Pontificia Universidad Javeriana

Facultad de Ingeniería

Proyecto patrón Publicador/Suscriptor

Estudiantes

Carlos David Mercado Gallego

Juan David Sánchez Alfonso

Juan Diego Bermude Jimenez

Juan Diego Arias Durán

Sistemas Operativos

Osberth De Castro Cuevas

Noviembre 5 del 2024

Desarrollo Proyecto

I. CONTEXTO

El proyecto se basa en el desarrollo de un sistema de comunicación de noticias utilizando el patrón de diseño Publicador/Suscriptor. Este patrón permite que diferentes procesos actúen como publicadores y suscriptores de noticias, enviando y recibiendo mensajes mediante un sistema de comunicación central. Los publicadores generan noticias y las envían sin conocer la identidad de los receptores, mientras que los suscriptores se suscriben a temas específicos de interés y reciben notificaciones solo cuando se publica contenido relevante para ellos. Este modelo es similar al de redes sociales como X/Twitter, donde los usuarios siguen temas o cuentas específicas y reciben actualizaciones basadas en sus preferencias.

Este proyecto propone la implementación de un sistema de noticias utilizando el modelo de Publicador/Suscriptor, con el objetivo de gestionar la transmisión y recepción de noticias entre procesos independientes. Este sistema de comunicación se caracteriza por permitir que los publicadores envíen noticias de diversas categorías (como arte, ciencia, política, etc.) sin necesidad de conocer los receptores finales, mientras que los suscriptores pueden suscribirse a temas específicos de interés y recibir únicamente aquellas noticias relevantes para ellos. La estructura del proyecto incluye tres actores principales: los publicadores, los suscriptores y un sistema de comunicación central que coordina la transmisión, asegurando que cada noticia llegue a los suscriptores adecuados en función de sus intereses.

A. Publicador.

En el contexto del proyecto, el publicador es el proceso encargado de enviar noticias al sistema de comunicación para que sean distribuidas a los suscriptores interesados. Los publicadores no tienen conocimiento directo de quiénes recibirán sus noticias, pues su función es simplemente generar y transmitir información. Cada publicador se invoca mediante un comando en la terminal, donde se especifican los argumentos necesarios:

- -p: pipe de comunicación “pipePSC”.
- -f: nombre del archivo que contiene las noticias.
- -t: intervalo de tiempo entre cada envío de noticias.

El publicador utiliza el pipe nominal pipePSC para enviar sus noticias al sistema de comunicación. Este pipe actúa como un canal unidireccional por donde el publicador deposita sus mensajes, que posteriormente serán procesados y distribuidos a los suscriptores pertinentes. El archivo de noticias que utiliza cada publicador debe cumplir con un formato específico: cada línea del archivo representa una noticia, comenzando con una letra mayúscula que identifica la categoría de la noticia (por ejemplo, "A" para Arte, "C" para Ciencia, etc.), seguida de dos puntos y el contenido del mensaje. Si alguna línea no respeta este formato, el publicador debe mostrar un mensaje de error antes de iniciar el envío, asegurando así que solo las noticias válidas ingresen al sistema.

Además, el parámetro timeN define el intervalo de tiempo en segundos entre el envío de cada noticia, permitiendo una periodicidad en la transmisión que contribuye a la organización y sincronización en el flujo de información. Durante el funcionamiento del sistema, es posible crear nuevos publicadores en cualquier momento, aumentando la flexibilidad y escalabilidad del proyecto. Cada publicador finaliza su operación una vez que ha transmitido todas las noticias de su archivo y ha cumplido con los tiempos establecidos, momento en el cual el sistema de comunicación continuará su trabajo hasta que se complete la recepción y distribución de todas las noticias

B. Suscriptor.

El suscriptor en este proyecto representa el proceso que se suscribe a ciertos tópicos de interés para recibir únicamente las noticias relevantes a sus preferencias. Cada suscriptor es invocado desde la terminal con la bandera -s seguido del nombre del nombre de pipe, que establece el canal de comunicación con el sistema de comunicación. Una vez en ejecución, el suscriptor solicita al usuario que elija los tópicos o categorías a las cuales desea suscribirse, tales como Arte, Ciencia, Política, Farándula y Espectáculos, o Sucesos. Esta elección permite que el suscriptor reciba solo las noticias que concuerdan con los temas seleccionados, manteniendo así la pertinencia de la información que recibe.

Una vez que el suscriptor ha indicado sus tópicos de interés, envía esta información al sistema de comunicación, que se encarga de almacenar la suscripción y gestionar el flujo de noticias según las preferencias de cada suscriptor. A partir de este momento, el suscriptor permanece en un estado de espera, listo para recibir las noticias filtradas y enviadas por el sistema de comunicación cuando algún publicador publique una noticia que corresponda a sus tópicos seleccionados. Las noticias recibidas se muestran en la pantalla de manera inmediata, ofreciendo al suscriptor un acceso en tiempo real a la información de su interés.

El sistema permite agregar o levantar suscriptores en cualquier momento durante la ejecución, lo que añade flexibilidad al sistema y permite una gestión dinámica de los usuarios interesados en recibir noticias. La conexión del suscriptor se mantiene activa hasta que el sistema de comunicación envía una notificación final indicando que se ha completado la transmisión de noticias. Este mecanismo asegura que los suscriptores reciban toda la información de su interés y se desconecten de forma ordenada al concluir el ciclo de comunicación

C. Sistema de comunicación.

El sistema de comunicación (SC) es el componente central en el modelo de Publicador/Suscriptor implementado en este proyecto. Su principal responsabilidad es gestionar la transmisión de noticias entre publicadores y suscriptores de manera eficiente y precisa. Al ejecutar el sistema, se reciben 3 banderas:

- -s: Nombre del pipe por el que se comunicaran los suscriptores (pipeSSC)
- -p: Nombre del pipe por el que se comunicaran los publicadores (pipePSC)

- -t: Tiempo de espera (timeF)

El SC actúa como un intermediario, recibiendo las noticias enviadas por los publicadores a través del pipe pipePSC y filtrando su distribución de acuerdo con las suscripciones activas de los suscriptores, quienes están conectados mediante el pipe pipeSSC. Este diseño garantiza que cada noticia llegue exclusivamente a aquellos suscriptores que hayan manifestado interés en el tópico específico de la noticia.

Al iniciar su ejecución, el sistema de comunicación registra tanto las noticias recibidas de los publicadores como las suscripciones de los suscriptores. Cuando un publicador envía una noticia, el SC analiza la categoría de la noticia y verifica cuáles suscriptores están interesados en recibirla. Si hay coincidencia entre los intereses de los suscriptores y el tema de la noticia, el SC envía la noticia a los suscriptores pertinentes a través del pipe correspondiente. Este proceso permite una distribución precisa y evita la transmisión de noticias irrelevantes a suscriptores que no han manifestado interés en ciertos tópicos.

El SC también cuenta con un parámetro de tiempo, timeF, que define el período de espera tras finalizar el envío de noticias por parte de todos los publicadores. Este tiempo adicional permite que el sistema se mantenga activo por si se crean nuevos publicadores durante el intervalo especificado. Al concluir este tiempo, el SC envía una señal de finalización a todos los suscriptores, indicando que el ciclo de noticias ha terminado y que pueden cerrar su conexión. Con este mecanismo, el sistema de comunicación asegura que todos los procesos se completen de manera ordenada, finalizando su operación sólo cuando ya no queda información por distribuir en el sistema

II. DISEÑO GENERAL.

Para desarrollar la solución al problema propuesto es necesario realizar un diseño previo, puesto que se requiere la toma de decisiones de funcionamiento general que no están explícitamente estipuladas en el contexto del problema, por lo que se tiene la flexibilidad de escoger la solución más adecuada para el problema.

