Sistemas Distribuidos

Práctica individual

Kaska

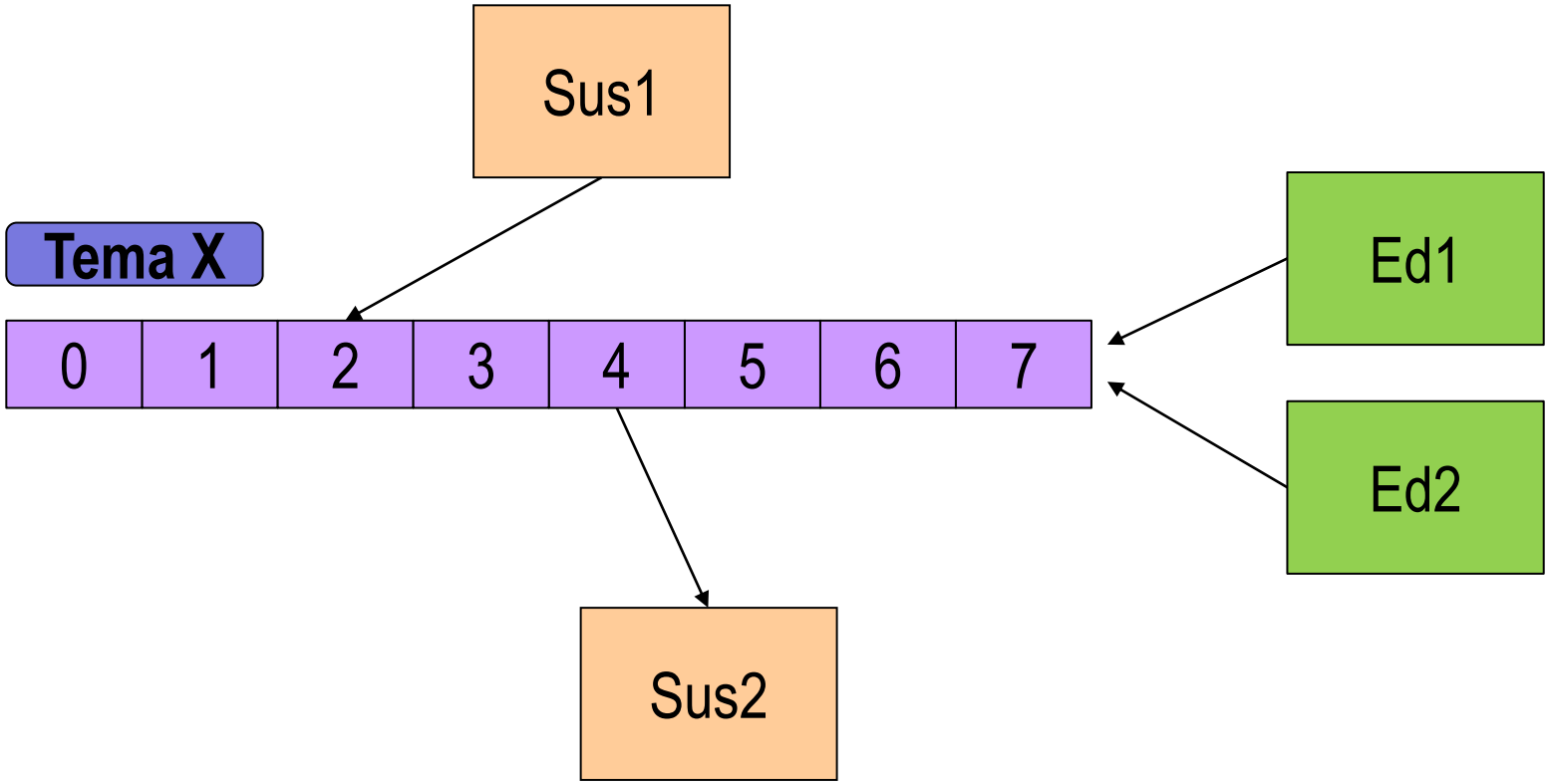
Introducción

- Individual
- Plazos: ordinaria 2 de junio; extraordinaria 5 de julio
- Desarrollar en máquina con Linux y entorno C
  - Máquina personal o triqui.fi.upm.es
    - En cualquier caso, debe entregarse en triqui
- Aviso importante sobre la copia de prácticas:
  - Se aplica normativa al respecto a todos los implicados
  - Alumnos que no se vean capacitados:
    - Enunciado con pasos detallados a partir de material de apoyo inicial
    - Soporte de tutorías
    - Con pequeño esfuerzo y enviándome todos los correos que hagan falta
      - Seréis capaces de superar la práctica
  - Alumnos presionados para dejar su práctica:
    - Tened en cuenta el punto anterior

Objetivo

- *EdSu* con esquema *pull* y uso de *streaming*
