操作系统,作业 1

该作业包括一些编程问题。评分时,我们将在 Gradescope 上编译和测试您的程序。通常,我们将使用如下选项进行编译。有些程序需要额外的选项。这些在需要时与编程问题一起描述。

gcc -Wall -g -std=c99 -o 程序 program.c

如果您在不同的系统上开发程序,您需要确保它在 Gradescope 上正确编译和运行 - 您可以在截止日期前无限次提交代码以查看自动评分器的反馈,以及在截止日期,我们只会对您最后一次提交的内容进行评分。请注意,自动评分器只能帮助您测试规范中给定的测试用例,但您有责任全面测试您的程序以确保它们满足所有要求。当我们对您的工作进行评分时,我们将添加额外的测试用例。此外,请记住,您的程序需要注释并始终如一地缩进,并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 分) 登录其中一台 EOS Linux 机器,使用联机手册页回答以下每个关于 Linux 系统调用的问题。与第一个任务一样,一定要查看联机手册中的系统调用部分。有时会有同名的库函数或 shell 命令。

   将您的答案写入文本文件,将其转换为名为 hw1 p1.pdf 的 pdf 文件,然后将其提交至 Gradescope 上的 HW1 Q1。

   (a) 假设您需要计算出一个文件有多大。您可以通过使用 fopen() 打开文件,然后使用 getchar() 读取 EOF 来完成此操作。如果统计读取的所有字节,则可以确定文件大小。

   计算文件大小的更好方法是使用 stat() 系统调用。描述如何使用 stat 来确定文件的大小。

   (b) 通常,您希望使用阻塞接口来读写文件。但是,操作系统将允许您使用非阻塞接口进行文件 I/O,如果您需要的话。当你打开一个文件时,你可以要求操作系统在文件变得可读或可写时向你的进程发送一个信号。您需要做什么才能以这种方式打开文件。

   (c) 操作系统通常会跟踪其每个进程的资源使用情况和其他统计数据。在 Unix 系统上,进程可以使用 getrusage() 系统调用来检查它。调用 getrusage() 填充结构的字段,报告有关进程执行的各种详细信息。

   使用这个结构,进程如何确定它处于运行状态的次数?

   (d) 如果在进程等待系统调用时传递信号,许多系统调用将立即返回。比如mq receive()就是这样的。如果在调用 mq receive() 期间传递了信号,程序如何判断 mq receive() 返回是因为信号而不是消息到达?

2. (10 分) 简要回答下列问题。将您的答案写入文本文件,将其转换为名为 hw1 p2.pdf 的 pdf 文件,然后将其提交给 Gradescope 上的 HW1 Q2。您不必为这些问题编写代码;用你自己的话回答每个问题。这些应该是您可能在考试中看到的问题类型的良好准备。

   (a) (2.5 分) 两个不同进程之间的上下文切换涉及哪些步骤? (b) (2.5 分)在两个不同线程之间的上下文切换涉及哪些步骤?

   相同的过程?

(c) (5 分) 假设以下两个函数将在两个不同的线程上同时执行（一个线程将调用 p(),另一个线程将调用 q()）。假设 p() 和 q() 中的各个语句 A、B、C 和 D 都将自动执行。例如,p()中A的执行可以发生在q()中C执行之前或之后,但不能同时发生。

```
无效 p() {
    一种;
    B;
}
```

```
无效q（）{
    C;
    D;
}
```

列出这两个函数中语句执行的所有可能交错（在你的答案中每行一个可能的执行）。对于每个可能的交错,给出一个跟踪字符串,显示这些原子语句的执行顺序。例如,字符串“ABCD”给出了一种可能的执行顺序。

3. (40 pts) 对于这个问题,你将编写一个名为 maxsum.c 的程序。它将能够使用多个进程和多个 CPU 内核来更快地解决计算量大的问题。

任务很简单。您将获得一系列正整数和负整数值,如下所示。

2	3	-2	4	1	7

您的工作是在具有最大总和的序列中找到一个连续的非空子序列。
例如,我们可以从上面的序列中得到的最大和是 15,这是通过将索引 0 和索引 5 的值相加得到的。

程序输入

您的程序将从标准输入读取输入。正如您在下面的示例中看到的,我们将使用输入重定向来让它从文件而不是终端读取输入。输入将是最多 1,000,000 个整数值的序列,每个输入行一个值。下面显示了第一个示例输入文件 input-1.txt 的内容。它包含上面显示的相同序列。

```
2个
3个
-2
4个
1 7
```