Primero que todo, se toma la decision de buena practica, de manejar todos los pipes nominales en el directorio “/tmp”, esto no afecta en nada el uso de los programas, puesto que serán los programas los que automáticamente agregue este prefijo al nombre brindado por las banderas de entrada.

A. Comunicación publicadores - sistema

Esta es la comunicación más sencilla de diseñar, puesto que no requiere de ningún cambio o adición sobre lo estipulado en el contexto del problema, simplemente se usara 1 solo pipe nominal cuyo nombre es dado a la hora de ejecutar el sistema y que sera el mismo que deben usar los publicadores, este pipe nominal será abierto en modo de escritura por parte de los publicadores, y en modo de lectura por parte

del sistema, por lo que es el sistema el que crea el pipe y los publicadores escriben en él, por eso mismo los publicadores no pueden ser ejecutados antes de ejecutar el sistema.

En cuanto a la información adicional que se debe tener en cuenta, esta que el sistema debe ser capaz de saber cuando hayan publicadores activos y cuando no, para esto se debe llevar algún tipo de control de los publicadores, puesto que el sistema debe detectar cuando no hayan publicadores activos para iniciar el conteo regresivo antes de finalizar la ejecución. Para llevar este control, dado que no se requiere información específica de los publicadores más que simplemente si están activos o no, se diseñó la siguiente solución.

1. El publicador antes de comenzar a enviar sus noticias, envía un primer mensaje por el pipe, este mensaje solo dice “abierto”, es un mensaje para indicar al sistema que un nuevo publicador se ha agregado.
    a. El sistema al recibir un mensaje en el pipe de los publicadores de tipo “abierto” aumenta en 1 una variable que lleva control de los publicadores activos.
2. El publicador envía todas sus noticias, con un intervalo de espera entre cada una de los segundos indicados en la bandera.
    a. El sistema recibe esos mensajes normales de noticias, y los procesa dirigiendolos a los suscriptores.
3. Una vez el publicador ha terminado de enviar todas sus noticias, envía un mensaje final que dice “cerrado”, indicando al sistema que ha terminado su trabajo de publicador.
    a. El sistema al recibir un mensaje en el pipe de publicadores que diga “cerrado” restará en 1 a la variable que lleva conteo de los publicadores activos.
4. Una vez la variable que lleva el conteo de los publicadores activos llegue a 0, el sistema sabe que ya no hay publicadores activos, por lo que inicia el conteo regresivo.
5. Al finalizar el conteo regresivo, si la variable de número de publicadores activos sigue siendo 0, entonces se finaliza la ejecución, si es diferente de 0, quiere decir que se unieron nuevos publicadores, por lo que no finaliza.

De esta forma el sistema no debe tener memoria de cada publicador en específico, basta con saber simplemente el número de publicadores para identificar cuando ya no hayan publicadores activos, y esto se logra tan solo con los mensajes de apertura y cierre enviados por los publicadores.

En el paso 5 del flujo explicado, es importante resaltar que, como se verá más adelante en la implementación, cada procesamiento de un mensaje se debe realizar en un thread separado, es decir que el thread que detecta que la variable llegó a 0, es el thread que espera el conteo regresivo, y al finalizar el conteo es posible que la variable ya no este en 0, pues puede que otro thread haya recibido un nuevo mensaje y haya aumentado el valor de la variable, es fundamental que se maneje así, en threads separados, para lograr el objetivo de que un thread sea el que haga

conteo pero el sistema pueda seguir recibiendo nuevos mensajes, en cuyo caso aumentan el valor de la variable.

El thread que realiza el conteo regresivo debe ser capaz de finalizar la ejecución de manera satisfactoria, esto implica que, este debe tener acceso a todos los pipes usados para poder cerrarlos antes de finalizar la ejecución, la forma en la que se logra esto es por medio de tener los descriptores de los pipes como variables globales, en la sección III se explicará más sobre la implementación.

B. Comunicación suscriptores- sistema

Esta comunicación es compleja y requiere de la toma de varias decisiones para el correcto funcionamiento de la misma. El nombre del pipe nominal de los suscriptores es dado a la hora de ejecutar el sistema, y debe coincidir con el nombre del pipe dado a los suscriptores, sin embargo, este pipe por sí solo no puede ser usado para todos los casos, es decir, no puede ser el pipe que todos los suscriptores usen para recibir las noticias, puesto que antes de esto es necesario realizar una suscripción en el sistema es decir, se requieren los siguientes pasos.

1. El suscriptor debe indicar a qué categorías de noticias quiere suscribirse.
2. El sistema registra al suscriptor.
3. Cuando se recibe una noticia, el sistema verifica si es de una categoría a la que el suscriptor esté suscrito, solo de ser así se envía la noticia.
4. Una vez el sistema detecta que ya no hay publicadores y ha pasado el tiempo límite entonces le indica al suscriptor que ya no habrán más noticias y finaliza.

Para conseguir esto, se evidencio que dado que se requiere un handshake entre el sistema y el suscriptor, en donde tanto el suscriptor como el sistema deben escribir y leer del pipe, entonces para tener mayor seguridad y no tener pipes de tipo escritura y lectura, se decide tener 2 pipes distintos, un pipe que sea solo para la solicitud de los suscriptores, es decir, en este pipe los suscriptores son escritores y el sistema es lector, y un segundo pipe que es solo para las respuestas a las solicitudes, en este pipe el sistema será escritor y los suscriptores lectores.

Dado que solo se tiene un nombre de pipe, que es el nombre que se ingresa al sistema y los suscriptores, se decide que no se tomará dicho nombre para 1 solo pipe, sino que se usará como nombre base para los 2 pipes que se crearan, al pipe de solicitudes se le agregara el posfijo “solicitud”, y al pipe de respuesta se le agregara el posfijo “respuesta”. respetando esta notación tanto en los suscriptores como en el sistema se consigue que ambos se comuniquen por los mismos pipes.

Tomando como ejemplo la bandera “-s pipeSuscrib”, entonces tanto el sistema como los suscriptores agregaran el prefijo y postfijo

- Pipe de solicitud: “/tmp/” + “pipeSuscrib” + “solicitud” = “/tmp/pipeSuscribsolicitud”.

- Pipe de respuesta: “/tmp/” + “pipeSuscrib” + “respuesta” = “/tmp/pipeSuscribrespiuesta”

Con esto se logra diseñar una solución funcional al handshake.

1. El suscriptor envía una solicitud por el pipe de solicitud.
2. El sistema recibe la solicitud por el pipe de solicitud.
3. El sistema envía la respuesta por el pipe de respuesta.
4. El suscriptor escucha y recibe la respuesta en el pipe de respuesta.

Una vez establecido el handshake, ahora el siguiente tema es, cómo y qué contenido tienen esos mensajes de solicitud y respuesta.

- El mensaje de solicitud enviado por el suscriptor debe contener su PID seguido de las letras relacionadas a las categorías a las que el suscriptor se quiere suscribir. La razón para incluir el PID es para que el sistema pueda diferenciar los múltiples suscriptores que se tienen, es el identificador único del suscriptor. El suscriptor debe tener al menos 1 letra acompañando al PID, puesto que el suscriptor debe suscribirse al menos a 1 categoría. El sistema podrá obtener el identificador único y las categorías siguiendo la deserializacion de esta estructura, es decir, el PID son todos los dígitos del inicio, números, hasta encontrar un carácter alfabético, al encontrar un caracter alfabetico, esas seran las llaves de las categorías a las que se suscribe,
    - EJEMPLO 1: PID = 12345, Categorías = “A”, “E”, “S”, mensaje solicitud = “12345AES”
    - EJEMPLO 2: PID =2568, Categorías = “P”, mensaje solicitud = “2568P”.

