进程通信

Linux
Android
Linux
OpenStack
Mac OS
Windows

进程通信

本讲内容

1. 进程通信概念与类型
2. 低级通信之信号通信
3. 高级通信之共享存储
4. 高级通信之消息通信
5. 高级通信之管道通信

进程通信概念与类型

1 基本概念

- 进程间通信IPC是指进程在系统里同时运行，并相互传递、交换信息
- 通过进程通信能够实现数据传输、共享数据、通知事件、资源共享、进程控制
- 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

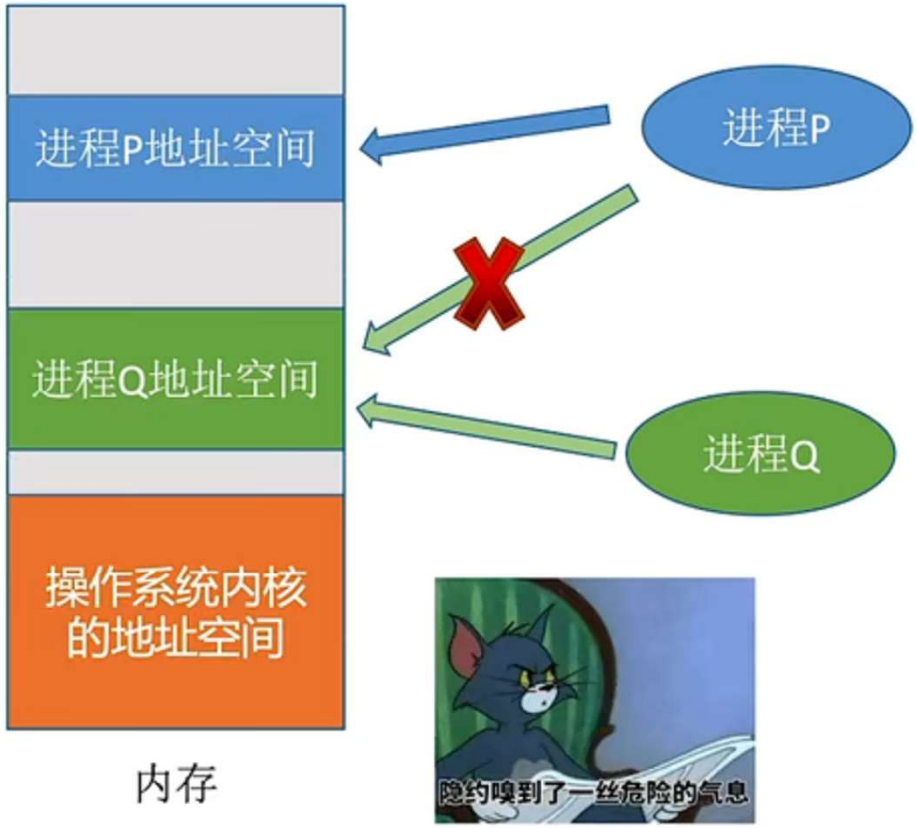
什么是进程间通信？

进程间通信（Inter-Process Communication， IPC）是指两个进程之间产生数据交互。

分享吃瓜文

为什么进程通信需要操作系统支持？

进程是分配系统资源的单位（包括内存地址空间），因此各进程拥有的内存地址空间相互独立。



为了保证安全，一个进程不能直接访问另一个进程的地址空间。

进程通信概念与类型

2 通信类型

低级通信

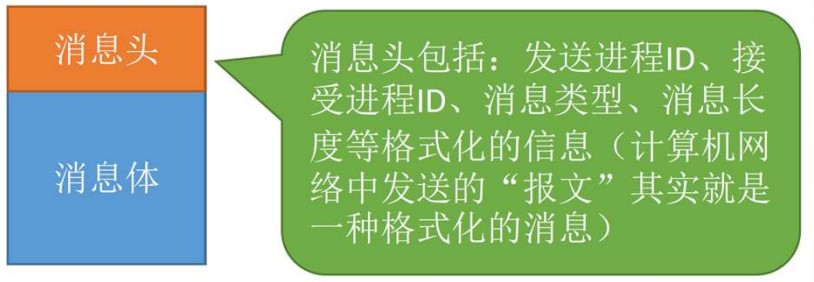
高级通信

进程通信概念与类型

3 通信方式

数据格式

- 字节格式：接收方不保留各次发送之间的分界
- 报文格式：接收方保留各次发送之间的分界
  分成定长报文/不定长报文和可靠报文/不可靠报文

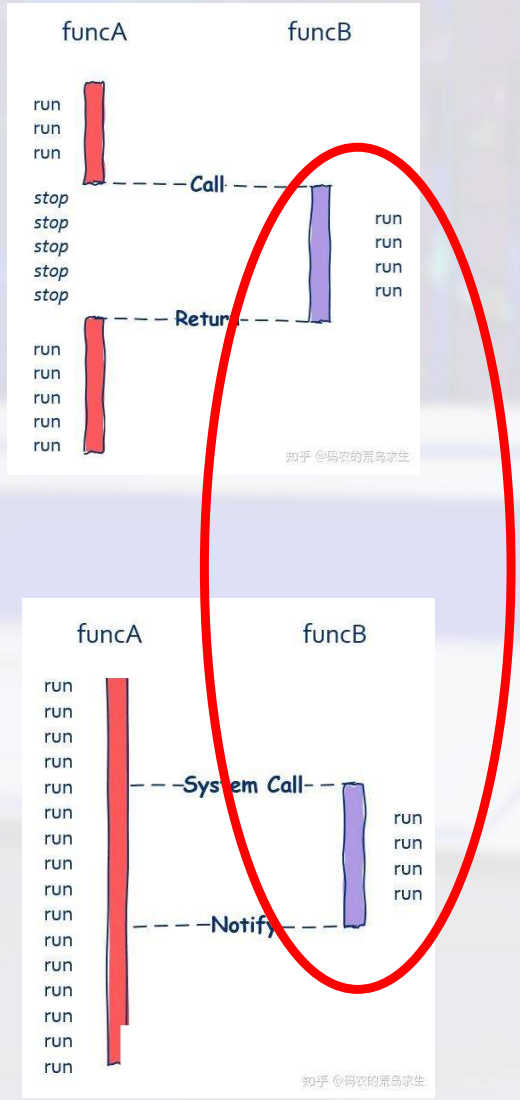


同步方式

- 阻塞操作：指操作方要等待操作结束
- 不阻塞操作：指操作提交后立即返回

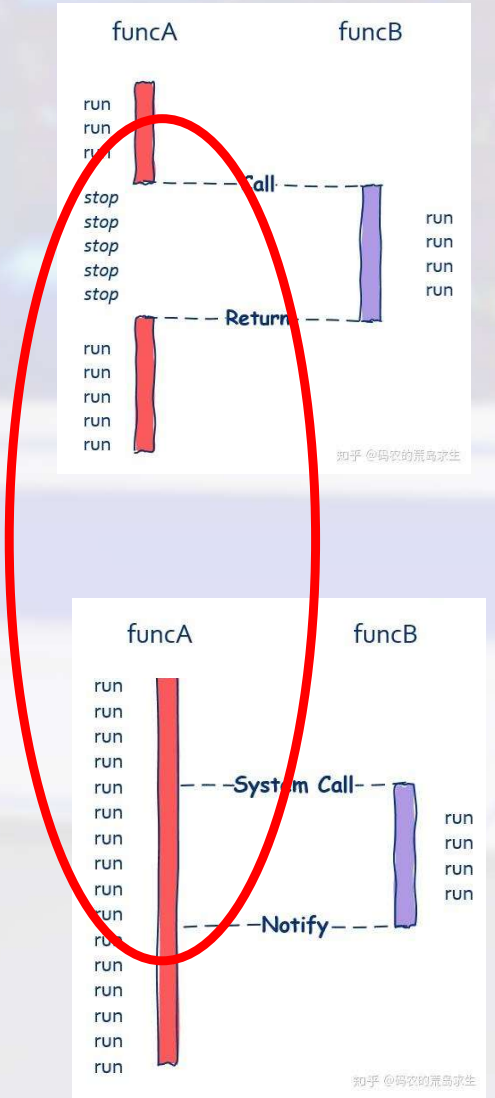
进程同步与异步

- “进程同步与异步”则是指在并发系统中，多个进程之间的消息通信机制
- 取决于被带调用的进程，不主动(同步)/主动(异步)通知调用进程
- 同步：调用进程就需要每隔一定时间检查一次，效率就很低
