

# Interview

面试题目与分析

屈庆磊

[quqinglei@icloud.com](mailto:quqinglei@icloud.com)

2013 年 5 月

## 目录

<b>1 Random questions</b>	<b>6</b>
1.1 编写一段代码，确定一个变量是有符号数还是无符号数	6
1.2 库函数和系统调用有什么区别	6
1.3 数组和指针的区别	6
<b>2 Stackoverflow</b>	<b>6</b>
2.1 File pointer VS File descriptor	6
2.2 What is a bus error?	7
<b>3 华为面试题</b>	<b>7</b>
3.1 写出判断 ABCD 四个表达式的是否正确，若正确，写出表达式中 a 的值（3 分）	7
3.2 某 32 位系统下，计算机 sizeof 的值	7
3.2.1 关于 sizeof 的思考	8
3.2.2 ♠ 关于 malloc 的思考	8
3.3 编写 strcat 函数	8
3.3.1 为什么 strcat 使用 char * 类型的返回值?	8
3.4 使用线程如何防止出现大量的波峰	8
3.5 函数模板和类模板的区别	9
3.6 一般数据库的日志满了，会出现什么情况，是否还能使用	9
<b>4 完美世界面试，职位：脚本开发工程师</b>	<b>9</b>
4.1 基础题	9
4.1.1 如何查看当前系统的配置，包括内存、CPU、硬盘、内核版本、操作系统维修保养或装饰内容	9
4.1.2 Linux 下如何更新主机名、IP 地址、DNS	9
4.1.3 如何查看、添加路由	9
4.1.4 某文件的组外权限为只读，所有者为全部权限，组内成员的权限为读与写，则文件的权限为	10
4.1.5 分别描述一下 \${}, \$?, \$*, \$\$ 在 bash 中的用途	10
4.1.6 将一个程序运行的结果输出到/tmp/finish.txt，错误信息追加到/tmp/error.txt	10
4.1.7 查找/home/system/ 路径内创建时间大于 30 天、属主为 root 用户、文件前缀为 sys 的 socket 类型文件，不包含子目录	10
4.1.8 有一个 tar 包文件 files.tar，目前需要将 files.tar 中的 test1.txt 解压到/tmp/将/home/system/sys.log 追加到 files.tar，需要怎么做	10
4.1.9 在对 Linux 系统分区进行格式化的时候需要对磁盘簇（或 i 节点密度）的大小进行选择，请说明选择的原则。	10

4.1.10	添加 itables 的策略, a 仅允许 192.168.253.30 通过 eth0 访问本机 80 端口其他连接全部拒绝; b 本机对外访问全部开放 . . . . .	11
4.1.11	如何将本地 80 端口的请求转发到 8080 端口, 当前主机为 192.168.16.1, 本地网卡为 eth0 . . . . .	11
4.1.12	如何使用 mysqldump 命令将表结构导出至/tmp/mysql.sql 文件, 不需要数据 . . .	11
4.1.13	如果 mysql 数据库中的 webdata 表索引损坏如何修复 . . . . .	11
4.1.14	在文件 a.txt 中查找以当前日期开头、中间包含一个 IP 地址、数字结尾的行, 写出查找过程 . . . . .	11
4.1.15	在 Linux 中将一个以空格分割的 CSV 文件导入到 mysql 数据库中, 并获取数据状态比如: 成功、失败条数 . . . . .	11
4.1.16	在 Linux 环境中如何查看远程 Linux 系统运行了多长时间 . . . . .	11
4.1.17	在 crontab 中设置/bin/test.sh 每周二 10 点运行一次, 写出 crontab 里的内容 . .	12
4.1.18	如何将/tmp/目录的所有.log 文件名更改为.txt 文件名 . . . . .	12
4.1.19	使用脚本语言, 如何将字符串 abcde 以空格为分割保存到数组 array 里 . . . . .	12
4.1.20	使用脚本语言将一个数组的元素按照数字从小到大的顺序输出 . . . . .	12
4.1.21	脚本环境里 (python) 如何打开、读取、关闭一个文件 . . . . .	12
4.2	应用题 . . . . .	12
4.2.1	用 shell 编程, 判断一个文件是不是字符设备, 如果是将其拷贝到/dev/路径 . . .	12
4.2.2	设计一个脚本, 添加新组为 class1, 然后添加属于这个组的用户, 形式为 stdxx 其中 xx 为 01 到 30 . . . . .	12
4.2.3	使用 bash 写一个磁盘清理、报警脚本, 具体要求如下 . . . . .	13
4.2.4	有如下两个文件, 需要对数据做累加处理, 规则如下 . . . . .	13
4.2.5	处理以下文件内容, 将域名取出并进行排序 . . . . .	13
4.2.6	有如下文件, 第一列为服务器名, 第二列为服务器 IP, 写程序实现读取各个服务器根分区磁盘占有量, 输出服务器名与占有量, 文件内容如下所示 . . . . .	14
5	websense 面试, 职位: C 工程师 . . . . .	14
5.1	Part 1 Basic Skills . . . . .	14
5.1.1	Explain the differences between a static library and a dynamic library in Linux System . . . . .	14
5.1.2	What are the Linux inter-process communication (IPC) mechanisms? How can a parent process communicate with its child process(es) . . . . .	14
5.2	Part 2 Networking . . . . .	16
5.2.1	Describe UDP and TCP, and the differences between them . . . . .	16
5.2.2	What's the differences between a Hub, a Switch and a Route . . . . .	16
5.2.3	What's the differences between ACTIVE FTP and PASSIVE FTP . . . . .	16
5.2.4	How many available IP address in subnet 210.27.48.21/30 . . . . .	16
5.3	Part 3 C/C++ Programming . . . . .	16