- El mensaje de respuesta a la solicitud será el PID del suscriptor al que se le está contestando, seguido del nombre del pipe nominal creado para ese suscriptor, en el que el suscriptor escucha las noticias para él. El PID y el nombre del pipe estarán separados por un espacio.
    - EJEMPLO 1: PID = 12345, Pipe creado = “/tmp/suscriptor12345”, mensaje respuesta = “12345 /tmp/suscriptor12345”.

Como se puede ver, el nombre del pipe de cada suscriptor debe ser único, y para asegurar que asi lo sea, se genera el nombre a partir de la base “/tmp/suscriptor” y se le adiciona el PID de ese suscriptor al nombre, asegurando así que sea totalmente único.

Luego de que el suscriptor envía la solicitud, como ya se mencionó, este entrar en un modo de lectura del pipe de respuesta, por donde sabe que le llegara la respuesta a su solicitud con el pipe creado, sin embargo, este pipe de respuesta se comparte con todos los demás suscriptores en espera de sus pipes, por lo que, para asegurar que los suscriptores solo reaccionen al que iba destinado para ellos, es que se envía el PID al inicio del mensaje, con esto los suscriptores verifican si el mensaje es para ellos o si deben ignorarlo y seguir esperando.

Si bien el nombre de los pipes creados siguen una estructura establecida, se decide que es mejor realizar este handshake y que sea el sistema el que le envie al suscriptor el nombre de su pipe, y no que el suscriptor lo dé por establecido como "/tmp/suscriptor${PID}", puesto que se quiere mantener el control de los pipes enteramente del lado del sistema.

Como se menciona, el sistema creará un pipe por cada suscriptor, y es por ese pipe por el que el suscriptor escucha las noticias y el sistema las enviara, es decir que el siguiente paso luego del handshake entre el sistema y el suscriptor, es que el suscriptor entre en un estado de lectura de ese pipe recibido, mientras que el sistema enviará a ese pipe todas las noticias que apliquen.

Luego cuando se entra en modo de lectura en el pipe personal, el suscriptor simplemente esperará a recibir noticias y las mostrará, no debe hacer filtrado ni nada, pues es el sistema el que se encarga de solo enviar las noticias que ese suscriptor espera, sin embargo, se tiene un formato especial para comunicar la finalización de la comunicación, una vez el sistema determine que ya no van a haber más publicadores, este envía un mensaje indicando eso, pero lo importante de este mensaje es que debe estar iniciado por "0:", por ejemplo:

- "0: Se han acabado las noticias".

De esta forma, los suscriptores siempre verifican con cada mensaje si este inicia por "0:", y cuando encuentren uno que sí lo haga, esa es la indicación de finalización, por lo que el suscriptor sólo muestra el mensaje, y finaliza su ejecución.

Con lo anterior, se puede suponer que entonces el sistema requiere de alguna estructura de control para cada uno de los suscriptores, la información esencial para cada suscriptor es.

1. PID del suscriptor.
2. Categorías a las que está suscrito.
3. Nombre del pipe generado.
4. Descriptor del pipe generado.

Es así que se decide diseñar una estructura que contenga esos campos de interés, y el sistema tendrá un arreglo de 100 suscriptores, es decir que la capacidad máxima es de 100 suscriptores.

```
struct Suscriptor {
    int pid;
    char categorias[5];
    char* nombre_pipe;
    int fd_pipe;
};
```

La forma en la que se estructura la información es la siguiente.

- pid: Contiene el número del PID del suscriptor.
- categorías: Es un arreglo de 5 caracteres, en principio todos los caracteres son '\0', y se llenan de izquierda a derecha las letras de las categorías a las que el suscriptor está suscrito, el tamaño del arreglo es de 5 porqué máximo puede estar suscrito a 5 noticias.
- nombre_pipe: Se guarda el nombre generado por si acaso, si bien para enviar los mensajes solo se requiere el descriptor, es bueno guardar también el nombre.
- fd_pipe: Es el descriptor del pipe abierto para el suscriptor, es fundamental que se guarde acá luego de crear y abrir el pipe, pues es el que se usará para enviar los mensajes.

Como ya se mencionó a la hora de explicar la comunicación Publicador-Sistema, cada mensaje recibido del Publicador con una noticia es manejado en un hilo separado, por lo que es necesario que todos estos hilos tengan acceso al arreglo de suscriptores, es por esto que dicho arreglo debe ser global, y así mismo, pasa a ser una zona crítica, pues no puede haber más de 1 hilo trabajando sobre el arreglo, porque si alguno de esos hilos está agregando un nuevo suscriptor y otro esta leyendo los suscriptores para enviar mensajes, entonces puede ocasionar consecuencias no esperadas.

En resumen, las decisiones de diseño general tomadas para el problema fueron las siguientes.

1. Sistema crea y abre el pipe de publicadores en modo de lectura.
2. Publicadores abren el pipe de publicadores en modo escritura.
3. El sistema crea un hilo por cada mensaje de publicadores recibidos.
4. Sistema lleva control de publicadores activos por medio de mensajes de saludo y despido del publicador, "abierto" indicando que se ha unido un nuevo publicador, y "cerrado" para indicar que un publicador ha acabado, sistema lleva control con una variable global (ZONA CRÍTICA), que lleva el conteo de los publicadores, si alcanza a 0, el hilo que recibió ese mensaje esperará los segundos establecidos, y si pasados esos segundos, la variable sigue en 0, finaliza la ejecución cerrando todos los pipes y enviando el mensaje de finalización a los suscriptores.
5. Se realizará un handshake entre sistema y suscriptor, esto consiste en tener 2 pipes iniciales, que son el nombre del pipe de suscriptores indicado para el sistema y los suscriptores, añadiendo dos postfijos, "solicitud" para el de solicitudes de los suscriptores, y "respuesta" para las respuestas a esas solicitudes, es decir que el sistema abre "solicitud" en lectura y "respuesta" en escritura, y los suscriptores lo inverso.
6. El suscriptor debe enviar su PID junto con las categorías a las que quiere suscribirse.
7. El sistema genera un pipe único para cada suscriptor y lo abre en modo escritura, le envia como respuesta a los suscriptores en el pipe "respuesta" el

mensaje con el PID del suscriptor al que está contestando, junto con el nombre del pipe generado para él.

8. El sistema guarda la información del suscriptor, PID, categorías suscritas, nombre del pipe, y descriptor del pipe en una estructura en un arreglo global.
9. El suscriptor abre el pipe con el nombre recibido en la respuesta, lo abre en modo de lectura, y lee todas las noticias que sabe que son para él.
10. El suscriptor finaliza la ejecución una vez recibe un mensaje iniciando por "0:".
11. Cada vez que el sistema reciba una noticia, creará un nuevo hilo para esa noticia, ese hilo recorre los suscriptores, verifica en sus categorías suscritas, y si la categoría del mensaje está en las suscritas por el suscriptor, entonces envía el mensaje por el pipe del suscriptor.
12. Si el sistema determina que ya ha pasado el tiempo límite sin publicadores, entonces envía un mensaje a todos los suscriptores indicando la finalización, cierra todos los pipes y finaliza la ejecución.

Una vez tomadas las decisiones de diseño anteriormente descritas, se puede continuar con la implementación, en donde se detallan cosas más cercanas al código, tales como el manejo de las zonas críticas identificadas.

III. IMPLEMENTACIÓN.

En esta sección se detallarán las partes importantes de la implementación, no se detallaran cosas mínimas, sino solo partes fundamentales y la estructura general.

A. Publicador

Para el publicador no se requieren múltiples funciones, es el componente mas sencillo, por lo que todo se realiza en la función main, separado en distintos tiempos.

