- 进程间通信方式
  - 引言
  - 共享内存
  - 信号量
    - 信号量的工作原理
    - 理解信号量
  - 管道
    - 匿名管道


文章已收录我的仓库：Java学习笔记

# 进程间通信方式

## 引言

在操作系统中，一个进程可以理解为是关于计算机资源集合的一次运行活动，其就是一个正在执行的程序的实例。从概念上来说，一个进程拥有它自己的虚拟CPU和虚拟地址空间，任何一个进程对于彼此而言都是相互独立的，这也引入了一个问题 —— 如何让进程之间互相通信？

由于进程之间是互相独立的，没有任何手段直接通信，因此我们需要借助操作系统来辅助它们。举个通俗的例子，假如A与B之间是独立的，不能彼此联系，如果它们想要通信的话可以借助第三方C，比如A将信息交给C，C再将信息转交给B —— 这就是进程间通信的主要思想 —— 共享资源。

这里要解决的一个重要的问题就是如何避免竞争，即避免多个进程同时访问临界区的资源。

## 共享内存

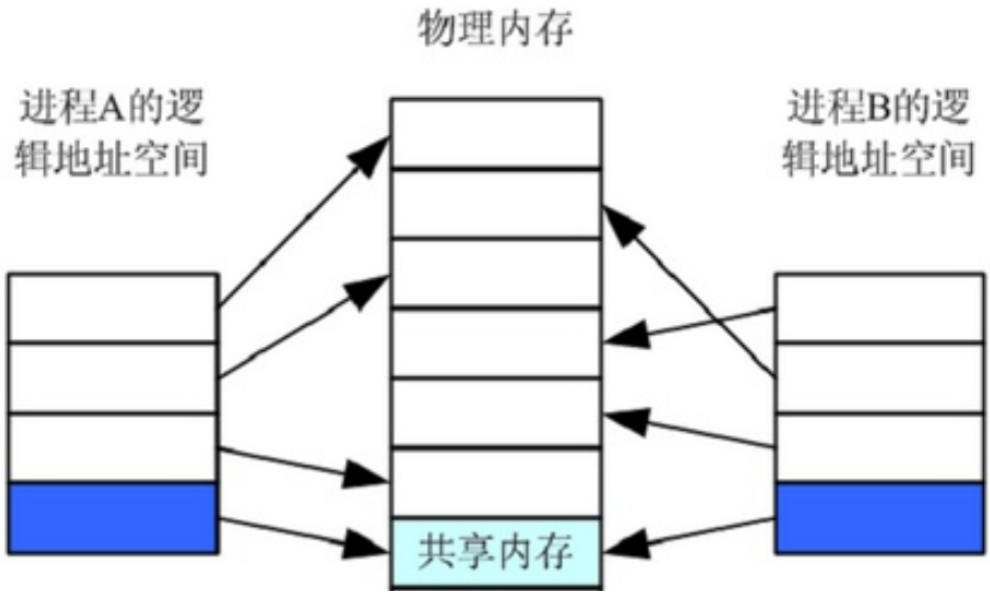
共享内存是进程间通信中最简单的方式之一。共享内存允许两个或更多进程访问同一块内存。当一个进程改变了这块地址中的内容的时候，其它进程都会察觉到这个更改。

你可能会想到，我直接创建一个文件，然后进程不就都可以访问了？

是的，但这个方法有几个缺陷：

- 访问文件需要陷入系统调用，由用户态切入内核态，然后执行内核指令。这样做效率是非常低的，并且是不受用户掌握的。
- 直接访问磁盘是非常慢的，比访问内存要慢上几百倍。从某种意义上说，这是共享磁盘不算共享内存。

Linux下采用共享内存的方式来使进程完成对共享资源的访问，它将磁盘文件复制到内存，并创建虚拟地址到该内存的映射，就好像该资源本来就在进程空间之中，此后我们就可以像操作本地变量一样去操作它们了，因此共享内存的速度是最快的。




