

## 第一次实验:Linux shell 环境命令及 Makefile 实验

姓名: 汪值      学号: 141270037      学院: 工程管理学院(选修)

环境: Linux ubuntu16 wz@ubuntu16

### 一. 使用 Linux Shell 命令完成以下操作

1. 查看当前登录在系统中的用户列表、系统中的用户总数和系统启动时间。

```
wz@ubuntu16: ~  
wz@ubuntu16:~$ who  
wz        tty7            2017-09-25 20:50 (:0)  
wz@ubuntu16:~$ users  
wz  
wz@ubuntu16:~$ who | wc -l  
1  
wz@ubuntu16:~$ uptime  
20:57:45 up 9 min,  1 user,  load average: 0.43, 1.11, 0.82  
wz@ubuntu16:~$
```

2. 将系统文件/etc/profile 复制到主用户目录, 并改名为 profile.txt, 查看此文件的内容, 并对非空行进行编号;重新打开此文件, 从 profile 的第 5 行开始显示, 每屏幕仅显示 5 行。复制, 查看

```
wz@ubuntu16: ~  
wz@ubuntu16:~$ cp /etc/profile ./profile.txt  
wz@ubuntu16:~$ cat profile.txt  
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))  
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).  
  
if [ "$PS1" ]; then  
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then  
    # The file bash.bashrc already sets the default PS1.  
    # PS1='\h:\w\$ '
```

对非空行编号

```
wz@ubuntu16: ~  
wz@ubuntu16:~$ cat profile.txt -b  
1 # /etc/profile: system-wide .profile file for the Bourne shell (sh(1))  
2 # and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).  
  
3 if [ "$PS1" ]; then  
4   if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then  
5     # The file bash.bashrc already sets the default PS1.  
6     # PS1='\h:\w\$ '  
7     if [ -f /etc/bash.bashrc ]; then
```

5-5 查看

```
wz@ubuntu16: ~  
wz@ubuntu16:~$ more -5 +5 profile.txt  
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then  
    # The file bash.bashrc already sets the default PS1.  
    # PS1='\h:\w\$ '  
    if [ -f /etc/bash.bashrc ]; then  
-More--(42%)
```

3.在主用户目录创建临时目录 tmp，在此目录下，将/etc 目录压缩成 etc.zip 文件，然后解压缩。

创建目录，压缩文件

```
wz@ubuntu16: ~/tmp
wz@ubuntu16:~$ mkdir tmp
wz@ubuntu16:~$ cd tmp
wz@ubuntu16:~/tmp$ zip etc.zip -r /etc
zip warning: name not matched: /etc/alternatives/ghostscript-current/Resource/CIDFSubst/DroidSans
adding: etc/ (stored 0%)
adding: etc/initramfs-tools/ (stored 0%)
adding: etc/initramfs-tools/update-initramfs.conf (deflated 51%)
adding: etc/initramfs-tools/hooks/ (stored 0%)
adding: etc/initramfs-tools/scripts/ (stored 0%)
adding: etc/initramfs-tools/scripts/nfs-bottom/ (stored 0%)
adding: etc/initramfs-tools/scripts/init-top/ (stored 0%)
```

解压文件

```
wz@ubuntu16: ~/tmp
wz@ubuntu16:~/tmp$ ls
etc.zip
wz@ubuntu16:~/tmp$ unzip -q etc.zip
wz@ubuntu16:~/tmp$ ls
etc  etc.zip
wz@ubuntu16:~/tmp$
```

4.查找/etc 目录下包含字符串“ss”的文件；复制/etc/passwd 文件到用户的主目录下，搜索这个文件中包含字符串“root”的行，并显示行号。

```
wz@ubuntu16: ~/tmp
wz@ubuntu16:~/tmp$ find . |xargs grep -ri "ss" -l
```

```
wz@ubuntu16: ~
wz@ubuntu16:~$ cp /etc/passwd ./
wz@ubuntu16:~$ grep -n "root" passwd
1:root:x:0:0:root:/root:/bin/bash
wz@ubuntu16:~$
```

5.创建一个新用户 user1，给该用户设置密码为 LoveLinux，将用户名更改为 user2。创建 user3,将 user3 的有效组切换为 admin。切换到 user3，在/home 目录下创建 dir 目录。切换到 user2，查看 user2 是否可以在 dir 目录下创建、删除文件。如果不可以修改这个目录的权限，或者修改这个目录的所有者、所属组，使得用户 user2 可以在这个目录下创建、删除文件。

新建 user1，设置密码，更改名 user2

```
Terminal
wz@ubuntu16: ~
wz@ubuntu16:~$ sudo useradd user1
wz@ubuntu16:~$ sudo passwd user1
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
wz@ubuntu16:~$ sudo usermod -l user2 user1
wz@ubuntu16:~$
```

创建 user3

```
Terminal
wz@ubuntu16: ~
wz@ubuntu16:~$ sudo useradd user3
wz@ubuntu16:~$ sudo usermod -g admin user3
usermod: group 'admin' does not exist
wz@ubuntu16:~$ sudo groupadd admin
wz@ubuntu16:~$ sudo usermod -g admin user3
wz@ubuntu16:~$ |
```

