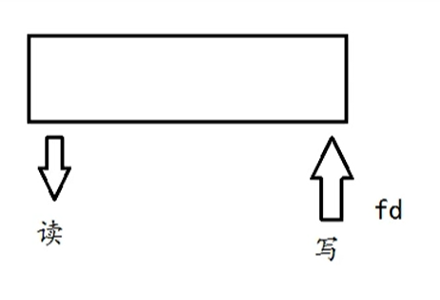
# 1、无名管道与有名管道

## 1.1 定义

（1）无名管道：用于有亲缘关系的进程之间通讯，比如：父子进程、兄弟进程。

（2）有名管道：用于非亲缘关系进程之间的通信。



## 1.2 有名管道

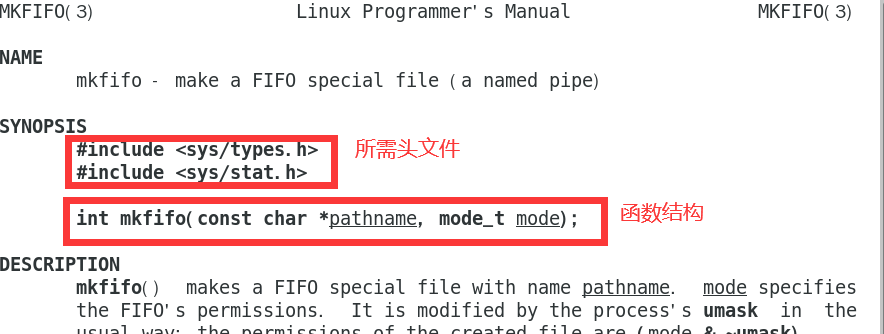
### （1）创建

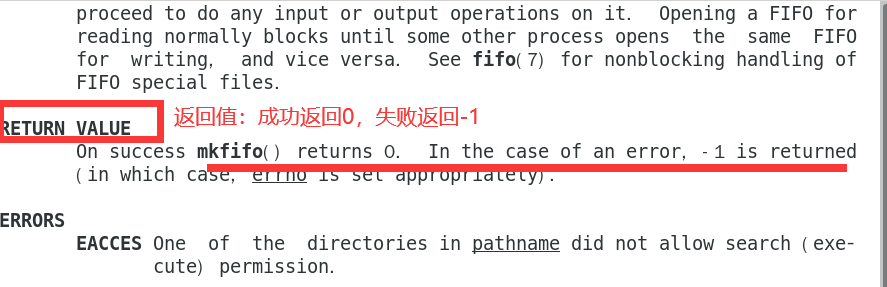
使用mkfifo()库函数进行创建，mkfifo同时也是一个用户指令。

*#man mkfifo //打开用户指令详情页*

*#man 3 mkfifo //打开Linux编程手册*

我们执行“*man 3 mkfifo*”查看函数信息，如下图所：





*int mkfifo(const char \*pathname,mode\_t mode);*

*↑ ↑ ↑*

*返回值（0、-1） 管道名 权限*

### （2）读管道

open() 方法用于打开一个文件，并且设置需要的打开选项，模式参数mode参数是可选的，默认为 0777。

open函数的结构如下：

*int open(const char \*pathname, int flags, mode\_t mode);*

***pathname*** *是要打开的文件的路径名*

***flags*** *用于指定打开文件的方式和行为*

***mode*** *是一个 mode\_t 类型的参数，用于指定创建新文件时的权限*

flags的参数选项：

*O\_CREAT:在文件打开过程中创建新文件*

*O\_RDONLY：以只读方式打开文件。*

*O\_WRONLY：以只写方式打开文件。*

*O\_RDWR：以读写方式打开文件。*

*O\_APPEND：在文件末尾追加数据，而不是覆盖现有内容。*

*O\_TRUNC：如果文件已经存在，将其截断为空文件。*

*O\_EXCL：与 O\_CREAT 一起使用时，如果文件已经存在，则 open() 调用将失败。*

*O\_SYNC：使文件写操作变为同步写入，即将数据立即写入磁盘。*

*O\_NONBLOCK：以非阻塞方式打开文件，即使无法立即进行读写操作也不会被阻塞。*

### （3）源码

1-1-fifo.c

*#include <stdio.h>*

*#include <sys/types.h>*

*#include <sys/stat.h>*

*#include <event.h>*

*#include <fcntl.h>*

*#include <unistd.h>*

*//当监听的事件满足条件时，会触发回调函数，通过回调函数读取数据*

*void fifo\_read(evutil\_socket\_t fd, short events, void \*arg)*

*{*

*char buf[32] = {0};*

*int ret = read(fd, buf, sizeof(buf));*

*if(ret == -1)*

*{*

*perror("open fifo");*

*\_exit(1);*

*}*

*printf("从管道读取数据：%s\n", buf);*

*}*

*int main()*

*{*

*//创建管道*