如同上图一样，进程将共享内存映射到自己的虚拟地址空间中，进程访问共享进程就好像在访问自己的虚拟内存一样，速度是非常快的。

共享内存的模型应该是比较好理解的：在物理内存中创建一块共享内存，进程将该共享内存绑定到自己的虚拟内存之中。

这里要解决的一个问题是如何将同一块共享内存绑定到自己的虚拟内存中，要知道在不同进程中使用`malloc`函数是会顺序分配空闲内存，而不会分配同一块内存，那么要如何去解决这个问题呢？

Linux操作系统已经想办法帮我们解决了这个问题，在`#include <sys/ipc.h>`和`#include <sys/shm.h>`头文件下，有如下几个shm系列函数：

- shmget函数：由ftok()函数获取需要共享文件资源标识符(IPC键)，将该资源标识符作为参数获取共享内存区域的唯一标识ID。

  > ftok()函数用以标识系统IPC资源，例如这里的共享资源、下文的消息队列、管道...都属于IPC资源。
  >
  > IPC: Inter-Process[ Communication](https://baike.baidu.com/item/Communication/20394231)，进程间通信），IPC是指两个进程的数据之间产生交互。

- shmat函数：通过由shmget函数获取的标识符，建立由共享内存到进程独立空间的映射。

- shmdt函数：释放映射。

通过上述几个函数，每个独立的进程只要有统一的**共享内存标识符**便可以建立起虚拟地址到物理地址的映射，每个虚拟地址将被翻译成指向共享区域的物理地址，这样就实现了对共享内存的访问。

共享内存标识符由内核维护，全局唯一，通过这个标识符进程可以向内核请求获得共享内存基址，并绑定到虚拟内存中。

还有一种相像的实现是采用mmap函数，mmap通常是直接对磁盘的映射——因此不算是共享内存，存储量非常大，但访问慢；**shmat与此相反，通常将资源保存在内存中创建映射，访问快，但存储量较小。**

不过要注意一点，操作系统并不保证任何并发问题，例如两个进程同时更改同一块内存区域，正如你和你的朋友在线编辑同一个文档中的同一个标题，这会导致一些不好的结果，所以我们需要借助信号量或其他方式来完成同步。

# 信号量

信号量是迪杰斯特拉最先提出的一种为解决`同步不同执行线程问题`的一种方法，进程与线程抽象来看大同小异，所以**信号量同样可以用于同步进程间通信**。

## 信号量的工作原理

信号量 s 是具有非负整数值的全局变量，由两种特殊的**原子操作**来实现，这两种原子操作称为 P 和 V ：

- P(s)：如果 s 的值大于零，就给它减1，然后立即返回，进程继续执行。；如果它的值为零，就挂起该进程的执行，等待 s 重新变为非零值。
- V(s)：V操作将 s 的值加1，如果有任何进程在等在 s 值变为非0，那么V操作会重启这些等待进程中的其中一个(随机地)，然后由该进程执行P操作将 s 重新置为0，而其他等待进程将会继续等待。

## 理解信号量

信号量并不用来传送资源，而是用来保护共享资源，理解这一点是很重要的，信号量 s 的表示的含义为*同时允许最大访问资源的进程数量*，它是一个全局变量。

无论何时只有 s 个进程能够访问共享资源，这就是信号量做的事情，他控制进入共享区的最大进程数量，这取决于初始化s的值。此后，在进入共享区之前调用P操作，出共享区后调用V操作，这就是信号量的思想。

在Linux下并没有直接的P&V函数，而是需要我们根据这几个基本的sem函数族进行封装：

- semget：初始化或获取一个信号量，这个函数需要接受ftok()的返回值以及初始s的值，它将全局计数变量s绑定在由ftok标识的共享资源上，并返回一个唯一标识的信

