线程基础：

进程是系统中程序执行和资源分配的最小单位。由于进程的地址空间是私有的，因此在进程间进行上下文切换时，系统开销比较大。为了提高系统性能，许多OS规范里引入轻量级进程(LWP：light weight process)的概念，也被称为线程(thread)。

通常，线程指的是共享地址空间的多个任务。在同一进程中创建的线程共享该进程的地址空间。

使用多线程的好处：

1. 大大提高任务切换的效率；
2. 避免了额外的TLB(快表---部分页表项)&cache(CPU---cache---memory)刷新。

注意：

1. 系统会为每一个用户进程创建一个task\_struct结构来描述该进程，该结构中包含一个指向该进程虚拟空间地址映射表的指针；实际上，task\_struct和地址空间映射表一起用来表示一个进程。(MMU就是通过地址空间映射表来实现虚拟地址和物理地址的转换的)
2. Linux里同样用task\_struct来描述一个线程，一个进程中的所有线程共用一份地址空间映射表。



一个进程中的多个线程可以共享以下资源：

1. 可执行的指令；
2. 静态数据；
3. 进程中打开的文件描述符；
4. 工作目录；
5. UID、GID

每个线程私有的资源如下：

1. 线程ID(TID)；
2. PC和相关寄存器；
3. 堆栈stack(局部变量和返回地址)；
4. 错误号errno；
5. 信号掩码和优先级；
6. 执行状态和属性。

线程库：

pthread\_create()：

|  |  |
| --- | --- |
| 头文件 | #include <pthread.h> |
| 原型 | int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,  void \*(\*start\_routine)(void \*), void \*arg); |
| 参数 | thread：创建的线程 |
| attr：指定线程的属性，NULL表示使用缺省属性 |
| start\_routine：线程函数的起始地址(函数指针) |
| arg：传递给线程函数的参数 |
| 返回值 | 成功：0 |
| 错误：返回错误码 |

注意：

start\_routine是一个指向返回值和参数类型均为void \*的函数的指针，即start\_routine存放的是线程函数的入口地址。在使用时要特别注意类型的匹配问题和对void \*指针的强制类型转换。

将一个其他类型的指针赋给void \*不需要进行强制类型转换，但在对一个void \*的指针进行运算和间接引用时必须进行强制类型转换。

pthread\_join()：

正如进程之间可以用wait()来处理僵尸进程一样，线程之间也有类似的机制。pthread\_join()可以用于将当前线程挂起等待其次线程的结束。这个函数是一个线程阻塞函数，调用它的函数将一直等到被等待的线程结束为止，当该函数返回时，被等待的线程的资源就被回收。

|  |  |
| --- | --- |
| 头文件 | #include <pthread.h> |
| 原型 | int pthread\_join(pthread\_t thread, void \*\*ptr); |
| 参数 | thread：要等待的线程 |
| ptr：二级指针，\*ptr指向被等待线程结束时的返回值(不为NULL时) |
| 返回值 | 成功：0 |
| 错误：返回错误码 |

注意：

函数传参时：传递一级指针，用于改变指针指向对象的内容；传递二级指针，用于改变一级指针的指向。

pthread\_exit()：

线程可通过调用pthread\_exit()来终止自身线程的运行。

|  |  |
| --- | --- |
| 头文件 | #include <pthread.h> |
| 原型 | void pthread\_exit(void \*ptr) |
| 参数 | ptr：一级指针，对同一进程中的调用pthread\_join()的线程是可用的 |
| 返回值 | void |

pthread\_cancel()：

在很多线程应用中，经常会遇到在别的线程中要终止另一个线程的问题，此时调用pthread\_cancel()可以实现这种功能。

|  |  |
| --- | --- |
| 头文件 | #include <pthread.h> |
| 原型 | int pthread\_cancel(pthread\_t thread); |
| 参数 | thread：要取消的线程 |
| 返回值 | 成功：0 |
| 错误：返回错误码 |