**Multithreading Benefits**

考虑到虽然诸多挑战，多线程技术还在使用的原因是多线程技术有诸多益处。一些益处如下：

* 更好的资源利用率。
* 某些情景下更简单的程序设计。
* 更具有响应式的程序。

**Better resource utilization**

想像一个程序从本地文件系统读取和处理文件。比如说从磁盘读取一个文件花费5秒，处理它花费2秒。处理两个文件则花费

5 秒读取文件 A

2 秒处理文件 A

5 秒读取文件 B

2 秒处理文件 B

-----------------------

14 秒总共

当从磁盘读取文件时大部分CPU时间用来等待磁盘读取数据。在此期间CPU相当空闲。它可以做些别的事。通过修改操作顺序，CPU可以更好的被利用。看下这个顺序：

5 秒读取文件 A

5 秒读取文件 B + 2 秒处理文件 A

2 秒处理文件 B

-----------------------

12 秒总共

CPU等待第一个文件被读取。然后开始读取第二个文件。当第二个文件被读取时，CPU处理第一个文件。记住，当等待文件从磁盘中被读取时，CPU近乎空闲。

总之，CPU可以在等待IO时做其他事。不必是磁盘IO。也可以是网络IO，或者机器上一个用户的输入。网络和磁盘IO通常比CPU和内存慢得多。

**Simpler Program Design**

如果你准备亲手在一个单线程应用中编程上面的读取和处理的顺序性，你可能不得不追踪每个文件的读取和处理状态。或者你可以开两个线程，每个线程只处理读和处理一个单一文件。每个线程将会阻塞等待磁盘读取他的文件。等待时，其他的线程可以使用CPU来处理它们已经读取的的文件的部分。结果是，磁盘全时刻保持忙碌，读取文件到内存中。这导向更好的磁盘和CPU的利用。也更容易编程，因为每个线程只需要追踪单一文件。

**More responsive programs**

另一个将单线程应用转为多线程应用的主要目标是，获取一个更具有响应式的应用。想像一个服务型应用监听某个端口来接收请求。当接收到一个请求，处理请求然后返回给监听。服务器循环简述如下：

while(server is active){

listen for request

process request

}

如果请求采取较长时间来处理，那段时间没有新的客户端能够发送请求给服务器。只有当服务器正在监听，请求才能被收到。

另一种设计为监听线程传递请求给工作线程，然后立即返回监听。工作线程将会处理请求并发送回复给客户端。设计简述如下：

while(server is active){

listen for request

hand request to worker thread

}

这种方式服务器线程将会很快返回监听。因此更多的客户端可以发送请求给服务器。服务器变得更具有响应式。

对于桌面应用是一样的。如果你点击一个按钮开始一个长时间任务，执行任务的线程是更新窗口或者按钮等的线程，那么应用在任务执行期间将会不再响应。或者任务可以被交付给一个工作线程。当工作线程忙于该工作时，窗口线程是空闲的，可以响应其他用户的请求。当工作线程完成时发信号给窗口线程。窗口线程然后可以更新应用的窗口，利用任务的结果。具备工作线程设计的程序将对用户表现出更加的响应性。