＃include<pthread.h>

int pthread\_create(

      pthread\_t \*restrict tidp,

      const pthread\_attr\_t \*restrict attr,

      void \*(\*start\_rtn)(void \*),

void \*restrict arg);

线程创建

tidp:线程id

attr:线程属性 若为NULL则设置默认值

start\_rtn:线程函数指针

arg:传递给线程函数的参数列表

返回值:0成功

void pthread\_exit(void \*rval\_ptr);

线程返回

rval\_ptr:线程返回值 可由pthread\_join()获得

int pthread\_join(pthread\_t thread, void \*\*rval\_ptr);

阻塞当前线程并且等待指定线程结束 并获取其返回值

一个线程 只能有一个与之对应的pthread\_join() 即一个线程不能被多个线程所等待

thread:指定等待的线程变量

rval\_ptr:接收返回值的指针

返回值:0成功

int pthread\_cancel(pthread\_t thread);

向指定线程发送终止信号 成功返回0

成功后需使用pthread\_join()

thread:指定的线程

int pthread\_attr\_init(pthread\_attr\_t \*attr);  
初始化一个线程对象的属性

attr:属性

返回值:0成功

int pthread\_attr\_destroy(pthread\_attr\_t \*attr);

释放线程对象属性

返回值:0成功

int pthread\_attr\_setscope(

pthread\_attr\_t \*attr,int scope);

设置绑定属性

具有绑定属性的线程可以保证在需要的时候总有一个内核线程与之对应

scope:PTHREAD\_SCOPE\_SYSTEM 绑定

PTHREAD\_SCOPE\_PROCESS 非绑定

返回值:0成功 -1出错

int pthread\_attr\_setdetachstate(

pthread\_attr\_t \*attr,int detach);

设置线程分离属性

分离属性情况下，一个线程结束时立即释放它所占有的系统资源

detach:PTHREAD\_CREATE\_DETACHED 分离

PTHREAD\_CREATE\_JOINABLE 非分离

返回值:0成功

int  pthread\_mutex\_init(

pthread\_mutex\_t \*mp,

const pthread\_mutexattr\_t \*mattr);

初始化互斥锁

mp:互斥锁地址

mattr:互斥锁属性 默认NULL 为快速互斥锁

PTHREAD\_MUTEX\_INITIALIZER

返回值:0成功

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

上锁

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

解锁

int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

判断是否上锁

int pthread\_mutex\_destroy(pthread\_mutex\_t \*mp);

销毁互斥锁

#include <semaphore.h> //POSIX信号量

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

sem:信号量

pshared:为实现 只能为0

value:信号量初值

int sem\_destory(sem\_t \*sem);

销毁

int sem\_wait(sem\_t \*sem);阻塞

int sem\_trywait(sem\_t \*sem);立即返回 成功0 否则-1

int sem\_post（sem\_t \*sem）;

相当于V操作

int sem\_getvalue(sem\_t \*sem);

得到信号量值