

## PRA5 实验核心目标

实现一个支持**进程间传递整数**的内核字符设备驱动 `p5d`，包括：

- 写入：一个进程写入一个整数（`int`）
- 读取：另一个进程读取这个整数
- 状态：通过 `ioctl()` 查询设备是否已有待读取的数据

## 知识点按模块总结


### 1. 字符设备驱动架构（pseudo-device）

- 使用 OpenBSD 的字符设备框架（`cdev_decl`、`cdevsw` 表）
- 设备编号注册在 `sys/arch/amd64/amd64/conf.c` 中
- 使用 `sys/conf/files` 注册源文件，`conf/GENERIC` 注册设备类型

 目的：将自定义设备 `p5d` 插入 OpenBSD 内核设备管理系统中

### 2. 内核模块状态管理（softc 结构）

- `struct p5d_softc` 保存设备当前状态，包括：
  - 等待标志位 `SEND_WAITING`
  - 当前整数值 `sc_num`
  - 自旋锁/互斥锁 `sc_mtx`

 目的：设备状态隔离管理，避免使用全局变量，引入锁保护并发访问

### 3. 内核同步机制（读写阻塞与唤醒）


- `write()`：如果已有数据未被读取，返回 `EBUSY`
- `read()`：如果没有数据，会 **阻塞等待**，直到其他进程写入
- 使用 `msleep()` 和 `wakeup_one()` 实现 **阻塞/唤醒**机制

 关键点：

- `write()` 是“非阻塞”的，失败就立即返回 `EBUSY`
- `read()` 是“阻塞”的，必须等待有数据再返回

### 4. 内核-用户空间数据交互（uio + uiomove）

- 使用 `uiomove()` 在用户进程和内核之间拷贝 `int` 类型数据
- 若 `uio->uio_resid != sizeof(int)` 则返回 `EINVAL`

 目的：实现对用户空间进程的安全数据交互（不直接访问指针）

5. ioctl(2) 命令的定义与处理

- 自定义 ioctl 命令（使用 `_IOR(...)` 宏）查询“是否有数据等待”
- 定义一个头文件 `p5d.h`，结构体中仅一个布尔变量 `psp_is_num_waiting`
- 内核通过 `p5dioclt()` 返回状态值

🔍 补充说明：

- `_IOR` 表示从内核“读取”数据结构
- 使用 ioctl 的好处是可以灵活扩展，而不污染 read/write 的语义

6. 用户态测试程序（非必须但推荐）

- 实现 `xnum` 工具或 `testing_tools` 二进制程序
- 提供三种功能：
  - `-s <number>`：发送整数
  - `-r`：读取整数
  - `-t`：测试设备状态（是否有等待中的整数）

🔍 测试意义：

- 验证设备的同步性与边界条件处理（重复写入、空读、阻塞行为）

🌱 通用系统开发知识点（横向延伸）

技术点	实践体现
字符设备驱动注册	pseudo-device 机制、cdevsw 表、conf.c、conf.h
线程同步	mutex + msleep/wakeup_one
内核内存管理	<code>malloc(M_DEVBUF)</code> 为 softc 分配空间
用户态交互	<code>uiomove()</code> + <code>ioctl()</code>
系统调用接口	<code>open()</code> 、 <code>read()</code> 、 <code>write()</code> 、 <code>ioctl()</code>

✅ 实验通过的判定标准

- ✅ 编译成功，无内核错误
- ✅ 使用 `xnum` 或 `testing_tools` 可正确：
  - 写入整数（第一次成功，第二次失败）
  - 查询状态 (yes/no)
  - 正确阻塞读并在写入后恢复
- ✅ `/dev/p5d` 创建成功，权限正确（`chmod 0666`）

