Sistema de reserva de turnos

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. (23/9) **Entrega 2:** clase 8.(07/10)
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

El consulado de Finlandia nos encargó el desarrollo de una aplicación cliente-servidor para administrar el sistema de reserva de turnos para realizar tramites, el mismo debe permitir asignar turnos, cancelarlos y consultar los turnos tomados.

Descripción

El sistema permitirá a los clientes reservar y cancelar turnos así como consultar las reservas realizadas.
La información referente a los turnos, la cantidad disponible y las reservas estarán contenidas en un archivo.
Se pide realizar dos programas: servidor y cliente. El servidor será el encargado de administrar la información mientras que el cliente consultará los datos al servidor y realizará modificaciones y consultas.

Formato archivo del consulado:
Los datos de cada trámite se separan por un caracter de nueva línea '\n'.
El formato utilizado es el siguiente:

```
<cod_tramite>|<fecha>|<disponibilidad>|[|<id_cliente>|<id_cliente>...|<id_clien
te>]\n
```

Los campos "`cod_tramite`" y "`fecha`" puede contener cualquier combinación de caracteres alfanuméricos, comas, puntos y guiones, tienen un tamaño fijo de 8 y 5 caracteres respectivamente, mientras que el resto de los campos sólo pueden contener caracteres numéricos de un tamaño máximo de 10 caracteres.
“id_cliente” es el DNI del cliente que hace la reserva

Servidor

Debe recibir como argumento de entrada el puerto por el que escuchará conexiones entrantes y el nombre de un archivo de texto del que se cargarán los datos existentes sobre los trámites y sus horarios disponibles. En dicho archivo se guardarán los datos nuevamente al finalizar la ejecución.

Ejecución del servidor: `./servidor <puerto> <ruta_archivo_consulado>`

Al iniciar el programa, se debe leer el archivo con la información de los turnos para trámites. A cada item leído del archivo le deberá asignar un ID numérico (número natural), válido para la sesión actual. El archivo debe existir.
Los datos de los turnos deberán cargarse a un contenedor (utilizar el contenedor que mejor se ajuste a las necesidades del trabajo) el cual deberá ser un TDA implementado íntegramente por el alumno. **No esta permitido utilizar librerías externas ni headers suministrados por materias anteriores**.
El servidor debe poder recibir conexiones simultáneas y atender los pedidos de los distintos clientes y cerrarse al ingresar por entrada estándar el caracter 'q'.

El programa enviará información respondiendo a las peticiones realizadas por los clientes. Todas las peticiones deben tener una respuesta. Las posibles respuestas del servidor son:

```
ok
error
<listado de tramites/dia>
<listado de reservas>
```

Para enviar el listado de turnos disponibles se utilizara el siguiente formato:

```
<id_item_sesion>\t<cod_tramite>\t<fecha>\t(<disponibilidad>)
```

Para enviar el listado de reservas se utilizará el siguiente formato:

```
<id_cliente>\t<id_item_sesion>\t<cod_tramite>\t<fecha>
```

id_item_sesion: es el identificador del tramite/fecha asignado al obtener los datos del archivo. Este valor deberá ser formateado para ocupar 4 caracteres (Ej. 0001).

Si el servidor recibe un comando que no reconoce o que está mal formado, envía al cliente el mensaje “error”.
Si no hay datos para satisfacer el pedido del cliente, también envía el mensaje “error”.
El mensaje “ok” será enviado en respuesta a una reserva realizada en forma exitosa.
Se debe actualizar la disponibilidad para los cupos de los trámites al realizar una reserva exitosa.
El servidor retorna 0 siempre.

Cliente

Debe recibir como argumento de entrada la IP del servidor y el puerto en el que el mismo escucha conexiones.

Ejecución del cliente: `./cliente <ip del servidor>:<puerto del servidor>`

El cliente se conectará al servidor, y podrá realizar las siguientes acciones por entrada estándar:

```
L
```
Pedir el listado de trámite/fecha disponibles: id, código del trámite, fecha y disponibilidad

```
R <id_cliente> <id_item_sesion>
```
Reservar turno para un trámite en una fecha determinada

```
C <id_cliente>
```
Consultar los turnos reservados

```
S
```
Finaliza la ejecución del cliente con valor de retorno 0.

