第三本关于UNIX的书，看起来真的挺难读的，很痛苦

第一部分：简介

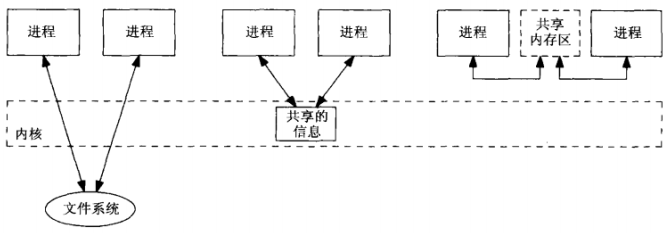
第一章：

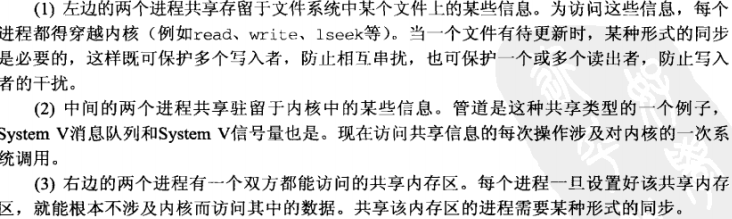
IPC（进程间通信）：运行在某个操作系统上的不同进程间的各种消息传递的方式

同步：因为有共享内存区这种方式 ，有共享也就要同步

消息传递发展阶段：管道 System V消息队列 Posix消息队列 远程过程调用

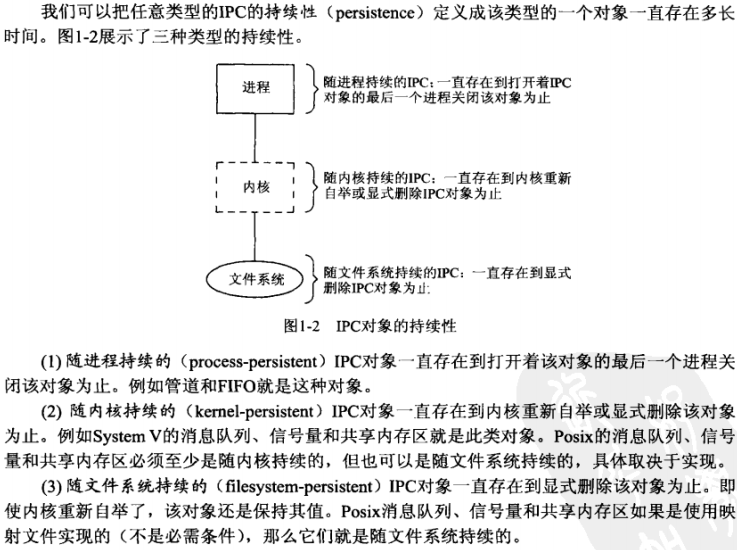
1、进程、线程与信息共享



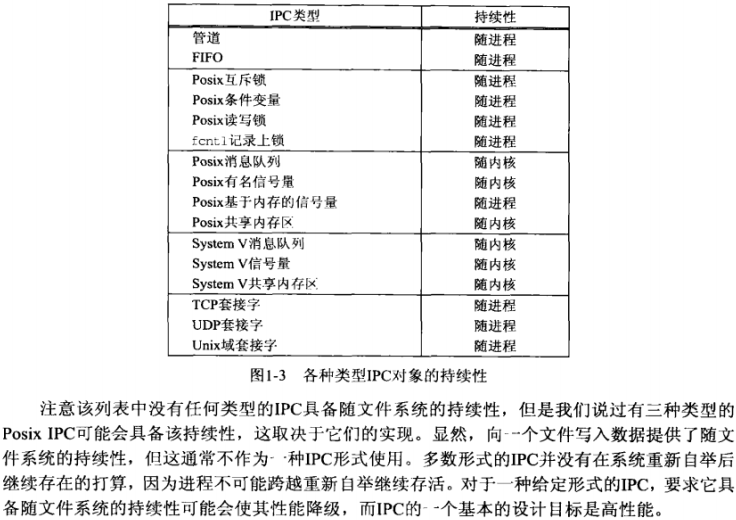


线程共享进程数据，同时不同线程访问数据需要有同步机制

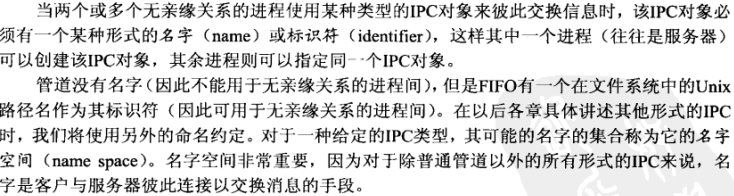
2、IPC对象的持续性



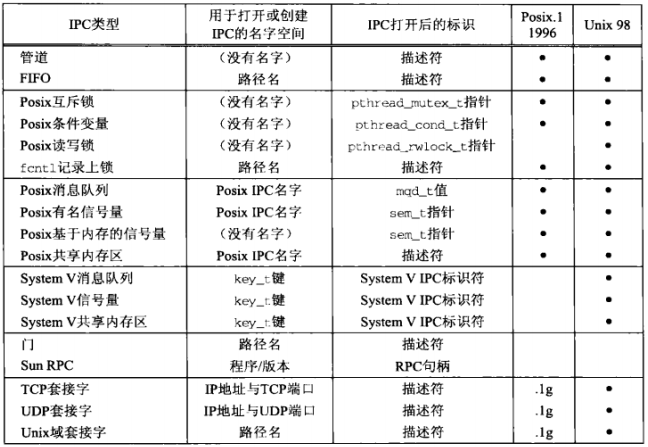
IPC对象如管道是随进程的并在内核中维护，如果管道打开用于读的进程关闭该管道后，内核会丢弃所有数据并删除管道



3、名字空间：



各种形式IPC的名字空间



4、fork、exec、exit对IPC对象的影响



5、出错处理：包裹函数

