线程的引入

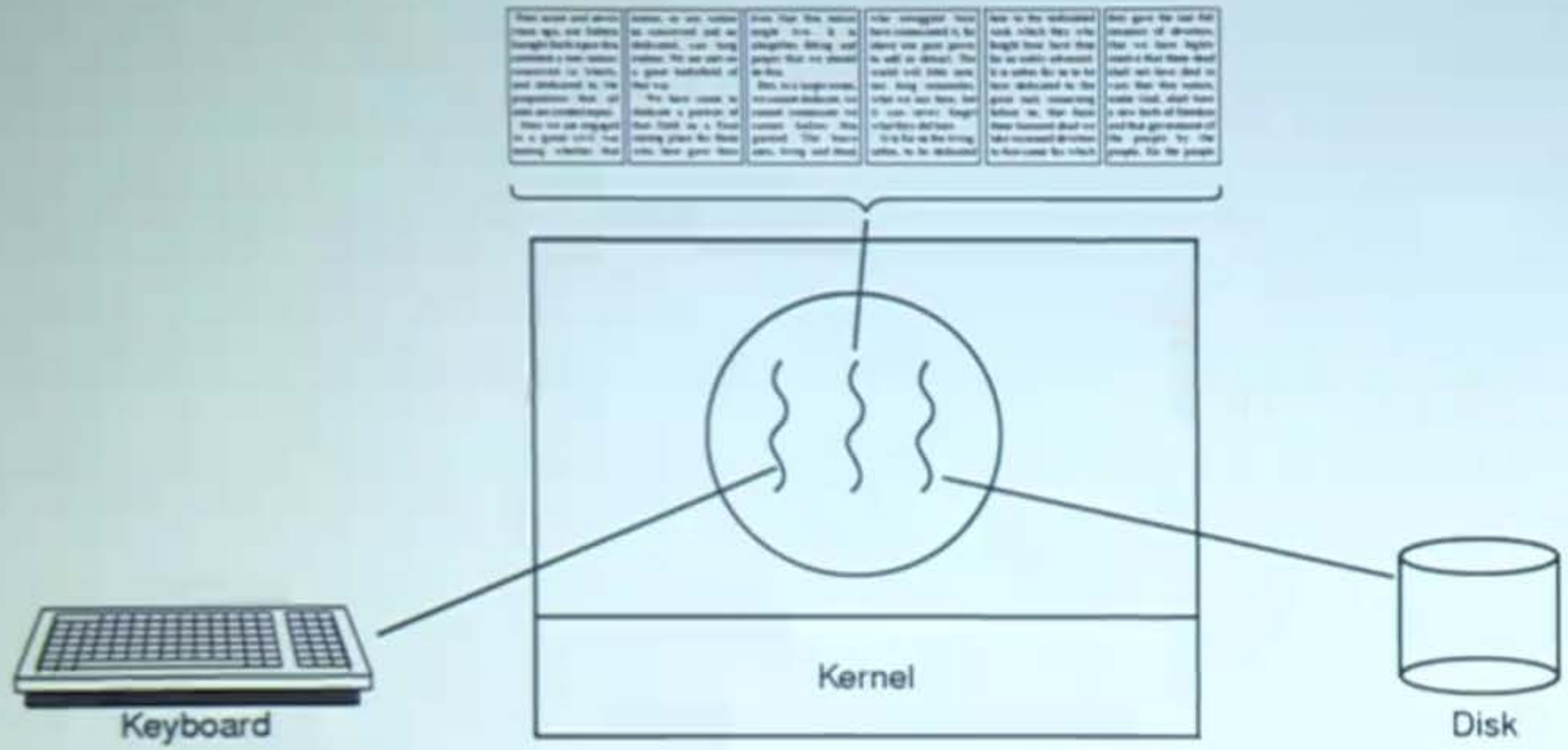
- 为什么在进程中再派生线程？

三个理由

- 应用的需要
- 开销的考虑
- 性能的考虑

第一个是应用的需要，第二个是开销的考虑 第三个呢是从性能的角度来看

应用的需要---示例1



有三个线程的字处理软件

很好地完成用户的需求 我们再举一个例子。

选自Tanenbaum《现代操作系

应用的需要——示例2(1/5)

