# Reporte

### Anexo

Queremos resaltar que todas las pruebas fueron desarrolladas en un computador en un MacBook pro-2020 de 8gb de RAM. El rendimiento en otros sistemas puede varias.

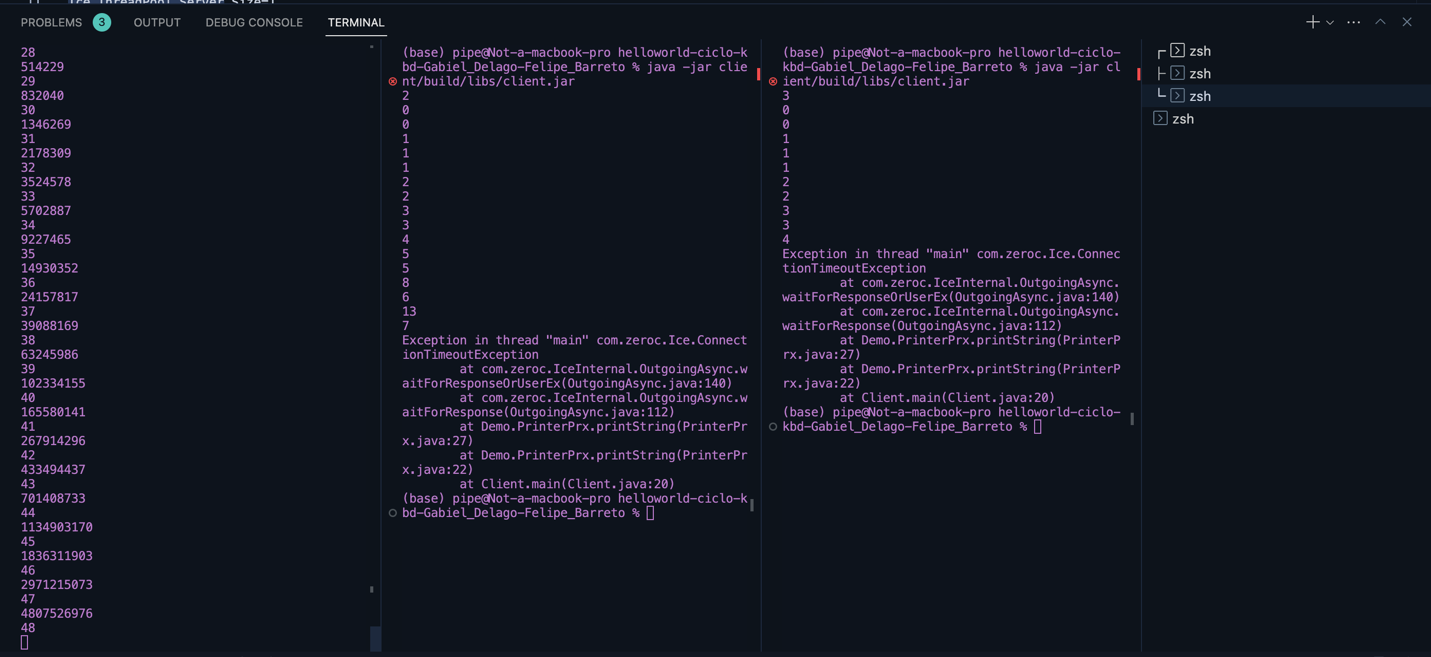
Concurrencia real: El sistema atiende varios procesos a la vez.

Concurrencia virtual: El sistema hace procesos lo suficientemente rápido por lo cual aparenta que atiende varios procesos a la vez.

## Parte 1

Lo primero que hicimos fue modificar el código de tal manera que apenas, el servidor, acabe de procésese un Fibonacci, el cliente le mande uno nuevo, esto incrementando de 1 en 1. Cabe resaltar, que la implementación del Fibonacci es con el uso de recursividad, por ende, no es la más optima, pero en este caso nos ayuda a evidenciar un timeout más rápido.

