Sistema de alerta temprana sobre el desbordamientos de ríos

Ejercicio N°2

Objetivos	• Diseño y construcción de sistemas con procesamiento concurrente • Diseño y construcción de sistemas con acceso distribuido • Encapsulación de Threads y Sockets en TDAs • Definición de protocolos de comunicación • Protección de los recursos compartidos
Instancias de Entrega	**Entrega 1**: clase 6 (22/09/2015). **Entrega 2:** clase 8 (06/10/2015).
Temas de Repaso	• POSIX Threads • Funciones para el manejo de Sockets
Criterios de Evaluación	• Criterios de ejercicios anteriores • Ausencia de secuencias concurrentes que permitan interbloqueo • Ausencia de condiciones de carrera en el acceso a recursos • Buen uso de Mutex, Condition Variables y Monitores para el acceso a recursos compartidos • Eficiencia del protocolo de comunicaciones definido • Control de paquetes completos en el envío y recepción por Sockets

Índice

Introducción

En la actualidad, la medición del nivel de los ríos para el aviso temprano de posibles inundaciones es una tarea manual haciendo que las alertas lleguen con poco tiempo de anticipación a las poblaciones cercanas. La idea es implementar un sistema automatizado de alerta temprana que en *real time* permita tener un vistazo de la situacion del río.
El presente trabajo práctico consta de la implementación (simplificada) de los dos softwares implicados en C++98: un **conector** que (junto a su sensor) realice las mediciones y un **servidor** que consolide los datos de múltiples conectores y resuelva las consultas.

Descripción

En la actualidad, la medición del nivel de los ríos para el aviso temprano de posibles inundaciones se realiza diariamente de forma manual. En días de lluvias, las mediciones se realizan cada hora para poder advertir a la población de las zonas propensas a inundarse.
A pesar de estos esfuerzos, las alertas llegan con poco tiempo de anticipación.

La idea es implementar un sistema de alerta temprana que realice mediciones todo el tiempo de forma automática. Con un flujo de información en *real time*, se puede tener predicciones más certeras y alertar a la población con horas de anticipación.

El sistema contará con varios sensores a lo largo del río, distribuidos en secciones, para la medición del nivel y caudal del agua.
La información de los sensores será procesada localmente y luego enviada a una central.
Via internet, se podrá acceder a la central y ver la situación actual del río por parte tanto de los pobladores como de los equipos de prevención y rescate.

El presente trabajo práctico consta de la implementación (simplificada) de los dos softwares necesarios para automatizar el sistema:

- el **conector** que se encargará de procesar los datos de los sensores y los enviará al servidor
- el **servidor** que centralizará los datos de los conectores y resolverá consultas de los **clientes externos** del sistema.

Cada sensor puede realizar varias muestras por segundo. Esta cantidad masiva de datos no pueden ser transmitida directamente al servidor ya que saturaría el link entre este y el sensor. Además, carecen de valor:

no es de esperarse que el nivel de un río ni su caudal cambien de forma significativa de un segundo a otro.
Para ello, se debe implementar un software que hará de **conector** entre el sensor real y el servidor.
El **conector** debe conectarse al servidor, leer por entrada estándar las mediciones del sensor y luego de haber leído N valores, tomar la moda [1] de estos como valor significativo, enviarla al servidor y descartar los N valores, repitiendo el proceso hasta que la salida estándar se cierre (el sensor se ha apagado).
La moda de un conjunto de valores es el valor más frecuente. En caso de haber 2 valores que tengan la misma frecuencia se debe tomar el de mayor valor.

El segundo elemento del sistema es el **servidor**. Este debe aceptar múltiples conexiones provenientes de los sensores y almacenar los valores recibidos.
Al mismo tiempo el servidor deberá poder atender consultas hechas por **clientes externos** y dar el estado actual del río por sección o en general.
El **servidor** se cerrará cuando se le pase una **q** a su entrada estándar.