1. Validación de argumentos: Se valida que los argumentos ingresados (banderas) sean los necesarios.

```c
//validacion de argumentos
char* nombre_pipe = NULL;
char* nombre_archivo = NULL;
int seg_espera = -1;
const char *pipe_prefix = "/tmp/";
if(argc < 7){
    perror("Numero de argumentos invalido");
    exit(1);
}
for(int i = 1; i < 7; i++){
    if(strcmp(argsv[i], "-p") == 0){
        nombre_pipe = malloc(strlen(argsv[i+1])+ 6);
        strcpy(nombre_pipe, pipe_prefix);
        strcat(nombre_pipe, argsv[i+1]);
    }
    if(strcmp(argsv[i], "-f") == 0){
        nombre_archivo = argsv[i+1];
    }
    if(strcmp(argsv[i], "-t") == 0){
        seg_espera = atoi(argsv[i+1]);
    }
}
if(nombre_pipe == NULL || nombre_archivo == NULL || seg_espera == -1){
    perror("Faltan flags");
    exit(1);
}
segundos_esperar = seg_espera;
```

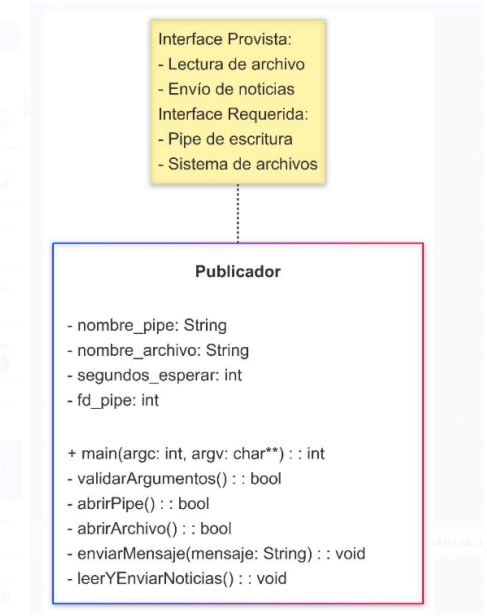
2. Se abre el archivo indicado en la bandera, se lee línea a línea verificando que cada línea siga los requerimientos de iniciar por una letra APESC seguido de ":", se guardan las lineas leidas, si alguna no cumple con el requerimiento, se muestra el mensaje y finaliza la ejecución.

```c
//leer el archivo linea a linea y verificar que el formato sea el correcto
while ((leido = getline(&linea, &longitud, archivo)) != -1) {
    //se verifica la letra inicial y los :
    if ((linea[0] == 'A' || linea[0] == 'P' || linea[0] == 'E' || linea[0] == 'S' || linea[0] == 'C') && linea[1] == ':' && strlen(linea) <= 83) {
        //Si la linea es valida, entonces se añade a las lineas validads
        lineas_validas[total_lineas] = malloc(strlen(linea) + 1);
        strcpy(lineas_validas[total_lineas], linea);
        total_lineas++;
    } else {
        //si la linea no es valida, entonces se muestra el error y finaliza la ejecucion
        perror("El archivo no cumple con el formato");
        free(linea);
        fclose(archivo);
        close(fd_pipe);
        exit(1);
    }
}
```

3. Una vez verificado el archivo y leídas todas las líneas, si todo está bien, entonces se envía el mensaje "abierto" por el pipe de publicadores.
4. Se itera sobre las líneas guardadas, se escriben las líneas por el pipe de publicadores, y luego de escribir cada línea se duerme la cantidad de segundos indicados por la bandera.
5. Luego de enviar todas las líneas, se envía el mensaje final "cerrado" y finaliza la ejecución.

```c
//enviar mensaje "abierto" por el pipe, esto indica al sistema que se unio un publicador
char* mensaje_abierto = "abierto";
write(fd_pipe, mensaje_abierto, strlen(mensaje_abierto) + 1);

//se envian todas las lineas por el pipe, por cada linea se duerme el tiempo estipulado por la flag
for (int i = 0; i < total_lineas; i++) {
    write(fd_pipe, lineas_validas[i], strlen(lineas_validas[i]) + 1);
    printf("\nNoticia enviada %s\n", lineas_validas[i]);
    sleep(segundos_esperar);
    free(lineas_validas[i]);
}

//luego de enviar todas las noticias se envia un mensaje de cerrado
// esto es para que el sepa que un publicador acaba de terminar, y pueda llevar control de si hay publicadores vivos
char* mensaje_cerrado = "cerrado";
write(fd_pipe, mensaje_cerrado, strlen(mensaje_cerrado) + 1);
```

Para el publicador no se requiere manejo de mutex ni concurrencia.



B. Suscriptor

El suscriptor requiere una implementación algo más compleja que el publicador, tiene las siguientes funciones:

1. ObtenerSuscripciones: Es la función encargada de recibir las suscripciones del usuario, realiza verificación de que la opción seleccionada sea valida, verifica que se seleccione minimo 1 categoria y maximo las 5 disponibles.
2. ObtenerPipe: Es la función que se encarga de leer del pipe de suscriptores, y cuando reciba un mensaje iniciado por su PID indicando que es para él, entonces lo procesa y extrae la cadena que indica el nombre del pipe.
3. EscucharSuscripcion: Es la función que dado el nombre del pipe obtenido por el sistema, lo abre y lee de él todas las noticias, si recibe el mensaje de finalización “0:”, entonces finaliza la ejecución.

Las funciones anteriormente descritas no tienen complejidad u operaciones avanzadas como manejo de mutex o zona crítica, solo manejan mensajes por los pipes dados, de hecho, todo el programa de suscriptor es single-threaded, no se requiere la creación de hilos puesto que todo puede realizarse secuencialmente.

La única función en la que se ahondara es en “ObtenerPipe” pues es la que escucha a los mensajes del pipe “respuesta”.

```c
//funcion que se encarga de esperar la respuesta del sistema a nuestra solicitud, el mensaje deberia incluir
//el nombe del pid creado para nosotros
void ObtenerPipe(char* pipe_resultado, char* nombre_pipe, int pid) {
    //se abre el pipe de respuesta de solicitudses
    int fd = open(nombre_pipe, O_RDONLY);
    if (fd == -1) {
        perror("Error al abrir el pipe de respuesta");
        return;
    }
    char buffer_entrada[200];
    //se lee indefinidamente los mensajes encontrados en el pipe
    while(1) {
        memset(buffer_entrada, 0, sizeof(buffer_entrada));
        ssize_t bytes_leidos = read(fd, buffer_entrada, sizeof(buffer_entrada) - 1);
        if (bytes_leidos > 0) {
            buffer_entrada[bytes_leidos] = '\0';
            int numero;
            char* endptr;
            numero = (int)strtol(buffer_entrada, &endptr, 10);
            //si el numero inicial del mensaje es nuestro PID, entonces esta si es nuestra respuesa
            //si no es, entonces era la de alguien mas y la ignoramos y seguimos esperando
            if(numero == pid) {
                printf("\nRespuesta recibida %s\n\n", buffer_entrada);
                char* pipe_start = strchr(endptr, '/');
                if (pipe_start != NULL) {
                    //guardamos en el pipe_resultado el nombre del pipe recibido por el que debemos escuchar
                    strncpy(pipe_resultado, pipe_start, strlen(pipe_start) + 1);
                }
                close(fd);
                break;
            }
        }
    }
}
```

La función abre el pipe de respuesta, si no se puede abrir entonces se indica el error y finaliza.
Luego se entra en un estado de escucha por ese pipe, cada vez que se recibe un mensaje se obtiene el número inicial del mensaje, es decir, el PID, y SOLO SI ese pid coincide con el PID del programa en ejecución, entonces es ahí que se sabe que es su respuesta y se procesa, el procesamiento consiste simplemente en obtener el nombre del pipe recibido y finalizar el ciclo y por ende la lectura del pipe respuesta.

