Hexo git 部署

服务器部署

连接服务器：

Ssh -p 端口 root@ip 密码：91JV2daoAmw0

安装

$ yum install git

$ yum install nginx

添加git用户

$ adduser git

$ chmod 740 /etc/sudoers

$ vim /etc/sudoers

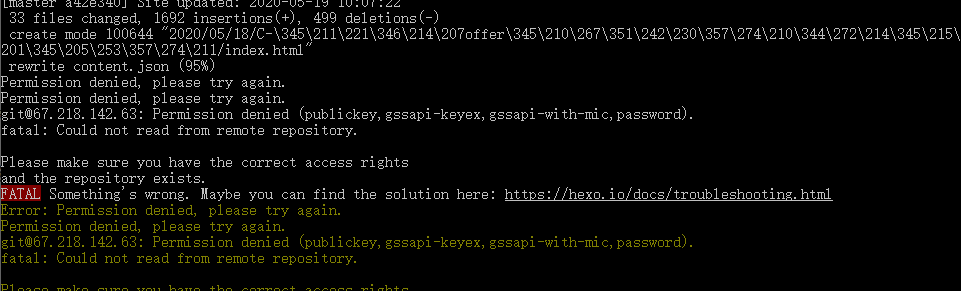
添加git ALL=(ALL) ALL



$ chmod 400 /etc/sudoers

$ sudo passwd git # 设置密码

### **给git用户添加ssh密钥**



$ su git

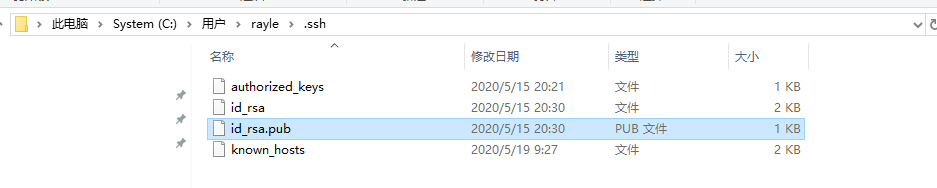
$ mkdir ~/.ssh *#创建.ssh文件夹*

$ touch ~/.ssh/authorized\_keys *#创建authorized\_keys文件*

$ chmod 600 ~/.ssh/authorzied\_keys

$ chmod 700 ~/.ssh

打开win10下面的.ssh/id\_rsa\_pub.........将其复制到服务器上的.ssh/authorized\_keys



$ vim ~/.ssh/authorized\_keys *#没有 此文件创建此文件*

在cmd登录服务器

C:\Users\rayle>ssh git@67.218.142.63

ssh: connect to host 67.218.142.63 port 22: Connection refused

错误解决：

查看服务器是否安装ssh

端口是否是22

$ yum install openssh-server # 安装ssh

$systemct start sshd.service 开启ssh

$systemctl restart sshd.service 重启

$ systemctl enable sshd.service 自动开启

$netstat -ntpl | grep 22 22是否开启

$vi /etc/ssh/sshd\_config #修改端口22

### **创建git仓库并使用git-hooks实现自动部署**

$ sudo mkdir /var/repo *#新建目录，这是git仓库的位置*

$ sudo mkdir /var/www #

$ sudo mkdir /var/www/hexo *#这是博客源文件的目录*

$ cd /var/repo *#转到git仓库的文件夹*

$ sudo git init --bare blog.git *#创建一个名叫blog的仓库*

$ sudo vim /var/repo/blog.git/hooks/post-update

添加这两行

**#!/bin/bash**

git --work-tree**=**/var/www/hexo --git-dir**=**/var/repo/blog.git checkout -f

$ cd blog.git/hooks/

$ sudo chown -R git:git /var/repo/

$ sudo chown -R git:git /var/www/hexo

$ sudo chmod +x post-update

启动nginx

$sudo nginx -s reload

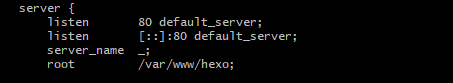
在浏览器输入67.218.142.63，可以访问到nginx

### **配置nginx**

### $ sudo vim /etc/nginx/nginx.conf

修改

root /var/www/hexo



Win10 本地配置

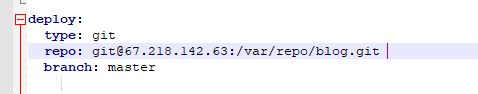
deploy:

