线程也达到了这一点。 当最后一个线程到达障碍点时，所有线程都被释放，并可以恢复并发执行。

假设屏障被初始化为N-必须在屏障点等待的线程数：

```
(N) ;
```

然后每个线程执行一些工作，直到它到达障碍点：

```
/*做一些工作一段时间*/障

碍点() ;

/*做了一段时间的工作*/
```

使用本章中描述的POSIX或Java同步工具，构造实现以下API的屏障：

- intinit(intn)-将屏障初始化为指定大小。
- 势垒点（空隙）-标识势垒点。 所有一切 当最后一个线程到达这一点时，线程将从屏障中释放出来。

每个函数的返回值用于识别错误条件。 每个函数在正常运行时返回0 如果发生错误，-1。 在源代码下载中提供了一个测试工具来测试障碍的实现。

方案拟订项目

项目1-设计一个线程池

第4.5.1节引入了线程池。 当使用线程池时，任务被提交到池中，并由来自池的线程执行。 工作使用队列提交到池，可用线程从队列中移除工作。 如果没有可用的线程，工作将保持排队直到一个可用。 如果没有工作，线程将等待通知，直到任务可用为止。

该项目涉及创建和管理线程池，它可以使用Pthreds和POSIX同步或Java完成。 下面我们提供与每项具体技术相关的细节。

I. 员额

本项目的POS IX版本将涉及使用P线程API创建许多线程，以及使用POS IX互斥锁和信号量进行同步。

客户

线程池的用户将使用以下API：

- 空池init()-初始化线程池。
- IN T池提交(void（*某些函数）(void*p)，void*p)- 其中，some函数是指向将由来自池的线程执行的函数的指针，p是传递给函数的参数。
- 空池关闭(void)-一旦所有任务都结束，就关闭线程池 已经完成了。

我们在源代码下载中提供了一个示例程序client.c，它说明了如何使用这些函数的线程池。

线程池的实现

在源代码下载中，我们提供C源文件threadpool.c作为线程池的部分实现。 您需要实现客户端用户调用的函数，以及支持线程池内部的多个附加函数。 执行工作将涉及以下活动：

1. 池init()函数将在启动时创建线程，并初始化互斥锁和信号量。
2. 池提交()函数部分实现，目前将要执行的函数以及它的数据放置在任务结构中。 任务结构表示将由池中的线程完成的工作。 池提交()将通过调用enqueue()函数将这些任务添加到队列中，工作线程将调用dequeue()从队列中检索工作。 队列可以静态实现（使用数组）或动态实现(使用链表）。

   池init()函数有一个int返回值，用于指示任务是否成功提交到池中（0表示成功，1表示失败）。 如果队列使用数组实现，如果尝试提交工作并且队列已满，则池init()将返回1。 如果队列作为链表实现，则池init()应该始终返回0，除非发生内存分配错误。
3. 工人()函数由池中的每个线程执行，其中每个线程将等待可用的工作。一旦工作可用，线程将从队列中删除它，并调用执行()来运行指定的函数。

   当工作提交到线程池时，可以使用信号量通知等待线程。 可以使用命名信号量或未命名信号量。 有关使用POS IX信号量的更多细节，请参阅第7.3.2节。

4. 在访问或修改队列时，必须使用互斥锁来避免竞赛条件。(第7.3.1节提供了关于P线程互斥锁的详细信息)
5. 池关机()函数将取消每个工作线程，然后通过调用p线程连接()等待每个线程终止。有关POS IX线程取消的详细信息，请参阅第4.6.3节。（信号量操作信号等待()是一个取消点，允许在信号量上等待的线程被取消。）

有关此项目的详细信息，请参阅源代码下载。特别是，自述文件描述了源文件和头文件，以及用于构建项目的Makefile。

II. Java

本项目的Java版本可以使用Java同步工具完成，如7.4节所述。同步可能取决于
(a) 使用同步/等待（)/通知()的监视器(第7.4.1节）或(B)信号量和可重入锁（第7.4.2节和第7.4.3节）。Java线程在4.4.3节中描述。

