# 多线程

1 线程的概念

多线程有许多标准， 如：POSIX1、openmp 等。我们主要讲POSIX1的函数接口及标准。

在POSIX标准中pthread\_t为一个线程的标识符。进程有进程号PID 线程的“线程号”就是pthread\_t。pthread\_t类型在不同平台的定义不同，可能是int，可能是struct等，所以要用专用的函数来对pthread\_t类型进行处理。比如判断两个pthread\_t是否相等用：pthread\_equal() 函数。

2线程的创建

//创建一个线程

pthread\_creat:

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,

void \*(\*start\_routine) (void \*), void \*arg);

pthread\_t \*thread, 新建线程的标识符

const pthread\_attr\_t \*attr, 规定创建线程的属性，空则表示默认属性创建。

void \*(\*start\_routine) (void \*), 线程要执行的函数指针

void \*arg 要执行函数的参数。

note:创建后的线程与main函数线程为同级的关系，执行mian函数的线程称为main线程。main线程和他创建出的线程谁先执行由调度决定。mian线程结束代表整个进程结束，所以main线程需要等待其他线程执行完再结束。

3 线程的退出

线程结束方式：

* 从线程要执行的函数中return
* 线程本身调用函数pthread\_exit
* 同进程的其他线程要求该线程结束

void pthread\_exit(void \*retval);

retval 为要返回的参数

线程结束的清理函数（结束钩子函数）：

pthread\_cleanup\_push

将函数挂入回调函数

pthread\_cleanup\_pop

决定回调函数是否执行

pthread\_cleanup\_push()/pthread\_cleanup\_pop()采用先入后出的栈结构管理（先挂入后执行），void routine(void \*arg)函数在调用pthread\_cleanup\_push()时压入清理函数栈，多次对pthread\_cleanup\_push()的调用将在清理函数栈中形成一个函数链，在执行该函数链时按照压栈的相反顺序弹出。execute参数表示执行到pthread\_cleanup\_pop()时是否在弹出清理函数的同时执行该函数，为0表示不执行，非0为执行；这个参数并不影响异常终止时清理函数的执行。

pthread\_cleanup\_push()/pthread\_cleanup\_pop()是以宏方式实现的。pthread\_cleanup\_push()带有一个"{"，而pthread\_cleanup\_pop()带有一个"}"，因此这两个函数必须成对出现，且必须位于程序的同一级别的代码段中才能通过编译。

pthread\_cleanup\_push();

...................

pthread\_exit(NULL);

...................

pthread\_cleanup\_pop();

这两句话之间任何导致该线程推出的语句都会触发回调函数执行,即使pthread\_cleanup\_pop()不会被执行到 也会触发

4 线程的回收

pthread\_join

//main线程等待其他线程--->wait()

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

pthread\_t thread, 要回收的线程

void \*\*retval 回收的状态

5 线程的取消: pthread\_cancel

pthread\_setcancelstate, 取消状态

pthread\_setcanceltype, 取消类型

pthread\_testcancel //设置一个取消点

POSIX默认线程取消点:会引发堵塞的系统调用

取消一个线程 先cancel再join

6 线程的分离:pthread\_detach

int pthread\_detach(pthread\_t thread);

7 互斥锁(1的信号量)

多个线程使用同一资源会相互竞争,要注意线程执行函数的值传递和地址传递

使用互斥锁可以避免竞争出现的错误

//锁的初始化

pthread\_mutex\_init()

int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,

const pthread\_mutexattr\_t \*restrict attr);

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

一种动态实现,一种静态实现

pthread\_mutex\_t \*restrict mutex, 初始化的锁的地址

const pthread\_mutexattr\_t \*restrict attr 锁的属性 NULL 为默认属性

//锁不上一直等 上锁

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

//解锁  
int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex)

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

//摧毁一个锁

pthread\_mutex\_destroy()

临界区的跳转注意要unlock

pthread\_once:

