22 | 热点问题答疑（2）：内核如何阻塞与唤醒进程？

2019-06-29 李号双

讲述：李号双
时长 07:15 大小 6.65M

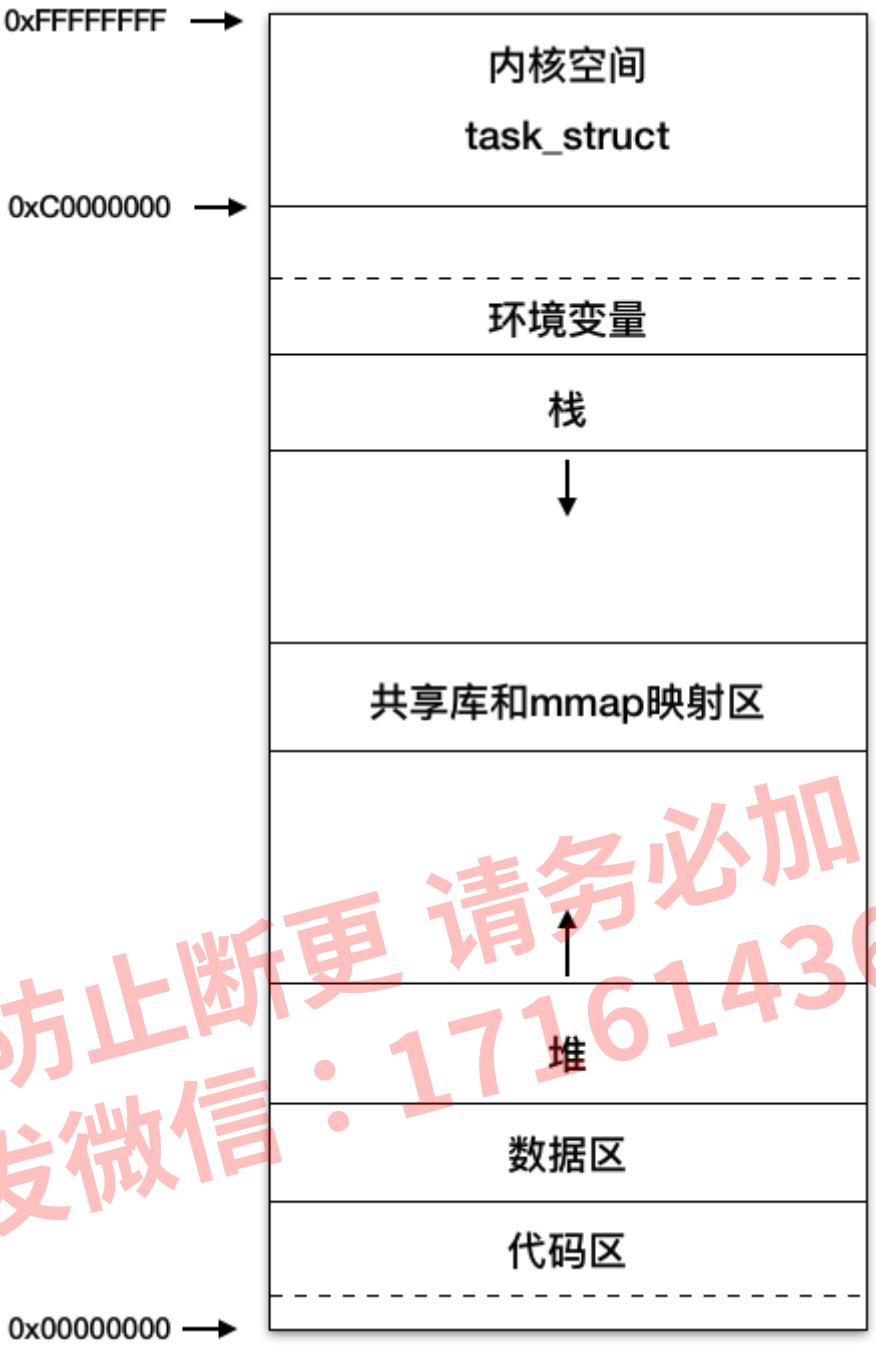
在专栏的第三个模块，我们学习了 Tomcat 连接器组件的设计，**其中最重要的是各种 I/O 模型及其实现**。而 I/O 模型跟操作系统密切相关，要彻底理解这些原理，我们首先需要弄清楚什么是进程和线程，什么是虚拟内存和物理内存，什么是用户空间和内核空间，线程的阻塞到底意味着什么，内核又是如何唤醒用户线程的等等这些问题。可以说掌握这些底层的知识，对于你学习 Tomcat 和 Jetty 的原理，乃至其他各种后端架构都至关重要，这些知识可以说是后端开发的“基石”。