- 异步：使用通知和回调的方式，效率则很高



进程阻塞与非阻塞

- 阻塞与非阻塞的重点在于进程等待消息时候的行为，关注的是发起并行调度的主调进程
- 调用者在等待消息的时候，进程是挂起状态，还是非挂起状态
- 阻塞：调用在发出去后，在消息返回之前，当前程会被挂起，直到有消息返回
- 非阻塞：调用在发出去后，不会阻塞当前进程，而会立即返回



进程阻塞与非阻塞

例 老张爱喝茶，煮开水

- 出场人物：老张，水壶两把（普通水壶，简称水壶；会响的水壶，简称响水壶）
- 老张把水壶放到火上，立等水开（同步阻塞）
- 老张觉得自己有点傻
- 老张把水壶放到火上，去客厅看电视，时不时去厨房看看水开没有（同步非阻塞）

进程阻塞与非阻塞

例

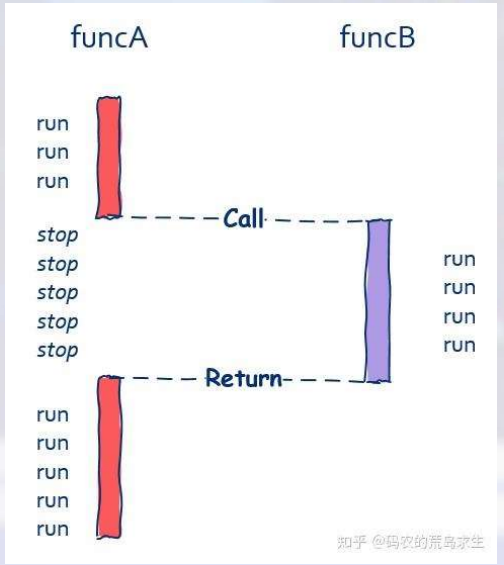
- 老张还是觉得自己有点傻，于是买了把会响笛的那种水壶，水开之后，能大声发出嘀~~~~的噪音
- 老张把响水壶放到火上，立等水开（异步阻塞）
- 老张觉得这样傻等意义不大，老张把响水壶放到火上，去客厅看电视，水壶响之前不再去看它了，响了再去拿壶（异步非阻塞）

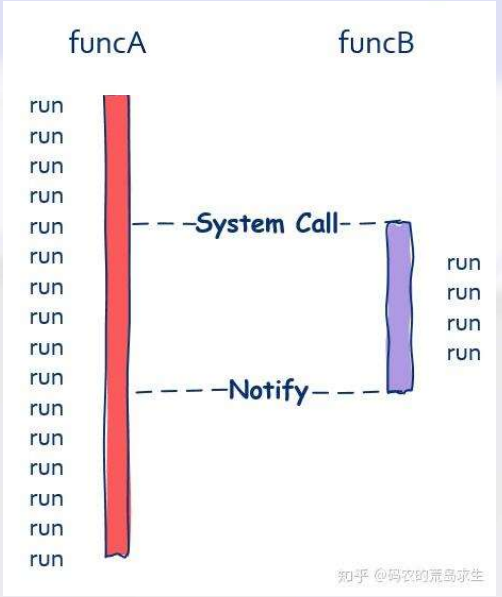
进程阻塞与非阻塞

例

- 调用进程，决定了是阻塞还是非阻塞；如果调用进程一直等，那就是阻塞，不等就是非阻塞

- 被调用进程，决定了是同步还是异步；有结果时再返回则同步，立马返回无结果，结果另行通知则异步





进程通信

本讲内容

1. 进程通信概念与类型
2. 低级通信之信号通信
3. 高级通信之共享存储
4. 高级通信之消息通信
5. 高级通信之管道通信

低级通信之信号通信

1 机制原理

每一个正整数，代表一个特定的信号

- 每个信号都对应正整数常量，即信号编号
- 进程之间传送事先约定的信息的类型，用于通知进程发生了某异常事件
- 进程通过信号机制来检查是否有信号。若有，中断正在执行的程序，转向对应的处理程序；结束后返回到断点继续执行，这是一种软中断

