最后的任务是试验网络。

我们将为您提供一些底层网络通信设施；

您将在这些基础之上构建更好的抽象，然后在构建分布式聊天程序时使用该抽象。

网络中的每个节点都将实现为 Nachos 的单独实例。

因此，您将为每个网络节点运行一个 JVM。

这些 JVM 将在同一台实际的机器上运行，即使我们假装它们是分布在网络上也是如此。

每个 Nachos“节点”都有到网络的单个连接，该连接使用 UDP 套接字实现。

nachos.machine.NetworkLink 类为您提供此功能。

每个节点都有一个唯一的链接地址，代表该节点的物理网络地址。

您可以使用 Machine.networkLink（）方法访问网络链接。

在 NetworkLink 对象上调用 getLinkAddress（）将返回该节点的链接地址。

地址的类型只是一个整数。

由于多个 Nachos 节点将在同一台实际计算机上运行，　　因此您将需要为每个节点使用不同的交换文件以避免干扰。

一种简单的方法是将网络链接地址附加到交换文件名。

您可以通过构建 proj4 目录并在同一台计算机上的两个不同窗口中同时运行 nachos 来测试基本的网络功能。

在 Nachos 初始化时，文本的第一行在其中包含其网络链接的地址。

例如，如果您看到：

nachos 5.0j 初始化中...配置中断计时器处理器控制台网络（1）用户检查分级机

这表示网络链接地址为 1（上面输出中网络后面的括号中的值）。

用于此分配的新计算机文件包括：

•machine / NetworkLink.java-物理网络硬件的仿真。

网络接口与控制台的网络接口相似，不同之处在于传输单元是数据包而不是字符。

网络在节点之间提供了有限大小的数据包的有序，不可靠的传输。

此类提供 send（）和 receive（）消息以在网络上发送和接收数据包。

由于此类模拟网络设备驱动程序，因此您必须提供机制来保护对网络链接的访问并确保正确使用它。

当接收到数据包时和发送数据包后，网络链接会产生中断；

您将使用这些中断在底层网络链接接口上实现高层通信机制。

•machine / Packet.java-网络传输单元。

数据包的最大大小由常量 machine.Packet.maxPacketLength 给出，恰好是 32 个字节。

每个数据包包含一个 4 字节的标头（有关详细信息，请参见 Packet.java）和一个 28 字节的有效负载。

使用常量 machine.Packet.maxContentsLength 来引用代码中有效负载的大小（即，不要硬编码 28 个字节的值）。

在网络目录中可以找到用于此分配的新内核文件，其中包括：

•network / NetKernel.java-支持联网的内核；

扩展 VMKernel。

•network / NetProcess.java-支持网络系统调用的进程；

扩展 VMProcess。

•network / PostOffice.java-以“邮局”为模型的简单消息抽象，在网络链接的顶部实现。

这提供了与特定端口（邮箱）之间的消息的同步传递和接收。

每台计算机有多个邮箱。

与原始网络链接相比，PostOffice 提供了更方便的通信抽象。

它在网络链接的节点到节点通信之上提供用户到用户的通信，但是不提供可靠的消息传递。

在此项目中，您将必须在 PostOffice 和网络链接抽象之上实现其他层。

例如，您将需要在基础不可靠的通信链接之上实施可靠的传递。

由于 PostOffice 并不完全适合该项目，因此可以随意使用它作为起点，但是如果愿意，也欢迎您从头开始。

请注意，NetKernel 扩展了 VMKernel，而 NetProcess 扩展了 VMProcess，这意味着此项目取决于您的第 3 阶段解决方案能否正常工作。

如果愿意，可以修改类以扩展 UserKernel 和 UserProcess，以使您的代码仅取决于项目阶段 2。为使其正常工作，请确保物理内存页的总数足以容纳整个内存页。

测试程序的代码。

由线控制

Processor.numPhysPages = 16

在 nachos.conf 文件中。

I.（75％）实现两个网络系统调用（在 syscall.h 中记录的 connect（）和 accept（）），它们在连接端点之间提供可靠的，面向连接的字节流通信。

