1.Fork()创建子进程

头文件： #include <sys/types.h>

#include <unistd.h>









pid\_t fork(void)；//pid\_t为int类型，进行了重载

pid\_t getpid();// 获取当前进程的 pid 值。

pid\_t getppid(); //获取当前进程的父进程 pid 值。

1.1fork()用于创建一个进程，所创建的进程复制父进程的代码段/数据段/BSS段/堆/栈等所有用户空间信息；在内核中操作系统重新为其申请了一个PCB，并使用父进程的PCB进行初始化；子进程复制并执行与父进程相同的代码，但不会执行代码中的fork(),避免无线生成子进程。

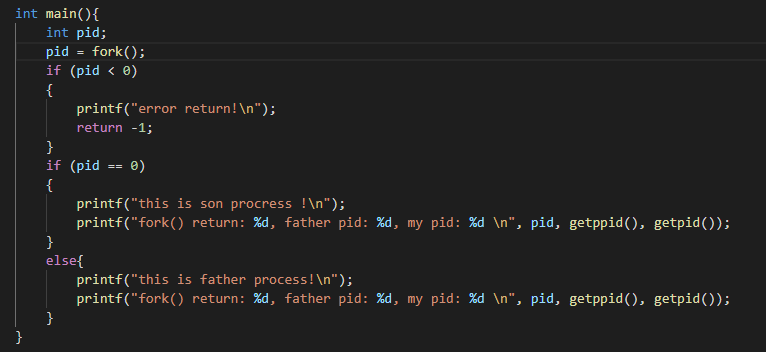
Fork()函数被调用一次，却能够返回两次，它可能有三种不同的返回值：

1.在父进程中，fork返回新创建子进程的进程ID；

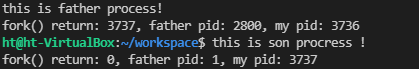
2.在子进程中，fork返回0；

3.如果出现错误，fork返回一个负值；

简单例子：

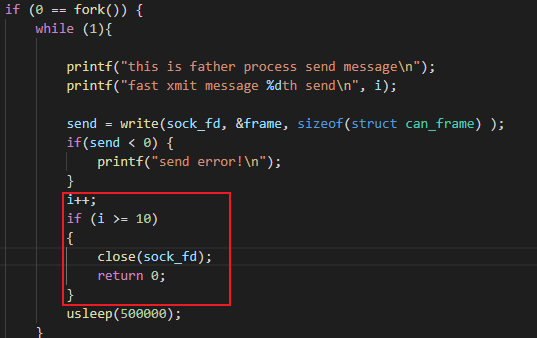


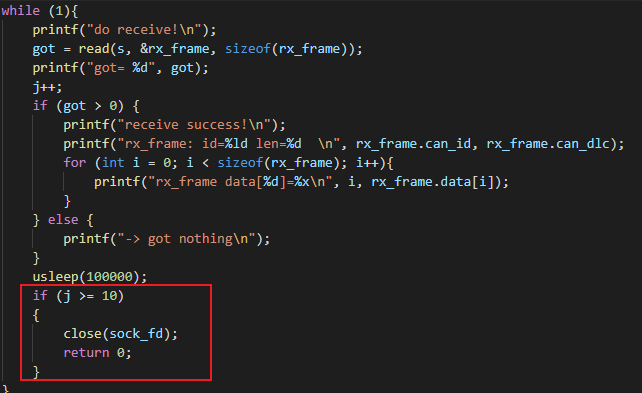
执行结果：



这里子进程显示父进程id为1，因为父进程执行完就被清除了，但进程不允许没有父进程，所以将子进程的“父进程”设为进程1.进程1是不会被清除的。父进程的pid与主进程保持一致，pid只会在子进程生成时变化。

