Tarea 1: Cliente eco TCP para medir performance
Redes
Plazo de entrega: 9 de septiembre 2024

José M. Piquer

1. Descripción

Su misión, en esta tarea, es modificar el cliente TCP con threads (`client_echo3.py`) para usarlo como un medidor de performance. Los servidores que usaremos para la medición son: `server_echo2.py`, `server_echo4.py` y `server_echo5.py`.

Ahora el cliente usa su archivo de entrada y el de salida como archivos binarios (así pueden probar con cualquier tipo de archivo), y recibe como argumento el tamaño de las lecturas y escrituras que se harán (tanto de/desde el socket como de/desde los archivos). En Python, para poder hacer esto, deben usar las funciones: `sys.stdin.buffer.read()` y `sys.stdout.buffer.write()`.

También deben modificar el código para detectar bien el término de la ejecución: el thread enviador debe contar los bytes enviados hasta el EOF, y el receptor debe esperar todos esos bytes hasta terminar, y ahí termina el programa.

El cliente que deben escribir recibe el tamaño de lectura/escritura, servidor y puerto.

```
./client_bw.py size host port < in > out
```

Para medir el tiempo de ejecución pueden usar el comando `time`.

Los servidores para las pruebas deben correrlos localmente en localhost. Dejaremos algunos corriendo en anakena también.

Un ejemplo de medición sería (suponiendo que tengo un servidor corriendo en el puerto 1818):

```
% time ./client_bw.py 1500 localhost 1818 < /etc/services > OUT
```

```
0.09 real         0.04 user         0.03 sys
```

Para enviar/leer paquetes binarios del socket usen send() y recv() directamente, sin pasar por encode()/decode(). El arreglo de bytes que usan es un bytearray en el concepto de Python.

2. Mediciones

El cliente sirve para medir eficiencia. Lo usaremos para ver cómo elegir el mejor servidor en diferentes situaciones y también probar cuánto afecta el largo de las lecturas/escrituras en la eficiencia.

Para probar en localhost necesitan archivos realmente grandes, tipo 0.5 o 1 Gbytes (un valor que demore tipo 5 segundos).

Diseñen un experimento que pruebe con:

- 1 archivo grande y distintos tamaños de lectura/escritura
- 3 archivos grandes en paralelo, distintos tamaños de lectura/escritura
- 20 archivos medianos en paralelo (que sumen lo mismo que la suma de los 3 grandes en bytes), distintos tamaños de lectura/escritura
- 100 archivos chicos en paralelo (que sumen lo mismo anterior), distintos tamaños de lectura/escritura

El experimento completo deben correrlo con los tres servidores provistos.

3. Entregables

Básicamente entregar el archivo con el cliente que implementa el protocolo y los scripts shell que utilizó para los experimentos.

En un archivo aparte responder las preguntas siguientes (digamos, unos 5.000 caracteres máximo por pregunta):

1. Explique los experimentos realizados, justificando las mediciones elegidas. Describa el computador, sistema operativo, etc donde corrió su experimento.
2. Muestre sus resultados y recomiende los mejores valores para los diversos casos según lo medido.

3. Trate de justificar las diferencias de tiempos medidos: ¿por qué influye el tamaño de las lecturas/escrituras? ¿por qué a veces es mejor un servidor y a veces otro?

4. Corra algunos experimentos más pequeños con anakena: ¿se parecen los resultados? ¿por qué?

4. Strings y bytearrays en sockets

Una confusión clásica en los sockets en Python es la diferencia entre enviar un string y/o un bytearray. Partamos por los strings, que Uds conocen mejor: los strings no son simplemente arreglos de bytes (alguna vez lo fueron, pero hoy pueden contener hasta alfabetos asiáticos y árabes), son codificaciones en un *encoding* particular. Los sockets no soportan strings, es decir, Uds no pueden llegar y enviar/recibir un string por el socket, deben convertirlo a un bytearray, que es una colección de bytes binarios primitivos, no se interpretan. La forma de convertir un string a un bytearray es aplicando la función *encode()* y un bytearray a un string, con la función *decode()*. Ojo que si se aplican a cualquier cosa, pueden fallar, particularmente *decode()* falla si uno le pasa cualquier cosa en el bytearray. Entonces, si quiero enviar/recibir el string 'niño' hago:

```
enviador:
        s = 'niño'
        sock.send(s.encode('UTF-8')) # UTF-8 es el encoding clásico hoy
receptor:
        s = sock.recv().decode()     # recibe s == 'niño'
        print(s)
```

En cambio, si recibo bytes y quiero escribirlos en un archivo cualquiera, no sé si hay strings o no dentro, entonces mejor es no transformarlo y siempre usar bytes puros:

```
enviador:
    data = fdin.read(MAXDATA)
    sock-send(data)

receptor:
    data = sock.recv()
    fdout.write(data)
```

En esta tarea sólo usaremos bytearrays.