连接端点是链接地址和端口号的组合。

用户程序通过使用带有远程网络链接 ID（也称为主机 ID）和端口号的 connect（）来连接到远程节点。

远程节点必须调用 accept（）系统调用以接受连接；

系统调用使用可以建立连接的本地端口号。

建立连接后，每个系统调用都会返回一个表示连接的新文件描述符（在 UNIX 术语中，此文件描述符称为“套接字”）。

然后，应用程序可以通过在套接字上调用 write（）将消息发送到网络，并可以通过在套接字上调用 read（）从网络读取消息。

连接的一端通过在套接字文件描述符上执行 close（）操作而关闭。

关闭连接后，套接字（在连接的任一端！）上的任何 read（）或 write（）调用都应返回-1，表示错误。

这里有一个重要的细节要考虑。

如果程序执行以下操作：

写（socket，buf，100）；

关闭（插座）；

然后，尽管连接已关闭，但远程主机可能尚未确认 100 字节的数据。

您的实现必须确保所有写入套接字的数据都传递到远程主机，即使在确认所有传出数据之前关闭套接字也是如此。

这也意味着，即使关闭了套接字，远程主机对套接字的任何读取调用也必须成功，直到已读取所有未决数据为止。

这意味着您不能在套接字关闭时简单地“删除”该套接字的状态。

您必须保持套接字为“活动”状态，直到所有数据传递完毕！

注意，上面给出的 close（）调用不应阻塞等待所有传出数据被传输；

close（）应该立即返回，并且网络层应处理剩余数据的传输。

套接字上的 read（）和 write（）实现必须提供无消息大小限制的全双工，可靠的字节流通信。

这意味着数据包可以同时在连接的两个方向上流动。

同样，如果一个节点使用大小为 N 的缓冲区调用 write（），则远程主机将通过调用 read（）来接收完全相同的数据（尽管可能需要多次调用 read（））。

read（）调用可能不会一次返回所有数据。

例如，如果节点 A 调用

写（套接字，缓冲区，100）；

/ *发件人* /

和节点 B 呼叫

读（套接字，缓冲区，10）；

/ *接收者* /

那么 B 将不得不在接收所有数据之前调用 read（）十次。

另外，read（）返回的数据可能少于用户请求的数据。

例如，如果 B 打电话

读取（套接字，缓冲区 1000）；

/ *接收者* /

那么如果只有 100 个字节可用，则 read（）将仅返回这 100 个字节。

通常，每次读取调用返回的数据量不必等于远程主机上的相应 write（）调用。

另外，read（）不会阻止等待数据在网络上接收；

它立即返回（返回值为 0）。

您可以自由地将 PostOffice 类用作实现的起点，也可以从网络链接的顶部开始。

最好不要修改 PostOffice 本身，而应该实现自己的通信接口，该接口使用 PostOffice（如果需要）。

您的实现应使用此处记录的协议。

此处提供了此协议的状态转换表。

在设计实现时，请考虑基础网络层中所有可能的故障情况。

基础网络链路可能会丢弃任意数据包，但不会重新排序数据包或破坏其内容。

（提示：这将使您的工作容易得多！）

为了测试您的实现，可以通过修改 nachos.conf 中的 NetworkLink.reliability 密钥的值来使 Nachos 网络链接随机丢弃数据包。