*int ret = mkfifo("fifo.tmp", 00700);*

*if(ret == -1)*

*{*

*perror("mkfifo");*

*\_exit(1);*

*}*

*//读取管道*

*int fd = open("fifo.tmp", O\_RDONLY);*

*if(fd == -1)*

*{*

*perror("open fifo");*

*\_exit(1);*

*}*

*//创建事件*

*struct event ev;*

*//初始化事件集合*

*event\_init();*

*//初始化事件（把fd和事件ev绑定）*

*//&ev:事件*

*//fd:关联的文件描述符*

*//EV\_READ:事件类型 执行过后事件集合就会移除掉*

*//EV\_PERSIST:事件类型 执行过后事件集合不会移除*

*//fifo\_read:需要实现的回调函数*

*//NULL:回调函数的参数*

*event\_set(&ev, fd, EV\_READ | EV\_PERSIST, fifo\_read, NULL);*

*//把事件添加到集合中*

*event\_add(&ev, NULL);*

*//开始监听*

*event\_dispatch();//死循环，如果集合中没有事件可以监听，则返回*

*return 0;*

*}*

1-2-ReadFifo.c

*#include <stdio.h>*

*#include <sys/types.h>*

*#include <sys/stat.h>*

*#include <event.h>*

*#include <fcntl.h>*

*#include <unistd.h>*

*#include <string.h>*

*int main()*

*{*

*int fd = open("fifo.tmp", O\_WRONLY);*

*if(ret == -1)*

*{*

*perror("open fifo");*

*\_exit(1);*

*}*

*char buf[32] = {0};*

*while(1)*

*{*

*scanf("%s", buf);*

*ret = write(fd, buf, strlen(buf));*

*if(ret == -1)*

*{*

*perror("write fifo");*

*\_exit(1);*

*}*

*if(!strcmp(buf, ":wq"))*

*{*

*break;*

*}*

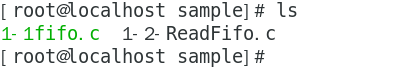
*memset(buf, 0, sizeof(buf));*

*}*

*}*

（4）编译

ls查看当前目录文件：

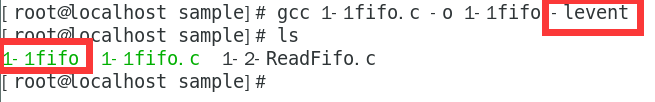


因为使用了event事件的结构体，需要引入外部库，编译后生成” 1-1fifo“文件

编译命令如下：

*#gcc 1-1fifo.c -o 1-1fifo -levent*

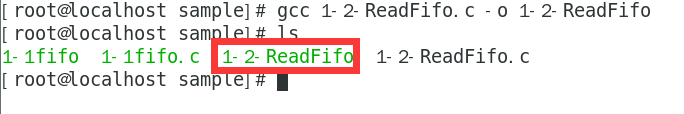
编译完成如图所示：



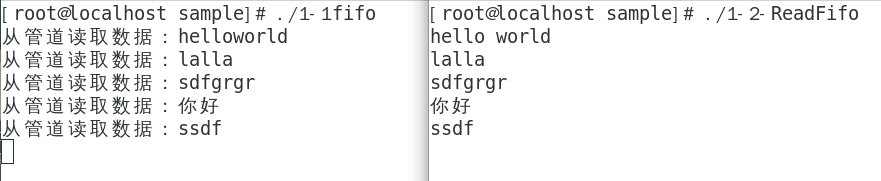
编译1-2-ReadFifo.c:

