南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS
计算机科学与技术学院
/人工智能学院

# 操作系统课程作业

162140222 黄钰轩

2024 年 4 月 17 日

**题目 1.** Go 语言是一门非常优秀的语言，具有与 C、CPP 同等量级的性能，与此同时规避了不安全指针、内存泄露等问题. 然而，Go 语言最大的特色是从语言层面支持并发 (Goroutine)，调研 Goroutine 的底层原理，形成研究报告.

**解答.** Go 语言中，每一个并发活动被称为 goroutine. 可以将 goroutine 理解为一个较为智能的线程，能够自行完成调度，合理占用多核的 CPU. 当启动一个 Go 程序的时候，会有一个 goroutine 调用 main 函数，这个 goroutine 被称为**主 goroutine**，通过

$$\text{go} < \text{函数名} >(\text{参数})$$

的方式新建 goroutine，执行调用的函数.

事实上，Goroutines 使用了一个已经存在了一段时间的概念，称为“协程”. 它实质上会将用户级别运行的一组独立执行的函数——也就是“协程”，多路复用到操作系统级别的一组实际线程上，这就是 Goroutines 非常高效的原因.

当一个协程存在阻塞，Go 运行时会自动将和阻塞协程在同一线程上面的排队等候的其他协程移动到其他不同的、可运行的线程上面，这样它们就不会被阻塞. 可以用一个简短的案例来演示：

- 如图1所示，假设有 2 个内核级线程与 1 个处理器，同时有 4 个 Goroutines 需要多路复用到内核级线程才能执行.

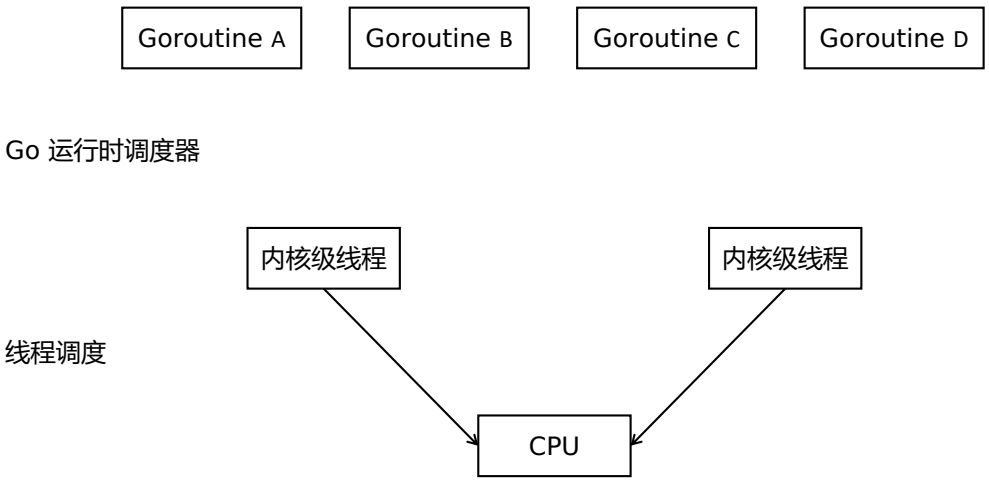


图 1: 条件假设

- 现假设出于某些原因，Go 的运行时调度程序决定如图2这样将 Goroutines 多路复用到线程上. Goroutine A 和 B 将使用左侧的内核线程运行，Goroutines C 和 D 将使用右侧的内核线程运行，同时假设 Goroutines 将从放置在 Goroutines 池底部的 Goroutine 开始分别运行.

