Universidad
Rey Juan Carlos

Práctica: Publicador/Suscriptor

versión 0.6.1

Sistema distribuidos y concurrentes

Grado de Ingeniería de Robótica Software

En esta cuarta práctica aplicaremos los conocimientos adquiridos en clase de teoría sobre el patrón de diseño publicador/suscriptor.

Deberás implementar un sistema publicador/suscriptor para envío de mensajes. La arquitectura deberá soportar un único broker centralizado y multitud de publicadores suscriptores.

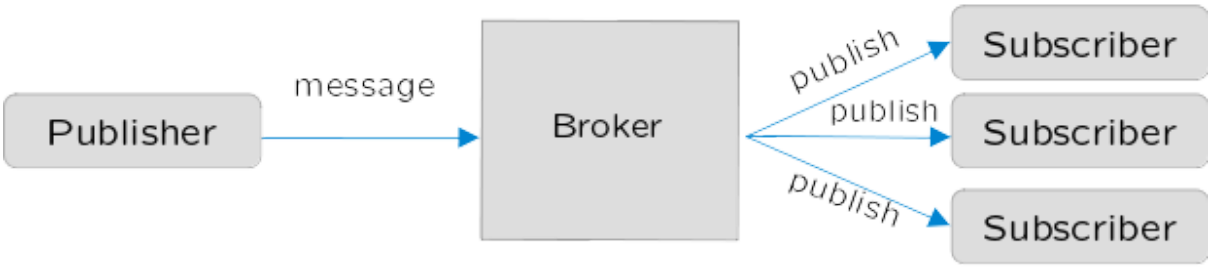
Existen 3 agentes en nuestra arquitectura: publicador, broker y suscriptor. Pueden existir multitud de instancias de publicador y suscriptor, pero sólo una única instancia de broker (todos los mensajes y datos pasan por el broker).

Broker

Solo existe un broker en el sistema. Dicho broker debe aceptar conexiones de registro de publicadores y suscriptores. Debe crear una conexión específica para cada uno de ellos.

La comunicación de broker con los publicadores y suscriptores para el registro se puede resumir en el siguiente esquema (intercambio de estructuras `message` y `response` definidas más adelante).

La comunicación cuando se publican mensajes con datos, se puede resumir de la siguiente manera (envío de estructura message entre publicador y broker, y envío de la estructura publish entre el broker y los suscriptores)



El broker aceptará un máximo de 10 topics distintos, 100 publicadores y 900 suscriptores. El broker tendrá 3 modos de funcionamiento:

- **secuencial**: Cada vez que reciba un mensaje de un publicador, mandará secuencialmente ese mensaje a los suscriptores registrados a ese topic siguiendo una política FIFO. Al ser secuencial, hasta que no termine la comunicación con un suscriptor no puede pasar al siguiente. Es el modo por defecto de funcionamiento del broker.
- **paralelo**: Cada vez que reciba un mensaje de un publicador, mandará en paralelo el mensaje a los suscriptores registrados a ese topic.
- **justo:** Cada vez que reciba un mensaje de un publicador, debe asegurarse que todos los suscriptores lo reciben “al mismo tiempo”. Utiliza los mecanismos de sincronización vistos en clase que más se ajusten a tus necesidades.

El broker acepta los siguientes parámetros:

```
./broker --port $BROKER_PORT --mode $MODE
```

El broker debe mostrar por la salida estándar únicamente los siguientes mensajes:

- Cuando se conectan satisfactoriamente un publicador o suscriptor

```
[SECONDS.NANOSECONDS] Nuevo cliente ($ID) Publicador/Suscriptor conectado : $TOPIC
Resumen:
      $TOPIC1: M Suscriptores - N Publicadores
      $TOPIC2: M Suscriptores -  N Publicadores
      ...
```

- Cuando se desconectan satisfactoriamente un publicador o suscriptor

```
[SECONDS.NANOSECONDS] Eliminado cliente ($ID) Publicador/Suscriptor : $TOPIC
Resumen:
      $TOPIC1: M Suscriptores - N Publicadores
      $TOPIC2: M Suscriptores - N Publicadores
      ...
```

- Cuando recibe un mensaje de un publicador.

```
[SECONDS.NANOSECONDS] Recibido mensaje para publicar en topic: $TOPIC - mensaje: $data - Generó: $time_generated_data
```

- Cuando envía un mensaje a los suscriptores

```
[SECONDS.NANOSECONDS] Enviando mensaje en topic $TOPIC a $N suscriptores.
```

Publicador

Los publicadores deben ser capaces de conectarse al broker para 1) registrarse y notificar en qué topic van a publicar y 2) publicar datos en ese topic. Los publicadores utilizarán las siguientes estructuras para comunicarse con el broker.

