操作系统,作业 2

这个作业包括两个编程问题。 -lpthread 标志用于在问题中使用 pthread
2. _XOPEN_SOURCE 的定义用于问题 3 中使用的共享内存调用。

gcc -Wall -g -std=c99 -D_XOPEN_SOURCE=500 -o program program.c -lpthread

另外,请记住您的程序需要注释和一致的缩进,而且您必须
记录您的来源并从头开始自己编写至少一半的代码。

1. (16 pts) 对于这个问题,我们将研究不同调度程序的行为方式以及这如何影响性能。假设下表描述了一组进程的下一个 CPU 突发。

   该表还给出了进程到达就绪队列的时间和进程的优先级值（较低的值代表较高的优先级,就像我们在课堂上所做的那样）。

处理突发长	度到达时间优先级		
P1	8	0	5
P2	5	4	7
P3	3个	9	9
P4		12	2
P5	9 10	15	6
P6	6个	18	8个

   对于以下每个调度算法,绘制一个类似于教科书 CPU 调度章节中的甘特图。此外,对于每个调度,计算平均等待时间和平均周转时间,其中周转时间定义为从进程到达就绪队列到 if 完成其 CPU 突发的时间。

   (a) SJF

   (b) SRTF (c)

   抢占式优先级 (d) 时间片 = 6

   的循环

   对于 SRTF,当前正在运行的进程的剩余突发时间与新到达的进程之间可能存在联系（我们在课堂示例中看到了这一点）。如果发生这种情况,只需让当前正在运行的进程继续运行（因此,在其他条件相同的情况下,避免上下文切换）。

   对于循环法,进入就绪队列的进程与另一个被抢占并返回就绪队列的进程之间可能存在联系（我们在课堂示例中看到了这一点）。

   如果发生这种情况,将被抢占的进程放在队列中进入进程的后面（这样新到达的进程将更早运行）。

   如果您愿意,您可以手工绘制甘特图并扫描它们。只要确保您的绘图足够清晰以便阅读即可。或者,您可以使用您选择的绘图程序绘制它们。无论哪种方式,在 Gradescope 上名为 HW2 Q1 的作业下,将您的图表作为名为 hw2 p1.pdf 的 PDF 文件提交。还请在您的 hw2 p1.pdf 文件中报告等待时间和周转时间。请务必在您的文件中包含标签,以便我们知道您报告的时间。

2. (40 pts) 对于这个问题,你要编写一个多线程程序,maxsum-thread.c,它与作业 1 中的 maxsum.c 问题做同样的工作。和前面的程序一样,你的maxsum-thread.c 将采用命令行参数,给出用于在具有最大总和的整数序列中查找连续非空子序列的工人数。它

还将采用一个可选参数报告,告诉它报告每个工人从自己的工作部分中找到的最大总和（就像在之前的任务中一样）。它将从标准输入中读取输入序列,就像在之前的任务中一样。

这次我们将使用线程作为工作者而不是单独的进程。这将稍微简化我们的实现,因为我们不必使用消息传递来传达工作人员的结果。

线程自动共享内存,所以我们可以让主线程 1获得每个工作线程的结果,只需将它留在传递给每个工作线程的参数结构中的一个字段中。

读取输入序列后,主线程将创建请求数量的工作线程。
工作人员将并行工作以找到最大的总和,每个工作人员只检查一些可能的范围。您可以根据需要划分工作,但您应该将其划分得足够均匀,以便在多核系统上获得多个工作程序的加速。你可能无法获得完美的线性加速,但你应该能够看到更多工作器的一些加速,直到你为每个 CPU 核心都有一个工作器。

主线程可能需要将一个值传递给每个工作人员,因此它会知道它应该处理问题的哪一部分。然后,在启动所有worker后,主线程会等待它们终止,然后比较它们的结果,报告最终的最大总和。为此,它需要从每个工作线程获取一些结果值。您可以处理将值传递给新线程并通过在传递给每个线程的参数结构中添加一个额外字段来获取它们的结果。 ,

完成程序后,您应该能够按如下方式运行它。您可以使用我们在作业 1 中使用的相同输入文件来测试您的解决方案。就像之前的作业一样,您可能会从执行到执行得到不同的命令;没关系。请注意,如果给出报告选项,每个线程将报告其线程 ID (tid) 而不是 pid。

```
$ ./maxsum-thread 5 report < input-3.txt 我是线程 4920。我找到的最
大总和是 145。
我是线程4921,我查到的最大和是153。
我是线程4922。我查到的最大和是159。
我是线程4923,我查到的最大和是170。
我是线程4924,我查到的最大和是152。
最大总和:170
```