低级通信之信号通信

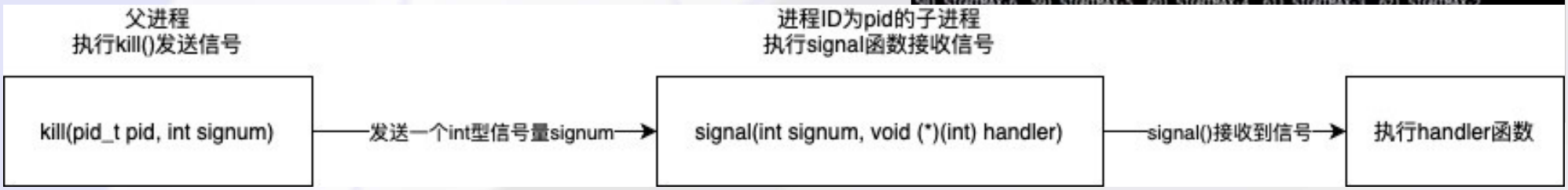
2 信号收发

Linux中，调用kill( ) 实现一个进程向另一个进程发送信号
因为大多数信号都是把另一个进程杀死，所以kill( )

- 发送信号：发送信号的程序用系统调用kill( )实现
- 接收信号：为每个进程内嵌相应的接收函数
- 接收之后：预置信号处理，接收信号的程序用signal( )

  来实现对处理方式的调用
  - 接收信号的进程按事先规定完成对事

```
root@ubuntu:~# ls | grep 1
1.txt
root@ubuntu:~# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
```



低级通信之信号通信

3 实例

- 创建两个子进程，父进程等着用户在键盘上，按下ctrl+c两个键的中断信号
- 一旦捕捉到中断信号^c后，父进程用系统调用kill( )向两个子进程发出信号，子进程捕捉到信号后分别输出下列信息后终止：

Child process1 is killed by parent!

Child process2 is killed by parent!

- 父进程等两个子进程终止后，输出如下的信息后终止：

Parent process is killed!

父子进程之间的信号传递

低级通信之信号通信

3 实例

1、kill( )

int kill(pid, sig)

pid是接收进程的标识符，参数sig是要发送的软中断信号

（1）pid>0时，发送信号给具有该 pid 的单个进程

（2）pid=0时，信号发送给与发送进程同组的所有进程

（3）pid=-1时，表示发送信号给除了调用进程和 init 进程（pid 为 1）以外的所有进程

（所有用户空间的进程均派生自ini进程）

低级通信之信号通信

3 实例

2、signal( )

signal(sig,function)

头文件为 #include <signal.h>

参数定义

int sig;

void (*func) ( )

捕捉信号sig后，执行function规定的操作

低级通信之信号通信

值	名 字	说 明
01	SIGHUP	挂起
02	SIGINT	中断，当用户从键盘按^c键或^break键时
03	SIGQUIT	退出，当用户从键盘按quit键时
04	SIGILL	非法指令
05	SIGTRAP	跟踪陷阱，启动进程，跟踪代码的执行
06	SIGIOT	IOT指令
07	SIGEMT	EMT指令
08	SIGFPE	浮点运算溢出
09	SIGKILL	杀死、终止进程
10	SIGBUS	总线错误
11	SIGSEGV	段违例，进程试图去访问其虚地址空间以外的位置
12	SIGSYS	系统调用中参数错，如系统调用号非法
13	SIGPIPE	向某个非读管道中写入数据
14	SIGALRM	闹钟。当某进程希望在某时间后接收信号时发此信号
15	SIGTERM	软件终止
16	SIGUSR1	用户自定义信号1
17	SIGUSR2	用户自定义信号2
18	SIGCLD	某个子进程死
19	SIGPWR	电源故障

低级通信之信号通信

3 实例

2、signal( )

signal(sig,function)

function 的解释如下：

（1）function=1时，对信号不予理睬，屏蔽该类信号

（2）function=0时，进程在收到sig信号后应终止自己

（3）function≠0，　≠1时，作为指向处理函数的指针

进程的创建

3 Linux进程创建

实例

fork( )返回值意义

- 0：在子进程中，pid变量保存的fork( )返回值为0，表示当前进程是子进程
- >0：在父进程中，pid变量保存的fork( )返回值为子进程的id值（进程唯一标识符）
- -1：创建失败

低级通信之信号通信

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
void waiting( ),stop( );
int wait_mark;

main( ) {
 int p1,p2,stdout;
 signal(SIGINT,SIG_IGN);      /*防止control-C 键盘中断*/
 while((p1=fork( ))==-1);      /*创建子进程p1*/
 if (p1>0) {
  while((p2=fork( ))==-1);      /*创建子进程p2 */
  if(p2>0) {
   wait_mark=1;
   signal(SIGINT,stop);        /*接收到^c信号，转stop*/
   waiting( );
   kill(p1,16);                /*向p1发软中断信号16*/
   kill(p2,17);                /*向p2发软中断信号17*/
   wait(0);                    /*同步*/
   wait(0);
   printf("Parent process is killed!\n");
   exit(0);
  } else {                    // if(p2>0)
   wait_mark=1;
   signal(17,stop);  /*接收到软中断信号17，转stop*/
   waiting( );
   printf("Child process 2 is killed by parent!\n");
   exit(0);
  }
 } else {                    // if(p1>0)
  wait_mark=1;
  signal(16,stop);                /*接收到软中断信号16，转stop*/
  waiting( );
  printf("Child process 1 is killed by parent!\n");
  exit(0);
 }
}

void waiting( ) {
 while(wait_mark!=0);
}

void stop( ) {
 wait_mark=0;
}
```

SIG_IGN 忽视信号

在父进程中运行

在父进程中运行

在P2进程中运行

在P1进程中运行

低级通信之信号通信

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
void waiting( ),stop( );
int wait_mark;

main( ) {
 int p1,p2,stdout;
 signal(SIGINT,SIG_IGN);      /*防止control-C 键盘中断*/
 while((p1=fork( ))==-1);      /*创建子进程p1*/
 if (p1>0) {
  while((p2=fork( ))==-1);      /*创建子进程p2 */
  if(p2>0) {
   wait_mark=1;
   signal(SIGINT,stop);          /*接收到^c信号，转stop*/
   waiting( );
   kill(p1,16);                  /*向p1发软中断信号16*/
   kill(p2,17);                  /*向p2发软中断信号17*/
   wait(0);                      /*同步*/
   wait(0);
   printf("Parent process is killed!\n");
   exit(0);
```