包裹函数也看了好几遍了。

6、Unix errno值：

当Unix函数发生错误时，全局变量errno将被设置为一个只是错误类型的正数。函数本身则通常返回-1。

在多线程环境中，每个线程必须有自己的errno变量。提供一个局限于线程的errno变量的隐式请求是自动处理的。

对Unix中“mq\_send函数返回EMSGSIZE错误”这样的 用语，来表示函数出错，返回-1，并在errno中设置了指定的常值

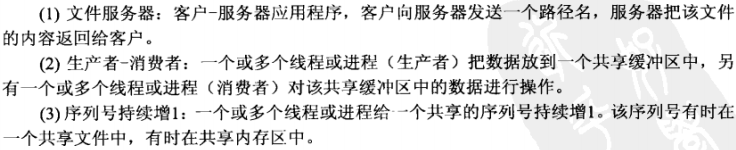
7、Unix标准

Posix标准：可移植操作系统接口首字母缩写。

Open Group：另一个制定标准的组织

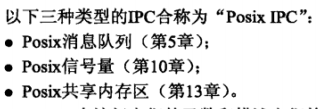
8、本书籍的IPC例子：

为分析各种特性，全书使用了三种交互模式

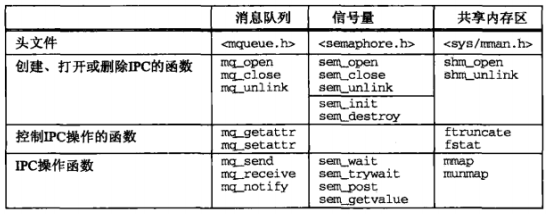


第二章：Posix IPC

1、三种类型的IPC合称为Posix IPC



2、下图汇总了所有Posix IPC函数



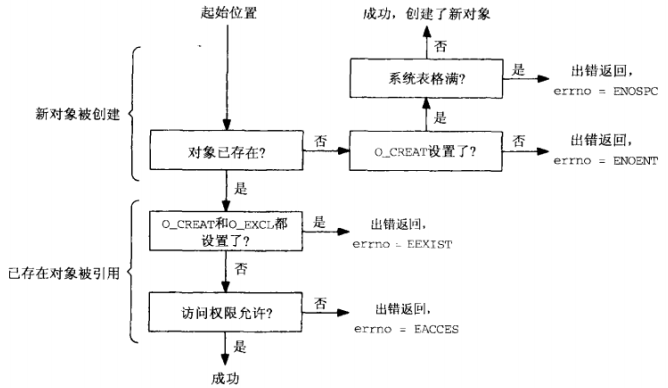
3、三种类型的Posix IPC都使用 Posix IPC名字 进行标识。可能是某个文件中的一个真正的路径名。