程序输出

您程序的最后一行输出应如下所示,其中 n 是最大总和。对于上面的示例,这将是 15。

```
最大总和:n
```

命令行参数

您的程序最多需要两个命令行参数。第一个参数给出要使用的工作进程数（见下文）。

您的程序会将单词“report”作为可选的第二个命令行参数。如果给出了 report 选项，那么每个子进程将在完成之前输出它的 pid 和它找到的最大总和。因此，如果在命令行中给出报告标志，程序将报告如下，其中 xx 是子进程的 pid 号，n 是子进程找到的最大总和：

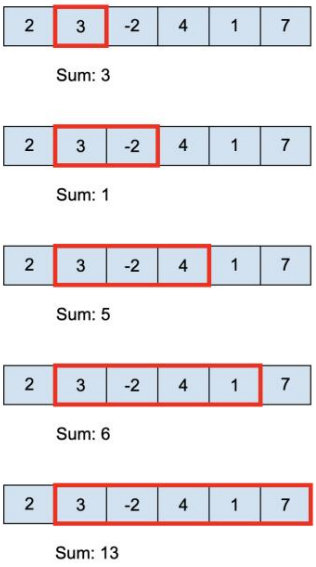
我是进程 xx。我找到的最大总和是 n。

由于您的程序应该报告每个子进程找到的最大总和，因此不同的执行可能会以不同的顺序报告它们。输出顺序将取决于执行时间以及您选择如何实施程序。如果顺序不同也没关系，但“Maximum Sum:”输出应该始终在最后。

部分实施

您会得到一个启动文件来帮助您实施。 starter 包括解析输入序列和命令行参数的代码。输入保证采用上述格式；您不必添加代码来检测和响应错误的输入。此外，您可以保证输入序列中的所有范围加起来都是一个适合带符号的 32 位整数的数字。您不必担心它会溢出。

您将添加代码以查找总和最大的值范围。我们将通过检查每个值范围来做到这一点，从索引 i 开始到某个索引 j 结束，其中 j ⩾ i。您应该能够在 O(n 个范围内的值中执行此操作，您可以组织代码，以便依次检查在同一位置开始的所有范围（或者，或者，在同一位置结束的所有范围）。下图说明了[illegible]



计算从索引 1 开始和结束的范围的总和需要常数时间（它只是值 3）。然后，我们可以计算从索引 1 开始到索引 2 结束的序列的总和

只添加一个值，-2。然后，我们可以通过仅添加一个值 4 来检查从索引 1 到索引 3 的值范围的总和。每次将序列扩展一个元素时，我们只需添加该元素的值。因此，检查特定范围值的总和应该只需要线性时间。

并行化

使用下面的 O(n，) 解决方案，处理大量值序列可能需要很长时间。如您所见)方法
我的解决方案在 100,000 个元素的序列上花费了几秒钟。但是，O(n 很容易并行化，将工作分配给多个进程。

在命令行上，用户指定程序应该创建多少个子进程。我们将每个子进程称为一个工人。读取输入序列后，parent 将 fork() 创建每个 worker，然后让 worker 完成解决问题的所有实际工作。

每个工作人员将负责检查序列中的某些值范围。您可以按自己的意愿分配工作，但要尽量平均分配，这样所有员工的工作量都差不多。例如，您可以让每个工作人员负责根据范围开始的位置（或者范围结束的位置）检查范围。如果你有 3 个工人，你可以让第一个负责检查从索引 0、索引 3、索引 6 等开始的范围。下一个工作人员可能负责从索引 1、索引 4、索引 7 开始的范围。 . .. 最后一个工人可能负责从索引 2、索引 5、索引 8 开始的范围。 . .. 这不是划分工作的唯一方法，但它是一种易于实施的技术。此外，它还允许每个工作人员按照上述方式有效地计算每个范围内的总和。

