Socket编程（套接字编程）

socket编程是编写通过网络进行通信的程序的标准方法。 虽然最初是为用 C 语言编程的 Unix 计算机开发的，但socket抽象是通用的，不依赖于任何特定的操作系统或编程语言。 这使得程序员可以在许多上下文中使用 socket 编写正确的网络程序。

这部分作业是基本 socket 编程入门，将编写两对 TCP 客户端和服务器程序，用于通过 Internet 发送和接收文本消息。 一对客户端/服务器必须用 C 语言编写。另一对可以用 Python 或 Go 编写。 您可以选择其中一种语言（只能选择一种）完成。 Python 解决方案更短，但您也需要了解 Go 的 socket 编程才有助于进行后续作业。

两种语言的客户端和服务器程序应满足以下规范。 请务必在编程之前和之后仔细阅读这些内容，以确保您的实现满足这些要求：

服务器要求

- 每个服务器程序都应该监听socket，等待客户端连接，从客户端接收消息，将消息打印到标准输出，然后无限期地等待下一个客户端。
- 每台服务器都应采用一个命令行参数：监听客户端连接的端口号。
- 每个服务器应该在**无限循环**中接受和处理客户端通信，从而允许多个客户端向同一服务器发送消息。 服务器应该只在响应外部信号 (例如按 `ctrl-c` 的 SIGINT) 时退出。
- 每个服务器应维护一个短的 (5-10个) 客户端队列，并依次处理多个客户端连接尝试。 在实际应用中，TCP服务器将派生一个新进程来并发地处理每个客户端连接，但这对于此分配来说不是必需的。
- 每个服务器都应该妥善处理 socket 编程库函数可能返回的错误值（请参阅下面每种语言的具体细节）。 与处理客户端连接相关的错误不应导致服务器在处理错误后退出。

客户端要求

- 每个客户端程序都应该联系服务器，从标准输入读取消息，发送消息，然后退出。
- 每个客户端都应该准确地读取和发送标准输入中的消息，，直到到达 EOF（文件结尾）。
- 每个客户端应该接受两个命令行参数：服务器的 IP 地址和服务器的端口号。
- 每个客户端必须能够通过迭代读取和发送消息块来处理任意大的消息，而不是首先将整个消息读入内存。
- 每个客户端都应该处理部分发送(当套接字仅传输上次发送调用中给出的部分数据时)，尝试重新发送剩余的数据，直到所有数据都发送完毕。
- 每个客户端都应该妥善处理 socket 编程库函数可能返回的错误值。

入门

在 Vagrant VM 上进行所有构建和测试。 您可以在 Vagrant VM（预装了 Emacs 和 Vim 文本编辑器）上编写代码，也可以直接在操作系统上编写代码（允许您使用已安装的任何编辑器）。 从终端运行 `vagrant ssh` 后，运行 `cd /vagrant` 以进入课程目录。

我们在 `assignment1/client_server/` 目录中提供了框架代码。 在开始编程之前，您应该阅读并理解此代码。

您应该仅在所提供的标有 `TODO` 注释的文件的位置进行编程。 每个客户端有一个 `TODO` 部分，每个服务器也有一个 `TODO` 部分。 可以添加函数，但不要更改文件名，因为它们将用于自动化测试。

以下部分提供了每种语言的客户端和服务器程序的详细信息。

C

经典的《Beej 网络编程指南》位于：https://beej.us/guide/bgnet/html/。 系统调用部分和客户端/服务器示例部分将是最相关的。 手册页对于查找单个函数也很有用（例如 `man socket`）。

文件 client-c.c 和 server-c.c 包含框架代码。 您需要在标记为 `TODO` 的位置添加socket编程和 I/O 代码。 参考解决方案在每个文件的 TODO 部分中大约有 70 行（注释和间隔都较好）代码。 您的实现可能更短或更长。

对于错误处理，您可以调用 `perror` 来设置全局变量 `errno` 的socket编程函数（Beej 的指南将告诉您执行哪些操作）。 对于那些不这样做的人，只需将消息打印到标准错误即可。

您应该通过在 `assignment1/client_server` 目录中运行 `make` 来构建您的解决方案。 您的代码必须使用提供的 Makefile 构建。 服务器应作为 `./server-c [port] > [output file]` 运行。 客户端应作为 `./client-c [server IP] [server port] < [message file]` 运行。 有关详细信息，请参阅“测试”。

Python

Python socket 编程的文档位于：https://docs.python.org/2/library/socket.html。 顶部的前几段、有关套接字对象的部分和第一个示例特别相关。

文件 `client-python.py` 和 `server-python.py` 包含框架代码。 您需要在标记为 `TODO` 的位置添加 socket 编程代码。 参考解决方案在每个文件的 `TODO` 部分中大约有 15 行（注释和间隔都较好）代码。 您的实现可能更短或更长。

