Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-1

6 de junio de 2019
Actividad Formativa

Actividad 09

Networking

Introducción

El malvado mafioso Florenzini —primo del Tini Tamburini— ha logrado borrar tu repositorio de IIC2233, y se dió a la fuga a una **"conferencia"**. Por suerte, te acuerdas que lo clonaste en un servidor justo antes de que el desastre sucediera. Para esta actividad deberás usar tus sofisticados conocimientos de *networking* para conectarte a dicho servidor y recuperar tu tarea.

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.

- `servidor/servidor.py`
  Este archivo **debe completarse** con los métodos necesarios para establecer el servidor y satisfacer las peticiones del cliente.
- `cliente/cliente.py`
  Este archivo **debe completarse** con los métodos necesarios para conectarse al servidor y ejecutar los comandos faltantes.
- `servidor/doggo.jpg`
  Este es el archivo que el cliente desea recuperar desde el 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 servidor.py` y escribe un puerto como `8080`[1]. 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.

[1] Los puertos a usar deben ser mayores a 1023, ya que los otros están restringidos a programas del sistema operativo

Conexión y protocolos

Para establecer conexión entre el servidor y el cliente debes hacer uso de la biblioteca `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. Los mensajes deben enviarse de manera serializada utilizando el formato `pickle`. El inicio del mensaje debe incluir **4 *bytes*** que indiquen el **tamaño total del mensaje** (el tamaño total no incluye a esos 4 bytes) antes de mandarse. Luego, el receptor debe recibir *chunks* de **máximo 2048 *bytes* (2 KB)**. Estos se deben concatenar hasta reconstruir el mensaje completo.

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

Funcionalidades del cliente

El cliente interactúa con el servidor enviándole mensajes con comandos, y el servidor envía una respuesta de acuerdo a la descripción de cada comando. Se deben implementar los siguientes comandos.

- `$ ls`
  El cliente solicita una lista con los archivos y carpetas de su directorio. El servidor responde con esta lista, y el cliente la despliega.
- `$ echo <nombre-archivo>`
  Con este comando debes pedirle un *input* de texto al usuario y luego enviar tanto `nombre-archivo` como el *input* de texto hacia el servidor. El servidor debe crear un archivo con el nombre `nombre-archivo`, y que contenga el texto ingresado como *input*. En caso de que el archivo ya exista, éste se deberá sobreescribir. El flujo de este comando es el siguiente:
  1. El usuario escribe `echo mi_archivo.txt` en la consola y apreta la tecla `enter`
  2. Se despliega una nueva línea donde se le pide el texto al usuario el cual luego es enviado hacia el servidor.
  3. El servidor crea el archivo de nombre `mi_archivo.txt`, y escribe dentro el texto que ingresó el usuario.
- `$ descargar <nombre-archivo>`
  El cliente envía el nombre de un archivo del directorio del servidor y el servidor responde enviando el archivo. El cliente lo guarda en su directorio. De esta forma, podemos recuperar archivos perdidos en el ataque. En caso de que el archivo no exista en el servidor, no se realizará ninguna acción.
- `$ cerrar_sesion`
  El cliente cierra su sesión y le avisa al servidor, quien lo desconecta. Este método ya está implementado[2].

[2] De nada.

¿Cómo utilizo las funcionalidades del cliente?

Existen dos maneras de utilizar las funcionalidades del cliente. La más simple e intuitiva consiste en ingresar exactamente un comando a la vez, tal como muestra el ejemplo a continuación.

```
$ descargar solución_AC09.py
```

La segunda manera consiste en ingresar varios comandos a la vez, separados a través de un punto y coma (`;`), tal como se muestra en el ejemplo a continuación. Estos deberán ser ejecutados en el orden en que fueron ingresados.

```
$ ls; echo texto.txt; descargar meme.png; cerrar_sesion
```

Note que la funcionalidad `descargar` podrá ser ejecutada luego de que el usuario ingrese el *input* requerido por la funcionalidad `echo`. En el fondo, este ejemplo es equivalente a haber ejecutado:

```
$ ls
$ echo texto.txt
$ descargar meme.png
$ cerrar_sesion
```

Manejo de errores

Tu programa debe ser capaz de manejar los distintos errores que se pueden producir a la hora de ingresar los comandos. Esto lo debes hacer a través del comando `validar` que encontrarás en el cliente.

