

实验 4 程序设计

实验目的：

1. 学习 Bourne shell 的 shell 脚本的基本概念
2. 学习编写 Bourne shell 脚本的一些基本原则
3. 通过写简短的脚本，学会编写 Bourne shell 脚本程序的方法
4. 学习如何使用 LINUX 的 C 语言工具完成代码编辑，编译，运行程序
5. 学习掌握 make 工具，Makefile 文件的 make 规则
6. 学习使用系统调用编写程序

实验要求：

本实验在提交实验报告时，需要有下面内容

- 源程序及注释；
- 程序运行结果的截图；

实验内容：

1. 编写一个 shell 脚本程序，它带一个命令行参数，这个参数是一个文件。如果这个文件是一个普通文件，则打印文件所有者的名字和最后的修改日期。如果程序带有多个参数，则输出出错信息。

```
#!/bin/bash
if [ $# -gt 1 ]; then
    echo 'too many parameters'
    exit
fi
if [ -f $1 ]; then
    ls -l $1|awk '{print $3,$6,$7,$8}'
    exit
fi
echo 'no such file'
```

```

reason@For-Napkin:~/workspace/temp$ chmod a+x 1.sh
reason@For-Napkin:~/workspace/temp$ 1 a.out
1: command not found
reason@For-Napkin:~/workspace/temp$ ./1.sh a.out
reason Jun 3 18:23
reason@For-Napkin:~/workspace/temp$ ./1.sh a.out test.cpp
too many parameters
reason@For-Napkin:~/workspace/temp$ ./1.sh alsdk
no such file

```

2. 写一个脚本程序用文件名和目录名作为命令行参数，如果文件是一个普通文件并在给出的目录中，则删除该文件。若文件（第一个参数）是一个目录，则删除此目录及目录下所有的文件和子目录。

```

#!/bin/bash

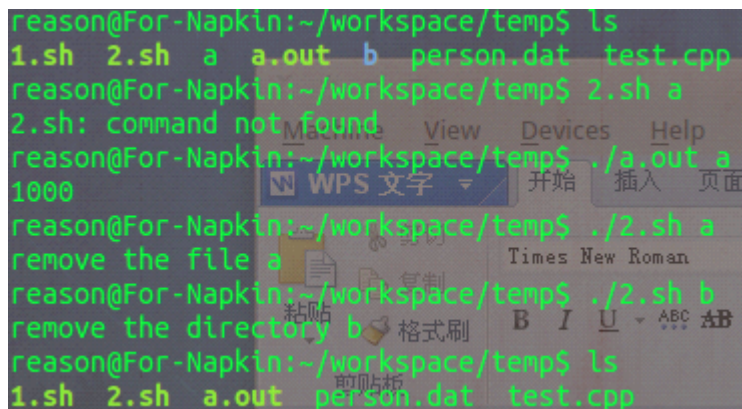
if [ $# -gt 1 ]; then
    echo 'too many parameters'
    exit
fi

if [ -d $1 ]; then
    echo "remove the directory $1"
    rm -rf $1
    exit
fi

if [ -f $1 ]; then
    echo "remove the file $1"
    rm -f $1
    exit
fi

echo 'no such file or directory'

```



```

reason@For-Napkin:~/workspace/temp$ ls
1.sh 2.sh a a.out b person.dat test.cpp
reason@For-Napkin:~/workspace/temp$ 2.sh a
2.sh: command not found
reason@For-Napkin:~/workspace/temp$ ./2.sh a
1000
reason@For-Napkin:~/workspace/temp$ ./2.sh a
remove the file a
reason@For-Napkin:~/workspace/temp$ ./2.sh b
remove the directory b
reason@For-Napkin:~/workspace/temp$ ls
1.sh 2.sh a.out person.dat test.cpp