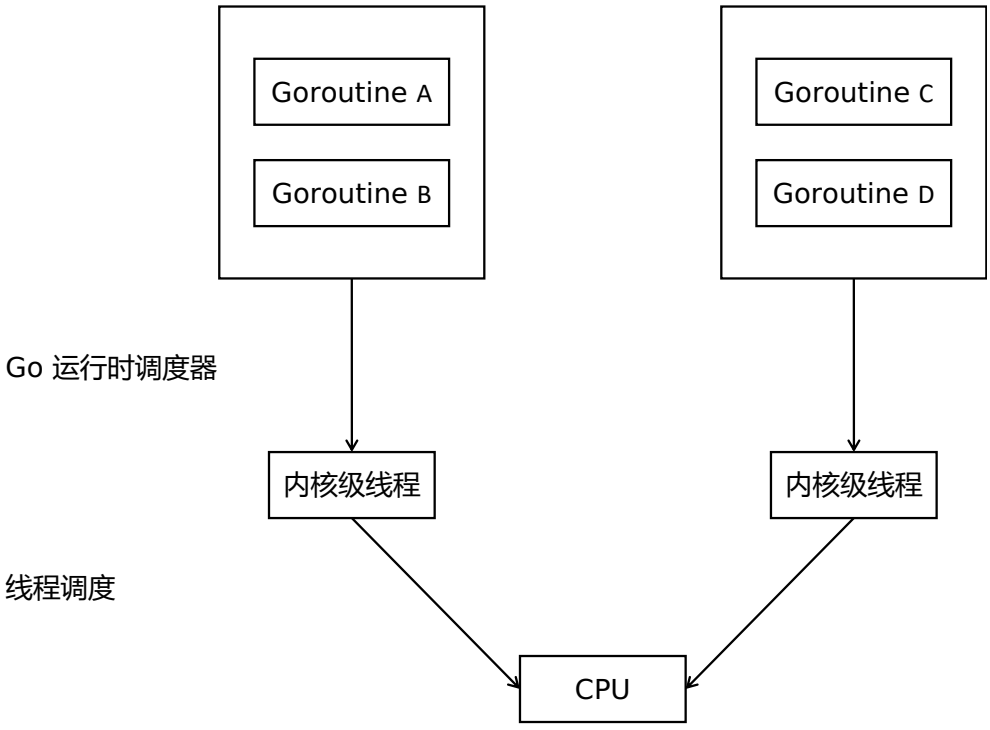


图 2: 初始情况

- 现在 Goroutines 开始运行。但此时“A”调用了一个阻塞的系统调用 (例如 read() 语句)，导致 Goroutine A 处于阻塞中，Goroutine B 被阻塞.

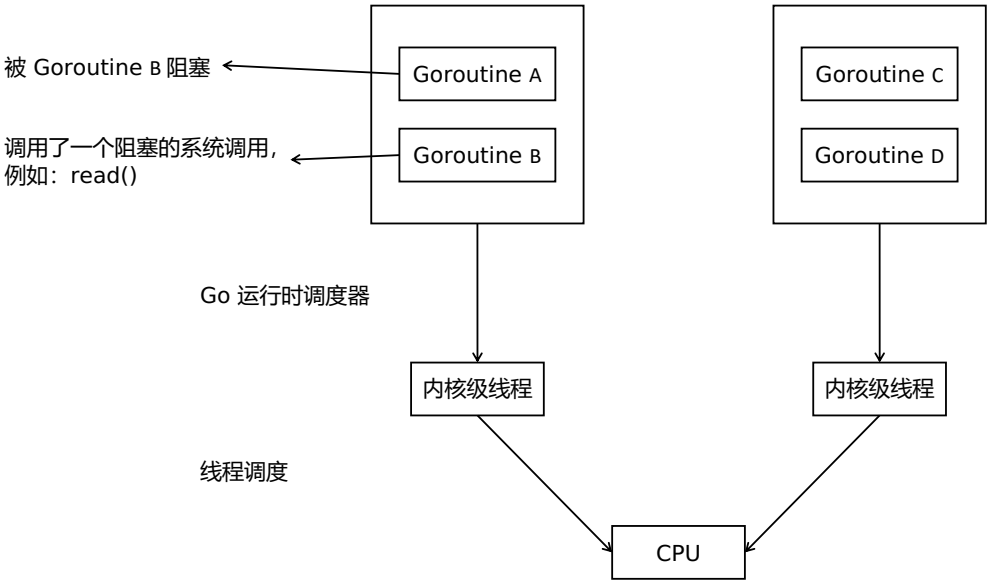


图 3: 阻塞出现

- Goroutine D 已经执行完毕.

