# OCV Learning Note

# **Mat - 基本图像容器**

## Mat

* OpenCV函数中输出图像的内存分配是自动完成的（如果不特别指定的话）。
* 使用OpenCV的C++接口时不需要考虑内存释放问题。
* 赋值运算符和拷贝构造函数（ ctor ）只拷贝信息头。
* 使用函数 [clone()](http://opencv.itseez.com/modules/core/doc/basic_structures.html" \l "mat-clone) 或者 [copyTo()](http://opencv.itseez.com/modules/core/doc/basic_structures.html" \l "mat-copyto) 来拷贝一副图像的矩阵。

### 多线程和多进程的区别

进程： 程序的一次执行，内存空间独立，同一进程的两段代码不能同时执行，除非引入线程

线程： 程序片段，占用资源少，栈独立。同一进程产生的线程共享内存空间。

### 多线程通信

同步

互斥

信号量

数据传输：一个进程的数据发送给另一进程

资源共享：多个进程共享同样的资源

通知事件：一个进程需要向另一个或一组进程发送消息

进程控制：Debug进程，SIGtrap

管道

SystemV进程间通信

POSIX进程间通信

进程间通信的分类（

文件，

文件锁

管道与有名管道

信号

**消息队列**

**共享内存**

**信号量**

**互斥量**

**条件变量**

**读写锁**

套接字）

文件系统(posix 映射文件 文件持续) 内核（systemV一直存在，显示删除，内核自举或重启） 共享的内存区（进程持续）

### 多线程通信2

* + 1. 死锁

多个进程之间相互等待对方的资源，在得到对方的资源之前不释放自己的资源

互斥条件，

请求和保持条件

不可剥夺

环路等待

* + 1. 防止死锁办法

资源一次性分配， 降低并发性

可剥夺资源

资源有序分配

银行家算法

* + 1. 信号量PV原语

互斥：PV操作在同一进程中

同步：PV在不同进程中

S>0 可用的资源个数

S=0 无可用资源，无等待进程

S<0 等待进程的个数

struct semaphore

{

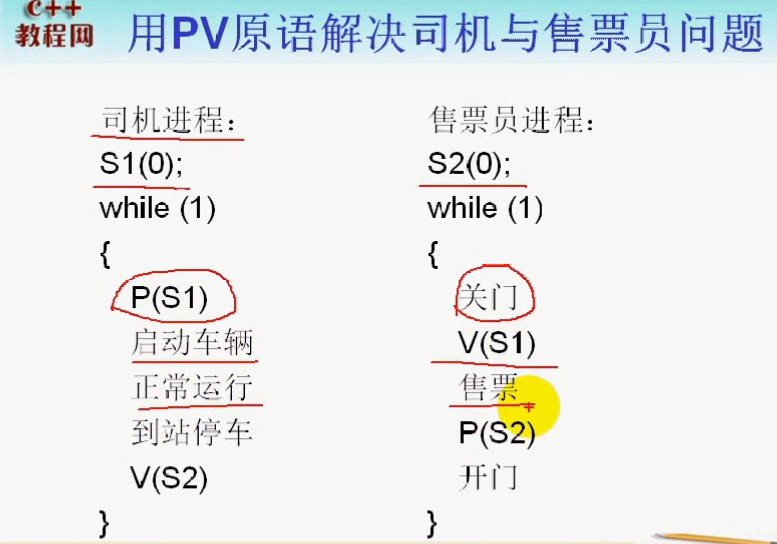
int value;

