# 一、多线程

## <1>关于线程的基本概念

### －线程的定义:

轻量级进程，程序执行的最小单位，系统独立调度和分派cpu的基本单位，它是进程中一个实体。一个进程中有多个线程，这些线程共享进程的所有资源，线程本身只包含一点必不可少的资源。

### －进程的定义:

一个正在执行的程序，它是资源分配的最小单位

### －线程和进程的区别

－多个线程共享进程的资源，而多进程编程中创建子进程需要copy整个父进程的资源，

－多线程通信比较容易

－多进程通信需要创建管道等，比较复杂。

### －线程的术语

－并发:

同一时刻，只能有一条指令执行，但多个进程指令被快速轮换执行，使得在宏观上具有多个进程同时执行的效果。

特点：看起来同时发生，单核处理器。

－并行

同一时刻，有多条指令在多个处理器上同时执行。

特点：真正的同时发生，多核处理器。

－同步

彼此有依赖关系的调用不应该“同时发生”，而同步就是要阻止那些同时发生的事情。

－异步

任何俩个彼此独立的操作是异步的，它表明事情独立发生。

### －线程的优势

１、在多处理器中开发程序的并行性

２、在等待慢速的IO操作时，程序可以执行其他操作，提高并发性

３、模块化的编程，能更清晰的表达程序中独立事件的关系，结构清晰

４、占用较少的系统资源（相对与多进程编程）

**多线程不一定多核处理器**

## <2>线程的生命周期

### －线程ID

### －创建线程

### －初始线程/主线程

1. 当ｃ程序运行时，首先运行main函数，在线程代码中，这个特殊的执行流被称作**初始线程**或者**主线程**。你可以在初始线程中做任何普通线程可以做的事情。
2. 主线程的**特殊性**在于，它在main函数返回的时候，会**导致进程的结束**，进程内所有的线程也将会结束。这可不是一个好的现象，你可以在主线程中调用**pthread\_exit**函数，这样进程就会等待所有线程结束时才终止。
3. 主线程**接受参数的方式是通过argc和argv**，而普通线程只有一个参数void\*。
4. 在绝大多数情况下，主线程在默认堆栈上运行，这个堆栈可以增长到足够的长度。而普通线程的堆栈是受限制的，一旦溢出就会产生错误。