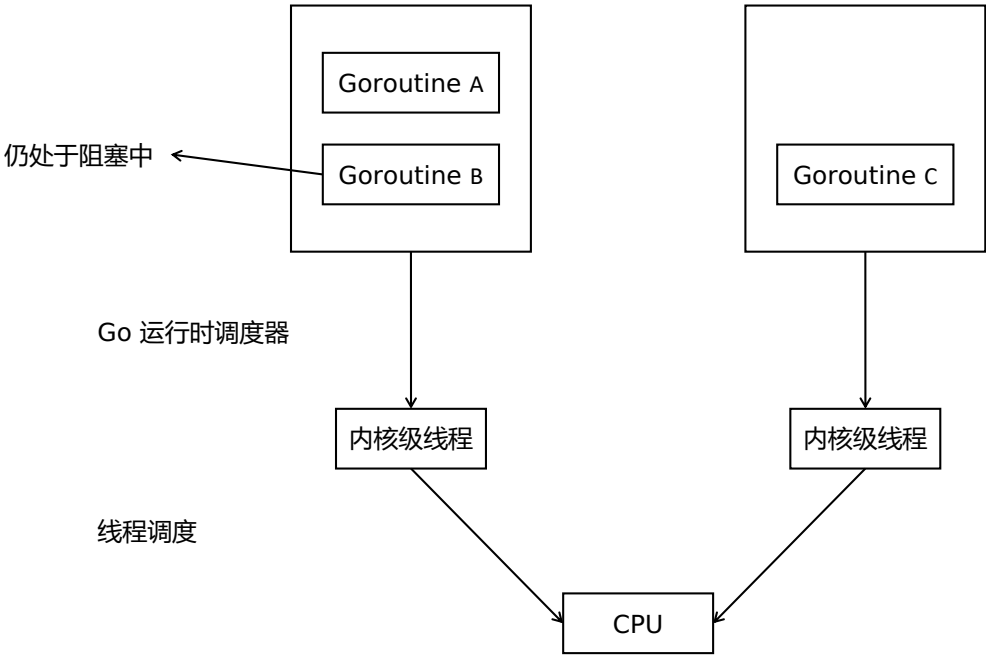


图 4: Goroutine D 执行完毕

- Go 的运行时调度程序现在可能会意识到，将 Goroutine A 移动到另一个可运行线程而不是永远等待 Goroutine B 执行完毕会是一个更好的主意.

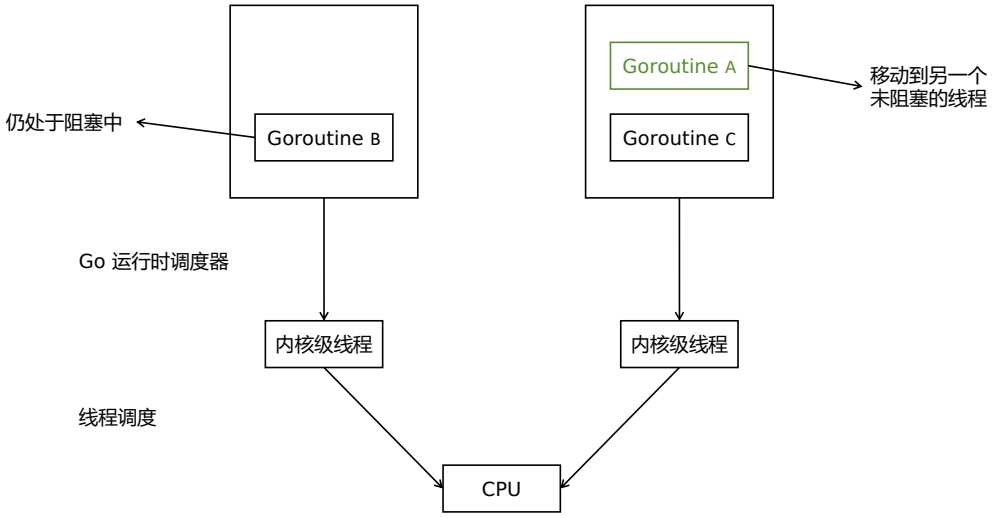


图 5: 将 B 移动到另一个 Goroutine 池

- Goroutine C 现在已经执行完毕.