5.3.1	Please implement the function that reverses a single linked list . . . . .	16
5.3.2	What would be the output of the following C program . . . . .	16
5.3.3	Predict the output of the following program code . . . . .	17
5.3.4	Write the C program language expression using variable "a" according to state- ment blew . . . . .	17
<b>6</b>	<b>某语音驾驶公司的面试题目</b>	<b>18</b>
6.1	用预处理指令声明一个常数，表示一年之内的秒数（忽略闰年问题） . . . . .	18
6.2	写一个标准的 MIN 宏，这个宏输入两个参数，并返回较小的那个 . . . . .	18
6.3	嵌入式系统中用到无限循环，怎么用 C 写死循环 . . . . .	18
6.4	用变量 a 为给出下面定义 . . . . .	18
6.5	关键字 static 的作用 . . . . .	19
6.6	关键字 const 的作用 . . . . .	19
6.6.1	如果上个题目回答正确，引出下一个题目 . . . . .	19
<b>7</b>	<b>下面函数的输出情况</b>	<b>19</b>
7.1	volatile . . . . .	20
7.1.1	加一个问题 . . . . .	20
7.2	嵌入式系统总是要用户对变量或寄存器进行位操作。给定一个整型变量 a，写两段代码， 第一个设置 a 的 bit 3，第二个清除 a 的 bit 3. 在以上操作中，要保持其他位不变。 . .	21
7.3	嵌入式系统经常需要访问某特定的内存位置。在某工程中，需要设置绝对位置为 0x67a9 的整型变量的值为 0xaa66，写代码去完成此任务。 . . . . .	21
7.4	下面函数的结果预测 . . . . .	21
7.5	写出下面代码的输出结果 . . . . .	22
7.6	关于内存的几个经典面试题目 . . . . .	22
7.7	关于 free() . . . . .	25
<b>8</b>	<b>以下问题来自网络</b>	<b>25</b>
8.1	typedef 在 C 代码中使用的次数很多，也可以用预处理去做类似的事情，思考一下哪个 比较好 . . . . .	25
8.2	如何引用一个已经定义的全局变量 . . . . .	26
8.3	全局变量可不可以定义在可被多个 C 文件所包含的头文件中，为什么 . . . . .	26
8.4	for(;;) 有什么问题？它是什么意思 . . . . .	26
8.5	请写出下面代码的输出结果 . . . . .	26
8.6	static 全局变量和普通全局变量有什么区别，static 局部变量和普通局部变量有什么区 别，static 函数和普通函数有什么区别 . . . . .	26
8.7	程序的存储 . . . . .	26

目 录	5
9 微软的面试，来自网络	27
9.1 求下面函数的返回值 . . . . .	27

## 1 Random questions

### 1.1 编写一段代码，确定一个变量是有符号数还是无符号数


```
#define ISUNSIGNED(a) (a >= 0) && (~a >= 0)
```

### 1.2 库函数和系统调用有什么区别

两码事，库函数里面可能包含系统调用，但系统调用里面肯定没有库函数，库函数的实现是基于用户的，系统调用是基于内核的。系统调用是为了方便应用使用的操作系统接口，而库函数是为了方便人们编写应用程序而引入的。这个就不详细说了。

### 1.3 数组和指针的区别

简单的说：一个数组是一个地址，一个指针是一个地址的地址。

- (A) 数组和指针都可以在初始化时赋予字符串常量，看上去一样，但实际上不一样，底层机制是不同的，指针在定义的时候，编译器并不会为指针所指向的对象分配内存空间，它只是分配指针变量的空间。除非以一个字符串常量堆其进行初始化，下面的定义创建了一个字符串常量，为其分配了内存空间。`char *p = "hello";`
- (B) 内容和复制比较，不能对数组进行字节复制和比较，对于两个数组，不能用 `a = b` 进行复制，应当使用标准的 `strcpy()` 函数，也不能使用如 `if (b == a)` 进行比较，应当使用 `strcmp()` 去比较。而对于指针，你必须得先申请一块空间，才能复制。
- (C) 计算内存容量，用运算符 `sizeof()` 可以计算出数组的容量，注意当数组名作为函数参数进行传递时，该数组  自动退化为该类型的指针。

## 2 Stackoverflow

### 2.1 File pointer VS File descriptor

File pointer

- (A) It is high level interface.
- (B) Passed to `fread()` and `fwrite()` functions.
- (C) Includes buffering, error indication and EOF dection, etc.
- (D) Provides higher portability and effeciency

File descriptor

- (A) Low/kernel level handler.
- (B) Pass to `read()` and `write()` of UNIX System Calls.
- (C) Doesn't include buffering and such features.
- (D) Less portable lacks effeciency.