每个工人都应该计算（并可选地报告）它在问题的一部分中找到的最大总和。完成后，工作人员会将最大的金额发送回其父进程，以便它可以比较并报告最终的最大金额。

如果用户要求您的程序报告，您的工作进程将在完成之前打印出他们发现的最大总和。您不必按任何特定顺序报告它们。

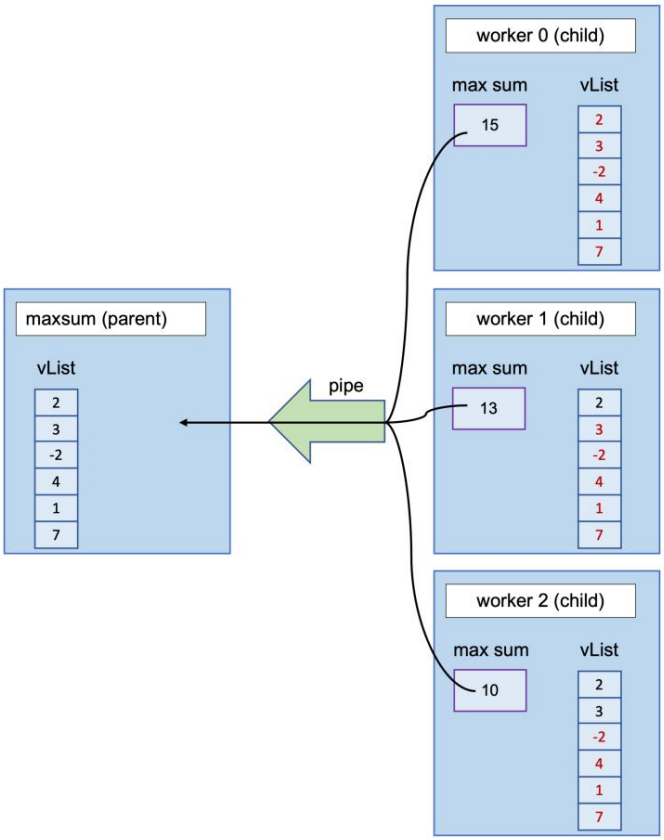
最后，您的父进程将打印出最终的最大总和。这将需要从子进程到父进程的通信（见下文）。一旦父级比较了值并确定了最大和，父级应该打印出最终的最大和。

进程间通信

当创建工人时，他们将自动获得在程序启动时读入的值列表的副本，因为孩子们获得了父母记忆中所有内容的副本。因此，在创建 worker 之后，parent 将等待 worker 在他们的工作部分中找到最大总和。

工作人员完成后，他们需要一种方法将其生成的本地本地最大总和发送回其父进程。我们将为此使用匿名管道。在创建它的孩子之前，父母会制作一个管道。然后，当一个孩子完成后，它将其本地最大的和写入管道的写入端，而父母将能够从另一端读取它。

我们将以二进制形式发送这个值；那会更容易。由于通信只是在同一主机上的进程之间进行，我们不必担心字节顺序或其他架构差异。父母和孩子将始终在同一硬件上运行。要通过管道发送整数的二进制表示，我们只需要使用该整数的起始地址作为要发送的缓冲区。整数的大小是我们要发送的字节数。在接收端，我们可以使用类似的技术将整数的内容读入一个 int 类型的变量中。



父进程读取每个子进程的局部最大和后,进行比较,并在执行结束时打印出最终的最大和。在退出之前,父进程应该使用 wait() 来等待它的每个子进程终止。

文件锁定

两个工作人员尝试同时将本地计算的计数写入管道的风险很小。对于中等数量的工人,我认为这不会真正造成问题,但无论哪种方式,它都很容易预防。在将其结果写入管道之前,工作人员将锁定管道。

这可以通过如下调用来完成。

```
lockf(fd, F_LOCK, 0);
```

然后,在写入它的值之后,一个工作人员可以通过如下调用让其他人使用管道:

```
lockf(fd, F_ULOCK, 0);
```

这是建议文件锁定的示例。如果两个 worker 同时尝试锁定管道,其中一个将被授予锁,另一个将自动等待直到第一个解锁它。只要我们所有的工人在使用之前锁定管道,他们中的两个就不能同时写入。

编译

在 EOS 和 Gradescope Linux 系统上使用 lockf() 调用需要一个额外的预处理器标志。
您需要像下面这样编译您的程序。 -lm 选项与数学库链接,以防您需要使用 sqrt() 函数。

```
gcc -Wall -std=c99 -D_XOPEN_SOURCE=500 -g maxsum.c -o maxsum
```