*#gcc 1-2-ReadFifo.c -o 1-2-ReadFifo*

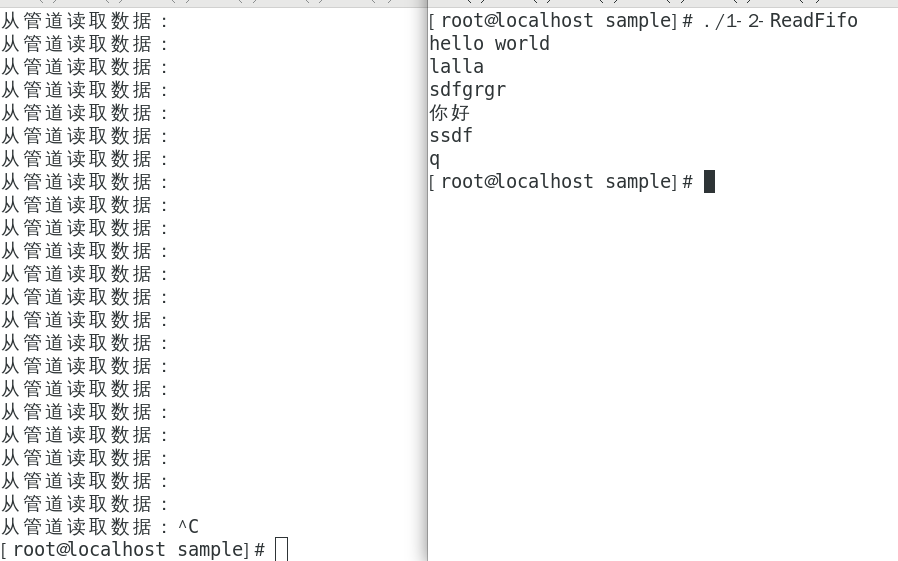
编译完成如图所示：



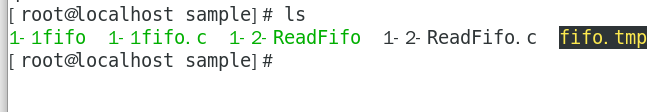
### （4）运行



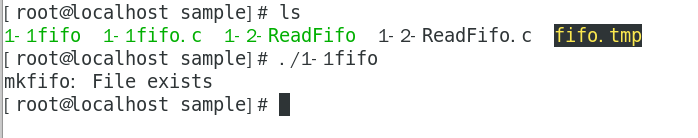
在1-2-ReadFifo界面输入q退出写入，管道读取不到数据ctrl+c异常退出：



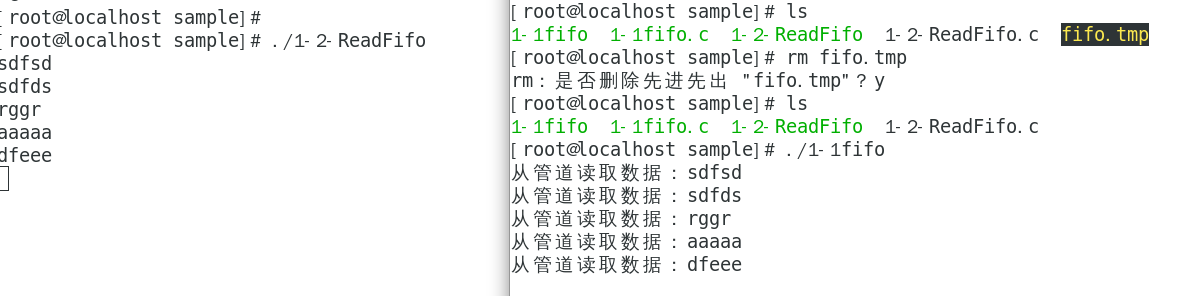
此时的目录已经生成管道文件“fifo.tmp”：



如果需要再次执行，需要删除“fifo.tmp”文件，如果不删除执行会出现以下情况：



删除后再执行：



# 2 信号事件

## 2.1 Linux中信号的产生

（1）当用户按下某些按键时，产生信号.