## 2.2 What is a bus error?

Bus error are rare nowadays on x86 and occur when processor cannot even attempt the memory access request, typically:

- using a pointer to something that was deallocated.
- using an uninitialized hence bogus pointer.
- using a null pointer.
- overflowing a buffer.

## 3 华为面试题目

### 3.1 写出判断 ABCD 四个表达式的是否正确，若正确，写出表达式中 a 的值（3 分）

```
int a = 4;
```

(A)  $a+ = (a++)$ ;

(B)  $a+ = (++a)$ ;

(C)  $(a++)+ = a$ ;

(D)  $(++a)+ = (a++)$ ;

解答：C, D 错误，左值不能是表达式。笔试题的答案是错误的，结果为： $A = 9, B = 10$

### 3.2 某 32 位系统下，计算机 sizeof 的值

// 更改版本，直接可以编译通过

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

```
#include <stdlib.h>
```

```
void Foo(char str[100])
```

```
{
```

```
    printf("D = [%d], ", sizeof(str)); /* D is a pointer too */
```

```
}
```

```
int main()
```

```
{
```

```
    int n = 10;
```

```
    char str[] = "http://www.ibegroup.com/";
```

```
    printf("A = [%d], ", sizeof(str)); // add '\0' at the end
```

```
    char *p = str;
```

```
    printf("B = [%d], ", sizeof(p)); /* p is a pointer,
```

```

        so sizeof(p) means sizeof a pointer */
printf("C = [%d], ", sizeof(n)); /* n is an integer, is 4 Bytes */

Foo("Hello world");

void *q = malloc(100);
printf("E = [%d]\n", sizeof(q)); /* q is a pointer */

return 0;
}
// result: A = [25], B = [4], C = [4], D = [4], E = [4]

```

### 3.2.1 关于 sizeof 的思考

sizeof 用来计算字符串的长度，得到的结果是字符串长度加一，sizeof(pointer) 得到的是指针的长度，而非字符串的长度，是存放指针地址的长度。在 Foo(char str[100]) 中，参数传进来的是一个指针。

### 3.2.2 ♠ 关于 malloc 的思考

char \*s = malloc(100) 所表达的意思是指针 s 的地址为分配的 100 个字节内存的 ♠ 首地址。

## 3.3 编写 strcat 函数

```

char* strncat(char *dest, const char *src, size_t n)
{
    size_t dest_len = strlen(dest);
    size_t i;

    for (i = 0 ; i < n && src[i] != '\0' ; i++)
        dest[dest_len + i] = src[i];
    dest[dest_len + i] = '\0';

    return dest;
}

```

### 3.3.1 为什么 strcat 使用 char \* 类型的返回值？

答曰：为了方便赋值给其他变量

## 3.4 使用线程如何防止出现大量的波峰

答曰：使用线程池，线程池具有可以同时提高调度效率和限制资源利用的好处，当线程池里的线程达到最大数时，其他线程都会排队等候。



### 3.5 函数模板和类模板的区别

答曰：函数模板的实例化是由编译器在处理函数调用时自动完成的，而类模板的实例化，必须由程序员在程序中显式的指定。

### 3.6 一般数据库的日志满了，会出现什么情况，是否还能使用

答曰：只能执行查询操作，不能执行更改、备份等操作，原因是任何写操作都得记录日志，也就是说基本处于不可使用状态。

## 4 完美世界面试，职位：脚本开发工程师

### 4.1 基础题

#### 4.1.1 如何查看当前系统的配置，包括内存、CPU、硬盘、内核版本、操作系统维修保养或装饰内容

```
# 内存 free or vmstat
free
vmstat
```

```
# CPU
lscpu
```

```
# 硬盘
fdisk -l
```

```
# 内核版本
uname -r
```

#### 4.1.2 Linux 下如何更新主机名、IP 地址、DNS

```
# 更新主机名
hostname inet.com
```

```
# 更改 IP 地址
ifconfig ethX x.x.x.x
```

```
# 添加一个新的 DNS
echo "nameserver x.x.x.x" >> /etc/resolv.conf
```

```
# 更改 DNS
vim /etc/resolv.conf ...
```

#### 4.1.3 如何查看、添加路由

```
# 查看路由
route
```

```
# 添加默认路由
route add default gw 192.168.1.10
```

4.1.4 某文件的组外权限为只读，所有者为全部权限，组内成员的权限为读与写，则文件的权限为

# `-rwxrw-r--` 也就是 `764`

4.1.5 分别描述一下 `${}`，`$?`，`$*`，`$$` 在 `bash` 中的用途

```
# ${} 在大括号内可以包含变量，使用此符号表示使用括号内的变量，
# $? 表示上一次执行后的返回结果是否是成功或者失败，0 为成功，其余都为失败，
# $$ 表示的是当前进程的的进程号。
```