样本执行

一旦您的程序开始运行,您应该能够按如下方式运行它。我包含了一些 shell 注释来帮助解释这些示例（以井号开头的行）。当然,您不需要输入这些内容,但如果您输入了,shell 应该会忽略它们。

```
# 在最小的测试用例上使用一个工人。
$ ./maxsum 1 report < input-1.txt 我正在处理 125820。我
发现的最大总和是 15。
最大总和:15

# 使用两个工人尝试下一个更大的测试用例。
# 你的输出顺序可能不同,但 \Maximum Sum:  报告 # 应该仍然是最后一个。

$ ./maxsum 2 report < input-2.txt 我正在处理 126659。我
找到的最大总和是 33。
我是进程 126660。我找到的最大总和是 35。
最大总和:35

# 在 input-3.txt（一个 100 元素的序列)上使用四个 worker $ ./maxsum 4 report < input-3.txt 我是进程
127708。我发现的最大总和是 153。

我是进程 127709。我找到的最大总和是 159。
我是进程 127710。我找到的最大总和是 170。
我是进程 127711。我找到的最大总和是 152。
最大总和:170

# 查看仅用一个工人处理输入 5 需要多长时间,一个 100,000 - # 元素序列。在这里,我们没有使用报告选项,因此它不会 # 减慢执行
速度。你可能会得到与#我不同的计时结果,这取决于你的实现和你运行#你的解决方案的系统。



$ time ./maxsum 1 < input-5.txt 最大总和:227655

真实的  0m14.165s
用户  0m14.157s
系统  0m0.001s

# 用 4 个工人再试一次。这应该花费大约相同数量的 # 总 CPU 时间,但是,如果你有多个内核,它应该更快地完成。拥有多个 CPU 内核,
worker 应该能够运行在
```

并行,减少从开始到结束的时间。阅读 # 时间命令的联机手册页,以确保您理解 # 时间输出的含义。

$ time ./maxsum 4 < input-5.txt 最大总和:
227655

真实的	0m3.620s
用户	0m14.261s
系统	0m0.003s

一定要在最大的输入文件上使用 time 命令,使用不同数量的 worker 来尝试你的程序。只要您拥有至少与工作人员一样多的内核,您应该会看到工作人员数量接近线性的加速。所以,如果你使用两个工人,花费的时间应该是一个工人的一半;如果您使用三个工人,则大约需要三分之一的时间。会有一些开销,并且可能会有一些不均匀的工作分配,所以不要期望完美的线性加速。

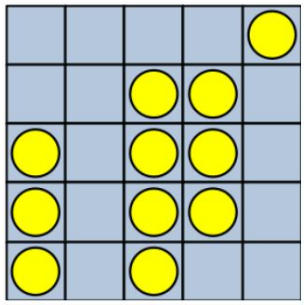
然而,如果你根本没有得到太多加速,那就表明你做错了什么,或者你可以在工人之间更平均地分配工作。我们将在测试您的解决方案时查看您的执行时间。要测试性能,您可能需要使用自己的 Linux 机器或 EOS Linux 实验室机器之一;您通过 remote.eos.ncsu.edu 访问的系统不适用于此测试(即,不要为此使用它们)。看起来它们各只有两个内核,而且由于它们是共享系统,您将与其他用户竞争 CPU 时间。

提交

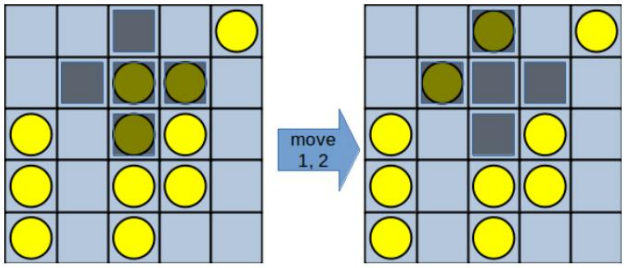
完成后,在名为 HW1 Q3 的作业下提交源文件 maxsum.c on Grade scope。

4. (40 pts) 对于这个问题,您将使用进程间通信 (IPC) 创建一对协同工作的程序 client.c 和 server.c。我已经为您编写了服务器的一部分。

