VxWorks I/O Interface

VxWorks I/O Interface

Introduction

Standard I/O

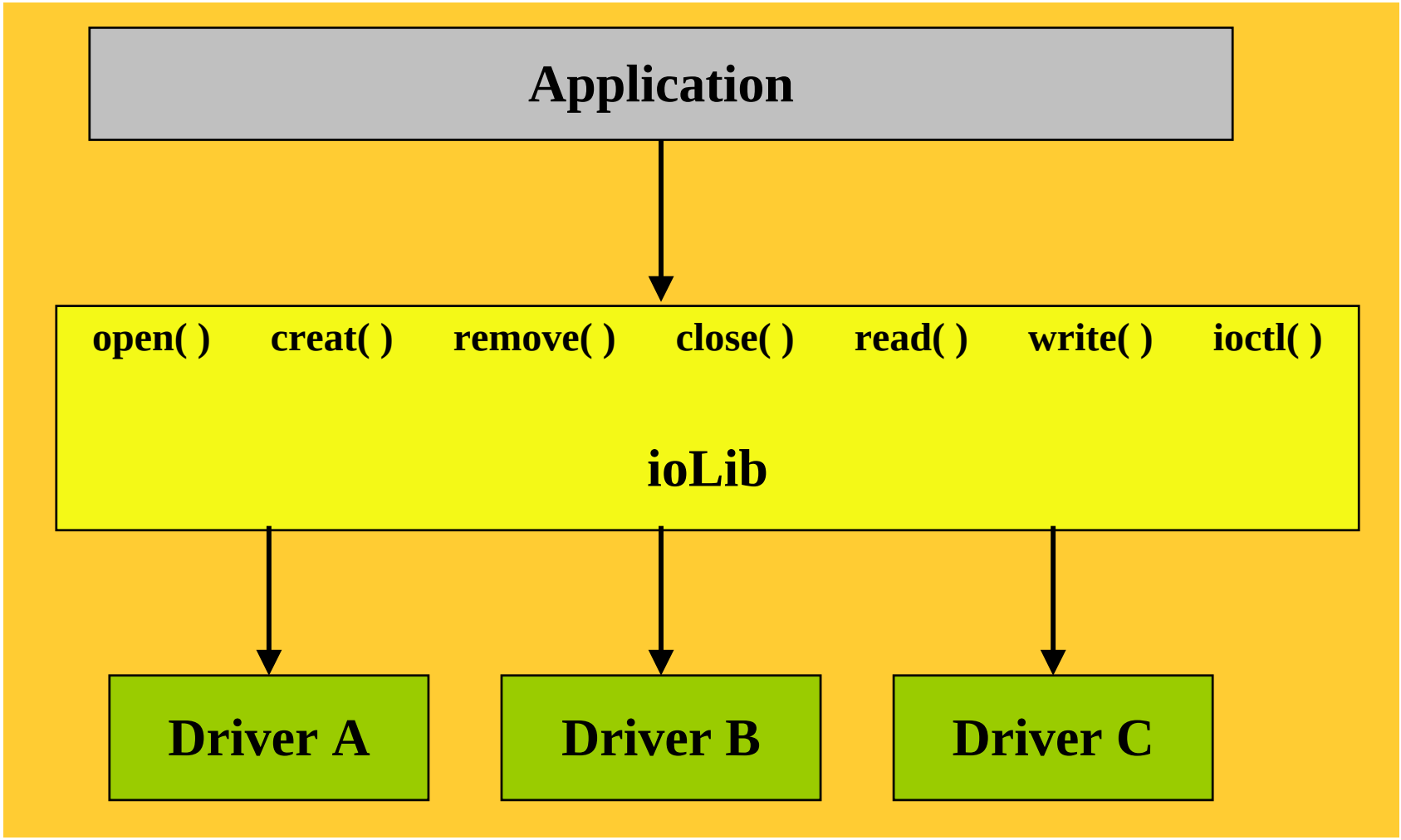
Support Routines

Supporting select( )

