Redes y Comunicaciones

Trabajo Práctico

2do Cuatrimestre de 2020

Documento	Nombre	Evaluación Individual
36161871	Martinez Carlos Sebastian	
40893218	Ponzo Marianela	
36168089	Rodriguez Julieta	
41716615	Velasquez Rojas Matias	
37009496	Zalazar Madeo Diego Leandro	
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.

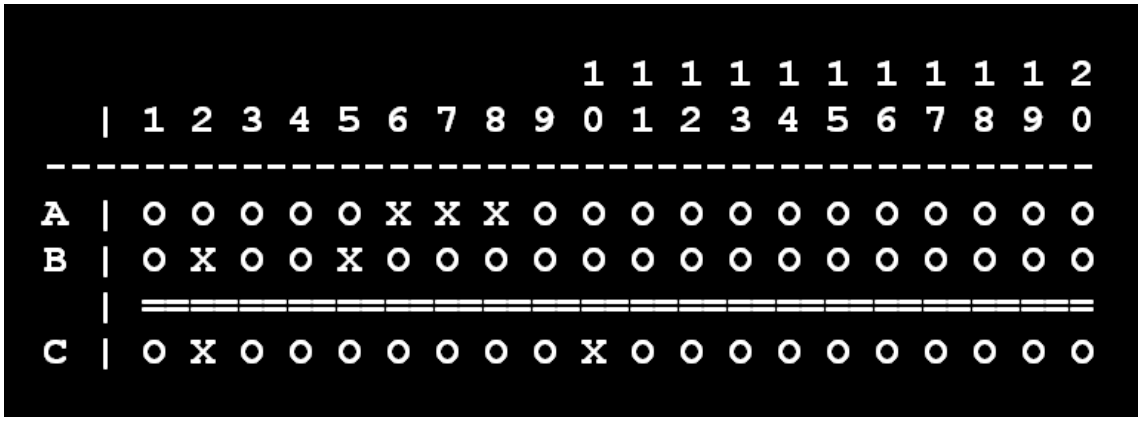
- Ya conectado al sistema, el cliente dispone de las siguientes opciones:
  - Alta servicio
  - Gestionar pasajes
  - Ver registro de actividades
  - Cerrar sesión

- **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 del trabajo práctico

Librerías

Para el desarrollo del presente trabajo práctico utilizamos el lenguaje de programación C y las siguientes librerías:

- **Stdio.h, stdlib.h, string.h**: Son archivos de cabecera que contienen las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones, estándar, de entrada y salida, así como la definición de tipos necesarias para dichas operaciones. Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras.
- **unistd.h:** La interfaz definida por esta librería típicamente se compone en gran parte de la llamada al sistema envoltura funciones tales como fork, pipey I / O primitivas (read, write, close, etc.).
- **sys/types.h:** Este archivo de encabezado contiene definiciones de varios tipos de datos utilizados en las llamadas al sistema
- **sys/socket.h:** Definirá la estructura sockaddr, que se usa para definir una dirección de socket que se usa en las funciones bind () , connect () , getpeername () , getsockname () , recvfrom () y sendto () .
- **netinet/in.h:** Definirá la estructura sockaddr_in se utiliza para almacenar direcciones para la familia de protocolos de Internet. Los valores de este tipo se deben convertir en struct sockaddr para su uso con las interfaces de socket definidas en este documento.
- **ctype.h:** Es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C diseñado para operaciones básicas con caracteres. Contiene los prototipos de las funciones y macros para clasificar caracteres.
- **time.h:** Relacionado con formato de hora y fecha es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C que contiene funciones para manipular y formatear la fecha y hora del sistema.

El sistema operativo elegido para la implementación del proyecto fue **Linux** y utilizamos el editor de código fuente **Visual Studio Code**.

Estructura de la aplicación

Para un mejor manejo de la aplicación, dividimos las funciones en los siguientes archivos:

Servicios.c & Servicios.h

- Definición de Servicio como struct:

```
typedef struct
{
    int id;//Identificador único del servicio
    int fecha;//Fecha formada por: anio * 10000 + mes * 100 + dia;
    int turno;//Entero que representa [1]Mañana[2]Tarde[3]Noche
    int partida;//Entero que representa 1]Mar del Plata[2]Bs As
    int asientos[COLUMNAS][FILAS];//Matriz de enteros, que representa
                                  [1]Reservado [0]Liberado
} Servicio;
```

- Interacción con servicios guardados: Búsqueda, modificación y guardado de servicios en el archivo binario

libServer.c & libServer.h

- Iniciar socket & conexión: iniciarServerSocket & aceptarCliente
- Envío de información: enviarTexto & enviarNumero (y enviarAsientos)
- Recepción de información: recibirNumero & recibirTexto
- Funciones para el login
- Gestión de nuevos servicios y reservas
- Funciones para registro de actividades: Guardado de información en archivo txt
- Envío de archivos (enviarLogCliente)

