GNU/Linux
系统资源管理命令

系统资源监控 -IPC

一个大型的应用系统，往往需要众多进程协作，进程间通信的重要性显而易见。

Linux 完美的继承了 UNIX 系统。从而 Linux 系统下的进程通信手段基本上是从 UNIX 平台上的进程通信手段继承而来的。

系统资源监控 -IPC

通信目的

进程通过与内核及其它进程之间的互相通信来协调它们的行为。

数据传输：一个进程需要将它的数据发送给另一个进程，发送的数据量在一个字节到几兆字节之间。

共享数据：多个进程想要操作共享数据，一个进程对共享数据的修改，别的进程应该立刻看到。

系统资源监控 -IPC

通信目的

通知事件：一个进程需要向另一个或一组进程发送消息，通知它（它们）发生了某种事件（如进程终止时要通知父进程）。

资源共享：多个进程之间共享同样的资源。为了作到这一点，需要内核提供锁和同步机制。

进程控制：有些进程希望完全控制另一个进程的执行（如 Debug 进程），此时控制进程希望能

系统资源监控 -IPC

UNIX IPC 包括：

1. 管道
2. FIFO
3. 信号

System V IPC 包括

1. system V 消息队列
2. System V 信号灯
3. System V 共享内存区

系统资源监控 -IPC

POSIX IPC 包括：

1.POSIX 消息队列

2.POSIX 信号灯

3.POSIX 共享内存区。

系统资源监控 -IPC

对于进程通信有两点需要简单说明一下：

1) 由于 UNIX 版本的多样性，电子电气工程协会（IEEE ）开发了一个独立的 UNIX 标准，这个新的 ANSI UNIX 标准被称为计算机环境的可移植性操作系统界面（POSIX ）。现有大部分 Unix 和流行版本都是遵循 POSIX 标准的，而 Linux 从一开始就遵循 POSIX 标准；

2)BSD 并不是没有涉足单机内的进程间通信（ socket 本身就可以用于单机内的进程间通信）

系统资源监控 -IPC

Linux 下进程间通信的几种主要手段：

管道（ pipe ）及有名管道（ named pipe ）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信

系统资源监控 -IPC

Linux 下进程间通信的几种主要手段：

信号（ Signal ）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身； linux 除了支持 Unix 早期信号语义函数 sigal 外，还支持语义符合 Posix.1 标准的信号函数 sigaction（实际上，该函数是基于 BSD 的，BSD 为了实现可靠信号机制，又能够统一对外接口，用 sigaction 函数重新实现了 signal 函数）

系统资源监控 -IPC

Linux 下进程间通信的几种主要手段：

报文（ Message ）队列（消息队列）：消息队列是消息的链接表，包括 Posix 消息队列 system V 消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

系统资源监控 -IPC

Linux 下进程间通信的几种主要手段：

共享内存：使得多个进程可以访问同一块内存空间，是最快的可用 IPC 形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

系统资源监控 -IPC

Linux 下进程间通信的几种主要手段：

信号量（ semaphore ）：主要作为进程间以及同一进程不同线程之间的同步手段。

套接口（ Socket ）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由 Unix 系统的 BSD 分支开发出来的，但现在一般可以移植到其它类 Unix 系统上： Linux 和 System V 的变种都支持套接字。

系统资源监控 -IPC

信号（ Signals ）是 Unix 系统中使用的最古老的进程间通信的方法之一。操作系统通过信号来通知进程系统中发生了某种预先规定好的事件（一组事件中的一个），它也是用户进程之间通信和同步的一种原始机制。一个键盘中断或者一个错误条件（比如进程试图访问它的虚拟内存中不存在的位置等）都有可能产生一个信号。Shell 也使用信号向它的子进程发送作业控制信号。