创建 dir, 不可以访问

```
wz@ubuntu16: ~
wz@ubuntu16:~$ su user3
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
user3@ubuntu16:/home/wz$ cd ..
user3@ubuntu16:/home$ sudo mkdir dir
user3@ubuntu16:/home$ ls
dir wz
user3@ubuntu16:/home$ su user2
Password:
user2@ubuntu16:/home$ cd dir
user2@ubuntu16:/home/dir$ touch readme
touch: cannot touch 'readme': Permission denied
user2@ubuntu16:/home/dir$ sudo touch readme
[sudo] password for user2:
user2 is not in the sudoers file. This incident will be reported.
user2@ubuntu16:/home/dir$ |
```

修改权限 chmod 777

```
wz@ubuntu16: ~
wz@ubuntu16:~$ su user3
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
user3@ubuntu16:/home/wz$ cd ..
user3@ubuntu16:/home$ sudo mkdir dir
user3@ubuntu16:/home$ ls
dir wz
user3@ubuntu16:/home$ su user2
Password:
user2@ubuntu16:/home$ cd dir
user2@ubuntu16:/home/dir$ touch readme
touch: cannot touch 'readme': Permission denied
user2@ubuntu16:/home/dir$ sudo touch readme
[sudo] password for user2:
user2 is not in the sudoers file. This incident will be reported.
user2@ubuntu16:/home/dir$ su user3
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
user3@ubuntu16:/home/dir$ sudo chmod 777 ../dir/
user3@ubuntu16:/home/dir$ su user2
Password:
user2@ubuntu16:/home/dir$ touch readme
user2@ubuntu16:/home/dir$ echo "this is lab01" >readme
user2@ubuntu16:/home/dir$ cat readme
this is lab01
user2@ubuntu16:/home/dir$ |
```

6.完全使用命令下载、 安装、运行并卸载 Linux 版本的 QQ。

已经不支持 Linux 版本 QQ

7.查看网络适配器的网络设置, 将 dhcp 动态 IP 的设置方式改为 static 静态 IP 的设置方式; 查看当前系统服务端口的监听状态。

查看网络配置为动态 IP

```
wz@ubuntu16:~$ sudo cat /etc/network/interfaces
[sudo] password for wz:
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
wz@ubuntu16:~$ |
```

编辑/etc/network/interfaces 文件, 添加如下内容:

```
wz@ubuntu16:~$ sudo cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
# iface lo inet loopback

iface eth0 inet static
address 192.168.3.101
netmask 255.255.255.0
gateway 192.168.3.1
broadcast 192.168.3.255
wz@ubuntu16:~$
```

编辑/etc/resolv.conf, 设置相应的 DNS:

```
wz@ubuntu16: ~
1 # Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
2 #     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
3 # nameserver 127.0.1.1
4 # search localdomain
5
6 nameserver 8.8.8.8
7 nameserver 8.8.4.4
```

重启网卡:

```
wz@ubuntu16: ~
wz@ubuntu16:~$ sudo /etc/init.d/networking restart
[ ok ] Restarting networking (via systemctl): networking.service.
wz@ubuntu16:~$
```

8.插入 u 盘, 在/mnt 下建立一个名叫 USB 的文件夹,然后将 u 盘挂载到/mnt/USB 下, 在此目录下创建一个 temp.txt 文件, 然后卸载 u 盘。

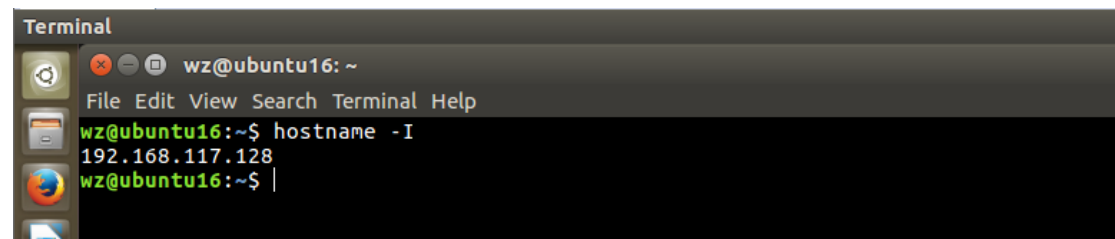
```
Terminal
root@ubuntu16: /mnt/usb
root@ubuntu16:/mnt# mkdir usb
root@ubuntu16:/mnt# ls
hgfs  usb
root@ubuntu16:/mnt# mount -t vfat /dev/sdb /mnt/usb/
sdb  sdb4
root@ubuntu16:/mnt# mount -t vfat /dev/sdb4 /mnt/usb/
root@ubuntu16:/mnt# cd usb
root@ubuntu16:/mnt/usb# ls
141270037 System Volume Information 打印材料.zip
NJUCS初试--kx 打印材料 算法
root@ubuntu16:/mnt/usb# cd ..
root@ubuntu16:/mnt# umount /dev/sdb4
umount: /dev/sdb4: mountpoint not found
root@ubuntu16:/mnt# umount /dev/sdb4
root@ubuntu16:/mnt# cd usb
root@ubuntu16:/mnt/usb# ls
root@ubuntu16:/mnt/usb# |
```