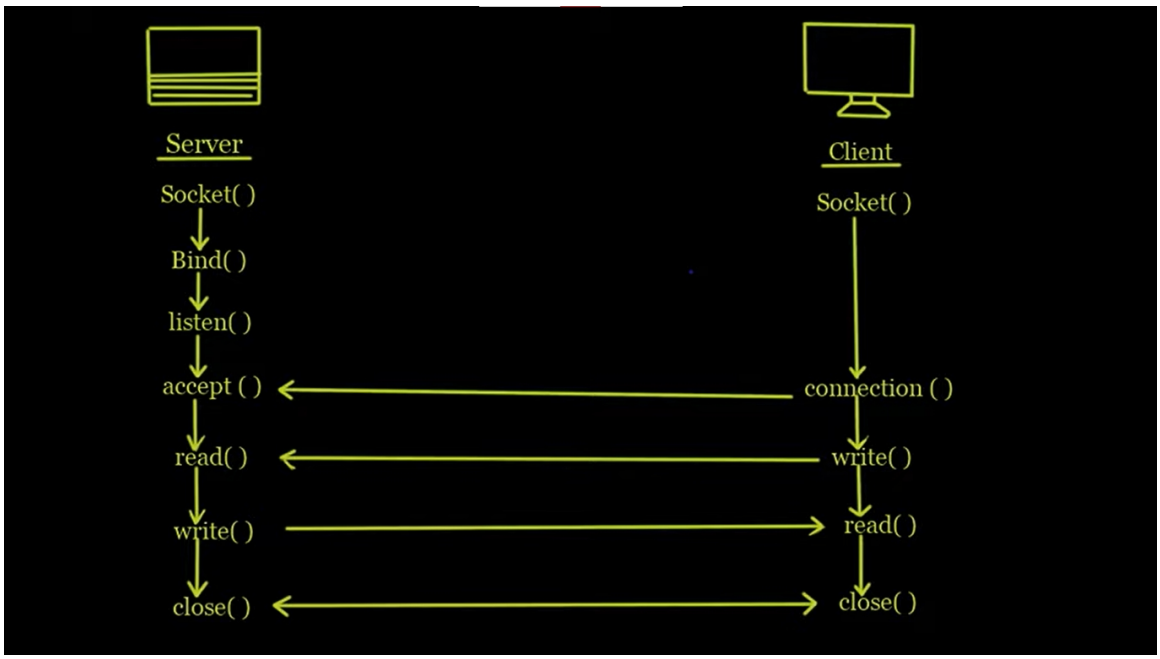
libClient.c & libClient.h

- Iniciar socket & conexión: inicarClientSocket
- Envío de información: enviarTexto & enviarNumero (y enviarAsientos)
- Recepción de información: recibirEImprimirTexto & recibirEImprimirNumero (recibirAsientos + recibirTexto + recibirEImprimirServicio)
- Funciones para el login
- Gestión de nuevos servicios y reservas

- Funciones para registro de actividades: Guardado de información en txt
- Recepción de archivos (solicitarLogCliente)

Relaciones entre las entidades

En líneas generales la estructura de la aplicación utilizada entre las entidades de Cliente y Servidor fue la representada en el siguiente gráfico:



Flujo de la información

Para el funcionamiento de la aplicación y el control del pasaje de información entre el cliente y el servidor procedimos de la siguiente manera:

En primer instancia definimos los siguientes procesos que deben realizarse en el servidor para responder a las distintas demandas del cliente:

N°	Nombre	Parámetros	Respuesta del servidor
0	Salir		- char[]: “Sesión Finalizada”
1	AltaServicio	- int: partida - int: dia - int: mes - int: anio - int: turno	- char[] respuesta: “Servicio creado”/”Ya existe el servicio” - int Idervicio

2	MostrarListadoServicios	- int: partida - int: dia - int: mes - int: anio - int: turno	- Servicio arrayServicios[]
3	MostrarAsientos	- int: idServicio	- int asientos[ ][ ] La matriz contiene 1 (si está reservado) y 0 (si está ocupado)
4	GestionarAsientos	- int: idServicio - int: fila - int: columna - int: operacion	- int: reserva 0: Se hizo el cambio 1: No se hizo el cambio
5	VerRegistroDeActividades	- char[]: Usuario	- int cantidad: numero de líneas por renderizar - char[] registro: linea de texto del file.log

Como primer paso tuvimos que realizar la conexión por socket, esto nos permitió intercambiar diferentes flujos de datos, de manera segura y ordenada, de esta forma podemos conectar el lado del servidor(socket) y del cliente con el fin de enviar datos y hacer consultas sobre los servicios que se van solicitando.

