Universidad Nacional de la Patagonia San Juan Bosco

Facultad de Ingeniería

Sede Trelew

Sistemas Distribiudos

Alumno:

Torres, Nelson Jonathan

Trabajo Práctico N°1

Cliente / Servidor. Sockets. RPC. Threads. Concurrencia. RFS.

2020

Punto1

1. Tome el código provisto en la carpeta p1 de la unidad 1.

a) Analice los fuentes client11.py y server11.py y modifíquelos para que la consulta del cliente y la respuesta del servidor sea más interactiva, ya sea mediante interfaz gráfica o línea de comandos. Además agregue un comando mediante el cual el cliente indica al servidor la finalización de su ejecución.

```
import socket

server_address = ('0.0.0.0', 8080)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(server_address)

#message = 'I am CLIENT\n'
#client.send(message.encode('utf-8'))
while True:
    respuesta = input('Ingrese una S para salir\n')
    client.send(respuesta.encode('utf-8'))
    from_server = client.recv(4096)
    if from_server.decode() == 's':
        #client.close
        break

client.close()
print('cliente fin')
print(from_server.decode())
```

En la parte del cliente agregue un bucle infinito del cual solamente saldrá presionando la tecla “**s**”. Lo que en cliente envía al servidor se hace a través de clien.send y va codificado.

```python
import socket

server_address = (('0.0.0.0', 8080))
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(server_address)
server.listen()

message = 'Tu conexion fue cerrada con Exito!!\n'

while True:
        print('Server disponible!')
        connection, client_address = server.accept()
        from_client = ''
        while True:
                data = connection.recv(4096)
                #connection.send(data.encode('utf-8'))
                if data.decode() == 's':break
                from_client += data.decode() #acumula los mensajes que
llegan.
                connection.send(from_client.encode('utf-8'))
        print('salio')
        connection.send(data)
        connection.close()
        print('cliente desconectado \n')
```

En el servidor se reciben los mensajes con connection.recv. Cuando llegue una “**s**” se la envia nuevamente al cliente para que cierre su conexión. Sino acumula los mensajes que llegan.
Cuando el cliente reciba la letra correspondiente cerrará el socket.

b) Analice los fuentes client12.py y server12.py para ver su funcionamiento. Modifique el tamaño de los buffers para que sean de longitud fija: 10 3 , 10 4 , 10 5 y 10 6 bytes. Explique las diferencias obtenidas al ejecutar en cada caso.

Cuando se probo con 100 y el resultado fue
Servidor

```
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/p1 (1)/
p12$ ./server 8888 100
Paso 1: La función read devolvió 100 bytes.
--
Paso 2: Se escribieron 99 bytes.
Paso 2: La función write devolvió 99 bytes.
--
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/p1 (1)/
p12$
```

Cliente.

```
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/p1 (1)/
p12$ ./client localhost 8888 100
Paso 1: Se escribieron 100 bytes.
Paso 1: La función write devolvió 100 bytes.
--
Paso 2: La función read devolvió 99 bytes.
```

Luego con 1000. Servidor

```
jony@jony-Torres:~/Descargas/Universidad/Sistema
p12$ ./server 8888 1000
Paso 1: La función read devolvió 1000 bytes.
--
Paso 2: Se escribieron 999 bytes.
Paso 2: La función write devolvió 999 bytes.
--
```

Cliente

```
jony@jony-Torres:~/Descargas/Universidad/Sistemas
p12$ ./client localhost 8888 1000
Paso 1: Se escribieron 1000 bytes.
Paso 1: La función write devolvió 1000 bytes.
--
Paso 2: La función read devolvió 999 bytes.
```

Con 100000

```
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
1)/p12$ ./server 888 100000
ERROR on binding: Permission denied
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
1)/p12$ ./server 888 10000
ERROR on binding: Permission denied
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
1)/p12$ ./server 8888 100000
Paso 1: La función read devolvió 65482 bytes.
--
Paso 2: Se escribieron 65482 bytes.
Paso 2: La función write devolvió 65482 bytes.
--
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
1)/p12$
```

Cliente

```
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
1)/p12$ .
/client localhost 8888 100000
Paso 1: Se escribieron 100000 bytes.
Paso 1: La función write devolvió 100000 bytes.
--
Paso 2: La función read devolvió 65482 bytes.
jony@jony-Torres:~/Descargas/Universidad/Sistemas Distribuidos/SD/p1 (
1)/p12$
```

Se puede observar que cuanto mas grande el buffer hay mas perdida de datos. Tambien, en ningun envio se pudo leer el total enviado.