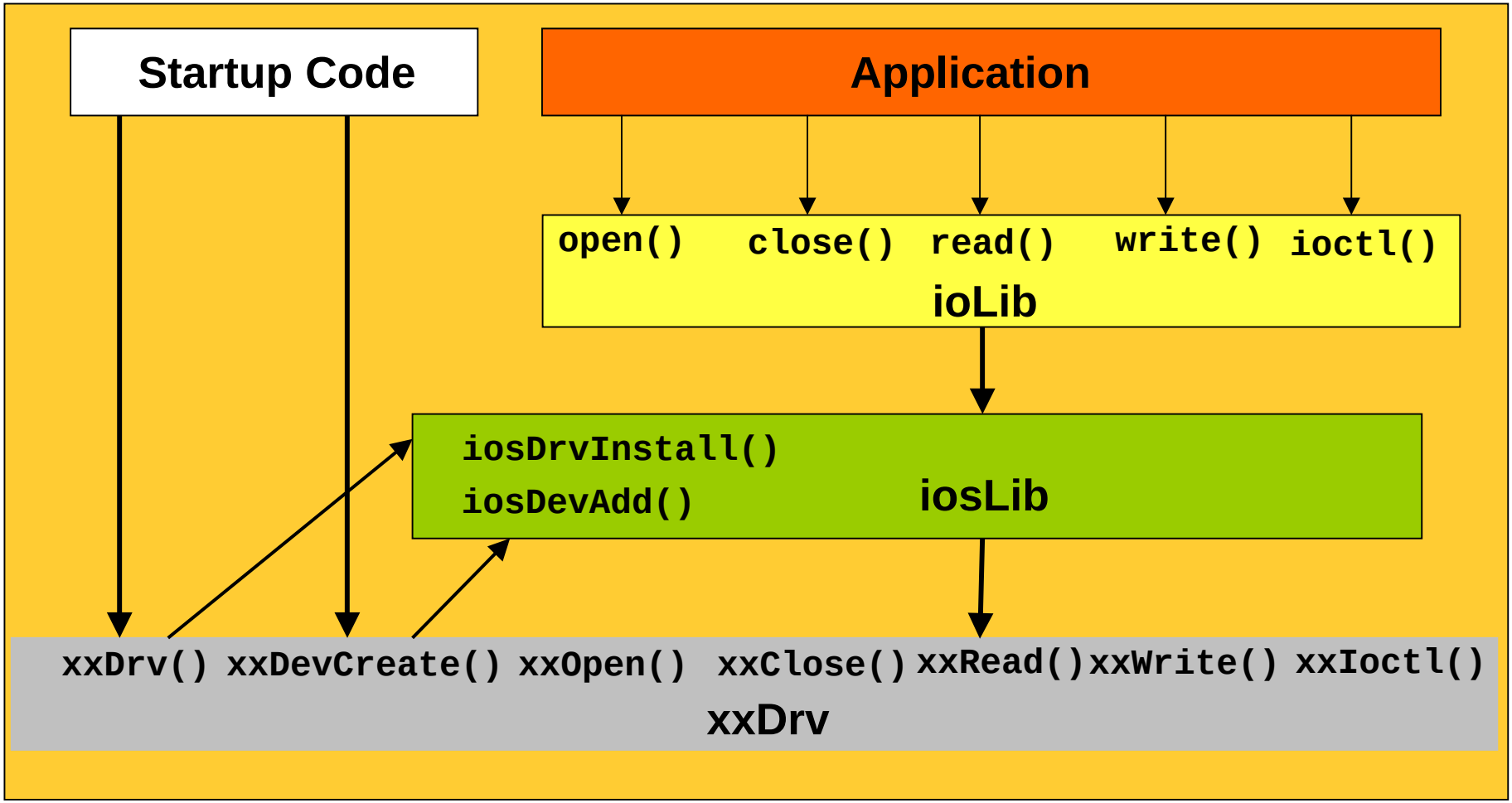
VxWorks I/O System Interface

- VxWorks 在 I/O 管理方面提供了一套统一的, 灵活对接多种类型设备的接口
- 7个标准 I/O 接口:
  - creat(filename, flags, mode)
  - remove(filename)
  - open(filename, flags, mode)
  - close(fd)
  - read(fd, &buf, nBytes)
  - write(fd, &buf, nBytes)
  - ioctl(fd, command, args)

I/O System Overview



I/O System Overview



Should this Driver use Standard I/O?

- 优势:
  - 代码简洁
  - 标准的用户接口
  - 支持 **select( )** 和 **IO** 重定向
  - 驱动的需求清晰
- 劣势:
  - 为了配合**IO**的架构, 额外增加了很多代码, 间接调用驱动函数的效率比直接调用要低
  - 对于一些简单的设备, 反而增加了中间环境, 让健壮性变差. 例如: 定时器或指示灯
  - 有些驱动不适合使用标准的 read( ) 或 write( ) 接口

Drivers Using the Standard Interface

- 文件系统
  - 最典型的是块设备文件系统
  - 非块设备文件系统, 例如 **tapeFs**
- ttyDrv
  - 为串口驱动提供用户接口
- 其他顺序存取型设备
  - **Graphics tablets**
  - **etc.**

VxWorks I/O Interface

Introduction

Standard I/O

Support Routines

Supporting select( )

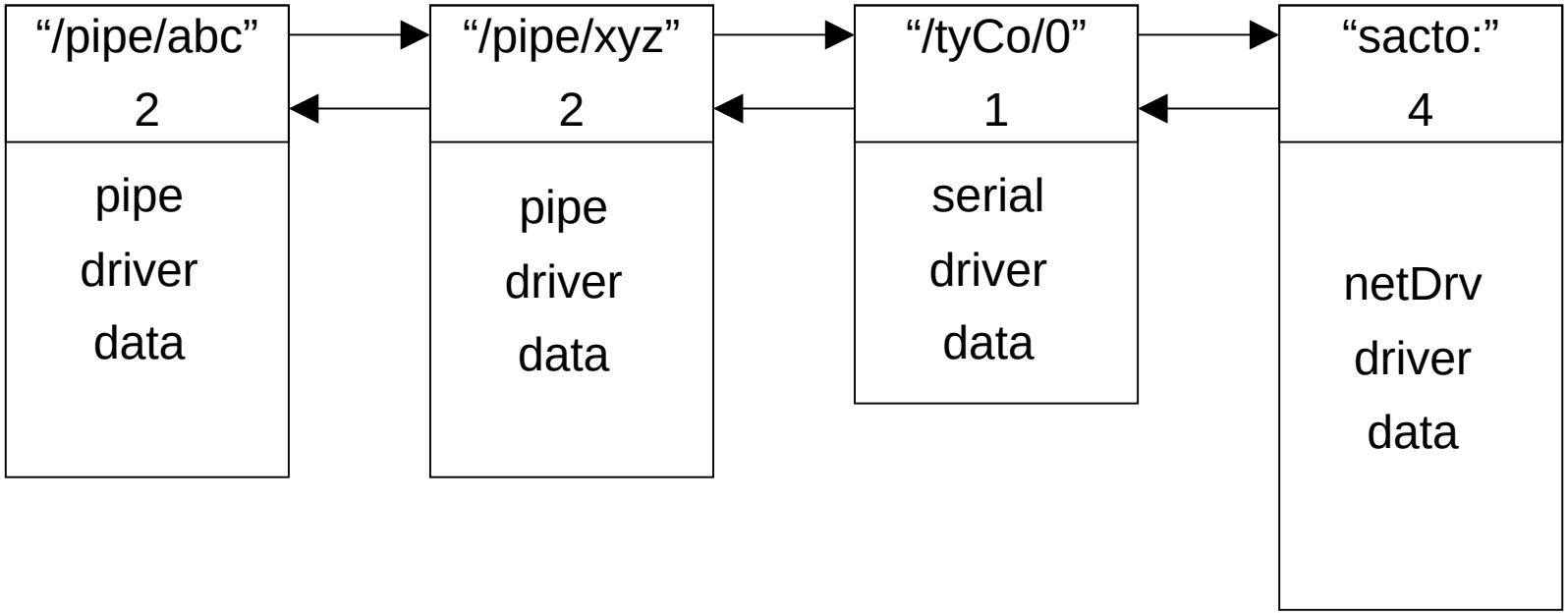
Driver Table

	creat	remove	open	close	read	write	ioctl
0							
⋮							
7	myOpen	NULL	myOpen	NULL	myRead	myWrite	myIoctl
8							