```

3. 本实验目的学习 c 程序设计，观察进程运行。自己去查找下面源程序中的函数（或系统调用）的功能。创建一个文件名为 `test.c` 的 c 语言文件，内容如下：

```
#include <stdio.h>
main()
{
    int i;
    i = 0;
    sleep(10);
    while (i < 5) {
        system("date");
        sleep(5);
        i++;
    }
    while (1) {
        system("date");
        sleep(10);
    }
}
```

在 shell 提示符下，依次运行下列三个命令：

```
gcc -o generate test.c
./generate >> dataFile &
tail -f dataFile
```

- 第一个命令生成一个 c 语言的可执行文件，文件名为 `generate`；
- 第二个命令是每隔 5 秒和 10 秒把 `date` 命令的输出追加到 `dataFile` 文件中，这个命令为后台执行，注意后台执行的命令尾部加上 `&` 字符；
- 最后一个命令 `tail -f dataFile`，显示 `dataFile` 文件的当前内容和新追加的数据：

在输入 `tail -f` 命令 1 分钟左右后，按 `<Ctrl-C>` 终止 `tail` 程序。用 `kill -9 pid` 命令终止 `generate` 后台进程的执行。

最后用 `tail dataFile` 命令显示文件追加的内容。给出这些过程的你的会话。

注：`pid` 是执行 `generate` 程序的进程号；使用 `generate >> dataFile &` 命令后，屏幕打印后台进程作业号和进程号，其中第一个字段方括号内的数字为作业号，第二个数字为进程号；也可以用 `kill -9 %job` 终止 `generate` 后台进程，`job` 为作业号。

```
reason@For-Napkin:~/workspace/temp$ 1.sh 2.sh a.out person.dat test.c test.cpp
reason@For-Napkin:~/workspace/temp$ gcc -o generate test.c
reason@For-Napkin:~/workspace/temp$ ./generate >> dataFile &
[1] 3175
reason@For-Napkin:~/workspace/temp$ tail -f dataFile
Thu Jun 7 14:18:27 CST 2012
Thu Jun 7 14:18:32 CST 2012
Thu Jun 7 14:18:37 CST 2012
Thu Jun 7 14:18:42 CST 2012
Thu Jun 7 14:18:47 CST 2012
Thu Jun 7 14:18:52 CST 2012
Thu Jun 7 14:19:02 CST 2012
Thu Jun 7 14:19:12 CST 2012
Thu Jun 7 14:19:22 CST 2012
^C
reason@For-Napkin:~/workspace/temp$ kill -9 3175
reason@For-Napkin:~/workspace/temp$ tail dataFile
Thu Jun 7 14:18:27 CST 2012
Thu Jun 7 14:18:32 CST 2012
Thu Jun 7 14:18:37 CST 2012
Thu Jun 7 14:18:42 CST 2012
Thu Jun 7 14:18:47 CST 2012
Thu Jun 7 14:18:52 CST 2012
Thu Jun 7 14:19:02 CST 2012
Thu Jun 7 14:19:12 CST 2012
Thu Jun 7 14:19:22 CST 2012
[1]+  Killed                  ./generate >> dataFile
```

- 第二个命令是每隔 5 秒和文件中，这个命令为后台
- 最后一个命令 `tail -f` 新追加的数据：

在输入 `tail -f` 命令 1 分钟后，`kill -9 pid` 命令终止 `generate` 后

最后用 `tail dataFile` 命令显示。

注：`pid` 是执行 `generate` 程序的屏幕打印后台进程作业号和进程号，第二个数字为进程号；也可以为作业号。

4. 设计两个程序，要求用消息队列实现聊天程序，每次发言后自动在后面增加当前系统时间。增加结束字符，比如最后输入“88”后结束进程。(完成本题的有关知识请参考教材第 7 章)

```
reason@For-Napkin:~/workspace/temp$ ./Napkin
This is Napkin:
reAsOn:hello (19:2:5)
Napkin:hi
reAsOn:How are you? (19:2:36)
Napkin:fine thanks~ and you?
reAsOn:just so so (19:2:54)
Napkin:ok
reAsOn is logging off: Identifier removed
reason@For-Napkin:~/workspace/temp$

reason@For-Napkin:~/workspace/temp$ ./reAsOn
This is reAsOn:
reAsOn:hello
Napkin:hi (19:2:28)
reAsOn:How are you?
Napkin:fine thanks~ and you? (19:2:47)
reAsOn:just so so
Napkin:ok (19:2:57)
reAsOn:88
reason@For-Napkin:~/workspace/temp$
```

