第三十章-条件变量

预备知识

通过本作业，您可以探索一些使用锁和条件变量的实际代码，以实现本章中讨论的各种形式的生产者/消费者队列。
您需要查看这些代码，以各种配置运行它，并使用它们来了解哪些方案有效，哪些无效，以及一些其他的问题。 阅读 README 文件了解详细信息。

不同版本的代码对应于不同的“解决”生产者/消费者问题的方式。大多数解决方式是不正确的；只有一个解决方案是正确的。
阅读本章以了解有关生产者/消费者问题以及代相关代码的更多信息。

第一步是下载代码，输入 make 来构建所有方案的实现。您应该会看到四个文件：

- main-one-cv-while.c: 通过单个条件变量解决生产者/消费者问题。
- main-two-cvs-if.c: 使用两个条件变量，并使用 if 检查是否需要睡眠。
- main-two-cvs-while.c: 使用两个条件变量，并使用 while 检查是否需要睡眠（这是正确的解决方案）
- main-two-cvs-while-extra-unlock.c: 首先释放锁并在 fill 条件变量周围获取锁

查看 pc-header.h 很有用，它包含所有这些不同主程序的公共代码，以便使用 Makefile 正确地构建代码。

每个程序可以使用以下标志:

```
-l 每个生产者生产的数量
-m 生产者/消费者共享的缓冲区大小
-p 生产者数量
-c 消费者数量
-P
-C
-v [verbose flag: 追踪发生了什么并打印]
-t [timing flag: 打印执行总时间]
```

前四个参数意思:

-l 指定每个生产者应该进行多少次循环(即每个生产者生产的数据量)

-m 控制共享缓冲区的大小(大于或等于 1),

-p -c 分别设置有多少生产者和消费者。

更有趣的是两个睡眠字符（sleep string），一个用于生产者，另一个用于消费者。
这些标志使您可以使线程在执行过程中的某些点处于休眠状态，从而切换到其他线程。
这样一来，您就可以很方便的使用每种解决方案，甚至可以研究特定问题或研究生产者/消费者问题的其他方面。

字符串（string）参数的指定方式如下。例如，如果有三个生产者，睡眠字符串应该分别为每个生产者指定睡眠时间，
并使用冒号作为分隔符。这三个生产者的睡眠字符参数看起来像这样：

```
sleep_string_for_p0:sleep_string_for_p1:sleep_string_for_p2
```

每个睡眠字符串依次是一个逗号分隔的列表，用于决定代码中每个睡眠点的睡眠时间。

Problem1

问题描述

我们的第一个问题集中在 main-two-cvs-while.c（有效的解决方案）上。
首先，研究代码。 你认为你了解当你运行程序时会发生什么吗？

问题解答

```
./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v
 NF              P0 C0
  0 [*---  --- ] p0
  0 [*---  --- ]    c0
  0 [*---  --- ] p1
  1 [u  0 f--- ] p4
  1 [u  0 f--- ] p5
  1 [u  0 f--- ] p6
  1 [u  0 f--- ]    c1
  1 [u  0 f--- ] p0
  0 [ --- *--- ]    c4
  0 [ --- *--- ]    c5
  0 [ --- *--- ]    c6
  0 [ --- *--- ] p1
  0 [ --- *--- ]    c0
  1 [f--- u  1 ] p4
  1 [f--- u  1 ] p5
  1 [f--- u  1 ] p6
  1 [f--- u  1 ]    c1
  1 [f--- u  1 ] p0
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
  0 [*---  --- ]    c6
  0 [*---  --- ] p1
  0 [*---  --- ]    c0
  1 [u  2 f--- ] p4
  1 [u  2 f--- ] p5
  1 [u  2 f--- ] p6
  1 [u  2 f--- ]    c1
  0 [ --- *--- ]    c4
  0 [ --- *--- ]    c5
  0 [ --- *--- ]    c6
  1 [f--- uEOS ] [main: added end-of-stream marker]
  1 [f--- uEOS ]    c0
  1 [f--- uEOS ]    c1
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
  0 [*---  --- ]    c6

Consumer consumption:
  C0 -> 3
```

Problem2

问题描述

指定一个生产者和一个消费者运行，并让生产者产生一些元素。
缓冲区大小从 1 开始，然后增加。随着缓冲区大小增加，程序运行结果如何改变?
当使用不同的缓冲区大小(例如 -m 10)，生产者生产不同的产品数量(例如 -l 100),
修改消费者的睡眠字符串(例如 -C 0,0,0,0,0,0,1)，full_num 的值如何变化?