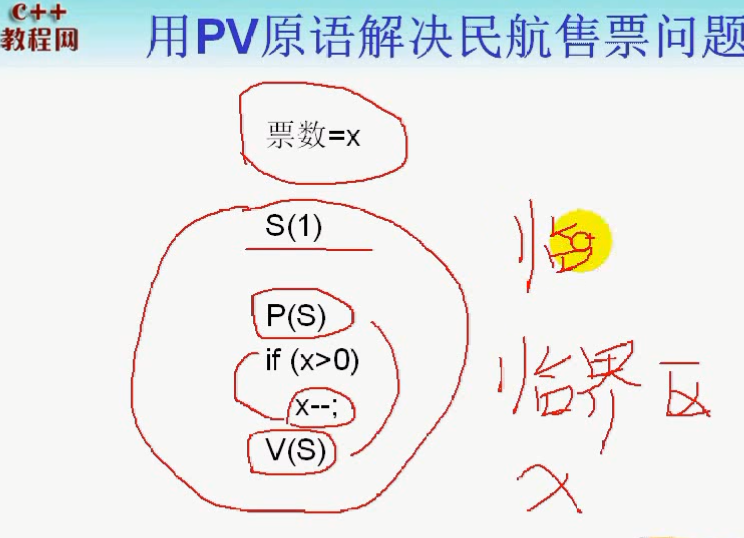
pinter PCB\_dueue

}

P(s){s.value-- if s.value<0 s.queue.insert else doing} //原子性

V(s){s.value++ if s.value<=0 s.queue.pop() else doing} //原子性





* + 1. 消息队列

从一个进程发送数据块到另一个进程

字节流， 消息

cat /proc/sys/kernel/msgmax

8912

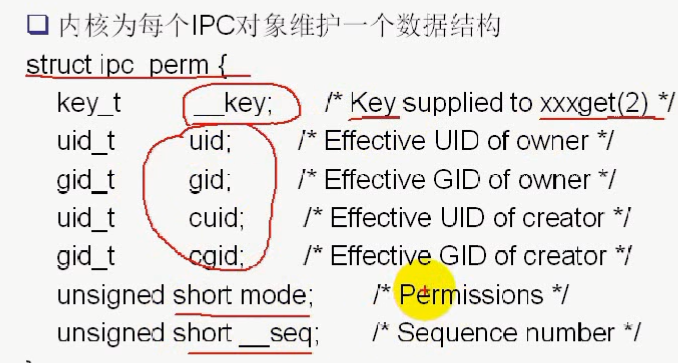
cat /proc/sys/kernel/msgmnb

16384

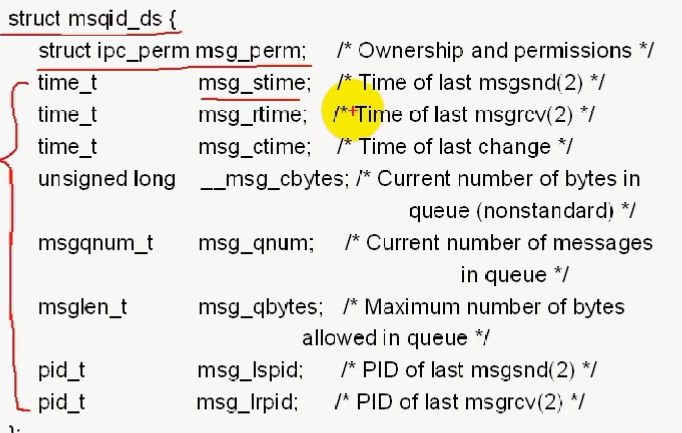
cat /proc/sys/kernel/msgmni

32000

内核为每个IPC对象维护一个数据结构



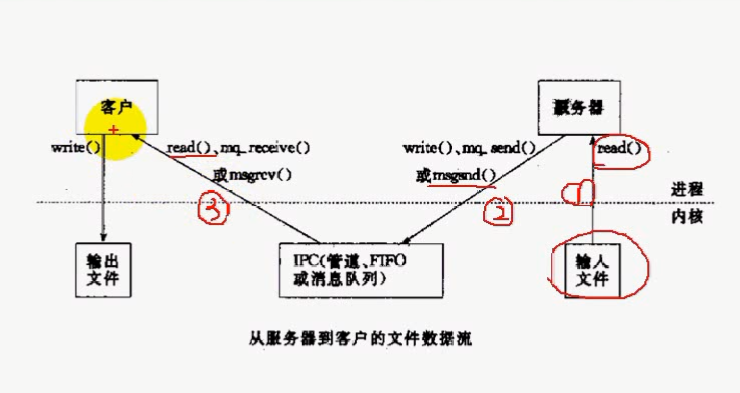
消息队列的数据结构



* + 1. 共享内存

最快的IPC形式，一旦这样的内存映射到共享他的进程的地址空间，这些进程间的数据传递不再涉及到内核，

共享内存可以映射到不同的进程的地址空间



映射不能改变文件的大学

内存区域以页面为单位，可以大于实际文件大小的空间

* + 1. dsga
    2. sdga

## static

* 静态成员函数属于类而不是类对象，所以没有this指针，就是读写静态成员的，不能访问普通成员
* 内存中只有一份数据，所有的对象共享，必须初始化，不能再类的定义中初始化
* 想当于nomember 函数，有利于成为callback，方便作为线程函数

## 虚函数virtual

### virtual 介绍和使用规则

* 虚函数的作用主要是实现了多态的机制，也就是通过基类指针调用实际派生类的成员函数
* 非类的成员函数不能定义为虚函数，
* 类的成员函数中静态成员函数和构造函数也不能定义为虚函数，但可以将析构函数定义为虚函数

### 虚析构

Delete一个指向派生类对象的基类指针，子类和基类的虚构函数都会调用。非虚析构的话，只调用基类的析构函数

### 虚函数表

 为了保证正确取到虚函数的偏移量，编译器必需要保证虚函数表的指针存在于对象实例中最前面的位置

typedef void(\*Fun)(void);

Base b;

Fun pFun = NULL;

cout << "虚函数表地址：" << (int\*)(&b) <<endl; //虚表的地址

cout << "虚函数表 — 第一个函数地址：" << (int\*)\*(int\*)(&b) <<endl;

pFun = (Fun)\*((int\*)\*(int\*)(&b)); // 将地址转化成函数地址

pFun();

(Fun)\*((int\*)\*(int\*)(&b)+0); // Base::f()

(Fun)\*((int\*)\*(int\*)(&b)+1); // Base::g()

(Fun)\*((int\*)\*(int\*)(&b)+2); // Base::h()

## 

## 异常处理exception

## 虚函数virtual

## 虚函数virtual

## 虚函数virtual