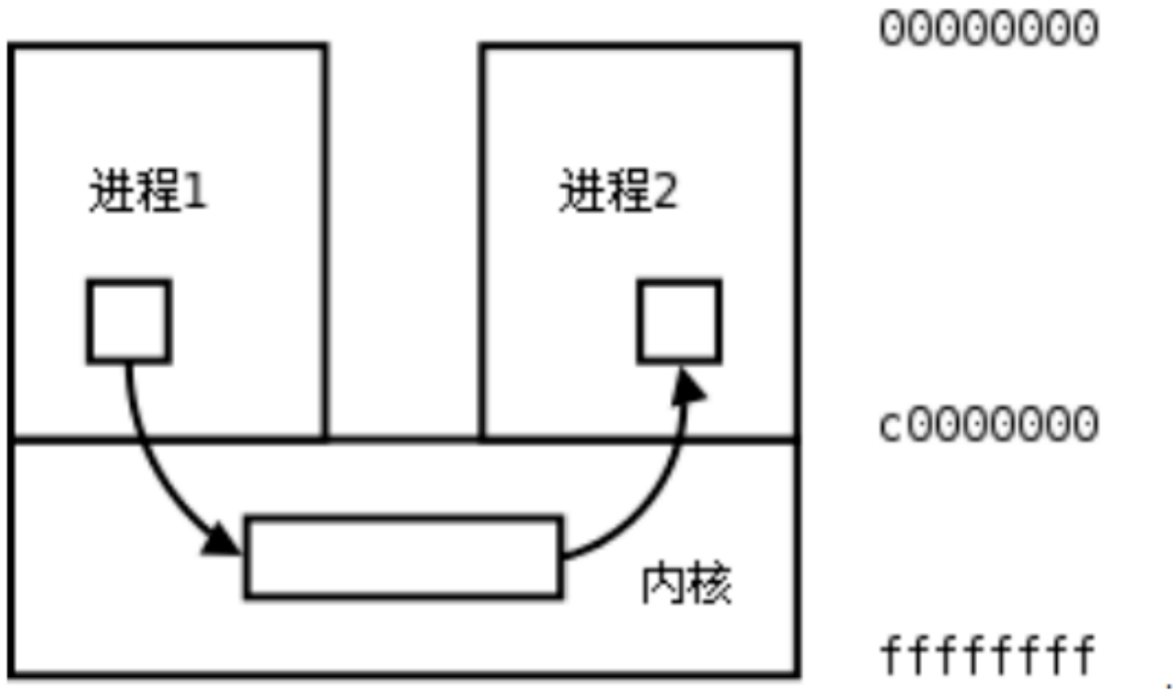
号量组ID。

- semop：这个函数接受上面函数返回的信号量组ID以及一些其他参数，根据参数的不同有一些不同的操作，他将对与该信号量组ID绑定的全局计数变量 s 进行一些操作，P&V操作便是基于此实现。
- semctl：这个函数接受上面函数返回的信号量组ID以及一些其他参数，主要进行控制信号量相关信息，如删除该信号量等。

# 管道

正如其名，管道就如同生活中的一根管道，一端输送，而另一端接收，双方不需要知道对方，只需要知道管道就好了。

管道是一种最**基本的进程间通信机制。** 管道由pipe函数来创建： 调用pipe函数，会在内核中开辟出一块缓冲区用来进行进程间通信，这块缓冲区称为管道，它有一个读端和一个写端。管道被分为匿名管道和有名管道。




## 匿名管道

匿名管道通过pipe函数创建，这个函数接收一个长度为2的Int数组，并返回1或0表示成功或者失败：

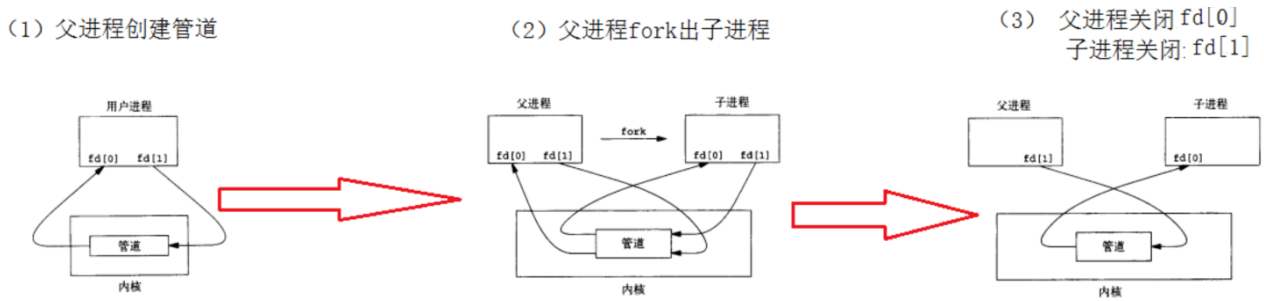
```
int pipe(int fd[2])
```

这个函数打开两个文件描述符，一个读端文件，一个写端，分别存入fd[0]和fd[1]中，然后可以作为参数调用`write`和`read`函数进行写入或读取，注意fd[0]只能读取文件，而fd[1]只能用于写入文件。