不同的标准有不同的IPC名字

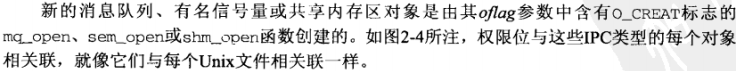
4、创建和打开IPC通道

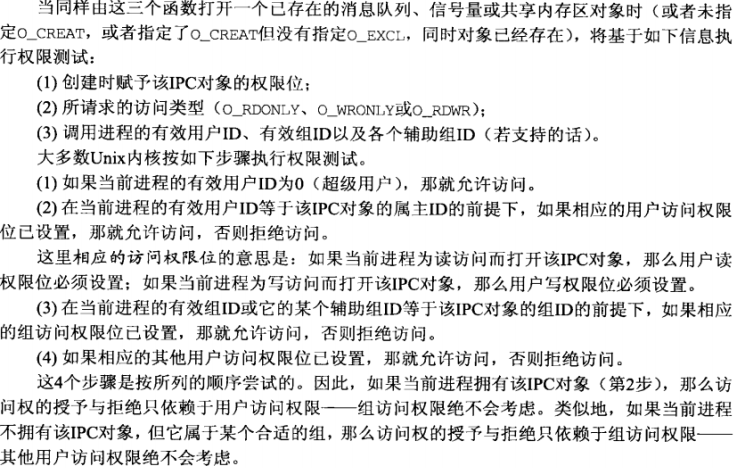
mq\_open、sem\_open、shm\_open,

函数中有参数用来设置IPC的属性



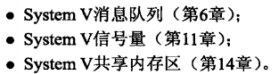
5、访问已存在的IPC的权限设置



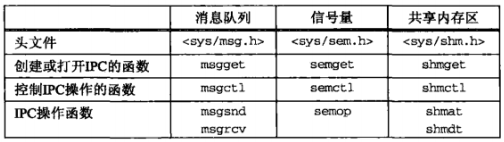


第三章：System V IPC

1、包含以下三种



所有System V IPC函数



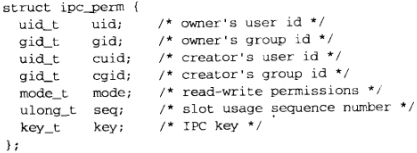
2、key\_t键和ftok函数



Key\_t是IPC标识，ftok把一个已存在的路径名和一个整数标识符转换成一个key\_t值，简称IPC键

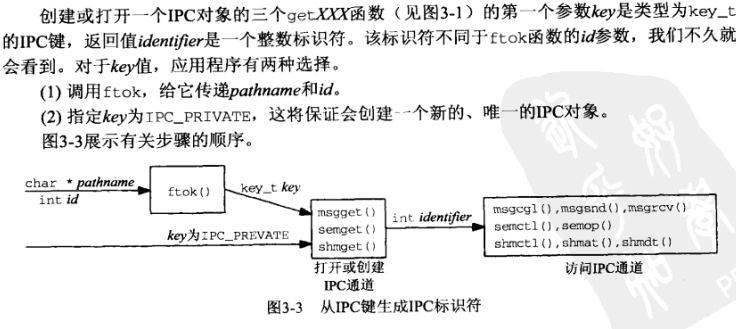
id用低8位的数据，

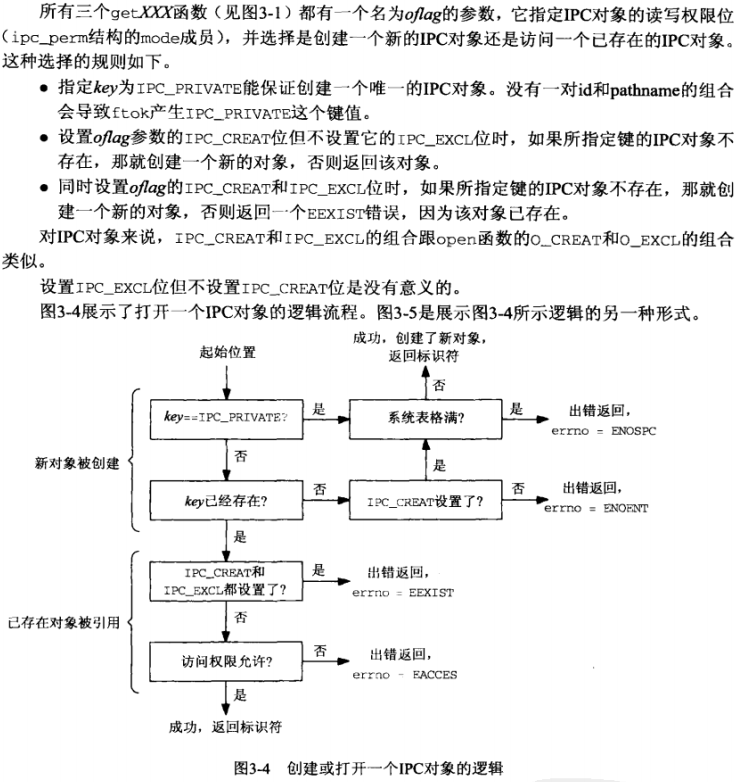
3、ipc\_perm结构

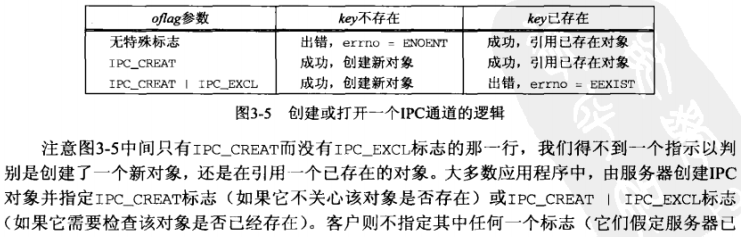


内核给每个IPC对象维护一个信息结构

