一、选择题(每题 2 分)

1. 下列不是用户进程的组成部分的是(D)

[A] 正文段　　[B] 用户数据段　　[C] 系统数据段　　[D] elf 段

2. 以下哪种不是进程的类型 (B)

[A] 批处理进程　　[B] 管理进程　　[C] 交互进程　　[D] 守护进程

3. 以下哪种方法无法查看进程的信息 (C)

[A] ps　　[B] 查看/proc 目录　　[C] kill　　[D] top

4. fork()的返回值不可能是(C)

[A] -1　　[B] 0　　[C] 1　　[D] 大于 10000 的正整数

5. 可以改变正在运行的进程优先级的命令是(B)

[A] nice　　[B] renice　　[C] kill　　[D] set

6. 下列哪个函数无法传递进程结束时的状态 (A)

[A]close　　[B] exit　　[C] _exit　　[D] return

7. 以下哪种用法可以等待接收进程号为 pid 的子进程的退出状态 (A)

[A] waitpid(pid, &status, 0)　　[B] waitpid(pid, &status, WNOHANG)
[C] waitpid(-1, &status, 0)　　[D] waitpid(-1, &status, WNOHANG)

8. 函数 waitpid 的返回值等于 0 时表示的含义是（B）

[A] 等待的子进程退出　　[B] 使用选项 WNOHANG 且没有子进程退出
[C] 调用出错　　[D] 不确定

9. 下列对无名管道描述错误的是（C）

[A] 半双工的通信模式
[B] 有固定的读端和写端
[C] 可以使用 lseek 函数
[D] 只存在于内存中

10.下列对于有名管道描述错误的是（D）

[A] 可以用于互不相关的进程间
[B] 通过路径名来打开有名管道
[C] 在文件系统中可见
[D] 管道内容保存在磁盘上

11. 下列不属于用户进程对信号的响应方式的是（B）

[A] 忽略信号　　[B] 保存信号　　[C] 捕捉信号　　[D] 按缺省方式处理

12. 不能被用户进程屏蔽的信号是（B）

[A] SIGINT　　[B] SIGSTOP　　[C] SIGQUIT　　[D] SIGILL

13. 默认情况下，不会终止进程的信号是（D）

[A] SIGINT　　[B] SIGKILL　　[C] SIGALRM　　[D] SIGCHLD 忽略信号

14. 下列不属于 IPC 对象的是（A）

[A] 管道　　[B] 共享内存　　[C] 消息队列　　[D] 信号灯

15. 下列哪种机制可以用于线程之间的同步（D）

[A] 信号　　[B] IPC 信号灯
[C] POSIX 有名信号量　　[D] POSIX 无名信号量

二、判断题(每题 1 分)

1. Linux 下进程的模式分为用户态，内核态和系统态（错）
2. 每个进程的进程号和父进程号在进程执行期间不会改变（对）　如果进程退出重新运行，那么进程号就会改变了
3. 子进程被创建后从 fork()的下一条语句开始执行（对）
4. 子进程的进程号等于父进程的进程号加 1（错）
5. 执行_exit()函数时不会清理 IO 缓冲（对）　exit()会刷新缓冲区
6. exec 函数族可以创建一个新的进程来执行指定的程序（错）
7. wait 函数无法接收子进程退出的状态（错）
8. 无名管道只能用于父子进程（对）
9. 对命名管道的读写严格遵循先进先出的规则（对）
10. 信号既可以发给前台进程也可以发给后台进程（对）
11. 可以用 signal()向指定的进程发信号（错）
12. 无法用信号实现进程间的同步（错）
13. 消息队列可以按照消息类型读取消息（对）
14. 消息队列的读写只能采用阻塞的方式（错）
15. 共享内存是一种最为高效的进程间通信方式（对）

三、简答题(30 分)

1. 请描述进程和程序的区别 (6 分)

1.进程是动态的，程序是静态的。进程是运行起来的程序。
2.进程有生命周期，而程序是指令的集合，不存在生命周期的概念。
3.程序与进程并不是一一对应的，一个进程的运行必然有一个与之对应的程序，而一个程序如果没有执行就没有与之对应的进程，也有可能有多个进程与之对应。
4.进程具有并发性，而程序没有

2. 指出静态库和共享库的区别(使用方法，对程序的影响) (8 分)