- 典型的应用

  Web服务器

- 工作方式
  - 从客户端接收网页请求（http协议）
  - 从磁盘上检索相关网页，读入内存
  - 将网页返回给对应的客户端
- 如何提高服务器工作效率？

  网页缓存（Web page Cache）

服务器的一个工作效率 但是呢，我们来看到，如果没有线程

示例2(2/5)—如果没有线程？

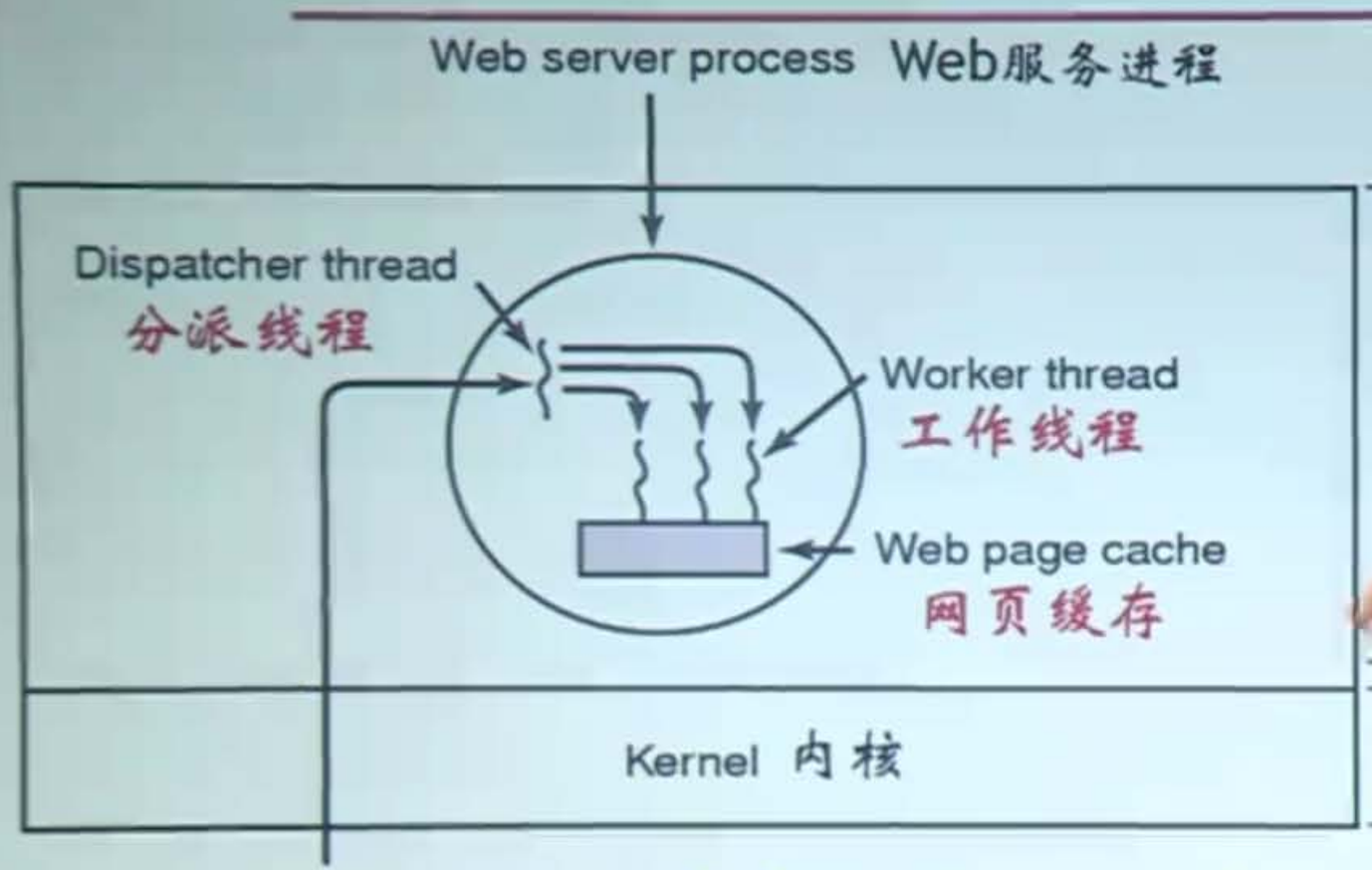
两种解决方案：

- 一个服务进程
  顺序编程；性能下降

- 有限状态机
  编程模型复杂；采用非阻塞I/O

因此，它的编程模型呢是比较复杂的 那么，我们能不能引进多线程呢？

示例2(3/5)



User space 用户空间

Kernel space 内核空间

一个多线程的Web服务器

选自Tanenbaum《现代操作系

示例2(4/5)

