1.Linux相关

1.常用指令

1.1 查看磁盘使用情况

1.2 查看具体文件夹大小

1.3 查看内存使用情况(相当于任务管理器)

1.4 查看进程

1.5 查看当前路径

1.6 网络相关指令

1.7 查看日志

1.8 awk命令

1.9 其他常用指令

2.Linux其他

3.source sh ./ bash 区别

二.操作系统相关

1.死锁

如何避免死锁的发生

2.线程和进程

2.1线程和进程的区别和关系

3.进程之间的通信方式以及优缺点?

2.3 多线程和单线程

2.4 线程的状态

2.5 线程安全

2.6 进程状态

1.Linux相关

1.常用指令

1.1 查看磁盘使用情况

df -h

```
root@izbp1hx8v6lzadot5plxh2z:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           799M  3.3M  796M   1% /run
/dev/vda1        40G   11G   27G  29% /
tmpfs           3.9G  4.0K  3.9G   1% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/vdb1       197G  357M  187G   1% /mnt/data1
tmpfs           799M     0  799M   0% /run/user/0
tmpfs           799M     0  799M   0% /run/user/1001
root@izbp1hx8v6lzadot5plxh2z:/#
```

1.2 查看具体文件夹大小

du -sh /path

```
[root@VM_0_6_centos ~]# du -sh /root
120M    /root
[root@VM_0_6_centos ~]#
```

1.3 查看内存使用情况(相当于任务管理器)

top

1.4 查看进程

ps -a //查看所有进程

ps -ef   //用标准的格式显示进程的(显示cmd)

ps -ef|grep xxxx //查看指定进程,一般查看pid,配合kill杀掉进程

1.5 查看当前路径

pwd

1.6 网络相关指令

ifconfig //查看网络情况
ifconfig -a

netstat -a //查看当前所有网络进程
netstat -nap|grep xxx 查看特定端口号的进程

1.7 查看日志

tail -f xxx //动态查看日志末尾
tail -100 xxx //查看文档最后100行
head -n 100 xxx//查看文档前100行

1.8 awk命令

awk '{print $1,$4}' xxx.log #默认使用空格和Tab分隔符
awk -F, '{print $1,$4}' xxx.log #-F指定分隔符

1.9 其他常用指令

cat xxx # 查看文件的内容
more xxxx # 查看文件的内容
chmod 744 xxxx //修改文件或者目录访问权限

3.source sh ./ bash 区别

1. source

在当前shell内去读取、执行a.sh，而a.sh不需要有"执行权限"

2. sh/bash

都是打开一个subshell去读取、执行a.sh，而a.sh不需要有"执行权限"

3 ./

打开一个subshell去读取、执行a.sh，但a.sh需要有"执行权限"

二.操作系统相关

1.死锁

死锁的起因是多个线程之间相互等待对方而被永远暂停（处于非Runnable）。

死锁的产生必须满足如下四个必要条件：

- **资源互斥：一个资源每次只能被一个线程使用**
- **请求与保持条件：一个线程因请求资源而阻塞时，对已获得的资源保持不放**
- **不剥夺条件：线程已经获得的资源，在未使用完之前，不能强行剥夺**
- **循环等待条件：若干线程之间形成一种头尾相接的循环等待资源关系**

如何避免死锁的发生

- **粗锁法：使用一个粒度粗的锁来消除“请求与保持条件”，缺点是会明显降低程序的并发性能并且会导致资源的浪费。**
- **锁排序法：（必须回答出来的点）**

1. 如果必须在不同的操作中控制多个锁，试图在所有方法中以相同的顺序锁定它们。
2. 然后，按照相反的顺序释放锁，并将锁及其解锁封装在一个类中。这样就不会在整个代码中分布与同步相关的代码。

