一、(36分) 简答下面问题：

1. 如果两个常规文件的长度一样，那么是否意味着它们占用相同大小的磁盘空间？为什么
   不一定。因为常规文件即使长度一样，但是有的文件有空洞，文件中的空洞并不要求在磁盘上占用存储区。(p54)
2. 读/写磁盘文件时，每次使用系统调用 read 和 write 是否都意味着读写磁盘驱动器？为什么？(p59)
   不一定。
   大多数文件系统为改善性能都使用某种预读技术。当检测到正在进行的顺序读取时，系统就试图读入比应用所要求的更多数据，下一次read就不用访问磁盘。
   当我们向磁盘写入数据时，内核通常先将数据复制进缓冲区中，然后排入队列，晚些时候再写入磁盘。
   学姐：取决于用户缓冲区的大小和是否同步写，一般为了提高效率，系统通常是批量的读写磁盘数据存入缓冲区，以减少磁盘操作次数。
3. 如果有一个文件，它的权限被设置成文件的所有者不可读。那么文件的所有者是否能将其改为自己可读？如果其他用户可读，那么文件的所有者是否能将文件改为自己可读？(p79~80,我不会这道题)
   如果文件所有者不可读，他通常可以更改权限以使自己可读。如果其他用户可读，所有者通常也有权限更改文件权限。(不确定)
   学姐：可以，可以。
4. 在作业控制中，进程组和进程对话有什么用途？(进程组:p233;进程会话:p234;作业控制:p237~240;不会回答)
   进程组：一个或多个进程的集合。通常，它们与同一作业关联，可以接收来自同一终端的各种信号。
   会话：一个或多个进程组的集合。
   作业控制：允许在一个终端上启动多个作业(进程组)，控制哪一个作业可以访问该终端以及哪些作业在后台运行。
   学姐：
   对话是进程组的集合，它们都是作业控制系统的功能。进程可以同时向一族进程发信号，或者使用waitpid等待进程组中的一个子进程；一个对话将组成该对话的进程组与同一个控制终端联系起来。
   我们可以将信号发送给一个进程组。进程组中的所有进程都会收到该信号。
   会话的意义在于将多个工作囊括在一个终端，并取其中的一个工作作为前台，来直接接收该终端的输入输出以及终端信号。其他工作在后台运行。
5. 如果存在一个未决信号，如何在不执行相关的信号处理函数的前提下清除它？(未决信号定义：p266)
   答案1：使用signal(SIGNO, SIG_IGN)清除未决信号。(p256:signo参数填写未决信号的信号名，指定SIG_IGN的意思是向内核表示忽略此信号;)
   答案2：使用sigwait函数，将要清除的未决信号加入sigwait第一个参数set中。将进程控制块中的信号处理方法单元对应的字段设置为SIG_IGN。
   不正确答案：

可以使用sigpending和sigprocmask函数来检查和屏蔽信号。(p267,p275,p276,但是这执行了相关的信号处理函数,别写)

sigprocmask：调用sigprocmask后，如果有任何未决的、不再阻塞的信号，则在sigprocmask返回前，至少会将其中一个信号递送给该进程。

sigpending:该函数返回一个信号集，其中的各个信号对于调用进程是阻塞的而不能递送，因而也一定是当前未决的

6. 为什么要避免进程僵尸？请给出两种避免进程僵尸的方法。

   P189:僵尸进程定义

   僵尸进程在系统中一直占用资源，会占用大量进程号，导致系统因为没有可用的进程号而不能产生新的进程。

   方法一：是父进程通过wait和waitpid等函数等待子进程结束；(P190)(子进程已经终止而且是僵尸进程，wait立即返回并取得该子进程的状态。)

   方法二：调用fork两次(P193)

   方法三：用signal(SIGCHLD,SIG_IGN)通知内核，这样子进程结束后，内核会回收。

7. 从 main 函数返回是否终止进程？函数 exit 与 _exit 有什么区别（至少指出两点）？

   不是，main函数返回后会执行登记的atexit函数，进行释放内存之类的工作，之后结束程序并将main的返回值交给操作系统。

   ①_exit立即进入内核，exit先执行一些清理处理，然后返回内核(p159)

   ②_exit并不执行标准I/O缓冲区的冲洗操作，exit执行标准I/O缓冲区的冲洗操作。(p188)

   ③exit使用头文件stdlib.h，因为它是由IOS C说明的；_exit使用unistd.h，因为它是由POSIC.1说明的。(p159)

   学姐：

   区别一：_exit函数直接调用exit 系统调用关闭进程，而exit函数在调用exit系统调用前会先调用atexit()注册的函数使用户可以在程序中之前执行自己的清理动作；

