Trabajo Práctico 2: `pthreads`
"Escape en Sistemas"

Sistemas Operativos - 2do cuatrimestre de 2014

1. Introducción y contexto

Durante el segundo parcial de la materia Sistemas Operativos del Departamento de Computación tuvo lugar un simulacro de evacuación que sorprendió a docentes y alumnos. El resultado del simulacro fue bueno (más del 90 % de evacuación en menos de 4 minutos[1]), sin embargo hay cuestiones a mejorar. Las autoridades del Departamento de Computación están intentando probar la efectividad de las salidas de emergencia de las aulas del pabellón (sobre todo del Aula Magna donde estaba siendo el parcial en el momento del simulacro) con el fin de facilitar la evacuación del edificio. Con este objetivo, han diagramado el siguiente modelo:

Un aula es un espacio rectangular de tamaño fijo, donde cabe cierta cantidad de personas por m^2. En caso de haber una emergencia, las personas empiezan a caminar hacia la puerta intentando salir. Una vez que atraviesan la puerta, un rescatista los saluda y les pone una máscara de oxígeno. Una vez que tienen la máscara puesta, deben esperar a completar un grupo de 5 personas para salir del edificio (a menos que sean los últimos) y ser libres y considerarse a salvo. Los rescatistas pueden atender a una sola persona a la vez y hay una cantidad limitada de ellos.

2. Especificación y diseño

Para simular esto se ha diseñado una implementación basada en un sistema cliente/servidor. El servidor, al comenzar, escucha en un puerto esperando que se conecte un cliente que representa a un alumno. El cliente envía el nombre del alumno, al igual que su posición inicial en el aula. Una vez hecho esto, se establece una comunicación bidirecciónal: El alumno intenta moverse hacia la puerta indicando una de cuatro posibilidades: Arriba, Abajo, Izquierda o Derecha. El servidor intenta mover a la persona hacia ese lugar y luego le responde al cliente el resultado: si pudo moverse, si no pudo hacerlo porque la cantidad de personas por m^2 alcanza el máximo posible. Desde el momento en que alcanza la puerta, el cliente se queda esperando una señal del servidor indicando que los rescatistas ya le han colocado la máscara y han podido salir del edificio. Para salir del edificio se debe completar un grupo de 5 personas con mascaras para moverse juntos hacia la salida y ser libres finalmente.

La versión actual del servidor soporta un único cliente a la vez, y no satisface a las autoridades por ser demasiado básica. Se pide desarrollar una nueva versión del servidor que permita atender en forma simultánea los pedidos de varios clientes.

3. Implementación actual

Actualmente se cuenta con dos programas, un servidor implementado en `servidor_mono.c` y un cliente en `cliente.c`. Además, se cuenta con una biblioteca de estructuras y funciones comunes a ambos programas en `biblioteca.c`. Todo se compila mediante un makefile. Al iniciar el proceso servidor queda habilitada la espera del próximo cliente. El aula es representada por una matriz de enteros de n x m, donde cada posición representa la cantidad de personas en ese lugar. La puerta

[1] Ver mail enviado por el Servicio de Higiene y Seguridad de la Facultad el día 24 de octubre.

se ubica en una posición fija. La posición de cada persona es mantenida tanto en su respectivo cliente como en el servidor.

3.1. Comunicación cliente/servidor

Para la comunicación entre el cliente y el servidor se utilizan las funciones enviar y recibir de la biblioteca provista, que se encargan de realizar el framing de los streams.

Establecida la conexión cliente/servidor, se utiliza el siguiente protocolo (ver también la fig. 1):

1. El cliente indica mediante `NOMBRE: n FILA: f COLUMNA: c`, su nombre y posición inicial. Para mantener la simplicidad esta posición no se chequea, por lo que será el cliente el encargado de enviar una posición válida. Tampoco se verifica la posición inicial al ingresar, y solo en este momento se permite sobrepoblar una posicion.