## 二、实现从 Windows 到所用 Linux 系统的远程连接

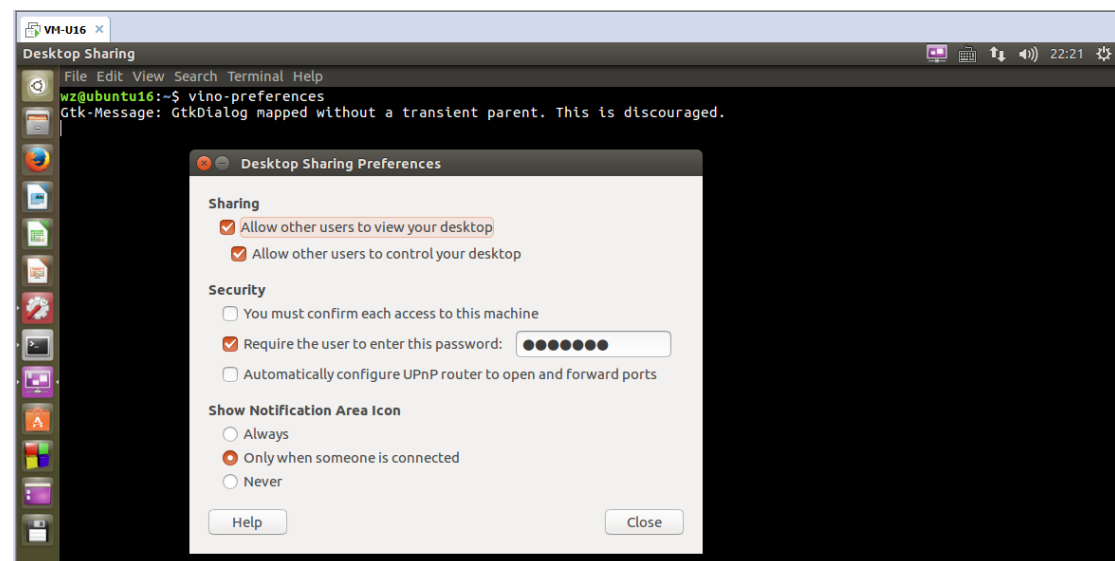
参考用 win7（64 位）远程桌面连接 linux（Ubuntu14.04）详细教程