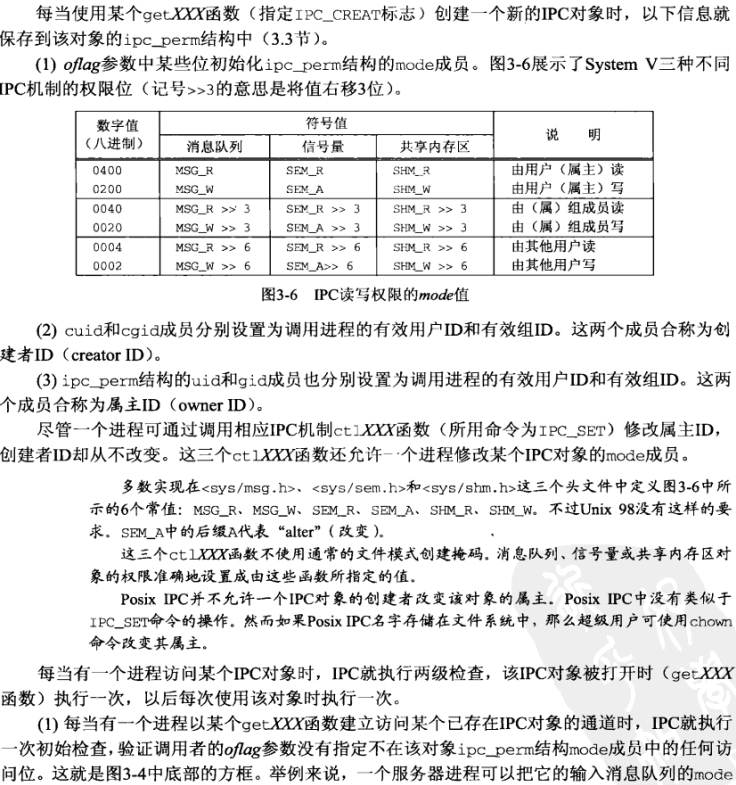
4、创建与打开IPC通道

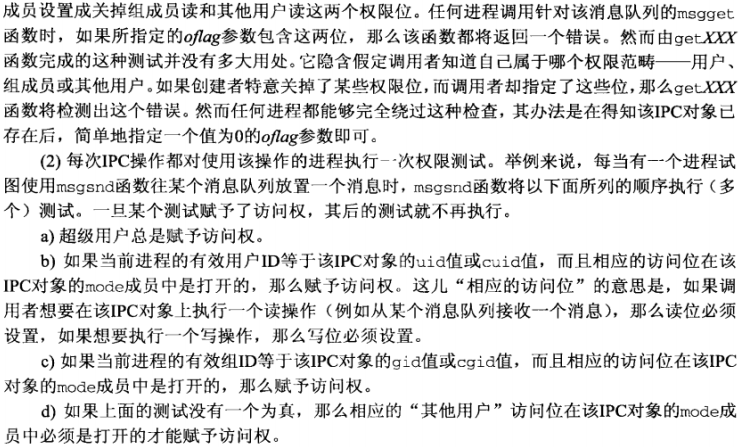






5、IPC权限





6、标识符重用

为了避免类似文件描述符特定于进程

为了避免标识符值范围过小被其他无关进程得到消息队列中的消息

因此创建新的IPC对象时返回的标识符是不断地增大，直到溢出，然后归零，其中内核用ipc\_perm中的seq变量进行记录，每当删除IPC对象时，就将seq值增大一个值作为下一次的IPC对象返回的标识符

7、ipcs和ipcrm程序：前者查看有关System V IPC特性各种信息，后者删除一个System V 消息队列，信号量集或共享内存区

8、内核限制：System V IPC的多数实现由内在的内核限制，例如消息队列的最大数目、每个信号集的最大信号量数等。

第二部分：消息传递（消息队列，管道，FIFO）

第四章：管道与FIFO

1、管道：两个int的数组，然后构建pipe，第一个读，第二个写，父进程创建子进程会复制这个管道的数据，但管道只有一个，父子进程可以通过这个管道通信。只有一个方向可以进行通信，因此是半双工的通道。

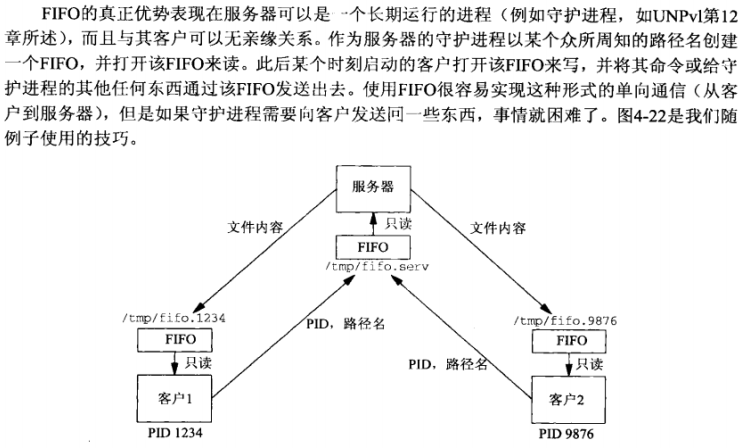
2、全双工管道：某些系统提供全双工管道，可以在一个管道中进行两端不同时的分别读和写