你可能有个疑问，这要怎么实现通信？其他进程又不知道这个管道，因为进程是独立的，其他进程看不到某一个进程进行了什么操作。

是的，‘其他’进程确实是不知道，但是它的子进程却可以！这里涉及到fork派生进程的相关知识，一个进程派生一个子进程，那么子进程将会复制（写时复制）父进程的内存空间信息，注意这里是复制而不是共享，这意味着父子进程仍然是独立的，但是在这一时刻，它们所有的信息又是相等的。因此子进程也知道该全局管道，并且也拥有两个文件描述符与管道挂钩，所以**匿名管道只能在具有亲缘关系的进程间通信。**

还要注意，匿名管道内部采用环形队列实现，只能由写端到读端，由于设计技术问题，管道被设计为半双工的，一方要写入则必须关闭读描述符，一方要读出则必须关闭写入描述符。因此我们说**管道的消息只能单向传递。**




**注意管道是堵塞的，如何堵塞将依赖于读写进程是否关闭文件描述符。如果读管道，如果读到空时，假设此时写端口还没有被完全关闭，那么操作系统会假设还有数据要读，此时读进程将会被堵塞，直到有新数据或写端口被关闭；如果管道为空，且写端口也被关闭，此时操作系统会认为已经没有东西可读，会直接退出，并关闭管道。**

**对于写一个已经满了的管道同理而言，如果管道已满，如果此时读端未关闭，则写端会被堵塞直到管道有剩余空间；但是如果读端已经关闭了，那么写端的写将会是无意义的，为防止写端继续写，操作系统会发送信号中止写端。**

**管道内部由内核管理，在半双工的条件下，保证数据不会出现并发问题。**

## 命名管道

了解了匿名管道之后，有名管道便很好理解了。在匿名管道的介绍中，我们说其他进程不知道管道和文件描述符的存在，所以匿名管道只适用于具有亲缘关系的进程，而命名管道则很好的解决了这个问题 —— 现在管道有一个唯一的名称了，任何进程都可以访问这个管道。

**注意，操作系统将管道看作一个抽象的文件，但管道并不是普通的文件，管道存在于内核空间中而不放置在磁盘(有名管道文件系统上有一个标识符，没有数据块)，访问速度更快，但存储量较小，管道是临时的，是随进程的，当进程销毁，所有端口自动关闭，此时管道也是不存在的，操作系统将所有IO抽象的看作文件，例如网络也是一种文件，这意味着我们可以采用任何文件方法操作管道，理解这种抽象是很重要的，命名管道就利用了这种抽象。**

Linux下，采用mkfifo函数创建，可以传入要指定的‘文件名’，然后其他进程就可以调用open方法打开这个特殊的文件，并进行write和read操作(那肯定是字节流对吧)。