问题解答

```
./main-two-cvs-while -l 3 -m 1 -p 1 -c 1 -v
./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 3 -m 3 -p 1 -c 1 -v
./main-two-cvs-while -l 3 -m 4 -p 1 -c 1 -v

./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 6 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 12 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 24 -m 2 -p 1 -c 1 -v

./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 0,0,0,0,0,0,1
 NF              P0 C0
  0 [*---  --- ] p0
  0 [*---  --- ]    c0
  0 [*---  --- ] p1
  1 [u  0 f--- ] p4
  1 [u  0 f--- ] p5
  1 [u  0 f--- ] p6
  1 [u  0 f--- ]    c1
  1 [u  0 f--- ] p0
  0 [ --- *--- ]    c4
  0 [ --- *--- ]    c5
  0 [ --- *--- ]    c6
  0 [ --- *--- ] p1
  1 [f--- u  1 ] p4
  1 [f--- u  1 ] p5
  1 [f--- u  1 ] p6
  1 [f--- u  1 ] p0
  1 [f--- u  1 ] p1
  2 [   2 *  1 ] p4
  2 [   2 *  1 ] p5
  2 [   2 *  1 ] p6
  2 [   2 *  1 ]    c0
  2 [   2 *  1 ]    c1
  1 [u  2 f--- ]    c4
  1 [u  2 f--- ]    c5
  1 [u  2 f--- ]    c6
  2 [*  2  EOS ] [main: added end-of-stream marker]
  2 [*  2  EOS ]    c0
  2 [*  2  EOS ]    c1
  1 [f--- uEOS ]    c4
  1 [f--- uEOS ]    c5
  1 [f--- uEOS ]    c6
  1 [f--- uEOS ]    c0
  1 [f--- uEOS ]    c1
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
```

```
  0 [*---  --- ]    c6

Consumer consumption:
  C0 -> 3

./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 1,0,2,0,0,0,1
NF             P0 C0
  0 [*---  --- ] p0
  0 [*---  --- ]    c0
  0 [*---  --- ] p1
  1 [u  0 f--- ] p4
  1 [u  0 f--- ] p5
  1 [u  0 f--- ] p6
  1 [u  0 f--- ] p0
  1 [u  0 f--- ] p1
  2 [*  0    1 ] p4
  2 [*  0    1 ] p5
  2 [*  0    1 ] p6
  2 [*  0    1 ] p0
  2 [*  0    1 ] p1
  2 [*  0    1 ] p2
  2 [*  0    1 ]    c1
  1 [f--- u  1 ]    c4
  1 [f--- u  1 ]    c5
  1 [f--- u  1 ]    c6
  1 [f--- u  1 ] p3
  2 [   2 *  1 ] p4
  2 [   2 *  1 ] p5
  2 [   2 *  1 ] p6
  2 [   2 *  1 ]    c0
  2 [   2 *  1 ]    c1
  1 [u  2 f--- ]    c4
  1 [u  2 f--- ]    c5
  1 [u  2 f--- ]    c6
  2 [*  2  EOS ] [main: added end-of-stream marker]
  2 [*  2  EOS ]    c0
  2 [*  2  EOS ]    c1
  1 [f--- uEOS ]    c4
  1 [f--- uEOS ]    c5
  1 [f--- uEOS ]    c6
  1 [f--- uEOS ]    c0
  1 [f--- uEOS ]    c1
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
  0 [*---  --- ]    c6

Consumer consumption:
  C0 -> 3

./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 0,1,0,0,0,0,1
 NF              P0 C0
  0 [*---  --- ] p0
  0 [*---  --- ]    c0
  0 [*---  --- ] p1
  1 [u  0 f--- ] p4
  1 [u  0 f--- ] p5
  1 [u  0 f--- ] p6
  1 [u  0 f--- ]    c1
```

```
  1 [u  0 f--- ] p0
  0 [ --- *--- ]    c4
  0 [ --- *--- ]    c5
  0 [ --- *--- ]    c6
  0 [ --- *--- ] p1
  1 [f--- u  1 ] p4
  1 [f--- u  1 ] p5
  1 [f--- u  1 ] p6
  1 [f--- u  1 ] p0
  1 [f--- u  1 ] p1
  2 [   2 *  1 ] p4
  2 [   2 *  1 ] p5
  2 [   2 *  1 ] p6
  2 [   2 *  1 ]    c0
  2 [   2 *  1 ]    c1
  1 [u  2 f--- ]    c4
  1 [u  2 f--- ]    c5
  1 [u  2 f--- ]    c6
  2 [*  2  EOS ] [main: added end-of-stream marker]
  2 [*  2  EOS ]    c0
  2 [*  2  EOS ]    c1
  1 [f--- uEOS ]    c4
  1 [f--- uEOS ]    c5
  1 [f--- uEOS ]    c6
  1 [f--- uEOS ]    c0
  1 [f--- uEOS ]    c1
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
  0 [*---  --- ]    c6

Consumer consumption:
  C0 -> 3
```