type: git

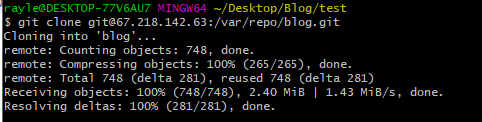
repo: git@67.218.142.63:/var/repo/blog.git

branch: master

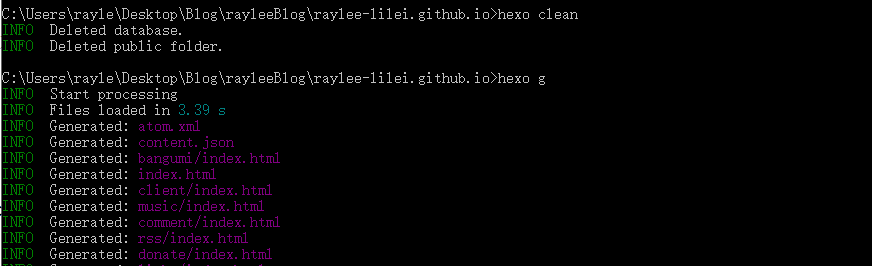
\_Config.yml文件

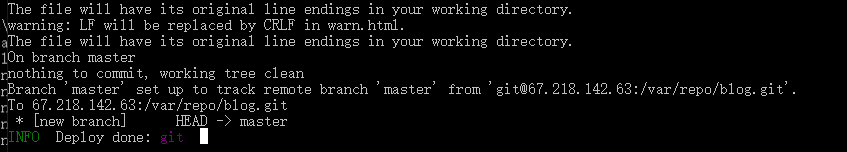


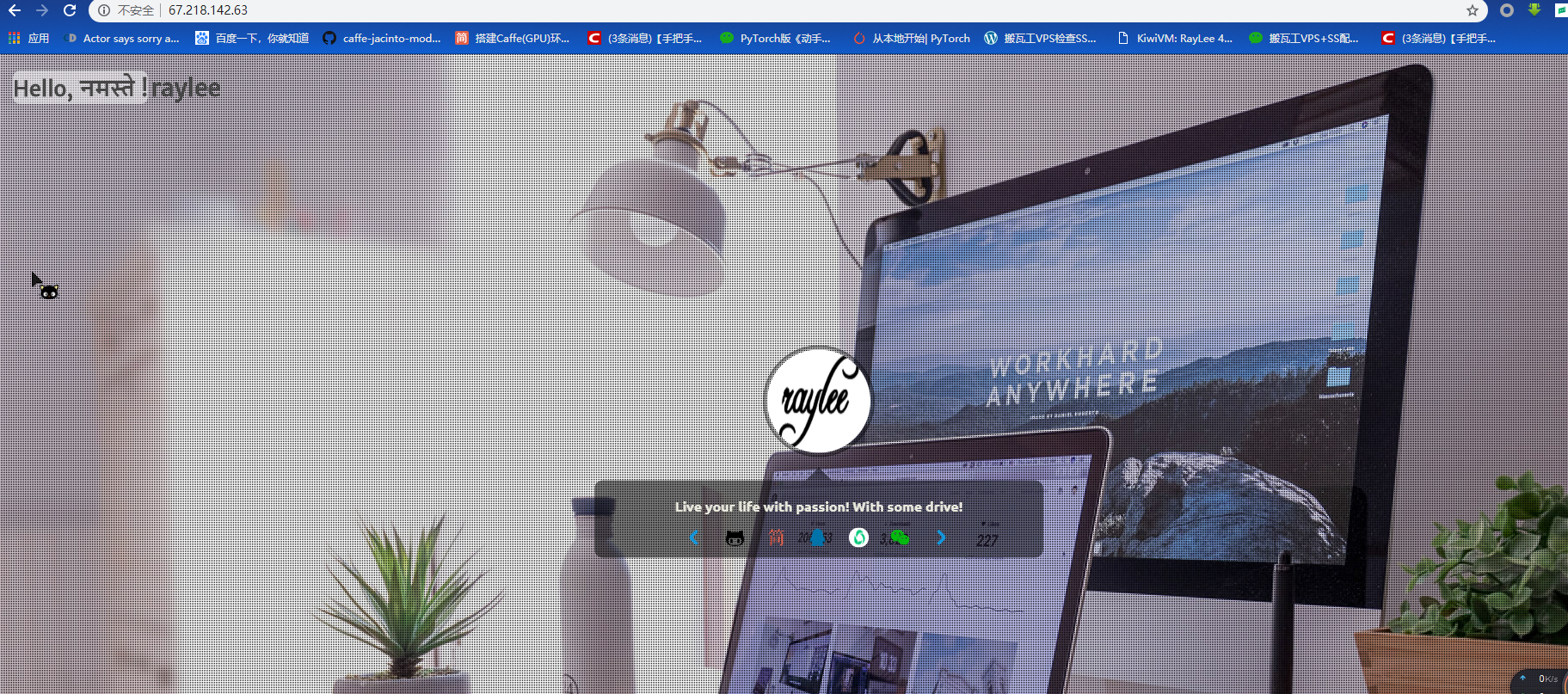
git clone git@67.218.142.63:/var/repo/blog.git