5. Fibonacci 序列为 0,1,1,2,3,5,8,...，通常，这可表达为:

$fib_0=0$

$fib_1=1$

$fib_n=fib_{n-1}+fib_{n-2}$

编写一个多线程程序来生成 Fibonacci 序列。程序应该这样工作:用户运行程序时在命令行输入要产生 Fibonacci 序列的数，然后程序创建一个新的线程来产生 Fibonacci 数，把这个序列放到线程共享的数据中(数组可能是一种最方便的数据结构)。当线程执行完成后，父线程将输出由子线程产生的序列。由于在子线程结束前，父线程不能开始输出 Fibonacci 序列，因此父线程必须等待子线程的结

束。(完成本题的有关知识请参考教材第8章)

```
reason@For-Napkin:~/Downloads/Pictures/55$ ls
5.c README
reason@For-Napkin:~/Downloads/Pictures/55$ gcc 5.c -lpthread
reason@For-Napkin:~/Downloads/Pictures/55$ ./a.out 10
0 1 1 2 3 5 8 13 21 34
reason@For-Napkin:~/Downloads/Pictures/55$ ./a.out 20
0 1 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
reason@For-Napkin:~/Downloads/Pictures/55$
```

下面题目为选做题，计算机学院学生或计算机学院的准学生为必做题，其他学生可以不做。

6. 在 linux 系统下的软件开发中，经常要使用 make 工具，要掌握 make 的规则。makefile 文件中的每一行是描述文件间依赖关系的 make 规则。本实验是关于 makefile 内容的，您不需要在计算机上进行编程运行，只要书面回答下面这些问题。

对于下面的 makefile:

```
CC = gcc
OPTIONS = -O3 -o
OBJECTS = main.o stack.o misc.o
SOURCES = main.c stack.c misc.c
HEADERS = main.h stack.h misc.h
power: main.c $(OBJECTS)
    $(CC) $(OPTIONS) power $(OBJECTS) -lm
main.o: main.c main.h misc.h
stack.o: stack.c stack.h misc.h
misc.o: misc.c misc.h
```

回答下列问题

- 所有宏定义的名字
- 所有目标文件的名字
- 每个目标的依赖文件
- 生成每个目标文件所需执行的命令
- 画出 makefile 对应的依赖关系树。
- 生成 main.o stack.o 和 misc.o 时会执行哪些命令，为什么？

答:

- 所有宏定义的名字

CC,OPTIONS,OBJECTS,SOURCES,HEADERS 共 5 个宏

- 所有目标文件的名字

power , main.o stack.o misc.o

c. 每个目标的依赖文件

power 依赖文件: main.o stack.o misc.o

main.o 依赖文件: main.c main.h misc.h

stack.o 依赖文件: stack.c stack.h misc.h

misc.o 依赖文件: misc.c misc.h

d. 生成每个目标文件所需执行的命令

power:

gcc -O3 -o power main.o stack.o misc.o -lm

main.o

gcc -c main.c -o main.o

stack.o

gcc -c stack.c -o stack.o

misc.o

gcc -c misc.c -o misc.o

7. 用编辑器创建 main.c, compute.c, input.c, compute.h, input.h 和 main.h 文件。

下面是它们的内容。注意 compute.h 和 input.h 文件仅包含了 compute 和 input 函数的声明但没有定义。定义部分是在 compute.c 和 input.c 文件中。