3、popen：创建一个管道并启动另外一个进程，该进程要么从该管道读出标准输入，要么往该管道写入标准输出。

4、FIFO：有名字的管道，每个FIFO有一个路径名与之关联，从而允许无亲缘关系的进程访问同一个FIFO。客户端和服务器不是一个共同祖先进程。

5、FIFO的优势：表现在服务器可以是一个长期运行的进程。

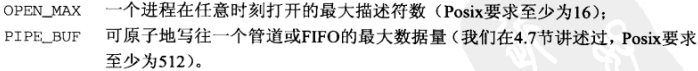
6、单个服务器，多个客户：客户传递给服务器数据时把自己创建的FIFO的路径给了服务器进程，服务器通过这个来个客户端进程写东西。



上面这种客户服务器模型中，多个客户再给一个服务器写数据时有原子性保证。

7、FIFO是一种只能在单台主机上使用的IPC形式。

8、系统加于管道和FIFO的限制：



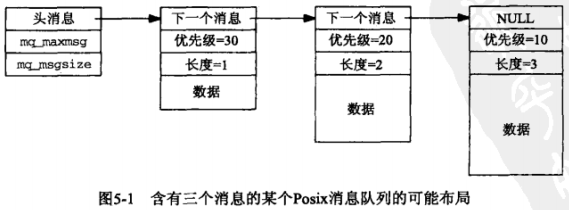
第五章：POSIX消息队列

1、消息队列可认为是一个消息链表，有足够写权限的线程可往队列中放置消息，有足够读权限的线程可从队列中取走消息。每个消息都是一个记录，发送者赋予一个优先级，某一个进程向一个队列写入消息之前，并不需要另外某个进程在该队列上等待消息的到达。与管道和FIFO相反

2、一个进程可以往某个队列中写入一些消息，然后终止，再让另一个进程在以后某个时刻读出这些消息。消息队列具有随内核的持续性。

3、Posix消息队列总是读最高优先级的最早消息，对System V消息队列的读则可以指定任意优先级；往空队列中放入消息时，Posix允许产生一个信号或者启动一个线程，System V消息队列则不提供类似机制。

4、队列中消息的属性：一个无符号整数优先级，长整数类型；消息的数据部分长度；数据本身。



5、mq\_open mq\_close mq\_unlink分别是创建并打开消息队列，关闭，删除消息队列

6、消息队列有使用它的进程个数的引用计数器，mq\_unlink可以在不为零时就将其name删除，mq\_close在引用为零时调用才析构

7、消息队列有四个属性，mqgetattr获取，mqsetattr设置

8、mq\_send和mq\_receive函数，往队列中放置一个消息和从队列中取走一个消息，取消息函数可以设置非阻塞模式，如果没有消息，就直接出错返回而不阻塞等待。

9、消息队列限制：mq\_mqxmsg 队列中的最大消息数 mq\_msgsize 给定消息的最大字节数 MQ\_OPEN\_MAX一个进程能够同时拥有的打开着消息队列的最大数目 MQ\_PRIO\_MAX任意消息的最大优先级值加1

10、mq\_notify函数：为避免接受消息的函数发生调用mq\_receive阻塞或者要轮询，Posix可以通知何时有一个消息放置到了某个空消息队列中。这种通知有两种方式可供选择：产生一个信号；创建一个线程来执行一个指定的函数。允许异步事件通知。

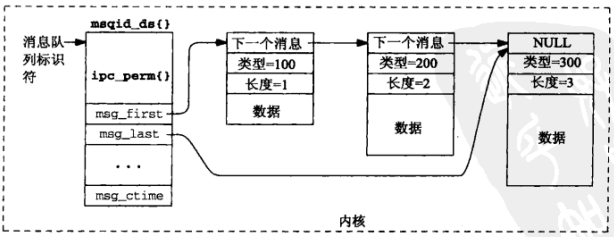
11、信号安全函数：可以在信号处理函数中调用的函数，所有标准IO都不行 避免从信号处理程序中调用任何函数的方法之一是：让处理程序仅仅设置一个全局标志，由某个线程检查该标志以确定何时接受一个消息。

12、共享内存IO实现 Posix消息队列

第六章：System V队列消息

1、 System V消息队列使用消息队列标识符标识。具有足够特权的任何进程都可以往一个给定队列放置一个消息。同样也可以从一个给定队列读取一个消息。

内核中的System V消息队列结构 ：



2、msgget：创建新的消息队列或访问已存在的消息队列。msgsnd：往消息队列上放置信息 msgrcv：从某个消息队列读出一个信息

msgctl：提供在一个消息队列上的各种控制操作。可删除消息队列或者给消息队列配置一些属性

3、客户-服务器例子：

消息队列上无法使用select和poll，和Posix一样，借助管道和子进程，子进程阻塞在接收调用中，当msgrcv返回，子进程从指定队列中读

出消息，放到管道中，服务器在管道或其他一些网络连接中select

第三部分 同步（之前是消息传递和概述 ）

第七章 互斥锁和条件变量

同步问题：互斥锁条件变量出自于Posix.1线程标准。

对于线程：他们共享进程的数据，可用来同步一个进程内的各个线程的。

对于进程：如果一个互斥锁或者条件变量存放在多个进程间共享的某个内存区中，Posix还允许它用这些进程间的同步。

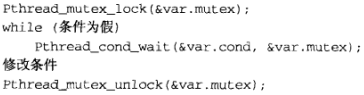
1. 互斥锁还是之前的apue中的那些，有阻塞于上锁的，有try的非阻塞式访问
2. 生产者消费者经典问题：隐式的，甚至生产者和消费者不知道内核在执行同步。