本地上传







成功！

面试总结

什么是进程：

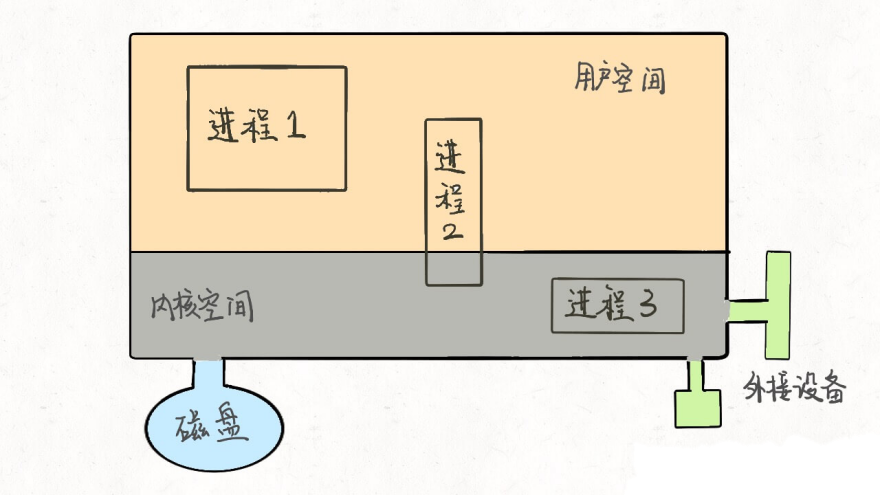
进程是对**运行时程序**的封装，是系统进行资源调度和分配的的基本单位，实现了操作系统的并发

进程一般是在用户空间，动态链接库可能会访问内核空间的资源

进程其实是由操作系统创建的，当我们用户写入程序由操作系统创建进程，程序创建的可执行文件载入内存中，包装成一个进程。进程有他自己的PID，进程状态，打开的文件。

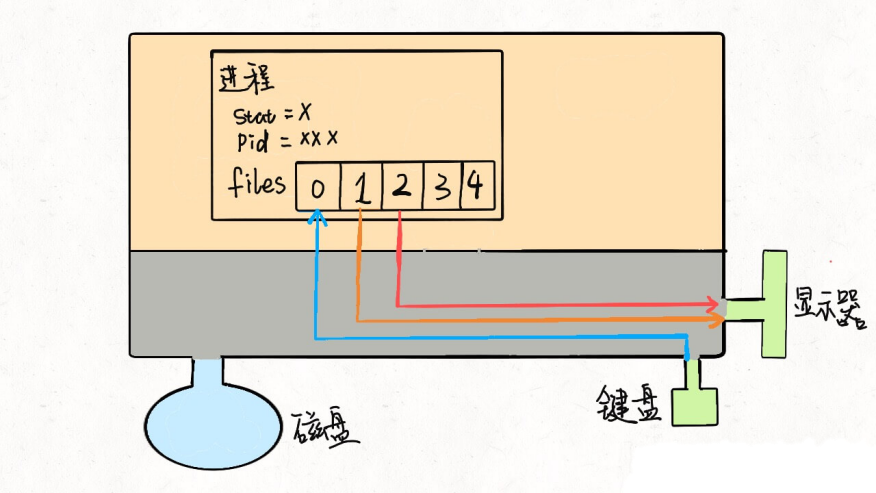
进程实际上对于操作系统 而言是一种数据结构task\_struct（这个数据结构也就是对进程的描述）

