Redes y Comunicaciones

Trabajo Práctico

2do Cuatrimestre de 2020

Documento	Nombre	Evaluación Individual
42.684.584	Ignacio Cazcarra	
34.182.053	Sergio Avalos	
42.043.432	Franco Borsani	
42.200.255	Franco Aguirre	
41.767.737	Tomas Rivera	
Evaluación Trabajo		

Objetivo

Desarrollar en lenguaje C o C++ un sistema de reserva de pasajes en ómnibus basado en dos aplicaciones que se ejecutarán en modo consola, una con el rol de servidor y otra con el rol de cliente, comunicadas por sockets.

El sistema será utilizado por una empresa de transporte de pasajeros que realiza viajes de ida y vuelta entre los siguientes destinos:

- Buenos Aires
- Mar del Plata

Esta empresa cuenta con unidades como para proveer servicios en 3 turnos: mañana, tarde o noche.

Requerimientos funcionales

Servidor

- Inicia y espera la conexión del cliente. Para esto hará las siguientes validaciones:
  a) Cuando un cliente intente conectarse, el servidor deberá verificar que no haya otro cliente conectado. En caso de haber un cliente conectado deberá seguir uno de los siguientes flujos:
     1. Si la sesión del cliente conectado expiró **(*2* minutos sin actividad)**, lo desconecta y acepta la conexión del nuevo.
     2. Si la sesión del cliente conectado está activa, niega la conexión del nuevo cliente enviándole el mensaje *"Solo puede haber un cliente conectado a la vez, por favor inténtelo más tarde"*.
  b) Al aceptar la conexión de un cliente, el servidor deberá pedir un usuario y contraseña, y deberá seguir uno de los siguientes flujos:
     1. Si el usuario y contraseña son correctos, se permite el ingreso al sistema.
     2. A los 3 intentos fallidos, el servidor rechaza la conexión informando al cliente *"Se superó la cantidad máxima de intentos de ingreso"*.
  c) Cuando un cliente sale, el servidor se queda esperando a otro.

- Toda la actividad relacionada con la administración de conexiones y sesiones de usuarios realizada por el servidor debe quedar registrada en un archivo de texto llamado ***sever.log***
  - A modo de ejemplo:

Archivo: server.*log*
Contenido:

2020-09-20_11:50: ==========
2020-09-20_11:50: Inicia Servidor
2020-09-20_11:50: ==========
2020-09-20_11:50: Socket creado. Puerto de escucha: 5000
….

- Las credenciales de los usuarios se consultan desde un archivo de texto con duplas, por ejemplo:

Usuario_1;password_1

Usuario_2:password_2

Usuario_n;password_n

- Los nombres de usuario pueden ser cualquier cadena de hasta 12 caracteres sin espacios (por ejemplo: Pedro, Juan_2020, etc)

- Toda la actividad realizada por cada usuario debe quedar registrada en archivos de texto independientes.
  - Debe haber un archivo por usuario, y cada vez que éste se conecta al servidor y realiza diferentes acciones, estas se van agregando al mismo.
  - Se recomienda identificar claramente el inicio de cada sesión.
  - A modo de ejemplo:

Archivo: *usuario_1.log*
Contenido:

2020-09-20_12:30: ==========
2020-09-20_12:30: Inicia sesión
2020-09-20_12:30: ==========
2020-09-20_12:34: IdServicio - Reserva_A1 2020-09-20_12:36: IdServicio - Libera_C10 2020-09-20_12:40: Cierra sesión

2020-09-20_17:30: ==========
2020-09-20_17:30: Inicia sesión
2020-09-20_17:30: ==========
…

- La información de los servicios se almacena en el servidor utilizando un archivo binario.

- Se debe poder visualizar el contenido total de este archivo binario, convirtiéndolo a un formato legible (puede ser a través de una tercera aplicación, también desarrollada en lenguaje C o C++).

Cliente

- Para conectarse al servidor, primero deberá especificar la dirección IP y el puerto, una vez que el servidor acepte la conexión, deberá introducir un usuario y contraseña.
  En caso que el servidor niegue la conexión, se deberá mostrar el mensaje recibido y mostrar las opciones para conectarse nuevamente.