在父进程中运行

在父进程中运行

```
  } else {              // if(p2=0)
   wait_mark=1;
   signal(17,stop);  /*接收到软中断信号17，转stop*/
   waiting( );
   printf("Child process 2 is killed by parent!\n");
   exit(0);
  }
 } else {              // if(p1=0)
  wait_mark=1;
  signal(16,stop);              /*接收到软中断信号16，转stop*/
  waiting( );
  printf("Child process 1 is killed by parent!\n");
  exit(0);
 }
}

void waiting( ) {
 while(wait_mark!=0);
}

void stop( ) {
 wait_mark=0;
}
```

在P2进程中运行

在P1进程中运行

进程通信

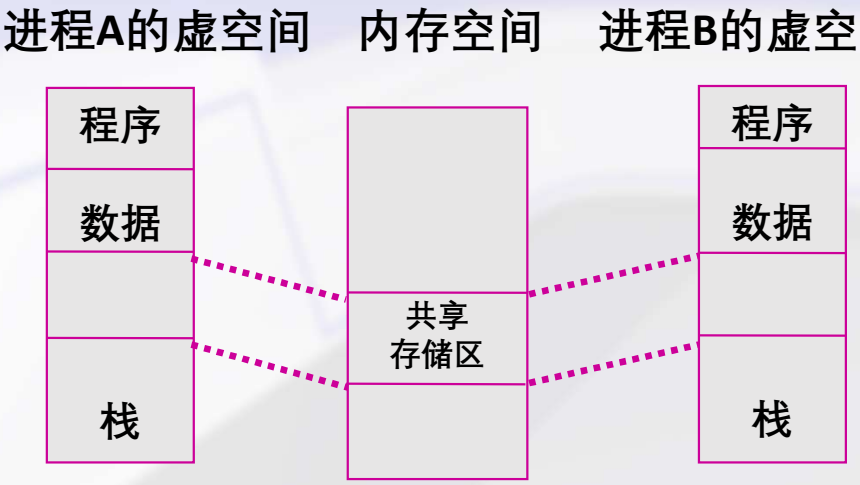
本讲内容

1. 进程通信概念与类型
2. 低级通信之信号通信
3. 高级通信之共享存储
4. 高级通信之消息通信
5. 高级通信之管道通信

高级通信之共享存储

1 机制原理

- 利用内存中间的一个共享存储区，实现两个进程之间的通信
- 把进程间通信转换成了对于内存中数据的读写操作，效率是最高的通信机制
- A进程往共享存储区中写，另一个进程B从共享存储区中读取



高级通信之共享存储

2 函数调用

基于一些函数调用，通过共享存储区两个进程可以交换大批量数据

1、shmget( )

创建、获得一个共享存储区。

系统调用格式：　shmid=shmget(key,size,flag)

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

第一进程调用 shmget进行了创建，另一个进程再次调用shmget就是获得刚刚创建的共享存储区，不会重复创建

高级通信之共享存储

2 函数调用

正整数常量用来表示这个存储区的名字

1、shmget( )

创建、获得一个共享存储区

系统调用格式： shmid=shmget(key,size,flag)

- key是共享存储区的名字
- size是其大小（以字节计）
- flag是用户设置的标志
- 返回shmid值提供给其他函数使用

flag 就是表示创建、还是获得

高级通信之共享存储

2 函数调用

共享存储区附接之后，用户就可以像使用自己的空间一样去使用这个虚拟地址空间

shmid=shmget(key,size,flag)

2、shmat( )

共享存储区附接，将共享存储区附接进程虚拟地址空间

系统调用格式： virtaddr=shmat(shmid,addr,flag)

- shmid是共享存储区的标识符
- addr是用户给定，将共享存储区附接到进程的虚地址空间
- flag规定读、写权限，值为0时，表示可读、可写
- 返回值是共享存储区所附接到的进程虚地址viraddr

后面进程就可以对这个虚地址viraddr进行操作和赋值了

高级通信之共享存储

2 函数调用

3、shmdt( )

把共享存储区从进程虚地址空间断开

系统调用格式：　　shmdt(addr)

- addr是要断开连接的虚地址，即由shmat( )所返回的虚地址
- 调用成功，返回0值，调用不成功，返回-1

virtaddr=shmat(shmid,addr,flag)

高级通信之共享存储

2 函数调用

4、shmctl( )

共享存储区的控制，对其状态进行读取和修改

系统调用格式：　　shmctl(shmid,cmd,buf)

shmid是共享存储区的标识符，　cmd操作命令，　buf是用户缓冲区地址。

高级通信之共享存储

2 函数调用

4、shmctl( ) 共享存储区控制，对其状态进行读取和修改

系统调用格式：　　shmctl(shmid,cmd,buf)

cmd操作命令用法如下：

- 用于查询共享存储区的情况，如长度、连接进程数、共享区的创建者标识符等
- 用于设置或改变共享存储区的属性，如共享存储区的许可权、连接进程计数等；
- 共享存储区的加锁和解锁，删除共享存储区标识符等。

高级通信之共享存储

3 实例

- 进程利用fork()创建两个子进程server和client进行通信(server负责收，client负责发)。
- client端建立或打开一个key为75的共享区，client填入9到0，client每发送一次数据后显示“(client)sent”。
- server端建立或打开一个key为75的共享区，等待其他进程发来的消息，server每接收到一次数据后显示“(server)received”。

client发送9，server接受9取走，然后向缓冲区写一个-1；
收到-1之后client再发送8，server接受8……以此类推，直到等于0停止。

高级通信之共享存储

```
main( )
{
  while ((i=fork( ))= =-1);
  if (!i) server( );
  system(“ipcs  -m”);
  while ((i=fork( ))= =-1);
  if (!i) client( );
  wait(0);
  wait(0);
}
```

第一个fork创建了server

第二个fork创建了client，然后让通信发生

高级通信之共享存储

server实现接收消息

```
void  server( )
{int x;
shmid=shmget(SHMKEY,1024,0777|IPC_CREAT); /*创建共享存储区*/
addr=shmat(shmid,0,0);                                    /*获取首地址*/
do
 {
   *addr = -1;
    while (*addr == -1);
    x=*addr
    printf("(server) received\n");
}while (*addr);
shmctl(shmid,IPC_RMID,0);                  /*撤消共享存储区，归还资源*/
exit(0);
}
```