https://bbs.csdn.net/topics/250018320

https://blog.csdn.net/nicolastsuei/article/details/84747125

指定获取锁的顺序，比如某个线程只有获得A锁和B锁，才能对某资源进行操作，在多线程条件下，如何避免死锁？

通过指定锁的获取顺序，比如规定，只有获得A锁的线程才有资格获取B锁，按顺序获取锁就可以避免死锁。这通常被认为是解决死锁很好的一种方法。

```
public static void main(String[] args) {
    final Object a = new Object();
    final Object b = new Object();
    Thread threadA = new Thread(new Runnable() {
        public void run() {
            synchronized (a) {
                try {
                    System.out.println("now i in threadA-locka");
                    Thread.sleep(1000l);
                    synchronized (b) {
                        System.out.println("now i in threadA-lockb");
                    }
                } catch (Exception e) {
                    // ignore
                }
            }
        }
    });

    Thread threadB = new Thread(new Runnable() {
        public void run() {
            synchronized (b) {
                try {
                    System.out.println("now i in threadB-lockb");
                    Thread.sleep(1000l);
                    synchronized (a) {
                        System.out.println("now i in threadB-locka");
                    }
                } catch (Exception e) {
                    // ignore
                }
            }
        }
    });

    threadA.start();
    threadB.start();
}
```

2.线程和进程

2.1线程和进程的区别和关系

1. 进程是资源分配的最小单位，线程是程序执行的最小单位（资源调度的最小单位）

2. 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据的，使用相同的地址空间，因此CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。

3. 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。

4. 但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。

3.进程之间的通信方式以及优缺点?

https://www.jianshu.com/p/c1015f5ffa74

1)管道

管道分为有名管道和无名管道

无名管道是一种半双工的通信方式,数据只能单向流动,而且只能在具有亲缘关系的进程间使用.进程的亲缘关系一般指的是父子关系。无明管道一般用于两个不同进程之间的通信。当一个进程创建了一个管道,并调用fork创建自己的一个子进程后,父进程关闭读管道端,子进程关闭写管道端,这样提供了两个进程之间数据流动的一种方式。

有名管道也是一种半双工的通信方式,但是它允许无亲缘关系进程间的通信。

无名管道：优点：简单方便；缺点：1）局限于单向通信2）只能创建在它的进程以及其有亲缘关系的进程之间;3）缓冲区有限；

有名管道：优点：可以实现任意关系的进程间的通信；缺点：1）长期存于系统中，使用不当容易出错；2）缓冲区有限

2)信号量

信号量是一个计数器,可以用来控制多个线程对共享资源的访问.,它不是用于交换大批数据,而用于多线程之间的同步.它常作为一种锁机制,防止某进程在访问

资源时其它进程也访问该资源.因此,主要作为进程间以及同一个进程内不同线程之间的同步手段.

优点：可以同步进程；缺点：信号量有限

3)信号

信号是Linux系统中用于进程间互相通信或者操作的一种机制，信号可以在任何时候发给某一进程，而无需知道该进程的状态。

如果该进程当前并未处于执行状态，则该信号就有内核保存起来，知道该进程回复执行并传递给它为止。

如果一个信号被进程设置为阻塞，则该信号的传递被延迟，直到其阻塞被取消是才被传递给进程。

(1) SIGHUP：用户从终端注销，所有已启动进程都将收到该进程。系统缺省状态下对该信号的处理是终止进程。

(2) SIGINT：程序终止信号。程序运行过程中，按 `Ctrl+C` 键将产生该信号。

(3) SIGQUIT：程序退出信号。程序运行过程中，按 `Ctrl+\\` 键将产生该信号。

(4) SIGBUS和SIGSEGV：进程访问非法地址。

(5) SIGFPE：运算中出现致命错误，如除零操作、数据溢出等。

(6) SIGKILL：用户终止进程执行信号。shell下执行 `kill -9` 发送该信号。

(7) SIGTERM：结束进程信号。shell下执行 `kill 进程pid` 发送该信号。

(8) SIGALRM：定时器信号。

(9) SIGCLD：子进程退出信号。如果其父进程没有忽略该信号也没有处理该信号，则子进程退出后将形成僵尸进程。

4)消息队列

消息队列是消息的链表,存放在内核中并由消息队列标识符标识.消息队列克服了信号传递信息少,管道只能承载无格式字节流以及缓冲区大小受限等特点.消息队列是UNIX下不同进程之间可实现共享资源的一种机制,UNIX允许不同进程将格式化的数据流以消息队列形式发送给任意进程.对消息队列具有操作权限的进程都可以使用msget完成对消息队列的操作控制.通过使用消息类型,进程可以按任何顺序读信息,或为消息安排优先级顺序.