线程池的实现

您的线程池将实现以下API：

- 线程池()-创建一个默认大小的线程池。
- 线程池(int大小)-创建大小为大小的线程池。
- 无效添加（可运行任务）-添加一个要由线程在 泳池。
- 无效关闭()-停止池中的所有线程。

我们在源代码下载中提供Java源文件Thread Pool.java作为线程池的部分输入。您将需要输入客户端用户调用的方法，以及支持线程池内部的几个附加方法。执行工作将涉及以下活动：

1. 构造函数将首先创建一些等待工作的空闲线程。
2. 工作将通过添加()方法提交到池中，该方法添加了实现Runnable接口的任务。添加()方法将Runnable任务放入队列(您可以使用JavaAPI中的可用结构，如java.util.List)。
3. 一旦池中的线程可用于工作，它将检查队列中的任何可运行任务。如果有这样的任务，空闲线程将从队列中删除任务并调用其运行()方法。如果队列为空，空闲线程将等待工作时通知

变得可用。（添加()方法可以使用通知()或信号量操作来实现通知，当它将可运行的任务放置到队列中以唤醒等待工作的空闲线程时。）

4. 关闭（)方法将通过调用它们的中断()方法来停止池中的所有线程。当然，这需要线程池执行的可运行任务检查它们的中断状态（第4.6.3节）。

有关此项目的详细信息，请参阅源代码下载。特别是，自述文件描述了Java源文件，以及关于Java线程中断的进一步详细信息。

项目2-睡眠教学助理

大学计算机科学系有一名助教(TA)，在正常的办公时间帮助本科生完成编程任务。助教的办公室很小，只有一张桌子，有一张椅子和电脑。办公室外面的走廊里有三把椅子，如果助教目前正在帮助另一个学生，学生们可以坐着等待。当没有学生在办公时间需要帮助时，助教坐在桌子旁打盹。如果一个学生在办公时间到达，发现助教在睡觉，学生必须唤醒助教寻求帮助。如果一个学生到达并发现助教目前正在帮助另一个学生，学生坐在走廊的椅子上等待。如果没有椅子，学生稍后会回来。

使用POSIX线程、互斥锁和信号量，实现协调TA和学生活动的解决方案。这项任务的详细情况如下。

学生和助教

使用P线程（4.4.1节），首先创建n个学生，每个学生将作为一个单独的线程运行。助教也将作为一个单独的线程运行。学生线程将交替在编程一段时间和寻求帮助的TA。如果助教有空，他们会得到帮助。否则，他们要么坐在走廊的椅子上，要么在没有椅子的情况下，恢复编程，稍后再寻求帮助。如果学生到达并注意到助教正在睡觉，学生必须使用信号量通知助教。当助教完成帮助学生时，助教必须检查是否有学生在走廊等待帮助。如果是这样的话，助教必须依次帮助这些学生。如果没有学生在场，助教可能会回到午睡。

也许模拟学生编程的最佳选择-以及为学生提供帮助的助教-是让适当的线程在随机的一段时间内睡眠。

第7.3节提供了POS IX互斥锁和信号量的覆盖范围。

详情请参阅该部分。

项目3----食堂问题

在第7.1.3节中，我们提供了一个解决使用监视器的餐用直升机问题的方法的大纲。 该项目涉及使用POS IX互斥锁和条件变量或Java条件变量来实现这个问题的解决方案。 解决方案将基于图7.7所示的算法。

这两种实现都需要创建五个哲学家，每个哲学家由数字0组成。 。 4. 每个哲学家将作为一个单独的线程运行。 哲学家在思考和饮食之间交替。 为了模拟这两个活动，让每个线程在一到三秒之间随机睡眠。

I. 员额

使用P线程创建的线程将在4.4.1节中讨论。当一个哲学家想吃东西时，她引用了这个功能

```
皮卡叉(INT哲学家号码) _
```

在那里，哲学家号码标识了希望吃饭的哲学家的数量。 当一个哲学家吃完饭后，她援引

```
退货叉(IN T哲学家编号)_
```

您的实现将需要使用POSIX条件变量，这些变量在7.3节中涵盖。

II. Java

当一个哲学家想吃东西时，她引用了一种方法-叉子（哲学家编号），其中哲学家编号标识了希望吃东西的哲学家的数量。 当一个哲学家吃完饭后，她调用返回叉子（哲学家编号）。