Sólo se podrá pedir una reserva por trámite/día por cliente.
El cliente deberá imprimir por pantalla todas las respuestas recibidas del servidor.
En caso de detectar un cierre de conexión por parte del servidor, el cliente deberá finalizar con valor de

retorno 1.

Protocolo a utilizar entre el cliente y el servidor

La comunicación entre el cliente y el servidor se llevará a cabo utilizando una cadena de caracteres finalizada con el caracter de nueva línea '\n'.

El servidor enviará información respondiendo a las peticiones realizadas y el cliente debe imprimirlas en pantalla.
`<data>\n`

Ej: En el caso de responder con el mensaje “error”:
`error\n`

El cliente, en cambio, enviará información que el servidor deberá interpretar, por lo tanto enviará el comando y los parámetros que requiera, separando todos los campos con pipes '|', finalizando con el caracter de nueva línea '\n'.

`<comando>[|<parametro_1>|<parametro_2>...|<parametro_n>]\n`

Ej: En el caso de reservar un turno para el tramite con ID 0001: `R 30111222 0001`
`R|30111222|1\n`

Existe un caso particular que es el listado de turnos: como el protocolo finaliza con un caracter de nueva línea, no pueden enviarse caracteres de nueva línea dentro del mensaje, por lo cual, la información de cada turnos será separada por el byte 0x03 ((char)3), y finalmente al terminar de armar el listado, se debe agregar el caracter de nueva línea, como indica el protocolo.

Para poder imprimir en pantalla un item de tramite/fecha por línea, es necesario que del lado del cliente se parsee el listado reemplazando los caracteres 0x03 por el caracter de nueva linea. Ejemplo:

El servidor envía el listado de esta forma:

```
0001\tCITIZ153\t05OCT\t(23)0x030002\tCITIZ153\t06OCT\t(10)0x030003\tCITIZ153\t0
7OCT\t(1)0x030004\tPASS0134\t20NOV\t(6)0x030005\tPASS0134\t23NOV\t(47)0x030006\
tPASS0134\t25NOV\t(5)0x030007\tVISA0115\t14OCT\t(8)0x03\n
```

Y el cliente, parsea el mensaje reemplazando los 0x03 por \n:
`0001\tCITIZ153\t05OCT\t(23)\n0002\tCITIZ153\t06OCT\t(10)\n0003\tCITIZ153\t07OCT`
`\t(1)\n0004\tPASS0134\t20NOV\t(6)\n0005\tPASS0134\t23NOV\t(47)\n0006\tPASS0134\`
`t25NOV\t(5)\n0007\tVISA0115\t14OCT\t(8)\n\n`

Luego, envía a la salida estándar el listado:

```
0001    CITIZ153    05OCT    (23)
0002    CITIZ153    06OCT    (10)
0003    CITIZ153    07OCT    (1)
0004    PASS0134    20NOV    (6)
0005    PASS0134    23NOV    (47)
0006    PASS0134    25NOV    (5)
0007    VISA0115    14OCT    (8)
```

Ejemplo de archivo del consulado:

```
CITIZ153|05OCT|23||30111222|23555666|23555999|33225566
CITIZ153|06OCT|10|
CITIZ153|07OCT|1||30111222|23555666|23555999|33225566
PASS0134|20NOV|6|
PASS0134|23NOV|47|
PASS0134|25NOV|5|
VISA0115|14OCT|8||33669988
```

Ejemplos de Ejecución

Servidor

```
# ./servidor 4321 consulado.txt
```

indica al servidor que debe escuchar conexiones entrantes en el puerto 4321, debe cargar los datos contenidos en el archivo consulado.txt y luego, al leer “q” por stdin, escribir los datos actualizados en el mismo archivo y terminar su ejecución.

Cliente

```
# ./cliente 127.0.0.1:4321
```

indica al cliente que debe conectarse con el servidor que se encuentra escuchando en la dirección ip 127.0.0.1, en el puerto 4321.