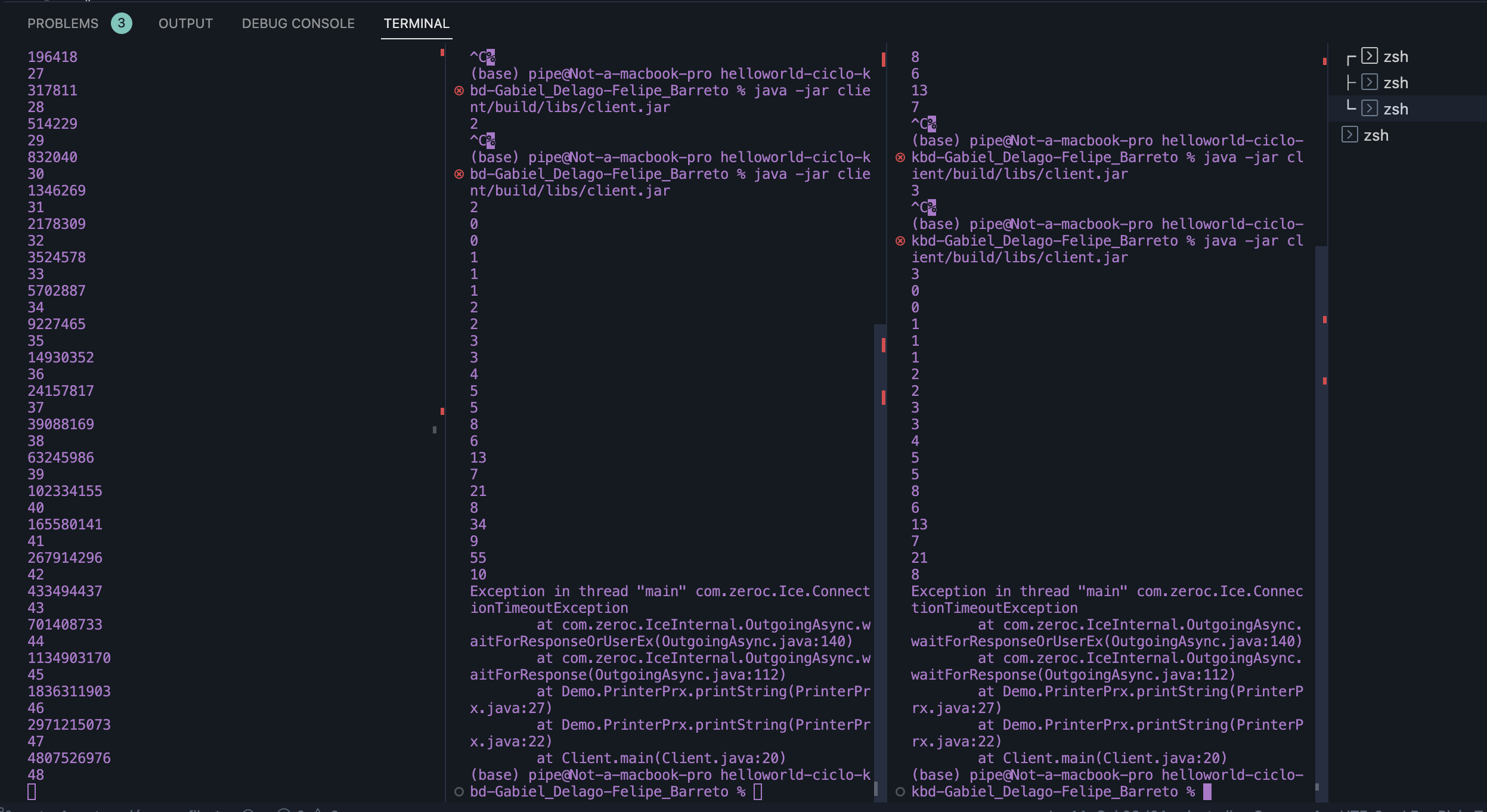
Ahora, empezamos a probar para encontrar las limitaciones del código, primero, queremos ver qué pasa si solo tenemos una sola thread en la thread pull de ice. Dado que la implementación del threadpull de ice sigue el patrón de follower/leader no se va a evidenciar concurrencia, dado que cuando se creen varios clientes (followers) estos van a caer sobre el mismo servidor (leader). Así se ve cuando tenemos 3 clientes y un servidor con solo una thread.