  - Inspirada en Apache Kafka pero muy simplificada
- *broker* almacena mensajes enviados por editores a los temas
- suscriptor guarda a qué temas está suscrito
  - Y *offset* de último mensaje leído de cada uno de esos temas
    - esa parte del estado no se almacena en el *broker* sino en biblioteca cliente
- suscriptor pide al *broker* un nuevo mensaje indicándole
  - a qué temas está suscrito y cuál es su offset para cada uno
- suscripción tema: *offset* inicial solo permite ver nuevos mensajes
- suscriptor puede modificar el *offset* de un tema
- suscriptor puede hacer persistente el *offset* de un tema

Modelo editor/suscriptor *streaming*

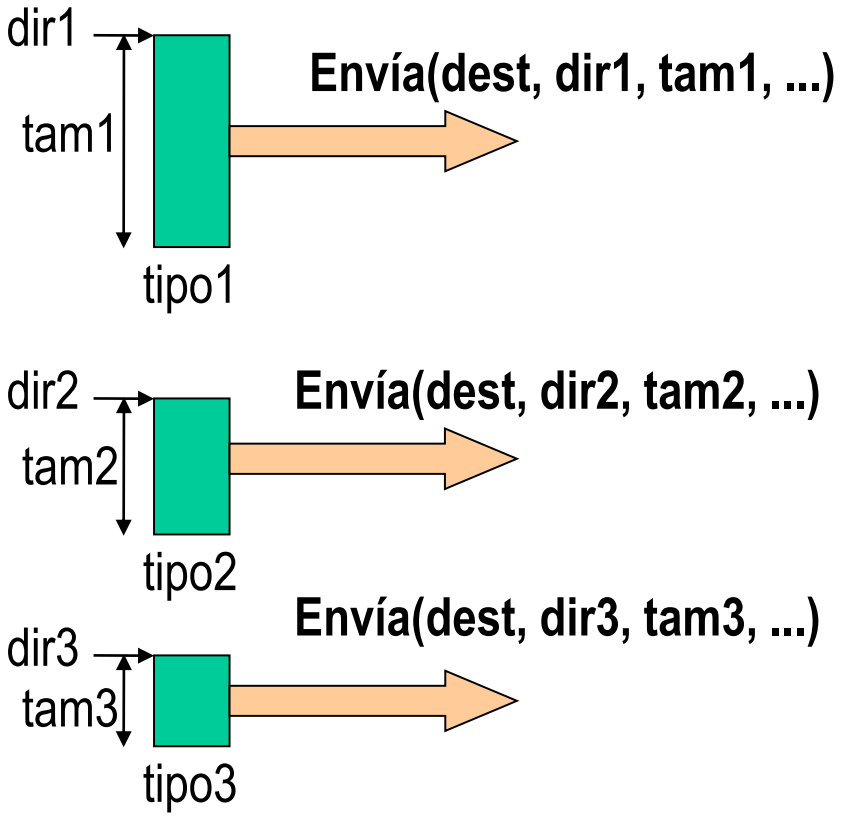


Cada suscriptor de un tema tiene asociado un *offset* en la cola que se guarda en el propio cliente