在专栏的留言中我也发现很多同学反馈对这些底层的概念很模糊，那今天作为模块的答疑篇，我就来跟你聊聊这些问题。

进程和线程

我们先从 Linux 的进程谈起，操作系统要运行一个可执行程序，首先要将程序文件加载到内存，然后 CPU 去读取和执行程序指令，而一个进程就是“一次程序的运行过程”，内核会给每一个进程创建一个名为`task_struct`的数据结构，而内核也是一段程序，系统启动时就被加载到内存中了。

进程在运行过程中要访问内存，而物理内存是有限的，比如 16GB，那怎么把有限的内存分给不同的进程使用呢？跟 CPU 的分时共享一样，内存也是共享的，Linux 给每个进程虚拟出一块很大的地址空间，比如 32 位机器上进程的虚拟内存地址空间是 4GB，从 0x00000000 到 0xFFFFFFFF。但这 4GB 并不是真实的物理内存，而是进程访问到了某个虚拟地址，如果这个地址还没有对应的物理内存页，就会产生缺页中断，分配物理内存，MMU（内存管理单元）会将虚拟地址与物理内存页的映射关系保存在页表中，再次访问这个虚拟地址，就能找到相应的物理内存页。每个进程的这 4GB 虚拟地址空间分布如下图所示：



进程的虚拟地址空间总体分为用户空间和内核空间，低地址上的 3GB 属于用户空间，高地址的 1GB 是内核空间，这是基于安全上的考虑，用户程序只能访问用户空间，内核程序可以访问整个进程空间，并且只有内核可以直接访问各种硬件资源，比如磁盘和网卡。那用户程序需要访问这些硬件资源该怎么办呢？答案是通过系统调用，系统调用可以理解为内核实现的函数，比如应用程序要通过网卡接收数据，会调用 Socket 的 read 函数：

复制代码

```
1 ssize_t read(int fd,void *buf,size_t nbyte)
```

CPU 在执行系统调用的过程中会从用户态切换到内核态，CPU 在用户态下执行用户程序，使用的是用户空间的栈，访问用户空间的内存；当 CPU 切换到内核态后，执行内核代码，使用的是内核空间上的栈。

从上面这张图我们看到，用户空间从低到高依次是代码区、数据区、堆、共享库与 mmap 内存映射区、栈、环境变量。其中堆向高地址增长，栈向低地址增长。

请注意用户空间上还有一个共享库和 mmap 映射区，Linux 提供了内存映射函数 mmap，它可将文件内容映射到这个内存区域，用户通过读写这段内存，从而实现对文件的读取和修改，无需通过 read/write 系统调用来读写文件，省去了用户空间和内核空间之间的数据拷贝，Java 的 MappedByteBuffer 就是通过它来实现的；用户程序用到的系统共享库也是通过 mmap 映射到了这个区域。

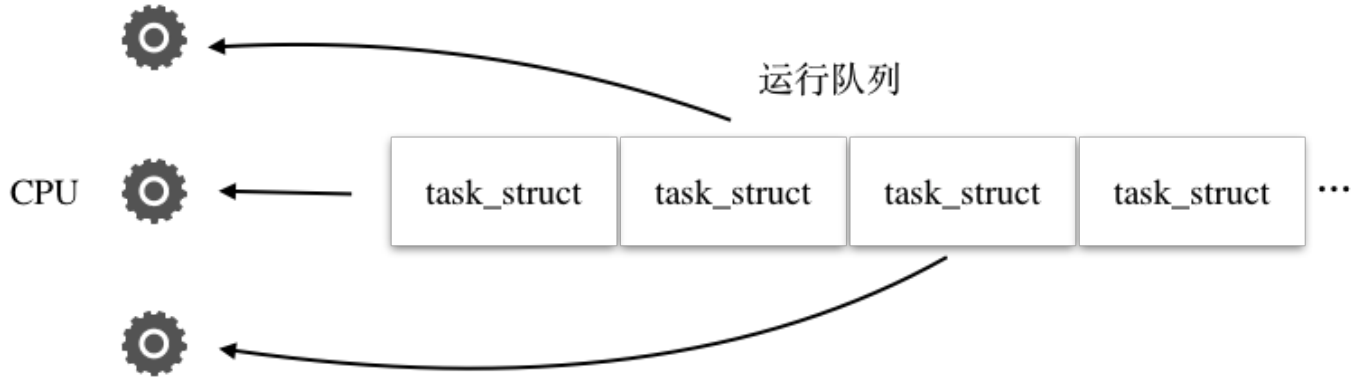
我在开始提到的`task_struct`结构体本身是分配在内核空间，它的`vm_struct`成员变量保存了各内存区域的起始和终止地址，此外`task_struct`中还保存了进程的其他信息，比如进程号、打开的文件、创建的 Socket 以及 CPU 运行上下文等。