Driver Number

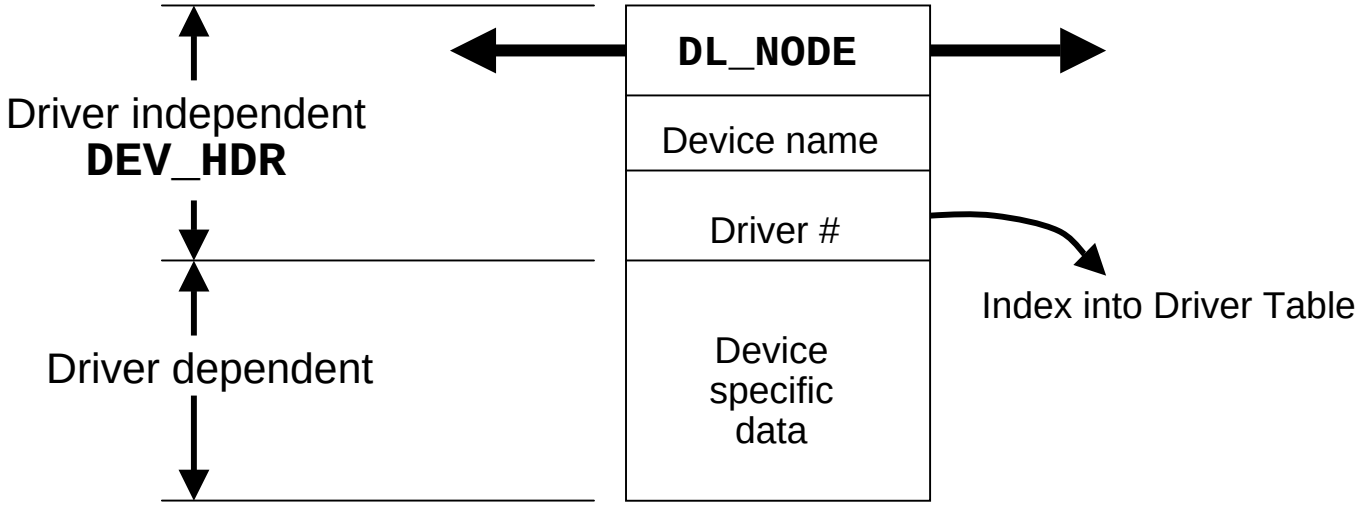
- 使用 iosDrvShow( ) 显示驱动列表

Device List



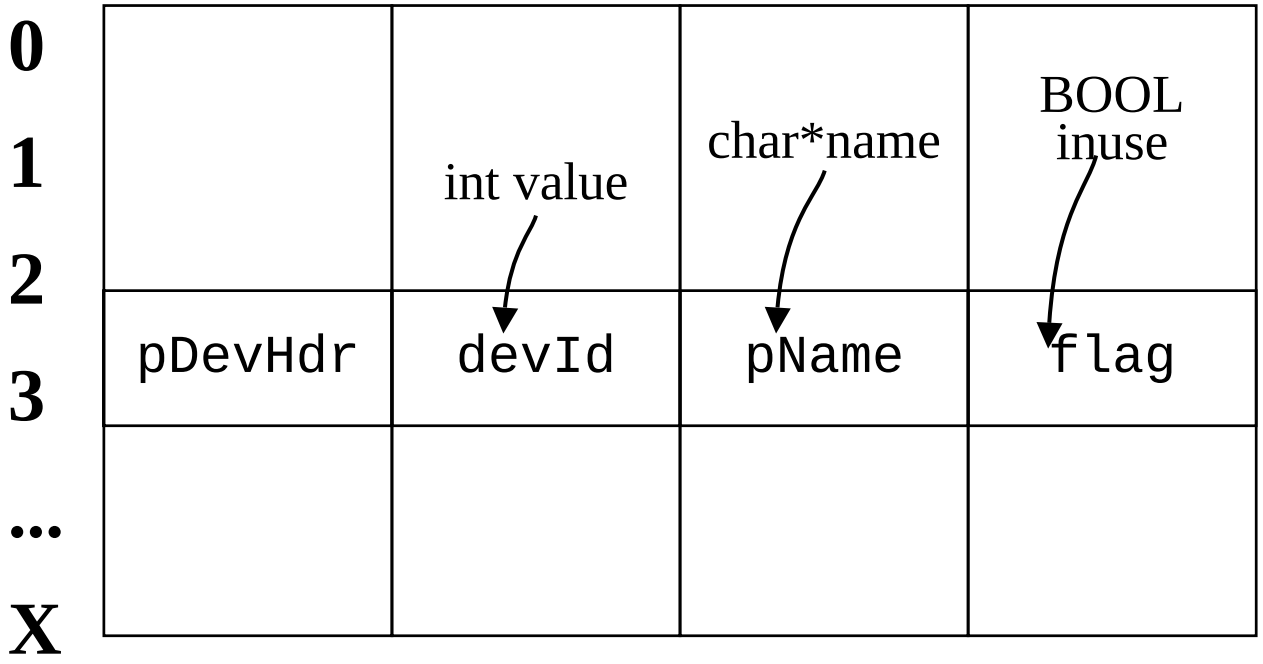
- 使用 devs( ) 查看设备链表

The Device Descriptor Structure



- 每个设备有一个设备描述符链接在设备列表中
- 设备描述符的第一个结构体是 *DEV_HDR*
- 驱动相关部分包含了特定设备的信息

File Descriptor Table



- 使用 iosFdShow( ) 查看当前可用的文件描述符(例如已经打开的文件), 文件名以及驱动编号

Device Driver Functions

- xxDrv( )
  - 把驱动安装到驱动列表
    - iosDrvInstall( )
  - 执行驱动的安装
  - 只能调用一次
- xxDevCreate( )
  - 把设备安装到系统设备列表
    - iosDevAdd( )
  - 执行设备的安装
  - 对每个设备只能调用一次
- 7个接口函数
  - 可以选择其中几个实现

Initializing A Driver

STATUS xxDrv (args...)

