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:

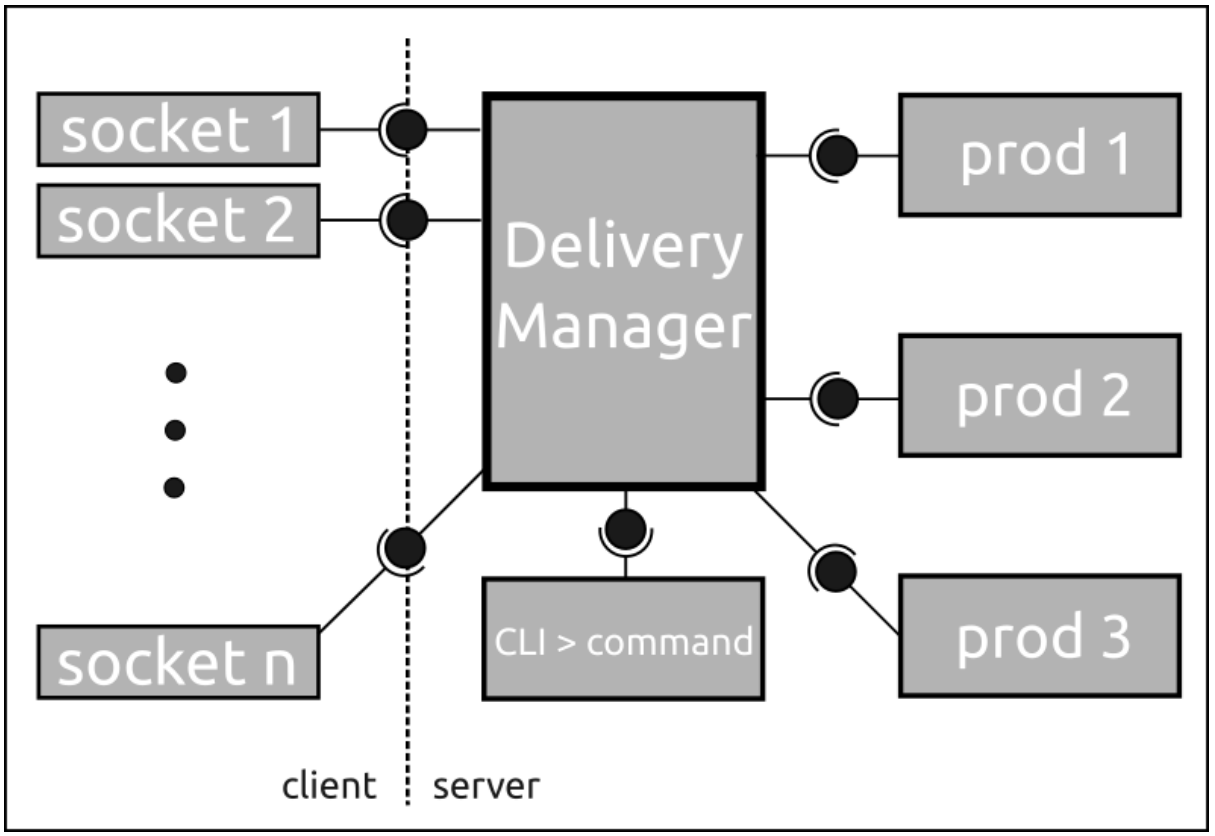

Figura 1

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.

Requerimientos
Para correr el programa, se requiere que la computadora sea un entorno de linux, con lalibrería libzip instalada.

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 productor 2 se encarga de decir cuanta memoria libre queda, por lo que este lee ese dato del archivo */proc/meminfo* de la segunda linea, lo divide por 1024 obteniendo así el valor en Mb y lo envía.
El productor 3 se encarga de enviar la carga del sistema normalizada. Para obtener la carga, se emplea la función **getloadavg()**, pero para normalizar, debemos dividirla por la cantidad de cores de la pc, y esto se saca de */proc/cpuinfo* en la línea 12. dividiendo, obtenemos el valor y lo enviamos.