La secuencia en la que se ejecutan las funciones anteriormente descritas es la siguiente.

1. Se validan los argumentos de entrada, en este caso solo el nombre del pipe suscriptor.

```c
//validacion de argumentos
char* nombre_pipe = NULL;
char noticias[] = {'x', 'x', 'x', 'x', 'x'};
if(argc < 3){
    perror("Numero de argumentos invalido");
    exit(1);
}
if(strcmp(argsv[1], "-s") != 0){
    perror("Flag no valida");
    exit(1);
}
nombre_pipe = argsv[2];
```

2. Se obtiene el nombre de los pipes de solicitud y respuesta, aplicando el prefijo “/tmp/”, y los postfijos “solicitud” y “respuesta”, se abre el pipe de solicitud.

```c
//el nomrbe ingresado por la flag solo es la base, puesto que se necesitan 2 pipes distintos
//uno de escritura y otro de lectura, tan solo para el handshake de las 2 partes
char nombre_pipe_solicitud[100];
snprintf(nombre_pipe_solicitud, 100, "/tmp/%ssolicitud", nombre_pipe);
char nombre_pipe_respuesta[100];
snprintf(nombre_pipe_respuesta, 100, "/tmp/%srespuesta", nombre_pipe);

//se abre el pipe de solicitud, se hace antes de si quiera mostrar el menu
//puesto que si no se abre con exito no tiene caso intentar
int fd_solicitud = open(nombre_pipe_solicitud, O_WRONLY);
if(fd_solicitud == -1){
    perror("Error al abrir el pipe de solicitud");
    exit(1);
}
```

3. Se imprime el menú y se llama a la función ObtenerSuscripciones que es la que recibe la información de la consola.

```c
printf("Escribe la letra de los topicos a suscribirse:\n");
printf("A: Noticia de Arte\n");
printf("E: Noticia de Farandula y Espectaculos\n");
printf("C: Noticia de Ciencia\n");
printf("P: Noticia de Politica\n");
printf("S: Noticia de Sucesos\n");
printf("0: YA NO MAS\n\n");

ObtenerSuscripciones(noticias);
```

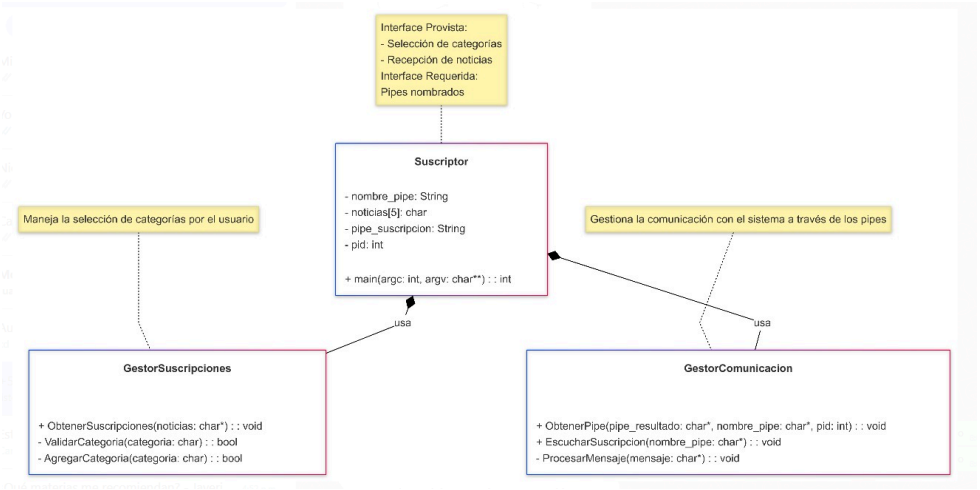
4. Se obtiene el PID del proceso actual y se genera la cadena de solicitud que contiene el PID y las categorías, y se escribe el mensaje por el pipe de solicitud.

```c
//se obtiene el pid del proceso, este sera el identificador unico o llave por la que se comunicaran
//y se envia el mensaje junto con las categorias a suscribirse
pid_t pid = getpid();
char* mensaje = malloc(64);
sprintf(mensaje, "%d", pid);
for (int i = 0; i < 5; i++) {
    if (noticias[i] != 'x') {
        strncat(mensaje, &noticias[i], 1);
    }
}
write(fd_solicitud, mensaje, strlen(mensaje) + 1);

close(fd_solicitud);
free(mensaje);
```

5. Se llama a ObtenerPípe que se encarga de escuchar por el pipe de respuesta hasta recibir el mensaje de respuesta para ese suscriptor y obtiene el nombre del pipe creado para él. Luego se llama a EscucharSuscripcion que escuchara en ese pipe todas las noticias hasta que reciba la noticia de finalización.

```
//ahora nos vamos a escuchar el pipe de respuesta
char pipe_suscripcion[100];
ObtenerPipe(pipe_suscripcion, nombre_pipe_respuesta, pid);

//Si llegamos aca implica que vamos bien, entonces vamos a escuchar el pipe de respuesta que
//nos envio el sistema para escuchar las noticias
EscucharSuscripcion(pipe_suscripcion);

return 0;
```

Con esto se completa la implementación del suscriptor que si bien está compuesta de más funciones, sigue teniendo una baja complejidad en donde las operaciones solo se basan en abrir pipes, verificar la correcta apertura, y escribir o recibir mensajes por los mismos.



C. Sistema

Este es el componente más complejo pues es el que maneja múltiples hilos, por lo que debe manejar zonas críticas, se explicara paso a paso.

```
struct Suscriptor {
    int pid;
    char categorias[5];
    char* nombre_pipe;
    int fd_pipe;
};

//todas estas variables globales son debido a la concurrencia del programa
//son variables que multiples threads deberan manejar
struct Suscriptor suscriptores[100]; //máximo 100 suscriptores
int total_suscriptores = 0; //control de cuantos suscriptores tenemos
int abiertos = 0;      //control de publicadores activos
int segundos_esperar = -1;    //segundos a esperar indicados por la flag

//decriptores de estos pipes son globales para que cualquier thread pueda cerrarlos
int fd_sus_solicitud;
int fd_sus_respuesta;
int fd_publicadores;

//semaforos para control de zonas criticas
sem_t mutex_abiertos;
sem_t mutex_noticias;
```

Primero que todo se tiene la estructura de Suscriptor, así como las variables globales a usar.

- suscriptores: Arreglo que contiene los suscriptores registrados.
- total_suscriptores: Mantiene control del número de suscriptores registrados en ese arreglo de hasta 100 suscriptores.
- abiertos: Mantiene control de publicadores activos.
- segundos_esperar : Es el número de segundos a esperar luego de que no hayan publicadores.
- pipes: los distintos pipes que se usarán, son globales para poder ser cerrados, es decir, estos son los descriptores que deben poder ser cerrados por cualquier hilo.
- mutex_abiertos: Semaforo para la protección de la zona crítica que maneja la variable “abiertos”.
- mutex_noticias: Semaforo para la protección de la zona crítica que altera “suscriptores” y “total_suscriptores”.