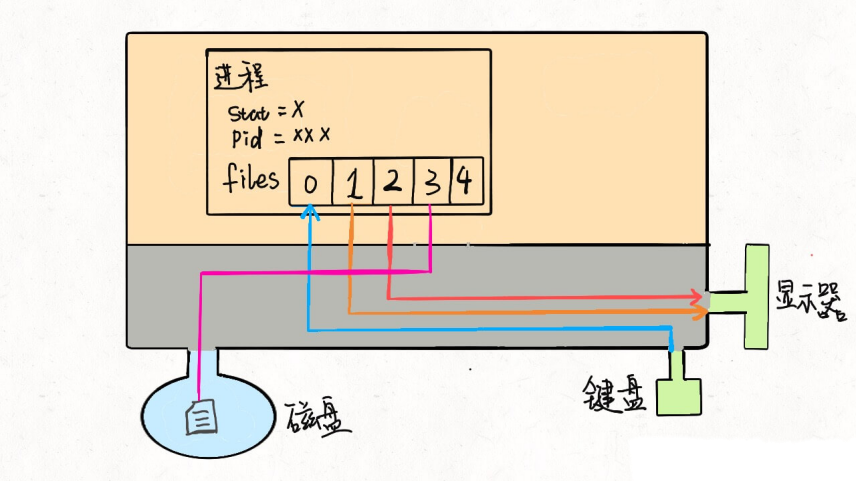
里面有个files 指针指向一个数组，里面装着该进程要打开的文件的指针



文件描述符：

一个进程会从files[0]读取输入，将输出写入files[1]，将错误信息写入files[2]

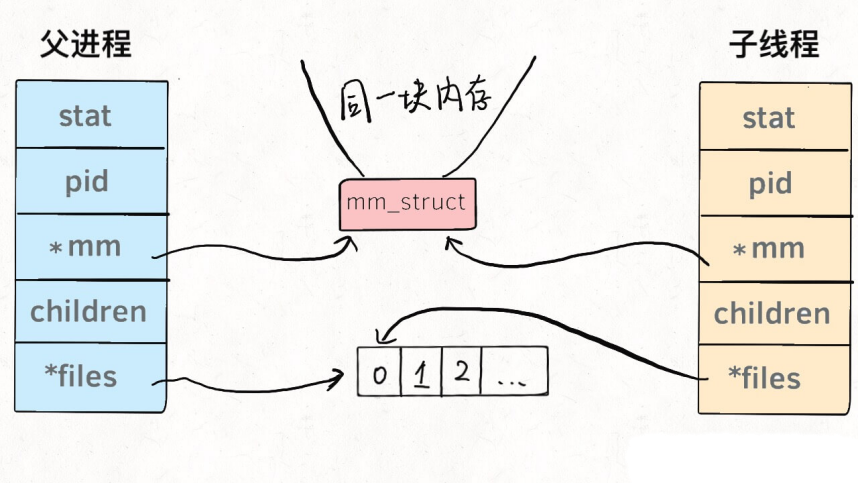




输入重定向，就会去files[0]读取，只要把files[0]指向一个文件，那么程序就会从这个文件中读取数据，而不是从键盘。

输出重定向就是把files[1]指向一个文件，那么程序的输出就不会写入到显示器，而是写入到这个文件中。

Linux 中线程和进程基本没有区别，系统调用fork()可以新建一个子进程，函数pthread()可以新建一个线程。但无论线程还是进程，都是用task\_struct结构表示的，唯一的区别就是**共享的数据区域不同**。