通用的编程原则，应该尽量减少一个互斥锁锁住的代码量

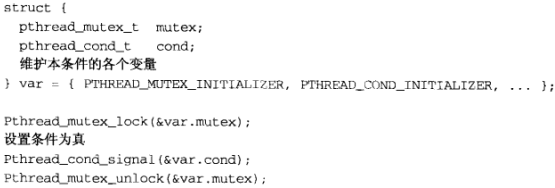
生产消费者问题又称为 共享缓冲区 问题，也就是一个或多个进程或线程共享一块数据或者缓冲区，因此会产生同步的问题。

1. 条件变量用于等待，互斥锁用于上锁，条件变量是类型为pthread\_cond\_t的变量，进程使用pthread\_cond\_wait进行等待，进入睡眠，当pthread\_cond\_signal通知时恢复。

测试条件变量进入睡眠



给条件变量发送信号的代码大体如下：

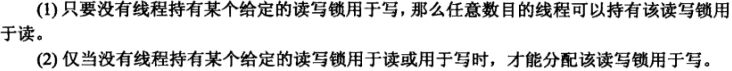


1. 与第三条内容相关的，还有定时的等待，pthread\_cond\_timedwait 等待超时了就会报错，多个进程的广播，pthread\_cond\_broadcast广播所有等待进程
2. 互斥锁和条件变量都有一定的属性， 可以用一些函数进行初始化和设置。可以允许在不同进程间共享某个互斥锁或条件变量，其前提是该互斥锁或条件变量必须存放在这些进程共享的内存区中。

第8章 读写锁

对于读和写关键区域进行区分，而不是用互斥锁把所有临界区内容一次只交给一个线程。可以提高访问临界区的效率。

1. 分配规则如下：



给读操作的线程更高的并发度

1. 读写锁的加锁函数有加写锁和加读锁两种，同样有两个对应的非阻塞的版本。
2. 读写锁的数据类型为pthread\_rwlock\_t，通过赋常值可以进行初始化，同样可以通过相应函数来销毁
3. 使用互斥锁和条件变量实现读写锁
4. 线程取消

第9章 记录上锁

指对一个文件中的一部分范围的字节进行上锁。

粒度很小，对Posix记录上锁来说，粒度就是单个字节。

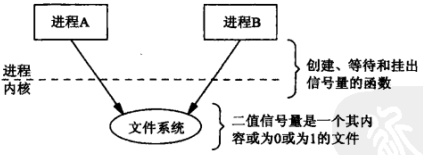
通常情况下粒度越小，允许同时使用的用户数就越多。

1. 记录上锁Posix接口是fcntl函数
2. fcntl记录上锁既可以用于读也可以用于写，对于一个文件的任一字节，最多只能存在一种类型的锁。一个给定字节可以有多个读出锁，只能有一个写入锁。
3. 当一个进程关闭自己打开文件的所有描述符或它本身终止时，该文件关联的所有锁被删除。锁不能通过fork由子进程继承。
4. 劝告性上锁和强制性上锁，劝告性上锁其他进程可以不管你的锁仍然访问文件，只要有权限，强制性上锁则不行。
5. 写入者的锁应该比读出者优先，有的系统实现如Solaris 2.6 BSD/OS 3.1却都是按照FIFO的原则分配锁的，而不管锁的类型
6. 记录上锁的一个常见用途是确保某个程序某个时刻只有一个副本在运行。如一个守护进程维护一个只有一行文本的文件，其中有它的进程ID，它打开这个文件，然后请求整个文件的写入锁。如果没有取得该锁，我们就知道该程序有另一个副本在运行，于是输出一个出错消息并终止。
7. NFS上锁，NFS就是网络文件系统，NFS的大多数实现支持fcntl记录上锁。

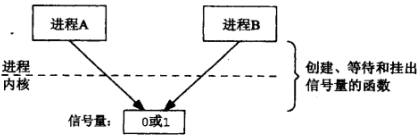
第10章 Posix信号量。

Posix信号量有两种：

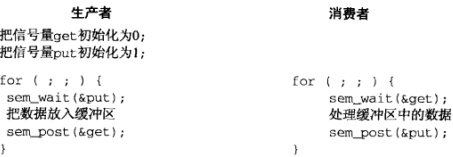
1. Posix有名信号量 和 基于内存的信号量 ，都可以用于进程或线程之间的同步。前者使用Posix IPC名字标识
2. P获取信号量，V释放信号量，PV操作。
3. Posix中的信号量不需要内核来维护，可能是与文件系统中的路径名对应的名字来标识的。