创建、获得一个共享存储区名为75，1k字节大小

将共享存储区附接进程虚拟地址空间

先把共享存储区置为空
循环等待，一直到收到数据
Client发的数据和-1都接收

当client发送最后一个值0 (*addr == 0) 跳出循环

高级通信之共享存储

client实现发送消息

```
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#define  SHMKEY  75
int  shmid,i;   int  *addr;
void  client( )
{  int i;
shmid=shmget(SHMKEY,1024,0777 |IPC_CREAT);                    /*打开共享存储区*/
addr=shmat(shmid,0,0);                                        存储区首地址*/
for (i=9;i>=0;i--)
  {  while (*addr! = -1);
     printf("(client) sent\n");
     *addr=i;
  }
exit(0);
}
```

shmget: server先创建75共享存储区，client就打开

shmat: 把共享存储区附接到client的虚地址空间

如果空间地址中值不是-1；说明上一轮发送的数据还未被server取走，client就空转等待

当*addr == -1说明，已经被取走，就打印提示并向地址写入新的数值，直到 i == 0

进程通信

本讲内容

1. 进程通信概念与类型
2. 低级通信之信号通信
3. 高级通信之共享存储
4. 高级通信之消息通信
5. 高级通信之管道通信

高级通信之消息通信

1 机制原理

- 消息是一个格式化的可变长信息单元
- 消息通信机制允许由进程给其他进程发送消息
- 进程收到多个消息时，可排成消息队列
- 消息队列有消息队列描述符方便用户和系统访问

高级通信之消息通信

1 机制原理

接收进程R通过指针,从消息队列找到消息读取

发送进程S可以发送源语,向指定的消息队列发送

高级通信之消息通信

2 函数调用

- msgget( )：创建一个消息，获得消息的描述符
- msgsnd（）：向指定的消息队列发送一个消息，并将该消息链接到该消息队列的尾部
- msgrcv( )：从指定的消息队列中接收消息
- msgctl( )：读取消息队列的状态并进行修改，如查询消息队列描述符、修改许可权及删除该队列等

进程通信

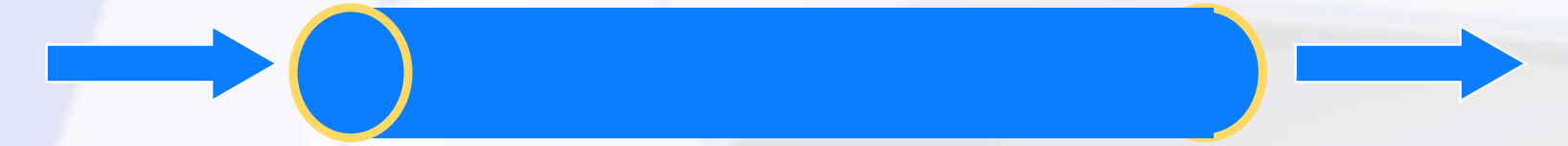
本讲内容

1. 进程通信概念与类型
2. 低级通信之信号通信
3. 高级通信之共享存储
4. 高级通信之消息通信
5. 高级通信之管道通信

高级通信之管道通信

1 机制原理

- 管道允许一个进程与另一个进程进行数据传递。管道可被视为一段内存或者buffer
- 管道是连接写进程和读进程的、并允许以生产者-消费者方式进行通信的硬盘上的共享文件，称为pipe文件(管道文件)

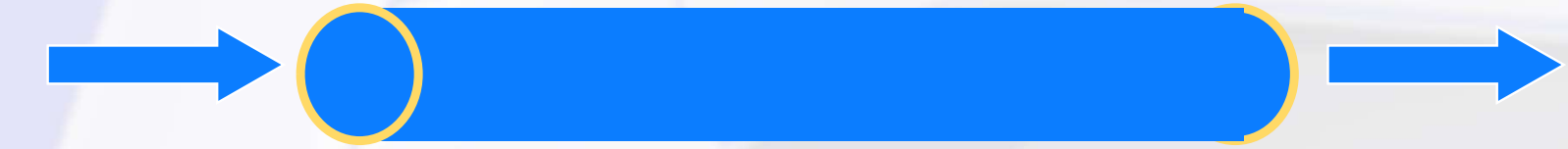


写入读出先进先出

高级通信之管道通信

1 机制原理

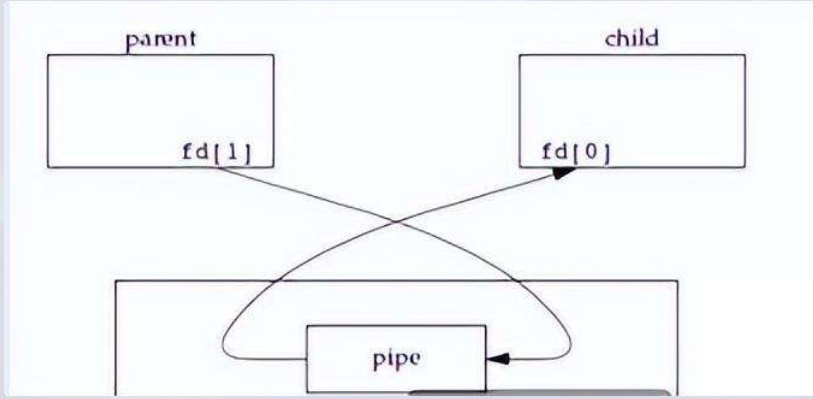
- 像现实生活中的管道，水可以从一端进入并从另一端流出。在操作系统的上下文中，水比作数据流
- 写进程从管道的写入端将数据写入管道，而读进程则从管道的读出端读出数据



写入读出先进先出

高级通信之管道通信

2 管道类型



- 无名管道：无名管道是一种最基本的管道通信形式，它通常用于有亲缘关系的进程通信，比如父子/兄弟进程。这种通信管道在创建它的进程及其子进程中是可见的，但在其他进程中是隐藏的。因此它只能用于相对私有的通信场景

  - 利用pipe( )建立起来临时的无名文件
  - 生命周期通常随进程结束而结束
  - 只支持单向数据流，需要两个管道实现双向通信

高级通信之管道通信

2 管道类型

- 有名管道：不像无名管道那样受限于创建它们的进程。有名管道通过文件系统中的一个名字来识别和访问，因此，任何拥有适当访问权限的进程都可以通过这个名字来访问管道，实现不同进程间的通信