main.c 包含的是两条显示给用户的提示信息。

```
$ cat compute.h
```

```
/* compute 函数的声明原形 */
```

```
double compute(double, double);
```

```
$ cat input.h
```

```
/* input 函数的声明原形 */
```

```
double input(char *);
```

```
$ cat main.h
```

```
/* 声明用户提示 */
```

```
#define PROMPT1 "请输入 x 的值: "
```

```
#define PROMPT2 "请输入 y 的值: "
```

```
$ cat compute.c
```

```
#include <math.h>
```

```
#include <stdio.h>
```

```
#include "compute.h"
```

```

double compute(double x, double y)
{
    return (pow ((double)x, (double)y));
}

$ cat input.c
#include <stdio.h>
#include "input.h"
double input(char *s)
{
    float x;
    printf("%s", s);
    scanf("%f", &x);
    return (x);
}

$ cat main.c
#include <stdio.h>
#include "main.h"
#include "compute.h"
#include "input.h"

main()
{
    double x, y;
    printf("本程序从标准输入获取 x 和 y 的值并显示 x 的 y 次方.\n");
    x = input(PROMPT1);
    y = input(PROMPT2);
    printf("x 的 y 次方是:%6.3f\n", compute(x, y));
}

$

```

为了得到可执行文件 `power`，我们必须首先从三个源文件编译得到目标文件，并把它们连接在一起。下面的命令将完成这一任务。注意，在生成可执行代码时不要忘了连接上数学库。

```

$ gcc -c main.c input.c compute.c
$ gcc main.o input.o compute.o -o power -lm
$

```

相应的 Makefile 文件是：

```

$ cat Makefile
power: main.o input.o compute.o
    gcc main.o input.o compute.o -o power -lm

```



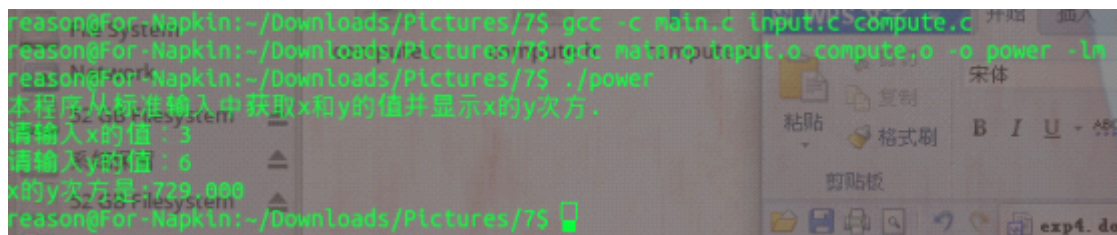
```
main.o: main.c main.h input.h compute.h
gcc -c main.c

input.o: input.c input.h
gcc -c input.c

compute.o: compute.c compute.h
gcc -c compute.c

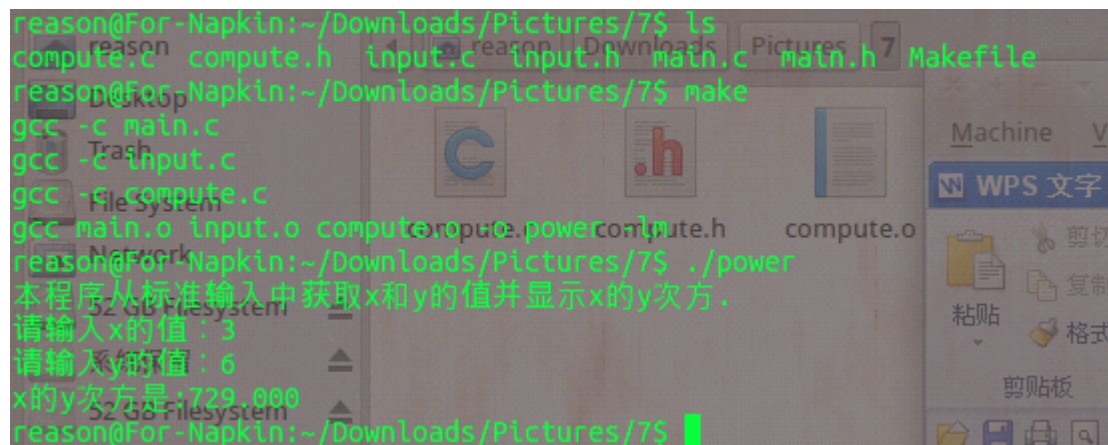
$
```