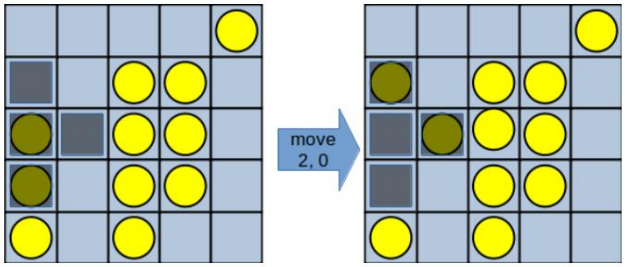
客户端和服务器协同工作以实现 1990 年代熄灯游戏的多人版本。该游戏使用如下图所示的棋盘。游戏使用 5×5 的灯格,每个灯都可以关闭或打开。



玩家通过选择网格上的一个单元格来移动。移动会切换所选单元格及其周围四个相邻单元格(上方、下方、左侧和右侧)的灯光状态。切换灯会将其从打开更改为关闭或从关闭更改为打开。例如,移动选择第 1 行第 2 列的单元格将切换五个灯的状态,如下所示。

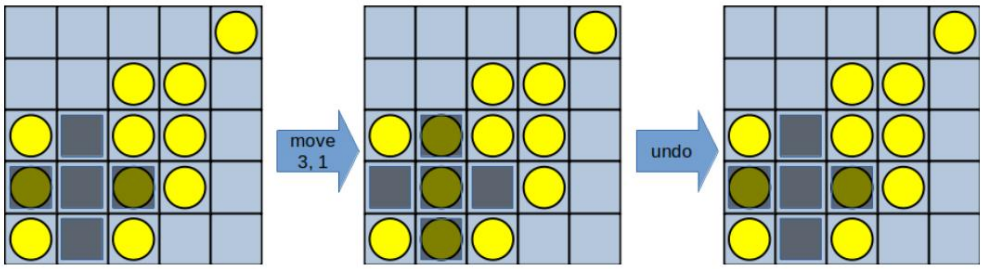


棋盘边缘的单元格没有四个邻居。玩家仍然可以在这些单元格上移动;他们只是只切换实际退出板上的单元格。例如,选择第 2 行第 0 列的单元格只会改变三个相邻的单元格;右边的单元格,上面一个和下面一个。所选单元格的左侧没有单元格。



您的游戏将支持一次撤消操作。下棋后,玩家可以选择撤销下棋,棋盘自动恢复到下棋前的状态。撤销历史只是一步长,所以在撤销后立即尝试撤销操作是无效的。

同样,在进行任何其他移动之前执行撤消是无效的。



熄灯游戏的维基百科页面位于https://en.wikipedia.org/wiki/Lights_Out_ （游戏）。

服务器操作

您的服务器将负责存储游戏板的状态并跟上最新的移动以支持撤消操作。客户端将向服务器发送命令以进行移动、撤消最近的移动或检查棋盘的当前状态。

服务器成功启动后,它将无限期地继续运行,响应客户端请求,直到用户使用 ctrl-C 终止它。

服务器需要一个命令行参数。当您启动它时,您将给它一个包含电路板初始状态的文件的名称。板描述文件将包含五行文本,每行有 5 个字符。这定义了灯的初始 5 × 5 状态。每个字符可以是句点“.”,表示灯已关闭,或星号,

* 指示灯亮着。

....*
..**。
*.**.
*.**.
...

上面的样本输入文件是 board-3.txt 随启动器一起提供的。正如预期的那样,它有五行,每行五个字符。它有 11 个亮着的灯和 14 个熄灭的灯。

如果用户使用错误的命令行参数运行服务器（例如,参数太多或缺少板文件名）,它应该失败退出并打印以下使用消息到标准

错误。

用法:服务器 <board-file>

如果给定的板文件无效（如果文件不存在,如果它不是这种格式或者它包含 * 和换行符)服务器应该不成功终止并打印除 . 以外的字符,如下使用标准错误消息,其中文件名是命令行上给出的板文件的名称:

无效的输入文件:文件名

在您的服务器中,您将需要提出自己的表示形式来将板的状态存储在服务器中。对于未来的任务,您将希望将关于棋盘状态的所有内容（灯光和最近的移动)存储在一个结构中,但您不需要为这个任务。

运行客户端

每次运行客户端时,您都用命令行参数指示它应该执行什么命令。它将命令发送到服务器,然后打印出服务器的响应。客户端可以通过以下方式运行:

- ./client move rc
  像这样运行客户端请求服务器在第 r 行,第 c 列移动。行从 0（顶部)到 4（底部)编号,列从 0（左侧)到 4（右侧)编号。如果缺少 r 或 c 参数或者它们不是 0 到 4 之间的整数,则该命令无效。客户端将向服务器发送一条消息,指示请求的移动,服务器应向客户端发送一个响应,指示移动成功或失败要求。在终止之前,如果命令成功,客户端将打印出“success”,如果失败则打印出“error”。

- ./client 撤消
  此命令请求服务器撤消最近的移动。服务器应该保留单步撤消历史记录,因此如果尚未进行任何移动,或者最近的一次移动已经撤消,则此命令无效。服务器将发回指示命令成功的响应,客户端将打印出“成功”或“错误”,就像移动命令一样。

- ./客户报告
  当用户输入此命令时,客户端将请求板当前状态的报告。它会将其打印为 5 × 5 的正方形“.”。和 * 字符,就像电路板输入文件的内容一样。客户端不需要为此命令打印出“成功”和“错误”;电路板的打印输出将显示命令成功。

错误条件

如果用户向客户端提供的命令行参数不是上述命令之一,则客户端应将“错误”打印到标准输出。每当客户端退出并显示“错误”消息时,它应该退出不成功。

您可以检测客户端中的一些使用错误,而无需向服务器发送请求。如果您想为这些报告“错误”消息而无需联系服务器,也可以。对于其他人,您将需要联系服务器并收到某种指示错误的响应。

如果您无法打开消息队列或尝试接收消息时出现问题,我们不需要特定的错误消息。如果发生这些错误(这可能有助于调试),您将想要报告,但任何合理的错误消息都可以。

过去,我看到很多学生在调用 mq receive() 时遇到问题。请务必查看相关文档,并检查此系统调用的返回值,如果失败则检查 errno 的值。这可能会为您提供有关问题所在的一些信息。

服务器终止

服务器将继续运行,直到您终止它,接受来自任何发送请求的客户端的请求。服务器将使用我们在课堂上看到的 sigaction() 调用来为 SIGINT 注册一个信号处理程序。当用户用 ctrl-C 杀死服务器时,服务器会捕获信号,打印出一个换行符,然后打印出板子最终状态的报告,格式与 report 命令相同。如果服务器如下所示启动,然后立即用 ctrl-C 终止,这就是您希望在终端中看到的内容。在电路板之前打印换行符可以使格式看起来更好。它可以防止电路板的第一行打印在与按 ctrl-C 时显示的“^C”相同的行上。

```
$ ./server board-3.txt
^C
....*
..**。
*.**.
*.**.
*.*..
```

客户端/服务器通信

客户端和服务器将使用 POSIX 消息队列进行通信。每次客户端运行时,它都会向服务器发送一条消息,然后等待响应。服务器的响应可能包含命令的结果(例如,用于报告)或者它可能只需要指示成功或失败(例如,用于移动或撤消)。

我们有一对来自课堂的示例程序,演示了如何创建和使用消息队列。消息队列是单向通信通道,允许我们在同一主机上的进程之间发送消息(任意字节序列)。每个消息队列都需要一个唯一的名称。

两个进程可以通过使用 mq open() 创建一个新消息队列或打开一个具有商定名称的现有消息队列来进行通信。然后,一个进程可以使用 mq send() 将消息放入队列,而另一个进程可以使用 mq receive() 按照发送消息的顺序从队列中获取消息的副本。当程序使用消息队列完成时,它可以使用 mq close() 关闭它。

消息队列将继续存在(甚至可能有一些排队的消息),直到它被 mq unlink() 删除。

消息队列让我们发送任意字节序列。您可以决定如何将客户端请求和服务器响应编码为消息。例如,您可以将它们编码为以空字符结尾的字符串,并用空格分隔消息中的不同值。对消息使用文本格式可能有助于调试,但如果您愿意,也可以使用二进制格式。如何对消息进行编码取决于您。

我们每个人都需要两个消息队列,一个向服务器发送请求,另一个向客户端发送响应。在单个主机上,每个不同的消息队列都需要一个唯一的名称。为了避免学生之间的名称冲突,我们将只在每个消息队列名称中包含我们的统一 ID。

将名称“/unityID-server-queue”用于用于向您的服务器发送消息的消息队列。
使用“/unityID-client-queue”作为将响应发送回客户端的队列名称。