- **xx** 是驱动名称
  - 例如 pipeDrv **or** ttyDrv
- 参数是具体驱动相关的
- 调用 iosDrvInstall( ) 把驱动增加到 **I/O** 系统驱动列表
- 只能调用一次
- 通常在用户的初始化代码中调用
  - usrAppInit( ) **in** usrAppInit.c

Adding Driver into Driver Table

iosDrvInstall (xxCreat, xxRemove, xxOpen,
xxClose, xxRead, xxWrite, xxIoctl)

- 所有接口函数都是可选的 – 如果某个接口函数没有实现, 用 **NULL**
- 如果调用成功返回驱动列表表项的编号, 否则返回**ERROR.** 一般在返回后将编号保存在全局变量xxDrvNum中
- 例如:

```
fooDrvNum = iosDrvInstall (fooOpen, NULL, fooOpen,
                           fooClose, fooRead, fooWrite, fooIoctl)
```

Example xxDrv( ) Routine

```
1  LOCAL int fooDrvNum = 0;
2
3  STATUS fooDrv (void)
4       {
5       /* If driver already installed, just return */
6       if (fooDrvNum > 0)
7               return (OK);
8
9       /* driver initialization, if any, here */
10
11      /* add driver to driver table */
12      if ((fooDrvNum = iosDrvInstall (fooOpen, NULL,
13              fooOpen, fooClose, fooRead, fooWrite,
14                      fooIoctl)) == ERROR)
15              return (ERROR);
16      return (OK);
17      }
```

Creating an Instance of Device

STATUS xxDevCreate (devName, args...)

- args 是跟具体驱动相关的
- 通过调用 iosDevAdd( ) 把指定的设备名字加入到系统设备列表
- 执行设备相关的初始化
- 典型操作: 为设备描述符结构体分配内存
- 如果设备的驱动还没有安装 **(**xxDrvNum **< 1),** 返回**ERROR,** 并置 **errno** 为 *S_ioLib_NO_DRIVER*

Adding to the Device List

STATUS iosDevAdd (pDevHdr, devName, drvNum)

pDevHdr	指向数据结构 *DEV_HDR*
devName	设备的名字
drvNum	从 iosDrvInstall( ) 返回的驱动编号

- 把设备描述符增加到设备列表
- 用设备名字和驱动编号来初始化结构体 *DEV_HDR*
- 如果设备名字 devName 已经存在, 返回**ERROR**
- 后续对设备名字的访问将自动关联到相应的驱动和硬件设备

Example Device Creation

```
1  LOCAL int fooDrvNum = 0;    /* initialized in fooDrv() */
                ...
2  STATUS fooDevCreate (char * devName)
3       {
4       FOO_DEV *       pFooDev;

5       /* Check if driver is installed */
6          if (fooDrvNum < 1)
7              {
8              errno = S_ioLib_NO_DRIVER;
9              return (ERROR);
10             }
11 /* Allocate and zero device structure */
12    if ((pFooDev = (FOO_DEV *) malloc (sizeof
        (FOO_DEV))) == NULL)
13      return (ERROR);
14    bzero (pFooDev, sizeof (FOO_DEV));
                ...
```

Example (Continued)

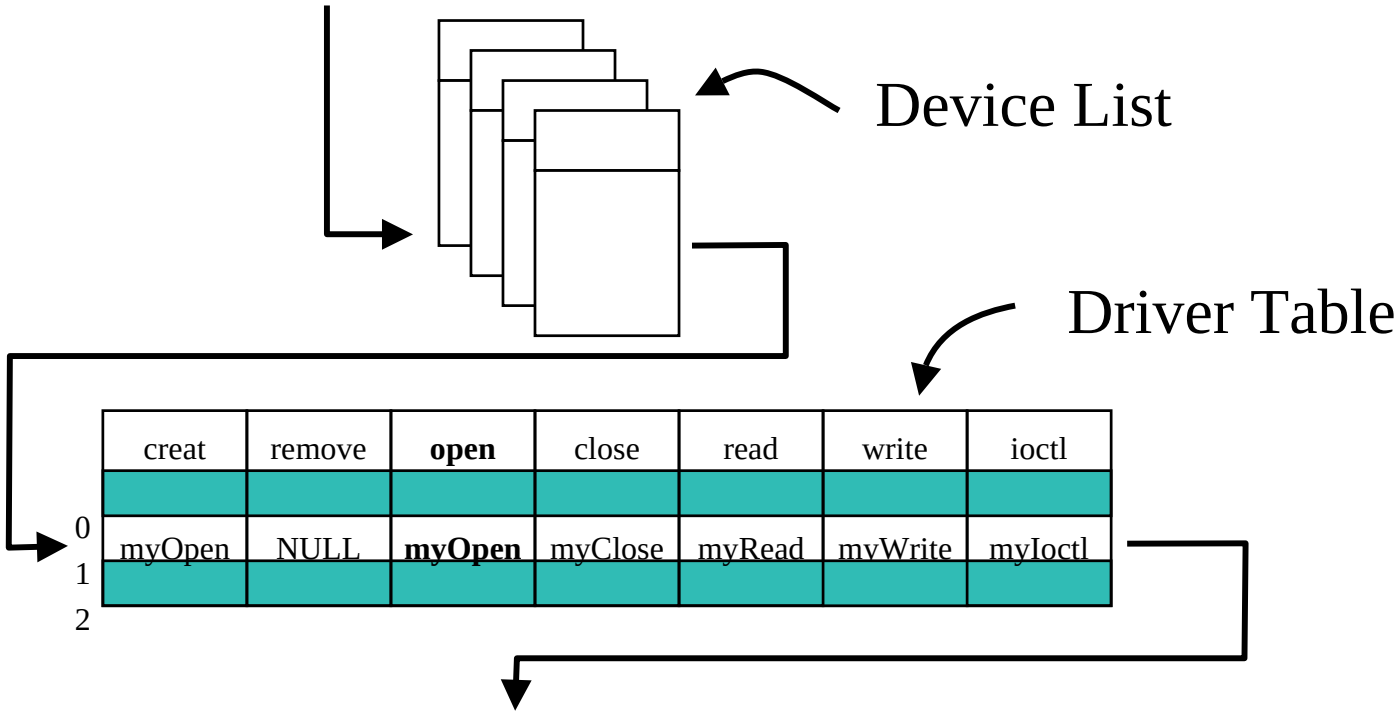
```
15   /* Init device specific portion of FOO_DEV here */

16   /* Perform any device-specific initialization here */

17      /* Add device to the device list */
18        if (iosDevAdd (&pFooDev->devHdr, devName,
        fooDrvNum) == ERROR)
19                      {
20                      free ((char *) pFooDev);
21                      return (ERROR);
22                      }
23         return (OK);
24         }
```

Opening A Device

```
fd = open (“/myDevice”, O_READ, 0)
```




```
myOpen (pDevHdr, ‘/0’, O_READ, 0)
```

What Happens on an open( )?