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

（a）分派线程

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

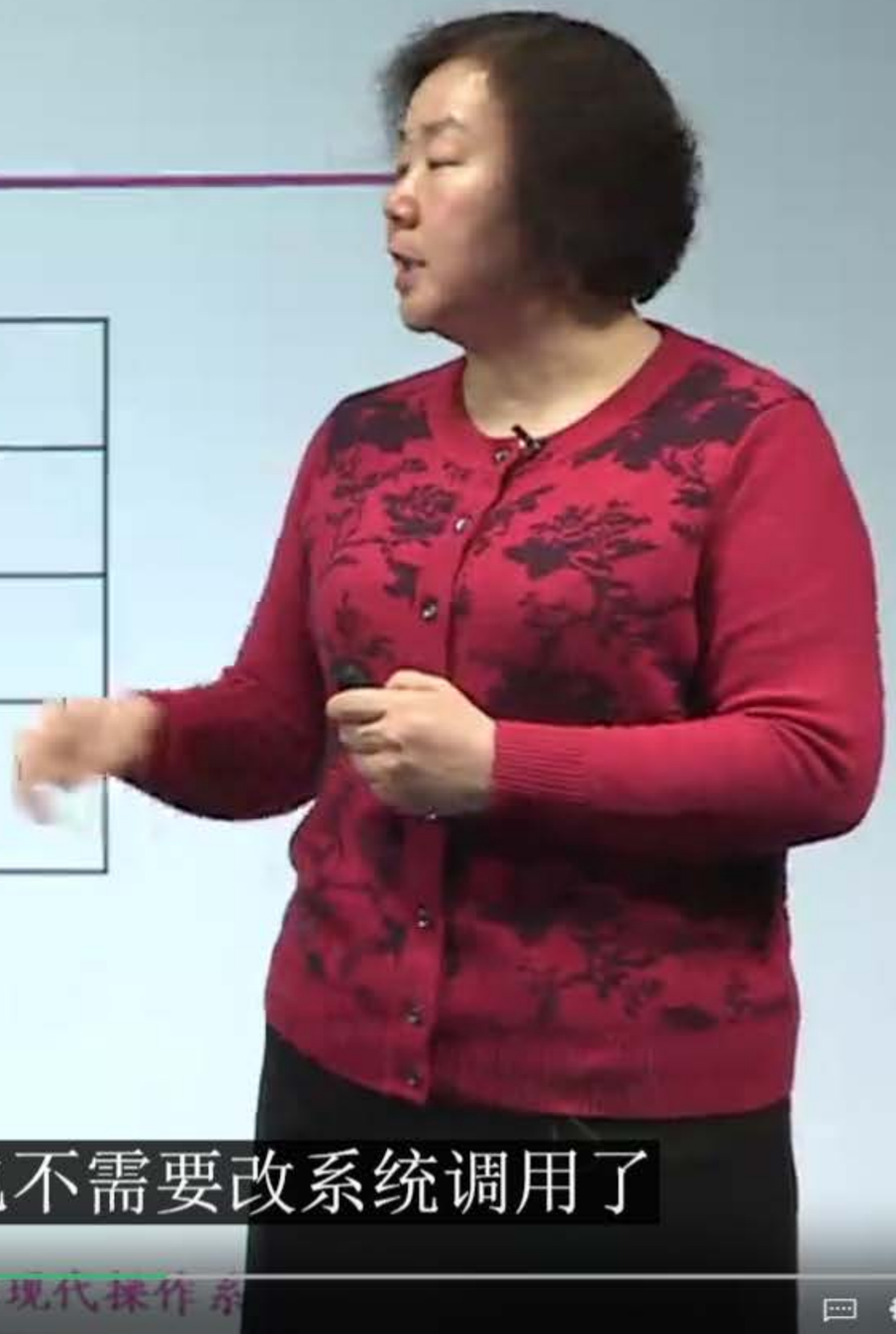
（b）工作线程

Web 服务器 那么这就是分派线程的工作和工作线程的工作。

示例2(5/5)

模型	特性
多线程	有并发、阻塞系统调用
单线程进程	无并发、阻塞系统调用
有限状态机	有并发、非阻塞系统调用、中断

构造服务器的三种方法



也很简单，直接去用阻塞系统调用，也不需要改系统调用了

选自Tanenbaum《现代操作系

2.开销的考虑

进程相关的操作：

- ✓ 创建进程
- ✓ 撤消进程
- ✓ 进程通信
- ✓ 进程切换

→ 时间/空间开销大，限制了并发度的提高

线程的开销小

- ✓ 创建一个新线程花费时间少（撤销亦如此）
- ✓ 两个线程切换花费时间少
- ✓ 线程之间相互通信无须调用内核（同一进程内的线程共享内存和文件）

那么这样的话呢，线程的各种操作呢，就带来了很高的效率。

3.性能的考虑

多个线程，有的计算，有的I/O

- 多个处理器

如何提高软件
运行性能?

那么这是从性能的角度看，引入线程有好处。

线程的基本概念

在同一进程
增加了多个
执行序列
（线程）

进程的两个基本属性

- 资源的拥有者
  - 进程还是资源的拥有者
- CPU调度单位
  - 线程继承了这一属性

线程：进程中的一个运行实体，是CPU的调度单

有时将线程称为轻量级进程

线程是一个运行实体，那么它有它自己的一些属性。

线程的属性

线程：

- 有标示符ID
- 有状态及状态转换 → 需要提供一些操作
- 不运行时需要保存的上下文
  有上下文环境：程序计数器等寄存器
- 有自己的栈和栈指针 √

- **共享所在进程的地址空间和其他资源**

- 可以创建、撤消另一个线程
  程序开始是以一个单线程进程方式运行的