Requisitos

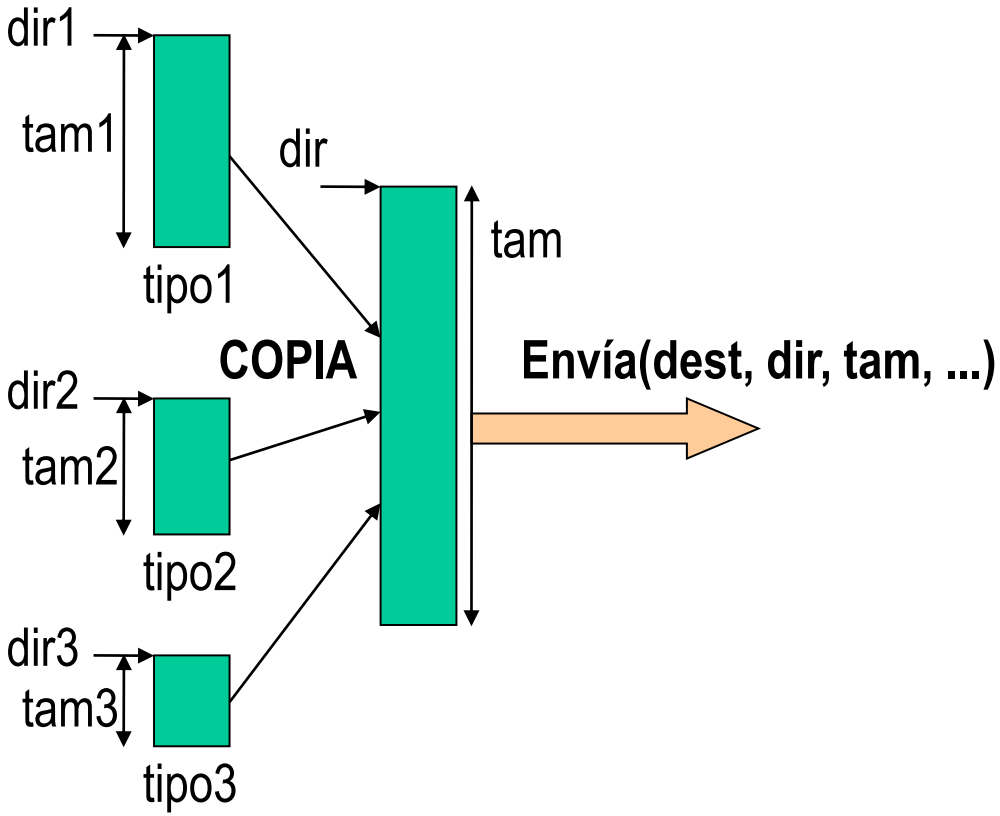
- Debe funcionar tanto en local como en remoto.
- C y sockets *stream* en entorno heterogéneo
- Un solo *broker* que almacena los mensajes
- Servicio concurrente con *threads* dinámicos: 1 por conexión
- Cliente (editor/suscriptor) conexión persistente con *broker*
- *tema* (*string*; longitud $\leq 2^{16}-1$)
  - Recuerde asegurar cadenas de caracteres recibidas terminan con nulo
- Mensajes pueden tener contenido binario
- Uso obligatorio de tipos de datos proporcionados: *map* y *queue*
- No límite en: nº temas, suscriptores y mensajes
- *zerocopy*: no copias de nombre de temas ni de mensajes
  - Ni múltiples envíos porque producen fragmentación
- Optimizar ancho de banda gastado