进程是对运行时程序的封装，是系统进行**资源**调度和分配的的基本单位，实现了操作系统的并发

线程是进程的子任务，是**CPU**调度和分派的基本单位，实现**进程内部**的并发，每个线程完成不同的任务，但是**共享同一地址空间**

区别：

1. 一个线程只属于一个进程，一个进程有多个线程
2. **进程**在执行过程中拥有**独立的内存单元**，而多个线程共享进程的内存，共享代码段，数据段，扩展段，每个线程拥有自己的栈段
3. 进程是资源分配的最小单位，线程是CPU调度的最小单位
4. 进程切换的开销也远大于线程切换的开销
5. 进程编程调试简单可靠性高，但是创建销毁开销大；线程正相反，开销小，切换速度快，但是编程调试相对复杂
6. 进程间不会相互影响 ；线程一个线程挂掉将导致整个进程挂掉
7. 由于同一进程中的多个线程具有相同的地址空间，致使它们之间的同步和通信的实现，也变得比较容易。进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信

线程多CPU

单核中需要线程锁。在抢占式操作系统中，通常为每个线程分配一个**时间片**，当某个线程时间片耗尽时，操作系统会将其挂起，然后运行另一个线程。如果这两个线程共享某些数据，不使用线程锁的前提下，可能会导致共享数据修改引起冲突。

线程切换

过程中需要保存当前线程Id、线程状态、堆栈、寄存器状态等信息。其中寄存器主要包括SP PC EAX等寄存器，其主要功能如下：

SP:堆栈指针，指向当前栈的栈顶地址

PC:程序计数器，存储下一条将要执行的指令

EAX:累加寄存器，用于加法乘法的缺省寄存器

进程通行方式

1. 管道 （普通管道：父子/兄弟之间 命名管道：）
2. 系统IPC

消息队列：具有**写权限**的进程可以按照一定得规则向消息队列中**添加**新信息；对消息队列有**读权限**得进程则可以从消息队列中**读取信息**

信号量：它是一个计数器，可以用来控制多个**进程**对共享资源的访问，实现进程间的互斥与同步

信号量用于进程间同步，若要在进程间传递数据需要结合共享内存

（1）P(sv)：如果sv的值大于零，就给它减1；如果它的值为零，就挂起该进程的执行

（2）V(sv)：如果有其他进程因等待sv而被挂起，就让它恢复运行，如果没有进程因等待sv而挂起，就给它加1.

信号：用于通知接收进程某个事件已经发生

共享内存：（互斥锁和信号量）同步

1. 套接字SOCKET：不同主机之间的进程通信

线程间通信的方式

临界区：多线程的串行化来访问公共资源

互斥量：只有拥有互斥对象的线程才有访问公共资源的权限

信号量：多个线程在同一时刻去访问同一个资源

事件（信号）：通过通知操作的方式来保持多线程同步

**C++ 从源文件到可执行文件发生了什么**

预处理阶段：头文件、宏定义进行分析替换——>预编译文件

编译阶段：生成汇编文件

汇编阶段：将汇编文件转成机器码——>可重定位目标文件

链接阶段：多个目标文件以及所需要的库连接成最终的可执行文件

**include头文件的顺序以及双引号””和尖括号<>的区别**

定义一个C

申明一个变量C

A.h B.h  
  
  
  
  
  
  
  
 A.cpp B.cpp  
预处理阶段 路径不一样

1. h(在前)

A.h(在后)

“”：当前头文件目录——编译器设定的头文件路径——系统变量指定的头文件路径

<>:编译器设定的头文件路径——系统变量指定的头文件路径

**malloc的原理，另外brk系统调用和mmap系统调用的作用分别是什么**

动态分配内存

减少内存碎片和系统调用的开销 采用 内存池

申请大的内存空间堆区  
  
  
  
  
 边界标记法 合并  
  
  
  
  
  
  
当申请内存小于128K时，会使用系统函数brk在**堆区**中分配；而当申请内存大于128K时，会使用系统函数mmap在**映射区**分配

块

数据库知识

Mysql建表约束：

主键约束：唯一确定一张表的记录，此字段不重复且不为空（插入）

Create table user(id int primary key,

Name varchar(20)

);

Id 不能为空，也不能重复

联合主键:只要主键值加起来不重复就可以，但是任何一个字段都不能为空

Create table user(id int ,

Name varchar(20),

Password varchar(20),

Priamary key(id,name)

);

Insert into user values(1,’lilei’,’1234’);

Insert into user values(2,’lilei’,’1234’);

也可以添加成功

自增约束

Create table user(id int primary key auto\_increment，

Name varchar(20)

);

Insert into user（name） values(’lilei’);

添加主键约束

Alter table user add primary key(id);

删除主键约束

Alter table user drop primary key;