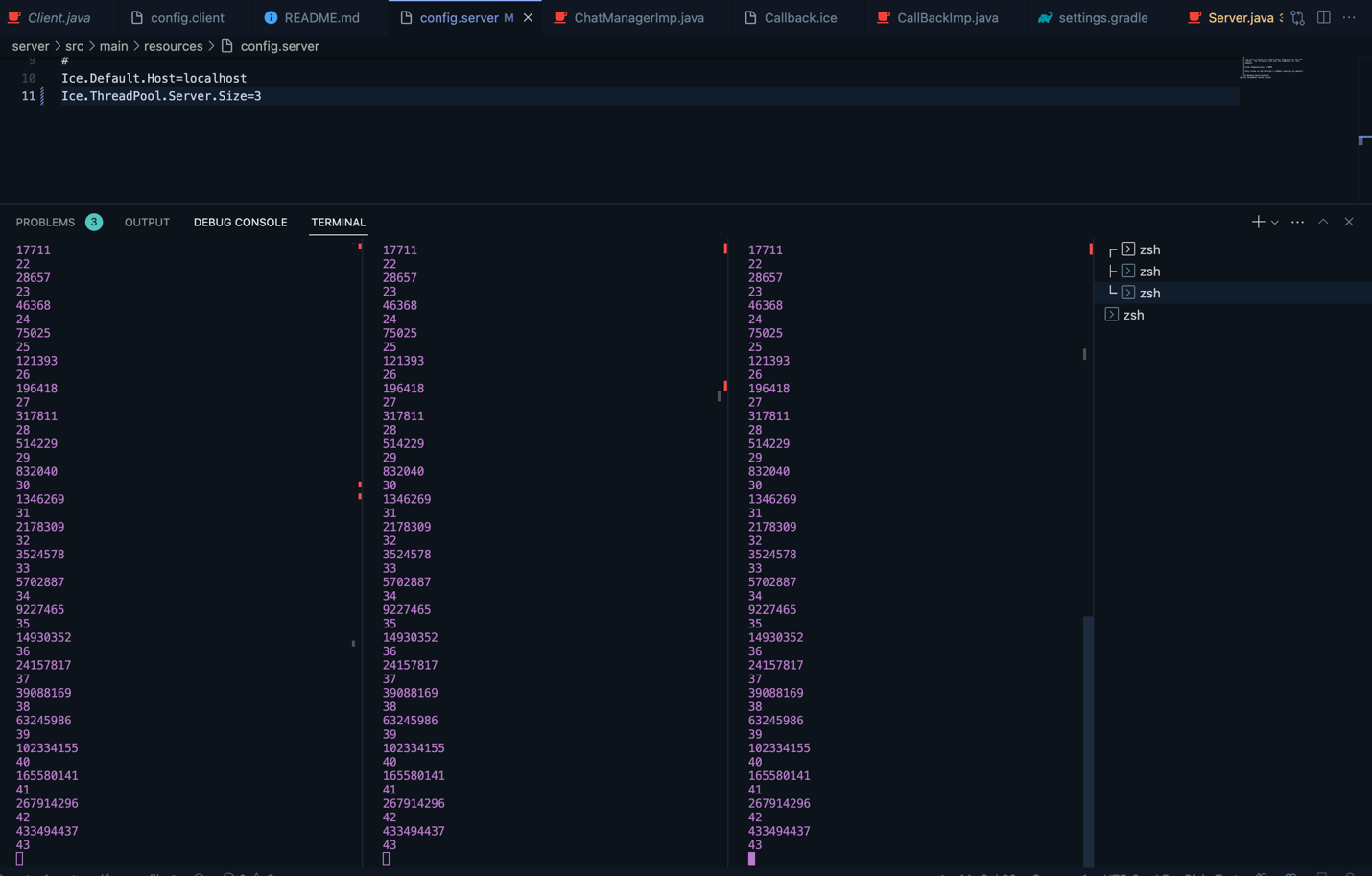
Podemos ver que mientras el cliente original sigue, el resto se van quedado atrás, hasta que el primero llega a un número que se demora calcular (por la implementación) y se llega a un timeout en los otros clientes.

La siguiente prueba, se corrieron con algunos procesos en segundo plano, en este caso, Spotify, varios contenedores de Docker y youtube