4.1.6 将一个程序运行的结果输出到 `/tmp/finish.txt`，错误信息追加到 `/tmp/error.txt`

```
1>/tmp/finish.txt 2>>/tmp/error.txt
```

4.1.7 查找 `/home/system/` 路径内创建时间大于 30 天、属主为 `root` 用户、文件前缀为 `sys` 的 `socket` 类型文件，不包含子目录

```
find /home/system/ -mtime +30 -maxdepth 1 -user root -name sys* -type s -print
```

4.1.8 有一个 `tar` 包文件 `files.tar`，目前需要将 `files.tar` 中的 `test1.txt` 解压到 `/tmp/` 将 `/home/system/sys.log` 追加到 `files.tar`，需要怎么做

```
mkdir -p /tmp/temp/
tar -xf files.tar -C /tmp/temp/
cp /tmp/temp/test1.txt /tmp/
rm files.tar
cp /home/system/sys.log /tmp/temp/
tar cf files.tar /tmp/temp/*
```

# or `[tbd]`

4.1.9 在对 `Linux` 系统分区进行格式化的时候需要对磁盘簇（或 `i` 节点密度）的大小进行选择，请说明选择的原则。

磁盘簇<sup>1</sup>（或 `i` 节点密度）是文件系统调度文件的基本单元。磁盘簇的大小，直接影响系统调度磁盘空间效率。当磁盘分区较大时，磁盘簇也应选得大些；当分区较小时，磁盘簇应选得小些。通常使用经验值。

<sup>1</sup> 文件系统是操作系统与驱动器之间的接口，当操作系统请求从硬盘里读取一个文件时，会请求相应的文件系统（`FAT 16/32/NTFS/extX`）打开文件。扇区是磁盘最小的物理存储单元，但由于操作系统无法对数目众多的扇区进行寻址，所以操作系统就将相邻的扇区组合在一起，形成一个簇，然后再对簇进行管理。每个簇可以包括 2、4、8、16、32 或 64 个扇区。显然，簇是操作系统所使用的逻辑概念，而非磁盘的物理特性。为了更好地管理磁盘空间和更高效地从硬盘读取数据，操作系统规定一个簇中只能放置一个文件的内容，因此文件所占用的空间，只能是簇的整数倍；而如果文件实际大小小于一簇，它也要占一簇的空间。所以，一般情况下文件所占空间要略大于文件的实际大小，只有在少数情况下，即文件的实际大小恰好是簇的整数倍时，文件的实际大小才会与所占空间完全一致。

4.1.10 添加 iptables 的策略，a 仅允许 192.168.253.30 通过 eth0 访问本机 80 端口其他连接全部拒绝；b 本机对外访问全部开放

*# a*

```
iptables -N inet # create a new chain
iptables -A inet --src 192.168.253.30 -j ACCEPT # allow 192.168.253.30
iptables -A inet -j DROP # drop everyone else
# use chain inet for packets coming to TCP port 80
iptables -I INPUT -m tcp -p tcp --dport 80 -j inet
```

*#b*

```
iptables -A INPUT -j ACCEPT
```

4.1.11 如何将本地 80 端口的请求转发到 8080 端口，当前主机为 192.168.16.1，本地网卡为 eth0

```
iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
```

4.1.12 如何使用 mysqldump 命令将表结构导出至/tmp/mysql.sql 文件，不需要数据

```
mysqldump -d -h localhost -u root -pmypassword databasename > /tmp/mysql.sql
```

4.1.13 如果 mysql 数据库中的 webdata 表索引损坏如何修复

```
mysqlcheck --repair --databases webdata
```

4.1.14 在文件 a.txt 中查找以当前日期开头、中间包含一个 IP 地址、数字结尾的行，写出查找过程

```
grep -n "[0-9]" a.txt | grep "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" | grep "[0-9]$"
```

4.1.15 在 Linux 中将一个以空格分割的 CSV 文件导入到 mysql 数据库中，并获取数据状态比如：成功、失败条数

```
> load data infile 'x.csv' into table x fields terminated by ' '
enclosed by '"' lines terminated by '\n';
```

*#or write a script*

4.1.16 在 Linux 环境中如何查看远程 Linux 系统运行了多长时间

```
# use ssh to connect to a remote machine, or set up a software on the remote machine
uptime | awk '{print $3}' | tr -d , # will tell the local machine's running time
```

4.1.17 在 crontab 中设置/bin/test.sh 每周二 10 点运行一次，写出 crontab 里的内容

```
# m h dom mon dow  command
23 0 10 * * 2 /bin/test.sh
```

4.1.18 如何将/tmp/目录的所有.log 文件名更改为.txt 文件名

```
cd /tmp/ && rename 's/log/txt/g' *.log
#or
rename 's/log/txt/g' /tmp/*.log
```

4.1.19 使用脚本语言，如何将字符串 abcde 以空格为分割保存到数组 array 里

[tbd]