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**.
Clientes emplea nodos como se muestra en la figura 1.

```
/*
    Estructura de los nodos de la lista
*/
struct lista { /* lista simple enlazada */
  int32_t fd;
  char* ip;
  int32_t port;
  int subs_1;
  int subs_2;
  int subs_3;
  int desconectado;
  struct lista *sig;
};
/*
```

Que guardan el file descriptor, la ip y puerto, para tener el socket del cliente. Además, tiene para colocar en que lista de suscripción está (subs_!, subs_2 y/o subs_3), si se ha desconectado, y el puntero al nodo siguiente.
Luego, cuando recibe el comando add, por socket (ip:puerto) se irán modificando, dentro de la estructura, los valores de subs_1, subs_2 o subs_3, según corresponda.
Cuando llegue un mensaje de un productor, este se loguea y luego recorre su lista de suscripción, enviandoselo a cada uno que tenga el campo en 1.
Para eliminar un cliente de una lista de suscripción, se recorre la lista de suscriptores 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 evaluar los clientes desconectados por 5 segundos, se plantea revisar la lista de **clientes_conectados** que es revisada cuando un cliente se conecta, para ver si estaba antes. Si no se encuentra, es agregado a la lista de manera normal. Por el

contrario, si se encuentra que ya estaba desconectado, es que se está conectando de nuevo, por lo que se le cambia el file descriptor y se le envía todos los mensajes encolados. Cada 5 segundos el código se fija si todos los clientes están conectados, cuando alguno no responde, se le modifica el campo desconectado por 1, y si a los 5 segundos no hubo conexión, esto es cuando revise de nuevo, el cliente es eliminado de la lista de **clientes_desconectados**.

A medida que vamos agregando clientes a la lista de distribución, el Delivery Manager podrá empezar a enviar cosas. Recorre la lista constantemente, y cuando ve que los campos de un suscripto tienen el flag de ese productor, envía el mensaje y loguea este, con su timestamp.
Para lograr el timestamp, se empleó la biblioteca time, con la función localtime. Esto provee ya una estructura que podemos trabajar y obtener año, mes, día, hora, minutos y segundos.

En cuanto al comando log, toma la carpeta donde se encuentran todos los logs, se comprime y se envía al cliente que le especifica.
Para comprimir, se empleó la librería suministrada en clase, y siguiendo las instrucciones de compilación que brindó un compañero, libzip, por lo que el código tiene la biblioteca *zip.h* y se compila con *-lzip*[4].

Durante el proceso, hubo problemas en la cuestión de validar por hash, ya que no coincidían los valores calculados por el server y por el cliente. Ese fue el problema más engorroso para el cliente.

Luego, en el server hubo problemas por un malentendido a la hora de diseñar el cierre de sesión. Se entendió que el servidor cerraba sesiones al cerrarse, y evitaba que quedan conectados los clientes, cosa que no es así.
Debido a esto, se agregó en el handler más detalles, este ya cerraba la cola de mensajes al cerrarse, pero además se agregó que recorra la lista de clientes conectados y los cerrará uno por uno. Así mismo, se le agregó un handler al cliente para que también cierre sesión.

Aún quedan algunas situaciones sobre manejo de errores en C que no están claras como el hecho de que al producirse un core por algún error en el código, se generen 4 instancias de proceso DeliveryManager corriendo en el mismo puerto, que hasta no ser eliminadas todas, este no puede funcionar correctamente.

Conclusión

Este trabajo 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. También se indaga mucho en la investigación y el uso de códigos y diseños sacados de internet, y pensar cómo hacer un diseño que todo funcione, siendo en momentos que parece que la solución que diseñamos es la correcta, pero al momento de implementarla, los fallos al no tener muy en claro el lenguaje C pasan factura.

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
[4] https://github.com/kuba--/zip