<http://blog.csdn.net/qq754438390/article/details/50042511>

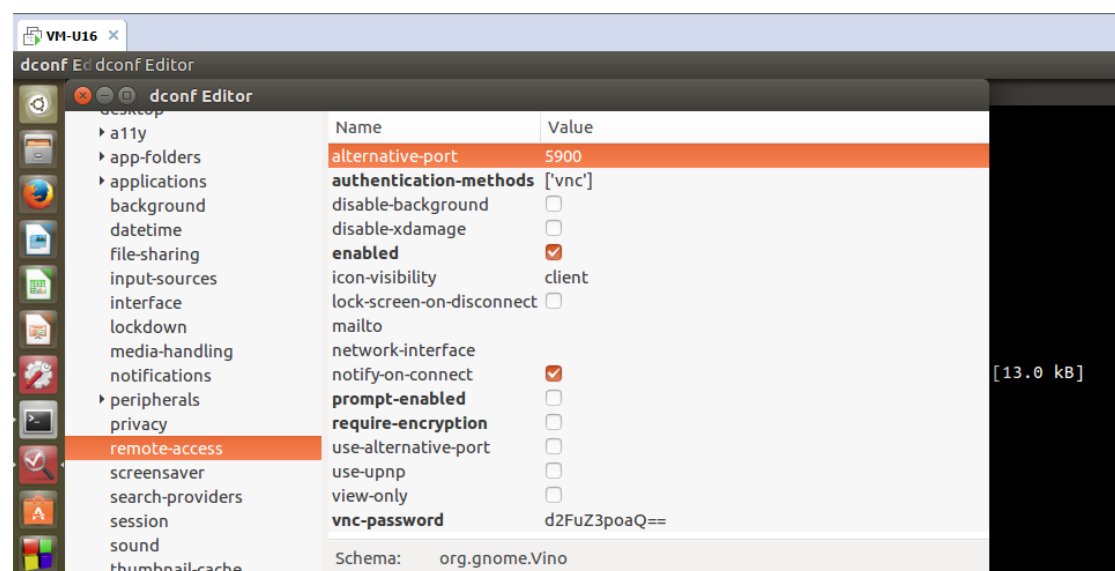
查看 Ubuntu IP 地址



打开桌面共享设置，选择相关选项设置

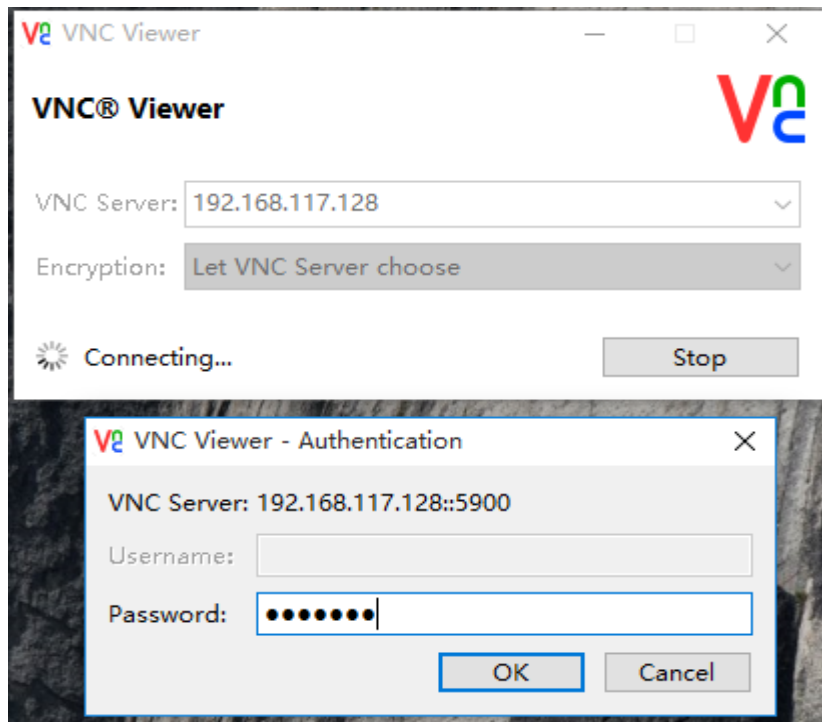


安装 dconf-editor 进行远程代理设置:

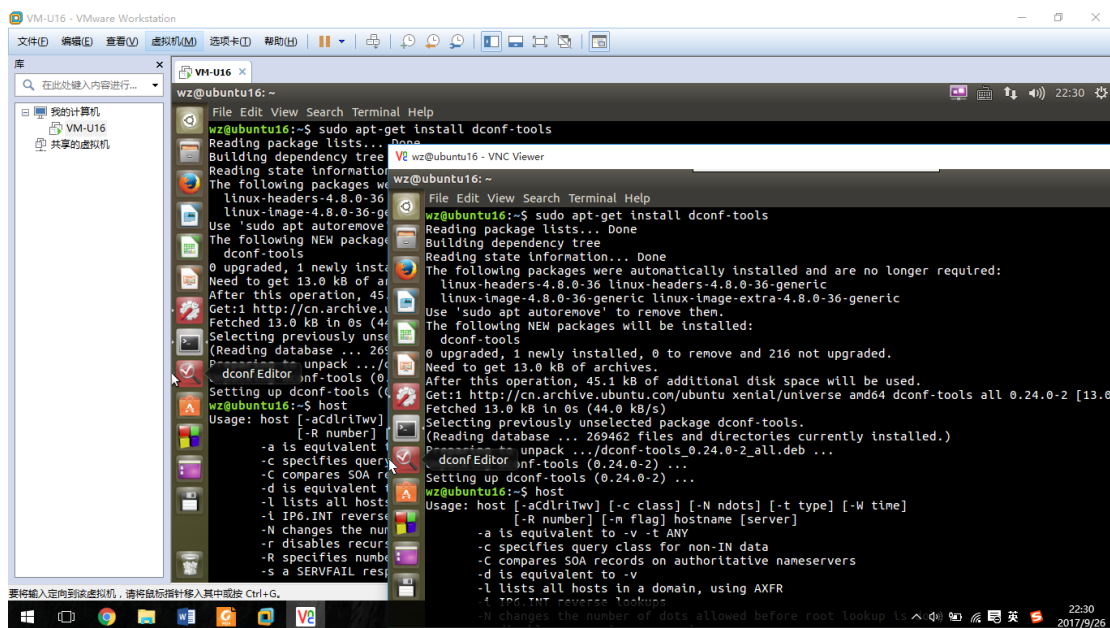


在 Windows 中打开 VNC-viewer，输入记下的 IP 地址 192.168.117.128

点击 connect 并且输入在 Desktop sharing 中设置的连接密码即可



已经成功进行远程连接 Linux



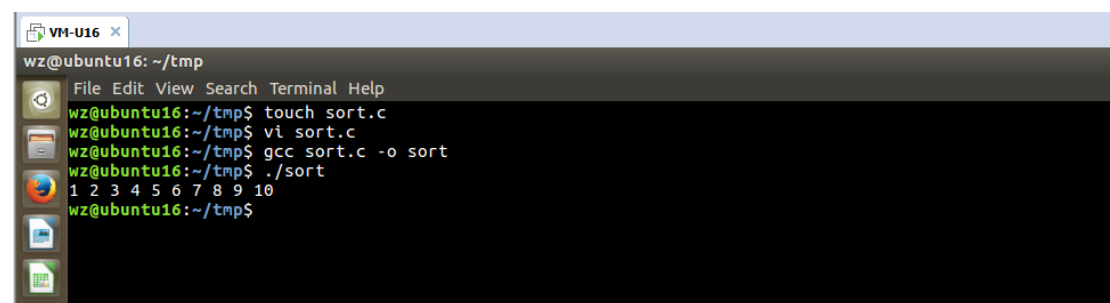
### 三、Makefile 实验

2. (1) 利用文本编辑器 ( vi ) 编写一种排序算法 sort.c, 对一个数组中的整数进行排序;