修改字段添加主键约束

Alter table user modify id int primary key;

唯一约束

修饰改字段的值不可以重复

Alter table user add unique(name);

Create table user(id,

Name,

Unique(name));

非空约束

Create table user（id，

Name varchar(20) not null）;

默认约束

插入字段时，若果没有传值，就会使用默认值

Create table user（id，

Name varchar(20)，

Age int default 10）;

外键约束

两个表 主表 副表

Create table classes（id int primary key，

Name varchar(10)）;

Create table student（id int primary key，

Name varchar(10),

Class\_id int,

Foreign key(class\_id) references classes(id)）;

主表中没有的值，附表是不能使用的

主表中的记录被附表引用，主表是不可以删除的

事务：转账

在执行多条sql 语句的时候保证前后执行效果一致，要么同时成功，要么同时失败

提供了一种撤销的机制

Mysql 默认自动提交，不能进行回滚

Set autocommit = 0之后就可以进行回滚（撤回操作）然后再提交

开启事务的方式

Set autocommit = 0

Begin

Start transaction

事务的特性

A 原子性 事务是最小的单元，不可再分割

C 一致性 同一个事务的多条sql要么同时失败要么同时成功

I 隔离性 两个事务具有隔离性

D 持久性 事务一旦结束，就不可以再更改

隔离性

1. Read-uncommitted 读未提交

A 更新了数据，但是没有commit，B也查看到了更新的数据结果，之后A执行rollback，数据回退了。

查看隔离级别

Mysql 默认是repeatable\_read

Select @@global.Transaction\_isolation

Select @@tx\_isolation

修改隔离级别

Set global transaction isolation level read uncommitted

出现脏读现象 ：一个事物读取另一个事物未提交数据

1. read-committed 读已经提交的 （未提交读取不到）

A查询了一次，B开启了一个事务，添加并commit数据，A过一段时间求平均值发现前后不一样

**出现不可重复读现象**

1. repeatable\_read

事务A和事物B同时操作一张表，事物A插入的数据不能被事物B所看到（A在操作时候还没有执行提交），出现了幻读现象

1. Serializable（串行化）

事物 A和事物B同时操作一张表的时候，A操作的时候还没有提交，B想 执行插入操作会卡住不动，指导A执行完commit 之后，B的操作会立即执行

带来了串行化问题 执行性能差

隔离级别越高，性能越差

read-committed > read-committed > repeatable-read >serializable

**Linux虚拟地址空间**

防止**不同进程同一时刻**在物理内存中运行而对**物理内存的争夺**和践踏。

所有进程共享同一物理内存，每个进程只把自己目前需要的虚拟内存空间映射并存储到物理内存上。

实际上并不立即就把虚拟内存对应位置的程序数据和代码拷贝到物理内存中，只是建立好虚拟内存和磁盘文件之间的映射就好，等到运行到对应的程序时，来拷才会通过缺页异贝数据。动态分配内存，比如malloc时，也只是分配了虚拟内存，当进程真正访问到此数据时，才引发缺页异常。

好处：

1. 扩大地址空间
2. 内存保护：每个进程运行在各自的虚拟内存地址空间，互相不能干扰对方
3. 公平内存分配。每个进程都相当于有同样大小的虚存空间。
4. 当进程通信时，可采用**虚存共享**的方式实现
5. 当不同的进程使用同样的代码时，物理内存中可以只存储一份这样的代码，不同的进程只需要把自己的虚拟内存映射过去就可以了，节省内存
6. 在内存中可以保留多个进程，系统并发度提高
7. 在程序需要分配连续的内存空间的时候，只需要在虚拟内存空间分配连续空间，而不需要实际物理内存的连续空间，可以利用碎片

代价：

虚存的管理需要建立很多数据结构，这些数据结构要占用额外的**内存**

虚拟地址到物理地址的转换，增加了指令的执行**时间**

页面的换入换出需要磁盘I/O，这是很耗时的

如果一页中只有一部分数据，会浪费**内存**

**缺页中断**

在地址映射过程中，所访问的目的页不在内存中，则系统产生异常中断

通过查询页表中的状态位来确定所要访问的页面是否存在于内存中，每当所要访问的页面不在内存时，会产生一次缺页中断，此时操作系统会根据页表中的外存地址在外存中找到所缺的一页，将其调入内存