Problem4

问题描述

我们来看一些 timings。 对于一个生产者，三个消费者，大小为 1 的共享缓冲区以及每个消费者在 c3 点暂停一秒，您认为需要执行多长时间?

（./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,1,0,0,0:0,0,0,1,0,0,0:0,0,0,1,0,0,0 -l 10 -v -t）

问题解答

```
./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,1,0,0,0:0,0,0,1,0,0,0:0,0,0,1,0,0,0
-l 10 -v -t
 NF        P0 C0 C1 C2
  0 [*--- ] p0
  0 [*--- ]    c0
  0 [*--- ]       c0
  0 [*--- ] p1
  1 [*  0 ] p4
  1 [*  0 ] p5
  1 [*  0 ] p6
  1 [*  0 ]          c0
  1 [*  0 ]    c1
```

```
1 [*  0 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]       c1
0 [*--- ]    c0
0 [*--- ]       c2
0 [*--- ]          c1
0 [*--- ]          c2
0 [*--- ] p1
1 [*  1 ] p4
1 [*  1 ] p5
1 [*  1 ] p6
1 [*  1 ]    c1
1 [*  1 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]       c3
0 [*--- ]    c0
0 [*--- ]       c2
0 [*--- ] p1
1 [*  2 ] p4
1 [*  2 ] p5
1 [*  2 ] p6
1 [*  2 ]    c1
1 [*  2 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]          c3
0 [*--- ]    c0
0 [*--- ]          c2
0 [*--- ] p1
1 [*  3 ] p4
1 [*  3 ] p5
1 [*  3 ] p6
1 [*  3 ]    c1
1 [*  3 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]       c3
0 [*--- ]    c0
0 [*--- ]       c2
0 [*--- ] p1
1 [*  4 ] p4
1 [*  4 ] p5
1 [*  4 ] p6
1 [*  4 ] p0
1 [*  4 ]    c1
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]          c3
0 [*--- ]    c0
0 [*--- ]          c2
0 [*--- ] p1
```

```
1 [*  5 ] p4
1 [*  5 ] p5
1 [*  5 ]       c3
1 [*  5 ] p6
1 [*  5 ] p0
0 [*--- ]       c4
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ]    c1
0 [*--- ]       c0
0 [*--- ]    c2
0 [*--- ] p1
1 [*  6 ] p4
1 [*  6 ] p5
1 [*  6 ] p6
1 [*  6 ]       c1
1 [*  6 ] p0
0 [*--- ]       c4
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ]          c3
0 [*--- ]       c0
0 [*--- ]          c2
0 [*--- ] p1
1 [*  7 ] p4
1 [*  7 ] p5
1 [*  7 ] p6
1 [*  7 ] p0
1 [*  7 ]       c1
0 [*--- ]       c4
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ]    c3
0 [*--- ]       c0
0 [*--- ]    c2
0 [*--- ] p1
1 [*  8 ] p4
1 [*  8 ] p5
1 [*  8 ] p6
1 [*  8 ]       c1
0 [*--- ]       c4
0 [*--- ] p0
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ]          c3
0 [*--- ]       c0
0 [*--- ]          c2
0 [*--- ] p1
1 [*  9 ] p4
1 [*  9 ] p5
1 [*  9 ] p6
1 [*  9 ]       c1
0 [*--- ]       c4
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ]    c3
0 [*--- ]       c0
0 [*--- ]    c2
```

```
  1 [*EOS ] [main: added end-of-stream marker]
  1 [*EOS ]          c3
  0 [*--- ]          c4
  0 [*--- ]          c5
  0 [*--- ]          c6
  0 [*--- ]       c1
  0 [*--- ]       c2
  1 [*EOS ] [main: added end-of-stream marker]
  1 [*EOS ]    c3
  0 [*--- ]    c4
  0 [*--- ]    c5
  0 [*--- ]    c6
  1 [*EOS ] [main: added end-of-stream marker]
  1 [*EOS ]       c3
  0 [*--- ]       c4
  0 [*--- ]       c5
  0 [*--- ]       c6

Consumer consumption:
  C0 -> 5
  C1 -> 5
  C2 -> 0

Total time: 12.01 seconds
```

```
指定 3 个消费者，1 个生产者，缓冲区大小为 1，
消费者睡眠点：c3、c3、c3，睡眠时间为 1 秒
每个生产者循环 10 次

如果消费者线程先执行，那么睡眠时间为 13s，
如果生产者者线程先执行，那么睡眠时间为 12s

实际结果：
Total time: 12.01 seconds
```