系统资源监控 -IPC

信号的生命周期中有两个阶段：

1．生成：当一个事件发生时，需要通知一个进程，这时生成一个信号。

2．传送。当进程识别出信号的到来，就采取适当的动作来传送或处理信号。在信号到来和进程对信号进行处理之间，信号在进程上挂起（ pending ）。

系统资源监控 -IPC

Linux Kernel 为进程生产信号，来响应不同的事件，这些事件就是信号源。主要的信号源如下：

1. 异常：进程运行过程中出现异常；

2. 其它进程：一个进程可以向另一个或一组进程发送信号；

3. 终端中断： Ctrl-C ， Ctrl-\ 等；

4. 作业控制：前台、后台进程的管理；

系统资源监控 -IPC

5. 分配额： CPU 超时或文件大小突破限制；

6. 通知：通知进程某事件发生，如 I/O 就绪等；

7. 报警：计时器到期。

系统资源监控 -IPC

管道：

单向的、先进先出的、无结构的、固定大小的字节流

它把一个进程的标准输出和另一个进程的标准输入连接在一起。

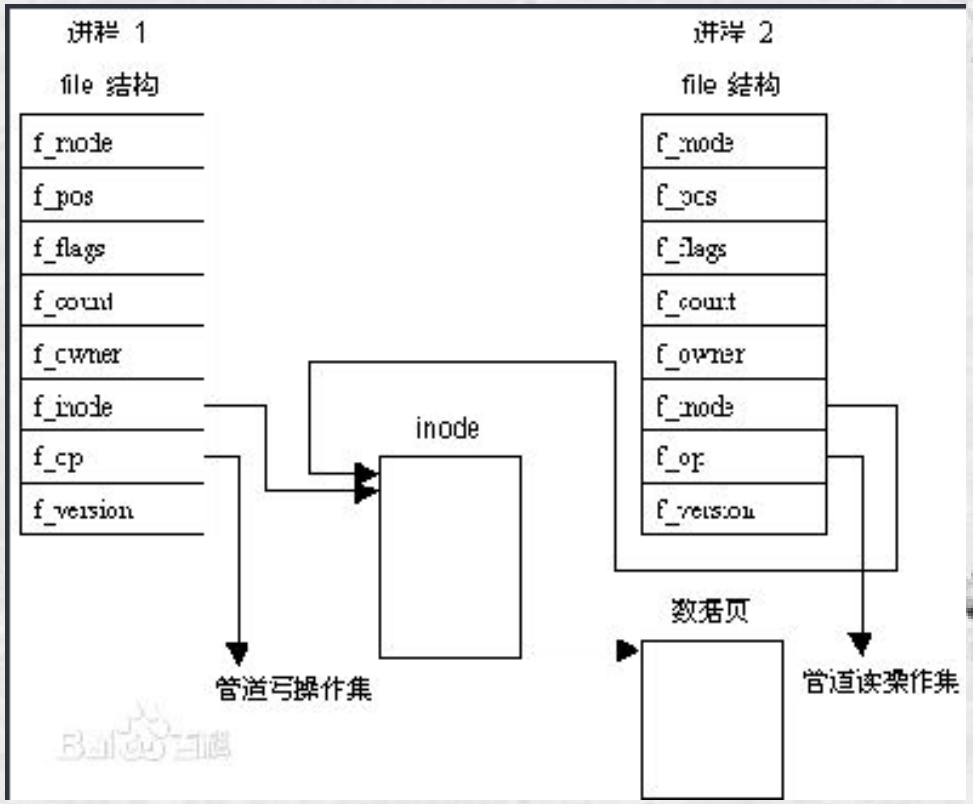
写进程在管道的尾端写入数据，读进程在管道的首端读出数据。

数据读出后将从管道中移走，其它读进程都不能再读到这些数据。

管道提供了简单的流控制机制。

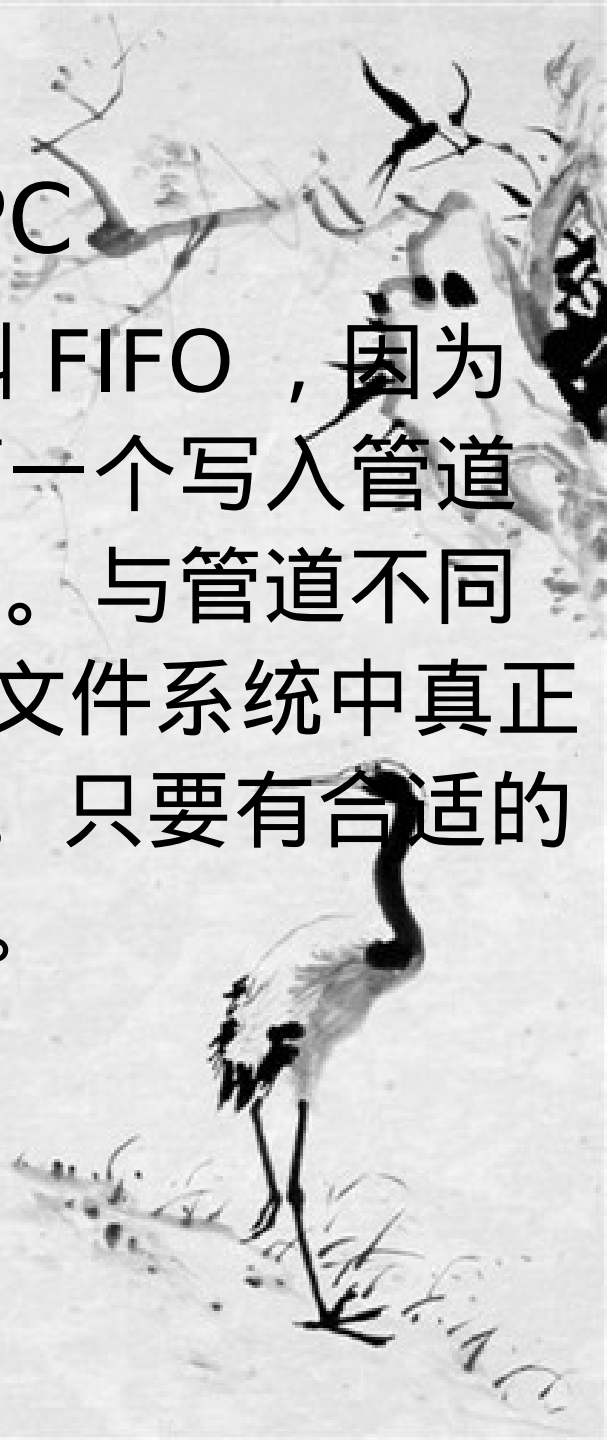
系统资源监控 -IPC

在 Linux 中，使用两个 file 数据结构来实现管道。这两个 file 数据结构中的 f_inode（ f_dentry ）指针指向同一个临时创建的 VFS I 节点，而该 VFS I 节点本身又指向内存中的一个物理页，如图所示。



系统资源监控 -IPC

Linux 也支持命名管道（也叫 FIFO ，因为管道工作在先入先出的原则下，第一个写入管道的数据也是第一个被读出的数据）。与管道不同，FIFO 不是临时的对象，它们是文件系统中真正的实体，可以用 mkfifo 命令创建。只要有合适的访问权限，进程就可以使用 FIFO 。