Ahora se comenzará con la descripción de la ejecución por partes.

```c
int main(int argc, char** argsv){
    //validacion de argumentos
    char* nombre_publicadores = NULL;
    char* nombre_suscriptores = NULL;
    const char *pipe_prefix = "/tmp/";
    if(argc < 7){
        perror("Numero de argumentos invalido");
        exit(1);
    }
    for(int i = 1; i < 7; i++){
        if(strcmp(argsv[i], "-p") == 0){
            nombre_publicadores = malloc(strlen(argsv[i+1])+ 6);
            strcpy(nombre_publicadores, pipe_prefix);
            strcat(nombre_publicadores, argsv[i+1]);
        }
        if(strcmp(argsv[i], "-s") == 0){
            nombre_suscriptores = malloc(strlen(argsv[i+1])+ 6);
            strcpy(nombre_suscriptores, pipe_prefix);
            strcat(nombre_suscriptores, argsv[i+1]);
        }
        if(strcmp(argsv[i], "-t") == 0){
            segundos_esperar = atoi(argsv[i+1]);
        }
    }
    if(nombre_publicadores == NULL || nombre_suscriptores == NULL || segundos_esperar == -1){
        perror("Faltan flags");
        exit(1);
    }
```

Primero que todo se validan las 3 posibles flags, pueden estar en cualquier orden, el programa es capaz de detectar esto, si alguna bandera no fue dada entonces se muestra el mensaje y finaliza.

```
//se inicializan los semadoros con 1 cupo inicial
sem_init(&mutex_abiertos, 0, 1);
sem_init(&mutex_noticias, 0, 1);

//se crea  el pipe de publicadores, solo es 1, por donde los publicadores enviaran sus noticias
unlink(nombre_publicadores);
mkfifo(nombre_publicadores, 0666);
```

Se inicializan los semáforos con un valor de 1 inicial, osea que hay un cupo, cabe resaltar que para el caso de este programa los semáforos se usarán para acceso individual, es decir que se usan como mutex normal. También se crea el pipe de publicadores, se usa la llamada unlink siempre antes de la "mkfifo" para eliminar el pipe si ya existía anteriormente y crear uno nuevo.

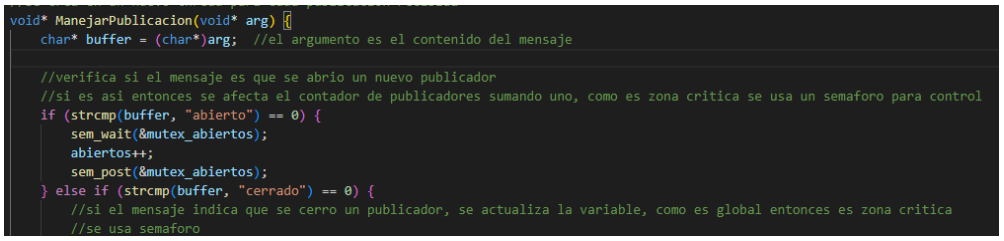
```
//creando hilo que handlea los publicadores
pthread_t  hilo_publicadores;
pthread_create(&hilo_publicadores, NULL, EscucharPublicadores, (void *)nombre_publicadores);

//creando hilo que handlea las solicitudes de  nuevos suscriptores
pthread_t  hilo_suscriptores;
pthread_create(&hilo_suscriptores, NULL, ManejarSuscriptores, (void *)nombre_suscriptores);

//join de threads para esperar a que los hilos terminen
pthread_join(hilo_publicadores, NULL);
pthread_join(hilo_suscriptores, NULL);

//liberar memoria
free(nombre_publicadores);
free(nombre_suscriptores);
return 0;
```

Se crean e inician 2 hilos iniciales, uno que maneja los publicadores, y otro que maneja las inscripciones (solicitudes de suscripciones), luego de esto se espera a que esos 2 hilos finalicen, para así no terminar la ejecución.

```
//funcion que escucha a los publicadores, es decir, escucha las publicaciones
//que estos envian
void* EscucharPublicadores(void* arg) {
    char* nombre_pipe = (char*)arg;
    fd_publicadores =  open(nombre_pipe, O_RDONLY);
    if(fd_publicadores == -1){
        perror("Error abriendo pipe de publicadores");
        exit(1);
    }
    char buffer_entrada[500];

    //se lee indefinidamente
    while (1) {
        memset(buffer_entrada, 0, sizeof(buffer_entrada));
        ssize_t bytes_leidos = read(fd_publicadores, buffer_entrada, sizeof(buffer_entrada) - 1);
        //cuando se recibe un mensaje simplemente se crea un thread para manejar esa publicacion
        //esto es porque el proceso de enviar las publicaciones puede tardar tiempo, pero no
        //queremos dejar de escuchar a las nuevas publicaciones
        if (bytes_leidos > 0) {
            buffer_entrada[bytes_leidos] = '\0';
            //IMPORTANTE, se crea una copia del contenido y es esa la que se le envia al thread
            //para poder seguir recibiendo mensajes y no afectar los threads manejando publicaciones
            char* buffer_copia = malloc(bytes_leidos + 1);
            strncpy(buffer_copia, buffer_entrada, bytes_leidos + 1);
            pthread_t hilo_publicacion;
            pthread_create(&hilo_publicacion, NULL, ManejarPublicacion, (void*)buffer_copia);
        }
    }
    close(fd_publicadores);
    return NULL;
}
```

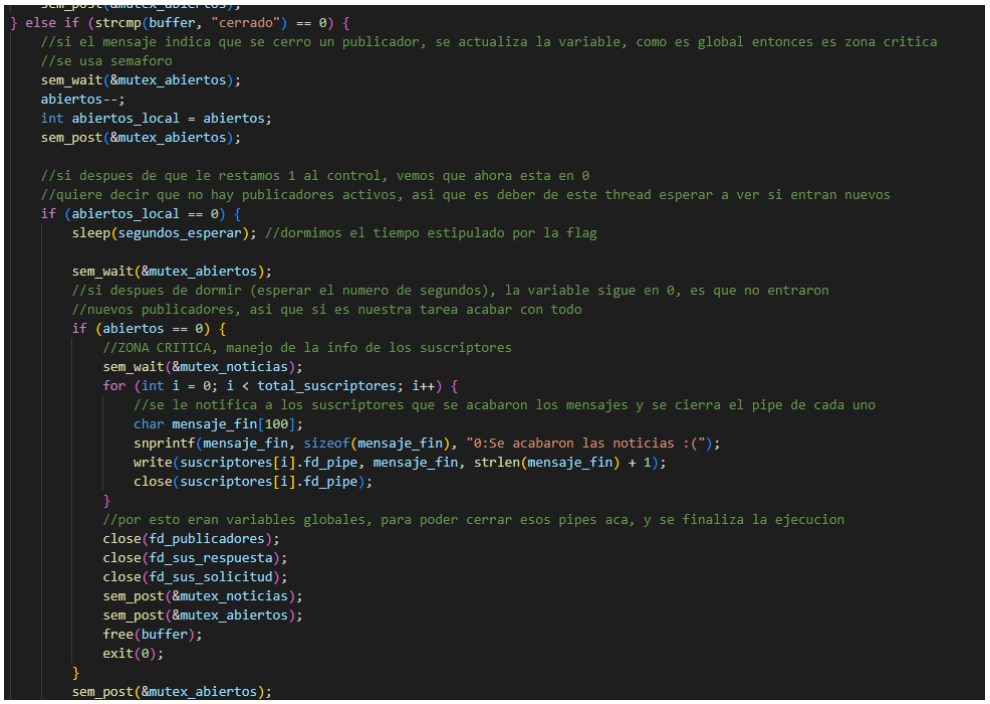
La función “EscucharPublicadores” es la que se ejecuta en uno de los dos hilos creados anteriormente, esta función abre el pipe de publicadores, en caso de no poder abrirlo informa el error y finaliza la ejecución, luego escucha indefinidamente por ese pipe, y cada vez que recibe un mensaje CREA UNA COPIA del mensaje y ejecuta un nuevo thread con la función “ManejarPublicacion”, a la que le pasa esa copia del mensaje, la idea de crear una copia es para que no afecte la sobreescritura del buffer si se recibe otro mensaje.