0:00 下面我们谈一下，为什么引入线程 有了进程这概念之后 应用程序可以并发地去执行了 那么为什么要在进程当中再派生出线程呢？ 这里给出三个主要的理由 第一个是应用的需要，第二个是开销的考虑 第三个呢是从性能的角度来看 我们举两个例子来谈一下应用的需要 我们最常用的这种字处理软件 如果只有一个进程，那么当你去编辑，你去输入的时候 那么排版的工作就做不了，就不能够去存盘 因此，我们通常会在录入输入一些文档的过程中 再派生出其它的线程来完成其它的任务 也就是在一个字处理软件这样一个进程中进程当中派生出 其它的任务。 那么，一个任务完成的是管理键盘的输入 你敲的若干的字符，敲的汉字都能够被接受 那么如果你删除了某一行，那么这个 文档就要重新做排版，那么，专门有一个任务是负责排版的 比如说你在 600 页删掉了一行 现在你要去处理 800 页的某一个段落 那么，排版的这个工作任务在执行 然后你可以到后面的地方去做其它的工作 如果我们为了使得我们的工作不至于丢失，所以我们会启动一个 定时保存。 因此，到点了以后，那么这个任务就开始工作，把 相应的内容保存在磁盘上。 所以，这是一个非常典型的一个 字处理软件的应用需要若干个任务来完成 很好地完成用户的需求 我们再举一个例子。 这个例子也是比较典型，是 Web 服务器 可以查询各种各样的网页。 那我们来看看 Web 服务器的工作方式 首先呢，它要从客户端来接收用户对网页的这个申请 通过 http 协议，用户可以发来一些查询网页的申请 那么，Web 服务器得到这个申请之后呢 就要到磁盘上去搜索相关的网页 那么这个磁盘的操作就使得 进程暂时停止，不能运行了。 因为要到磁盘上去搜索 当磁盘的内容完成之后，那么把内容读入内存 然后这个进程才能继续去执行。 那么进程把 从磁盘返回的网页，返回给对应的客户端 这个用户就得到了他所查询的结果。 那么每次 到磁盘上搜索相关的网页，那么 进程就会停在那里，这样的话性能就比较慢。 怎么样去提高 服务器的工作的效率呢？ 通常情况下，一个常用的手段是 在服务器的内存里头，开辟一个网页的缓存 Web page Cache，保存了常用的常被查找的这些网页 那么当 Web 服务器从客户端接收了网页请求之后呢 先到网页缓存当中去查找 如果找到了，直接把结果返回给客户端 就不用到磁盘上去找了。 但是如果没找到的话呢，就先 到磁盘上去搜索相关的网页，得到了之后呢 填入写入网页缓存 然后再把结果返回给用户。 那么，通过网页缓存这个技术呢可以提高 Web 服务器的一个工作效率 但是呢，我们来看到，如果没有线程 那么 Web 服务器的工作呢，尽管采用了 网页缓存，那么还是性能不高 我们来看一下为什么。 如果 只有一个服务进程，没有线程的情况下，那我们只能设定一个服