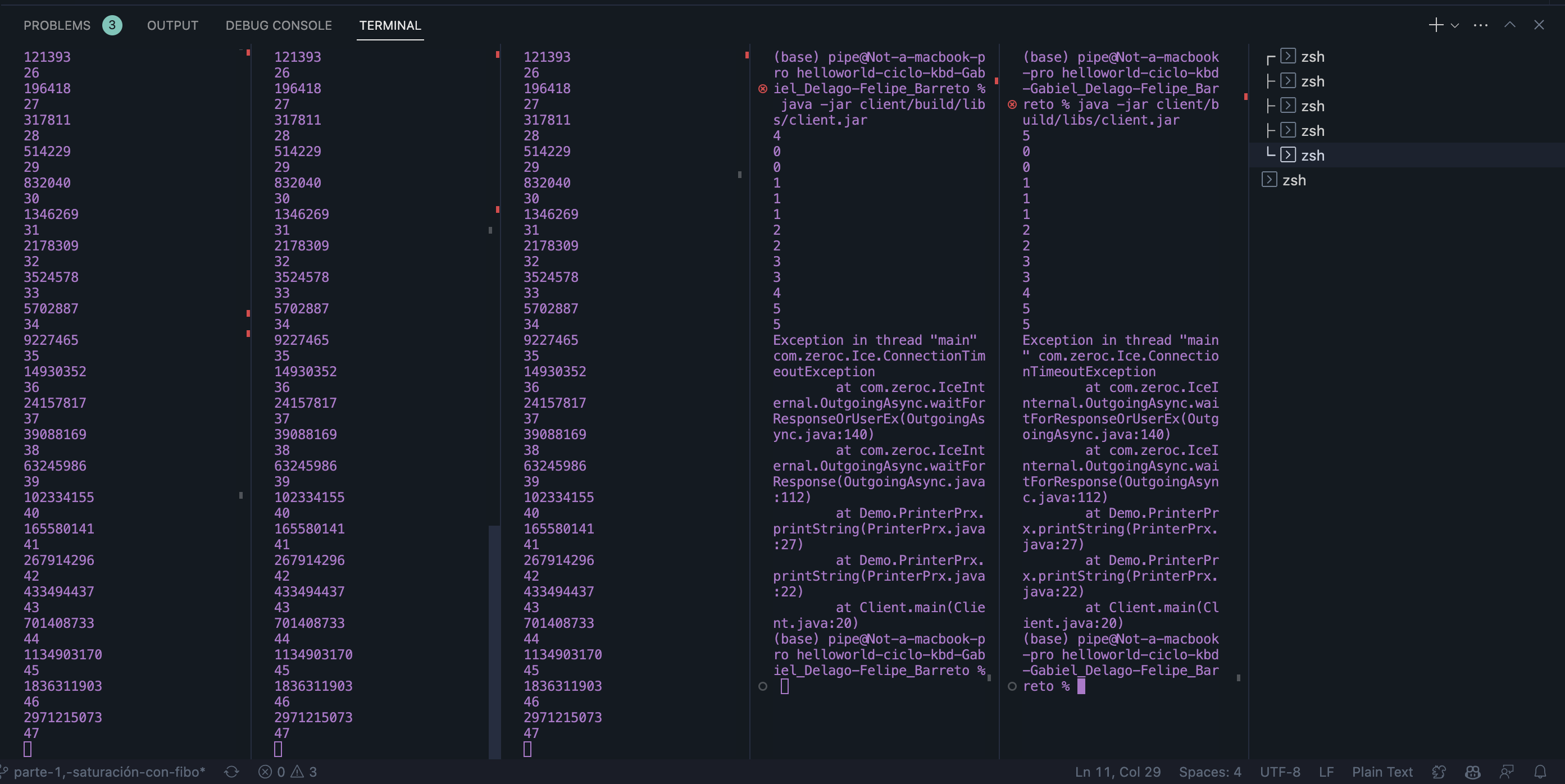
Además, podemos llegar a un timeout más rápido si el computador está haciendo otros procesos, para la primera prueba solo estaba el visual, Chrome y este doc abierto



Dicho esto, veamos qué pasa si tenemos varios threads en la thread pull, según la implementación del patron follower/leader, esto nos dará concurrencia real, por lo cual el timeout no se debería dar o se debería dar mucho después