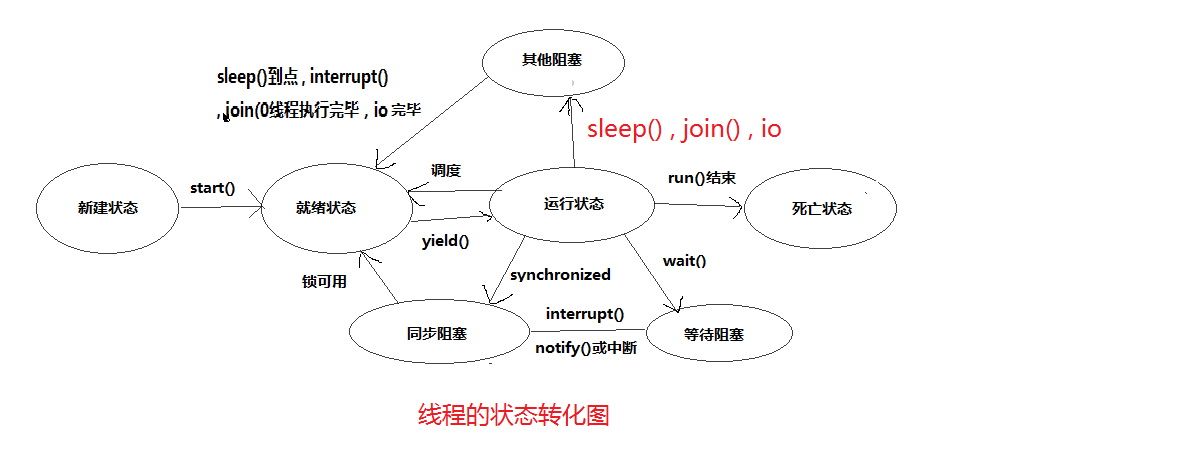
### －线程的四个基本状态

－就绪：线程能够运行，但在等待可用的处理器

－运行：线程正在运行，在多核系统中，可能同时有多个线程在运行

－阻塞：线程在等待处理器以外的其他条件

－终止：线程从启动函数中返回，或者调用pthread\_exit函数，或者被取消



就绪:当线程刚**被创建**就处于就绪状态，或者当线程**被解除阻塞**以后也会处于就绪状态。

就绪的线程在等待一个可用的处理器，当一个运行的线程**被抢占**时，它立刻又回到就绪状态。

运行：当处理器选中一个就绪态的线程执行时，它立刻变成运行状态。

阻塞：试图加锁一个已经被锁住的互斥量，等待某个条件变量，调用singwait等待尚未发生的信号，执行无法完成的I/O信号，由于内存页错误。

### －回收

－线程的分离属性：

分离一个正在运行的线程并不影响它，仅仅是通知当前系统该线程结束时，其所属的资源可以回收，一个没有被分离的线程在终止时会保留它的虚拟内存，包括他们的堆栈和其它系统资源，有时这种线程被称为“僵尸线程”。

**创建线程时默认是非分离的**

如果线程具有分离属性，线程终止时会被立刻回收，回收将释放掉所有在线程终止时未释放的系统和进程资源，包括保存线程返回值的内存空间、堆栈、保存寄存器的内存空间等。

终止被分离的线程会释放所有的系统资源，但是你必须释放有该线程占用的程序资源。由malloc或者mmap分配的内存可以在**任何时候由任何线程释放**，条件变量、互斥量、信号灯可以由任何线程销毁，只要它们被解锁了或者没有线程等待。但是只有互斥量的主人才能解锁它，所以在线程终止前，你需要解锁互斥量。

## <2>线程的基本控制

### －线程终止

－exit是危险的

如果进程中的任意一个线程调用了exit,\_Exit,\_exit,那么整个进程就会终止。

－不终止进程的退出方式

－普通的单个线程有一下３中方式退出，这样不会终止进程

1. 从启动线程程中返回，返回值是线程的退出码
2. 线程可以被同一进程中的其它线程取消
3. 线程调用pthread\_exit(void \*rval)函数，rval是退出码

Return 和pthread\_exit的区别

### －线程连接

- int pthread\_join(pthread\_t tid,void \*\*rval);

**调用该函数的线程会一直阻塞，直到指定的线程tid调用pthread\_exit、从启动线程返回或者被取消。参数tid就是指定线程的id**

参数rval是指定线程的返回码，如果线程被取消，那么rval被设置为PTHREAD\_CANCELED 该函数调用成功会返回０，失败返回错误码。

调用pthread\_join会使指定的线程处于分离状态，如果指定线程**已经处于分离状态**，那么调用就会失败。

Pthread\_detach可以分离一个线程，线程可以自己分离自己

int pthread\_detach(pthread\_t thread);

### －线程取消

**－取消函数**

Int pthread\_cancel( thread\_t tid ) 取消tid指定的线程，成功返回０。但是取消只是发送一个请求，并不意味着等待线程终止，**而且发送成功也不意味着tid一定会终止**。

**－取消状态：就是线程对取消信号的处理方式，忽略或者响应，线程创建时默认响应取消信号**

Int pthread\_setcancelstate(int state,int \*oldstate)

设置本线程对Cancel信号的反应，state有两种值：PTHREAD\_CANCEL\_ENABLE(缺省)和PTHREAD\_CANCEL\_DISABLE,

分别表示收到信号后设为CANCEL状态和忽略CANCEL信号继续运行；如果不为

NULL则存入原来的Cancel状态以便恢复

**－取消类型：是线程对取消信号的响应方式，立即取消或者延时取消，线程创建时默认时延时取消**

Int pthread\_setcanceltype(int type,int \*oldtype)

设置本线程取消动作的执行时机，type有两种值：PTHREAD\_CANCEL\_DEFFERED和PTHREAD\_CANCEL\_ASYCHRONOUS,仅当Cancel状态为Enable时有效，分别表示收到信号后继续运行至下一个取消点在退出和立即执行取消动作（退出）；oldtype 如果不为NULL则存入运来的取消动作类型值。(asychronous 异步,deferred 推迟 )

**－取消点**

取消一个线程，它通常需要被取消线程的配合。线程在很多时候会查看自己是否有取消请求，如果有就主动退出，这些查看是否有取消的地方称为取消点

很多地方都是包含取消点，包括

Pthread\_join()、pthread\_testcancel()、pthead\_cond\_waid()、sem\_wait() sigwait()、write 、read 大多数会阻塞的系统调用

通过man pthreads查看那些取消点

### －向线程发送信号

### －清除操作