  - 在文件系统中长期存在的、具有路径名的文件，用系统调用mknod( )建立
  - 可由非亲缘关系的进程之间共享和使用
  - 生命周期独立于创建它的进程

在这个例子中，ll命令的输出成为了grep命令的输入

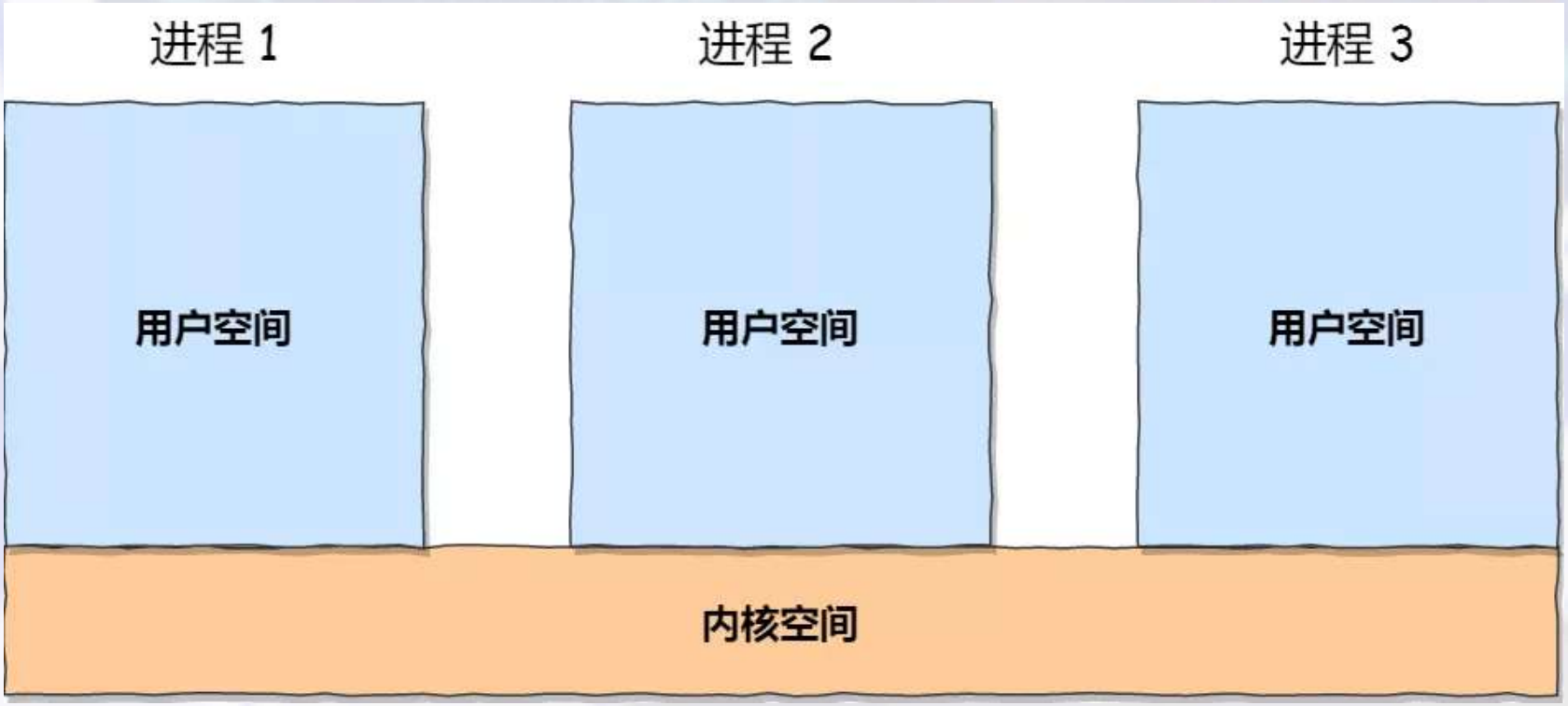
```
1 $ ll | grep ooxx
2 -rw-rw-r-- 1 vagrant vagrant    7 Dec 21 06:07 ooxx.txt
```