Datos múltiples: varios envíos



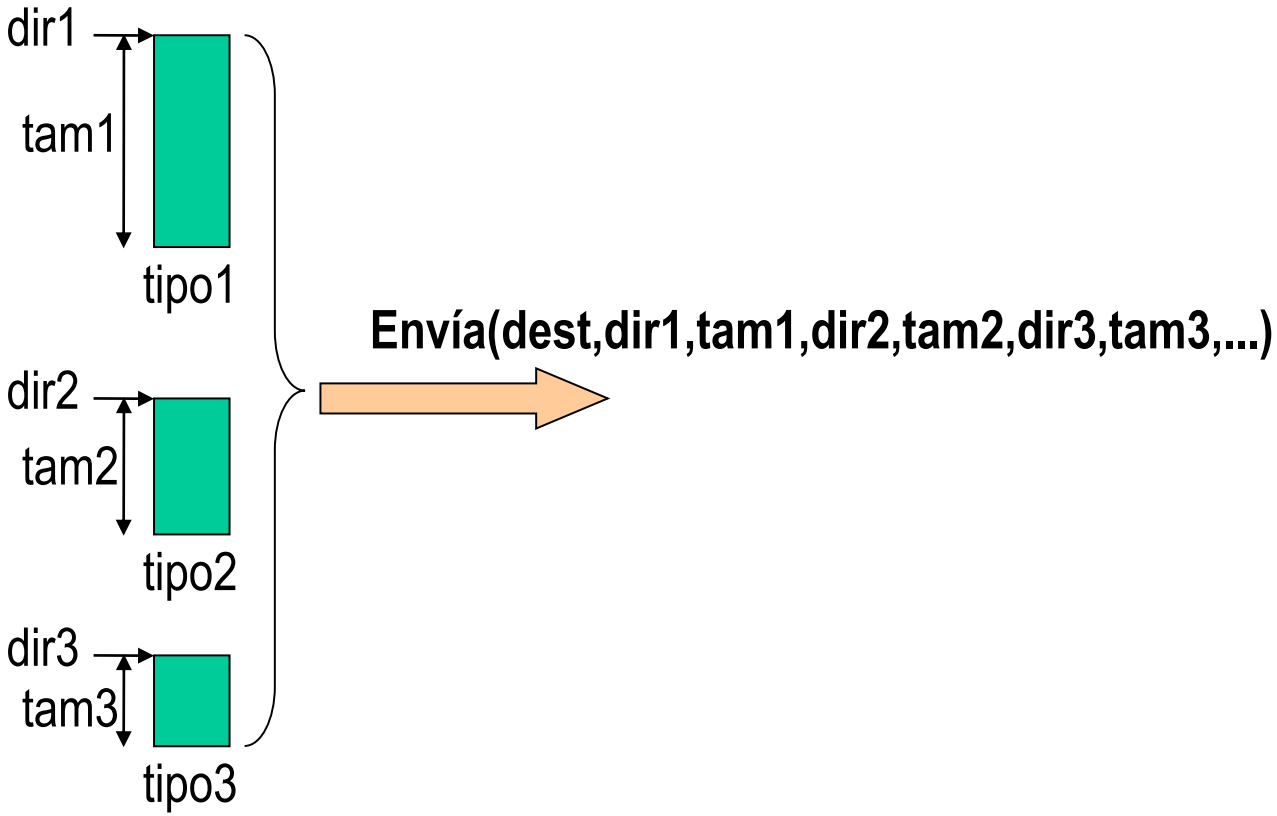
sobrecarga de llamadas + fragmentación de mensajes