Para realizar el registro, el publicador debe mandar una estructura 'message' correctamente inicializada, con la operación a realizar y el topic.

```
enum operations {
      REGISTER_PUBLISHER = 0,
      UNREGISTER_PUBLISHER,
      REGISTER_SUBSCRIBER,
      UNREGISTER_SUBSCRIBER,
      PUBLISH_DATA
};

struct publish {
   struct timespec time_generated_data;
   char            data[100];
};

struct message {
   enum operations    action;
   char[100]          topic;
   // Solo utilizado en mensajes de UNREGISTER
   int                   id;
   // Solo utilizado en mensajes PUBLISH_DATA
   struct publish data;
};
```

El broker debe contestar con la estructura *response* indicando si la operación de registro se ha realizado correctamente y devolviendo un ID único de publicador (los ids de los publicador empieza en 0 y son incrementales).

```
enum status {
      ERROR = 0,
      LIMIT,
      OK
};
```

```
struct response {
   enum status response_status;
   int id;
};
```

El tipo enumerado status define los siguientes estados:

- OK: El registro fue correcto
- LIMIT: Se llegó al límite de los publicadores/suscriptores/topics que se pueden almacenar.
- ERROR: Cualquier otro tipo de error

El id debe establecerse a -1 para los estados de LIMIT y ERROR.

El publicador debe mostrar por salida estándar únicamente los siguientes mensajes:

- Cuando se conecta al broker

```
[SECONDS.NANOSECONDS] Publisher conectado con el broker correctamente.
```

- Cuando se registra correctamente/incorrectamente con el broker

```
[SECONDS.NANOSECONDS] Registrado correctamente con ID: $ID para topic $TOPIC
[SECONDS.NANOSECONDS] Error al hacer el registro: error=$status
```

- Cuando publica información (tiempo en epoch)

```
[SECONDS.NANOSECONDS] Publicado mensaje topic: $topic - mensaje: $data - Generó: $time_generated_data
```

- Cuando hace el unregister del broker (debe recibir el response).

```
[SECONDS.NANOSECONDS] De-Registrado ($ID) correctamente del broker.
```

El publicador deberá estar siempre ejecutando y operativo, hasta que se pulse Control+C, lo que llevará al proceso a realizar el unregister en el broker, y terminar correctamente.

El publicador debe permitir la configuración del topic y del broker a través de la línea de comandos.

```
./publisher --ip $BROKER_IP --port $BROKER_PORT --topic $TOPIC
```

Para realizar tus pruebas, haz que el publicador genere información para publicar cada 3 segundos. En ese momento de generar la información, es cuando debes guardar el generated_time. La información a publicar es el uso de la CPU donde está ejecutando ese publicador. Para ello, lee y manda la información que encontrarás en `/proc/loadavg` (consulta `man proc` para entender el significado de los números)

Suscriptor

Los suscriptores deben ser capaces de conectarse al broker para 1) registrarse y notificar a qué topic quieren subscribirse y 2) Recibir datos siempre que haya disponible en ese topic.

El suscriptor debe registrarse usando la estructura `message` correctamente inicializada. El broker contestará con la estructura `response` notificando el estado del registro. Después debe quedar a la espera de recibir datos en ese topic a través de la estructura `publish`. Asegúrate que el broker y los suscriptores mantienen la conexión abierta aunque no haya datos que enviar.

Los identificadores definidos por el broker para los suscriptores deben ser únicos (los ids de los suscriptores empiezan en 1 y son incrementales). El suscriptor deberá estar siempre ejecutando y operativo, hasta que se pulse Control+C, lo que llevará al proceso a terminar, y a realizar el unregister en el broker.

