一、选择题 （5x6=30）

1. 用户与内核的接口是什么

```
系统调用
```

2. 删除文件夹的命令

```
rmdir 或 rm -rf
```

3. 重定向-输出文件内容

```
>>
```

4. 切换用户的命令

```
su user
```

二、简答题（2x10=20）

1. 为什么Linux引入makefile？和其他脚本的区别？使用makefile编译系统有哪些特点？

```
Makefile 是 Linux 系统引入的一种脚本语言，它用于管理编译过程，可以自动完成编译、链接、安装等任务。与其他脚本语言相比，Makefile 具有以下几个特点：

简洁：Makefile 语法简单，易于阅读和编写，相对于其他脚本语言来说，Makefile 的语法规则更加简洁明了。

可维护性：Makefile 可以自动检测文件变化，并在编译时自动更新依赖项，从而提高编译效率，同时也提高了代码的可维护性。

灵活性：Makefile 可以根据需要进行灵活的配置和修改，例如可以修改编译选项、链接选项、依赖项等等。

自动化：Makefile 可以自动完成编译、链接、安装等任务，从而简化了开发流程，提高了工作效率。

与其他编译系统相比，使用 Makefile 具有以下特点：

高效性：Makefile 可以根据依赖关系自动完成编译和链接任务，减少了手动操作的错误和时间成本。

可扩展性：Makefile 可以根据需要进行灵活的配置和修改，从而方便地扩展和修改编译过程。

可维护性：Makefile 可以自动检测文件变化，并在编译时自动更新依赖项，从而提高编译效率，同时也提高了代码的可维护性。

灵活性：Makefile 可以根据需要进行灵活的配置和修改，例如可以修改编译选项、链接选项、依赖项等等。

```

总之，Makefile 是一种高效、灵活、可扩展、可维护的编译系统，它可以提高编译效率和代码的可维护性，同时也可以方便地扩展和修改编译过程。在 Linux 系统中，Makefile 被广泛应用于编译、安装、维护等方面。

2. 硬链接与软链接的区别？（至少三点）用shell命令和函数如何创建？
    1. 硬链接和软链接是 Linux 中用于链接文件或目录的两种不同方式。它们的主要区别在于它们链接的目标文件或目录不同，以及它们的安全性和访问控制方面。以下是它们的区别：
        1. **链接的目标文件或目录不同**：硬链接和软链接链接的目标文件或目录是不同的。硬链接链接的是已经存在的文件或目录，而软链接链接的是一个不存在的文件或目录。
        2. **安全性和访问控制**：由于硬链接和软链接的目标文件或目录不同，因此它们的安全性和访问控制也有所不同。对于硬链接，它们共享相同的链接文件或目录，因此对于硬链接的所有者来说，只需要控制链接文件或目录的访问权限即可控制整个链接文件或目录的访问权限。而对于软链接，它们链接的目标文件或目录是不同的，因此需要分别控制每个软链接文件或目录的访问权限，以确保访问控制的有效性。
        3. 链接的数量：硬链接和软链接可以链接到同一个文件或目录上，但是它们的链接数量是受到限制的。在 Linux 中，每个文件或目录最多可以链接 9 个硬链接，每个硬链接最多可以链接 9 个文件或目录。而对于软链接，它们可以链接到任意数量的文件中，但是每个软链接只能链接到一个文件或目录。

三、编程题（15+20+15=50）

1. 看代码，是否有缓冲区溢出？如何改进？
2. 用系统调用实现输出给定文件夹中所有文件的名字 用空格隔开，并在文件夹及文本文件的输出后注上“（文件夹）”“（文本文件）”

```c
//New Bing
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>

// 定义一个函数，用于递归地列出给定路径下的所有文件
void listFilesRecursively(char *path)
{
    struct dirent *dp;
    DIR *dir = opendir(path);
    char newPath[1024];

    // 如果无法打开目录，则返回
    if (!dir)
        return;

    // 读取目录中的每个条目
    while ((dp = readdir(dir)) != NULL)
    {
        // 跳过 "." 和 ".." 条目
        if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0)
        {
            // 输出文件名
            printf("%s %s\n", path, dp->d_name);

            // 构造新路径
            strcpy(newPath, path);
```

```
            strcat(newPath, "/");
            strcat(newPath, dp->d_name);

            // 检查新路径是否为目录
            struct stat path_stat;
            stat(newPath, &path_stat);
            if (S_ISDIR(path_stat.st_mode))
                // 如果是目录，则递归调用此函数
                listFilesRecursively(newPath);
        }
    }

    // 关闭目录
    closedir(dir);
}

int main()
{
    char path[1024];

    // 获取用户输入的路径
    printf("Enter path to list files: ");
    scanf("%s", path);

    // 调用函数，列出给定路径下的所有文件
    listFilesRecursively(path);

    return 0;
}
```

```
//ChatGPT
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <dirent.h>

#define MAX_PATH_LENGTH 1024

int main(int argc, char *argv[]) {
    char full_path[MAX_PATH_LENGTH];
    char *path = argv[1];
    struct stat st;

    if (argc < 2) {
        printf("Usage: %s directory\n", argv[0]);
        exit(1);
    }

    if (stat(path, &st) != 0) {
        printf("Error: could not find directory %s\n", path);
        exit(1);
    }

    if (st.st_mode & S_IFMT) {  // 如果目录不是只读的
        printf("%s\n", path);
    } else {
        for (;;) {
```

```
            DIR *dp;
            struct dirent *dent;
            if ((dp = opendir(path)) == NULL) {
                printf("Error: could not open directory %s\n", path);

                exit(1);
            }
            while ((dent = readdir(dp)) != NULL) {
                char buffer[MAX_PATH_LENGTH];
                sprintf(buffer, "%s/%s", path, dent->d_name);
                if (stat(buffer, &st) == 0) {  // 如果文件不是只读的
                    printf("%s\n", buffer);
                }
            }
            closedir(dp);
        }
    }

    return 0;
}

```

1. shell脚本编程，获得用户输入的100个整数，并输出其最大值，最小值，总和。

```
#!/bin/bash

# 获取用户输入的整数列表
echo "请输入 100 个整数，用空格分隔:"
read -a integers

# 初始化最大值、最小值和总和为 0
max=0
min=0
sum=0

# 循环遍历数组中的每个整数
for i in "${integers[@]}"
do
  # 计算当前整数的值
  value=$(printf "%03d" "$i")

  # 如果当前整数大于最大值，则更新最大值
  if [ "$value" -gt "$max" ]
  then
    max="$value"
  fi

  # 如果当前整数小于最小值，则更新最小值
  if [ "$value" -lt "$min" ]
  then
    min="$value"
  fi

  # 计算总和
  sum=$((sum + value))
done

```

```
# 输出最大值、最小值和总和
echo "最大值为:$max"
echo "最小值为:$min"
echo "总和为:$sum"
```