您的解决方案将实现以下接口：

```
公共接口餐厅
{
  /*一个哲学家在想吃东西的时候打电话给*/公共空
   虚采取叉子(int哲学家编号)；

   /*被一个哲学家召唤，当它吃完*/公共空虚返回叉子
  (int哲学家号码)；
}
```

它将需要使用Java条件变量，这些变量在第7.4.4节中涵盖。

项目4-生产者-消费者问题

在7.1.1节中，我们提出了一个基于信号量的解决方案，使用有界缓冲区解决生产者-消费者问题。 在本项目中，您将使用图5.9和5.10所示的生产者和消费者流程设计有界缓冲区问题的编程解决方案。 第7.1.1节中提出的解决方案使用三个信号量：空和满，它们计数缓冲区中的空和满槽数，互斥信号量，这是一个二进制（或互斥）信号量，它保护缓冲区中项目的实际插入或删除。 对于这个项目，您将使用标准计数信号量表示空和满，并使用互斥锁，而不是二进制信号量来表示互斥。 生产者和消费者-作为单独的线程运行
将项目从与空、满和互斥结构同步的缓冲区移动。 您可以使用P线程或WindowsAPI来解决这个问题。

自助餐

在内部，缓冲区将由一个固定大小的类型缓冲区项数组(将使用typedef定义)组成。 缓冲区项对象的数组将被操作为循环队列。 缓冲区项的定义以及缓冲区的大小可以存储在头文件中，例如：

```
/*buffer.h*/
typedefin t缓冲区项；
#defined Buffer_SIZE5
```

缓冲区将使用两个函数进行操作，插入项（)和删除项()，这两个函数分别由生产者和消费者线程调用。 概述这些功能的骨架如图7.14所示。
插入项（)和删除项()函数将使用图7.1和图中概述的算法同步生产者和消费者
7.2. 缓冲区还需要一个初始化函数，初始化互斥对象互斥以及空的和全的信号量。
主()函数将初始化缓冲区并创建单独的pro-Ducer和Consumer线程.. 一旦它创建了生产者和消费者线程，主()函数将休眠一段时间，并在唤醒后终止应用程序。 主要()功能将通过命令行的三个参数：

1. 在结束之前睡多久
2. 生产者线程的数量
3. 消费线程的数量

此函数的骨架如图7.15所示。

```
#包括 “缓冲.h”

/*缓冲区*/
缓冲区项缓冲区[Buffer SIZE] ;

插入项 (缓冲项) _          {
   /*将项目插入缓冲区
     如果成功, 则返回0, 否则
     返回-1表示错误条件*/
}

删除项目(缓冲区项目*项目){
   /*从缓冲区中删除对象, 将其
     放入项中
     如果成功, 则返回0, 否则
     返回-1表示错误条件*/
}
```

图7.14缓冲区操作大纲。

生产者和消费者线程

生产者线程将在睡眠一段随机时间和插入随机整数到缓冲区之间交替。随机数将使用RAND()函数产生，该函数产生0到RANDMAX之间的随机整数。消费者也会在一段随机的时间内睡觉，在觉醒时，会试图从缓冲区中删除一个项目。生产者和消费者线程的大纲如图7.16所示。

```
#包括 “缓冲.h”

主(IN Targc, char*argv[])      {
  /*1. 获取命令行参数argv[1], argv[2], argv[3]*/
  /*2. 初始化缓冲区*/
 /*3. 创建生产者线程*/
  /*4. 创建消费线程*/
  /*5. 睡眠*/
  /*6. */出口
}
```