4.1.20 使用脚本语言将一个数组的元素按照数字从小到大的顺序输出

```
#!/bin/bash
array=(a c b "f f" 3 5)

readarray -t sorted < <(<for a in "${array[@]}"; do echo "$a"; done | sort)

for a in "${sorted[@]}"; do echo "$a"; done
```

4.1.21 脚本环境里 (python) 如何打开、读取、关闭一个文件

```
#open
f = open('filename', 'r')

#read
lines = f.read()

#close
f.close()
```

## 4.2 应用题

4.2.1 用 shell 编程，判断一个文件是不是字符设备，如果是将其拷贝到/dev/路径

```
#!/bin/bash
[ -c /xxx/filename ] && cp /xxx/filename /dev/
```

4.2.2 设计一个脚本，添加新组为 class1，然后添加属于这个组的用户，形式为 stdxx 其中 xx 为 01 到 30

```
#!/bin/bash
groupadd class1
```

```

for i in {9901..9930}
do
    xx=`echo $i | sed 's/99//g'`
    useradd -g class1 std$xx
    echo std$xx | passwd std$xx --stdin
    echo -e "user std$xx password is std$xx" >> /root/newuser.txt
done

```

#### 4.2.3 使用 bash 写一个磁盘清理、报警脚本，具体要求如下

- 脚本在 crontab 中运行，每天 01:00 点钟运行
- 检查当前服务器/和/home 分区磁盘占用情况，当超出阈值就清理该路径里修改时间大于 30 天的文件，阈值可由用户作为参数传递给脚本
- 清理日志后发送邮件给指定人员，报告清理之前、清理之后磁盘占用情况

#### 4.2.4 有如下两个文件，需要对数据做累加处理，规则如下

文件 a 和 b 中的第一列重复部分做累加，不重复部分直接输出

```

# a.txt                                # b.txt
def1 100                                def1 50
def2 100                                def2 50
def3 100                                def9 50

# 结果如下
def9 50
def1 150
def2 150
def3 100

```

#### 4.2.5 处理以下文件内容，将域名取出并进行排序

```

http://www.baidu.com/index.html
http://www.baidu.com/1.html
http://post.baidu.com/index.html
http://www.baidu.com/3.html
http://post.baidu.com/2.html
# 得到如下结果
# 域名出现的次数 域名
3 www.baidu.com
2 post..baidu.com
1 mp3.baidu.com

```

4.2.6 有如下文件，第一列为服务器名，第二列为服务器 IP，写程序实现读取各个服务器根分区磁盘占有量，输出服务器名与占有量，文件内容如下所示

```
Server1 192.168.253.13
Server2 192.168.253.10
Server3 192.168.253.234
Server4 192.168.253.215
Server5 192.168.253.100
```

# 输出如下所示

```
Server1 75%
Server2 91%
Server3 10%
Server4 45%
Server5 76%
```

## 5 websense 面试，职位：C 工程师

### 5.1 Part 1 Basic Skills

#### 5.1.1 Explain the differences between a static library and a dynamic library in Linux System

(static) lib Static libraries (.a): Library of object code which is linked with, and becomes part of the application.

(dll-A) Dynamically linked shared object libraries (.so): There is only one form of this library but it can be used in two ways.

(dll-B) Dynamically linked at run time but statically aware. The libraries must be available during compile/link phase.

(dll-C) The shared objects are not included into the executable component but are tied to the execution.

(dll-D) Dynamically loaded/unloaded and linked during execution (i.e. browser plug-in) using the dynamic linking loader system functions.

#### 5.1.2 What are the Linux inter-process communication (IPC) mechanisms? How can a parent process communicate with its child process(es)

In computing, inter-process communication (IPC) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.

(Signals) Signals are one of the oldest inter-process communication methods used by Unix TM systems. They are used to signal asynchronous events to one or more processes. A signal could be generated

by a keyboard interrupt or an error condition such as the process attempting to access a non-existent location in its virtual memory. Signals are also used by the shells to signal job control commands to their child processes.

**(Pipes)** In Linux, a pipe is implemented using two file data structures which both point at the same temporary VFS inode which itself points at a physical page within memory.

**(Sockets)** Network thing.

**(System V IPC Mechanisms)** Linux supports three types of interprocess communication mechanisms that first appeared in Unix TM System V (1983). These are message queues, semaphores and shared memory. These System V IPC mechanisms all share common authentication methods. Processes may access these resources only by passing a unique reference identifier to the kernel via system calls. Access to these System V IPC objects is checked using access permissions, much like accesses to files are checked. The access rights to the System V IPC object is set by the creator of the object via system calls. The object's reference identifier is used by each mechanism as an index into a table of resources. It is not a straight forward index but requires some manipulation to generate the index. All Linux data structures representing System V IPC objects in the system include an `ipc_perm`

structure which contains the owner and creator process's user and group identifiers. The access mode for this object (owner, group and other) and the IPC object's key. The key is used as a way of locating the System V IPC object's reference identifier. Two sets of keys are supported: public and private. If the key is public then any process in the system, subject to rights checking, can find the reference identifier for the System V IPC object. System V IPC objects can never be referenced with a key, only by their reference identifier.