Datos múltiples: envío con copia



sobrecarga por copias

Datos dispersos: envío *gather*

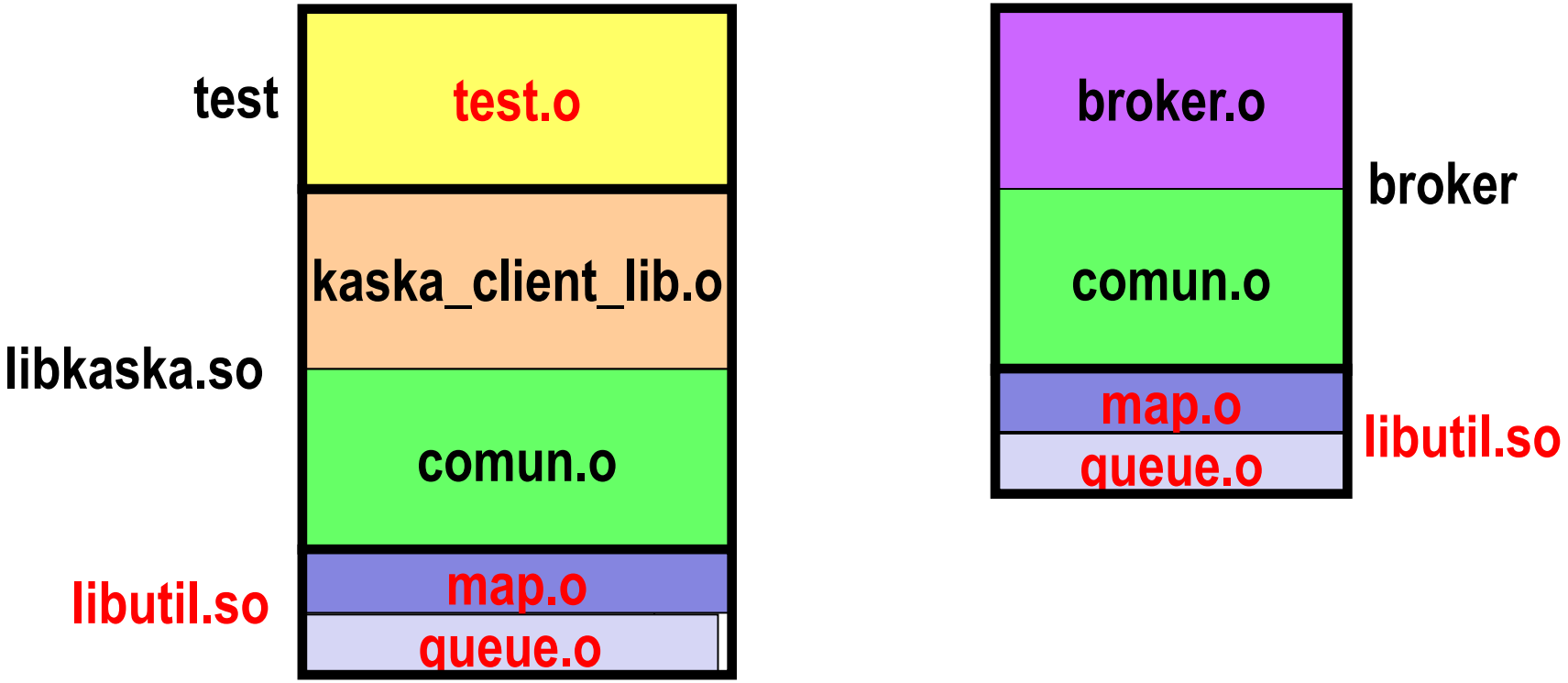


Uso de *writev*

API ofrecida a las aplicaciones

```
int create_topic(char *topic);
int ntopics(void);
int send_msg(char *topic, int msg_size, void *msg);
int msg_length(char *topic, int offset);
int end_offset(char *topic);
int subscribe(int ntopics, char **topics);
int unsubscribe(void);
int position(char *topic);
int seek(char *topic, int offset);
int poll(char **topic, void **msg);
int commit(char *client, char *topic, int offset);
int commited(char *client, char *topic);
```

Arquitectura software



Descarga y ejecución de pruebas

```
wget https://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/sd/kaska-2023.tgz
tar xvf kaska-2023.tgz
```

```
cd broker; make
./broker 12345
```

```
cd clients; make
export BROKER_PORT=12345
export BROKER_HOST=host_del_broker
./test
```

```
cd test; make
BROKER_PORT=12345 BROKER_HOST=host_del_broker ./test
```