   区别二：exit函数还会检查文件的打开情况，把文件缓冲区中的内容写回文件，清理I/O缓冲。

8. 什么是系统调用？是否有在C语言标准里定义的库函数是系统调用？

   简略版：内核的接口被称为系统调用。(p1)

   详细版：所有的操作系统都提供多种服务的入口点，由此程序向内核请求服务。各种版本的UNIX实现都提供良好定义、数量有限、直接进入内核的入口点，这些入口点被称为系统调用(p17)

   open、close

   学姐：C语言标准里定义的函数都不是系统调用，但是可

   能该函数本身其实就是调用的系统调用。

9. 如果删除一个已经打开的文件，那么能否继续读写这个打开的文件？

   能。

10. 子进程的ID是大于0还是等于0？它是否与父进程共享内存空间？它继承父进程对信号的处理方式吗？

    子进程的ID大于0。(实验)

    不与父进程共享内存空间。(子是父的副本，只共享正文段)(p182)

    子进程继承父进程对信号的处理方式(信号屏蔽和安排)(p185~186)

11. 什么是竞争条件（race condition）？使用信号机制是否可以避免？(p196)
    当多个进程都企图对数据进行某种处理，而最后的结果又取决于进程运行的顺序时，认为发生了竞争条件
    学姐：两个或多个进程读写某些共享数据，而最后的结果取决于进程运行的精确时序，称为竞争条件。
    能避免，使用信号机制。
12. 大多数函数是不可重入的，因为(p263)
    （a）已知它们使用静态数据结构；
    （b）它们调用malloc或free
    （c）它们是标准I/O函数。标准I/O库的很多实现都以不可重入方式使用全局数据结构。
    注意，虽然在本书的某些实例中，信号处理程序也调用了printf函数，但这并不保证产生所期望的结果，信号处理程序可能中断主程序中的printf函数调用。

二、(15分) 阅读教材程序清单 4-7 (p99)，回答问题

在p105~108

1. dopath 函数返回值的有什么意义?
   ~~返回的是参数func的返回值。(教材注释)~~
   程序递归的遍历目录，返回值非0时结束递归遍历并且返回错误信息。
2. 程序遍历目录树的次序是深度优先还是广度优先？为什么?
   深度优先。因为当它判别到这是个目录之后，会继续读这个目录里的子目录和文件，并对每一个继续调用dopath。
3. 程序遍历目录树时，在每个节点执行的函数是什么？完成什么功能?
   对目录节点执行dopath，完成对子目录的遍历目录树。
   对文件节点执行参数func，完成对文件的对应的func功能。这里main函数里调用的是myfunc，这里分别记录了普通文件、目录文件、块特殊文件、字符特殊文件、FIFO、套接字、符号链接的数量。
   学姐：执行myfunc函数，根据第三个参数的不同，执行不同的操作。
   如果是FTW_F则将该节点对应的文件类型数量的变量加1；如果是FTW_D则表示该文件是目录文件，将统计目录文件数量的变量加1；
   如果是FTW_DNR显示错误信息“无法打开指定目录”；
   如果是FTW_NS显示错误信息“获取文件stat错误”。
4. 在 dopath 函数中，语句（不含引号）“ptr[-1] = 0;” 是否可以省略?
   (是不是题错了)fullname[n]=0不可省略。假如这一次文件或目录的fullname比上一次短，不加终止符截断的话，就会出现错误的更长的文件路径。
5. 在程序中是否可以使用系统调用stat获取目录项 i 节点的信息？为什么？(p74)
   不可以。i 节点可能是符号链接。lstat函数类似于stat, 但是当命名的文件是一个符号链接时，lstat返回该符号链接的有关信息，而不是由该符号链接引用的文件的信息。当以降序遍历目录层次结构时，需要用到lstat。

三、（15分）阅读下列程序代码，请说明第13、14行代码的目的是什么?

```
#include "apue.h"
#include <sys/wait.h>

int main(void)
{
    char buf[MAXLINE];  /* from apue.h */
    pid_t pid;
    int status;

    printf("%s "), /* print prompt (print requires %s to print %) */

    while (fgets(buf, MAXLINE, stdin) != NULL) {
        if (buf[strlen(buf) - 1] == '\n') //13
            buf[strlen(buf) - 1] = 0;     //14

        if ((pid = fork()) < 0) {
            err_sys("fork error");
        } else if (pid == 0) {  /* child */
            execlp(buf, buf, (char *)0);
            err_ret("couldn't execute: %s", buf);
            exit(127);
        }

        /* parent */
        if ((pid = waitpid(pid, &status, 0)) < 0)
            err_sys("waitpid error");
        printf("%s ");
    }
    exit(0);
}
```

如果buf结尾有回车符的话，把它换成终止符，即当fgets的输入内容没有达到(MAXLINE-1)的长度时，会以回车为截断。判断buf结尾是否是回车，是的话删除最后的回车。
模拟shell，实际输入是没有回车符的。

