# Sistemas Distribuidos

## Práctica EdSu

### Proyecto práctico de tipo individual

# Desarrollo de prácticas

- En Linux, usando C/sockets (individuales) y Java RMI (grupo)
- Máquina del centro de cálculo: **triqui.fi.upm.es**
- 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 (sólo desde **triqui**)
  - Corrección automática (0, 12, 15, 18 y 21 horas)
  - Resultado de corrección → correo cuenta del alumno en **triqui**
  - Número de entregas ilimitado
  - Última entrega se considera la versión definitiva

# Objetivo

- Diseño de sistema editor/subscriptor en C con sockets
  - Basado en temas
  - Uso de un intermediario
  - Modelo *push*
  - Estructura de un evento: tema + valor
  - Uso de sockets *stream* en entorno con máquinas heterogéneas
- Dos versiones:
  - Básica (7 puntos): temas estáticos (fichero de configuración)
  - Avanzada (3 puntos): temas dinámicos
- Tres componentes:
  - **Intermediario** (programa)
  - **Subscriptor** (módulo) y **Editor** (módulo)
  - Una aplicación puede incluir ambos módulos (ambos roles)

# Editor y subscriptor (v. básica)

- ***IMPORTANTE**: Debe funcionar correctamente tanto si los procesos están en distintas máquinas como en la misma. Además, debe permitir varios subscriptores en una máquina.*

---

- Máquina y puerto del intermediario
  - Variables de entorno SERVIDOR y PUERTO
- Operaciones editor

  *int **generar_evento(**const char *tema, const char *valor);*
- Operaciones subscriptor

  *int **inicio_subscriptor**(void (*notif_evento)(const char *, const char *),*
  *void (*alta_tema)(const char *), void (*baja_tema)(const char *));*
  *int **alta_subscripcion_tema**(const char *tema);*
  *int **baja_subscripcion_tema**(const char *tema);*

# Ejemplo de aplicación subscriptora

```
static void notificacion_evento(const char *t, const char *e){
        printf("\n-> Recibido evento %s con valor %s\n", t, e);
        ……..}
int main(int argc, char *argv[]) {
        ……..
       inicio_subscriptor(notificacion_evento, NULL, NULL);
        ……..
        if (alta_subscripcion_tema(argv[i])<0)
                fprintf(stderr, "Error en subscripción a tema %s\n", argv[i]);
        ……..
        if (baja_subscripcion_tema(argv[i])<0)
                fprintf(stderr, "Error baja subscripción a tema %s\n",  argv[i]);
}
```

# Intermediario (versión básica)

- *./intermediario puerto fichero_temas*
- Estructura de datos: relación temas y subscriptores
- Recomendación: secuencial y 1 conexión/petición
- Mensajes recibidos por intermediario:
  - Inicio suscriptor no contacta con intermediario
  - Alta suscripción (de suscriptor):
    - Asocia remitente a tema; error si tema no existe
  - Baja suscripción (de suscriptor):
    - Desasocia remitente del tema; error si tema no existe o no suscrito
  - Evento (de intermediario)
    - Envía mensaje a suscriptores interesados en el tema
    - Error si tema no existe
    - Cada suscriptor invoca función de notificación de la aplicación

# Manejo de eventos en suscriptor

- Problemática
  - Asíncronos y *contracorriente*
- Una posible solución básica (uso a criterio del alumno):
  - Suscriptor usa socket separado para recibir notificaciones
  - En *inicio_subscriptor* se crea socket y *thread* que lo supervisa
  - En alta y baja subscriptor envía puerto de ese socket
  - Intermediario envía notificaciones a ese socket
  - *Thread* recibe notificaciones por ese socket e
  - invoca función recibida en *notif_evento* que ejecutará en su contexto
  - Problema: si esa función es larga, bloqueante o llama a la biblioteca
    - No se considera esa problemática para el proyecto
    - Posibles soluciones:
      - Crear nuevo *thread* para ejecutarla
      - Gestionar notificaciones con un bucle de eventos

# Una solución básica

***test_subscriptor.c***
YA HECHO

***subscriptor.c***

**inicio_subscriptor()**
- socket() // para recibir eventos
- bind() // cualquier puerto libre ***P***
- listen()
- pthread_create() // escucha eventos

**alta()**
- socket();
- preparar_mensaje(alta,tema,P)
- getenv SERVIDOR y PUERTO
- connect(); send(); recv(); close();
- return OK o error;

**baja()** // igual que alta excepto
- preparar_mensaje(baja,tema,*P*)

**thread()**
- while (1)
  - accept() // espera conexiones a *P*
  - recv() // recibe tema y valor
  - notifica_evento(tema, valor)

***intermediario.c***

**main()**
- iniciar est datos de fichero argv2
- socket()
- bind() // puerto argv1
- listen()
- while (1)
  - accept() // obtiene *IP* del cliente
  - recv()
  - Si alta: busca tema asociado y añade *IP* y *P* del subscriptor
  - Si baja: busca tema asociado y elimina *IP* y *P* del subscriptor
  - Si evento: busca tema asociado y por cada subscriptor *S*
    - socket();
    - connect(); *IP* y *P* de S
    - send(); // evento
    - close()
  - send() // envía OK o error

***test_editor.c***
YA HECHO

***editor.c***

**generar_evento()**
- socket();
- preparar_mensaje(evento,tema,valor)
- getenv SERVIDOR y PUERTO
- connect();
- send();
- recv();
- close();
- return OK o error;

# Editor (versión avanzada)

- Nuevas operaciones
  - *int* ***crear_tema****(const char *tema);*
  - *int* ***eliminar_tema****(const char *tema);*
- Crear tema
  - Intermediario actualiza estructuras de datos
  - Envía anuncio de alta de tema a todos los suscriptores
  - Subscriptor invoca función de aviso de alta de la *app*
- Eliminar tema
  - Intermediario actualiza estructuras de datos
  - Envía anuncio de baja de tema a todos los suscriptores
  - Subscriptor invoca función de aviso de baja de la *app*

# Subscriptor (versión avanzada)

- Nueva operación
  - *int* ***fin_subscriptor****();*
- Extensión del inicio subscriptor
  - Se especifican los tres parámetros
  - En esta versión sí contacta con intermediario
  - Intermediario incluye suscriptor en estructuras de datos
  - Subscriptor será informado de temas existentes
    - Usando propio mecanismo de anuncio de temas
- Fin de suscriptor
  - Intermediario elimina subscriptor de **todas** sus estructuras