**(Message Queues)** Message queues allow one or more processes to write messages, which will be read by one or more reading processes. Linux maintains a list of message queues, the `msgque` vector; each element of which points to a `msqid_ds` data structure that fully describes the message queue. When message queues are created a new `msqid_ds` data structure is allocated from system memory and inserted into the vector.

**(Semaphores)** In its simplest form a semaphore is a location in memory whose value can be tested and set by more than one process. The test and set operation is, so far as each process is concerned, uninterruptible or atomic; once started nothing can stop it. The result of the test and set operation is the addition of the current value of the semaphore and the set value, which can be positive or negative. Depending on the result of the test and set operation one process may have to sleep until the semaphore's value is changed by another process. Semaphores can be used to implement critical regions, areas of critical code that only one process at a time should be executing.

Say you had many cooperating processes reading records from and writing records to a single data file. You would want that file access to be strictly coordinated. You could use a semaphore with an initial value of 1 and, around the file operating code, put two semaphore operations, the first to test and decrement the semaphore's value and the second to test and increment it. The first process to access the file would try to decrement the semaphore's value and it would succeed, the semaphore's value now being 0. This process can now go ahead and use the data file but if another process wishing to use it now tries to decrement the semaphore's value it would fail as the result would be -1. That process will be suspended until the first process has finished with the data file. When the first process has finished with the data file it will increment the semaphore's value, making it 1 again. Now the waiting process can be woken and this time its attempt to increment the semaphore will succeed.

**(Shared Memory)** Shared memory allows one or more processes to communicate via memory that appears in all of their virtual address spaces. The pages of the virtual memory is referenced by page table entries in each of the sharing processes' page tables. It does not have to be at the same address in all of the processes' virtual memory. As with all System V IPC objects, access to shared memory areas is controlled via keys and access rights checking. Once the memory is being shared, there are no checks on how the processes are using it. They must rely on other mechanisms, for example System V semaphores, to synchronize access to the memory.

## 5.2 Part 2 Networking

5.2.1 Describe UDP and TCP, and the differences between them

5.2.2 What's the differences between a Hub, a Switch and a Route

5.2.3 What's the differences between ACTIVE FTP and PASSIVE FTP

5.2.4 How many available IP address in subnet 210.27.48.21/30

## 5.3 Part 3 C/C++ Programming

5.3.1 Please implement the function that reverses a single linked list

```
typedef struct node
{
    int data;
    struct node *next;
} linknode;

linknode *reverse(linknode *head);
```

5.3.2 What would be the output of the following C program

```
#include <stdio.h>
int main()
{
    int a[3] = {2, 3, 4};
    char *p;
    p = a;
    p = (char *)((int *)p + 1);
    printf("%d\n", *p);
    return 0;
}

/* answer is 3 */
```



### 5.3.3 Predict the output of the following program code

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

int main()
{
    fork(); fork(); fork();
    printf("Hello world\n");
    return 0;
}
/* the answer is six line "Hello World" */

#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world\n");
    fork(); fork(); fork();
    return 0;
}
/* the answer is one line "Hello World" */
```

第一个问题的答案： $2^n - 2$  行 *Hello World*，也就是  $2^3 - 2 = 6$  行 *Hello World*

第二个问题的答案：一行 *Hello World*

### 5.3.4 Write the C program language expression using variable "a" according to statement below

- (a) An array of 10 pointers to integers
- (b) A pointer to an array of 10 integers
- (c) A pointer to a function that takes an integer as an argument and returns an integer
- (d) An array of 10 pointers to functions that take an integer argument and return an integer

```
//[a] an array of 10 pointers to integers
int *a[10];
```

```
//[b] a pointer to an array of 10 integers
int (*a)[10];
```

```
//[c] a pointer to a function that takes an ...
int (*a)(int);
```

```
//[g] an array of 10 pointers to functions that takes an integer argument
// and return an integer
int (*a[10])(int);
```

## 6 某语音驾驶公司的面试题目

### 6.1 用预处理指令声明一个常数，表示一年之内的秒数（忽略闰年问题）

```
#define YEAR_SECONDS (365 * 24 * 60 * 60)
//#define YEAR_SECONDS 31536000
```

### 6.2 写一个标准的 MIN 宏，这个宏输入两个参数，并返回较小的那个

```
//header file of param.h
#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))
```

### 6.3 嵌入式系统中用到无限循环，怎么用 C 写死循环

```
while(1) { }
# or
for(;;) { }
```

### 6.4 用变量 a 为给出下面定义

- (a) 一个整型数
- (b) 一个指向整型数的指针
- (c) 一个指向指针的指针，它指向的指针是指向一个整型数
- (d) 一个有 10 个整型数的数组
- (e) 一个有 10 个指针的数组，该指针是指向一个整型数的
- (f) 一个指向有 10 个整型数数组的指针
- (g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数
- (h) 一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数

```
//(a) an integer
int a;
```

```
//(b) a pointer to an integer
int *a;
```

```
//(c) a pointer to a pointer to an integer
int **a;
```

```
//(d) an array of 10 integers
int a[10];
```

```
//(e) an array of 10 pointers to integers
```

```
int *a[10];