系统资源监控 -IPC

示例

[root@localhost ~]# mkfifo -m 777 myfifo

----m mode ，这里 mode 指出将要创建 FIFO 的八进制模式

[root@localhost ~]# cat /etc/passwd > myfifo &

--- 将 cat 命令的输出作为此 myfifo 的输入，并放在后太运行

--- 管道先进先出，读写一次为一个完整过程，所以不加 & 放

入后台会锁死界面，称为阻塞

--- 造成阻塞的原因有两种：

系统资源监控 -IPC

示例

[root@localhost ~]# cat myfifo

--- 再用 cat 命令从该 myfifo 中读出数据进行处理

系统资源监控 -IPC

从 IPC 的角度看，管道提供了从一个进程向另一个进程传输数据的有效方法。但是，管道有一些固有的局限性：

1. 读数据的同时也将数据从管道移去，因此，管道不能用来对多个接收者广播数据。

2. 管道中的数据被当作字节流，因此无法识别信息的边界。

3. 如果一个管道有多个读进程，那么写进程不能发送数据到指定的读进程。同样，如果有多个写进程，那么没有办法判断是它们中那一个发送的数据。

系统资源监控 -IPC

对于无法满足其他程序使用 IPC.UNIX 中 System V UNIX （ 1983 ）中首次引入了另外三种进程间通信机制（ IPC ）机制：消息队列、信号灯和共享内存（ message queues ， semaphores and shared memory ）。