1. **I/O** 系统查询设备列表, 对设备名称做最长匹配
2. **I/O** 系统为本次**open**分配一个文件描述符表项
   - 如果没有文件描述符可用, open( ) 返回 *ERROR* 并设置 errno 为 *S_iosLib_TOO_MANY_OPEN_FILES*
3. **I/O** 系统调用具体驱动的 xxOpen 函数
   - 如果 xxOpen( ) 返回 *ERROR*, open( ) 也返回 *ERROR*
   - 如果 xxOpen( ) 返回的不是 *ERROR*, **I/O** 系统把 pDevHdr, 返回值 **(**设备**ID),** 和名字 写入文件描述符表, 并置该表项 **inuse** 标志为**TRUE(**标记已使用**)**
4. open( ) 返回文件描述符
   - 这个文件描述符是文件描述符表项的序号

Driver Open Routine

int xxOpen (pDevHdr, name, flags, mode)

pDevHdr	设备描述符指针
name	设备名字指针
flags	open( ) 的flag: *O_RDONLY*, 等
mode	open( ) 的权限

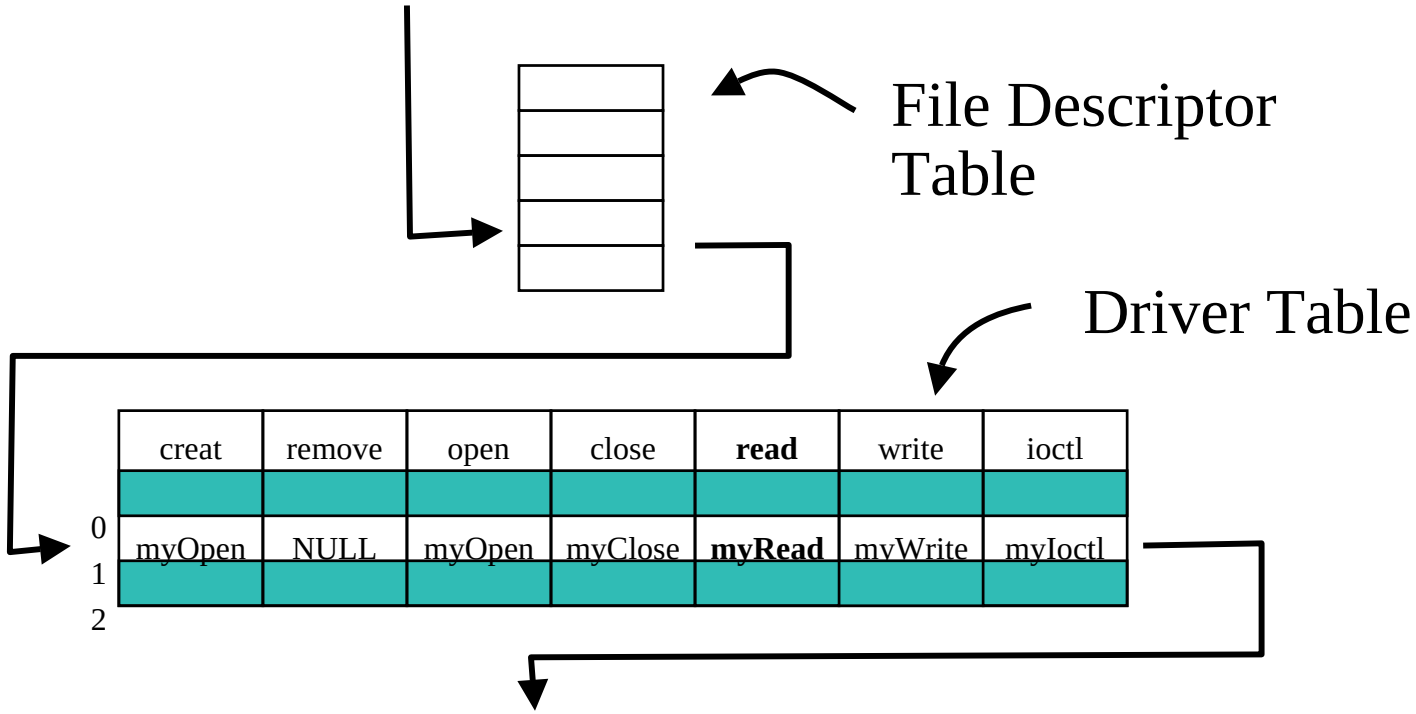
- 返回 *ERROR* 或设备**ID**, 通常是 pDevHdr
- 设备**ID**的指针由具体的设备驱动维护, 只有驱动才能识别
- 初始化相应的通道或文件
- 通常情况下, 多次尝试打开同一个文件将返回失败

myOpen( ) Example

```
1  int myOpen (DEV_HDR * pDevHdr, char * name, int flags,
               int mode)
2       {
3       /* File-descriptor-specific initialization goes here, */
4       /* though it is very rare to have any */
5       ...
6       return ((int) pDevHdr);
7       }
```

Reading from A Device

```
read (fd, &buf, nBytes)
```




```
myRead (devId, &buf, nBytes)
```