注意，命名管道适用于任何进程，除了这一点不同外，其余大多数都与匿名管道相同。

# 消息队列

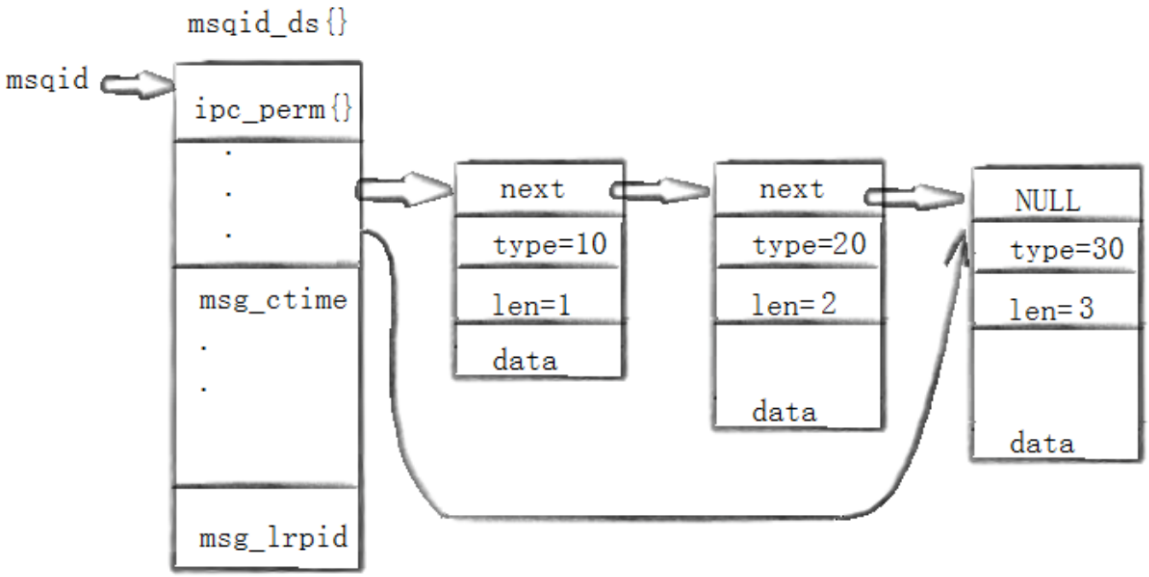
## 什么是消息队列？

消息队列亦称报文队列，也叫做信箱，是Linux的一种通信机制，这种通信机制传递的数据会被拆分为一个一个独立的数据块，也叫做消息体，消息体中可以定义类型与数据，克服了无格式承载字节流的缺陷(现在收到void*后可以知道其原本的格式惹):

```
struct msgbuf {
    long mtype;         /* 消息的类型 */
    char mtext[1];      /* 消息正文 */
};
```

