FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES
UNIVERSIDAD NACIONAL DE CÓRDOBA

Sistemas Operativos 2

Trabajo Práctico

Comunicación Interprocesos

Docentes
Martina, Agustín
Martinez, Pablo
Morales, Julián
Abratte,Diego
Maschio, Alfonso

Alumno: Vargas Rodríguez, Diego Rubén 36.983.867

Abril de 2021

Indice

Introducción

El sistema operativo Linux emplea sistemas para comunicar los procesos entre ellos. En clase hemos visto varios, teniendo un pantallazo de que hacen cada uno. Este trabajo práctico nos pide que empleemos estos sistemas de comunicación para comunicar varios procesos, a la vez que se conectan sockets y se envían mensajes siguiendo el modelo Publisher Suscriber.

Consigna:

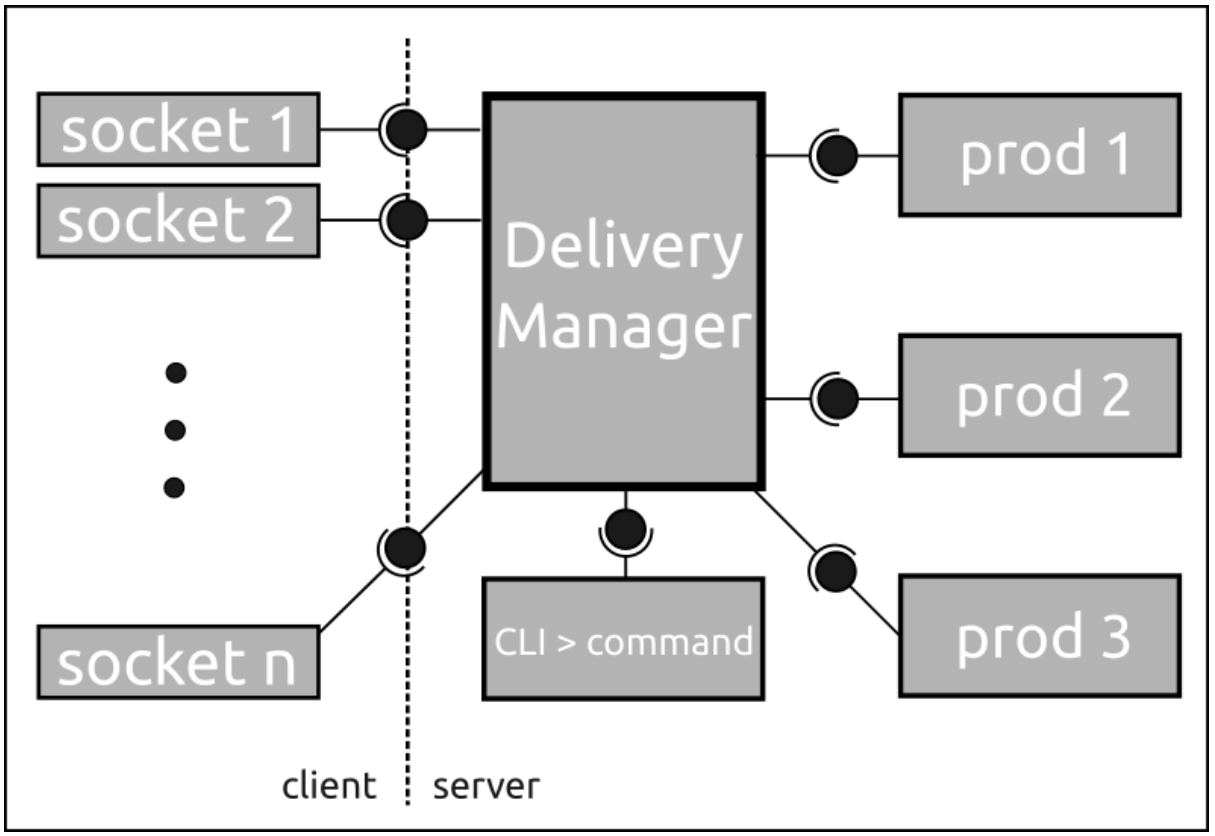

Figura 1

Se pide que se desarrolle un programa que siga la figura 1. Por un lado, hay que codificar los socket clientes para que se conecten a un Delivery Manager, y por otro, codear el Delivery Manager para que tenga 3 procesos asociados que crearán mensajes, los cuales pueden ser enviados a los socket conectados.

Delivery Manager

Se debe implementar un proceso que se encargue de recibir mensajes y enviarlos a todos los suscriptores válidos. A este proceso, lo vamos a llamar *DeliveryManager*. El mismo debe contemplar lo siguiente:

- Debe poseer una interfaz (CLI) que acepte únicamente los siguientes comandos:
    - add <socket> <productor> : Este comando agrega el socket a una lista correspondiente al servicio, para ser validado.
    - delete <socket> <productor>: Este comando borra el host como suscriptor dejando de enviarle mensajes.
    - log <socket>. Este comando comprime el log local del *DeliveryManager* y lo envía a <socket>.
- El *DeliveryManager* debe validar que los hosts agregados sean aptos para recibir mensaje del tipo <Mensaje>|<Checksum>.
- Una vez validado el host, todos los mensajes que recibe el *DeliveryManager* de un productor, deben ser enviados a los suscriptores correspondientes.
- En caso que se desconecte un suscriptor, el servicio para los demás suscriptores no debe verse afectado. Luego de cinco segundos de mensajes fallidos, debe ser eliminado de la lista. Si el suscriptor vuelve antes de esos 5 segundos, debe enviarle todos los mensajes encolados.
- El *DeliveryManager* debe loguear todos los mensajes enviados, tanto el origen como el destino, también se agrega o se elimina un suscriptor y cuando es validado. El formato del log debe <datetime> <Mensaje>.