Los **clientes externos** no formarán parte de este trabajo pero aun así, el servidor deberá poder responder sus consultas.

Formato de Línea de Comandos

El conector recibirá la IP y puerto del servidor a donde se conectará, el nombre de la sección del río donde está colocado (él y su sensor) y la cantidad N de muestras que debe leer para calcular la moda.

```
./conector <ip> <puerto> <seccion> <N>
```

El servidor recibirá el puerto en donde estara escuchando

```
./servidor <puerto>
```

En ambos casos, **<ip>** hace referencia a una IP o bien a un hostname. Para el parámetro **<puerto>** sucede lo mismo, puede ser un puerto numérico o el nombre de un servicio.

Códigos de Retorno

Tanto el servidor como el conector deben retornar 0 como código de retorno si no hubo ningún error o 1 si lo hubo: parámetros inválidos, conecciones cerradas antes de tiempo.

Entrada y Salida

Formato de entrada para el conector

El nivel y caudal del río vendrán por la entrada estándar en formato de texto con un par nivel-caudal en cada línea.
El formato de cada una de estas líneas es el siguiente:

```
n <xxx> c <yyy>\n
```

donde **<xxx>** y **<yyy>** son valores enteros y representarán el nivel y caudal del río en ese momento. Puede haber un número arbitrario de espacios entre los valores y los tags. Sólo se garantiza que habrá al menos un espacio. Toda línea finaliza con un salto de línea ('**\n'**).

Mensajes entre el conector y el servidor

Cuando el conector inicie la conexión con el servidor, deberá informar en que sección del río se encuentra. Para ello, el conector enviará una única vez el siguiente mensaje

```
conector seccion <xxx>\n
```

donde **<xxx>** es el nombre de la sección del río donde el sensor está colocado.

Una vez establecida la comunicación con el servidor y enviado este primer mensaje, el conector está listo para enviar los datos procesados. Estos envíos se realizarán a través de mensajes con formato:

```
actualizar nivel <xxx> caudal <yyy>\n
```

donde **<xxx>** y **<yyy>** son la moda de los últimos N elementos leídos por el conector.

Al cerrarse la entrada estándar del conector, este finalizara la conexión con el servidor enviandole el siguiente mensaje:

```
fin\n
```

Mensajes entre el cliente externo y el servidor

El servidor no solo acepta mensajes provenientes de un conector sino que también provenientes de un cliente externo.
El servidor deberá aceptar 2 tipos de consultas. En ambos casos, una vez que el servidor responda la consulta, este deberá cerrar la conexión con el cliente externo.
El primer tipo de consulta es de la forma:

```
consultar seccion <xxx>\n
```

donde **<xxx>** es el nombre de una sección del río.
El servidor responderá con

```
respuesta\n
seccion <xxx> nivel <yyy> caudal <zzz>\n
fin\n
```

donde **<xxx>** es el nombre de una sección del río consultada y donde **<yyy>** y **<zzz>** son los valores del nivel y caudal del río.

El segundo tipo de consulta que el servidor aceptará es una versión general del la anterior:

```
consultar\n
```

la cual, el servidor responderá con

```
respuesta\n
seccion <xxx1> nivel <yyy1> caudal <zzz1>\n
seccion <xxx2> nivel <yyy2> caudal <zzz2>\n
...
fin\n
```

donde **<xxx1>** es el nombre de una sección del río y donde **<yyy1>** y **<zzz1>** son sus valores del nivel y caudal. Idem para **<xxx2>**, **<xxx3>**, … .
Para tener un formato determinístico, las secciones en la respuesta deben estar ordenadas de menor a mayor según el orden ASCII.