优点：可以实现任意进程间的通信，并通过系统调用函数来实现消息发送和接收之间的同步，无需考虑同步问题，方便；缺点：信息的复制需要额外消耗CPU的时间，不适宜于信息量大或操作频繁的场合

5)共享内存

共享内存就是映射一段能被其他进程所访问的内存,这段共享内存由一个进程创建,但多个进程都可以访问.共享内存是最快的IPC(进程间通信)方式,它是针对其它进程间通信方式运行效率低而专门设计的.它往往与其他通信机制,如信号量,配合使用,来实现进程间的同步与通信.

优点：无须复制，快捷，信息量大；

缺点：1）通信是通过将共无法实现享空间缓冲区直接附加到进程的虚拟地址空间中来实现的，因此进程间的读写操作的同步问题；2)利用内存缓冲区直接交换信息，内存的实体存在于计算机中，只能同一个计算机系统中的诸多进程共享，不方便网络通信

6)套接字：可用于不同及其间的进程通信

优点：1）传输数据为字节级，传输数据可自定义，数据量小效率高；2）传输数据时间短，性能高；3）适合于客户端和服务器端之间信息实时交互；4）可以加密,数据安全性强

缺点：1）需对传输的数据进行解析，转化成应用级的数据

2.3 多线程和单线程

- 多线程是指在**一个进程中，并发执行了多个线程**，每个线程都实现了不同的功能
- 在单核CPU中，将CPU分为很小的时间片，在每一时刻只能有一个线程在执行，是一种微观上轮流占用CPU的机制。由于**CPU轮询**的速度非常快，所以看起来像是“同时”在执行一样
- 多线程会存在**线程上下文切换**，会导致程序执行速度变慢
- 多线程不会提高程序的执行速度，反而会降低速度。但是对于用户来说，可以**减少用户的等待响应时间，提高了资源的利用效率**

2.4 线程的状态

1. 初始(NEW)：新创建了一个线程对象，但还没有调用start()方法。
2. 运行(RUNNABLE)：Java线程中将就绪（ready）和运行中（running）两种状态笼统的称为“运行”。

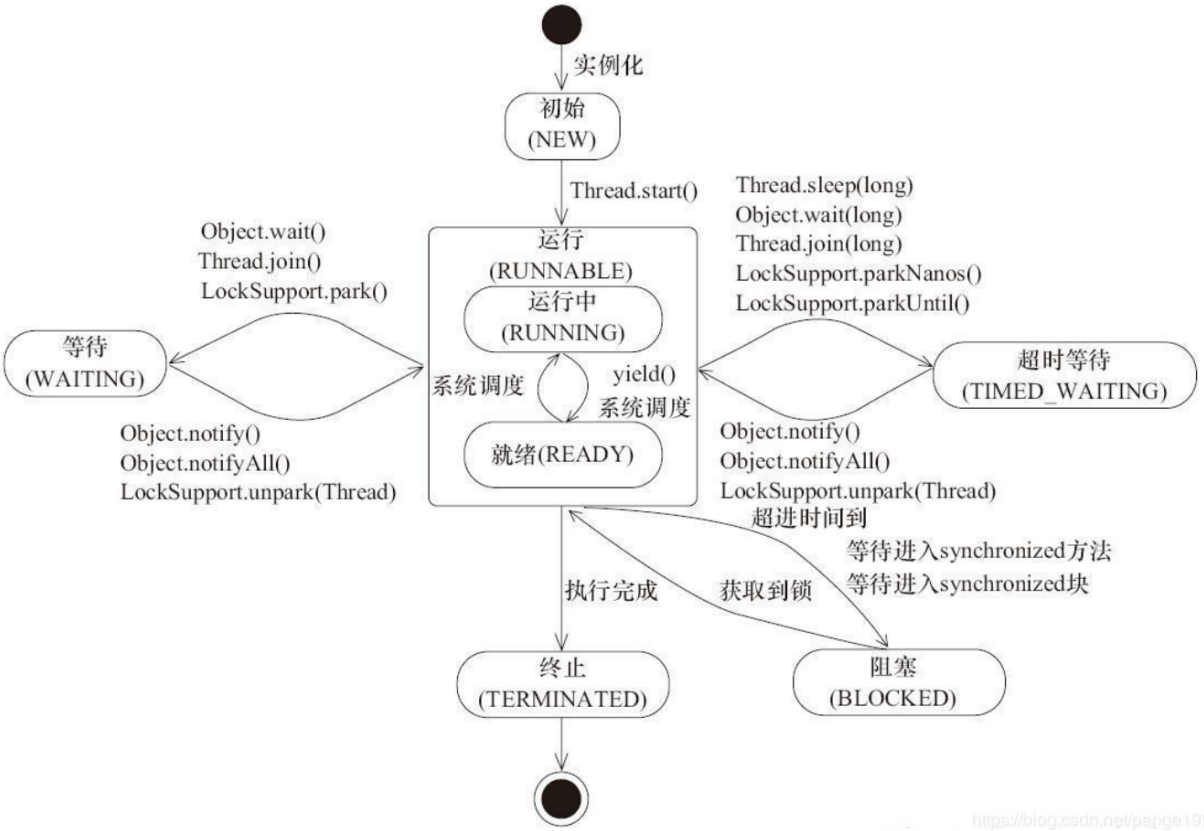
线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取CPU的使用权，此时处于就绪状态（ready）。就绪状态的线程在获得CPU时间片后变为运行中状态（running）。