Suponiendo que el servidor cuenta con los datos listados en el ejemplo de archivo del consulado:
(en rojo lo ingresado por entrada estándar en el cliente)
(en negro la petición del cliente, formateada para que pueda interpretarla el servidor)
(en azul las respuestas del servidor que el cliente envía a la salida estándar)

- **Petición 1**

  L
  L
  El servidor debe retornar el listado, con el formato anteriormente explicado:

```
0001    CITIZ153    05OCT    (23)
0002    CITIZ153    06OCT    (10)
0003    CITIZ153    07OCT    (1)
0004    PASS0134    20NOV    (6)
0005    PASS0134    23NOV    (47)
0006    PASS0134    25NOV    (5)
0007    VISA0115    14OCT    (8)
```

- **Petición 2**

  `R 23333333 0003`

  `R|23333333|3`

  El cliente reserva un turno para el trámite CITIZ153 del dia 7 de octubre, el servidor debe actualizar los datos y enviar una respuesta

  `ok`

- **Petición 3**

  `R 23333333 0007`

  `R|23333333|7`

  El cliente reserva otro turno para otro trámite, el servidor debe actualizar los datos y enviar una respuesta

  `ok`

- **Petición 4**

  `R 23333333 0009`

  `R|23333333|9`

  El cliente reserva un turno para un trámite/fecha que no existe, el servidor debe retornar el mensaje “error”

  `error`

- **Petición 5**

  `R 23333334 0003`

  `R|23333334|3`

  El cliente reserva un turno para un trámite/dia que no tiene disponibilidad, el servidor debe retornar el mensaje “error”

  `error`

- **Petición 6**

  `R 23333333 0007`

  `R|23333333|7`

  El cliente reserva un turno para un trámite/dia, y ya tenía un turno reservado para ese trámite en ese dia, el servidor debe retornar el mensaje “error”

  `error`

- **Petición 7**

  `C 23333333`

  `C|23333333`

  El cliente consulta las reservas realizadas, el servidor debe enviar los datos consultados, si no hay

reservas con ese id de cliente, se debe retornar el mensaje “error”

```
23333333   0003  CITIZ153    07OCT
23333333   0007  VISA0115    14OCT
```

- **Petición 8**

  L

  L

  El servidor debe retornar el listado de los trámites/dia disponibles:

```
0001    CITIZ153    05OCT    (23)
0002    CITIZ153    06OCT    (10)
0003    CITIZ153    07OCT    (0)
0004    PASS0134    20NOV    (6)
0005    PASS0134    23NOV    (47)
0006    PASS0134    25NOV    (5)
0007    VISA0115    14OCT    (7)
```

- **Petición 9**

  S

  S

  Cierra la aplicacion.

Estado final del archivo del servidor

Finalmente cuando se cierra el servidor, al leer “q” por entrada estándar, se deben grabar los datos actualizados en el archivo:

```
CITIZ153|05OCT|23||30111222|23555666|23555999|33225566
CITIZ153|06OCT|10|
CITIZ153|07OCT|0||30111222|23555666|23555999|33225566|23333333
PASS0134|20NOV|6|
PASS0134|23NOV|47|
PASS0134|25NOV|5|
VISA0115|14OCT|7||33669988|23333333
```

Restricciones

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

1. El sistema debe desarrollarse en ISO C (C99) .
2. Está prohibido el uso de variables globales.
3. Los sockets a utilizar deben ser bloqueantes y trabajar bajo protocolo TCP.
4. Los threads y mutexes a utilizar deben ser de la implementación de hilos de POSIX, conocida como pthreads [1].
5. El empleo de sockets y threads debe ser abstraído en TDAs según las recomendaciones indicadas por la cátedra. La resolución del trabajo debe estar orientada al uso de TDAs (tipos de datos

abstractos), que provean una interfaz que oculte la implementación; está terminantemente prohibido violar el concepto de abstracción.

6. Debe utilizarse la cantidad necesaria de memoria, apelando a la reserva en forma dinámica cuando sea necesaria; no se considerará correcto utilizar memoria en exceso.

Referencias

[1] http://en.wikipedia.org/wiki/POSIX_Threads