- 与写操作的流程相同

Read/Write Routine

int xxWrite(deviceId, pBuf, nBytes)

int xxRead(deviceId, pBuf, nBytes)

deviceId	从 xxOpen( ) 的返回值
pBuf	指向读或写的**buffer**首地址
nBytes	读或写的字节数

Ioctl( ) Routine

int xxIoctl (deviceId, cmd, arg)

deviceId	从 xxOpen( ) 的返回值
cmd	操作命令字
arg	命令需要的参数, 通常用作指针

- 返回值是命令指定的, **ERROR**保留, 用作返回失败
- 未知命令应该返回识别并且 errno 应该设置为 *S_ioLib_UNKNOWN_REQUEST*

Ioctl( ) Arguments

- 参数定义:
  - 整数值
  - 输入参数的地址
  - 输出参数的地址
  - 未使用
- 当参数用作地址的时候, 通常是一个数据结构的指针

Ioctl( ) Example

```
1  LOCAL FOO_STATE fooState;
2  LOCAL int fooSpeed;
3  LOCAL SEM_ID fooSem;
4       ...
5  int fooIoctl (int fooDevId, int cmd, int arg)
6       {
7       int status;
8
9       switch (cmd)
10              {
11              case FOO_SPEED_SET:
12                      fooSpeed = arg;
13                      status = OK;
14                      break;
15              case FOO_SPEED_GET:
16                      if (arg)
17                              *(int *)arg = fooSpeed;
18                      status = fooSpeed;
19                      break;
```

Ioctl( ) Example (Continued)

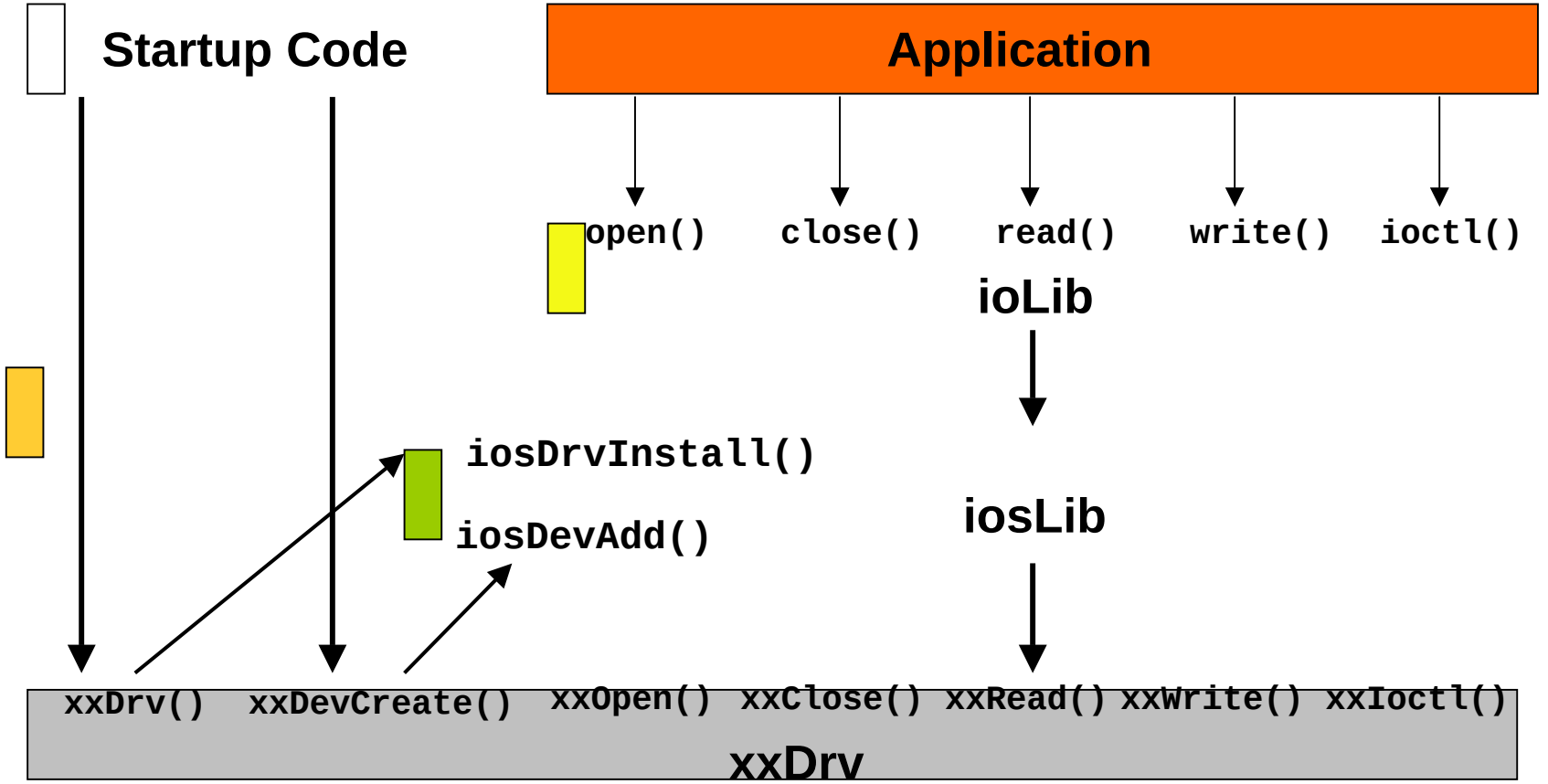
```
20              case FOO_STATE_SET:
21                      semTake (fooSem, WAIT_FOREVER);
22                      fooState = * (FOO_STATE *) arg;
23                      semGive (fooSem);
24                      status = OK;
25                      break;
26              case FOO_STATE_GET:
27                      semTake (fooSem, WAIT_FOREVER);
28                      * (FOO_STATE *) arg = fooState;
29                      semGive (fooSem);
30                      status = OK;
31                      break;
32              default:
33                      errno = S_ioLib_UNKNOWN_REQUEST;
34                      status = ERROR;
35              }
36      return (status);
37      }
```

The xxClose( ) Routine

STATUS xxClose (devId)

- 如果 devId 是 **xx** 驱动合法的数据结构, 执行必要的 **reset** 操作 **(恢复到未open的状态)**
- 如果 xxOpen( ) 申请了内存, 在此处释放
- 有些驱动没有 **Close** 函数
  - 这些驱动的 **close( )** 仅仅是释放了文件描述符

I/O System Summary



VxWorks I/O Interface

Introduction

Standard I/O

Support Routines

Supporting select( )