Problem8

问题描述

现在让我们看一下 main-one-cv-while.c。您是否可以假设只有一个生产者，一个消费者和一个大小为 1 的缓冲区，配置一个睡眠字符串，让代码运行出现问题。

问题解答

```
一个消费者一个生产者不会出现问题，见书P257
```

Problem9

问题描述

现在将消费者数量更改为两个。 为生产者消费者配置睡眠字符串，从而使代码运行出现问题。

问题解答

即使不配置睡眠字符串，也可能出现如下情况：
生产者生产后，缓冲区满了，唤醒了两个正在睡眠的消费者中的一个，然后进入睡眠（Mutex_lock）
消费者消费后，唤醒另一个消费者，进入睡眠（Mutex_lock），
新的消费者线程被唤醒，发现缓冲区为空，进入睡眠（Cond_wait）,此时三个线程都进入睡眠

无法配置睡眠字符串，使得代码运行必定出现问题，因为这取决于操作系统的线程调度。

Problem10

问题描述

现在查看 main-two-cvs-if.c。 您是否可以配置一些参数让代码运行出现问题？
再次考虑只有一个消费者的情况，然后再考虑有一个以上消费者的情况。

问题解答

一个消费者一个生产者不会出现问题。
出现问题的情况：
一个生产者，两个消费者
生产者生产完成时，消费者 c1 还没有进入临界区，消费者 c2 在 Cond_wait 处等待，
生产者唤醒一个消费者，c1 抢先执行，执行完后缓冲区为空，c2 开始执行，发现缓冲区为空，do_get 执行发生错误！

./main-two-cvs-if -m 1 -c 2 -p 1 -l 10 -C 2:0,0,0,3 -P 1 会出现错误

```
./main-two-cvs-if -m 1 -c 2 -p 1 -l 10 -C 2:0,0,0,3 -P 1
error: tried to get an empty buffer
```

Problem11

问题描述

最后查看 main-cvs-while-extra-unlock.c。在向缓冲区添加或取出元素时释放锁时会出现什么问题？
给定睡眠字符串来引起这类问题的发生？ 会造成什么不好的结果？

问题解答

由于do_get 和 do_fill 在锁外面，所以锁起不到任何作用。