务进程 那么说，如果设定多个服务进程可不可以啊？但是它 多个服务进程，每个进程有自己独立的地址空间，所以它不能共享信息，所以不可以 所以我们说一个服务进程，这个服务进程呢只能是顺序编程 也就是说，如果它到磁盘上去搜寻网页，它就不能再去接收客户端的请求 因此造成了这种服务器的性能呢是下降的 那么如果说我采用一些特殊的手法来解决 这个问题呢，比如说，我们采用了有限状态机的方法 所谓有限状态机的方法呢实际上是用一个复杂的编程模型来 自己模拟一些并发的工作，就进程自己来模拟并发的工作。 比如说 接收了一个用户请求之后，如果要到 磁盘上搜寻这个网页，那么原本这个进程会被 暂停，那么这个时候呢，我们就要改造这个搜寻网页的 操作，把它改造成一个叫做非阻塞的 I/O 也就是到磁盘上去搜寻网页 那么这个进程还可以继续做，与这个网页内容 无关的一些工作。 所以它叫非阻塞 I/O，当然你这要改变这个 I/O 的一些操作 那么当一个进程调用了一个 去调用一个非阻塞 I/O，去查询网页的时候，那么 磁盘在工作，那么这个进程就可以回来做别的事情，它就可以继续去接收 用户的客户端请求。 那么接收请求之后，有个请求来了，那么它 继续可能在 Web Cache 里找到了这网页，就返回去，如果没找到网页呢，继续再去调用磁盘 但是这里头就有个问题，如果 磁盘的这个结果返回了，究竟是哪一个客户端的请求呢？ 所以这个时候，这个进程要自己把这些信息记录下来 然后磁盘返回了请求之后，就要判断是哪一个请求 哪一个客户端的请求，然后返回给对应的客户 端。 因此，它的编程模型呢是比较复杂的 那么，我们能不能引进多线程呢？ 好，我们来看一下在一个多线程的这样一个 Web 服务器，它的工作方式是什么样的 那么把线程分成两类 一类线程呢叫分派线程，一个就够啦，分派线程的主要工作就是监听客户端 客户端只要有个请求就把请求读进来 但是它不完成客户端的请求，它把这个请求分派给其它的线程来完成 那么就是工作线程。 有一堆工作线程在 Web 服务器上 那么它呢都完成的是服务客户请求的 所以，分派线程获得了客户端请求之后就把它分给某一个工作线程 那么这个工作线程呢还跟前面一样，先到 Web Cache 这个里头 去查找这个网页是不是在里头，如果在里头就返回给 客户端，如果不在里头就要启动磁盘，到磁盘上去搜寻网页 那么它到磁盘上搜寻网页，那么这是一个阻塞的 I/O，那它就被阻塞，等待 但是没有关系，因为还有其它的工作线程。 也就是说分派线程可以把活派 给其它的工作线程。 所以，分派线程接活，然后分派给工作线程干活 那么这就是一个多线程的一个 Web