Removing Devices

- 在 **VxWorks** 中, 一般不删设备
  - 一旦设备创建了, 在系统的运行周期内一直存在
- 为调试目的做删除的除外
- 没有函数释放设备申请的相关资源
  - 驱动开发人员必须自己写这样的函数

Finding a Device Descriptor

DEV_HDR * iosDevFind (name, pNameTail)

name	设备名字
pNameTail	指向名字的未能匹配部分 **(char **)**

- 如果查找成功, 返回与设备名相关的数据结构 *DEV_HDR*, 如果没找到, 返回缺省设备, 如果没有缺省设备, 返回*NULL*
- 如果 **name** 与设备名字不能完全匹配, 本函数会将 **name** 中未匹配的部分作为输出参数赋给 pNameTail
- 可能在调试或者删除设备时会用到

The Default Directory

- 缺省目录时系统启动时通过 **ftp** 加载 **vxWorks image** 的主机目录
  - 在设备列表中未发现的名字, 使用本路径
  - 如果 iosDevFind( ) 接收到一个拼错的名字或者访问一个不存在的设备将会引起歧义
- 使用 ioLib 的库函数 ioDefPathGet( ) 和 ioDefPathSet( ) 获取和设置缺省路径
- 所有驱动删除函数都应该仔细检查名字, 以确保他们删除的设备或驱动是正确的
  - 检查数据结构 *DEV_HDR* 中的驱动序号是很容易实现的

Deleting a Device

void iosDevDelete (pDevHdr)

- 从设备列表中删除 pDevHdr 所描述的设备
- 过程与 iosDevAdd( ) 相反

- 避免在设备上做**open**操作

- 任何已经**open**的文件描述符都变为无效
  - 用户应该在删除之前关闭这些文件描述符

Example Remove Device Routine

```
1  #include "vxWorks.h"
2  #include "iosLib.h"
3
4  STATUS rmDev (char * devName)
5       {
6       DEV_HDR *       pDevHdr;
7       char *  pNameTail;
8
9       pDevHdr = iosDevFind (devName, &pNameTail);
10
11      if (pDevHdr == NULL || *pNameTail != '\0')
12         return (ERROR);
13
14      iosDevDelete (pDevHdr);
15      free(pDevHdr);
16      return (OK);
17      }
```

Removing a Driver

iosDrvRemove (drvNum, forceClose)

- 从驱动列表删除驱动, 并从设备列表删除所有使用了这个驱动的设备
- 如果 forceClose 为 *TRUE*, 关闭所有与这个驱动相关的文件描述符
- 如果 forceClose 为 *FALSE*, 任何打开的文件描述符都变为无效

File Descriptor to Device ID Conversion

int iosFdValue (fd)

- 返回 **Device ID (**驱动函数 xxOpen( ) 的返回值**) , fd** 与 **Device ID** 是一一映射的关系**. fd** 层通过这个映射屏蔽了具体设备的对外联系**,** 让不同种类的设备在应用上看起来是一样的
- 如果 **fd** 非法返回*ERROR*

VxWorks I/O Interface

Introduction

Standard I/O

Support Routines

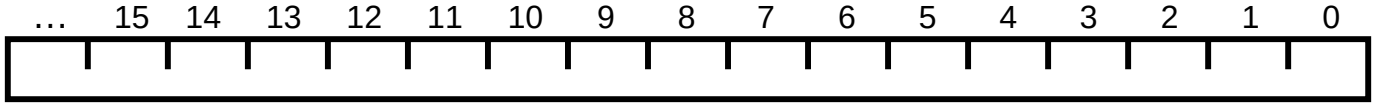
Supporting select( )

Supporting select( )

- 调用 select( ) 可能把任务阻塞在文件描述符上
- 任务一直阻塞到数据可读, 或设备可写
- 允许设置超时时间
- 支持 select( ) 是可选项
- 一般用于 **I/O** 系统的字符设备

select( ) User Macros

- **Select** 使用的数据结构 struct fd_set 如下:



- 每一位代表一个被检测的文件描述符
- 系统定义了一些宏帮忙操作 struct fd_set:

FD_SET(fd, &fdset)	在 **fdset** 中将 **fd** 对应的位置**1**
FD_CLR(fd, &fdset)	清空 **fdset** 中 **fd** 指定的位
FD_ZERO(&fdset)	清空 **fdset** 中所有的位域
FD_ISSET(fd, &fdset)	测试是否 **fdset** 中的 **fd** 位被置**1**

Using select( )