// (f) a pointer to an array of 10 integers
int (*a)[10];

// (g) an array of 10 pointers to functions that takes an integer argument
// and return an integer
int (*a[10])(int);
```

## 6.5 关键字 static 的作用

- (1) 在函数体内，存储于静态存储区<sup>2</sup> 所以它在被下次函数调用的时候仍然保存上一次更改的值。
- (2) 在一个文件模块内，一个被声明为静态的变量可以被模块内的所有函数访问，但不能被此文件模块外的函数所访问，它属于一个本地的全局变量。
- (3) 在文件模块内，一个被声明为静态的函数只可被这一文件模块内的其他函数所调用。

## 6.6 关键字 const 的作用

简单的回答：只读变量

### 6.6.1 如果上个题目回答正确，引出下一个题目

```
const int a;
int const a; /* 第一个和这二个意义相同，a 是一个常整型数，不可改变 */

const int* a; /* a 是一个指向常整型数的指针，也就是说整型数是不可修改的，但指针可以修改。 */

int* const a; /* a 是一个指向整型数的常指针，指针指向的整型数是可以修改的，但指针不可修改 */

int const* a const; /* a 是一个指向常整型数的常指针（指针指向的整数是不可修改的，指针也不可以修改 */
```

## 7 下面函的输出情况

```
void foo(void)
{
    unsigned int a = 6;
    int b = -20;
    (a + b > 6)? puts(">6"): puts("<=6");
}

/* answer: >6 */
```

---

<sup>2</sup>内存存在程序编译的时候就分配好了，这块内存存在程序整个运行期间都存在，它主要存放静态数据、全局数据和常量，包括字符串常量

## 7.1 volatile

volatile 提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据。如果没有 volatile 关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致的现象。

主要的使用场合：

- (1) 并行设备的硬件寄存器如状态寄存器
- (2) 一个中断服务子程序中会访问到的非自动变量
- (3) 多线程场合中中被几个任务共享的变量

### 7.1.1 加一个问题

- (1) 一个参数可以是 const 还可以是 volatile 吗？解释为什么
- (2) 一个指针可以是 volatile 吗，为什么
- (3) 下面代码中的函数有什么错误

```
int square(volatile int *ptr)
{
    return *ptr**ptr;
}
```

- (1) 可以是，一个例子是只读的状态寄存器。它是 volatile 因为它可能被意想不到地改变。它是 const 因为程序不应该试图修改它。
- (2) 可以是，尽管这不常见，一个例子是当一个中断服务子程序修改一个指向 buffer 的指针时
- (3) 变态的代码，不对，下面的代码可以解释这个问题

```
int square(volatile int *ptr)
{
    return *ptr * *ptr;
}
```

/\* 等价于下面的代码 \*/

```
int square(volatile int *ptr)
{
    int a, b;
    a = *ptr;
    b = *ptr; /* 由于 *ptr 可能被意想不到的改变，因此 a 和 b 有可能是不同的
                所以这段代码不正确 */

    return a * b;
}
```

/\* 可以修改为如下代码，但不提倡胡乱用 volatile \*/

```
long square(volatile int *ptr)
{
    int a;
    a = *ptr;
    return a * a;
}
```

7.2 嵌入式系统总是要用户对变量或寄存器进行位操作。给定一个整型变量 *a*，写两段代码，第一个设置 *a* 的 bit 3，第二个清除 *a* 的 bit 3。在以上操作中，要保持其他位不变。

这个问题对于非嵌入式程序员来说可能会答错

```
#define BIT3 (0x01 << 3)

static int a;

void set_bit3(void)
{
    a |= BIT3;
}

void clear_bit3(void)
{
    a &= ~BIT3;
}
```

7.3 嵌入式系统经常需要访问某特定的内存位置。在某工程中，需要设置绝对位置为 0x67a9 的整型变量的值为 0xaa66，写代码去完成此任务。

```
int *ptr;
ptr = (int*)0x67a9;
*ptr = 0xaa55;
```

7.4 下面函数的结果预测

```
#include <stdio.h>

int main()
{
    int i = 10, j = 10, k = 3;
    k *= i + j;
    /*      same as:
           k = k * (i + i);
    */
    printf("%d\n", k);
    return 0;
}
```

```
}
/* result: 60 */

/* 这个结果我真没有预测到，但只有二货才会写这种代码，考察优先级，
赋值运算的优先级最低 */
```

## 7.5 写出下面代码的输出结果

```
#include <stdio.h>
int inc(int a)
{
    return ++a;
}

int multi(int* a, int* b, int* c)
{
    return (*c = *a * *b);
}

typedef int (FUNC1)(int in);
typedef int (FUNC2)(int*, int*, int*);

void show(FUNC2 fun, int arg1, int* arg2)
{
    FUNC1 *p = &inc;
    int temp = p(arg1);
    fun(&temp, &arg1, arg2);
    /*
        arg2 = temp * 10 == 11 * 10 == 110
    */
    printf("%d\n", *arg2);
}

int main()
{
    int a;
    show(multi, 10, &a);
    return 0;
}

/* answer: 110 */
```

## 7.6 关于内存的几个经典面试题

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