El suscriptor debe mostrar por salida estándar únicamente los siguientes mensajes:

- Cuando se conecta al broker

```
[SECONDS.NANOSECONDS] Subscriber conectado con broker
```

- Cuando se registra correctamente/incorrectamente con el broker

```
[SECONDS.NANOSECONDS] Registrado correctamente con ID: $ID para topic $TOPIC
[SECONDS.NANOSECONDS] Error al hacer el registro: $RESPONSE_STATUS
```

- Cuando recibe la información:
  En este caso, lo que nos interesa es saber la latencia desde que se generó el mensaje en el publicador hasta que se recibe en el suscriptor. Por tanto cada vez que recibas un mensaje en el suscriptor debes obtener la hora actual de la recepción y calcular la latencia en segundos teniendo una resolución de microsegundos (por ejemplo 0.000001).

```
[SECONDS.NANOSECONDS] Recibido mensaje topic: $topic - mensaje: $data - Generó: $time_generated_data - Recibido: $time_received_data - Latencia: $latency.
```

- Cuando hace el unregister del broker

```
[SECONDS.NANOSECONDS] De-Registrado ($ID) correctamente del broker.
```

El suscriptor debe permitir la configuración del broker a través de la línea de comandos

```
./subscriber --ip $BROKER_IP --port $BROKER_PORT --topic $TOPIC
```

Evaluación

Cuando tengas todo el sistema funcionando será el momento de realizar una evaluación en términos de latencia de comunicación para los diferentes modos de ejecución del broker.
Para ello:

- Arranca el broker, publicador y suscriptor en máquinas distintas del laboratorio (no ejecutes los 3 procesos en tu máquina, de hecho es obligatorio ejecutar cada proceso en una máquina distinta).
- Asegúrate que las máquinas estén correctamente sincronizadas en tiempo. Consulta el wiki de la asignatura:
  https://gitlab.etsit.urjc.es/roberto.calvo/sdc/-/wikis/Sincronizacion

- Ejecuta N suscriptores asociados a un mismo TOPIC. Asegúrate que guardas su salida estándar en un fichero para su posterior análisis. Puedes utilizar el siguiente comando para ejecutar un comando, ver su salida estándar y además guardarla a fichero
  - `$ comando_shell | tee fichero.txt`
- Ejecuta 1 publicador asociado al mismo TOPIC y deja que genere al menos 100 datos nuevos para publicar.
- Para la ejecución de los 3 procesos

Ahora es el momento de analizar las salidas estándar de todos los suscriptores, extrae todas las latencias calculadas para calcular su valor máximo, mínimo, medio y su desviación estándar. La desviación estándar cuantifica la variación o la dispersión de un conjunto de datos numéricos.

- Se recomienda utilizar comandos de shell para extraer los valores de latencia de los ficheros de texto (grep, cat cut, …).
- Se recomienda utilizar un script sencillo en python para calcular el máximo, mínimo, medio y desviación estándar utilizando numpy (np.max, np.min, np.mean y np.std)

Repite el escenario de evaluación para los 3 modos del broker (secuencial[0], paralelo[1] y justo[2]) para valores de N suscriptores de: 50, 500, 900. Deberás generar un fichero CSV con los resultados (`results_latency.csv`) que debes entregar junto con el código. El fichero debe tener el siguiente formato

```
# Modo, N, min, max, avg, std  (no incluyas esta línea)

0,   50, $MIN, $MAX, $AVG, $STD
0,  500, $MIN, $MAX, $AVG, $STD
0,  900, $MIN, $MAX, $AVG, $STD
[...]
```

En total debes tener 9 líneas, una para cada uno de los escenarios (3 modos x 3 valores de N). **Consideraciones**

- Para programar los argumento de tu programa consulta la página de manual de *getopt* y *getopt_long* o el siguiente enlace: https://linux.die.net/man/3/getopt_long
- Las trazas/logs deben aparecer siempre con el tiempo epoch `[SECONDS.NANOSECONDS]`
- Presta atención a las definiciones estáticas de variables que se guardan en el stack del proceso. Utiliza `malloc` y `free` si piensas que puedes exceder el tamaño máximo del stack.
- **Debes controlar cualquier problema de concurrencia que puedas tener**. Puedes usar todas las primitivas de concurrencia vistas en clase que necesites y establecerlas en el punto que se considere oportuno para garantizar el correcto funcionamiento del sistema.
- Toda la lógica de comunicaciones y mensajes debe quedar encapsulada en stubs. Puedes utilizar stubs diferentes para cada programa, pero NO repitas código.
- Puedes utilizar pthreads y sockets con la configuración que creas más conveniente (bloqueante, no-bloqueante).
- La práctica debe compilar y ejecutar en los ordenadores del laboratorio.
- Únicamente deben aparecer las trazas descritas en el enunciado.
- Los procesos deben llamarse “publisher”, “broker” y “subscriber”
- Ejecuta cada proceso en una máquina física distinta.
- **NO ESTÁ PERMITIDO espera activa en ningún caso.**
- Sigue el estilo de código definido para esta asignatura.