图7.15骨架程序大纲。

```
#包括<stdlib.h>/*为rand()*/#包括 “buffer.h”

无效*生产者(无效*Param)缓冲
  项；

    (真实)  {
     /*睡眠时间随机*/睡眠 (.)  ;
     /*生成随机数*/项=rand()；
     如果 (插入项目(项目))
       fprintf (“报告错误条
       件”)  ;
     其他的
         “生产者”              %d\n “, 项目)；
}

void*consumer(void*param)缓冲
  项目；

    (真实)  {
     /*睡眠时间随机*/睡眠 (.)  ;
     如果 (删除项目(和项目))
       fprintf (“报告错误条件”)  ;
     其他的
       印刷( “消费消费          %d\n “, 项目)；
}
```

图7.16 生产者和消费者线程的概要。

如前所述，您可以使用P线程或WindowsAPI来解决这个问题。 在下面的章节中，我们提供了更多关于这些选择的信息。

线程创建和同步

使用P线程API创建线程将在4.4.1节中讨论。 第7.3节提供了使用P线程的互斥锁和信号量的覆盖范围。 有关P线程线程创建和同步的具体说明，请参阅这些部分。

Windows线程

第4.4.2节讨论使用WindowsAPI创建线程。 有关创建线程的具体说明，请参阅该部分。

视窗静音锁

互斥锁是Dispatcher对象的一种类型，如7.2.1节所述.. 下面说明如何使用CreateMutex()函数创建互斥锁：

```
#包括<windows.h>

曼德尔·穆特克斯；
互斥=创建互斥(NULL、false、NULL)；
```

第一个参数是指互斥锁的安全属性。通过将此属性设置为NULL，我们防止进程的任何子代创建此互斥锁以继承锁的句柄。第二个参数指示互斥锁的创建者是否是锁的初始所有者。传递false值表示创建互斥体的线程不是初始所有者。（我们很快就会看到互斥锁是如何获得的。）第三个参数允许我们命名互斥。然而，由于我们提供了NULL的值，所以我们不命名互斥。如果成功，Create Mutex()将HANDLE返回到互斥锁；否则，它将返回NULL。

在7.2.1节中，我们将调度员对象识别为信号或无信号。具有信号的Dispatcher对象（例如互斥锁）可用于所有权。一旦它被获取，它就会移动到非信号状态。当它被释放时，它返回信号。

互斥锁是通过调用等待单对象()函数来获取的。该函数与指示等待多长时间的标志一起传递给锁。下面的代码演示了如何获取上面创建的互斥锁：

```
等待单个对象(Mutex，IN FINITE)；
```

参数值IN FINITE表示我们将等待无限多的时间来使锁可用。可以使用其他值，如果锁在指定时间内不可用，则允许调用线程超时。如果锁处于信号状态，等待单个对象()立即返回，并且锁变得未信号。通过调用ReleaseMutex ()释放锁(移动到信号状态）-例如：

```
释放互斥（互斥）；
```

Windows信号量

WindowsAPI中的信号量是Dispatcher对象，因此使用与互斥锁相同的信令机制。信号创建如下：

```
#包括<windows.h>

汉德尔·塞姆；
信号=创建信号量（空，1，5，空）；
```

第一个和最后一个参数标识信号量的安全属性和名称，类似于我们描述的互斥锁。 第二和第三个参数表示信号量的初始值和最大值。 在此实例中，信号量的初始值为1，其最大值为5。 如果成功，创建信号量()将HAND LE返回到互斥锁；否则，它将返回NULL。

信号量与互斥锁具有相同的等待单对象()功能。 我们通过使用以下语句获取在此示例中创建的信号量Sem：

等待单个对象(Sem，IN FINITE)；

如果信号量的值>为0，则信号量处于信号状态，因此由调用线程获取。 否则，调用线程将无限期地阻塞-就像我们正在指定IN FINITE一样-直到信号量返回到信号状态。

对于Windows信号量，信号()操作的等效方法是
释放信号量()功能.. 该功能通过了三个参数：

1. 信号灯的手
2. 增加信号量的多少
3. 指向信号量前一个值的指针我们可以使用以下语句

将Sem增加1：

释放信号符(Sem，1，NULL)；

如果成功，释放信号（)和释放互斥()返回一个非零值，否则返回0。