2a. En un servidor con estado el servidor almacena alguna información del cliente mediante un id único (tocken) que usará en próximas llamadas

En un servidor sin estado, no se almacena nada del cliente.

B. El servidor es con estado.

```python
from structures import *

server_address = (('0.0.0.0', 8081))
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(server_address)
server.listen()

#acumulador = defaultdict(lambda: 0)

payload=payload_X()
acum=0
while True:
    print('Server disponible!')
    connection, client_address = server.accept()
```

En vez de utilizar una estructura para almacenar el acumulador, utilizo una variable inicializada en cero. Cuando el cliente cierre la conexión se reiniciara el acumulador sin guardar registros de lo que tenia el cliente.

3a.

```
syntax = "proto3" ;
message Path {
string value = 1 ;
}
message PathFiles {
repeated string values = 2 ;
}
service FS {
rpc ListFiles(Path) returns (PathFiles){};
}

service OpenFile {
rpc open (Path)
returns (PathFiles) {};
}

service ReadFile {
rpc read(Path)
returns (PathFiles) {};
}
service CloseFile {

rpc close(Path)
returns (PathFiles) {};
}
```

Dentro de Proto3 ingrese los servicios a ofrecer.

Genere las clases gRPC para python con los comandos

$ pip install grpcio
$ pip install grpcio-tools
y genere las clases

$ python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. file_system.proto

4: (Suposiciones)

A-Como no pude completar el punto 3 y por ende no poder hacer pruebas, supongo que si se cancela en medio de la operación a un cliente, el servidor guardara el estado en donde esta, y cuando se reanude se terminara la operación.

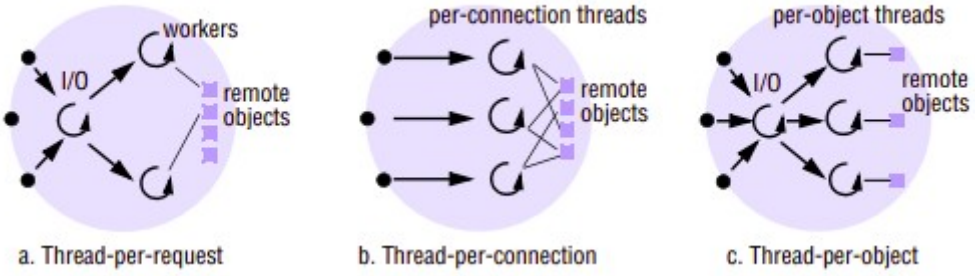
B- Cuando se cancela el servidor supongo que habrá error en algún momento de la ejecucion normal, ya sea cuando se intente leer mas partes del archivo o cerrar la sesion.

6-Supongo que con la implementación con hilos, los clientes serán atendidos con distintos hilos para cada cliente (cuando intenten leer). Ya que no es un recurso que deba bloquearse para otros clientes.

Hilo por requerimiento: el subproceso de E / S genera un nuevo subproceso de trabajo para cada solicitud, y ese trabajador se destruye a sí mismo cuando ha procesado la solicitud en su objeto remoto designado. Esta arquitectura tiene la ventaja de que los subprocesos no compiten por una cola compartida y el rendimiento se maximiza potencialmente porque el subproceso de E / S puede crear tantos trabajadores como solicitudes pendientes hay. Su desventaja es la sobrecarga de las operaciones de creación y destrucción de subprocesos.

Hilos por conexión: La arquitectura hilo por conexión asocia un hilo con cada conexión. El servidor crea un nuevo hilo de trabajo cuando un cliente establece una conexión y lo destruye cuando el cliente cierra la conexión. En el medio, el cliente puede realizar muchas solicitudes a través de la conexión, dirigidas a uno o más

Hilos por objetos: La arquitectura hilo por objeto asocia un hilo con cada objeto remoto. Un subproceso de E / S recibe solicitudes y las pone en cola para los trabajadores, pero esta vez hay una cola por objeto.



a. Thread-per-request b. Thread-per-connection c. Thread-per-object

Supongo que es por requerimiento, ya que el hilo una vez que el cliente cierra la conexión se eliminaria.