	静态库	共享库
加载	编译过程中被加载到可执行文件	可执行程序运行时载入内存
依赖性	静态存在，没有依赖性	有依赖性，没有找到库就不能运行
体积	大	小
使用方法	包含对应头文件	需手动加载

3. 写出设置信号处理函数和用户定义的信号处理函数的原型 (6 分)

```
void  (*signal (int signum, void(* handler) (int) ) ) (int) ;
typedef void(*sig_t)(int);
sig_t signal(int signum, sig_t handler);
void handler(int signum);
```

4. 程序代码如下，请按执行顺序写出输出结果 (6 分)

```
int main()
{   pid_t   pid1,pid2;

    if((pid1=fork()) = = 0)
{
     sleep(3);
     printf("info1 from child process_1\n");
     exit(0);
     printf("info2 from child process_1\n");
}
else
{
   if((pid2=fork()) = = 0)
   {
        sleep(1);
        printf("info1 from child process_2\n");
        exit(0);
   }
   else
   {
        wait(NULL);
        wait(NULL);
        printf("info1 from parent process\n");
        printf("info2 from parent process");
        _exit(0);
   }
}
```

```
info 1 from child process_2
info 1 from child process_1
info 1 from parent process
info 2 from parent process
```

5. 列出任意四种进程间通信的方式(4 分)

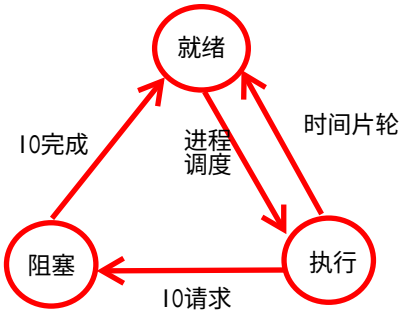
管道，信号，消息队列，共享内存

四、问答题(25 分)

1. 指出创建守护进程的步骤(10 分)

1.调用fork产生一个子进程，同时父进程退出，所有后续工作都在子进程中完成。
2.脱离控制终端，创建新会话：setsid();
3.禁止进程重新打开控制终端
4.改变当前工作目录
5.重设文件权限
6.关闭所有文件描述符

2. 请画出 Linux 中进程的状态切换图(6 分)



3. 编写程序实现如下功能(9 分)：

reader.c　　从 argv[1]所指定的文件中读取内容，依次写到管道 /home/linux/myfifo 中

writer.c　　从管道/home/linux/myfifo 中读取内容，写到 argv[1]所指定的文件中并保存

代码中可省略头文件，/home/linux/myfifo 无需创建

```
#define FIFO_PATH "myfifo"
int main(int argc, char*argv[])
{   int file_fd, fifo_fd, num_r;
char readbuf[255] = {0};
if(mkfifo(FIFO_PATH, 0666) < 0 && errno != EEXIST)
{
perror("mkfifo failed");
exit(0);
}
else
{
printf("create fifo sucess\n");
}
file_fd = open(argv[1], O_RDWR);
fifo_fd = open(FIFO_PATH, O_RDWR|O_CREAT, 0666 );
if(file_fd == -1)
{
perror("open failed");
exit(0);
}
if(file_fd > 0)
{
while(num_r = read(file_fd, readbuf, 255))
{
if(num_r > 0)
write(fifo_fd, readbuf, 255);
else
printf("read failed\n");
}
}
close(file_fd);
close(fifo_fd);
return 0;
}
```

```
#define FIFO_PATH "myfifo"
int main(int argc, char*argv[])
{
int file_fd, fifo_fd, num_r;
char readbuf[255] = {0};
if(mkfifo(FIFO_PATH, 0666) < 0 && errno != EEXIST)
perror("mkfifo failed");
exit(0);
}
else
{
printf("create fifo sucess\n");
}
file_fd = open(argv[1], O_RDWR|O_CREAT, 0666);
fifo_fd = open(FIFO_PATH, O_RDWR|O_CREAT, 0666 );
if(file_fd == -1)
{
perror("open failed");
exit(0);
}
if(file_fd > 0)
{
while(num_r = read(fifo_fd, readbuf, 255))
{
if(num_r > 0)
{
write(file_fd, readbuf, 255);
printf("%s\n", readbuf);
}else
printf("read failed\n");
}
}
close(file_fd);
close(fifo_fd);
return 0;
}
```