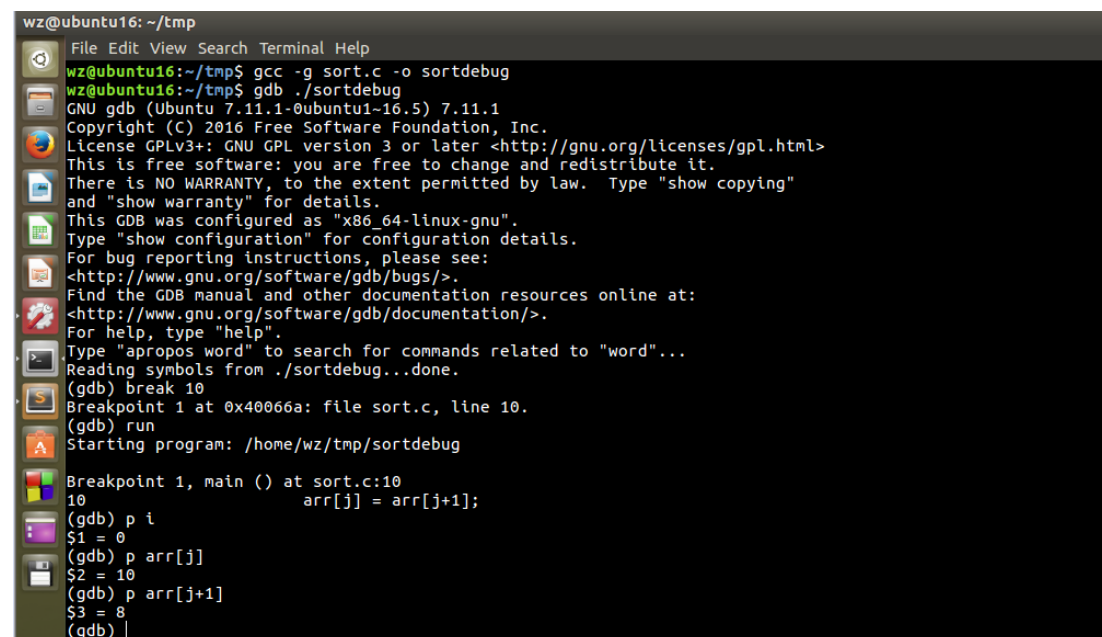
```
1 #include<stdio.h>
2 #define COUNT 10
3 int main(){
4     int arr[COUNT]={10,8,9,3,2,4,7,6,5,1};
5
6     for (int i = 0; i <COUNT; ++i) {
7         for(int j=0; j<COUNT-i-1; j++){
8             if(arr[j] > arr[j+1]){
9                 int tmp = arr[j];
10                arr[j] = arr[j+1];
11                arr[j+1] = tmp; }
12         }
13     }
14     for(int i=0;i<COUNT;i++){
15         printf("%d ",arr[i]);
16     }
17     printf("\n");
18     return 0;
19 }
```

(2) 利用 gcc 手动编译、运行该程序;



```
wz@ubuntu16: ~/tmp
wz@ubuntu16:~/tmp$ touch sort.c
wz@ubuntu16:~/tmp$ vi sort.c
wz@ubuntu16:~/tmp$ gcc sort.c -o sort
wz@ubuntu16:~/tmp$ ./sort
1 2 3 4 5 6 7 8 9 10
wz@ubuntu16:~/tmp$
```

(3) 利用 gdb 手动加入断点进行调试, 在屏幕上打印断点信息, 以及任何一个变量的值

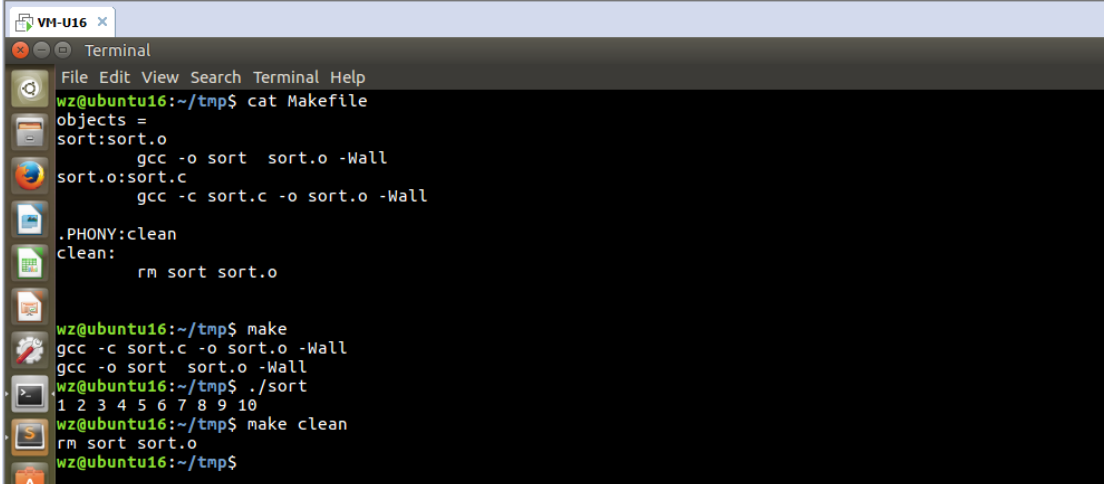


```
wz@ubuntu16: ~/tmp
wz@ubuntu16:~/tmp$ gcc -g sort.c -o sortdebug
wz@ubuntu16:~/tmp$ gdb ./sortdebug
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sortdebug...done.
(gdb) break 10
Breakpoint 1 at 0x40066a: file sort.c, line 10.
(gdb) run
Starting program: /home/wz/tmp/sortdebug

Breakpoint 1, main () at sort.c:10
10         arr[j] = arr[j+1];
(gdb) p i
$1 = 0
(gdb) p arr[j]
$2 = 10
(gdb) p arr[j+1]
$3 = 8
(gdb) |
```



