文件描述符 – 学习Linux

原文链接

在Linux通用*I/O*模型中，*I/O*操作系列函数(系统调用)都是围绕一个叫做**文件描述符**的整数展开。 这不禁让人产生疑问：这个整数代表什么？ 一个数值代表一个文件吗？ 随便传一个整数进去调用可以吗？

图解

理解具体情况，需要了解由内核维护的*3*个数据结构：

- 进程级**文件描述符表**(*file descriptor table*)
- 系统级**打开文件表**(*open file table*)
- 文件系统**i-node表**(*i-node table*)

这*3*个数据结构之间的关系如*图-1*所示：

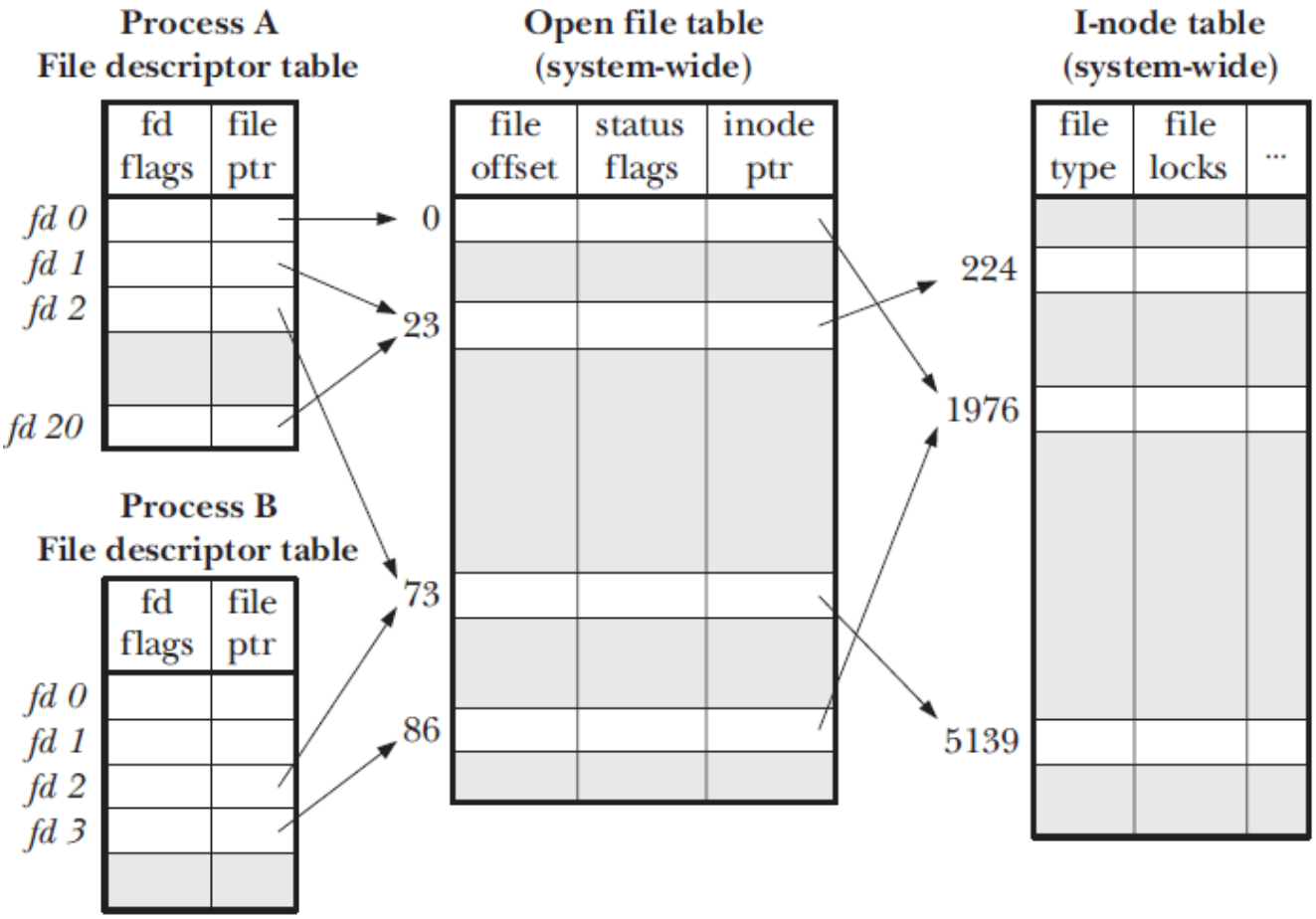

图-1：文件描述符、打开文件及*inode*关系

文件描述符表

内核为每个进程维护一个**文件描述符表**，该表每一条目都记录了单个文件描述符的相关信息，包括：

- **控制标志**(*flags*)，目前内核仅定义了一个，即`close-on-exec`
- **打开文件描述体指针**

打开文件表

内核对所有打开的文件维护一个系统级别的**打开文件描述表**(*open file description table*)，简称**打开文件表。** 表中条目称为**打开文件描述体**(*open file description*)，存储了与一个打开文件相关的全部信息，包括：

- **文件偏移量**(*file offset*)，调用*read()*和*write()*更新，调用*lseek()*直接修改
- **访问模式**，由*open()*调用设置，例如：只读、只写或读写等
- **i-node 对象指针**

i-node 表

每个文件系统会为存储于其上的所有文件(包括目录)维护一个*i-node*表，单个*i-node*包含以下信息：

- **文件类型**(*file type*)，可以是常规文件、目录、套接字或*FIFO*
- **访问权限**
- **文件锁列表**(*file locks*)
- **文件大小**
- 等等

*i-node*存储在磁盘设备上，内核在内存中维护了一个副本，这里的*i-node*表为后者。 副本除了原有信息，还包括：**引用计数**(从打开文件描述体)、所在**设备号**以及一些临时属性，例如文件锁。

场景解析

*图-1*中，详细描述了两个进程诸多文件描述符，以及相互关系。

文件描述符复制

在进程*A*中，文件描述符*1*和文件描述符*20*都指向同一个打开文件描述体(标号*23*)。 这很可能是通过调用*dup()*系列函数形成的。

文件描述符复制，在某些场景下非常有用，比如：标准输入/输出重定向。 在*shell*下，完成这个操作非常简单，大部分人都会，但是极少人思考过背后的原理。

大概描述一下需要的几个步骤，以标准输出(文件描述符为*1*)重定向为例：

1. 打开目标文件，返回文件描述符*n*；
2. 关闭文件描述符*1*；
3. 调用*dup*将文件描述符*n*复制到*1*；
4. 关闭文件描述符*n*；

子进程继承文件描述符

进程*A*的文件描述符*2*和进程*B*的文件描述符*2*都指向同一个打开文件描述体(标号*73*)。 这种情形很可能发生在调用*fork()*派生子进程之后，比如A调用*fork()*派生出*B*。 这时，*B*作为子进程，从父进程*A*继承了文件描述符表，其中包括图中标明的文件描述符*2*。 这就是**子进程继承父进程打开的文件**这句话的由来。

当然了，进程*A*通过Unix套接字将一个文件描述符传递给*B*也会出现类似的情形，但一般文件描述符数值是不一样的。 同时为*2*要非常凑巧才发生。