- Ya conectado al sistema, el cliente dispone de las siguientes opciones:

  - Alta servicio

  - Gestionar pasajes

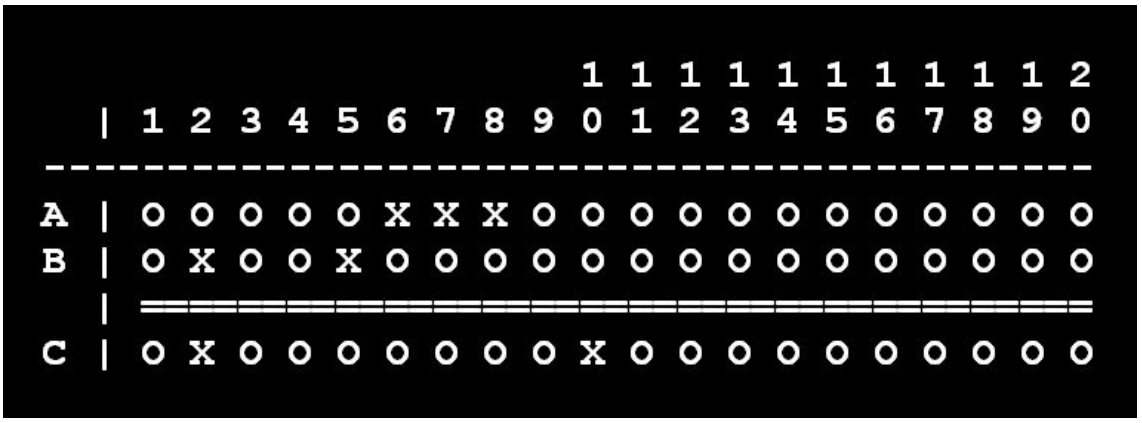
  - Ver registro de actividades

  - Cerrar sesión

- **Alta servicio**: esta opción permite crear un servicio. Para ello el usuario debe ingresar el Origen, la fecha y el turno.
  Con estos datos se consulta al servidor, que deberá verificar si ya está creado este servicio y si no lo está, deberá crearlo (asumiendo el ómnibus vacío). En ambos casos el servidor debe devolver el resultado de la operación solicitada, y el cliente debe notificar adecuadamente al usuario.

- **Reservar pasajes:** esta opción permite reservar pasajes. Para ello el usuario puede realizar consultas por Origen, por fecha, por turno, o por combinación de estas. El servidor devolverá un listado de los servicios que cumplan con dicho criterio.

El usuario puede entonces elegir entre alguno de los servicios disponibles, para realizar la reserva de pasajes, o crear un nuevo servicio (**Alta servicio**).
Si el usuario desea gestionar pasajes en alguno de los servicios disponibles, debe indicar cual, y se presentará por pantalla el esquema del ómnibus según el estado de ocupación, a modo de ejemplo:



Donde cada letra representa:

O = Asiento libre

X = Asiento ocupado

Los micros son todos de un piso, con 20 asientos por fila, dispuestos en un sector de asientos dobles de un lado, el pasillo y una fila de asientos individuales.

Teniendo la disposición de los asientos en pantalla, se presenta un menú con las acciones correspondientes:

- Reservar un asiento
- Liberar un asiento
- Elegir otro servicio
- Volver al menú anterior

- **Ver registro de actividades:** mediante esta función se solicita al servidor que transfiera al cliente el archivo de registro de actividades del usuario activo. Una vez recibido, se muestra por pantalla el contenido del mismo.

- **Cerrar sesión:** el cliente se desconecta del servidor dejándolo libre para recibir nuevas conexiones.

  - Se recomienda utilizar menús para simplificar la selección de las diferentes opciones.
  - La presentación de los datos debe ser clara (borrando la pantalla cuando corresponda, presentando páginas si las opciones son demasiadas, etc.).

Requerimientos no funcionales

Lenguaje de programación C/C++.

Entorno de desarrollo a elección del grupo.

Sistema operativo Linux o Windows, a elección del grupo.

Presentación

19/10/2020: **Consultas**.

01/11/2020 09:59hs: **Entrega del TP vía Campus Virtual**

02/11/2020: **Defensa del TP.**

15/11/2020 09:59hs: **Entrega del recuperatorio del TP vía Campus Virtual**

16/11/2020: **Recuperatorio del TP.**

Normas de entrega.

El trabajo entregado deberá contener un documento incluyendo:

- El enunciado del trabajo práctico (TODO este documento, incluyendo el anexo).
- La estrategia de resolución del trabajo práctico. Es un texto descriptivo de cómo se estructuró la aplicación, cómo se separaron las capas, relaciones entre las entidades, es decir, todo aquello que consideren significativo para explicar la resolución del trabajo.
- El código fuente del proyecto anexado en una carpeta aparte o subido a un repositorio público (Github, GDrive, etc).
- Todas las hojas deben estar numeradas.
- Video (o enlace a video) en formato mp4 mostrando las pruebas del sistema. La especificación de las pruebas a realizar será indicada oportunamente.

El incumplimiento de cualquiera de las normas de entrega implicará la desaprobación del trabajo práctico.