（2）硬件异常产生信号：除数为0，无效的存储访问等等.这些情况通常由硬件检测到，将其通知内核然后内核产生适当的信号通知进程。例如，内核对正访问一个无效存储区的进程产生一个SIGSEGV信号

（3）进程用kill函数 将信号发送给另一个进程

（4）用户可用kill命令将信号发送给其他进程.

## 2.2 信号类型

常用的有以下几种信号：

（1）SIGHUP：从终端上发出的结束信号

（2）SIGINT：来自键盘的中断信号( ctrl + c )

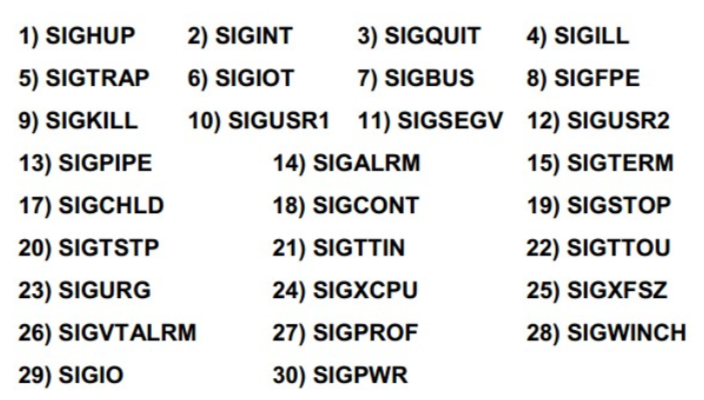
（3）SIGKILL：该信号结束接收信号的进程

（4）SIGTERM：kill命令发出的信号

（5）SIGCHLD：标识子进程停止或结束的信号

（6）SIGSTOP：来自键盘( ctrl + z )或调试程序的停止执行信号

完整的信号如图所示：



## 2.3 实例

### （1）源码

以下源码中使用到了SIGINT，运行起来后，ctrl+c两次发出信号后结束运行。

*#include <stdio.h>*

*#include <signal.h>*

*#include <event.h>*

*#include <sys/types.h>*

*#include <sys/stat.h>*

*int signal\_count=0;*

*void signal\_handler(evutil\_socket\_t fd, short events, void \*arg)*

*{*

*struct event \*ev = (struct event\*)arg;*

*printf("收到信号 %d\n", fd);*

*signal\_count++;*

*if(signal\_count >= 2)*

*{*

*//把事件从集合中删除*

*event\_del(ev);*

*}*

*}*

*int main()*

*{*

*//创建事件集合*

*struct event\_base \*base = event\_base\_new();*

*//创建事件集合*

*struct event ev;*

*//把事件和信号绑定*

*int ret = event\_assign(&ev, base, SIGINT, EV\_SIGNAL | EV\_PERSIST, signal\_handler, &ev);*

*if(ret == -1)*

*{*

*perror("event\_assign");*

*\_exit(1);*

*}*

*//把事件添加到集合中*

*event\_add(&ev, NULL);*

*//监听自己的集合使用event\_base\_dispatch，监听全局变量中的集合使用event\_dispatch*

*event\_base\_dispatch(base);*

*//自己创建的集合需要释放*

*event\_base\_free(base);*

*return 0;*

*}*

（2）编译

因为使用了event事件结构体等，所以编译时引入event库，使用以下命令编译：

*# gcc 2-signal.c -o 2-signal -levent*

编译完成后结果如下图：