(1)、创建上述三个源文件和相应头文件，用 gcc 编译器，生成 power 可执行文件，并运行 power 程序。给出完成上述工作的步骤和程序运行结果。



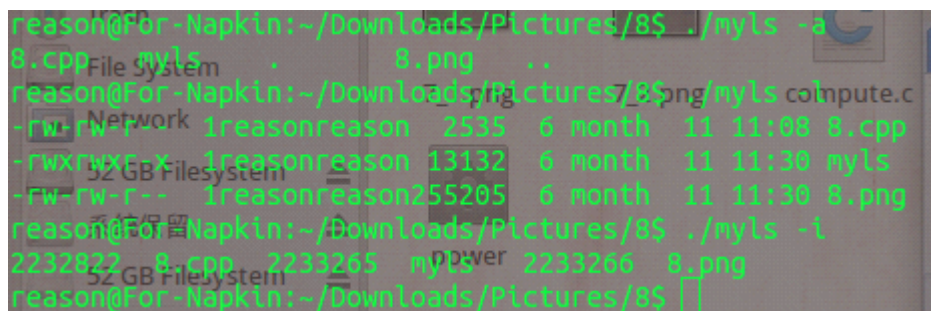
```
reason@For-Napkin:~/Downloads/Pictures/75$ gcc -c main.c input.c compute.c
reason@For-Napkin:~/Downloads/Pictures/75$ gcc main.o input.o compute.o -o power
reason@For-Napkin:~/Downloads/Pictures/75$ ./power
本程序从标准输入中获取x和y的值并显示x的y次方。
请输入x的值: 3
请输入y的值: 6
x的y次方是: 729.000
reason@For-Napkin:~/Downloads/Pictures/75$
```

(2)、创建 Makefile 文件，使用 make 命令，生成 power 可执行文件，并运行 power 程序。给出完成上述工作的步骤和程序运行结果。



```
reason@For-Napkin:~/Downloads/Pictures/75$ ls
compute.c  compute.h  input.c  input.h  main.c  main.h  Makefile
reason@For-Napkin:~/Downloads/Pictures/75$ make
gcc -c main.c
gcc -c input.c
gcc -c compute.c
gcc main.o input.o compute.o -o power
reason@For-Napkin:~/Downloads/Pictures/75$ ./power
本程序从标准输入中获取x和y的值并显示x的y次方。
请输入x的值: 3
请输入y的值: 6
x的y次方是: 729.000
reason@For-Napkin:~/Downloads/Pictures/75$
```

8. 用 C 语言写一个名字为 myls 程序，实现类似 Linux 的 ls 命令，其中 myls 命令必须实现 -a、-l、-i 等选项的功能。要求 myls 程序使用系统调用函数编写，**不能使用 system() 函数调用 ls 命令来实现**。命令 man ls 可以得到更多 ls 选项的含义。(完成本题的有关知识请参考教材第 5 章)



```
reason@For-Napkin:~/Downloads/Pictures/85$ ./mysl -a
8.cpp      .          8.png      ..
reason@For-Napkin:~/Downloads/Pictures/85$ ./mysl -l
-rw-r--r-- 1 reasonreason 2535  6 month  11 11:08 8.cpp
-rwxr-xr-x 1 reasonreason 13132 6 month  11 11:38 mysl
-rw-r--r-- 1 reasonreason 255205 6 month  11 11:30 8.png
reason@For-Napkin:~/Downloads/Pictures/85$ ./mysl -i
2232822 8.cpp 2233265 mysl 2233266 8.png
reason@For-Napkin:~/Downloads/Pictures/85$
```