Una vez definidos implementamos funciones dentro del archivo **libServer** para responder a los mismos y generamos una estructura **switch-case** en el main del archivo servidor.c donde se recibirán las peticiones del cliente y se llamará a cada proceso según el valor recibido en los mensajes **recibirOperación()**.

Teniendo en cuenta que para el correcto funcionamiento de la comunicación por sockets es necesario que para cada función **send()** en un proceso cliente o servidor debe existir un mensaje **recv()** en el otro proceso, implementamos en el archivo **libClient.c** funciones simétricas a las

implementadas para el servidor, de modo que se respete la cantidad y orden de los mensajes enviados en cada proceso, así como el tipo de dato que se está comunicando.

Con el objetivo de iniciar la aplicación se ejecuta el archivo servidor indicando como parámetro el puerto donde debe escuchar el proceso. Este crea un socket, lo inicializa y pone a la espera de clientes.En el momento que un usuario quiere acceder a los servicios ejecuta un proceso cliente, indicado la dirección ip de la computadora donde está ejecutando el servidor (en este caso usamos una **dirección de Loopback**) y el puerto donde está a la escucha.

En este punto empieza la comunicación entre procesos. Tanto el cliente como el servidor llamar una función **login()**, donde se establece el pasaje de mensajes necesarios para validar el usuario y contraseña. Si esta validación falla **3** veces el cliente es **desconectado** y el servidor vuelve a la espera de nuevos clientes.

Otro motivo de desconexión del cliente es por tiempo de espera. Para lograrlo fue necesario setear la opción **SO_RECVTMEO** y el una estructura **timeval** que indica de espera al crear el socket de servidor. Estos parámetros indican el tiempo que puede quedar bloqueada una función de entrada (read o recv) antes de retornar un valor de error e interrumpir el proceso. En la función main del archivo servidor.c se declara una variable para manejo de errores que debe ser un parámetro pasado por referencia a todos los llamados a función que encapsulen algún llamado a **recv()** o **read()** para, en caso de ocurrir algún error por tiempo de espera, se pueda modificar su valor en base a este gestionar el manejo de error en el proceso principal.

Una vez aceptado un cliente el flujo de información de la aplicación lo marca la función **menú()** detallada en el archivo **libClient.c**. Es en esta donde se muestran los menús y “formularios” que debe completar el cliente según lo que requiera del servidor. Según las elecciones que realice dentro de los menús, la aplicación cliente le indica al servidor qué función debe ejecutar enviando

un mensaje **llamarOperacion()**. En este punto ambos procesos (cliente y servidor) sincronizan las funciones que ejecutan para responder a las necesidades del usuario. Después de finalizada la

ejecución de ambas funciones, el cliente vuelve al flujo del menú para requerir otros servicios, y el servidor queda a la escucha para responder ante nuevas operaciones.

De esta manera se siguen comunicando hasta que el cliente envía el valor 0 en **llamarOperacion()**, lo que significa que quiere cerrar la sesión. En ese caso se desconecta y finaliza el proceso cliente, y el servidor queda a la escucha para recibir nuevos clientes.

Transferencia de Datos

Las principales funciones desarrolladas en la aplicación vinculadas a la transferencia de datos son:

Nombre	Librería utilizada	Función utilizada	Descripción
recibirNumero()	<sys/socket.h>	recv(*cliente, &valor, 4, 0);	La función recibirá un mensaje desde un conector en modo conexión o en modo sin conexión. Se utiliza con sockets conectados porque no permite que la aplicación recupere la dirección de origen de los datos recibidos.
recibirTexto()		recv(*cliente, m, 1024, 0);	
enviarTexto()		send(*cliente, &num, 1024, 0);	Esta rutina es utilizada para enviar datos sobre un canal de comunicación tanto del lado del cliente como del lado servidor de la aplicación.
enviarNumero()		send(*cliente, &num, 4, 0);	
enviarArrayServicio()		write(*cliente, num, 16);	La función intentará escribir nbyte bytes

			desde el búfer al que apunta buf en el archivo asociado con el descriptor de archivo abierto, fildes .

Para poder visualizar el contenido total del archivo binario, convirtiéndolo a un formato legible se desarrolló una aplicación creada en **app\leerArchivoBinario.c** donde se lee la información del archivo "servicios.bin" ubicado en **app\servicios.bin** y se visualiza por consola siguiendo el siguiente formato:

```
ID: 1
Fecha: 1/1/2020
Origen: Mar del Plata
Turno Mañana
10000000000000000000
10000000000000000000
10000000000000000001
```

Id: Identificador único del servicio

Fecha: Fecha formada por anio, mes, dia

Origen: Entero que representa 1]Mar del Plata[2]Bs As

Turno: Entero que representa [1]Mañana[2]Tarde[3]Noche

Asientos: Matriz de enteros, que representa [1]Reservado [0]Liberado