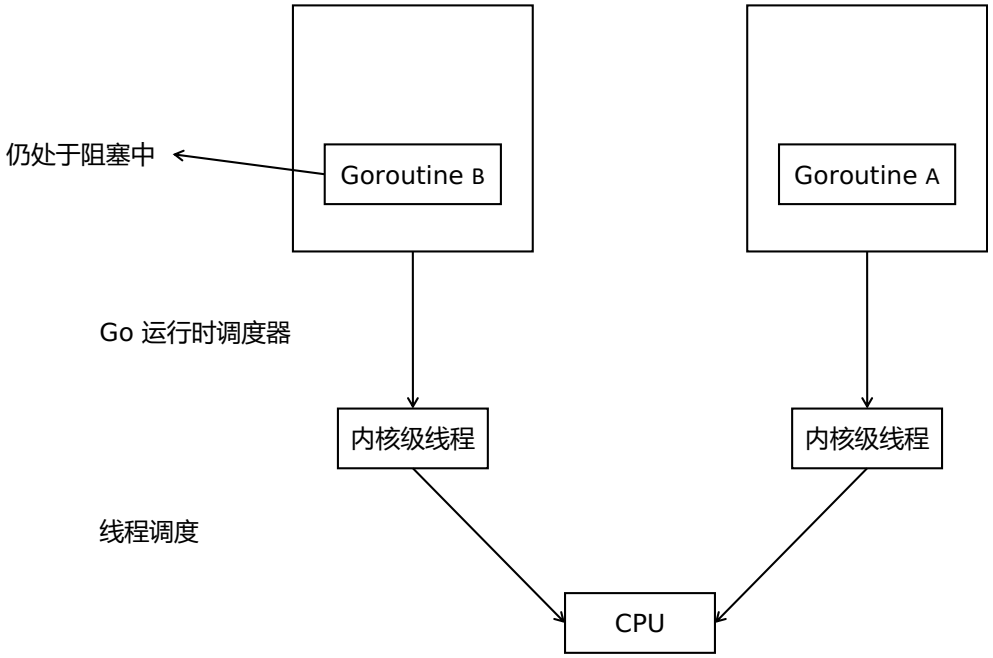


图 6: Goroutine C 执行完毕

- Goroutine A 现在也完成了，而左边被 Goroutine B 占用的内核线程还处于阻塞中的状态.

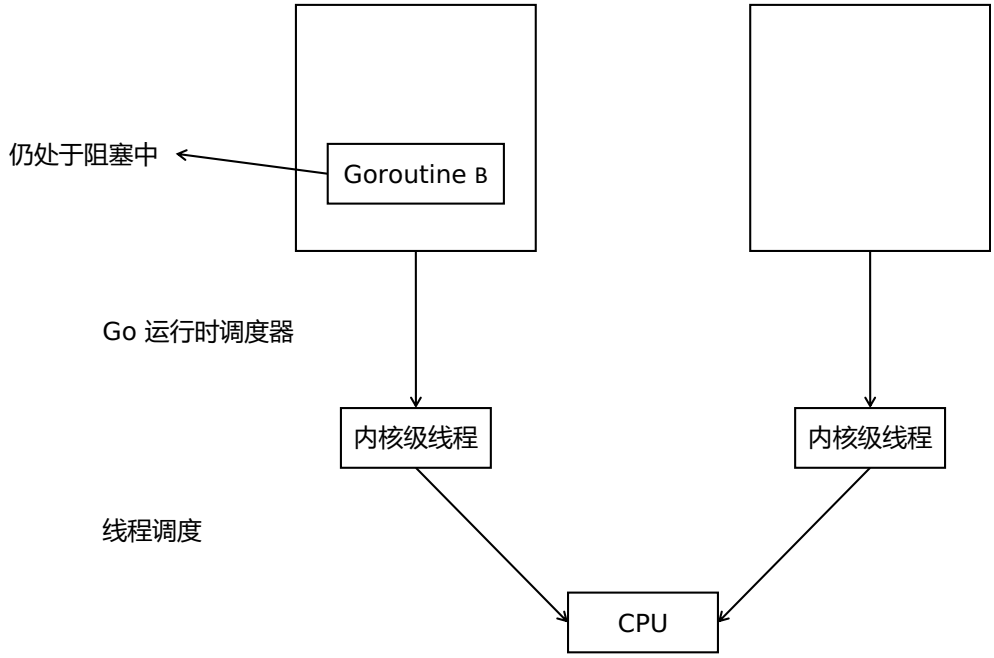


图 7: Goroutine A 执行完毕

如果 Go 运行时调度器没有将阻塞的 Goroutine 重新调度到另一个可运行的线程，那么 Goroutine A 到现在仍然不会被执行.