2. El cliente indica el próximo movimiento, enviando `ARRIBA`, `ABAJO`, `IZQUIERDA` o `DERECHA`.

   El servidor recibe el pedido y responde `OK` u `OCUPADO` según corresponda. (ahora no se permite sobreponer posiciones)

3. Las partes repiten este ciclo hasta que el cliente haya salido del lugar. Una vez que esto ocurre, el servidor espera por un rescatista para el alumno y cuando lo obtiene, le coloca la máscara, luego deberá esperar a armar un grupo de salida (5 personas o menos si es el ultimo grupo). Una vez armado el grupo, se envía al cliente `LIBRE!`

Si la conexión se interrumpe en medio de la huída, el servidor da de baja al cliente automáticamente, sacándolo de la habitación.

3.2. Módulo `biblioteca.c`

Este módulo provee la mayoría de las funcionalidades standard que son necesarias para una buena abstracción del problema permitiendo reutilizar la mayor parte del código en ambos servidores. Esto incluye una interfaz sencilla para enviar y recibir mensajes, implementando la mensajería de alto nivel (protocolo de la aplicación, "business logic") entre dos procesos o threads sobre una conexión TCP, usando `sockets` y primitivas de nivel más bajo. Entre otras cosas aporta manejo básico de algunos errores y resuelve tecnicismos de terminación de strings.

3.3. Módulo `cliente.c`

El programa `cliente` intenta ante todo establecer una conexión TCP al puerto 5555 de la dirección IP del servidor, que recibe como parámetro por línea de comando. Si lo logra, ejecuta indefinidamente el ciclo principal, en el que:

1. Lee (del usuario local) el nombre y la posición inicial.
2. Indica al servidor este nombre y posición.
3. Lee (del usuario local) la siguiente dirección.
4. Envía al servidor el pedido.
   - En caso de que el servidor responda `OK`, actualiza su estado a la nueva posición.
   - Si responde `OCUPADO` no hace nada.
5. Si la posición actual no es la de salida, vuelve a 3. Si no, se queda en espera del mensaje `LIBRE!`.

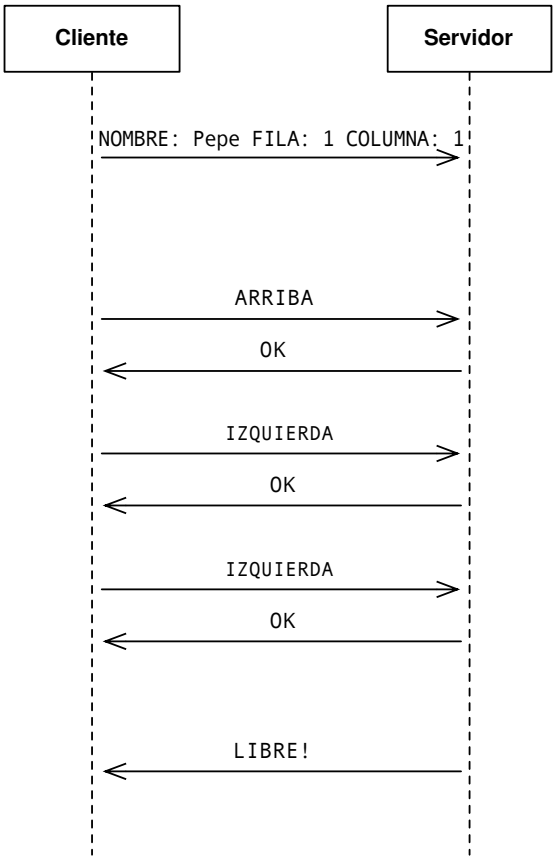

Figura 1: Ejemplo de escape exitoso (con puerta en posicion 0, -1).

3.4. Módulo `servidor_mono.c`

La implementación mono-thread del servidor, que puede utilizarse como base para el desarrollo de la nueva versión (también puede no usarse; es sólo una *fuerte* sugerencia). Mantiene el estado del aula, actualizándolo cuando un cliente se conecta y se mueve por la habitación. Para esto, muchas veces se vale de las funciones de `biblioteca.h`.

3.5. Módulo `server_tester.py`

Como herramienta alternativa se incluye un módulo en python que realiza un trabajo similar al del cliente. Este módulo permite conectar varios clientes a la vez y hacerlos avanzar de variadas maneras automáticamente. Su función es que los programadores puedan probar el comportamiento del server de manera simple. (Obrservacion: Este cliente no comunica concurrentemente varios clientes, pero el sever deberá soportarlo)

4. Diseño propuesto para la nueva versión

Luego de un riguroso análisis de la situación, las necesidades del cliente y las restricciones del caso, se ha logrado pergreñar un revolucionario diseño que se ilustra en la fig. 2.