La función ManejarPublicacion es algo compleja, por lo que se explicara por partes.



```c
void* ManejarPublicacion(void* arg) {
    char* buffer = (char*)arg;  //el argumento es el contenido del mensaje

    //verifica si el mensaje es que se abrio un nuevo publicador
    //si es asi entonces se afecta el contador de publicadores sumando uno, como es zona critica se usa un semaforo para control
    if (strcmp(buffer, "abierto") == 0) {
        sem_wait(&mutex_abiertos);
        abiertos++;
        sem_post(&mutex_abiertos);
    } else if (strcmp(buffer, "cerrado") == 0) {
        //si el mensaje indica que se cerro un publicador, se actualiza la variable, como es global entonces es zona critica
        //se usa semaforo
```

Primero se revisa el mensaje, si el mensaje es “abierto” entonces se entra en zona crítica. SE TOMA UN CUPO DEL SEMÁFORO “mutex_abiertos”, se modifica el valor de la variable abiertos sumando 1, y luego se devuelve el cupo del semáforo.



```c
} else if (strcmp(buffer, "cerrado") == 0) {
    //si el mensaje indica que se cerro un publicador, se actualiza la variable, como es global entonces es zona critica
    //se usa semaforo
    sem_wait(&mutex_abiertos);
    abiertos--;
    int abiertos_local = abiertos;
    sem_post(&mutex_abiertos);

    //si despues de que le restamos 1 al control, vemos que ahora esta en 0
    //quiere decir que no hay publicadores activos, asi que es deber de este thread esperar a ver si entran nuevos
    if (abiertos_local == 0) {
        sleep(segundos_esperar); //dormimos el tiempo estipulado por la flag

        sem_wait(&mutex_abiertos);
        //si despues de dormir (esperar el numero de segundos), la variable sigue en 0, es que no entraron
        //nuevos publicadores, asi que si es nuestra tarea acabar con todo
        if (abiertos == 0) {
            //ZONA CRITICA, manejo de la info de los suscriptores
            sem_wait(&mutex_noticias);
            for (int i = 0; i < total_suscriptores; i++) {
                //se le notifica a los suscriptores que se acabaron los mensajes y se cierra el pipe de cada uno
                char mensaje_fin[100];
                snprintf(mensaje_fin, sizeof(mensaje_fin), "0:Se acabaron las noticias :(");
                write(suscriptores[i].fd_pipe, mensaje_fin, strlen(mensaje_fin) + 1);
                close(suscriptores[i].fd_pipe);
            }
            //por esto eran variables globales, para poder cerrar esos pipes aca, y se finaliza la ejecucion
            close(fd_publicadores);
            close(fd_sus_respuesta);
            close(fd_sus_solicitud);
            sem_post(&mutex_noticias);
            sem_post(&mutex_abiertos);
            free(buffer);
            exit(0);
        }
        sem_post(&mutex_abiertos);
```

Si el mensaje recibido es igual a “cerrado”, entonces SE TOMA UN CUPO DEL SEMÁFORO “mutex_abiertos”, se le resta 1 a la variable abiertos, y se devuelve el cupo del semáforo.

Si luego de restarle 1 a la variable, se ve que la variable ahora es 0, es porque se quedó sin publicadores, por lo que el hilo duerme los n segundos indicados por la bandera. Luego de dormir esos segundos vuelve a verificar si la variable sigue

siendo 0, en caso de si serlo, es porque en ese transcurso no hubo nuevos publicadores, así que se inicia el proceso de finalización.

En el proceso de finalización SE TOMA UN CUPO del semáforo “mutex_noticias” para evitar que otro hilo toque los suscriptores, y se recorren todos los suscriptores registrados, a cada uno se le envía el mensaje indicando la finalización, y se cierra el pipe de cada suscriptor. Luego de recorrer todos los suscriptores se cierran los pipes globales (por esto es que se necesitaba que fueran globales, para que cualquier hilo que le tocara esta tarea pudiera hacerla) y finaliza la ejecución completa del programa.

```
} else {
    //si no era ni abierto ni cerrado entonces era un mensaje normal, por lo que se maneja normal
    char llave = buffer[0];

    //ZONA CRITICA, así que se usa el semaforo para prevenir afectacion
    sem_wait(&mutex_noticias);
    for (int i = 0; i < total_suscriptores; i++) {
        for (int j = 0; j < 5; j++) {
            //si el usuario esta suscritor a la categoria del mensaje entonces se le envia la noticia a su pipe personal
            if (suscriptores[i].categorias[j] == llave) {
                write(suscriptores[i].fd_pipe, buffer, strlen(buffer) + 1);
                printf("Enviando noticia a suscriptor %d: %s\n", suscriptores[i].pid, buffer);
                break;
            }
        }
    }
    sem_post(&mutex_noticias);  //FIN DE ZONA CRITICA
}
free(buffer);
return NULL;
```

Si el mensaje no era ni “abierto”, ni “cerrado”, entonces era un mensaje de noticia normal, por lo que SE TOMA UN CUPO del semáforo “mutex_noticias”, se recorren los suscriptores, por cada suscriptor se recorren sus suscripciones para ver si el suscriptor estaba suscrito a esa categoría de mensaje recibida, y si si lo esta, entonces se le escribe en su pipe la noticia. Al recorrer todos los suscriptores se devuelve el cupo del semáforo “mutex_noticias”.

Con lo anterior se finaliza la parte de manejar los publicadores y las noticias, solo queda uno de los dos hilos creados inicialmente, el que maneja los suscriptores.

```c
//funcion que se encarga de manejar a los nuevos suscriptores, es decir, solicitudes de suscripcion
void* ManejarSuscriptores(void* arg){
    //con el nombre base del pipe recibido por la flag, se crean y abren los pipes
    char* nombre_pipe_base = (char*)arg;
    char nombre_pipe_solicitud[100];
    snprintf(nombre_pipe_solicitud, 100, "%ssolicitud", nombre_pipe_base);
    char nombre_pipe_respuesta[100];
    snprintf(nombre_pipe_respuesta, 100, "%srespuesta", nombre_pipe_base);

    unlink(nombre_pipe_solicitud);
    if (mkfifo(nombre_pipe_solicitud, 0666) == -1) {
        perror("Error creando el pipe de solicitud");
        exit(1);
    }

    unlink(nombre_pipe_respuesta);
    if (mkfifo(nombre_pipe_respuesta, 0666) == -1) {
        perror("Error creando el pipe de respuesta");
        exit(1);
    }
    fd_sus_solicitud = open(nombre_pipe_solicitud, O_RDONLY);
    if(fd_sus_solicitud == -1){
        perror("Error abriendo pipe de solicitud");
        exit(1);
    }
    fd_sus_respuesta = open(nombre_pipe_respuesta, O_WRONLY);
    if(fd_sus_respuesta == -1){
        perror("Error abriendo pipe de respuesta");
        exit(1);
    }
```

Primero que todo, la función “ManejarSuscriptores” crea y abre los pipes de suscriptores, tanto la solicitud como la respuesta. Si en alguno de esos pipes hay un error se indica y finaliza la ejecución.

```c
//se escucha indefinidamente por el pipe de solicitudes
char buffer_entrada[20];
while(1){
    memset(buffer_entrada, 0, sizeof(buffer_entrada));
    ssize_t bytes_leidos = read(fd_sus_solicitud, buffer_entrada, sizeof(buffer_entrada) - 1);
    if (bytes_leidos > 0) {
        printf("----recibiendo solicitud %s \n", buffer_entrada);
        buffer_entrada[bytes_leidos] = '\0';
        char letras[5];
        int pid;
        //se deserializa el mensaje, que es, obtener el pid y las categorias a las que se quiere suscribir
        DeserializarMensajeSuscriptor(buffer_entrada, letras, &pid);

        //se crea un pipe personal para ese nuevo suscriptor
        struct Suscriptor nuevo_suscriptor;
        nuevo_suscriptor.pid = pid;
        strncpy(nuevo_suscriptor.categorias, letras, 5);
        nuevo_suscriptor.nombre_pipe = malloc(100);
        snprintf(nuevo_suscriptor.nombre_pipe, 100, "/tmp/suscriptor%d", pid);
        unlink(nuevo_suscriptor.nombre_pipe);
        if (mkfifo(nuevo_suscriptor.nombre_pipe, 0666) == -1) {
            perror("Error creando el pipe para el suscriptor");
        }
        nuevo_suscriptor.fd_pipe = open(nuevo_suscriptor.nombre_pipe, O_RDWR);
        if (nuevo_suscriptor.fd_pipe == -1) {
            perror("Error abriendo pipe para el suscriptor");
        }
```