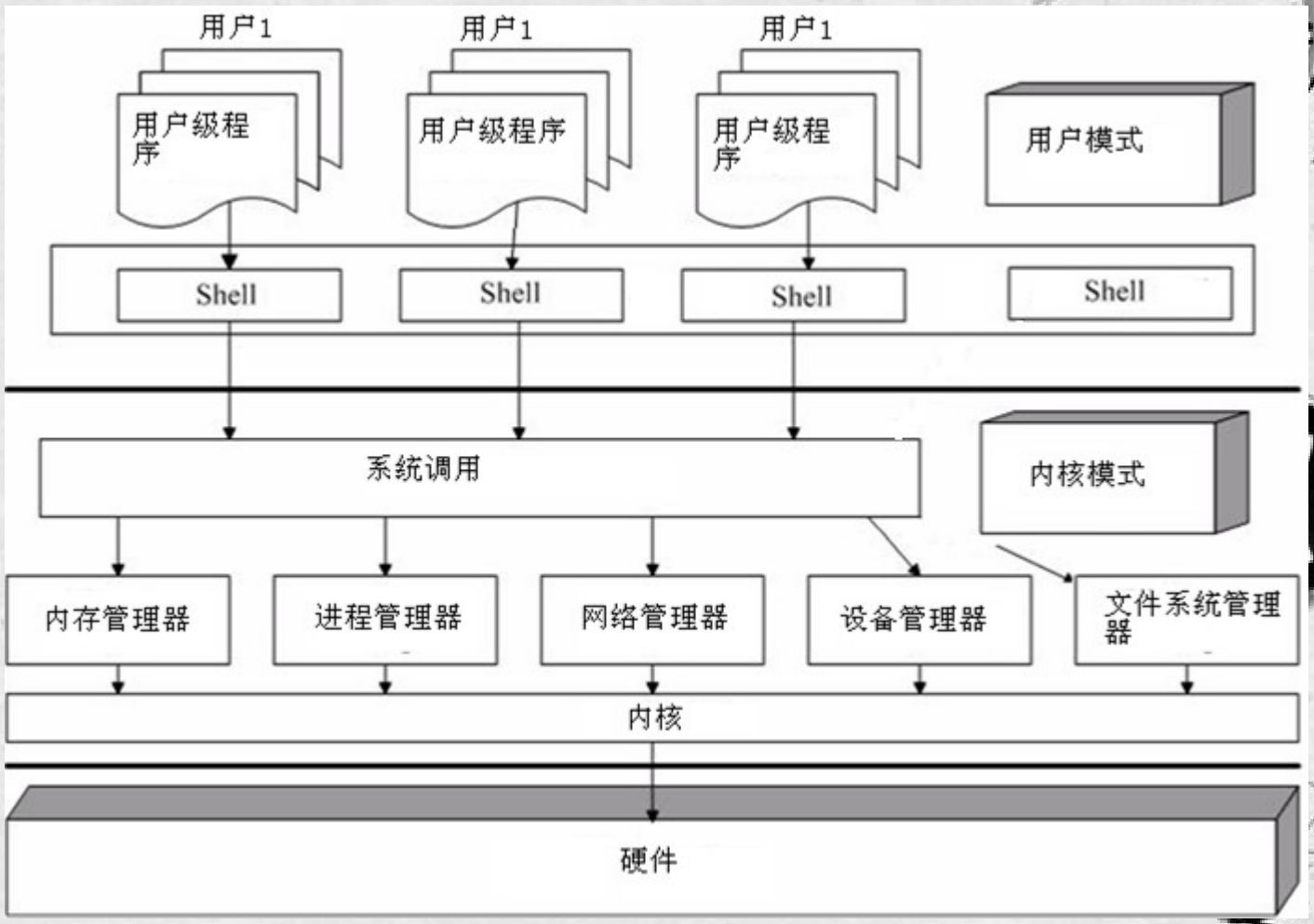
Linux 完全支持 Unix System V 中的这三种 IPC 机制。

系统资源监控 -IPC

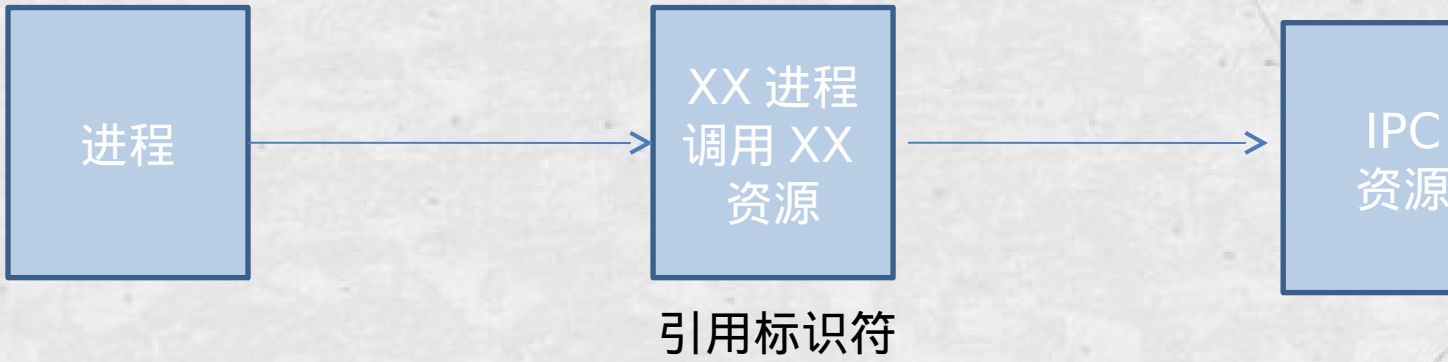
System V IPC 机制共享通用的认证方式。进程在使用某种类型的 IPC 资源以前，必须首先通过系统调用创建或获得一个对该资源的引用标识符。进程只能通过系统调用，传递一个唯一的引用标识符到内核来访问这些资源。在每一种机制中，对象的引用标识符都作为它在资源表中的索引。

如图：

系统资源监控 -IPC


用户1
用户1
用户1
用户级程序
用户级程序
用户级程序
用户模式
Shell
Shell
Shell
Shell
系统调用
内核模式
内存管理器
进程管理器
网络管理器
设备管理器
文件系统管理器
内核
硬件