Evaluación:

La Evaluación de los trabajos prácticos contará con una etapa grupal y una individual.

- Grupal: Se realizará un conjunto de pruebas sobre el trabajo presentado por los alumnos en presencia de los mismos. Se deberá aprobar la totalidad de las pruebas.
- Individual: Se realizará una evaluación individual oral o escrita para cada alumno. Los temas a evaluar podrán ser, por ejemplo: preguntas sobre el trabajo práctico, codificación de alguna primitiva o modificación del trabajo práctico, etc.

La nota final del trabajo se calculará en función de las notas obtenidas en forma grupal e individual. La nota grupal será el promedio entre la primera presentación y el recuperatorio (en caso de necesitarlo). Por este motivo, SOLO deberán presentarse aquellos grupos que hayan concluido TODO el trabajo práctico ya que no se harán evaluaciones parciales.

Cuestiones de autoría:

Todo el código debe ser desarrollado íntegramente por cada grupo.

No se permite la reutilización de código de cuatrimestres anteriores o de otras materias.

Ante cualquier duda se deberá consultar con los docentes.

La reutilización de código sin consulta previa será condición suficiente para la desaprobación de la materia.

Anexo – Correcciones

Prueba	Resultado	Comentario

Estrategia de resolución

Aplicación

La aplicación hace uso del protocolo de comunicación TCP / IP. Para el transporte de los datos se utilizó TCP (Protocolo de Control de Transmisión), para pasar un flujo de bytes desde un entorno local (cliente) a un servidor. Haciendo uso de sockets, se pudo vincular la dirección IP a un puerto de escucha mediante el cual accederán los usuarios. Esta comunicación está definida por:

una dirección IP local
una dirección IP remota
el protocolo de transporte TCP
un puerto

El servidor se quedará a la escucha de conexiones, donde cuando un cliente se conecte el mismo se despertará para recibir la información enviada. Ésta trabajará en forma de "bucle" siempre y cuando se envíe y reciba información constantemente (evitando períodos de inactividad). Finalmente, cuando un cliente decida cerrar sesión o se desconecte, el servidor le cerrará el socket y también se finalizan las operaciones de Windows Sockets para los subprocesos (WSACleanup) donde, finalmente, estará nuevamente en modo espera.
La capa de aplicación manejó todo lo relacionado a la codificación y al diálogo entre hosts. La de transporte, con TCP, de la conducción adecuada de los datos, la de Internet, para trabajar con el uso de IPs y direccionamiento, y la de Acceso a la red para establecer las relaciones entre componentes.
Para el trabajo en equipo se utilizó el lenguaje de programación C++ con el IDE CodeBlocks en Windows, el cual nos facilita WinSock, que es una biblioteca de funciones DLL de este sistema operativo para implementar el protocolo de comunicación y trabajar el envío y la recepción de paquetes.
El proyecto está compuesto por 3 aplicaciones: Lector binario, Servidor y Cliente.

Servidor: El servidor se compone por la clase *serverClass* y las clases *escribirArchivo* y *leerArchivo.* La primera se ocupa de configurar el socket del servidor y manejar las peticiones que se realizan desde el lado del cliente. También se encarga de constatar que los usuarios que intentan ingresar al sistema tengan un usuario y contraseña válidos.
La clase *escribirArchivo* se encarga de guardar los datos en archivos y actualizarlos en caso de que sea necesario. Por ejemplo, lo relacionado a los servicios y las butacas ocupadas.

La clase *leerArchivo* lee los archivos y devuelve su contenido. También se utilizó para almacenar métodos que sirven para comparar fechas.

Lector binario: Se encarga de transformar los archivos binarios en un formato legible. Este se utilizó para visualizar el contenido del archivo binario que contenia la información de los servicios.

Cliente: Para el caso del cliente se utilizó solo una clase debido a la estrategia de negocio adoptada: Nuestro grupo decidió usar Codeblocks (el cual solo permite abrir una pantalla a la vez). Por eso, tuvimos que crear un ejecutable para el cliente. La característica del ejecutable es que no compilaba sí estaba compuesto por más de una clase por lo que decidimos dejarlo todo en un mismo archivo. La aplicación de Cliente se encarga de configurar el socket de cliente, establecer conexión con el servidor, gestionar las opciones que puede ingresar el usuario y comunicarse con el servidor para obtener el resultado de las mismas.

Estructura de la aplicación

Almacenamiento de la información (archivos binarios y de texto):
Para el almacenamiento de información no se usaron estructuras o vectores, sino que para esto solo se utilizaron archivos.

