1 -守护进程

1. 守护进程的特点
    - 后台服务进程
    - 独立于控制终端
    - 周期性执行某任务
    - 不受用户登录注销影响
    - 一般采用以d结尾的名字(服务)
2. 进程组
    - 进程的组长?
    - 组里边的第一进程
    - 进程组的ID == 进程组的组长的ID
    - 进程组组长的选则
    - 进程中的第一个进程
    - 进程组ID的设定
    - 进程组的id就是组长的进程ID
3. 会话 - 多个进程组
    - 创建一个会话注意事项:
    - 不能是进程组长
    - 创建会话的进程成为新进程组的组长
    - 有些linux版本需要root权限执行此操作(ubuntu不需要)
    - 创建出的新会话会丢弃原有的控制终端
    - 一般步骤:先fork, 父亲死, 儿子执行创建会话操作(setsid)
    - 获取进程所属的会话ID
    - pid_t getsid(pid_t pid);
    - 创建一个会话
    - pid_t setsid(void);
4. 创建守护进程模型
    - fork子进程,父进程退出
        - 必须
    - 子进程创建新会话
        - 必须
        - setsid();
    - 改变当前工作目录chdir
        - 插了一个U盘,a.out, 在U盘目录中启动a.out
        - a.out启动过程中,U盘拔掉了
        - 不是 必须的。
    - 重设文件掩码
        - 子进程会继承父进程的掩码
        - 增加子进程程序操作的灵活性
        - umask(0);

        - 不是必须的
    - 关闭文件描述符

      close(0);  close(1)  close(2)  释放资源  不是必须的

        - 执行核心工作
            - 必须的

5. 练习: 写一个守护进程, 每隔2s获取一次系统时间, 将这个时间写入到磁盘文件.
    - 创建守护进程
    - 需要一个定时器,2s触发一次
        - setttimer
        - sleep
    - 信号捕捉

```
//
// Created by bruce on 18-5-19.
//

#include <signal.h>
#include <unistd.h>
#include <cstdlib>
#include <sys/stat.h>
#include <time.h>
#include <fcntl.h>
#include <cstring>
#include <sys/time.h>

void dowork(int no)
{
    //得到当前系统事假
    time_t curtime;
    time(&curtime);
    char* pt = ctime(&curtime);
    int fd  = open("./temp+++.txt",O_CREAT|O_WRONLY|O_APPEND,0664);
    write(fd,pt,strlen(pt)+1);
    close(fd);
}
int main()
{
    pid_t pid = fork();
    if(pid>0)
    {
        exit(1);
    }
    else if(pid==0)
    {
        setsid();
        //改变当前进程工作目录
        chdir("~");
        //重设文件掩码
        umask(0);
```

```
        //关闭文件描述符
        close(STDIN_FILENO);
        close(STDOUT_FILENO);
        close(STDERR_FILENO);
        //执行核心操作
        //注册信号捕捉
        struct sigaction act;
        act.sa_flags = 0;
        act.sa_handler = dowork;
        sigemptyset(&act.sa_mask);
        sigaction(SIGALRM,&act,NULL);
        //创建定时器
        struct itimerval val;
        val.it_value.tv_sec=2;
        val.it_value.tv_usec = 0;
        val.it_interval.tv_usec = 0;
        val.it_interval.tv_sec = 1;
        setitimer(ITIMER_REAL,&val,NULL);
        //保证一直处于运行状态
        while(1);
    }
    return 0;
}
```