1.
0: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100
1: B 向 A 支付 1.
1: A 向 B 支付 1.
:: A 余额: 100; B 余额: 100
2: A 向 B 支付 1.
2: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100
3: A 向 B 支付 1.
3: B 向 A 支付 1.
:: A 余额: 100; B 余额: 100
4: B 向 A 支付 1.
4: A 向 B 支付 1.
:: A 余额: 99; B 余额: 101
5: A 向 B 支付 1.
5: B 向 A 支付 1.
:: A 余额: 99; B 余额: 101
6: B 向 A 支付 1.
6: A 向 B 支付 1.
:: A 余额: 99; B 余额: 101
7: B 向 A 支付 1.
7: A 向 B 支付 1.
:: A 余额: 100; B 余额: 100
8: B 向 A 支付 1.
8: A 向 B 支付 1.
:: A 余额: 100; B 余额: 100
9: B 向 A 支付 1.
9: A 向 B 支付 1.
:: A 余额: 100; B 余额: 100

real    0m0.193s
user    0m0.176s
sys     0m0.178s

```

3 如果你是真正的Linux大佬，你该如何进一步优化需求？

- 确定锁的执行顺序

在 `transfer` 函数中，无论转账方向如何（ $A \rightarrow B$ 或 $B \rightarrow A$ ），始终按照账户名的字母顺序获取锁。比如说A和B转账时总是先锁A再锁B就可以消除循环等待，避免死锁。

- 使用独立锁

每个账户（如A、B）使用独立的锁文件，转账时只锁定涉及的账户，减少锁的争用

实验总结

通过本次实验，我深入掌握了Linux Shell中文件系统、管道、权限、内存、网络等方面的命令，并能够熟练运用这些命令进行系统管理和操作。此外，我还学会了编写简单的Shell脚本，能够自动化完成一些系统任务。实验过程中，我不仅加深了对Linux操作系统的理解，还培养了独立思考和解决问题的能力。这些知识和技能将为我后续的学习和实验环境部署提供有力的支持。