Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2018-2

15 de noviembre de 2018
Evaluada

Actividad 13

Networking

Introducción

El malvado Mafioso Enzini no te da descanso: esta vez ha logrado contaminar tu computador para que no puedas acceder a tus archivos. Sin embargo, sólo uno de ellos ha sobrevivido a este terrible ataque. Un desconocido programador te ha ayudado dándote acceso a parte del código de un antiguo servidor. Debes utilizar tus conocimientos de *networking* para poder conectarte y manejar tus archivos.

Instrucciones

Esta actividad tiene como propósito establecer una conexión entre un **servidor** y un **cliente** mediante el uso de *sockets*, para crear un sistema de transferencia de archivos. Para lograr esto, el cliente podrá realizar solicitudes a través de comandos que el servidor deberá responder.

Archivos

Los siguientes archivos deben ser utilizados para completar la actividad.

- `server/server.py`
  Este archivo **debe completarse** con los métodos necesarios para establecer el servidor y satisfacer las peticiones del cliente.
- `client/client.py`
  Este archivo **debe completarse** con los métodos necesarios para conectarse al servidor y ejecutar los comandos faltantes.
- `server/doggo.jpg`
  Este es el archivo que el cliente desea recuperar desde el servidor.
- `client/fine.gif`
  Este es el archivo sobreviviente del ataque que debe ser enviado al servidor.

Antes de comenzar a escribir código con frenesí, respira hondo y tómate unos quince minutos para leer ambos archivos en Python. Debes entender qué hace (y cómo) cada uno de ellos. Además, intenta crear una conexión entre el servidor y el cliente. Te entregamos suficiente código para lograr esto sin escribir una sola línea. Primero, enciende el servidor con `python server.py` y escribe un puerto como `8080`. Luego, de forma análoga, haz lo mismo con el cliente. Revisa qué aparece como *output* en ambos terminales. Por último, escribe `help` desde el lado del cliente.

Conexión y protocolos

Para establecer conexión entre el servidor y el cliente debes hacer uso de la librería `socket`. Debes completar las funciones `send` (en el servidor) y `receive` (en el cliente) para poder establecer comunicación desde el servidor hacia el cliente. Estos métodos no se encuentran implementados en las clases entregadas por lo que debes implementarlos para que cumplan con el protocolo explicado más adelante.

Por otro lado, los métodos `receive` (en el servidor) y `send` (en el cliente) están **parcialmente implementados**, ya que funcionan sólo para enviar mensajes de tamaño menor o igual a 128 *bytes*. Debes modificar ambos métodos —tanto en el servidor como en el cliente— para que utilicen el protocolo explicado más adelante y permita enviar mensajes de tamaños mayores a 128 *bytes*.

El **protocolo de comunicación** cliente–servidor debe permitir enviar mensajes de tamaño arbitrario. Esto se debe implementar a través de `4 bytes` que indiquen el **tamaño total del mensaje** antes de mandarse. Luego, se deben ir recibiendo *chucks* de **máximo de 4.096 *bytes***. Estos se deben concatenar hasta reconstruir el mensaje completo.

Importante: Es obligación utilizar la librería `pickle` para realizar la serialización de la información.
El uso de JSON **no está permitido**.

Funcionalidades del cliente

Los siguientes métodos deben ser implementados en el cliente para la interacción con el servidor.

- `$ ls`
  El servidor envía una lista con los archivos y carpetas de su directorio, desplegándose los nombres de cada uno en la consola del cliente.
- `$ upload <nombre-archivo>`
  El cliente entrega el nombre de un archivo en su directorio y lo envía al servidor. De esta manera, podemos respaldar (en el servidor) los archivos que sobrevivieron en nuestro computador.
- `$ download <nombre-archivo>`
  El cliente entrega el nombre de un archivo del directorio del servidor y el servidor lo envía. Luego, el cliente lo guarda en su directorio. De esta forma, podemos recuperar archivos perdidos en el ataque.
- `$ logout`
  El cliente cierra su sesión y le avisa al servidor, quien lo desconecta.
  (Este método ya está implementado[1].)

Funcionalidades del servidor

Los siguientes métodos deben ser implementados en el servidor para responder a las solicitudes de un cliente.

- `def list_filenames(self)`
  El servidor envía información sobre archivos y carpetas de su directorio al cliente.
- `def send_file(self, filename)`
  El servidor recibe el nombre de un archivo y lo envía al cliente.

[1] De nada.

- `def save_file(self, filename)`
  El servidor recibe un archivo del cliente y lo guarda en su directorio.
- `def disconnect(self)`
  El servidor desconecta al cliente actual, lo que permite recibir un nuevo cliente.
  (Este método ya está implementado[2].)

Bonus (cinco décimas)

Para obtener este fabuloso bonus, un cliente deberá poder enviar y recibir múltiples archivos utilizando los mismos comandos `upload` y `download`, una única vez. Para conseguir esto, debes realizar las modificaciones pertinentes en los métodos del cliente y del servidor. Por ejemplo, un cliente debería poder escribir: `download dog.gif cat.jpg fox.png` para descargar tres archivos desde el servidor.

Notas importantes

- Para **enviar y recibir** los archivos se debe hacer uso de **manejo de *bytes***.
- El servidor **siempre se mantiene en ejecución** y sólo puede aceptar **una conexión a la vez**.
- La estructura de las carpetas **debe mantenerse**: es necesario que los archivos `server.py` y `client.py` estén en carpetas separadas. Además, el archivo `doggo.jpg` **debe** estar en la misma carpeta donde esté `server.py`. De la misma forma, el archivo `fine.gif` **debe** estar en la misma carpeta donde esté `client.py`.

Uso de `.gitignore`

Para esta actividad debes utilizar un archivo `.gitignore` para no subir los archivos `doggo.jpg` y `fine.gif`. De lo contrario, tu actividad sufrirá un descuento de **tres décimas** por el uso incorrecto de `.gitignore` y **tres décimas** más por subir los archivos al repositorio. Este enlace te puede ayudar sobre "cómo eliminar del repositorio remoto" un archivo, luego de agregarlo al `.gitignore`.

[2] De nada, nuevamente.

Requerimientos

- (1,8p) Conexión y protocolos
  - (0,8p) Protocolo para recibir datos: método `send` en ambos participantes.
  - (1,0p) Protocolo para enviar datos: método `receive` en ambos participantes.
- (2,1 pts) Funcionalidades del cliente
  - (0,5p) Método `ls`.
  - (0,8p) Método `upload` y manejo de errores relacionados.
  - (0,8p) Método `download` y manejo de errores relacionados.
- (2,1p) Funcionalidades del servidor
  - (0,5p) Método `list_filenames`.
  - (0,8p) Método `send_file` y manejo de errores relacionados.
  - (0,8p) Método `save_file` y manejo de errores relacionados.
- (0,5p) Bonus

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC13/`
- **Hora del último *push* válido:** 16:40