系统资源监控 -IPC



系统资源监控 -IPC

Message Queues（消息队列）

消息队列就是消息的一个链表，它允许一个或多个进程向它写消息，一个或多个进程从中读消息。

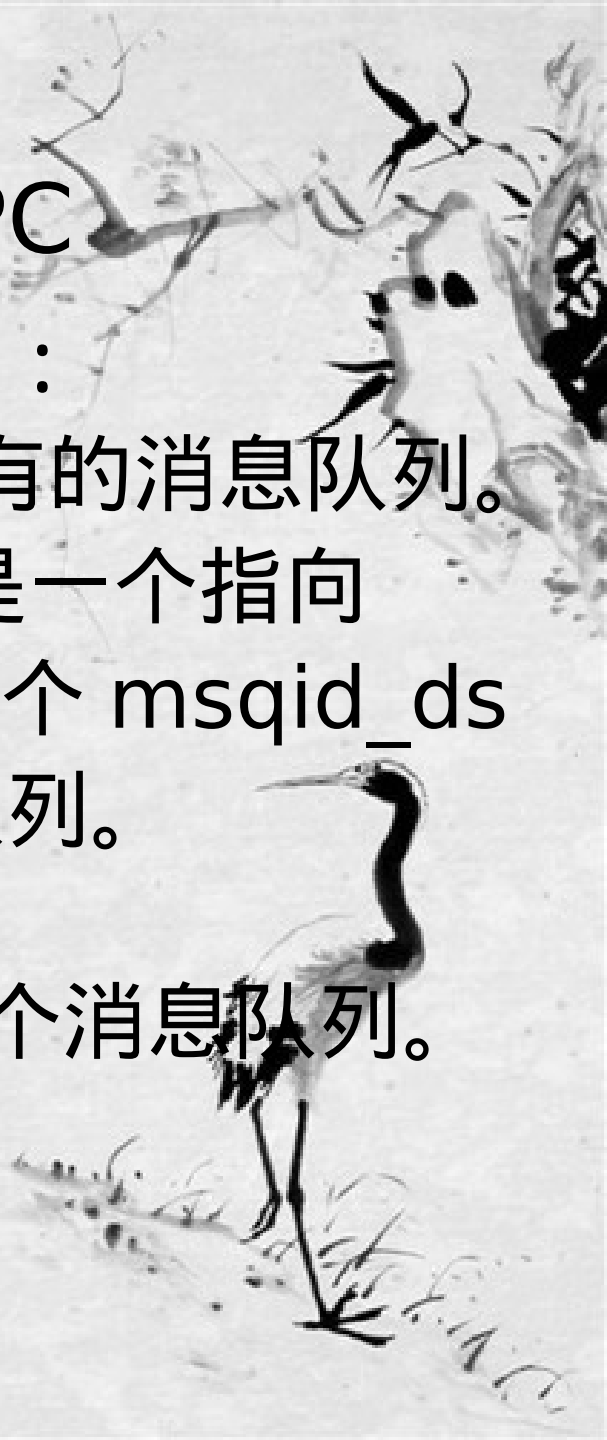
系统资源监控 -IPC

Linux 维护了一个消息队列向量表：

msgque ，来表示系统中所有的消息队列。

该向量表中的每一个元素都是一个指向 msqid_ds 数据结构的指针，而一个 msqid_ds 数据结构完整地描述了一个消息队列。

系统中同时最多可以有 128 个消息队列。



系统资源监控 -IPC

Linux 提供了四个消息队列操作。

1. 创建或获得消息队列（ msgget ）

2. 发送消息 (msgsnd)

3. 接收消息 (msgrcv)

4. 消息控制 (msgctl)