同管道类似，它有一个不足就是每个消息的最大长度是有上限的，整个消息队列也是长度限制的。

内核为每个IPC对象维护了一个数据结构struct ipc_perm，该数据结构中有指向链表头与链表尾部的指针，保证每一次插入取出都是O(1)的时间复杂度。




### 1. msgget

**功能：**创建或访问一个消息队列

**原型：**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflag);
```

**参数：** key：某个消息队列的名字，用ftok()产生，消息队列为PIC资源，该key标识了此消息队列，如果传入key存在，则返回对应消息队列ID，否则，创建并返回消息队列ID。 msgflag：有两个选项IPC_CREAT和IPC_EXCL，单独使用IPC_CREAT，如果消息队列不存在则创建之，如果存在则打开返回；单独使用IPC_EXCL是没有意义的；两个同时使用，如果消息队列不存在则创建之，如果存在则出错返回。

**返回值**：成功返回一个非负整数，即消息队列的标识码，失败返回-1

**2. msgctl 功能**：消息队列的控制函数

**原型：**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

**参数：** msqid：由msgget函数返回的消息队列标识码 cmd：有三个可选的值，在此我们使用IPC_RMID

- IPC_STAT 把msqid_ds结构中的数据设置为消息队列的当前关联值
- IPC_SET 在进程有足够权限的前提下，把消息队列的当前关联值设置为msqid_ds数据结构中给出的值
- IPC_RMID 删除消息队列

**返回值：** 成功返回0，失败返回-1

**3. msgsnd** **功能：**把一条消息添加到消息队列中

**原型：**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

**参数：** msgid：由msgget函数返回的消息队列标识码 msgp：指针指向准备发送的消息 msgze：msgp指向的消息的长度（不包括消息类型的long int长整型） msgflg：默认为0

**返回值：**成功返回0，失败返回-1

> **4. msgrcv 功能**：是从一个消息队列接受消息
>
> **原型：** ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
>
> **参数：**与msgsnd相同
>
> **返回值：**成功返回实际放到接收缓冲区里去的字符个数，失败返回-1

### 特点

- 与管道不同，消息队列的生命周期随内核，不会随进程销毁而销毁，需要我们显示的调用接口删除或使用命令删除。
- **消息队列可以双向通信。**
- **克服了管道只能承载无格式字节流的缺点。**

## 信号

### 关于信号

一个进程可以发送信号给另一个进程，一个信号就是一条消息，可以用于通知一个进程组发送了某种类型的事件，该进程组中的进程可以采取处理程序处理事件。

| 序号 | 名称 | 默认行为 | 相应事件 |
|---|---|---|---|
| 1 | SIGHUP | 终止 | 终端线挂断 |
| 2 | SIGINT | 终止 | 来自键盘的中断 |
| 3 | SIGQUIT | 终止 | 来自键盘的退出 |
| 4 | SIGILL | 终止 | 非法指令 |
| 5 | SIGTRAP | 终止并转储存储器（1） | 跟踪陷阱 |
| 6 | SIGABRT | 终止并转储存储器（1） | 来自 abort 函数的终止信号 |
| 7 | SIGBUS | 终止 | 总线错误 |
| 8 | SIGFPE | 终止并转储存储器（1） | 浮点异常 |
| 9 | SIGKILL | 终止（2） | 杀死程序 |
| 10 | SIGUSR1 | 终止 | 用户定义的信号 1 |
| 11 | SIGSEGV | 终止并转储存储器（1） | 无效的存储器引用（段故障） |
| 12 | SIGUSR2 | 终止 | 用户定义的信号 2 |
| 13 | SIGPIPE | 终止 | 向一个没有读用户的管道做写操作 |