Python socket 函数将自动引发异常并提供有用的错误消息。 不需要额外的错误处理。

服务器应作为 `python server-python.py [port] > [output file]` 运行。 客户端应作为 `python client-python.py [server IP] [server port] < [message file]` 运行。 有关详细信息，请参阅“测试”。

Go

Go socket 编程的文档位于：https://golang.org/pkg/net/。 顶部的概述和 Conn 类型的部分最为相关。

文件 `client-go.go` 和 `server-go.go` 包含框架代码。 您需要在标记为 `TODO` 的位置添加 socket 编程代码。 参考解决方案在每个文件的 `TODO` 部分中有大约 40 行（注释和间隔都较好）代码。 您的实现可能更短或更长。

Go `Listen` 函数默认维护一个连接客户端的队列。 无需额外编程。

您应该通过在 `assignment1/client_server` 目录中运行 `make go` 来构建您的解决方案。 您的代码必须使用提供的 Makefile 构建。 服务器应作为 `./server-go [port] > [output file]` 运行。 客户端应作为 `./client-go [server IP] [server port] < [message file]` 运行。 有关详细信息，请参阅“测试”。

测试

您应该通过尝试将消息从客户端发送到服务器来测试您的实现。 服务器可以在后台运行（在命令后附加 `&`）或在单独的 SSH 窗口中运行。 您应该使用 `127.0.0.1` 作为服务器 IP，并使用 10000 到 60000 之间的高服务器端口号。您可以使用命令 `fg` 杀死后台服务器，将其带到前台，然后按 `ctrl-c`。

Bash 脚本 `test_client_server.sh` 将通过尝试在客户端和服务器的所有 4 种组合（C 客户端到 C 服务器、C 客户端到 Python/Go 服务器等）之间发送多个不同的消息来测试您的实现。 消息如下：

1. 短消息 “Go Tigers！\n”
2. 随机生成的长的字母数字消息
3. 随机生成的长二进制消息
4. 从不同的客户端顺序发送到一个服务器的几条短消息
5. 从不同的客户端并发地发送到一个服务器的几个长的、随机的字母数字消息

运行脚本为

`./test_client_server.sh [python|go] [server port]`

如果出现权限错误，请运行 `chmod 744 test_client_server.sh` 以授予脚本执行权限。

对于每个客户端/服务器对，如果消息发送和接收正确，测试脚本将打印“SUCCESS”。

确保在运行 `test_client_server.sh` 之前构建 C 语言客户端/服务器对。

调试提示

以下是一些调试技巧

- 框架代码中定义了缓冲区大小和队列长度常量，可以使用它们。 如果它们没有在特定文件中定义，则不需要用到。 如果您没有使用它们中的任何一个，要么您硬编码了一个值（这是不好的风格），要么您很可能做错了什么。
- 在 C、Python 和 Go 中，有多种方法可以从 `stdin/stdout` 读取和写入。 任何方法都是可以接受的，只要它不会一次将无限量的数据读取到内存中，并且不会修改消息。
- 如果您使用缓冲 I/O 写入标准输出，请确保调用 `flush`，否则长消息的末尾可能无法写入。
- 请记住在客户端程序结束时关闭socket。
- 测试时，请确保使用 `127.0.0.1` 作为客户端的服务器 IP 参数，并确保客户端和服务器程序使用相同的服务器端口。
- 如果您收到 “address already in use” 错误，请确保没有服务器正在运行。 否则，请使用命令 `logout` 和 `vagrant ssh` 重新启动 ssh 会话。
- 如果您遇到其他连接错误，请尝试使用 10000 到 60000 之间的其他端口。

可能出现的问题：

- **Q：我应该在 recv() 上设置 MSG_WAITALL 标志吗?**

  A：不应该。这会导致 recv() 在接收到指定数量的数据之前不会返回。 然而，服务器无法提前知道这个数量，所以你应该继续调用recv()，直到没有任何东西可以接收。

- **Q：当用户按 ctrl-c 时，我是否需要处理 SIGINT 等信号来清理服务器进程?**

  A：不需要，在这个作业中没有必要。 对信号的默认响应已经足够好了。 但请记住，一般来说这样做是一个很好的做法。

- **Q：我应该使用流套接字还是数据报套接字?**

A：请使用流套接字，以确保传递准确的消息。

- **Q：客户端是否应该等待接收服务器的回复?**

  A：不，在这个任务中，它应该在发送所有数据后立即退出。

- **Q：服务器是否应该同时（在单独的进程中）处理客户端连接?**

  A：不，正如客户端要求中所述，本次作业不需要这样做。 所以不需要使用fork()!