请随意在 EOS Linux 机器或您自己的系统上试用。如果您拥有一个有很多内核的系统,您可能会看到很多加速。请务必在最终提交之前至少检查一次 Gradescope 上自动评分器的反馈。尽管您的代码可能会在几乎任何系统上运行,但我们仍将在 Gradescope 上对其进行测试。

完成后,在 Gradescope 上名为 HW2 Q2 的作业下提交源文件 maxsum-thread.c。 _

3. (40 pts) 对于这个问题,你要为家庭作业 1 中的客户端/服务器程序提供类似的功能。这一次,你将不使用服务器和进程间通信来维护游戏状态,而是存储有关游戏板的信息和共享内存段中的最新移动。任何需要访问游戏的程序都可以获取并附加共享内存段。然后它可以像内存中的任何其他数据结构一样查看或修改游戏状态。

您仍然需要两个不同的程序,但它们不会像以前那样做同样的工作。您的其中一个程序将称为 reset.c。它的工作是读取初始游戏板状态,就像1 There is not really a main thread。在这里,我指的是在 main 中开始运行的线程,它不是工作线程。

服务器在之前的分配中创建了一个共享内存段,其中包含有关游戏的任何所需信息。 reset.c 程序会在创建共享内存段并根据板描述文件对其进行初始化后退出。

另一个程序将称为 lightsout.c。它将解释命令行参数中给出的用户命令,并对存储在共享内存中的游戏板进行请求的更改。您还可以使用一个名为 common.h 的头文件,您的两个程序都可以包含它。

您的 lightsout.c 程序的使用方式类似于上次作业中的客户端程序。用户将使用命令行参数运行它,说明在游戏中要做什么。但是,它不必向单独的服务器程序发送请求,而是直接访问和修改存储在共享内存段中的游戏状态。然后,在同一主机上运行的 lightsout.c 程序的其他副本将自动看到游戏的变化。

您应该能够通过以下方式运行 lightsout.c 程序（就像您可以在作业 1 中运行客户端一样）：

- ./lightsout move rc 像这样运
  行lightsout 程序将在r 行、c 列移动。行和列参数以及有效移动的定义就像在之前的分配中一样。如果请求有效,程序将执行请求的移动并打印出“成功”或“错误”,就像客户在任务 1 中所做的那样。

- ./lightsout 撤消
  像这样运行 lightsout 程序将撤消最近的移动。它将打印出“成功”或“错误”,以指示它是否成功（即,如果没有要撤消的动作,则为错误）。 · ./lightsout 报告

  像这样运行 lightsout 程序将打印一个 5×5 的正方形字符,显示电路板的当前状态。

入门文件

课程网站有三个源文件的起始文件。它们大部分是空的,但它们包含一些用于报告错误的函数,并且可能包含您需要的所有内容。

错误条件

reset.c 程序应该期望有一个命令行参数给出游戏板的初始状态,就像服务器在之前的任务中所做的那样。如果命令行参数无效,它应该将以下用法消息打印到标准错误,然后失败退出。

用法:重置 <board-file>

如果给定的板文件丢失或无效,它应该将以下消息打印到标准错误（其中文件名是命令行上给出的板文件的名称)并退出失败。

无效的输入文件:文件名

对于 reset.c 和 lightsout.c 程序,如果在其中一个系统调用中遇到错误（例如无法创建或附加其共享内存）,您应该打印出您选择的有意义的错误消息并然后退出。这些消息还将帮助您在开发程序时诊断问题。

创建共享内存

对于这个问题,您需要将板的所有状态信息和最近的移动组织到一个结构中。我们将此结构称为 GameState。在这个结构中拥有整个表示将使存储在共享内存中变得容易。要创建共享内存,我们只需要请求一个 GameState 实例大小的段。

当你附加共享内存时,你可以将从 shmat() 返回的指针转换为指向 GameState 的指针。然后,您可以通过该指针轻松访问共享内存的内容,就好像共享内存只是 GameState 的一个实例一样。注意,我们使用共享内存的方式很像我们使用 malloc() 的方式;我们告诉 shmget() 我们需要多少内存来存储一个 GameState 结构,然后我们使用 shmat() 返回的指针作为指向 GameState 新实例的指针。

共享内存段将在我们程序的多次执行中持续存在。第一次运行 reset.c 时,它将创建共享内存并根据给定的板文件填充它。此后,运行 lightsout.c 将获取并附加该共享内存段(无需创建新内存段)。它将能够查看由 reset.c 创建的 GameState 结构,并根据用户提供的命令行参数更改棋盘表示或根据需要访问最近的动作。

