线程池概念和使用

概念：

通俗的讲就是一个线程的池子，可以循环的完成任务的一组线程集合

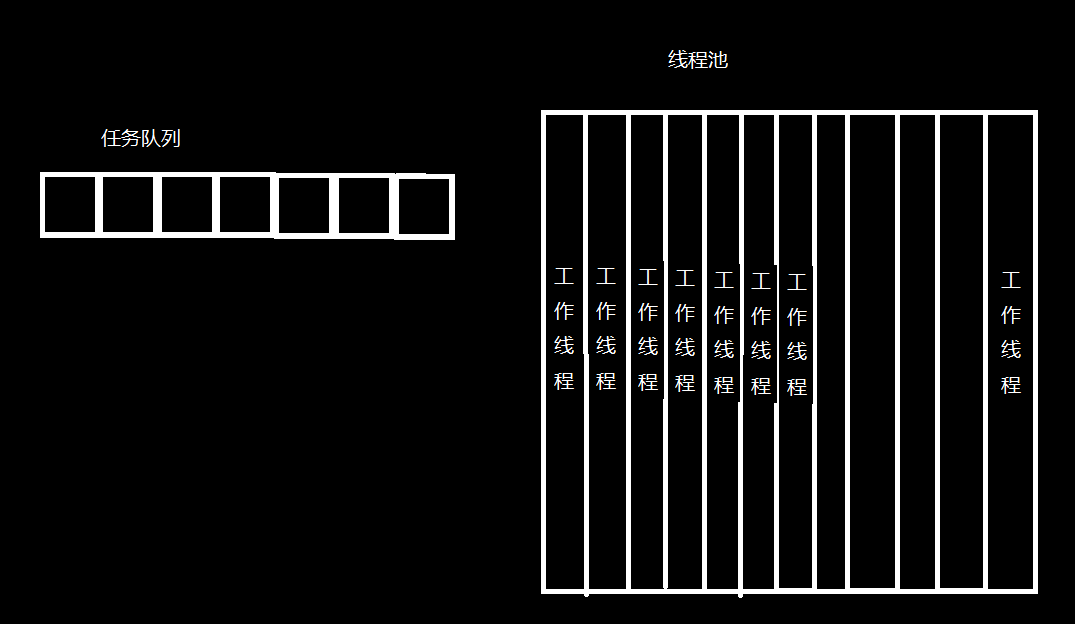
必要性：

我们平时创建一个线程，完成某一个任务，等待线程的退出。但当需要创建大量的线程时，假设T1为创建线程时间，T2为在线程任务执行时间，T3为线程销毁时间，当 T1+T3 > T2，这时候就不划算了，使用线程池可以降低频繁创建和销毁线程所带来的开销，任务处理时间比较短的时候这个好处非常显著。

线程池的基本结构：

1 任务队列，存储需要处理的任务，由工作线程来处理这些任务

2 线程池工作线程，它是任务队列任务的消费者，等待新任务的信号



线程池的实现：

1创建线程池的基本结构：

任务队列链表

typedef struct Task;

线程池结构体

typedef struct ThreadPool;

1. 线程池的初始化：

pool\_init()

{

创建一个线程池结构

实现任务队列互斥锁和条件变量的初始化

创建n个工作线程

}

1. 线程池添加任务

pool\_add\_task

{

判断是否有空闲的工作线程

给任务队列添加一个节点

给工作线程发送信号newtask

}

1. 实现工作线程

workThread

{

while(1){

等待newtask任务信号

从任务队列中删除节点

执行任务

}

}

1. 线程池的销毁

pool\_destory

{

删除任务队列链表所有节点，释放空间

删除所有的互斥锁条件变量

删除线程池，释放空间

}

编译错误：

error: ‘ThreadPool {aka struct ThreadPool}’ has no member named ‘head’

意义：ThreadPool 结构体没有head这个成员。

解决：检查是否拼写错误。

error: too few arguments to function ‘pthread\_mutex\_init’

意思：pthread\_mutex\_init这个函数参数少了

解决:检查函数的参数，添加对应的参数

线程的GDB调试：

显示线程

info thread

切换线程

thread id

GDB为特定线程设置断点

break location thread id

GDB设置线程锁，

set scheduler-locking on/off

on：其他线程会暂停。可以单独调试一个线程