Productores

Se deben implementar tres productores:

- Un productor que envía un mensaje random con una tasa de X/segundos.
- Un productor que envía la memoria libre del Sistema, cada Y/segundos.
- Un productor que debe enviar load del sistema normalizado, cada Z/segundos. Pueden elegir otro productor, justificándose. X, Y y Z deben ser distintos.

Suscriptores

- Puede existir hasta mil suscriptores, y deben ser capaz de vivir en una misma instancia.
- Los suscriptores deben esperar que el *DeliveryManager* los suscriba mediante el comando add, es decir, que lo suscriba.
- Los suscriptores deben validar el checksum de los mensajes recibidos y loguear el mensaje, para luego ser descartados.

Restricciones

El diseño debe contemplar toda situación no descrita en el presente documento y se debe hacer un correcto manejo de errores.

Desarrollo

Para llevar a cabo el proyecto, se empezó por lograr que dos socket se conecten y envíen mensajes. Los socket suministrados en clase fueron modificados según los errores que el compilador emitía para que funcionen con las flags correspondientes y se modificó en primera instancia para que el que escriba los mensajes sea el servidor y los clientes reciban.

Luego se empleó las funciones fork() y execv() para crear procesos hijos en el servidor, ahora llamado *Delivery Manager*. Se crean 3 procesos que serán los productores. Los cuales se comunicarán con el Delivery Manager a través de Cola de Mensajes.
La tasa de mensajes por segundo que envíen estos procesos productores se manipula desde el archivo recursos.h donde está definido *X, Y y Z.*
El productor 1 se encarga de crear un mensaje aleatorio, para lo que pusimos una función rand para que emita un número aleatorio y lo pase por mensaje.

El siguiente paso fue lograr que varios clientes se conectasen a la vez, para poder cumplir el requerimiento de 5000 clientes. Para ello, se empleó el código sacado del sitio de ibm[1] y se lo agregó al código, de forma tal que empleando la librería y función poll(), se lograse varias conexiones. Además, hubo que modificar configuraciones, modificando el valor en el archivo /proc/sys/net/core/somaxconn.

Siguiente a eso, se desarrolló una interfaz de línea de comando o CLI para pasarle los comandos que se solicita en el enunciado. Esta CLI es otro proceso hijo que se crea en el Delivery Manager, y se comunica por la misma cola de mensajes que los demás productores. Cuando llega un mensaje de esta al DM, este valida los comandos y realiza las funciones que solicita.

Para lograr la suscripción, se necesita por un lado que los clientes se vayan conectando al Delivery Manager, este les irá haciendo *accept* y se agrega a una lista de clientes conectados llamada **clientes_conectados**. Luego, cuando recibe el comando add, por socket (ip:puerto) se irán guardando estos clientes en listas de suscripción. Hay una por cada productor: **suscriptos1**, **suscriptos2** y **suscriptos3**. Para esto, recorre la lista de **clientes_conectados** y cuando lo encuentra, toma file descriptor, ip y puerto, y lo agrega a la lista que se le asignó con el productor. Cuando llegue un mensaje de un productor, este se loguea y luego recorre su lista de suscripción, enviandoselo a cada uno.
Para eliminar un cliente de una lista de suscripción, se recorre la lista de suscriptos en busca de ese cliente, por ip y puerto, cuando hay coincidencia, el nodo que lo contiene es borrado, no sin antes desplazar el puntero del nodo anterior, al que le sigue al que será eliminado, de esta forma la lista garantiza su existencia.

Para almacenar los clientes que se van conectando al Delivery Manager, se implementa listas con campos para guardar file descriptors, ip y puertos. Con eso podemos encolar los que se conectan, como así también agregarlos a una de las listas de distribución. Para las listas, se emplearon los códigos que pasaron por el slack [2] y se modificaron para que almacene los datos solicitados.

Para evaluar los clientes desconectados por 5 segundos, se plantea una lista de **clientes_desconectados** que es revisada cuando un cliente se conecta, para ver si estaba antes. Si no se encuentra, es agregado a la lista **clientes_conectados**, comunmente. Por el contrario, si se encuentra que ya estaba desconectado, es que se está conectando de nuevo, por lo que se busca en la lista **clientes_conectados** y se cambia el file descriptor. Cada 5 segundos el código se fija si todos los clientes están conectados, cuando alguno no responde, es enviado a la lista de **clientes_desconectados** y quitado de las listas de distribución, y si a los 5 segundos no hubo conexión (mediante un flag se avisa esto), el cliente es eliminado de la lista de **clientes_desconectados** también.

Hasta acá llega el trabajo realizado hasta el tiempo de entrega pactado, luego se seguirá terminando para cumplir la parte de log.

Conclusión

Este trabajo se trata de emplear el lenguaje C e investigar mucho sobre las herramientas de linux. Todo lo que es socket y cola de mensaje se entendió bien en la teoría, pero en la práctica incurre a grandes tropiezos. El manejo de errores y trabajar con variables ha demostrado ser la parte más conflictiva, teniendo constantemente errores que trabajar, y es donde se va la mayor parte del tiempo.

Bibliografía

[1] Uso de poll()
https://www.ibm.com/docs/en/i/7.4?topic=designs-using-poll-instead-select
[2] Listas en C http://www.pedrogonzalezruiz.net/listas/listas.html
[3] Uso de MD5
https://stackoverflow.com/questions/7627723/how-to-create-a-md5-hash-of-a-string-in-c