```
void getMemory1(char *p)
{
    p = (char *)malloc(100);
}

void test1(void)
{
    char *str = NULL;
    getMemory1(str);
    strcpy(str, "Hello World");
    printf("%s", str);
}

int main()
{
    test1();
    return 0;
}

/* result: Segmentation fault (core dumped) */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *getMemory2(void)
{
    char p[] = "Hello world"; /* static char p[] = "Hello world" 这样可以，把数据放在
                               静态区，而不是栈上 */
    return p;
}

void test2(void)
{
    char *str = NULL;
    str = getMemory2();
    printf("%s", str);
}

int main()
{
    test2();
    return 0;
}

/*result: 基本上应该输出的是乱码，因为返回的是个栈元素 */

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

```
char *getMemory3(void)
{
    return "Hello world"; /* Hello world 存储在常量区，正确 */
}

void test3(void)
{
    char *str = NULL;
    str = getMemory3();
    printf("%s", str);
}

int main()
{
    test3();
    return 0;
}

/* result: 结果正确， 为:Hello world 字符串存放在程序的常量区，而且没有被修改过 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void getMemory4(char **p, int num)
{
    *p = (char *)malloc(num);
}

void test4(void)
{
    char *str = NULL;
    getMemory4(&str, 100);
    strcpy(str, "Hello world");
    printf("%s", str);
    //free(str);
}

int main()
{
    test4();
    return 0;
}

/* result: 内存没释放 依然错误 */

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



```

void test5(void)
{
    char *str = (char *) malloc(100);
    strcpy(str, "Hello ");
    free(str);
    if (str != NULL) {
        strcpy(str, "world");
        printf("%s", str);
    }
}

int main()
{
    test5();
    return 0;
}

/* str 为野指针，打印的结果不得而知 */

```

## 7.7 关于 free()

内存被 free() 后只是内核中把那块内存的标志设置为了可用，但这块内存如果不被别的程序使用，仍然可用，所以在 free() 执行后仍然可以使用那块内存。

```

#include <stdio.h>
#include <stdlib.h>
int main()
{
    char *s = malloc(100);
    snprintf(s, 100, "Hello World");
    free(s); // 虽然在这里释放了，但我们仍然能输出 s，这种做法是不对的，应该用完后 free
    printf("[%s]\n", s);
    return 0;
}

```

## 8 以下问题来自网络

### 8.1 typedef 在 C 代码中使用的次数很多，也可以用预处理去做类似的事情，思考一下哪个比较好

```

/* 显然是用 typedef 比较好，因为宏只是简单的代码替换，会出问题，如下所示 */
#define DPS struct int*
typedef struct int* TPS

DPS a, b; // ==> int *a, b;
TPS a, b; // ==> int *a, *b;

```

## 8.2 如何引用一个已经定义的全局变量

`extern`

## 8.3 全局变量可不可以定义在可被多个 C 文件所包含的头文件中，为什么

可以，在不同的 C 文件中用 `static` 形式声明同名的全局变量，一般情况下是在 C 文件中定义，在头文件中使用 `extern dataType x` 声明。

## 8.4 `for(;1;)` 有什么问题？它是什么意思

和 `while(1)` 相同

## 8.5 请写出下面代码的输出结果

```
#include <stdio.h>
int main()
{
    int a, b, c, d;
    a = 10;
    b = a++;
    c = ++a;
    d = 10 * a++;
    printf("b = %d, c = %d, d = %d\n", b, c, d);
    return 0;
}

/* b = 10, c = 12, d = 120 */
/* 此问题很容易让人犯错 */
```

## 8.6 `static` 全局变量和普通全局变量有什么区别，`static` 局部变量和普通局部变量有什么区别，`static` 函数和普通函数有什么区别

这个问题在以上问题中出现过，这里在说一遍：`static` 全局变量属于静态全局变量，全局变量本身就属于静态存储，静态全局变量当然也是如此，区别是非静态全局变量的作用域是整个源程序，当一个源程序有多个文件组成，非静态全局变量在所有文件中都有效，而静态全局变量则限制了其作用域，它只在定义该变量的文件内有效。`static` 函数与普通函数的作用域不同，只在当前文件中使用的函数可以声明为 `static` 函数。`static` 全局变量和普通全局变量的区别：`static` 全局变量只初始化一次。`static` 局部变量和普通局部变量的存储位置有别，一个是静态区，一个是栈区。`static` 函数和普通函数的区别：`static` 函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝。

## 8.7 程序的存储

局部变量存储在栈中，全局变量存放在静态区，动态申请的存放在堆中。

## 9 微软的面试，来自网络

### 9.1 求下面函数的返回值

```
#include <stdio.h>
int func(int x)
{
    int countx = 0;
    while(x) {
        countx++;
        x = x&(x-1);
    }
    return countx;
}

int main()
{
    printf("[%d]\n", func(9999));
    return 0;
}
```