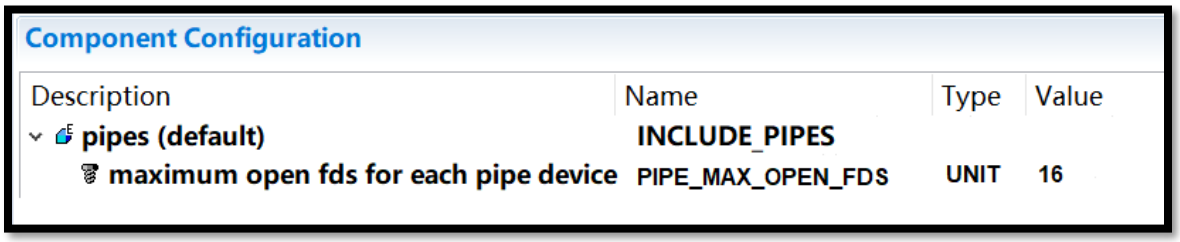
VxWorks 下的管道 Pipe 介绍

很多 OS 都会提供一种进程通信机制：管道。VxWorks 也提供了管道 - Pipe，而且可以用于任务间通信



之前提到过，多任务的通信机制可以分为两类：事件通知、数据传递。其中数据传递的主要手段是消息队列。而这个管道，就是在消息队列上面又封装了一层，以 IO 设备的形式来提供服务

```
STATUS pipeDevCreate
    (
    char  *name,
    size_t nMessages,
    size_t nBytes
    );
STATUS pipeDevDelete
    (
    char *name,
    BOOL  force
    );
```

因此，使用管道之前，需要先创建这个虚拟的 IO 设备；用完之后，可以删除这个设备；而具体的数据操作，就是调用基本 IO 的几个函数

```
#define  O_RDONLY         0
#define  O_WRONLY         1
#define  O_RDWR           2
#define _FNONBLOCK    0x4000
#define  O_NONBLOCK _FNONBLOCK
int open
    (
    char *name,
    int   flags,/* O_RDONLY / O_WRONLY / O_RDWR / O_NONBLOCK */
    int   mode  /* not used */
    );
int close (int fd);
int read  (int fd, char *buffer, int maxBytes);
int write (int fd, char *buffer, int nBytes);
int ioctl (int fd, int function,...);
```

在 Shell 里练一下

```
-> pipeDevCreate "/myPipe",10,100          创建
value = 0 = 0x0
-> devs                                    查看
drv name
  3 /myPipe
-> open "/myPipe",2                        打开
value = 18 = 0x12
-> write 18,"hello",5                      写入
value = 5 = 0x5
-> aaa=malloc(100)
New symbol "aaa"
aaa = 0x8fdbfe0:
-> bzero aaa,100
value = 0 = 0x0
-> read 18,aaa,100                         读出
value = 5 = 0x5
-> printf "aaa=%s\n",aaa
aaa=hello
value = 10 = 0xa
-> close 18                                关闭
value = 0 = 0x0
->
```

而 ioctl()主要支持这几个 option

```
#define FIONREAD     1  /* get num chars available to read */
#define FIOFLUSH     2  /* flush any chars in buffers */
#define FIONMSGS    17  /* return num msgs in pipe */
```

```
-> open "/myPipe",2                        打开
value = 18 = 0x12
-> write 18,"hello",5                      发送，长度5
value = 5 = 0x5
-> write 18,"helloworld",10                发送，长度10
value = 10 = 0xa
-> bbb=0
New symbol "bbb"
bbb = 0x8ff1fe0: value = 0 = 0x0
-> ioctl 18,1,&bbb                         FIONREAD
value = 0 = 0x0
-> bbb
bbb = 0x8ff1fe0: value = 5 = 0x5           第一条消息长度5
-> ioctl 18,17,&bbb                        FIONMSGS
value = 0 = 0x0
-> bbb
bbb = 0x8ff1fe0: value = 2 = 0x2           共2条消息
-> ioctl 18,2                              FIOFLUSH
value = 0 = 0x0
-> ioctl 18,17,&bbb
value = 0 = 0x0
-> bbb
bbb = 0x8ff1fe0: value = 0 = 0x0
```

除了这些基本功能，管道还支持 Select 操作

```
int select
    (
    int     width,
    fd_set *pReadFds,
    fd_set *pWriteFds,
    fd_set *pExcFds,
    struct timeval *pTimeOut
    );
```

支持 POSIX 的 stat 信息获取

```
02: #include <stdio.h>
03: #include <ioLib.h>
04: #include <pipeDrv.h>
05: #include <sys/stat.h>
06:
07: #define PIPE_NAME "/myPipe"
08:
09: void testPipe()
10:     {
11:     int fd;
12:     struct stat myStat;
13:
14:     pipeDevCreate(PIPE_NAME, 10, 100);
15:     fd = open(PIPE_NAME, O_RDWR, 0);
16:     write(fd, "hello", 5);
17:
18:     ioctl(fd, FIOFSTATGET, &myStat);
19:     printf("number of open: %d\n", myStat.st_nlink);
20:     printf("pipe size: %d\n", myStat.st_size);
21:
22:     close(fd);
23:     pipeDevDelete(PIPE_NAME, 1);
24:     }
-> testPipe
number of open: 1
pipe size: 1000
value = 0 = 0x0
->
```

并可以通过 POSIX 的 fcntl()来调整 O_NONBLOCK

```
#define F_GETFL      3  /* Get file flags */
#define F_SETFL      4  /* Set file flags */
int fcntl
    (
    int fd,
    int command,
    ...
    );
-> open "/myPipe",2    O_RDWR
value = 18 = 0x12
-> bbb=0
bbb = 0x8ff1fe0: value = 0 = 0x0
-> fcntl 18,3          F_GETFL -> O_RDWR
value = 2 = 0x2
-> fcntl 18,4,0x4000   F_SETFL -> O_NONBLOCK
value = 0 = 0x0
-> fcntl 18,3
value = 16386 = 0x4002
->                     F_GETFL -> O_RDWR|O_NONBLOCK
```

那管道与消息队列有什么区别呢

- 消息队列
  - ✓ 收发时都可以指定超时
  - ✓ 消息有两种优先级
  - ✓ 由内核提供，效率更高
  - ✓ 直接使用 show()命令就可以查看
- 管道
  - ✓ 基于消息队列封装，效率略差一点点
  - ✓ 使用基本 IO 就可以操作
  - ✓ 可用于 IO 重定向 - ioTaskStdSet()
  - ✓ 可用于 select()