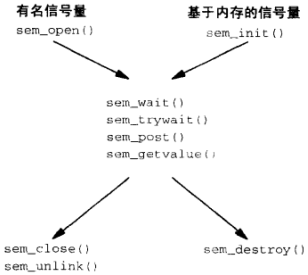
由内核维护的信号量则是下面这种模型



1. 一个进程可以在某个信号量上执行3种操作。创建 等待（P） 挂出（V）
2. 二值信号量和互斥锁一样。
3. 互斥信号量不同于互斥锁的地方在于互斥信号量加锁的进程不一定是解锁的进程。看下面生产者消费者模型



1. Posix提供的两种信号量 有名信号量和基于内存的信号量 使用的函数

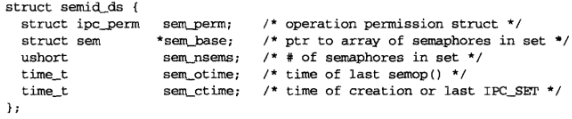
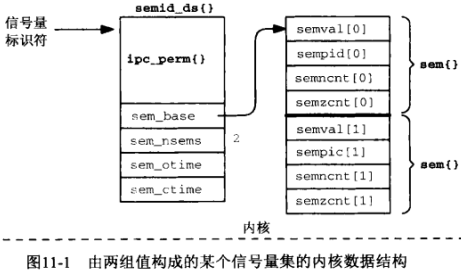
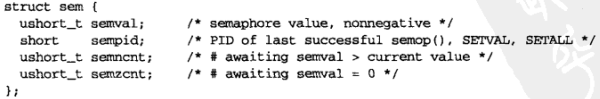


其中getvalue返回信号量的值 post是V

1. 生产者消费者问题：三个信号量组合解决
   1. 用名为mutex的二值信号量保护两个临界区：一个是往共享缓冲区中插入一个数据条目进行写的，一个是往共享缓冲区中取出一个条目进行读的
   2. 用名为nempty的计数信号统计共享缓冲区中的空槽位数，初始化为缓冲区中的槽位数
   3. 名为nstored的计数信号量统计共享缓冲区中已填写的槽位数。该信号量初始化为0。
2. 同样可以使用基于内存的信号量进行编写生产者消费者问题
3. 生产者消费者可以解决文件复制时双缓冲问题，可以支持多个缓冲区
4. 进程可以共享信号量，，Posix标准中fork后父进程中打开的信号量仍然在子进程中打开

第11章： System V信号量

System V信号量定义了如下概念给信号量增加了一级复杂度

1. 计数信号量集：一个或多个信号量构成一个集合，其中每个都是计数信号量。每个集合的信号量存在一个限制，一般在25个数量级上。当我们谈论Posix信号量时，所指的是单个计数信号量。当我们谈论System V信号量时，所指的是计数信号量集。
2. 
3. 

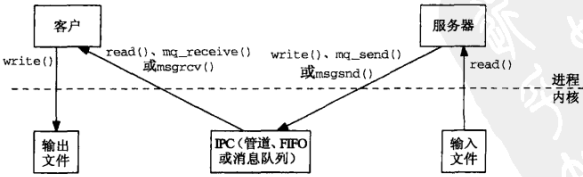
第四部 共享内存区

共享内存区是所有可用IPC形式中最快的。

因为当这样的内存区映射到共享它的进程的地址空间，这些进程间数据的传递就不再涉及内核。

既然是共享内存区，那就需要考虑同步的问题，上一部分已经考虑过了。

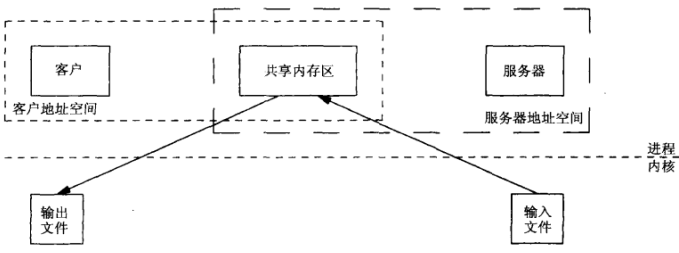
1. 考虑数据从服务器到客户进程的过程



这里通常需要进行四次数据复制，而且这四次复制是在内核和某个进程间进行的，往往开销很大。

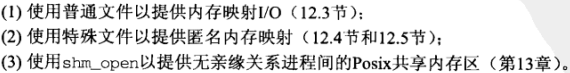
上图的IPC形式的问题在于，两个进程要交换信息时，这些信息必须经由内核传递。

1. 通过让两个或多个进程共享一个内存区，共享内存区这种IPC形式提供了绕过上述问题的办法。当然，这些进程需要协调或同步对该共享内存区的使用
2. 共享内存区完成上图工作如下：