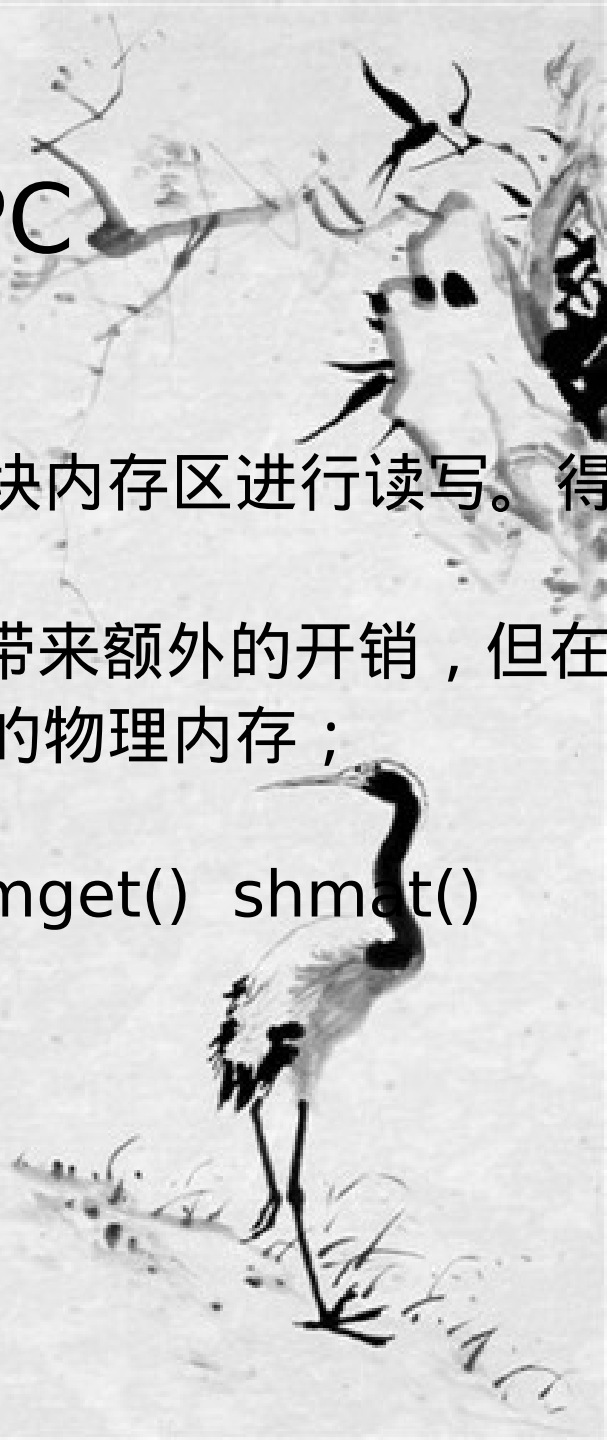
系统资源监控 -IPC

共享内存

通常由一个进程创建，其余进程对这块内存区进行读写。得到共享内存有两种方式：

1. 映射 /dev/mem 设备：不给系统带来额外的开销，但在现实中并不常用，因为它控制存取的是实际的物理内存；

2. 内存映像文件。此方式是通过 shmget() shmat() shmdt 函数族来实现共享内存。



系统资源监控 -IPC

IPC 函数说明

shmget()

创建、获取共享内存

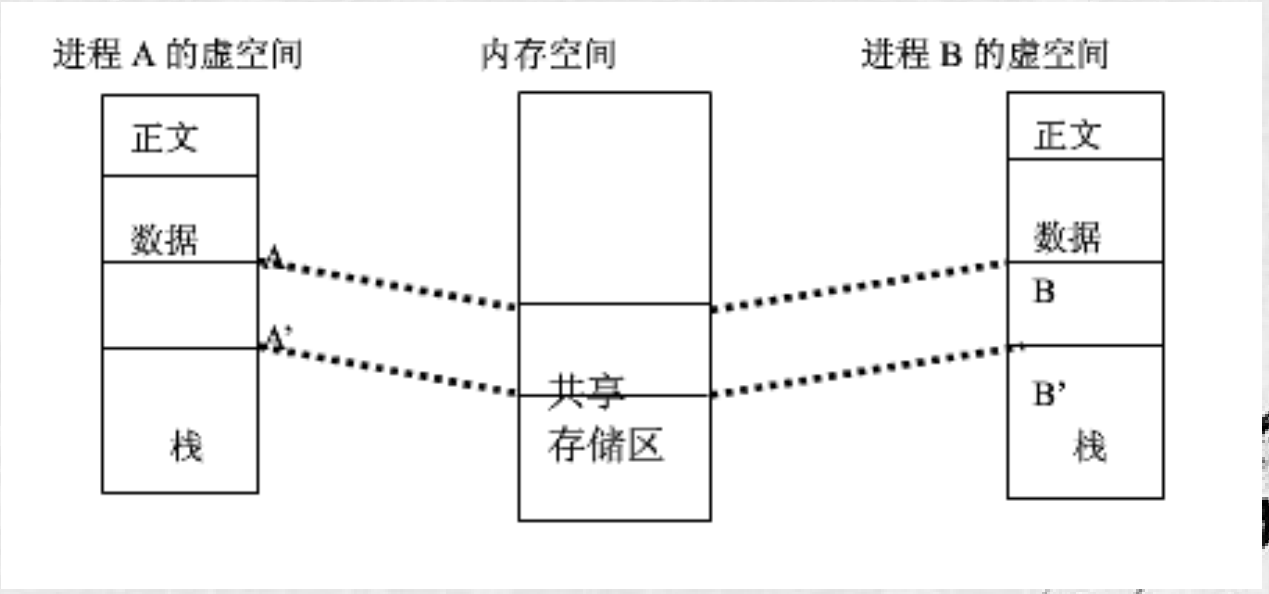
shmat()