四、

```c
#include <signal.h>
#include "ourhdr.h"

/**
 * signal - 设置指定信号的处理函数。
 *
 * @signo: 信号编号。
 * @func:  信号处理函数的指针。
 *
 * 返回值:
 * - 成功时，返回之前的信号处理函数指针。
 * - 失败时，返回 SIG_ERR。
 */
sigfunc *signal(int signo, Sigfunc *func) {
    struct sigaction act, oact;

    // 设置新的信号处理函数
    act.sa_handler = func;

    // 初始化信号屏蔽集为空
    sigemptyset(&act.sa_mask);

    // 初始化信号标志位
    act.sa_flags = 0;

    // 针对 SIGALRM 信号的特殊处理（可能的非阻塞行为）
    if (signo == SIGALRM) {
#ifdef SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT; /* SunOS 系统: 防止系统调用自动重启 */
#endif
    } else {
#ifdef SA_RESTART
        act.sa_flags |= SA_RESTART; /* SVR4 或 4.4BSD 系统: 允许系统调用自动重启 */
#endif
    }

    // 设置信号动作并保存之前的信号处理配置
    if (sigaction(signo, &act, &oact) < 0)
        return (SIG_ERR);

    // 返回之前的信号处理函数
    return (oact.sa_handler);
}
```

1. **进入信号处理函数时，是否自动阻塞相同的信号？**

   是的。

   根据 POSIX 标准，当一个信号处理函数正在执行时，该信号会被自动阻塞，直到信号处理函数返回。这可以防止信号处理函数的递归调用。
2. **进入信号处理函数后，是否自动恢复默认的信号处理方式？**

   不是。

   代码中的 `sigaction` 并未设置 `SA_RESETHAND` 标志，因此信号处理方式不会自动恢复到默认处理方式。信号处理函数会持续有效，直到被显式更改。
3. **若一个阻塞的低速系统调用被信号中断，那么这个系统调用是否重启？**

   视情况而定。

   - 如果定义了 `SA_RESTART` 并设置了该标志，则低速系统调用会自动重启。
   - 如果没有设置 `SA_RESTART`，信号中断后系统调用将返回 `-1`，并将 `errno` 设置为 `EINTR`。
   - （学姐答案：）如果是被SIGALRM中断则不重启，否则会重启
4. **若其子进程终止，则系统是否会向父进程发送信号（如果你认为会发送，则请说明发送什么信号）？**

   是的。

   系统会向父进程发送 `SIGCHLD` 信号，以通知子进程已终止、停止或继续。父进程可以通过捕捉 `SIGCHLD` 信号来处理子进程的状态变化。

五、

```c
#include"apue.h"
#include"error.c"//自己加的
#include<sys/wait.h>
static int glob;//global static
int main(void){
    pid_t pid;
    int var;//auto
    var=0;
    if((pid=fork())<0)
        err_sys("fork error");
    else if(pid==0){
        glob++;
        exit(1);
    }
    if(wait(NULL)!=pid)
        err_sys("fork error");
    else if(pid==0){
        glob++;
        var++;
        exit(2);
    }
    printf("glob = %d var = %d\n",glob,var);//这里应该是题写错了
    exit(0);
}
```

(p183)
我的运行结果：
glob = 0 var = 0
(学姐答案：都是1)

六、

```
#include "apue.h"
#include"error.c"//自己加的
#include "sys/wait.h"
int main(void){
    char buf[MAXLINE];
    pid_t pid;
    int status;

    while (fgets(buf, MAXLINE, stdin) != NULL) {
        if (buf[strlen(buf) - 1] == '\n')
            buf[strlen(buf) - 1] = 0; /* replace newline with null */
        if ((pid = fork()) < 0) {
            err_sys("fork error");
        } else if (pid == 0) { /* child */
            execlp(buf, buf, (char *)0);
            err_ret("couldn't execute: %s", buf);
            exit(127);
        } /* parent */
        if ((pid = waitpid(pid, &status, 0)) < 0)
            err_sys("waitpid error");
        printf("%% ");
    }
    exit(0);
}
```

（1）子进程的进程ID等于0吗?
不等于。
（2）程序中的函数调用err_ret("couldn't execute: %s", buf)在pid==0时一定被执行吗？为什么?
不一定。因为可能用户直接给完整路径，这样就能成功执行buf，就不用调用err_ret了。

（3）在程序里的waitpid语句起什么作用？如果省略的话，程序的行为如何?
~~waitpid并不等待在其调用之后的第一个终止子进程，它有若干个选项，可以控制它所等待的进程。~~
等待进程id为pid的子进程结束，并将终止状态存放在status中；省略的话父进程可能会在子进程结束前进入下一轮while循环，可能产生僵死进程。