忙等待和CPU调度

在本练习中,您可以做两件事。您可以根据经验测量 EOS linux 机器使用的量子,并且您可以理解作为同步技术的繁忙等待有多么糟糕。

我正在为您提供部分实现 pingpong.c,以帮助您入门。它主要只包含一个全局变量
和一个预处理器常量。您可以自己编写有趣的部分。你的工作是创建两个新线程,一个
一个叫 ping,一个叫 pong。 ping 线程将运行左边的代码然后退出,pong 线程将运行
右边的代码然后退出。创建这些线程后,主线程将等待它们完成。你的
程序不需要打印任何东西。我们只对运行需要多长时间感兴趣。

平

```
for ( int i = 0; i < ITERATIONS; i++ ) while ( nextTurn != 0 )
        ;
    下一回合 = 1;
}
```

乒乓球

```
对于 ( int i = 0; i < ITERATIONS; i++ ) while ( nextTurn != 1 )
        ;
    下一转 = 0;
}
```

可以看到,这两个线程是使用全局变量nextTurn轮流执行的。 ping 线程将等待直到它
值为零,然后将其设置为一。 pong 线程一直等到变量的值为 1,然后将其设置回零。他们
对固定次数的迭代执行此操作,默认为 500。

事实证明,这对于这些线程轮流来说是一种非常糟糕的方式,尤其是在单处理器上。考虑什么
发生在 ping 内部。一旦将 nextTurn 设置为 1,它就会回到 while 循环,使用 busy waiting 等待直到
pong 将其设置回来。当 ping 在此循环中时,它将继续在 CPU 上运行,并且（在单处理器上）,它将
有效地阻止 pong 运行,直到它用完它的整个量子。然后,一旦 ping 被抢占,pong 就会
有机会做同样的事情。所以,在单处理器上,这个程序的运行时间不依赖于处理器的速度
CPU 但取决于这两个线程消耗 1000 个量子需要多长时间。

越来越难找到用于尝试此练习的单核系统。 remote.eos.ncsu.edu 的机器几年来都是单核的,但现在看来不再是这样了（上次我检查过）。如果
你不确定你是否在单处理器上,你可以通过运行 shell 命令来检查：

```
cat /proc/cpuinfo
```

这将生成一份关于每个 CPU 或 CPU 核心能力的报告;如果它只报告一个核心,那么你就在
单处理器。如果它报告更多,则说明您使用的是多处理器。

在多处理器上,您可以使用 taskset 命令限制要使用的 CPU 内核。它是一种指定硬处理器亲和性的机制。此命令允许您通过提供二进制掩码来指定要使用的 CPU 内核
（实际上是十六进制,但您可以将其视为二进制）。你像这样使用它：

```
任务集 掩码命令运行该命令的参数
```

掩码 0x1 表示仅使用核心编号零（因为掩码仅设置了最低位）。 0x2 的掩码表示
只使用第一个核心（因为掩码只有第一个位）。掩码 0x3 表示同时使用核心 0 和 1（因为
掩码的位号为 0 和 1）。因此,如果您按如下方式运行 pingpong 程序,它将仅在核心 0 上运行：

```
任务集 0x1 ./pingpong
```

我们可以对程序的执行进行计时,以估计特定机器上的量子长度。一旦你的
实施工作正常,请在其中一台远程 EOS Linux 机器上试用。我不知道是否所有这些机器
具有相同的硬件/配置,但是当我尝试使用我的解决方案时,运行大约需要 10 秒
一个核心。使用 time 命令和 taskset 命令为程序的执行计时（在这里,你告诉
是时候运行 taskset 了,你告诉 taskset 只使用第一个 CPU 核心来运行你的 pingpong 程序）：

```
时间任务集 0x1 ./pingpong
```

考虑一下这告诉您这些远程 Linux 机器的量子长度。你能估计
从这个量子长度？ （你不必为此交出任何东西,只要想想你会如何回答。）

如果您在多处理器上运行 pingpong.c,您会发现它运行得更快。多核,一个线程不行
仅通过独占一个 CPU 内核来防止另一个内核运行。尝试计时你的执行
pingpong 程序,但这次让它使用 CPU 内核 0 和 1。这不会告诉您有关系统的任何信息
量子长度,但它会让你看到一个糟糕的同步解决方案有多糟糕。

完成后,上交已完成的 pingpong.c 程序的源代码。提交文件至
名为 EX05 的 Gradescope 作业。
.