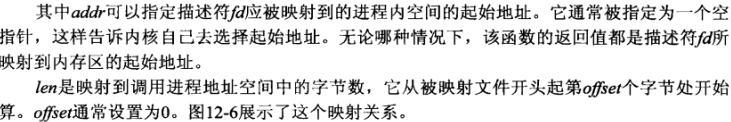


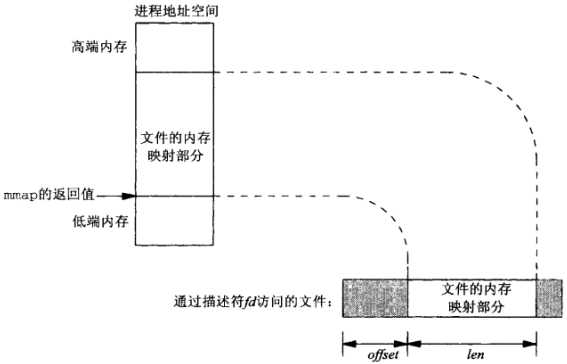
本图中数据只复制两次：一次从输入文件到共享内存区，另一次从共享内存区到输出文件。

1. mmap munmap msync函数，这三个函数中mmap函数把一个文件或者一个Posix共享内存区对象映射到调用进程的地址空间。使用该函数有三个目的：

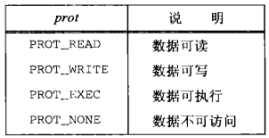
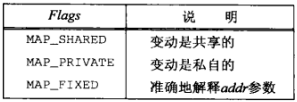




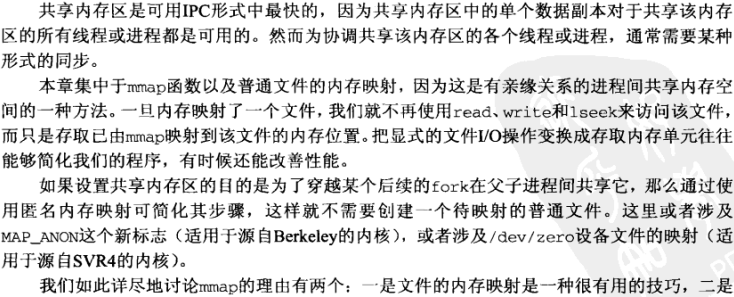


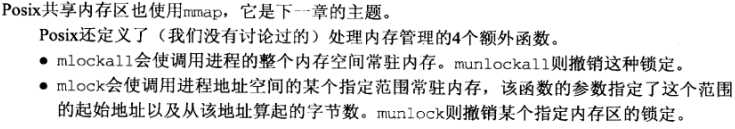


mmap的两个参数的选项：

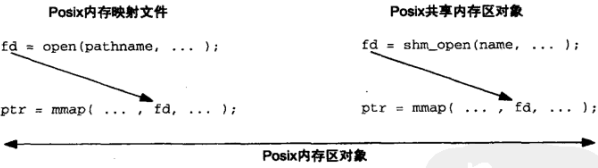
1. munmap函数：从某个进程的地址空间中删除一个映射关系
2. msync函数：文件和加载到内存中的内容，如果内存中更改了，为了确信硬盘上的文件内容与内存映射区的内容一致，需要调用msync来进行同步
3. BSD匿名内存映射：如果是fork穿越父子进程共享内存区，可以只是用匿名内存映射，避免文件的创建和打开
4. 小结





第13章：Posix共享内存区

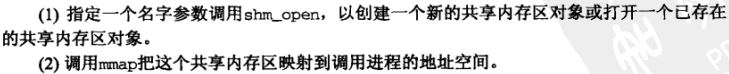
1. 无亲缘关系进程间共享的内存区包括在内：Posix提供了两种：
   1. 内存映射文件：由open函数打开，由mmap函数把得到的描述符映射到当前进程地址空间中的一个文件。
   2. 共享内存区对象：由shm\_open打开一个Posix.1 IPC名字，返回的描述符由mmap函数映射到当前进程的地址空间。



两种方式都是用mmap函数来建立映射区，但是不同的地方在于一个用open，一个用shm\_open函数来创建mmap中的文件描述符。

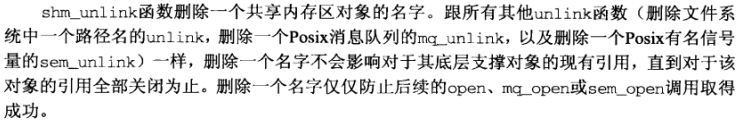
1. shm\_open和shm\_unlink函数

Posix共享内存区涉及以下两个步骤要求：



传递给shm\_open的名字参数随后由希望共享该内存区的任何其他进程使用

Shm\_open返回值是一个整数描述符，随后用作mmap的第五个参数

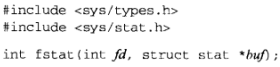


1. ftruncate和fstat函数

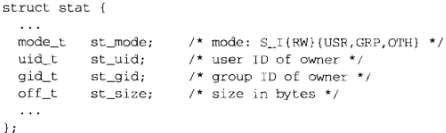
处理mmap的时候，普通文件或共享内存区的大小都可以通过调用ftruncate修改。



对于一个已经存在的共享内存区对象，可以调用fstat来获取有关该对象的信息



stat结构有12个或以上的成员，当fd指代共享内存区对象时，只有四个成员含有信息



1. 小结