进程通信

本讲内容

1. 进程通信概念与类型
2. 低级通信之信号通信
3. 高级通信之共享存储
4. 高级通信之消息通信
5. 高级通信之管道通信

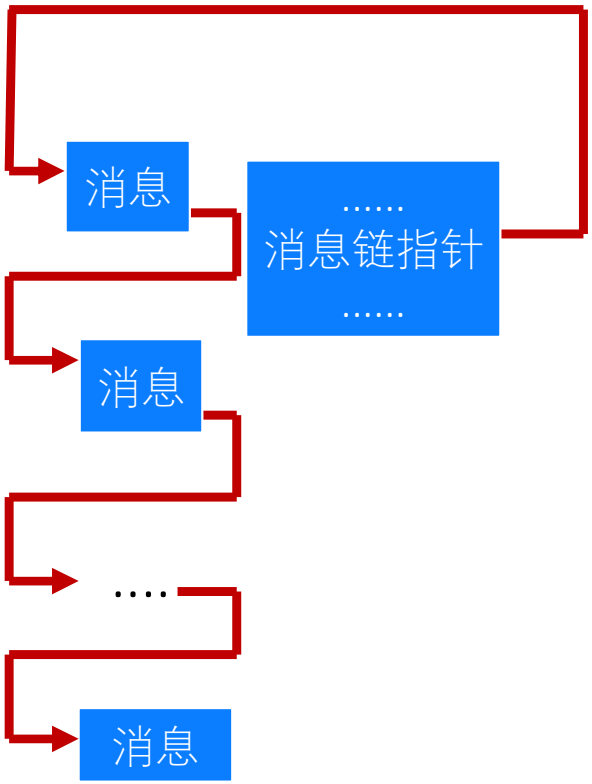
进程通信

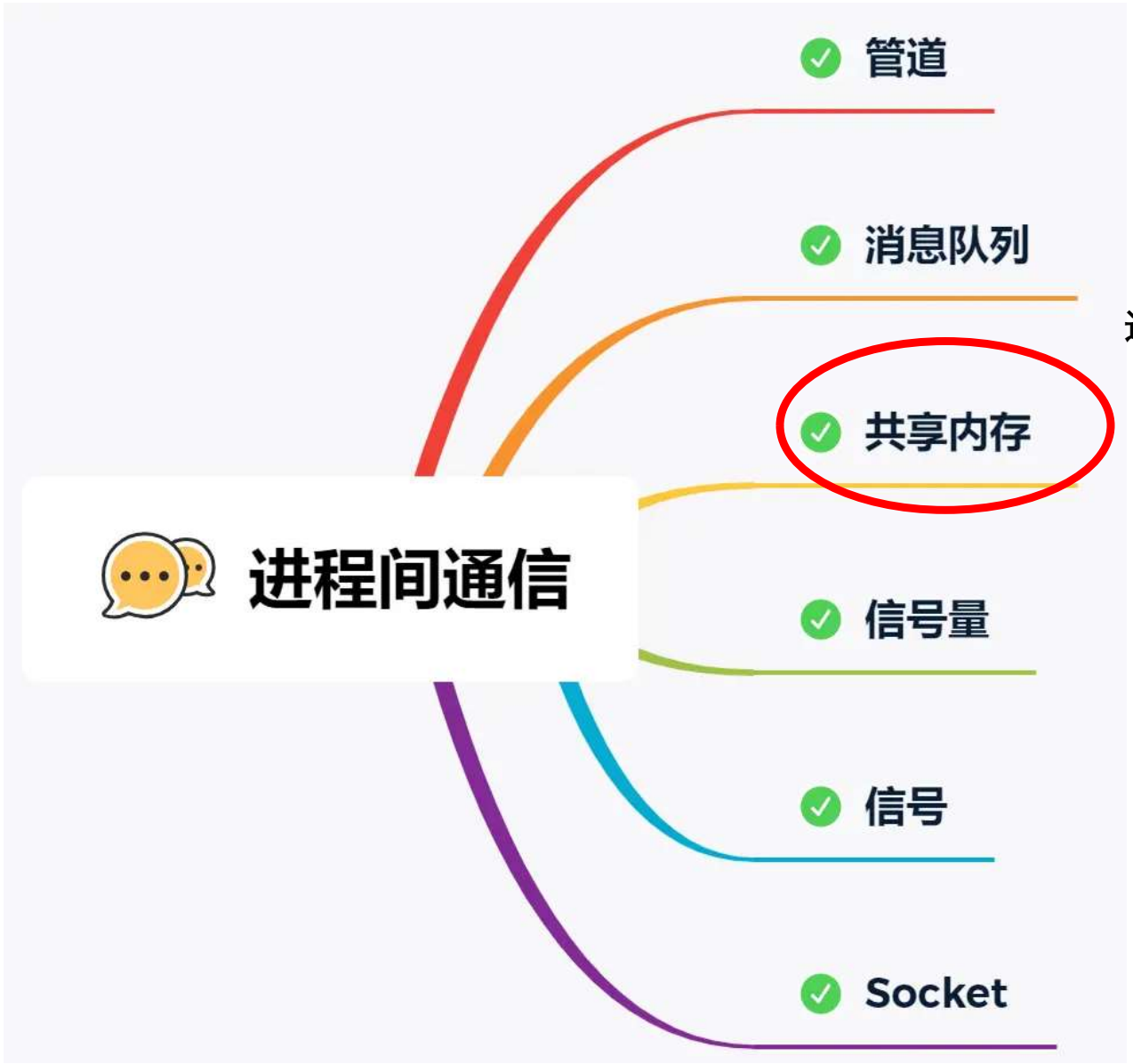


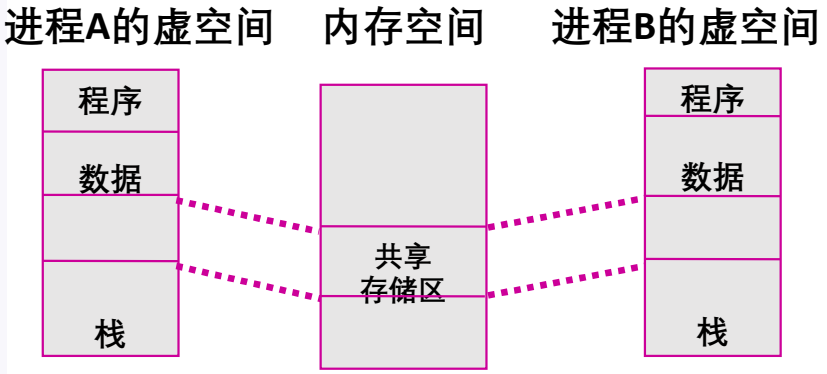

进程间通信
管道
消息队列
共享内存
信号量
信号
Socket

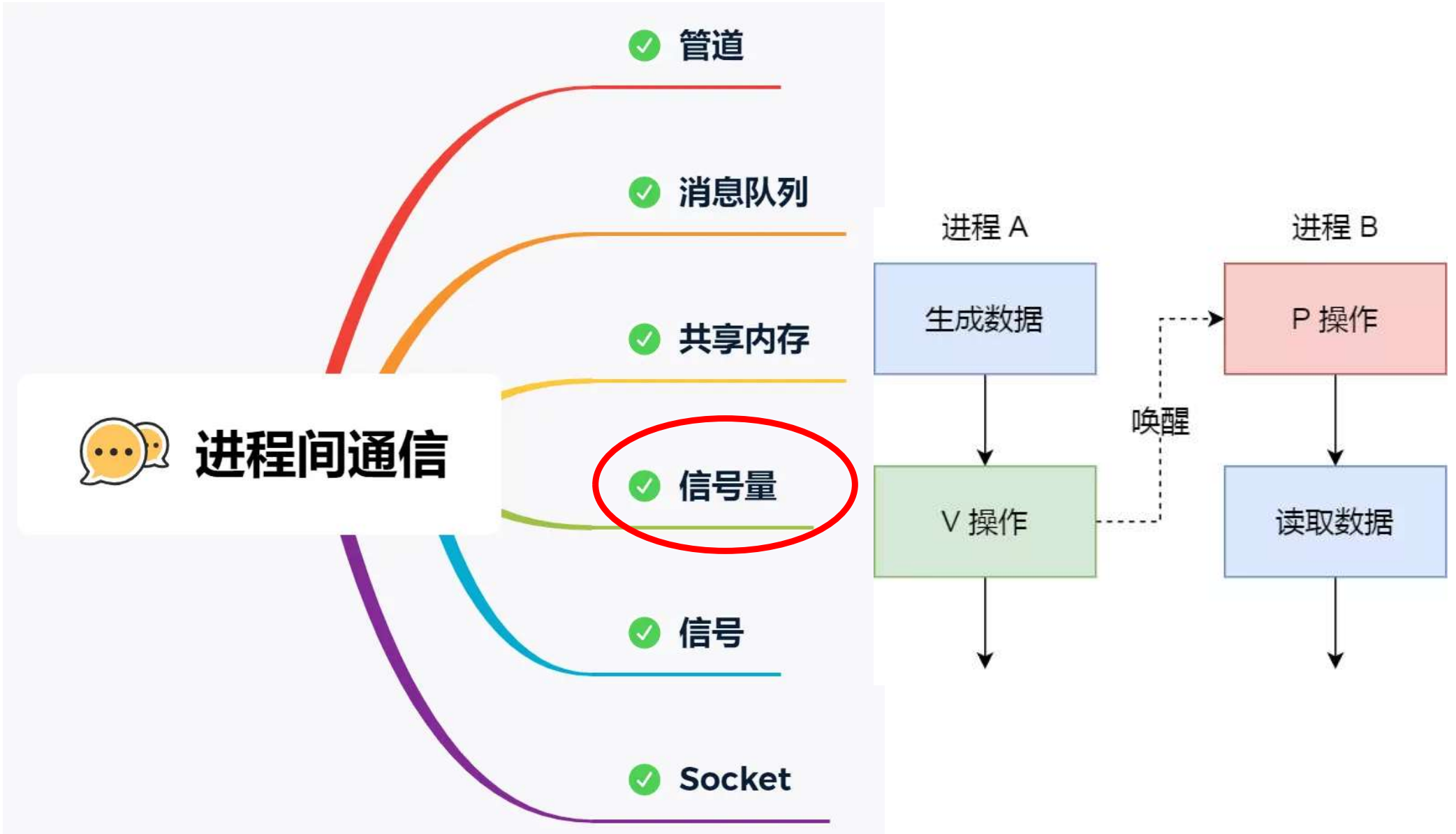

发送进程
接收进程


进程间通信
管道
消息队列
共享内存
信号量
信号
Socket


消息
.....
消息链指针
.....
消息
....
消息


进程间通信
管道
消息队列
共享内存
信号量
信号
Socket

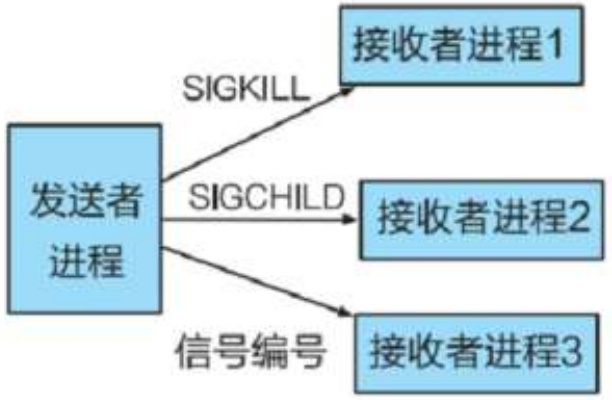

进程A的虚空间
内存空间
进程B的虚空间
程序
数据
栈
共享
存储区
程序
数据
栈


管道
消息队列
共享内存
信号量
信号
Socket
进程间通信
进程 A
进程 B
生成数据
P 操作
唤醒
V 操作
读取数据

进程间通信