3. 阻塞(BLOCKED)：表示线程阻塞于锁。

4. 等待(WAITING)：进入该状态的线程需要等待其他线程做出一些特定动作（通知或中断）。

5. 超时等待(TIMED_WAITING)：该状态不同于WAITING，它可以在指定的时间后自行返回。

6. 终止(TERMINATED)：表示该线程已经执行完毕。



2.5 线程安全

线程安全的三大特性：原子性，可见性与有序性

- **原子性是一组操作要么完全发生，要么没有发生，其余线程不会看到中间过程的存在。注意，原子操作+原子操作不一定还是原子操作。**
- **可见性是指一个线程对共享变量的更新对于另外一个线程是否可见的问题。**
- **有序性是指一个线程对共享变量的更新在其余线程看起来是按照什么顺序执行的问题。**
- 可以这么认为，**原子性 + 可见性 -> 有序性**

2.6 进程状态

https://blog.csdn.net/qicheng777/article/details/77427157

进程的三种基本状态：

1.运行态： 占有cpu,并且就在cpu上执行。

2.就绪态：已经具备运行条件，但由于没有空闲cpu,而暂时不能运行。（也就是cpu没有调度到它）

3.阻塞态：因等待某一事件而不能运行。

其实这个很好理解，如果看过我之前关于线程的讨论的话应该很容易理解。注意：单核处理机同一时间只有一个进程处于运行态，双核则是两个。

还有两个状态：

4.创建态：进程正在被创建，系统为其初始化PCB，分配资源。

5.终止态：进程正在从系统中撤销，回收进程的资源，撤销其PCB。

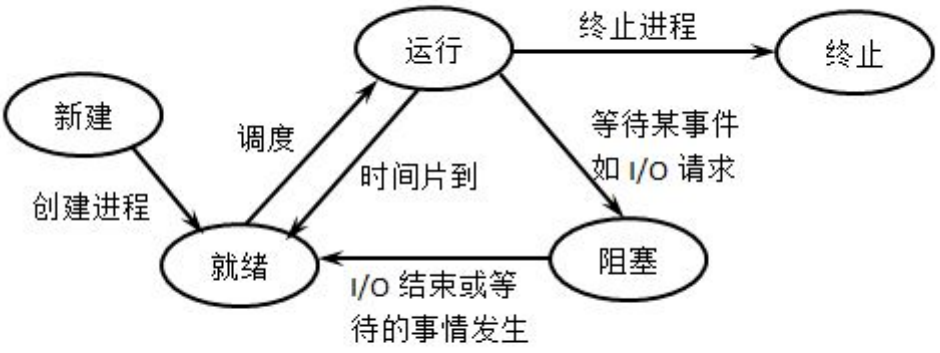


图 2.进程的五态模型