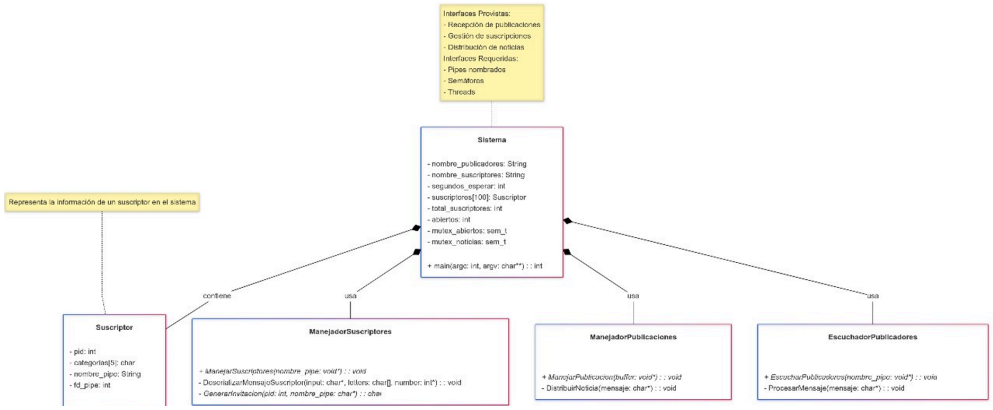
Luego de abrir los pipes, se entra a escuchar indefinidamente por el pipe de solicitud, si se recibe una solicitud, entonces se procesa, para esto, se llama a la función “DeserializarMensajeSuscriptor” que se encarga de extraer el PID y las categorías a las que ese suscriptor se quiere suscribir. Luego de obtener esa información, se crea un nuevo suscriptor y se le asigna el valor del PID y las categorías suscritas. Luego se genera el nombre del pipe de ese suscriptor al

añadirle su PID al prefijo “/tmp/suscriptor”. Al obtener ese nombre, se crea y abre el pipe para ese suscriptor.

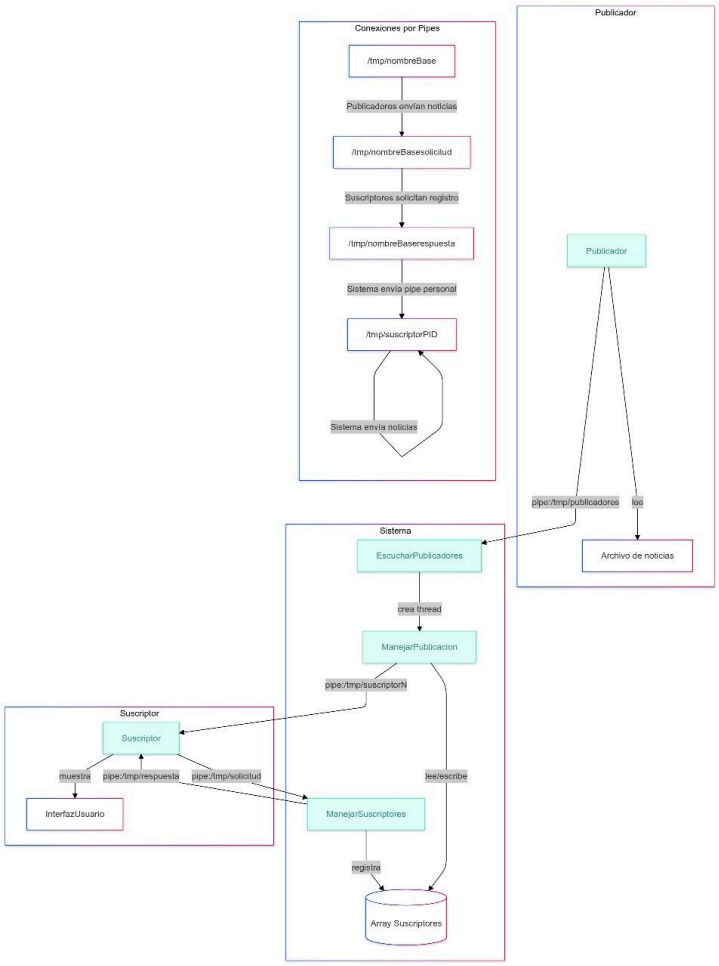
```
        //ZONA CRITICA, se agrega el nuevo suscriptor al areglo de suscriptores globales
        sem_wait(&mutex_noticias);
        suscriptores[total_suscriptores++] = nuevo_suscriptor;
        sem_post(&mutex_noticias);

        //se genera y envia la invitacion al suscriptor
        char* mensaje = GenerarInvitacion(pid, nuevo_suscriptor.nombre_pipe);
        printf("\n----invitacion enviada %s\n\n", mensaje);
        write(fd_sus_respuesta, mensaje, strlen(mensaje) + 1);
        free(mensaje);
    }
}
close(fd_sus_solicitud);
close(fd_sus_respuesta);
return NULL;
```

Luego de tener la información del suscriptor en una estructura, es necesario añadir ese suscriptor al arreglo de suscriptores, esto implica escribir en el arreglo global y en la variable que lleva el conteo de suscriptores, por lo que es ZONA CRITICA, asi que se toma un cupo del semáforo “mutex_noticias”, se agrega la estructura, y luego se devuelve ese cupo.

Finalmente se genera la invitación llamando a “GenerarInvitacion”, esta función solo se encarga de juntar el PID y el nombre del pipe en 1 sola cadena tal como se establece en el formato de respuesta, y con esto, se envía el mensaje al pipe de suscriptores.



En el siguiente diagrama se puede ver un resumen de los 3 componentes y como es la interacción entre ellos por medio de los pipes de cada uno.



IV. CONCLUSIONES.

En conclusión, la implementación del sistema de noticias utilizando el patrón Publicador/Suscriptor ha demostrado ser un método efectivo para gestionar la transmisión de información entre procesos de forma modular y eficiente. La estructura diseñada permite que los publicadores puedan enviar noticias sin conocer a los receptores, mientras que los suscriptores reciben únicamente las noticias de los tópicos que les interesan. Esto optimiza la distribución de información, permitiendo una comunicación selectiva y personalizada.

Además, el sistema de comunicación central desempeña un papel crucial, garantizando que cada noticia se envíe a los suscriptores adecuados mediante un proceso de filtrado que considera las suscripciones activas. La inclusión de mecanismos como pipes nominales y el uso de threads y semáforos permite una administración segura y ordenada de los datos, asegurando que el sistema pueda escalar sin comprometer la sincronización o el desempeño.

Finalmente, el diseño del proyecto no solo se alinea con los principios del patrón de Publicador/Suscriptor, sino que también incorpora prácticas de buena organización de recursos y control de procesos, lo cual facilita su extensibilidad y robustez. Este sistema proporciona una base sólida para futuras mejoras, donde podrían añadirse

nuevas funcionalidades o categorías sin alterar la estructura fundamental del proyecto.