3. 针对 sort.c 利用文本编辑器创建一个 makefile 文件, 通过 make 编译次程序, 并运行。



```
VM-U16 x
Terminal
File Edit View Search Terminal Help
wz@ubuntu16:~/tmp$ cat Makefile
objects =
sort:sort.o
    gcc -o sort sort.o -Wall
sort.o:sort.c
    gcc -c sort.c -o sort.o -Wall
.PHONY:clean
clean:
    rm sort sort.o

wz@ubuntu16:~/tmp$ make
gcc -c sort.c -o sort.o -Wall
gcc -o sort sort.o -Wall
wz@ubuntu16:~/tmp$ ./sort
1 2 3 4 5 6 7 8 9 10
wz@ubuntu16:~/tmp$ make clean
rm sort sort.o
wz@ubuntu16:~/tmp$
```

4. (1) 修改 sort.c, 在排序完成后创建一个进程 ;

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

#define COUNT 10
int main(){
    int arr[COUNT]={10,8,9,3,2,4,7,6,5,1};

    for(int i = 0; i <COUNT; ++i)    {
        for(int j=0; j<COUNT-i-1; j++){
            if(arr[j] > arr[j+1]){
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp; }
        }
    }

    for(int i=0;i<COUNT;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");

    int pid = fork();
    if( pid == 0 ) { // child
        printf("\n 创建进程, pid=%d.\n", getpid());
    }
    else if( pid < 0 ) { // error
```



```

        printf("创建进程失败\n");
    }

    printf("排序结束\n");
    return 0;
}

```

```

wz@ubuntu16: ~/tmp
File Edit View Search Terminal Help
wz@ubuntu16:~/tmp$ make
gcc -c sort.c -o sort.o -Wall
gcc -o sort sort.o -Wall
wz@ubuntu16:~/tmp$ ./sort
1 2 3 4 5 6 7 8 9 10
排序结束
wz@ubuntu16:~/tmp$
创建进程, pid=6916.
排序结束
wz@ubuntu16:~/tmp$ |

```

(2) 创建完成后父进程打印有序队列的首地址，然后休眠 5 秒钟；

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int insert(int i, int queue[], int n);
int queue[10] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

#define COUNT 10

int main(){
    int arr[COUNT]={10,8,9,3,2,4,7,6,5,1};

    for(int i = 0; i <COUNT; ++i)    {
        for(int j=0; j<COUNT-i-1; j++){
            if(arr[j] > arr[j+1]){
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp; }
        }
    }

    for(int i=0;i<COUNT;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
}

```

```

int pid = fork();
if( pid == 0 ) { // child
    printf("\n 创建子进程, pid=%d.\n", getpid());

    printf("开始休眠\n");
    sleep(5);
    printf("休眠结束\n");
}
else if( pid > 0 ) { // error
    int pw = wait(NULL);
    printf("捕捉到子进程, pid=%d\n", pw);
    printf("queue 首地址: %p\n", &queue);
}
else{
    printf("fork 进程失败!");
}

printf("程序末尾...\n\n");
return 0;
}

```

(3) 子进程调用一个在 insert.c 中实现的插入函数，在有序队列中插入一个整数，然后打印队列的首地址。

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

#define COUNT 10

int insert(int i, int queue[], int n);

void sort(int arr[], int n);

int queue[COUNT] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

int main(){
    int arr[COUNT]={10,8,9,3,2,4,7,6,5,1};

    for(int i = 0; i <COUNT; ++i)    {
        for(int j=0; j<COUNT-i-1; j++){

```

```

        if(arr[j] > arr[j+1]){
            int tmp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = tmp; }

    }

}

for(int i=0;i<COUNT;i++){
    printf("%d ",arr[i]);
}
printf("\n");

int pid = fork();
if( pid == 0 ) { // child
    printf("\n 创建子进程, pid=%d.\n", getpid());
    // printf("开始休眠\n");
    // sleep(5);
    // printf("休眠结束\n");
    insert(10, queue, COUNT);
    printf("queue 首地址: %p\n", &queue);
}
else if( pid > 0 ) { // error
    int pw = wait(NULL);
    printf("捕捉到子进程, pid=%d\n", pw);
    // printf("queue 首地址: %p\n", &queue);
}
else{
    printf("fork 进程失败!");
}