服务器 那么这就是分派线程的工作和工作线程的工作。 我们小节一下，一个 Web 服务器的这个实现有三种方法 没有线程引入的时候，我们可以用单线程进程的方法 但这样的话呢，我们用的是阻塞的系统调用，同时呢，没有并发 或者是我们采用比较复杂的编程 模型，用有限状态机的方法来实现，那么，可以有并发 那么你要改造系统调用，把它改造成一个非阻塞系统调用 你还要和中断处理结合起来 来完成整个的工作。 所以编程模型比较复杂 那么有了多线程，首先系统的并发度提高了 一个进程里头又可以并发起来，然后它的编程模型 也很简单，直接去用阻塞系统调用，也不需要改系统调用了 因此呢，引入多线程之后有很多的好处 那么我们再从开销的角度来看线程比进程的优势。 那么进程有很多操作，这些操作呢，往往是时间、空间都耗费比较多。 那么线程的开销就小得多了，比如说 创建一个新的线程或者撤消一个新的线程，花费的时间比较少。 两个线程切换，只需要几条指令就可以完成了。 同时呢，线程之间其实通信是没有 不需要内核的介入，直接就可以相互通信了。 那么这样的话呢，线程的各种操作呢，就带来了很高的效率。 所以从开销角度上讲，线程比进程有优势。 另外呢，我们如何提高一个软件的这个运行的性能呢？ 如果一个进程里头又有多个线程，有这些线程，有的计算，有的去 I/O，当有多个处理器的时候，这样就可以充分发挥这个 优势了。 所以当多处理器的情况下，一个进程就可以有很多的任务同时在执行。 那么这是从性能的角度看，引入线程有好处。 那么有了线程这样一个引入 的动机，我们来看看线程的模型应该是什么样的。 首先我们来看看，线程它所基于的一个、这个 原始的这个进程。 那么进程 有两个基本的属性，一个呢，进程是资源的拥有者， 同时进程也是 CPU 的一个调度单位。 但是有了线程之后，线程就继承了 这样一个属性，也就是线程成为了 CPU 的调度单位。 而进程呢，依然还是 管理资源，然后是资源的一个拥有者。 所以线程是什么呢？线 程实际上呢，是进程中的一个运行实体。 从运行的角度，它是一个运行的实体，它是一个 CPU 的调度单位。 有的时候，我们把线程呢、早期把它称之为轻量级进程。 也就是说，在进程当中，又增加了多个执行序列， 让这些执行序列可以并发执行，以提高软件的运行效率。 所以强调的是，增加了多个执行序列，叫线程。 线程是一个运行实体，那么它有它自己的一些属性。 这些属性呢，属于线程 自己，所以不同的线程呢，实际上有不同的值。 首

先要有一个唯一的标示，ID，所以同一个进程的不同线程你要区分。那么线程是上 CPU 的，所以它有状态。它也有状态的转换，也需要提供一些针对线程的操作。在线程不运行的时候，要保存 上下文环境，因此呢要有一些相关的寄存器，要保存在线程的相对的数据结构里头。线程在执行过程中，需要自己的栈，有自己的栈 指针。那么这些内容，是不同的线程是不一样的信息。但是同一个进程的不同线程呢，它们是共享所在进程地址空间的 内容和这个进程所拥有的资源的，所以这是非常重要的，也就是 线程之间的通信，或者其他的一些操 作带来了便利的地方，它们是共享同一个进程的 地址空间和有关的资源。那么线程当然也可以创建、撤消另一个线程。因为什么呢？因为我们一开始，当创建 进程以后，实际上是只有一个线程，我们称之为一个主线程。然后之后由它再创建其他的线程，所以程序开始 的时候呢，我们可以看成是一个单线程的进程在运行。