- **Error de comando.**

  En caso de que se ingrese un comando inválido, se deberá mostrar el siguiente mensaje de error en la pantalla del cliente.

  ```
  el comando '<comando>' no es válido
  ```

  A continuación se encuentran algunos ejemplos de comandos inválidos y las respuestas esperadas.

  ```
  $ descargr .gitignore
   el comando 'descargr' no es válido

  $ print index.html
   el comando 'print' no es válido

  $ mkdir
   el comando 'mkdir' no es válido
  ```

- **Error de argumentos.**

  En caso de que el comando ingresado sí sea válido, pero el número de argumentos no lo sea, se deberá mostrar el siguiente mensaje de error en la pantalla del cliente.

  ```
  el número de argumentos no es válido
  ```

  A continuación se encuentran algunos ejemplos de comandos válidos con un número inválido de argumentos y las respuestas esperadas.

```
$ cerrar_sesion civilización.py
 el número de argumentos no es válido

$ descargar
 el número de argumentos no es válido

$ ls -A
 el número de argumentos no es válido

$ echo archivo texto
 el número de argumentos no es válido
```

- **Error al ingresar múltiples comandos.**

  Las instrucciones para el caso en el que se ingresen varios comandos en una misma línea es la siguiente. Se deberá ejecutar con normalidad todo comando que no posea errores, mostrando un mensaje de error únicamente cuando corresponda y en el momento indicado, tal como si hubiesen sido ingresados en líneas distintas.

Funcionalidades del servidor

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

- `def lista_directorio(self)`
  El servidor envía información sobre archivos y carpetas de su directorio al cliente.
- `def enviar_archivo(self, path)`
  El servidor recibe el nombre de un archivo y lo envía al cliente.
- `def crear_archivo(self, path)`
  El servidor recibe el nombre del archivo a crear, imprime el contenido (`content`) en consola y luego lo escribe.
- `def desconectar(self)`
  El servidor desconecta al cliente actual, lo que permite recibir un nuevo cliente.
  (Este método ya está implementado[3].)

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 `servidor.py` y `cliente.py` estén en carpetas separadas. Además, el archivo `doggo.jpg` **debe** estar en la misma carpeta donde esté `servidor.py`.

[3] De nada, nuevamente.

Uso de `.gitignore`

Para esta actividad debes utilizar un archivo `.gitignore` para no subir el archivo `doggo.jpg`. Este enlace te puede ayudar sobre "cómo eliminar del repositorio remoto" un archivo, luego de agregarlo al `.gitignore`.

Requerimientos

- (1,6 pts) Conexión y protocolos.
  - (0,6) Protocolo para enviar datos: método `send` en ambos participantes.
  - (1,0) Protocolo para recibir datos: método `receive` en ambos participantes.
- (2,0 pts) Funcionalidades del cliente.
  - (0,4) Método `ls`.
  - (0,6) Método `echo`.
  - (0,6) Método `descargar`.
  - (0,4) Manejo de múltiples comandos ingresados a la vez.
- (0,6 pts) Manejo de errores.
  - (0,2) Error de comando.
  - (0,2) Error de argumentos.
  - (0,2) Errores al ingresar múltiples comandos a la vez.
- (1,8 pts) Funcionalidades del servidor.
  - (0,4) Método `lista_directorio`.
  - (0,7) Método `crear_archivo` y manejo de errores relacionados.
  - (0,7) Método `enviar_archivo` y manejo de errores relacionados.

Descuentos

- (0,5 p) Mal/Incorrecto uso de `.gitignore`. Se sube el archivo `doggo.jpg` al repositorio de GitHub.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC09/`
- **Hora del *push*:** 16:30

Auto-evaluación

Como esta corresponde a una actividad formativa, te extendemos la instancia de responder, después de terminada la actividad, una auto-evaluación de tu desempeño. Esta se habilitará a las **16:50 de jueves 6 de junio** y tendrás plazo para responderla hasta las **23:59 del día siguiente, viernes 7 de junio**. Puedes acceder al formulario mediante el siguiente enlace:

`https://forms.gle/iGpCL1eEGRmSWa8z6`

El asistir, realizar la actividad y responder la auto-evaluación otorgará como bonificación al alumno 2 décimas para sumar en su peor actividad sumativa del semestre.