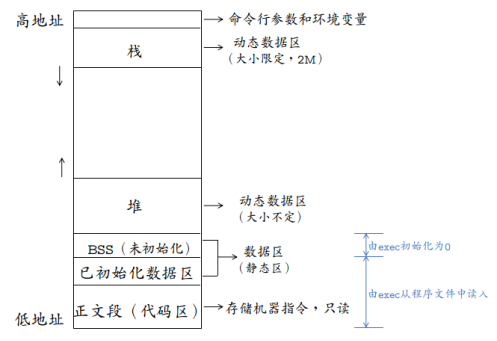
**程序的内存结构**

**BSS段：**通常用来存放程序中未初始化的全局变量和静态变量的一块内存区域

**data段：**存放程序中已初始化的全局变量的一块内存区域

**text段：**存放程序执行代码的一块内存区域

text段和data段在编译时已经分配了空间，而BSS段并不占用可执行文件的大小，它是由链接器来获取内存的

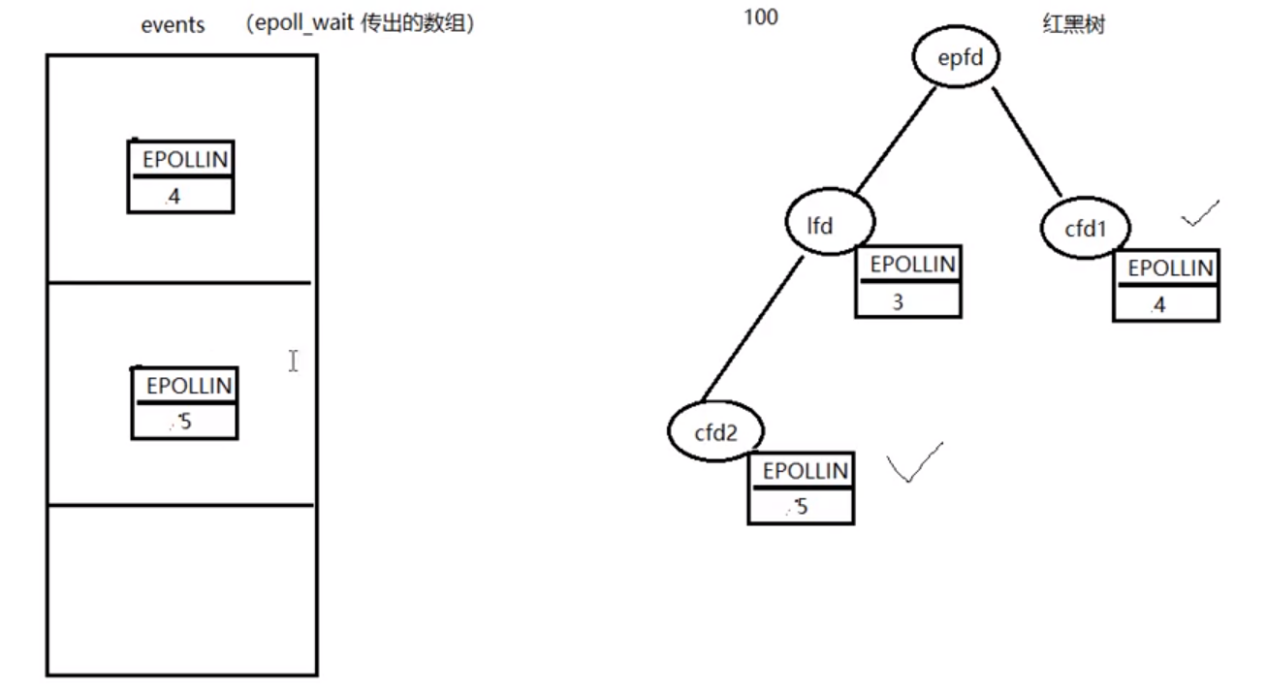


可执行程序在运行时又多出两个区域：栈区和堆区。

栈区 存放函数的参数值、局部变量等 高——>低

堆区 用于动态分配内存，位于BSS和栈中间的地址区域。由程序员申请分配和释放。采用链式存储结构。频繁的malloc/free造成内存空间的不连续，产生碎片 低——>高

Epoll



Select