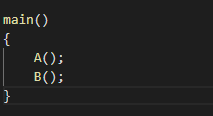
使用多进程发送时须在子父进程中都单独添加退出条件，单独退出一个对另一个无影响。



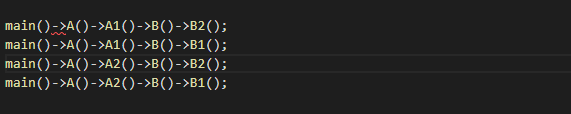


1.2，当主程序中有多个函数包含fork()函数时，情况会比较麻烦。

如



其中1为fork()生成的父进程，2为子进程。下图为程序执行流程。



A,B函数在一个大函数中时，就会导致A，B以外的函数重复执行四次。重复执行的次数为（fork()数量 \* 2）

2.进行自发自收

2.1文件描述符：在进行linux下can通信时，会创建socketCan中的文件描述符



正常情况下，sock\_fd的值为3，因为系统创建了值为0，1，2的文件描述符：

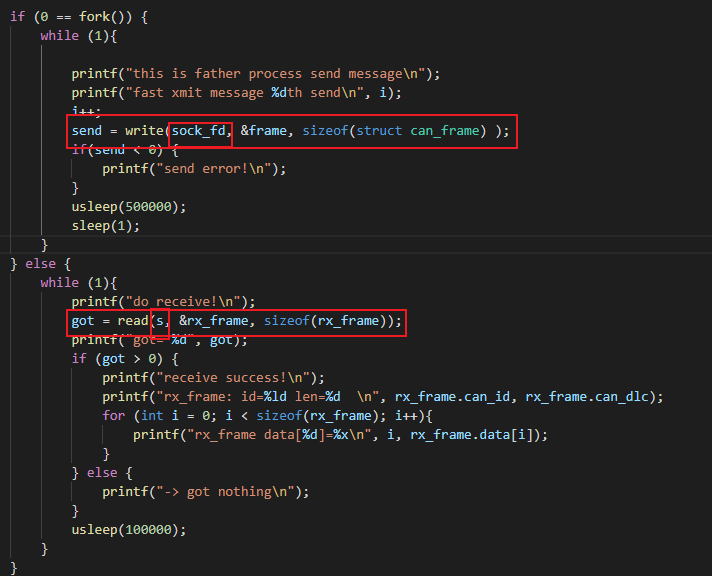
0 stdin，对应宏为：STDIN\_FILENO，函数 scanf() 使用的是标准输入；

1 stdout，对应宏为：STDOUT\_FILENO， 函数 printf() 使用的是标准输出；

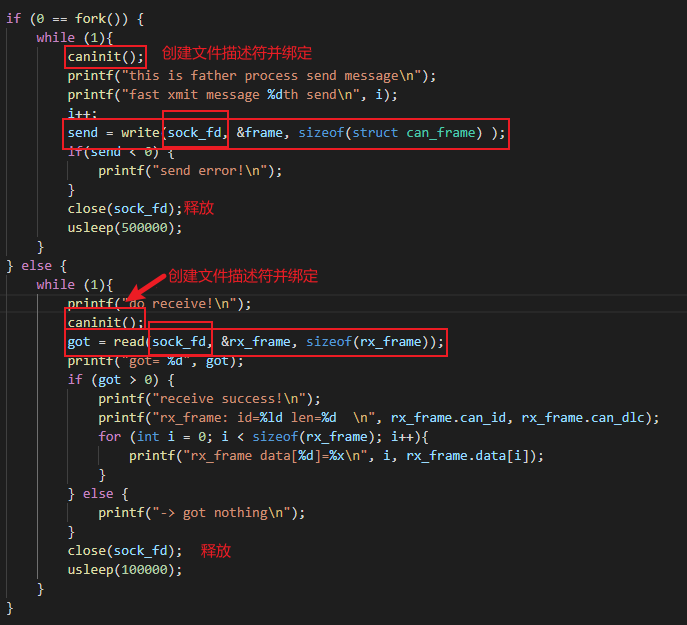
2 stderror，对应的宏为：STDERR\_NO， 表示标准出错处理；

所以用户创建的第一个文件描述符值为3；在未close当前文件描述符，继续创建其他文件描述符则会在此值上增加。

在单个程序中进行收发时，收发两个操作都会占用一个文件描述符。



1.要实现同时收发需使用两个文件描述符，一个值为3，另一个为4.

2.只创建一个文件描述符实现功能

在两个进程中单独创建文件描述符，若无释放步骤，则每次创建的文件描述符会保留并且新建的值会加1。

使用select()函数进行单个程序完成收发

头文件：#include <sys/select.h>

相关函数：

void FD\_CLR(int fd, fd\_set \*set); //将fd移出集合set

int FD\_ISSET(int fd, fd\_set \*set); //判断fd 是否在集合set中

void FD\_SET(int fd, fd\_set \*set); //添加fd到集合set中

void FD\_ZERO(fd\_set \*set); //清空集合set

程序中将同一文件描述符添加到存储读和写的集合中，也即函数中的readfds, writefds.

int select(int maxfdp, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*errorfds, struct timeval\*timeout);

select函数会监视传入的文件描述符集合中文件描述符的状态并有返回值。

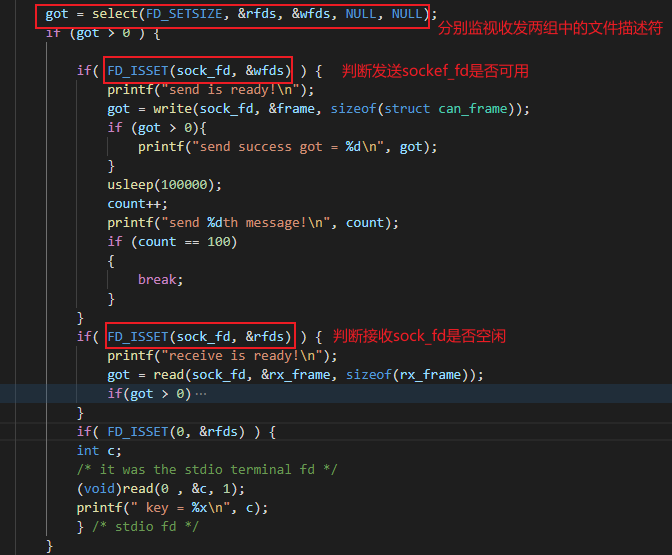
返回值：返回状态发生变化的描述符总数。  
负值：select错误

正值：某些文件可读写或出错

0：等待超时，没有可读写或错误的文件

当select返回值大于0时，则用 FD\_ISSET() 分别判断readfds和writefds中是否存在我们需要使用的文件描述符，可用则分别进行发送和接收。selset在判断时会将不满足可读/可写的文件描述符从对应的集合中去掉。

在下图操作结果中可见实现了同一程序同一文件描述符收发，但是收到的消息不来自于同程序中的发送部分。



运行结果：