这是一个介于 0 和 1 之间的浮点数，表示数据包将被成功传递的可能性。

•您的实施应继续以高达 10%的下降率（即 0.9 的可靠性）良好运行。

所谓“运行良好”，是指如果丢失率为 10%或更低，则交付给应用程序的实际数据速率不应灾难性地降低。

•有效端口号在 0 到 127 之间（含 0 和 127）。

如果需要，您可以支持更大范围的端口，但是不应允许使用负端口号。

端口在 syscall.h 中指定为 C 类型 int。

•请注意，给定节点上可能有多个程序接受同一端口上的连接。

例如，程序 A 可能会执行以下操作：

接受（100）；

并接受来自远程节点的连接，然后程序 B（与程序 A 在同一节点上）可以执行以下操作：

接受（100）；

接受来自远程节点的另一个连接。

然后，A 和 B 都在端口 100 上接收数据。您的实现必须处理这种情况。

•您需要为使用 connect（）建立的连接分配一个本地端口号-此端口号不必与远程节点上的端口相同。

传出连接（使用 connect（）建立）的本地端口号可能与另一个现有传入连接（使用 accept（）建立）的本地端口号相同。

（如果您不明白为什么会这样，则需要更仔细地考虑协议规范！）

•syscall.h 包含一个小错字。

read（）系统调用应返回读取的字节数；如果没有可用字节，则返回 0；如果有错误或连接已关闭，则返回-1。

如果套接字已被远程主机关闭，则 read（）应该返回以前接收的任何数据，并且在用户读取了所有接收到的数据后返回-1。

•您应使用固定窗口大小为 16 个数据包的滑动窗口协议。

这意味着每个连接都维护一个“信用计数”，这是在收到确认之前可以发送的数据包的数量。

（通过“数据包”，这里我们指的是具有最大长度的基础网络链接数据包，如上所述）。

在发送数据包时，节点首先等待信用计数变为非零，然后递减信用

计数。

来自远程主机的确认会增加信用计数。

由于连接是全双工的，因此对于每个通信方向都需要一个单独的滑动窗口。

同样，每个连接都应保持自己的滑动窗口。

这意味着，如果到同一端口有多个连接，则每个连接（在每个方向上）一次最多可以发送 16 个未确认的数据包。

•连接上的每个单向数据流在发送方需要一个不同的发送缓冲区，在接收方需要一个不同的接收缓冲区。

每个发送缓冲区可以任意增大，但是每个接收缓冲区可以容纳不超过 16 个数据包。

•您不应在每个套接字上使用一个（或多个）线程来实现协议。

相反，您可以使用所有套接字共享的一小部分线程-例如，一个线程用于发送数据，另一个线程用于接收数据，另一个线程实现超时。

•如果数据包到达接收方，并且其用于连接的接收缓冲区已满，则接收方应丢弃该数据包而不进行确认。

•实现字节流通信。

这意味着在一次调用中发送 2 个字节在功能上等同于在两次单独的调用中发送相同的 2 个字节（每个字节一个调用）。

•读/写系统调用不应等待来自远程主机的数据包。

如果端口上没有可读取的数据，则读取将立即返回。

write 将端口上的数据排队，然后返回而无需等待确认。

（但是，连接将阻塞，直到可以建立连接为止。）

二。

（25％）通过使用消息传递原语在多个（至少 3 个）用户之间实现类似于 IRC 的“网络聊天”应用程序，来证明您的消息传递原语有效。

网络聊天类似于 UNIX 对话实用程序，除了它可以用于 N 向对话（不只是 2 向对话）；每个用户坐在不同的计算机上，任何人键入的任何内容都会广播到连接到聊天室的其他所有人。

您应该编写两个程序：一个聊天服务器（称为 chatserver.c）和一个聊天客户端（称为 chat.c）。

该服务器在一个节点上运行，并接受来自聊天客户端的连接。

聊天客户端连接到聊天服务器，并接受来自控制台的用户输入。

用户键入一行文本后，该行将被传输到聊天服务器。

然后，服务器将消息广播到与其连接的所有客户端。

然后，聊天客户端收到的每条消息都应显示在控制台上。

用户可以动态连接聊天室和从聊天室断开连接。

•聊天客户端程序 chat.c 应该带有一个命令行参数：运行聊天服务器的计算机的网络链接地址。

•您的聊天服务器必须使用端口 15；

所有聊天客户端都连接到服务器上的同一端口。

•聊天客户端可以随时连接到服务器，也可以随时断开连接。

您的服务器必须在对话过程中的任何时候处理客户端进入和离开聊天室的问题。

•所有用户应对发送和接收的邮件具有一致的看法。

这意味着每个客户端上的每个单独消息必须相同。

消息以不同的顺序出现在不同的客户端上是可以接受的，只要来自特定客户端的消息序列以其键入的顺序出现即可。

但是，来自不同客户端的消息交织可能会有所不同。

•聊天客户端必须仅显示在其他客户端上键入的完整消息。

它不应显示部分消息（即少于一行文本）。

另外，聊天客户端必须避免将来自控制台的用户输入与从聊天服务器接收到的消息进行交错。

一种简单的方法是让聊天客户端在用户键入一行时不显示来自聊天服务器的数据。

•即使未从控制台读取任何输入，聊天客户端也必须继续显示其他聊天室用户键入的消息。

这意味着在显示其他客户端的输出之前，您可能不会阻止等待从控制台读取数据的等待。

（为使显示保持清晰，如果聊天客户端在用户开始键入该行之后阻止其等待该行被读取，则可以。）

•聊天客户端必须在行首收到一个句点（“。”）的用户输入后正常退出。

收到标准输入的任何数据后，聊天服务器应退出。