在 Linux 中，线程是一个轻量级的进程，轻量级说的是线程只是一个 CPU 调度单元，因此线程有自己的`task_struct`结构体和运行栈区，但是线程的其他资源都是跟父进程共用的，比如虚拟地址空间、打开的文件和 Socket 等。

阻塞与唤醒

我们知道当用户线程发起一个阻塞式的 read 调用，数据未就绪时，线程就会阻塞，那阻塞具体是如何实现的呢？

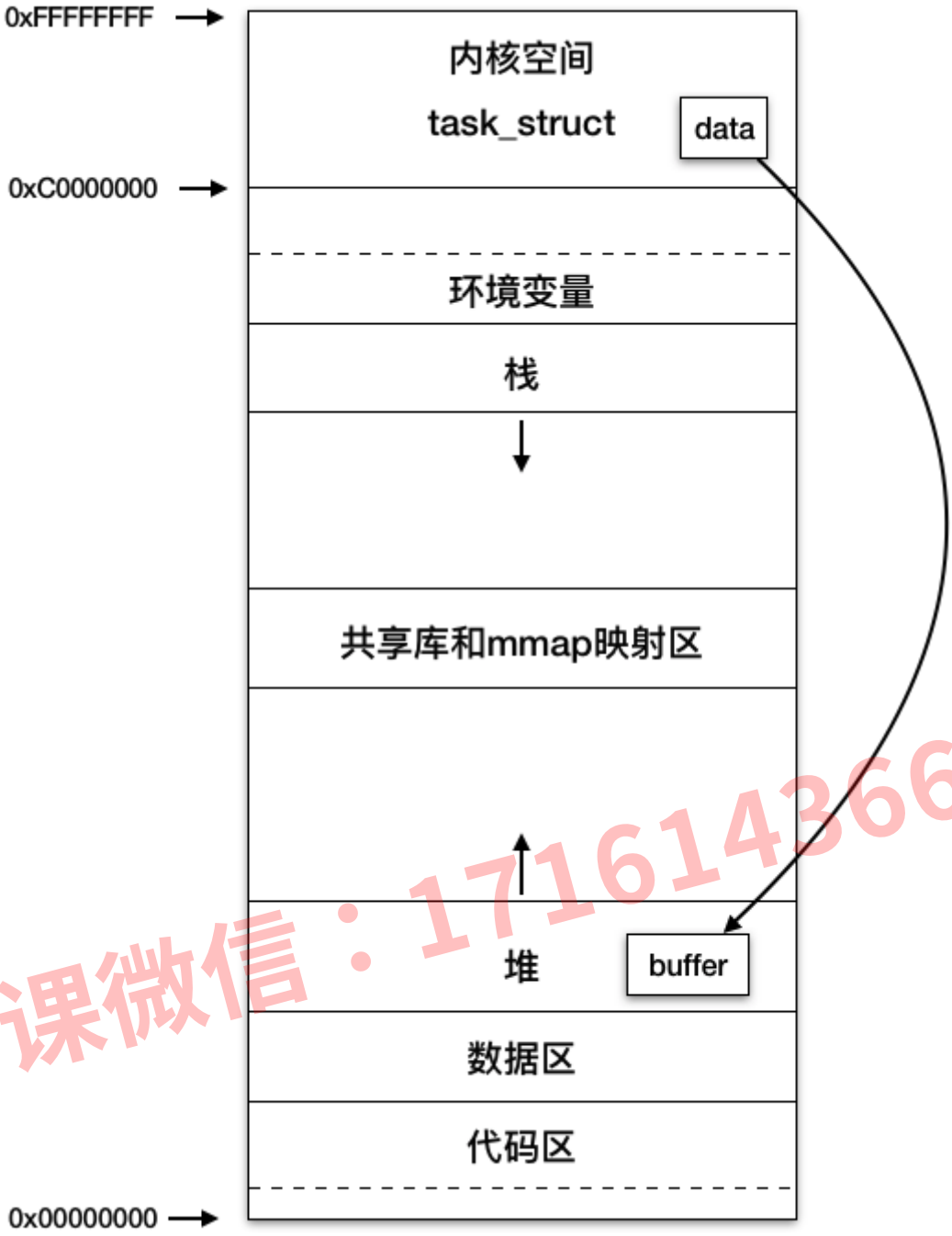
Linux 内核将线程当作一个进程进行 CPU 调度，内核维护了一个可运行的进程队列，所有处于`TASK_RUNNING`状态的进程都会被放入运行队列中，本质是用双向链表将`task_struct`链接起来，排队使用 CPU 时间片，时间片用完重新调度 CPU。所谓调度就是在可运行进程列表中选择一个进程，再从 CPU 列表中选择一个可用的 CPU，将进程的上下文恢复到这个 CPU 的寄存器中，然后执行进程上下文指定的下一条指令。