共享内存段将让多个程序同时访问列表2 。与我们的客户端服务器一样,我们应该能够在同一台主机上打开多个终端并从任何终端访问同一游戏板。

样本执行

一旦您的程序开始运行,您应该能够运行它,如下所示。在这里,我将展示如何从同一主机上运行的多个终端访问共享游戏板。左边的命令来自一个终端会话,右边的命令来自不同的终端。

我已将命令从上到下排序以显示我运行它们的顺序。例如,当我们在右侧终端中移动时,我们可以在左侧终端中使用报告命令查看结果。与命令混合在一起的 shell 注释只是给出了这个示例正在做什么的一些解释(您不需要输入它们)。

```
# 设置初始游戏板。 $ ./reset board-3.txt

# 查看游戏板。 $ ./lightsout 报告
....*
..**。
*.**.
*.**.
*.*..

                                                    # 采取行动
                                                    $ ./lightsout 移动 1 1
                                                    成功

# 在另一个终端查看移动的结果。
$ ./lightsout 报告
.*..*
```

[2] 如果您确实同时运行了两个 lightsout.c 副本,则可能会出现竞争条件。如果他们都试图同时修改共享内存的内容,他们可能会相互干扰并使其处于未知状态。我们将忽略此作业的这个潜在问题。也许我们会在作业 3 上修复它。

**.*.
****。
*.**.
...

再走一步,有一个
查看结果。
$ ./lightsout 移动 0 4
成功

$ ./lightsout 报告
.*.*.
.
****。
*.**.
...

从另一个终端撤消移动并查看结果。
$ ./lightsout 撤消
成功
$ ./lightsout 报告
.*..*
**.*.
****。
*.**.
...

尝试一些无效的命令。 $ ./lightsout 撤消

错误

$ ./lightsout 移动 5 5
错误
$ ./lightsout abc
错误

$ ./reset board-4.txt 无效输入文件：
board-4.txt $ ./reset 用法:reset <board-file> $ ./reset
bad-filename 无效输入文件:bad-filename

命名您的共享内存

您将需要自己的密钥（数字）来创建共享内存段。对于我们的课堂示例，我使用了我刚编写的硬编码密钥 9876，但如果我们中的一些人可能在相同的共享系统上进行开发，我们就不能都使用此密钥。相反，让我们使用 ftok() 函数为类的每个成员分配不同的键。此函数将文件系统中的路径名映射到唯一键。只需使用您的 EOS 主目录的名称作为路径，您应该会得到一个对您来说独一无二的密钥。3

重置共享内存

您的 lightsout.c 程序将在您每次运行时使用相同的共享内存段。如果此内存的内容被损坏（例如，由于您在开发程序时出现错误），您将需要重置此内存的内容。您的 reset.c 程序将使这变得容易。每当您需要擦除旧的 GameState 数据并用新数据覆盖它时，您应该能够重新运行重置。

如果您需要完全删除您的共享内存段以便 reset.c 可以创建一个新的，有几个命令行实用程序可以帮助您。 ipcs shell 命令将列出所有共享内存段（以及一些其他 IPC 对象）。您可以通过运行带有 -m 选项的 ipcrm 来删除共享内存段，后跟要删除的段的 id。 ipcs 报告每个段的 ID 以及段大小。如果您拥有多个共享内存段（可能由您开始运行的其他程序创建），您可以使用大小来帮助您决定删除哪一个。当我尝试这样做时，我的 GameState 的共享内存段比我拥有的其他段小得多。

访问共享内存

与我们在作业 1 中使用的消息队列一样，共享内存仅适用于在同一主机上运行的程序。如果您从两个不同的终端访问共享内存（如示例中所示），您需要确保终端登录在同一台主机上。如果您使用的是 remote.eos.ncsu.edu，请使用 hostname 命令确定您登录的机器并在该特定主机上登录您正在使用的任何其他终端。或者，如果您使用的是 Linux 台式机，则可以轻松地在您面前的主机上打开任意数量的终端。

提交您的作品

完成程序后，提交三个源文件。这应该包括您的通用头文件 common.h、重置程序的实现文件 reset.c 和 lightsout 程序的实现文件 lightsout.c。将所有这些提交到 Gradescope 上名为 HW2 Q3 的作业中。

3这条路走对很重要，否则助教可能无法给你的作业打分。如果您不确定 EOS 上的主目录是什么，可以在 EOS Linux 机器上输入 echo $HOME。此外，对于 ftok()，有时会混淆如何处理 proj id 参数。它只是为了让您根据相同的路径名获得多个唯一键。因为你只需要一个密钥，你可以为这个参数使用任何你想要的常量，但一定要每次都使用相同的常量以获得相同的密钥。