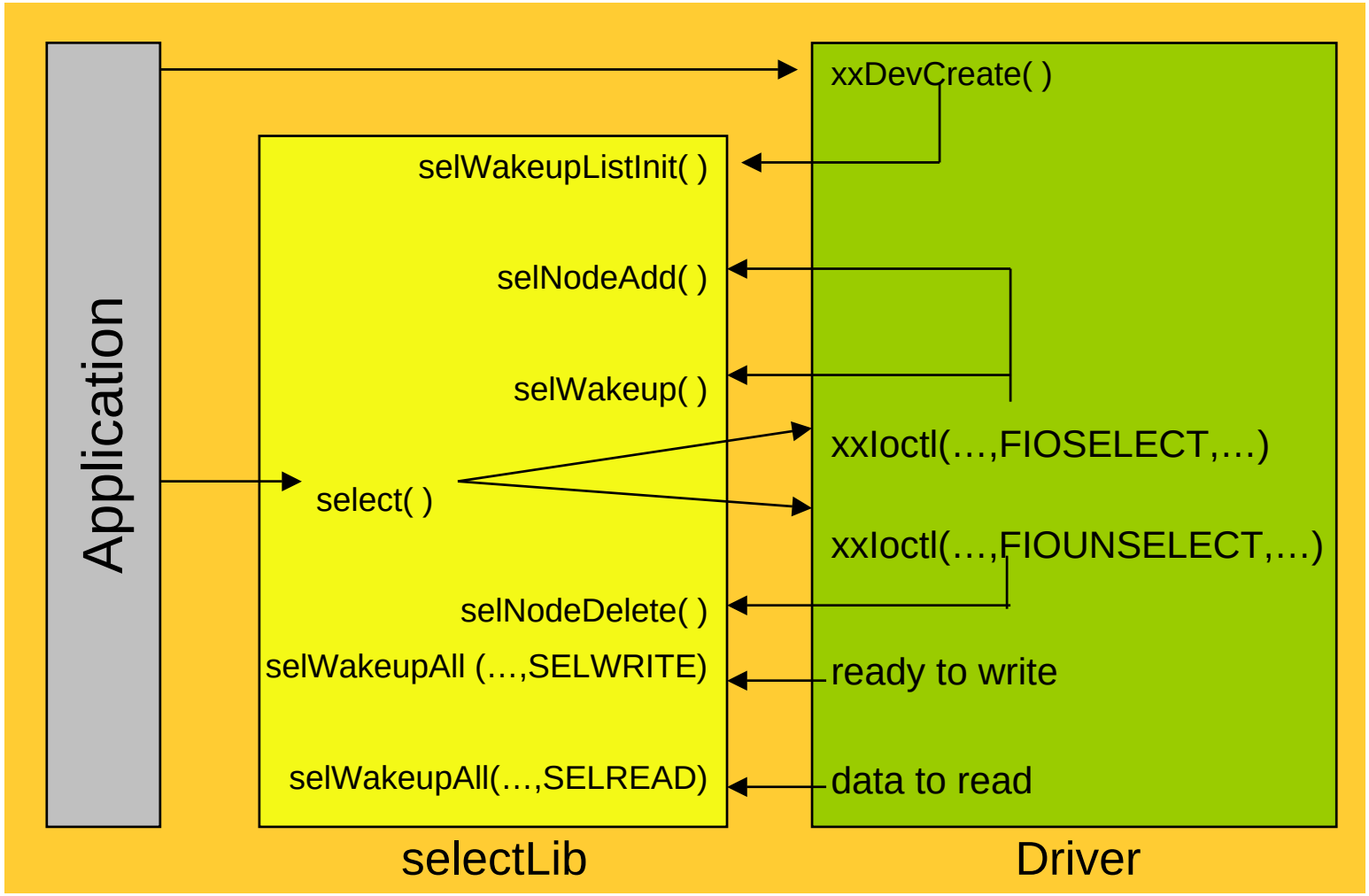
int select (width, pReadFds, pWriteFds, pExceptFds, pTimeOut)

width	struct fd_set 中被检测的位的数量
pReadFds	struct fd_set 结构体指针, 保存了要读的文件描述符
pWriteFds	struct fd_set 结构体指针, 保存了要写的文件描述符
pExceptFds	**UNIX** 兼容选项, 不处理
pTimeOut	struct timeval 结构体指针, *NULL* 表示永远等待

select( ) User Example

```
1  struct fd_set readFds;
2  int fds[NUM_FDS], width = 0;
        ...
3  FD_ZERO (&readFds);
4  for (i=0; i<NUM_FDS; i++)
5       {
6       FD_SET (fds[i], &readFds);
7       width = (fds[i] > width) ? fds[i] : width;
8       }
9  width++;
10 if (select(width, &readFds, NULL, NULL, NULL) ==  ERROR)
11      return (ERROR);
12 for (i=0; i<NUM_FDS; i++)
13      {
14      if (FD_ISSET (fds[i], &readFds))
15         {
16        /* do something now that fds[i] is ready */
17         }
18      }
```

Implementing select( )



Steps to Implement select( )

- 修改 xxDevCreate( ) 初始化 select 的唤醒列表
- 修改 xxIoctl( ) 支持 *FIOSELECT* 和 *FIOUNSELECT* 命令
- 修改 xxRead( ) 和 xxWrite( ), 当有数据可读或可写的时候调用 selWakeupAll( )
  - 可选地, 把 selWakeupAll( ) 放到设备地读或写地中断服务程序中

select Initialization

- 驱动必须声明一个 *SEL_WAKEUP_LIST* 结构体, 典型的在设备描述符结构体中
- xxDevCreate( ) 必须调用:

```
SEL_WAKEUP_LIST * pWakeupList;
selWakeupListInit (pWakeupList)
```

- 当设备可写的时候, 调用:

```
selWakeupAll (pWakeupList, SELWRITE);
```

- 当设备有数据可读的时候, 调用:

```
selWakeupAll (pWakeupList, SELREAD);
```

select Write( ) Example

```
/* Fill ring buffer until we have finished request or until the ring
    buffer is full */

    bytesWritten = rngBufPut (pRingDev->ringId, pBuf, nBytes);

/* If we managed to write anything, wake up pending read tasks */

    if (bytesWritten > 0)
        selWakeupAll (&pRingDev->selList, SELREAD);
```

select xxIoctl( )Example

```
case FIOSELECT:
   selNodeAdd (&pRingDev->selList, (SEL_WAKEUP_NODE *) arg);
   if ((selWakeupType ((SEL_WAKEUP_NODE *) arg) == SELREAD)
               && (rngNBytes (pRingDev->ringId) > 0))
        selWakeup ((SEL_WAKEUP_NODE *) arg);

   if ((selWakeupType ((SEL_WAKEUP_NODE *) arg) == SELWRITE)
               && (rngFreeBytes (pRingDev->ringId) > 0))
        selWakeup ((SEL_WAKEUP_NODE *) arg);
   break;
case FIOUNSELECT:
   selNodeDelete (&pRingDev->selList, (SEL_WAKEUP_NODE *) arg);
   break;
```

Summary

- xxDrv( ) – 每个驱动调用一次
  - 安装**7**个驱动函数
  - 调用 iosDrvInstall( )
  - 返回 *STATUS*
- xxDevCreate( ) – 每个设备调用一次
  - 分配并初始化设备描述的数据结构
  - 调用 iosDevAdd( )
  - 返回 *STATUS*

Summary (Continued)

- xxOpen( )
  - 初始化 **open** 相关的数据结构
  - 返回 *device ID* 或 *ERROR*
- xxClose( )
  - **Reset** 相关的数据结构
  - 释放 xxOpen( ) 时申请的内存
- xxRead( ) / xxWrite( )
- xxIoctl( )
  - 执行已经注册的命令
  - 不能识别的命令返回失败