Fases de la práctica

1. Creación de temas (2 puntos)
2. Publicación de mensajes (2 puntos)
3. Suscripción (2 puntos)
4. Lectura de mensajes (2 puntos)
5. *Offsets* persistentes (2 puntos)

Fase 1: Creación de temas

- Versión inicial: *servidor.c→broker.c; cliente.c→kaska_client_lib.c*
- ¿Cómo codificar operaciones y respuestas en el protocolo?
  - Podemos usar un *int* y así nos valen los ejemplos de sockets
- Estructura de datos para almacén de mensajes por temas
  - Mapa que asocia nombre de tema (clave) con descriptor de tema (valor)
    - Activación de sincronización interna por acceso desde múltiples *threads*
  - Descriptor de tema puede ser *struct* con nombre y cola
  - Similar a demo de *util*
- *create_topic*:
  - Crea descriptor de tema y cola e inserta entrada en el mapa
- *ntopics*:
  - Devuelve el tamaño del mapa

Fase 2: Envío de mensaje

- Descriptor de mensaje
  - Tamaño y referencia al mensaje
- Cola: lista de descriptores de mensajes
- *send_msg*:
  - Requiere enviar un entero adicional (*offset*)
  - *broker* debe crear descriptor de mensaje e insertar al final de la cola
- *message_length*:
  - Usa *queue_get* para acceder al mensaje especificado
- *end_offset*:
  - Devuelve tamaño de la cola correspondiente

Fase 3: Suscripción

- Todo el código corresponde a la biblioteca de cliente
- Uso de mapa para asociar temas suscritos con su *offset*
  - Biblioteca no *multithread*: no necesaria sincronización interna
- *subscribe*:
  - Crea mapa de temas suscritos (no incremental: error si ya existía uno)
  - Uso de *end_offset* para validar tema y obtener *offset* actual
  - Ruptura de *zerocopy*: *strdup* del nombre del tema
- *unsubscribe*: libera mapa (error si no existía)
- *position*: obtiene *offset* local de un tema usando *map_get*
- *seek*: actualiza *offset* local de un tema usando *map_get*

Fase 4: Lectura de mensaje (*poll*)

- Ya hemos llegado al 6: descripción más somera
- Parte biblioteca:
  - Itera por temas suscritos
  - Por cada uno envía a *broker*, el tema y el *offset* (como *msg_length*)
  - Si error o no hay mensaje, sigue iterando
  - Si hay mensaje, deja de iterar y devuelve mensaje
  - Llamada *poll* debe comenzar iteración del mapa
    - justo después de donde terminó la de la llamada previa
- Parte biblioteca:
  - Similar a tratamiento de *msg_length* pero retornando el mensaje

Fase 5: *Commit* de *offsets*

- App puede reanudar procesamiento después de reinicio
- App controla cuándo hacer persistentes en el *broker* los *commits*
  - Si se mantienen en la memoria del *broker*
    - Permite caída de la app, pero no del *broker*
- Kafka almacena en disco y replica toda la información relevante
- Almacenar en disco todos los mensajes complica la práctica
  - Pero sí vamos a guardar los *offsets*
- Nuevos argumentos en los programas:
  - *./test id_cliente; ./broker puerto dir*
- Offset se almacena en fichero *dir/id_cliente/tema*
- *commit*: escribe el *commit* en el fichero
- *commited*: lee el *commit* del fichero

Entrega de la práctica

- Alumno puede usar su propia máquina pero entrega en *triqui*
- Ciclo de vida de la práctica:
  - Descarga de material de apoyo de página web de asignatura
  - Instalación material de apoyo
  - Desarrollo de (parte de) la funcionalidad pedida
  - Entrega de la práctica (solo desde triqui)
    - **entrega.sd kaska.2023**
  - Corrección automática (0, 12, 15, 18 y 21 horas)
    - Se activará a mitad del periodo de prácticas
  - Resultado de corrección → correo cuenta del alumno en triqui
  - Número de entregas ilimitado
  - Última entrega se considera la versión definitiva