printf("程序末尾...\n\n");
return 0;
}

```

(4) 针对 sort.c 和 insert.c 利用文本编辑器创建一个 makefile 文件, 通过 make 编译此程序, 并运行。

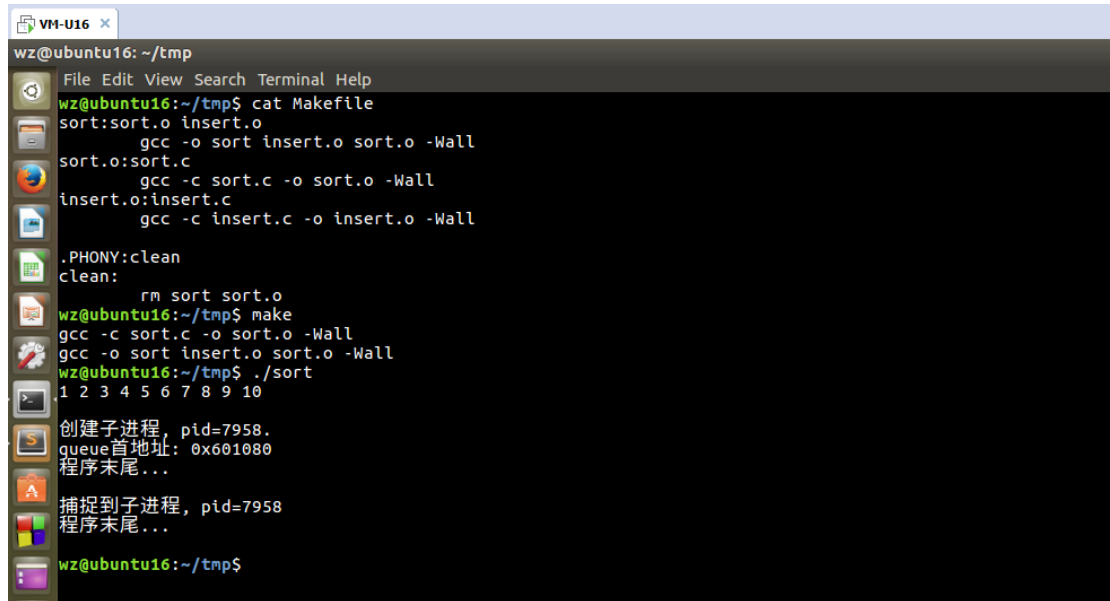
```

sort:sort.o insert.o
    gcc -o sort insert.o sort.o -Wall
sort.o:sort.c
    gcc -c sort.c -o sort.o -Wall
insert.o:insert.c
    gcc -c insert.c -o insert.o -Wall

.PHONY:clean
clean:

```

```
rm sort sort.o
```



```
wz@ubuntu16: ~/tmp
File Edit View Search Terminal Help
wz@ubuntu16:~/tmp$ cat Makefile
sort:sort.o insert.o
    gcc -o sort insert.o sort.o -Wall
sort.o:sort.c
    gcc -c sort.c -o sort.o -Wall
insert.o:insert.c
    gcc -c insert.c -o insert.o -Wall
.PHONY:clean
clean:
    rm sort sort.o
wz@ubuntu16:~/tmp$ make
gcc -c sort.c -o sort.o -Wall
gcc -o sort insert.o sort.o -Wall
wz@ubuntu16:~/tmp$ ./sort
1 2 3 4 5 6 7 8 9 10
创建子进程, pid=7958.
queue首地址: 0x601080
程序末尾...
捕捉到子进程, pid=7958
程序末尾...
wz@ubuntu16:~/tmp$
```

(5) 分析运行结果，写出你的发现。

fork () 会创建子进程，父进程和子进程的 PID 不一样，其余的都一样，fork () 函数返回的值如果是 0，就是子进程，如果非 0 就是父进程。

并且一旦创建了子进程，父子进程之间就没有什么关系了，在时间上是相互独立运行的。

5. 阅读 Linux 源码中的/Documentation/kbuild/makefiles.txt 文件 (网上有中文版)，并根据此文档分析并注释/kernel 目录下的 Makefile 文件。

```
#
# Makefile for the linux kernel.
#

#obj-y 是内核编译的最终目标，这个 MakeFile 文件是 kernel 的最开始的 MakeFile 文件，
#所以 kernel 下面所有文件夹的模块都编译到这个目标里。
obj-y      = fork.o exec_domain.o panic.o \
             cpu.o exit.o softirq.o resource.o \
             sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
             signal.o sys.o umh.o workqueue.o pid.o task_work.o \
             extable.o params.o \
             kthread.o sys_ni.o nsproxy.o \
             notifier.o ksysfs.o cred.o reboot.o \
             async.o range.o smpboot.o ucount.o

obj-$(CONFIG_MODULES) += kmod.o
obj-$(CONFIG_MULTIUSER) += groups.o
```

```

ifdef CONFIG_FUNCTION_TRACER
# Do not trace internal ftrace files
CFLAGS_REMOVE_irq_work.o = $(CC_FLAGS_FTRACE)
endif

# Prevents flicker of uninteresting __do_softirq()/__local_bh_disable_ip()
# in coverage traces.
KCOV_INSTRUMENT_softirq.o := n
# These are called from save_stack_trace() on slub debug path,
# and produce insane amounts of uninteresting coverage.
KCOV_INSTRUMENT_module.o := n
KCOV_INSTRUMENT_extable.o := n
# Don't self-instrument.
KCOV_INSTRUMENT_kcov.o := n
KASAN_SANITIZE_kcov.o := n