| 14 | SIGALRM | 终止 | 来自 alarm 函数的定时器信号 |
| 15 | SIGTERM | 终止 | 软件终止信号 |
| 16 | SIGSTKFLT | 终止 | 协处理器上的栈故障 |
| 17 | SIGCHLD | 忽略 | 一个子进程停止或者终止 |
| 18 | SIGCONT | 忽略 | 继续进程如果该进程停止 |
| 19 | SIGSTOP | 停止直到下一个 SIGCONT（2） | 不来自终端的停止信号 |
| 20 | SIGTSTP | 停止直到下一个 SIGCONT | 来自终端的停止信号 |
| 21 | SIGTTIN | 停止直到下一个 SIGCONT | 后台进程从终端读 |
| 22 | SIGTTOU | 停止直到下一个 SIGCONT | 后台进程向终端写 |
| 23 | SIGURG | 忽略 | 套接字上的紧急情况 |
| 24 | SIGXCPU | 终止 | CPU 时间限制超出 |
| 25 | SIGXFSZ | 终止 | 文件大小限制超出 |
| 26 | SIGVTALRM | 终止 | 虚拟定时器期满 |
| 27 | SIGPROF | 终止 | 剖析定时器期满 |
| 28 | SIGWINCH | 忽略 | 窗口大小变化 |
| 29 | SIGIO | 终止 | 在某个描述符上可执行 I/O 操作 |
| 30 | SIGPWR | 终止 | 电源故障 |

图 8-25　Linux 信号

Linux下`unistd.h`头文件下定义了如图中的常量，当你在shell命令行键入`ctrl + c`时，内核就会前台进程组的每一个进程发送`SIGINT`信号，中止进程。

我们可以看到上述只有30个信号，因此操作系统会为每一个进程维护一个int类型变量sig，利用其中30位代表是否有对应信号事件，每一个进程还有一个int类型变量block，与sig对应，其30位表示是否堵塞对应信号(不调用处理程序)。如果存在多个相同的信号同时到来，多余信号正常情况下会被存储在一个等待队列中等待。

我们要理解进程组是什么，每个进程属于一个进程组，可以有多个进程属于同一个组。每个进程拥有一个进程ID，称为`pid`，而每个进程组拥有一个进程组ID，称为`pgid`，默认情况下，一个进程与其子进程属于同一进程组。

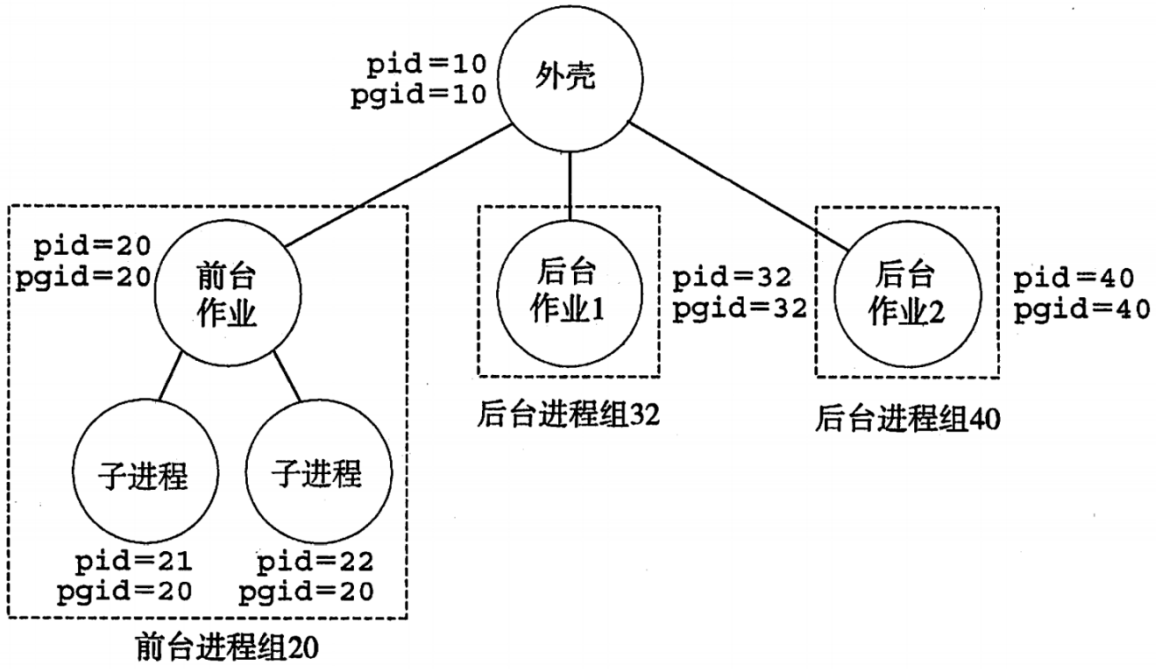



图 8-27　前台和后台进程组

软件方面(诸如检测键盘输入是硬件方面)可以利用kill函数发送信号，kill函数接受两个参数，进程ID和信号类型，它将该信号类型发送到对应进程，如果该pid为0，那么会发送到属于自身进程组的所有进程。

接收方可以采用signal函数给对应事件添加处理程序，一旦事件发生，如果未被堵塞，则调用该处理程序。

Linux下有一套完善的函数用以处理信号机制。

### 特点

- **信号是在软件层次上对中断机制的一种模拟，是一种异步通信方式。**
- 信号可以直接进行用户空间进程和内核进程之间的交互，内核进程也可以利用它来通知用户空间进程发生了哪些系统事件。
- 如果该进程当前并未处于执行态，则该信号就由内核保存起来，直到该进程恢复执行再传递给它；如果一个信号被进程设置为阻塞，则该信号的传递被延迟，直到其阻塞被取消时才被传递给进程。
- 信号有明确生命周期，首先产生信号，然后内核存储信号直到可以发送它，最后内核一旦有空闲，会适当处理信号。
- 处理程序是可以被另一个处理程序中断的，因此这可能造成并发问题，所以**在处理程序中的代码应该是线程安全的，**通常通过设置block位图以堵塞所有信号。

## 套接字

Socket套接字是用与网络中不同主机的通信方式，多用于客户端与服务器之间，在Linux下也有一系列C语言函数，诸如socket、connect、bind、listen与accept，对于原理的

学习，更好的是对Java中的套接字socket源码进行剖析。

## 结语

对于工作而言，我们可能一辈子都用不上这些操作，但作为对于操作系统的学习，认识到进程间是如何通信还是很有必要的。

面试的时候对于这些方法我们不需要掌握到很深的程度，但我们必须要讲的来有什么通信方式，这些方式都有什么特点，适用于什么条件，大致是如何操作的，能说出这些，基本足以让面试官对你十分满意了。