进程控制：进程创建/进程终止/进程等待/程序替换

进程创建：

pid_t fork(void)

流程：clone()1、创建pcb；2、复制信息；3、内存数据发生改变的时候为子进程重新开辟空间，拷贝数据（写时拷贝技术）

pid_t vfork(void)------创建一个子进程，父子进程共用同一个虚拟地址空间

共用同一块虚拟地址空间，使用同一个栈，会造成调用栈混乱

因此父进程调用vfork创建子进程，会被阻塞，直到子进程exit()退出/进行了程序替换，重新创建了自己的虚拟地址空间之后，父进程才会vfork返回进行运行

进程终止：

进程退出的场景：

正常退出，结果符合预期

正常退出，结果不符合预期

异常退出，结果不能作为判断基准

进程的返回值，只用了一个字节来保存

在linux系统中erron.h这个头文件中有两个全局变量：

1、int errno；用于保存每次系统调用的错误编号

2、char *sys_errlist[] ： 系统调用的错误原因描述

3、perror(char* info):打印上一次系统调用的错误原因在info之后

4、strerror(int errno)：根据给定的erron来获取错误信息

进程退出方法：

1、main函数中的return；

2、void _exit(int status):系统调用接口，2号手册；

3、void exit(int status):库函数3号手册

进程等待：等待子进程退出，获取子进程返回值，允许操作系统释放子进程资源；避免产生僵尸进程

如何等待：
pid_t wait(int* status)；-----阻塞函数
等待任意一个子程序退出，通过status获取返回值，释放子进程资源，最终返回退出的子进程的pid
pid_ t waitpid(pid_t pid, int *status, int options)；
pid>0:则等待指定的子进程；pid=-1:则等待任意一个子进程
options=0：则默认阻塞等待子进程退出；options=WNOHANG则waitpid为非阻塞，若没有子进程已经退出，则立即返回
waitpid返回值为0则表示当前没有子进程退出，-1则出错
wait/waitpid不是子进程退出的时候才回收，而是只要有已经退出的子进程，就都可以回收
阻塞：为了完成一个功能发起调用，当时当前若不具备完成功能的条件；则一直挂起等待，直到条件满足完成功能后返回
非阻塞：为了完成一个功能发起调用，当时当前若不具备完成功能的条件；则立即报错返回

返回值的获取：
只有进程正常退出的时候，获取进程的返回值，才有意义
判断进程是否正常退出：通过status数据的低7位是否为0，若为0，这进程是正常退出
获取返回值：返回值存储在status的低16位中的高8位
status&0x7f：获取程序异常退出信号值
(status >>8)& 0xff：获取程序的退出返回值
WIFEXITED(status)：判断进程是否正常退出，正常则退出则返回真
WEXITSTATUS(status)：获取退出子进程的返回值

程序替换：替换一个程序进程正在运行的程序
在一个进程中，可以先将另一段程序加载到内存中，将自己的页表映射到新的程序位置，初始化pcd中的虚拟地址空间中的代码和数据段信息；这时候这个pcd（当前进程）则运行另一段程序

int execve(char *filename,char *argv[],char *env[]);
filename:新的程序代码文件名称（带有路径的文件名）
argv：新程序的运行参数信息
env：新程序的环境变量信息
返回值：失败返回-1，成功则进程已经运行新的程序了

int execl(const char *path, const char *arg, ...);
path：带有路径的新程序名称；
arg...：新程序的运行参数
int execlp(const char *file, const char *arg, ...);
file：只需要一个文件名即可（会去PATH环境变量指定的路径下去找相应的程序）
int execle(const char *path, char *arg, ...,char * env[]);
env：为新程序自定义环境变量

有没有p的区别：程序名称是否需要带路径
有没有e的区别：是否自定义环境变量

int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);

l和v的区别：程序运行参数的赋予方式不同

自主实现一个minishell，完成shell的基本功能
fflush(stdout); 手动刷新缓冲区
fgets(buf,1024,stdin)；从标准输入读取数据
buf[strlen(buf)-1]='\0';将最后的换行符修改为字符串结束标志
execvp(argv[0], argv); 子进程进行程序替换

waitpid(pid, NULL, 0);阻塞等待子进程命令执行完毕

不能直接对shell进行程序替换，因为替换后就没有shell