Ejemplos de Ejecución

Ejemplo 1

Lanzamos el servidor en el puerto 3333

```
consola1> ./servidor 3333
```

Preguntamos el estado actual del río

```
consola2> echo -ne “consultar\n” | nc 127.0.0.1 3333
respuesta
fin
```

Lanzamos un conector que envía una sola muestra al servidor

```
consola3> echo -ne “n 1 c 2\n” | ./conector 127.0.0.1 3333 S1 1
```

Preguntamos el estado actual del río

```
consola2> echo -ne “consultar\n” | nc 127.0.0.1 3333
respuesta
seccion S1 nivel 1 caudal 2
fin
```

Lanzamos otro conector, esta vez hará la moda de 3 muestras y se conectará al localhost en vez de usar la ip directamente

```
consola3> echo -ne “n 1 c 3\nn 0 c 2\nn 1 c 3\n” | ./conector localhost 3333 S2
3
```

Preguntamos el estado actual del río (nótese el orden alfabético en la respuesta).

```
consola2> echo -ne “consultar\n” | nc 127.0.0.1 3333
respuesta
```

```
seccion S1 nivel 1 caudal 2
seccion S2 nivel 1 caudal 3
fin
```

O podemos consultar sólo una sección

```
consola2> echo -ne “consultar seccion S2\n” | nc 127.0.0.1 3333
respuesta
seccion S2 nivel 1 caudal 3
fin
```

Ejemplo 2

En este caso simularemos al servidor con un netcat para ver como nos llegan las mediciones de los conectores.

Lanzamos el servidor en el puerto 1080 (asociado al servicio socks)

```
consola1> nc -l 1080
```

Lanzamos un conector, esta vez hará la moda de 3 muestras primero y luego hará la moda de otras 3 muestras.

```
consola2> echo -ne “n 1 c 3\nn 0 c 2\nn 1 c 3\nn 2 c 3\nn 1 c 3\nn 2 c 3\n” | ./conector localhost socks S1 3
```

En la consola 1 veremos los mensajes que le llegaron al servidor

```
consola1>
conector seccion S1
actualizar nivel 1 caudal 3
actualizar nivel 2 caudal 3
fin
```

Lanzamos otro conector más, esta vez hará la moda de N=4 muestras. Nótese que para el caudal hay 4 valores medidos: 3,3,2,2 y que los valores **más frecuentes** son 3 (2 veces) y 2 (2 veces). En este caso de empate, se toma **la moda de mayor valor**.

```
consola2> echo -ne “n 2 c 3\nn 1 c 3\nn 2 c 2\nn 2 c 2\n” | ./conector localhost socks S2 4
```

En la consola 1 veremos los mensajes que le llegaron al servidor

```
consola1>
conector seccion S2
actualizar nivel 2 caudal 3
fin
```

Lanzamos un tercer conector, esta vez hará la moda de N=2 muestras. Nótese que la entrada consiste en 3 mediciones, dado que el conector procesara cada 2 muestras, la última quedará descartada.

```
consola2> echo -ne “n 1 c 3\nn 1 c 3\nn 5 c 4\n” | ./conector localhost socks S3 2
```

En la consola 1 veremos los mensajes que le llegaron al servidor

```
consola1>
conector seccion S3
actualizar nivel 1 caudal 3
fin
```

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C++98.
2. Está prohibido el uso de variables globales.
3. Use **istringstream** y **ostringstream** en donde le sea conveniente. [2]
4. Use la estructura de datos **map** en donde le sea conveniente. [3]
5. La abstracción Socket sabe enviar y recibir datos pero no debe saber en que consisten eso datos: no puede saber la clase Socket sobre los mensajes “actualizar”, “respuesta”, “fin”.
6. El conector no puede tener en memoria todas las muestras obtenidas, puede a lo sumo tener hasta N muestras a la vez.

Referencias

[1] Moda (estadística): http://es.wikipedia.org/wiki/Moda_%28estad%C3%ADstica%29
[2] sstream (C++): http://www.cplusplus.com/reference/sstream/
[3] map (C++): http://www.cplusplus.com/reference/map/map/