El servidor involucra inicialmente un único thread ("master", "hilo principal") que escucha posibles conexiones TCP entrantes en el puerto 5555. Al arribar un pedido de conexión, crea un nuevo thread (un "worker") dedicado en exclusividad a atender el diálogo con la consola solicitante. Es importante notar que en la implementación actual el servidor llama directamente

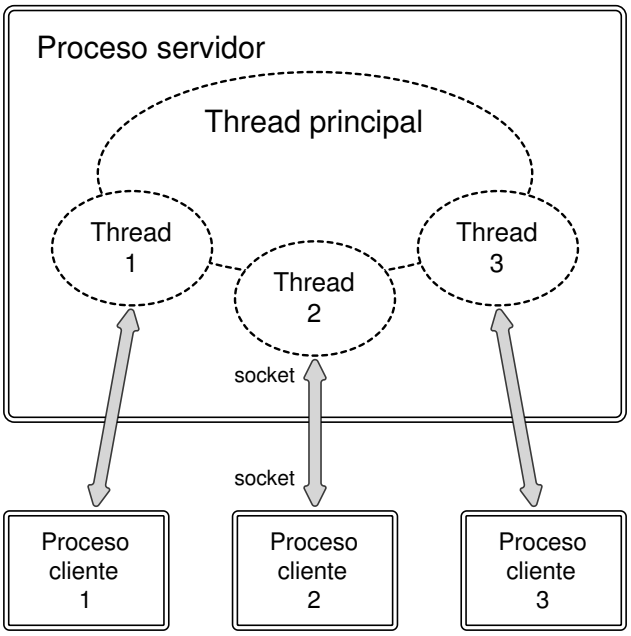

Figura 2: Croquis del nuevo diseño propuesto.

`atendedor_de_alumno` pásandole el file descriptor correspondiente a la conexión con el cliente y el puntero al aula. Estos dos parámetros también deberán ser enviados al crear el thread, pero mediante una estructura auxiliar.

Una vez creado el "worker thread", el principal vuelve a su estado anterior, quedando a la espera de nuevos pedidos de conexión en el puerto 5555.

5. Escalamiento

Las autoridades del departamento temen un incremento masivo en la cantidad de alumnos para los siguientes cuatrimestres y quisieran saber si el software es capaz de escalar a 1.000.000 clientes o mas. Se pide entonces que haga un análisis de la solución entregada, y proponga modicaciones de hardware y/o de software que podrán aplicarse para poder atender a muchos clientes concurrentemente.

6. Aspectos importantes a tener en cuenta

6.1. Sólo el servidor puede modificarse

A los efectos de asegurar que la actualización sea absolutamente transparente para el usuario, no está permitido modificar el código del cliente. El nuevo servidor debe ser 100 % compatible con el cliente original.

6.2. Sincronización

Solo podrá utilizarse los mecanismos de sincronización que provee la librería Pthreads (Mutex y Variables de condición).

Para moverse entre A y B se deberá actualizar ambas posiciones en el aula simultáneamente.

Se debe garantizar que el sistema es libre de deadlock.

6.3. Paralelismo

Se busca obtener el mayor grado de paralelismo, por lo tanto se debe minimizar la estructura que se bloquea en la exclusión mutua.

6.4. Fechas y forma de entrega

Se deberá entregar un informe completo, que incluya número de grupo nombre, LU y mail de los integrantes. Además se deberá incluir una descripción detallada de la implementación realizada, indicando los archivos creados y/o modificados, las dificultades encontradas, etc. Justificar en todos los casos todas las decisiones tomadas. Se evaluará la legibilidad del código, que deberá estar debidamente comentado.

Tanto el informe como el código deberán entregarse en un único comprimido llamado

`TP2- «Apellido integrante 1»-«Apellido integrante 2»-«Apellido integrante 3»`.

La fecha de entrega límite es el día **Miercoles 12 de noviembre a las 23.59 hs**.

El Trabajo se considera entregado cuando algunos de los alumnos reciba un mail de "RECIBIDO OK" confirmando la recepción del mismo.