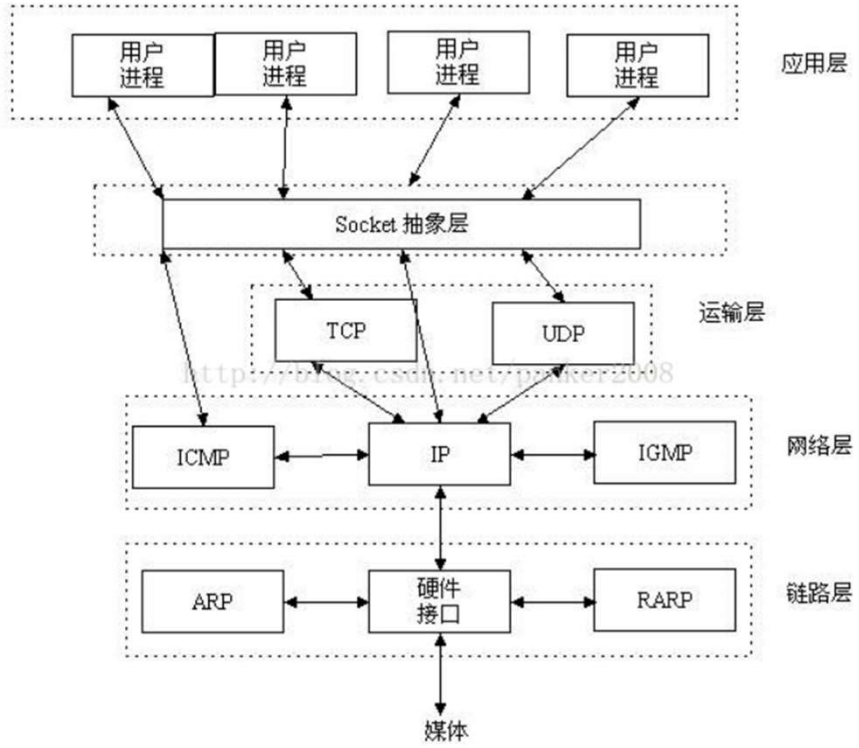
- 管道
- 消息队列
- 共享内存
- 信号量
- 信号
- Socket

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```



进程间通信

- 管道
- 消息队列
- 共享内存
- 信号量
- 信号
- Socket



进程通信

Linux
Android
Linux
OpenStack
Mac OS
Windows

1、设与某资源关联的信号量初值为3，当前值为-1。若M表示该资源的可用个数，N表示等待该资源的进程数，则M是 0 ，N是 1 。

1. Linux中的进程通信机制有信号通信机制、 共享存储区 、 消息通信机制 、和管道通信机制。

1. 设与某资源R关联的信号量初值为5，当前值为-2，下列说法错误的是 C 。

A．系统初始状态有5个可用R资源
B．当前可用R资源数目为0
C．当前状态下执行P操作的进程不会陷入阻塞态
D．当前等待使用R资源的进程数目为2

9、设某个信号量的S的初值为5。若执行某个V(S)时，发现（ A ）时，则唤醒相应等待队列中等待的一个进程。

A、S的值小于等于0　　B、S的值大于等于5

C、S的值小于5　　D、S的值大于5