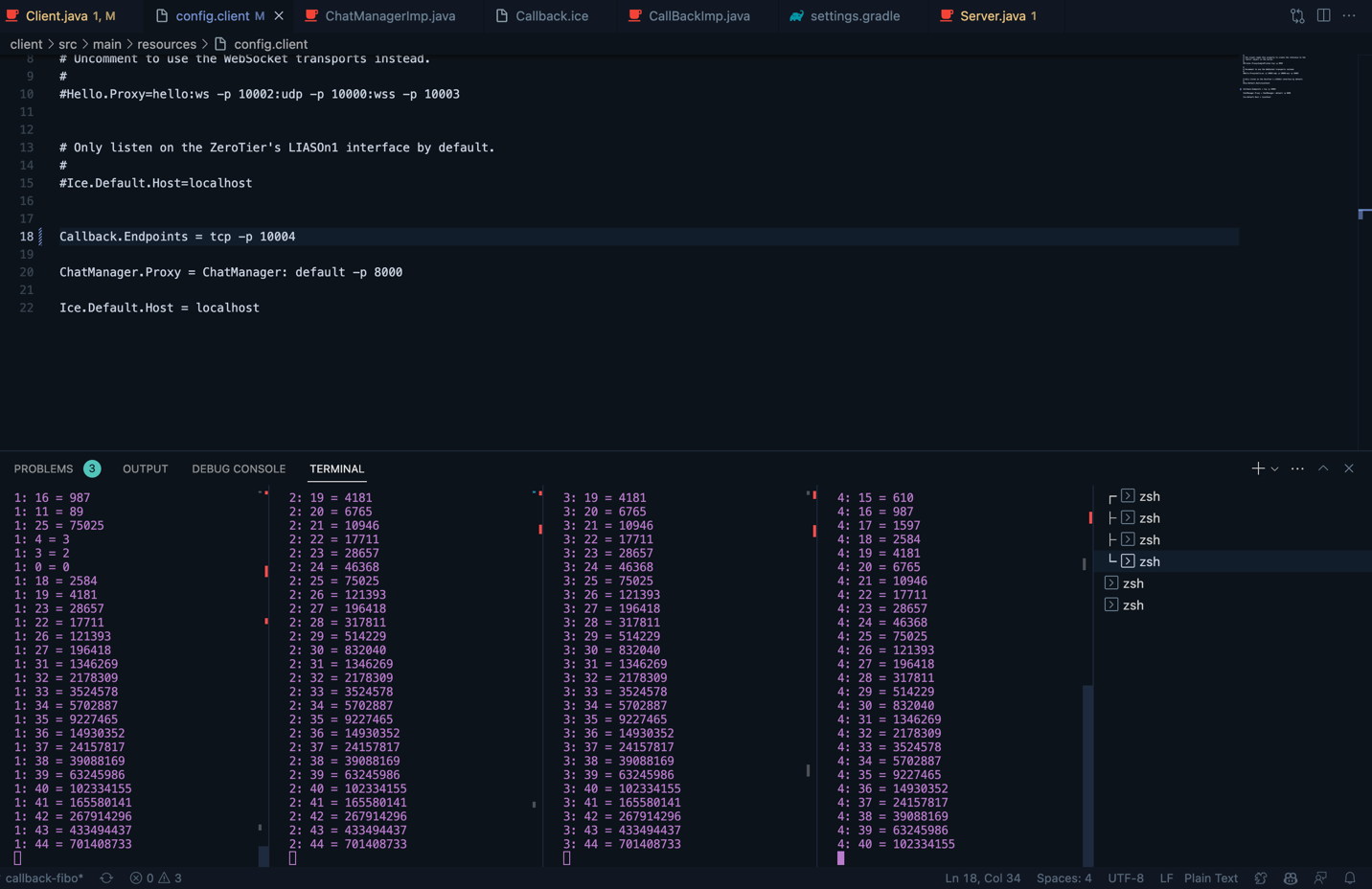
Al tener 3 threads y 3 clientes, el programa funciona de manera concurrente. Dicho esto, veamos qué pasa si se añaden más clientes.



De nuevo, tenemos el mismo problema donde se llega a un timeout por no tener sufientes threads en la thread pull.

## Parte 2

En esta instancia, ahora tenemos el patron de diseño callback implementado, por ende, en teoría, no deberíamos tener este problema.



Efectivamente, no tuvimos el mismo problema, llegando a 4 clientes con 3 threads en la threadpull, esto se debe a que, el cliente ya no se queda esperando una respuesta, este solo manda el número de Fibonacci que quiere que se calcule, y listo, el servidor le manda una respuesta cuando este cálculo esté listo.

También cabe resaltar que cada llamado crea un nuevo thread por ende se llega a ver que las respuestas no llegan en orden, por ejemplo, en el cliente 1 recibió la respuesta del Fibonacci 25 antes que la del 4, aunque no estamos muy seguros del porqué, lo único que se nos ocurre es que a ese proceso se le asigno un hilo de los núcleos de eficiencia del Mac, que son más lentos que el resto, aunque no estamos seguros.

Finalmente, dado que a tenemos 3 threads en el threadpull y cada llamado crea su propia thread, tenemos concurrencia real.