//规定一个函数只执行一次

int pthread\_once(pthread\_once\_t \*once\_control,

void (\*init\_routine)(void));

pthread\_once\_t once\_control = PTHREAD\_ONCE\_INIT;

8线程条件变量

//初始化和销毁

pthread\_cond\_init

pthread\_cond\_destroy

//发送cond消息

pthread\_cond\_broadcast //发送给所有pthread\_cond\_wait

pthread\_cond\_signal //发送给一个pthread\_cond\_wait

//等待条件变量

pthread\_cond\_timedwait

pthread\_cond\_wait

条件变量是利用线程间共享的[全局变量](https://so.csdn.net/so/search?q=%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F&spm=1001.2101.3001.7020" \t "/home/ww/文档\\x/_blank)进行同步的一种机制，主要包括两个动作：一个线程等待"条件变量的条件成立"而挂起；另一个线程使"条件成立"（给出条件成立信号）。为了防止竞争，条件变量的使用总是和一个互斥锁结合在一起。

在调用pthread\_cond\_wait()前必须由本线程加锁（pthread\_mutex\_lock()），而在更新条件等待队列以前，mutex保持锁定状态，并在线程挂起进入等待前解锁。在条件满足从而离开pthread\_cond\_wait()之前，mutex将被重新加锁，以与进入pthread\_cond\_wait()前的加锁动作对应。

简单来说 pthread\_cond\_wait的功能就是：

解锁----->挂起----->（收到cond消息）----->加锁----->结束挂起

常见用法：

pthread\_mutex\_lock();

while(condtion)

{

pthread\_cond\_wait();

}

/\*

条件满足时要进行的操作

\*/

pthread\_mutex\_unlock();

9互斥量与信号量（信号量可以由锁封装得到）

互斥量可以看作信号为1的信号量，互斥量可以同来当作锁使用，也可以用来通知。

目前，在stm32上我的信号量只是用来通知的功能，未涉及到锁的使用。且在stm32进程通信仅仅涉及到两个线程对一个变量的操作（信号的作用）：

1写完，发信号 ------> 2写

而linux是大于两个线程对一个变量的操作（锁的作用）：

所有的线程都在一直抢锁，判断是否符合自己的条件，再写变量。

改用通知法：抢锁，不符合自己的条件就阻塞，释放锁，另一个抢锁，改变条件，发通知，解锁。

10 读写锁

11 线程属性

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

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

pthread\_attr\_setaffinity\_np(3), pthread\_attr\_setdetachstate(3),

pthread\_attr\_setguardsize(3), pthread\_attr\_setinheritsched(3),

pthread\_attr\_setschedparam(3), pthread\_attr\_setschedpolicy(3),

pthread\_attr\_setscope(3), pthread\_attr\_setstack(3),

pthread\_attr\_setstackaddr(3), pthread\_attr\_setstacksize(3),

pthread\_create(3), pthread\_getattr\_np(3),

pthread\_setattr\_default\_np(3), pthreads(7)

12 互斥量属性

pthread\_mutexattr\_init

pthread\_mutexattr\_destroy

//设置互斥量进程间共享（clone时共享一些资源有关）

pthread\_mutexattr\_getpshared

pthread\_mutexattr\_setpshared

//设置互斥量类型

pthread\_mutexattr\_gettype

pthread\_mutexattr\_settype

13 条件变量的属性

pthread\_condattr\_init

14 读写锁属性

15 重入

* 多线程IO

常用的为有锁的IO函数：

gets,gets,putc,pus ........