# cond_syscall is currently not LTO compatible
CFLAGS_sys_ni.o = $(DISABLE_LTO)

#添加变量的字符串值，以上的.o 文件包含了 kernel 的模块如：进程调度，创建进程等。
obj-y += sched/
obj-y += locking/
obj-y += power/
obj-y += printk/
obj-y += irq/
obj-y += rcu/
obj-y += livepatch/

obj-$(CONFIG_CHECKPOINT_RESTORE) += kcmp.o
obj-$(CONFIG_FREEZER) += freezer.o
obj-$(CONFIG_PROFILING) += profile.o
obj-$(CONFIG_STACKTRACE) += stacktrace.o
obj-y += time/
obj-$(CONFIG_FUTEX) += futex.o
ifeq ($(CONFIG_COMPAT),y)
obj-$(CONFIG_FUTEX) += futex_compat.o
endif
obj-$(CONFIG_GENERIC_ISA_DMA) += dma.o
obj-$(CONFIG_SMP) += smp.o
ifneq ($(CONFIG_SMP),y)
obj-y += up.o
endif
obj-$(CONFIG_UID16) += uid16.o

```

```
obj-$(CONFIG_MODULES) += module.o
obj-$(CONFIG_MODULE_SIG) += module_signing.o
obj-$(CONFIG_KALLSYMS) += kallsyms.o
obj-$(CONFIG_BSD_PROCESS_ACCT) += acct.o
obj-$(CONFIG_CRASH_CORE) += crash_core.o
obj-$(CONFIG_KEXEC_CORE) += kexec_core.o
obj-$(CONFIG_KEXEC) += kexec.o
obj-$(CONFIG_KEXEC_FILE) += kexec_file.o
obj-$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o
obj-$(CONFIG_COMPAT) += compat.o
obj-$(CONFIG_CGROUPS) += cgroup/
obj-$(CONFIG_UTS_NS) += utsname.o
obj-$(CONFIG_USER_NS) += user_namespace.o
obj-$(CONFIG_PID_NS) += pid_namespace.o
obj-$(CONFIG_IKCONFIG) += configs.o
obj-$(CONFIG_SMP) += stop_machine.o
obj-$(CONFIG_KPROBES_SANITY_TEST) += test_kprobes.o
obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
obj-$(CONFIG_AUDITSYSCALL) += auditsc.o
obj-$(CONFIG_AUDIT_WATCH) += audit_watch.o audit_fsnotify.o
obj-$(CONFIG_AUDIT_TREE) += audit_tree.o
obj-$(CONFIG_GCOV_KERNEL) += gcov/
obj-$(CONFIG_KCOV) += kcov.o
obj-$(CONFIG_KPROBES) += kprobes.o
obj-$(CONFIG_KGDB) += debug/
obj-$(CONFIG_DETECT_HUNG_TASK) += hung_task.o
obj-$(CONFIG_LOCKUP_DETECTOR) += watchdog.o
obj-$(CONFIG_HARDLOCKUP_DETECTOR_PERF) += watchdog_hld.o
obj-$(CONFIG_SECCOMP) += seccomp.o
obj-$(CONFIG_RELAY) += relay.o
obj-$(CONFIG_SYSCTL) += utsname_sysctl.o
obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
obj-$(CONFIG_TRACEPOINTS) += tracepoint.o
obj-$(CONFIG_LATENCYTOP) += latencytop.o
obj-$(CONFIG_ELF_CORE) += elfcore.o
obj-$(CONFIG_FUNCTION_TRACER) += trace/
obj-$(CONFIG_TRACING) += trace/
obj-$(CONFIG_TRACE_CLOCK) += trace/
obj-$(CONFIG_RING_BUFFER) += trace/
obj-$(CONFIG_TRACEPOINTS) += trace/
obj-$(CONFIG_IRQ_WORK) += irq_work.o
obj-$(CONFIG_CPU_PM) += cpu_pm.o
obj-$(CONFIG_BPF) += bpf/
```

```
obj-$(CONFIG_PERF_EVENTS) += events/

obj-$(CONFIG_USER_RETURN_NOTIFIER) += user-return-notifier.o
obj-$(CONFIG_PADATA) += padata.o
obj-$(CONFIG_CRASH_DUMP) += crash_dump.o
obj-$(CONFIG_JUMP_LABEL) += jump_label.o
obj-$(CONFIG_CONTEXT_TRACKING) += context_tracking.o
obj-$(CONFIG_TORTURE_TEST) += torture.o

obj-$(CONFIG_HAS_IOMEM) += memremap.o

$(obj)/configs.o: $(obj)/config_data.h

targets += config_data.gz
$(obj)/config_data.gz: $(KCONFIG_CONFIG) FORCE
    $(call if_changed,gzip)#调用函数 解压缩的

    filechk_ikconfiggz = (echo "static const char kernel_config_data[] __used =
MAGIC_START"; cat $< | scripts/basic/bin2c; echo "MAGIC_END;")
    #显示目标集的内容, 输出到 scripts 文件夹里 bin2c 文件夹里的文件
#重定向输出到内容
targets += config_data.h
$(obj)/config_data.h: $(obj)/config_data.gz FORCE
    $(call filechk,ikconfiggz)
```