部分实施

启动时,服务器会创建这两个消息队列,打开一个用于读取,另一个用于写入。然后,服务器反复从服务器队列中读取消息并通过客户端队列发回响应。 Moodle 课程站点有部分服务器实现。它已经创建了所需的消息队列并在完成后取消链接它们。您将需要添加代码来存储和初始化存储在服务器中的游戏板和相关状态。您还需要代码来读取、处理和响应消息,并处理按下 ctrl-C 时发送的 SIGINT,在程序退出之前报告电路板的最终状态。

除了部分服务器实现之外,您还会得到一个 common.h 头文件,其中定义了客户端和服务器都需要的一些常量。您需要更新此文件以在消息队列名称中使用您的统一 ID。我们不为客户提供部分实施。您可以自己编写该部分。客户更容易;我的还不到服务器大小的一半。请记住,服务器负责创建这两个消息队列。客户只需要打开并使用它们;确保在客户端启动时不会意外删除或覆盖消息队列。

汇编

要使用 POSIX 消息队列,您需要将您的程序与 rt 库链接。此外,要使用 sigaction(),您需要定义预处理器符号 POSIX SOURCE。您应该能够使用如下命令编译您的客户端和服务器:

```
gcc -D_POSIX_SOURCE -Wall -g -std=c99 -o server server.c -lrt

gcc -Wall -g -std=c99 -o client client.c -lrt
```

执行

一旦您的代码开始工作,您应该能够让服务器在一个终端窗口中运行,并通过在不同的窗口中运行客户端来反复连接到它。

通过 remote.eos.ncsu.edu 或 remote-linux.eos.ncsu.edu 登录会将您带到多个不同主机的池中,每次连接时不一定是同一台特定机器。当心。如果您使用 putty 或其他 ssh 客户端登录该地址两次,您实际上可能无法将两个终端窗口连接到同一主机。消息队列只允许您在同一主机上的进程之间进行通信,因此如果您不小心登录到两台不同的机器上,它们将无法工作。

您可以在每个终端中键入主机名以查看它在哪个主机上运行。为确保您在同一主机上获得两次登录,您可以登录一次,使用主机名确定您所在的主机,然后使用第二个直接登录到同一主机(而不是 remote.eos.ncsu.edu)远程登录窗口。

样本执行

一旦您的客户端和服务器正常工作，您应该能够按如下方式运行它们。此处，左侧列显示在一个终端中运行的服务器，右侧列显示在不同终端窗口中重复运行的客户端。我已经放置了一些垂直空间来显示这些命令的执行顺序，并且我已经包含了一些注释来解释一些步骤。我们启动左边的服务器，运行几次客户端，然后杀死服务器。当服务器终止时，您可以看到 ctrl-C 的打印输出（打印为 ^C），然后是服务器报告板的最终状态。

从 board-3.txt 开始运行服务器 $ ./server board-3.txt

通过客户端查看棋盘

$ ./客户报告

....*

..**。

*.**.

*.**.

...

走一步看看结果

$ ./客户移动 0 0

成功

$ ./客户报告

**..*

*.**.

*.**.

*.**.

...

在左下角再走一步 $ ./client move 4 0

成功

$ ./客户报告

**..*

*.**.

*.**.

..**。

.**..

撤消最后一步。

$ ./client 撤消

成功

撤消历史只是一步深度。 $ ./client 撤消

错误

第一次撤消后的结果。

$ ./客户报告**..*

*.**.
*.**.
*.**.
...

尝试来自客户端的一些无效命令。 $ ./client 移动 abc xyz

错误

$ ./客户移动 5 5

错误

$ ./client 错误命令

错误

使用 ctrl-C 关闭服务器并查看面板的最终状态
^C
**..*
*.**.
*.**.
*.**.
...

为服务器尝试一些无效的参数。

错误的板文件。
$ ./server board-4.txt 无效的输入文件：
board-4.txt

参数太多 $ ./server abc usage:
server <board-file>

提交

完成后，在 Gradescope 上名为 HW1 Q4 的作业下提交 client.c、修改后的 server.c 和修改后的 common.h 的源代码。从前几年教这门课，我发现学生有时会忘记提交他们的头文件，所以一定要提交你的 common.h 文件。否则，我们将无法编译您的代码。