实验 4　系统程序设计

实验目的：

1. 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
2. 学习掌握 make 工具，Makefile 文件的 make 规则
3. 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

- **源程序及详细的注释；**
- **程序运行结果的截图；**
- **必要的文档**

实验提示：

在编写第 4 题的程序时，可以参考“Linux 程序设计”和“UNIX 环境高级编程”等参考教材。

实验内容：

1. 在操作系统分析及实验课程中要对 linux 内核进行修改，用 make 工具，需要掌握 make 的规则。makfile 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，你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 $(OPJECTS)
   $(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

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

2. 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));
}
```

提示：若您的 linux 系统没有中文系统，可以把程序中的汉字翻译成英文。

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。
（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。

3. 本实验目的观察使用带-f 选项的 tail 命令及学习如何使用 gcc 编译器，并观察进程运行。自己去查阅资料获取下面源程序中的函数（或系统调用）的作用。首先复制 smallFile 文件（实验 1 中创建的），文件名为 dataFile；然后创建一个文件名为 lab4-3.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 lab4-3.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。

提示：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。

4. 编程开发一个 shell 程序

shell 或者命令行解释器是操作系统中最基本的用户接口。写一个简单的 shell 程序——myshell，它具有以下属性：

1．这个 shell 程序必须支持以下内部命令：

1) cd <directory> ——把当前默认目录改变为<directory>。如果没有<directory>参数，则显示当前目录。如该目录不存在，会出现合适的错误信息。这个命令也可以改变 PWD 环境变量。
2) clr ——清屏。
3) dir <directory> ——列出目录<directory>的内容。
4) environ ——列出所有的环境变量。
5) echo <comment> ——在屏幕上显示<comment>并换行（多个空格和制表符可能被缩减为一个空格）。
6) help ——显示用户手册，并且使用 more 命令过滤。
7) quit ——退出 shell。
8) shell 的环境变量应该包含 shell=<pathname>/myshell，其中<pathname>/myshell 是可执行程序 shell 的完整路径(不是你的目录下的硬连线路径，而是它执行的路径）。

2．其他的命令行输入被解释为程序调用，shell 创建并执行这个程序，并作为自己的子进程。程序的执行的环境变量包含一下条目：

parent=<pathname>/myshell。

3．shell 必须能够从文件中提取命令行输入，例如 shell 使用以下命令行被调用：

myshell batchfile

这个批处理文件应该包含一组命令集，当到达文件结尾时 shell 退出。很明显，如果 shell 被调用时没有使用参数，它会在屏幕上显示提示符请求用户输入。

4．shell 必须支持 I/O 重定向，stdin 和 stdout，或者其中之一，例如命令行为：programname arg1 arg2 < inputfile > outputfile

使用 arg1 和 arg2 执行程序 programname，输入文件流被替换为 inputfile，输出文件流被替换为 outputfile。

stdout 重定向应该支持以下内部命令：dir、environ、echo、help。

使用输出重定向时，如果重定向字符是>，则创建输出文件，如果存在则覆盖之；如果重定向字符为>>，也会创建输出文件，如果存在则添加到文件尾。

5．shell 必须支持后台程序执行。如果在命令行后添加&字符，在加载完程序后需要立刻返回命令行提示符。

6．命令行提示符必须包含当前路径。

提示：

1) 你可以假定所有命令行参数（包括重定向字符<、>、>>和后台执行字符&）和其他命令行参数用空白空间分开，空白空间可以为一个或多个空格或制表符（见上面 4 中的命令行）。

2) 程序的框架：

```
#include <stdio.h>
#include <unistd.h>
#define MAX LINE 80 /* The maximum length command */
int main(void){
char *args[MAX LINE/2 + 1]; /* command line arguments */
int should run = 1; /* flag to determine when to exit program */
while (should run) {
  printf("myshell>");
  fflush(stdout);
  /**
  * After reading user input, the steps are:
  *内部命令：
  *…..
  *外部命令：
  * (1) fork a child process using fork()
  * (2) the child process will invoke execvp()
  * (3) if command included &, parent will invoke wait()
  *…..
  */
  }
  return 0;
}
```

项目要求：

1．设计一个简单的命令行 shell，满足上面的要求并且在指定的 Linux 平台上执行。

2．写一个关于如何使用 shell 的简单的用户手册，用户手册应该包含足够的细节以方便 Linux 初学者使用。例如，你应该解释 I/O 重定向、程序环境和后台程序执行。用户手册必须命名为 readme，必须是一个标准文本编辑器可以打开的简单文本文档。

3．源码必须有很详细的注释，并且有很好的组织结构以方便别人阅读和维护。结构和注释好的程序更加易于理解，并且可以保证批改你作业的人不用很费劲地去读你的代码。

4．在截止期之前，要提供很详细的提交文档。

5．提交内容为源码文件，包括文件、makefile 和 readme 文件。批改作业者会重新编译源码，如果编译不通过将没办法打分。

6．makefile 文件必须能用 make 工具产生二进制文件 myshell，即命令提示符下键入 make 就会产生 myshell 程序。

7．根据上面提供的实例，提交的目录至少应该包含以下文件：

makefile
myshell.c
utility.c
myshell.h
readme

提交：

需要 makefile 文件，所有提交的文件将会被复制到一个目录下，所以不要在 makefile 中包含路径。makefile 中应该包含编译程序所需的依赖关系，如果包含了库文件，makefile 也会编译这个库文件的。

为了清楚起见，再重复一次：不要提交二进制或者目标代码文件。所需的只是源码、makefile 和 readme 文件。提交之前测试一下，把源码复制到一个空目录下，键入 make 命令编译。

所需的文档要求：

首先，源码和用户手册都将被评估和打分，源码需要注释，用户手册可以是你自己选择的形式（但要能被简单文本编辑器打开）。其次，手册应该包含足够的细节以方便 Linux 初学者使用，例如，你应该解释 I/O 重定向、程序环境和后台程序执行。用户手册必须命名为 readme。