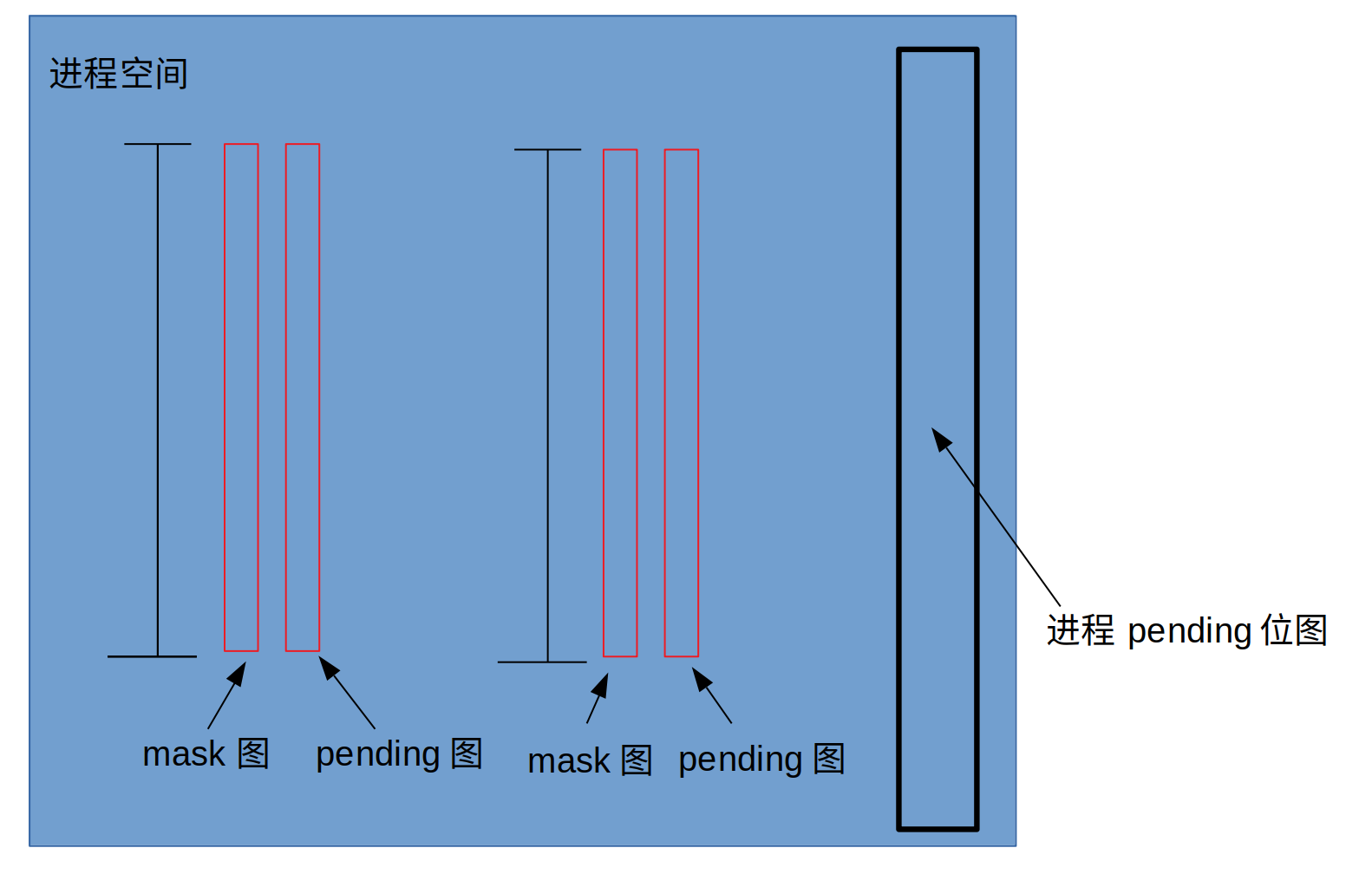
不加锁的IO函数如下：

getc\_unlocked, getchar\_unlocked, putc\_unlocked, putchar\_unlocked

* 多线程与进程与信号

每个进程只有进程级pending位图，每个线程有mask和线程级pending位图。

进程级pending位图记录进程收到的信号，线程pending图记录线程收到的信号。发生调度是mask分别与进程pending图和线程pending图按位与，判断接收到的信号。



函数：

pthread\_sigmask

sigwait

pthread\_kill

* 线程与fork

在线程中fork，POSIX标准中，线程中fork，得到的进程中只含有调用fork的那个线程。

OpenMP