而阻塞的本质就是将进程的`task_struct`移出运行队列，添加到等待队列，并且将进程的状态的置为`TASK_UNINTERRUPTIBLE`或者`TASK_INTERRUPTIBLE`，重新触发一次 CPU 调度让出 CPU。

那线程怎么唤醒呢？线程在加入到等待队列的同时向内核注册了一个回调函数，告诉内核我在等待这个 Socket 上的数据，如果数据到了就唤醒我。这样当网卡接收到数据时，产生硬件中断，内核再通过调用回调函数唤醒进程。唤醒的过程就是将进程的`task_struct`从等待队列移到运行队列，并且将`task_struct`的状态置为`TASK_RUNNING`，这样进程就有机会重新获得 CPU 时间片。

这个过程中，内核还会将数据从内核空间拷贝到用户空间的堆上。



当 read 系统调用返回时，CPU 又从内核态切换到用户态，继续执行 read 调用的下一行代码，并且能从用户空间上的 Buffer 读到数据了。

小结

今天我们谈到了一次 Socket read 系统调用的过程：首先 CPU 在用户态执行应用程序的代码，访问进程虚拟地址空间的用户空间；read 系统调用时 CPU 从用户态切换到内核态，执行内核代码，内核检测到 Socket 上的数据未就绪时，将进程的`task_struct`结构体从运行队列中移到等待队列，并触发一次 CPU 调度，这时进程会让出 CPU；当网卡数据到达时，内核将数据从内核空间拷贝到用户空间的 Buffer，接着将进程的`task_struct`结构体

重新移到运行队列，这样进程就有机会重新获得 CPU 时间片，系统调用返回，CPU 又从内核态切换到用户态，访问用户空间的数据。

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。

上一篇　21 | 总结：Tomcat和Jetty的高性能、高并发之道

精选留言 (5)

写留言

刘章周
2019-06-29

老师，每节课后的思考题什么时侯也答疑下

作者回复: 嗯，基本上每篇的留言回复里都能找到答案

2

-W.LI-
2019-06-29

感谢老师，万分感谢。上次有个问题我不明白，老师还帮我查阅源码确认了。李老师，还有http那个老师是最最负责的真的万分感谢。
向老师，我要把计算机组成原理和操作系统自己看一遍看不懂就看两遍。

作者回复: 😁

1

nightmare
2019-06-30

老师今天讲了线程和进程，进程和线程都是统一在内核空间建立task_truct，根据代码是否有系统调用在用户态和内核态来做上下文切换，然后还讲了read的系统调用过程以及进程的虚拟内存和物理内存的机制，有一点没明白，是每个进程都会有一个虚拟内核空间吗？然后进程的虚拟内核空间映射到系统管理的内核空间上？

飞翔
2019-06-29

用户态和用户空间是啥关系？

作者回复: 你可以理解为CPU上有个开关，可以设置CPU的工作模式：用户态和内核态。在用户态模式下访问用户空间，也就是低地址的3GB。

Liam
2019-06-29

如果是通过mmap读数据，流程是怎样的呢？

1 如果没有数据，是否会阻塞？
2 不需要拷贝数据？意思是用户进程可以直接读mmap，不需要拷贝到堆吗？

作者回复: mmap不支持Socket读写，只支持磁盘文件。

通过mmap将文件映射到内存后，直接写读写内存，内核会负责将数据刷新到磁盘文件。