将共享内存映射给进程

shmdt()

将内存脱离进程

系统资源监控 -IPC



系统资源监控 -IPC

主要的 IPC 方法

方法	操作系统
文件	多数操作系统都有提供
信号	多数操作系统都有提供
Socket	多数操作系统都有提供
消息队列(en:Message queue)	多数操作系统都有提供
管道(en:Pipe)	所有的 POSIX systems, Windows.
具名管道(en:Named Pipe)	所有的 POSIX systems, Windows.
信号量(en:Semaphore)	所有的 POSIX systems, Windows.
共享存储器	所有的 POSIX systems, Windows.
Message passing(en:Message passing) (不共享)	用于 MPI paradigm, Java RMI, CORBA, MSMQ, MailSlot 以及其他.
Memory-mapped file(en:Memory-mapped file)	所有的 POSIX systems, Windows.

系统资源监控 -ipcs

命令 :ipcs

功能 : 分析消息队列、内存共享与信号量

语法结构 :ipcs ［ 选项 ］

选项 :
-m: 输出有关共享内存 (shared memory) 的信息
-q: 输出有关信息队列 (message queue) 的信息
-s: 输出有关信号进行进程间通信的信息
-a: 输出 -m,-q,-s 的所有信息

系统资源监控 -ipcs

选项：

-t: 输出信息的详细变化时间

-p: 输出以 IPC 方式的进程 ID

-c: 输出以 IPC 方式的创建者和拥有者及权限

-l: 输出 ipc 各种方式的在该系统下的限制条件信息

-u: 输出当前系统下 ipc 各种方式的状态信息（共享内存，消息队列，信号）

系统资源监控 -ipcrm

命令 :ipcrm

功能 :

1. 移除一个消息对象
2. 移除共享内存段
3. 移除一个信号集，同时会将与 ipc 对象相关链的数据也一起移除。
4. 只有管理员 ipc 对象的创建者才可执行

语法格式 :ipcrm [-M key | -m id | -Q key | -q id | -S key | -s id] ...

系统资源监控 -ipcrm

选项：

-M　以 shmkey 删除共享内存

-m　以 shmid 删除共享内存

-Q　以 msgkey 删除消息队列

-q　以 msgid 删除消息队列

-S　以 semkey 删除信号灯

系统资源监控 -ipcrm

示例：

1. 删除 id 为 501 的共享内存区域

```
#ipcrm -m  501
```

2. 删除 oracle 的信号、内存共享

```
#cat  ipcrm2oracle.sh
for i in `ipcs |grep oracle|awk   '{print $2}'`
do
ipcrm -m $i
ipcrm -s $i
done
```