- El archivo binario llamado info_servicios contiene los datos de cada servicio junto a las butacas reservadas del mismo. Este es el archivo que se compila luego en el lector binario para obtener la información y el estado de todos los servicios. Este archivo también se utiliza para corroborar que no se cree un servicio con las mismas características cuando se quiere dar de alta un servicio.
- La actividad de cada usuario se almacena en archivos de texto log. Estos archivos son únicos para cada uno de los usuarios.
- La actividad relacionada con la administración de conexiones y sesiones (inicio de sesión, cierre de sesión, puerto de escucha conectado, etc.) se almacena en un archivo de texto llamado server.log
- Cada servicio tiene un archivo binario que contiene el estado de cada una de las butacas. Obviamente, cuando se crea un nuevo servicio también se genera automáticamente un archivo binario para dicho servicio.

Mensajería:
Toda la mensajería ocurre entre el cliente y servidor y sirve para saber qué acción tomar. Hay una función explicada más abajo del lado del servidor llamada manejarPeticiones. Básicamente cuando el usuario elige una opción, se le envía un

mensaje al servidor con el nombre de esa acción (por ejemplo, si el cliente quiere ver el registro de actividades se le envía un mensaje “Registro” al servidor) para que el mismo sepa que devolver. A partir de ahí, empiezan a intercambiar información según el tipo de petición.
Si el cliente sobrepasa los dos minutos sin actividad, el servidor se desconectará del cliente y quedará abierto a nuevas conexiones, de modo que ya no se podrá intercambiar información con el antiguo cliente a menos que este vuelva a ingresar con los datos.

Funciones principales a explicar

-renovacionDeMicrosDisponibles(): Básicamente lo que hace esta primitiva es actualizar el estado de los servicios en el archivo *info_servicios.bin*. Esto ocurre cuando se reserva o se libera una butaca.

-crearServicio(string userName , Server*& servidor): Luego de que el cliente ingrese los datos para crear un servicio (los cuales son verificados como válidos en la clase cliente), se los utiliza para verificar en el archivo “*info_servicios.bin*” (que contiene los servicios) si ya existe. En caso de que no exista, lo crea y lo agrega al archivo *info_servicios.bin*. Si existe, simplemente se le avisa al cliente que ya existía dicho servicio.

-Registro en archivos: Los métodos relacionados a este procedimiento funcionan todos de manera similar. Primero se verifica si el archivo está presente. En caso de que esté se escribirá o modificará sobre el archivo y se escribirá al final del mismo. Caso contrario se creará y se pondrán los datos que sean pasados por parámetro (por lo general será un string). Para ambos casos se trabajarán los archivos como ficheros de salida (ofstream).

-checkUser(Server *&Servidor): Una vez que el cliente establece conexión con el servidor, esta función busca en el archivo “usuarios.dat” si el usuario y la contraseña ingresada por el usuario existen. El usuario puede fallar un máximo de 3 veces a la hora de ingresar el usuario y contraseña. A la 3era vez, se lo desconecta.

-gestionarAsiento(): En el ocurre todo lo relacionado con el transporte de información acerca de la reserva y liberación de asientos. El cliente le envía una posición y según la operación (liberar o reservar) el servidor lee el archivo del autobús y se fija si dicha posición está disponible. Si está disponible, actualiza el asiento.

-ManejoDePeticiones para los menús: Básicamente, cada vez que el usuario elige una opción, ésta se le envía como petición al servidor. Este último recibe la petición, la procesa y comienza a interactuar con el socket del cliente mediante el transporte de información. Por ejemplo, a la hora de reservar un asiento el cliente le envía al servidor la posición que quiere reservar y el servidor se fija en el archivo del autobús elegido si está disponible.

-Iniciar butacas (para inicio del servicio): Se encarga de inicializar un autobús vacío (todos sus asientos disponibles). Se utiliza cuando se da de alta un servicio.

-getIdServicio: A la hora de guardar en el log de cada usuario las butacas que reservaban o liberaban, se utilizaba una id para poder identificar al autobús. Para ellos se utilizó la fecha sin espacios ni guiones, la primera letra del origen (‘B’ de “Buenos Aires” o ‘M’ de Mar del plata) y la primera letra del turno (‘M’ de mañana, ‘T’ de tarde o ‘N’ de noche). Ejemplo: Para el servicio 23/11/2020, origen Buenos Aires turno noche, la id sería 23112020BN.

Drive con el código y el video de testeo

Aclaración: al principio del video se nombra mal a uno de los integrantes. El apellido es **Rivera**, no Silvestre.
https://drive.google.com/drive/folders/1uMZQ-t2jgnBD3a4W7ryzB-kSd6qq3Mye

Alternativa para el video (Youtube):
https://www.youtube.com/watch?v=mlQjkctfqgU&ab_channel=IgnacioCazcarra