Guía de ejercicios para los Trabajos Prácticos

1- Chat Multicliente con Sockets:

Crea un servidor de chat que permita la conexión de múltiples clientes.

Utiliza ServerSocket para aceptar conexiones entrantes de clientes.

Cada cliente debe ejecutarse en su propio hilo (thread).

Utiliza un semáforo para sincronizar la emisión y recepción de mensajes entre clientes.

2- Sistema de Reservas de Asientos en un Cine:

Diseña un servidor que simule un sistema de reservas de asientos en un cine.

Los clientes pueden solicitar la disponibilidad de asientos y realizar reservas.

Utiliza ServerSocket para aceptar las solicitudes de los clientes y gestiona las reservas utilizando semáforos para evitar conflictos.

3- Juego Multijugador en Red:

Crea un juego multijugador simple, como un juego de preguntas y respuestas o un juego de adivinanza.

Los clientes se conectan al servidor para participar en el juego.

Utiliza threads para gestionar la lógica del juego para cada cliente y semáforos para evitar condiciones de carrera.

4- Juego de TaTeTi Multijugador en Línea:

Desarrolla un juego de TaTeTi que permita a múltiples (2) jugadores jugar partidas en línea.

Los jugadores pueden conectarse para colocar su símbolo asignado y jugar en tiempo real.

Utiliza sockets para la comunicación entre los jugadores, threads para gestionar las partidas y semáforos para evitar movimientos concurrentes en el tablero.

5- Simulación de Elecciones Presidenciales:

Diseña un sistema que simule el proceso de elecciones presidenciales en un país ficticio. El sistema debe incluir los siguientes componentes:

Servidor de Elecciones:

Implementa un servidor que gestiona el proceso de votación.

Utiliza un ServerSocket para aceptar conexiones de votantes.

Cada votante se ejecuta en su propio hilo (thread) para emitir su voto.

Utiliza semáforos para garantizar que cada votante pueda emitir un voto y para llevar un registro del número de votos emitidos.

Votantes:

Cada votante es un cliente que se conecta al servidor de elecciones a través de sockets.

Los votantes eligen un candidato presidencial y emiten su voto.

Cada votante debe ser un thread independiente para simular la emisión de votos concurrentes.

Conteo de Votos:

El servidor de elecciones debe contar los votos emitidos por cada candidato y determinar al ganador.

Utiliza semáforos para evitar condiciones de carrera durante el conteo de votos.

El servidor debe enviar el resultado de las elecciones a todos los clientes al final del proceso.

Interfaz de Resultados:

Crea un cliente especial que actúe como interfaz de resultados para que los votantes y observadores puedan ver el progreso y el resultado de las elecciones.

6- Simulador de Penales:

Diseña un simulador de penales que permita a los jugadores y porteros participar en una sesión de tiros de penal. El ejercicio implica la gestión de turnos, el cálculo de resultados y la interacción entre jugadores y porteros. El sistema consta de los siguientes componentes:

Servidor de Penales:

Implementa un servidor que gestiona la sesión de tiros de penal.

Utiliza ServerSocket para aceptar conexiones de jugadores y porteros.

Usa threads para gestionar los turnos y la ejecución de tiros de penal.

Jugadores:

Cada jugador es un cliente que puede solicitar el turno para patear penales.

Los jugadores envían sus tiros al servidor y reciben retroalimentación sobre si el tiro fue un gol o no.

Utiliza sockets para la comunicación entre los jugadores y el servidor.

Porteros:

Cada portero puede aceptar o rechazar solicitudes de jugadores para patear penales.

Los porteros intentan detener los tiros de penal de los jugadores y envían retroalimentación sobre si lograron detener el tiro.

Utiliza sockets para la comunicación entre los porteros y el servidor.

Gestión de Turnos:

El servidor debe administrar los turnos de los jugadores y porteros.

Implementa semáforos para garantizar que un jugador pueda patear y un portero pueda detener un tiro en el momento adecuado.

Puntuación y Resultados:

Lleva un registro de la puntuación de los jugadores y porteros y muestra los resultados en tiempo real.

Utiliza sockets para transmitir los resultados a los jugadores y porteros.

7- Sistema de Gestión de Estacionamiento

Diseña un sistema de gestión de estacionamiento que simule la administración de plazas de estacionamiento en un estacionamiento público. El ejercicio implica la gestión de plazas, entrada y salida de vehículos, y la sincronización de múltiples vehículos que intentan estacionarse o salir del estacionamiento. El sistema constará de los siguientes componentes:

Servidor de Estacionamiento:

Implementa un servidor que administra las plazas de estacionamiento y controla la entrada y salida de vehículos.

Utiliza ServerSocket para aceptar conexiones de vehículos.

Usa threads para gestionar las solicitudes de estacionamiento y salida concurrentes.

Vehículos:

Cada vehículo es un cliente que puede solicitar una plaza de estacionamiento o la salida del estacionamiento.

Los vehículos envían solicitudes al servidor y reciben confirmaciones sobre la disponibilidad de plazas o la salida.

Utiliza sockets para la comunicación entre los vehículos y el servidor.

Gestión de Plazas:

El servidor debe llevar un registro de las plazas disponibles y asignar plazas a los vehículos que ingresan.

Implementa semáforos para garantizar la sincronización segura de la asignación de plazas y la liberación de plazas al salir.

Informes y Estadísticas:

Implementa un sistema que genere informes y estadísticas sobre la ocupación del estacionamiento.

8- Control de Semáforos en una Calle:

Diseña un sistema que controle los semáforos en una sola calle, en lugar de un cruce de calles. El objetivo es simular la operación de semáforos de tráfico en una única calle. El sistema constará de los siguientes componentes:

Servidor de Control de Semáforos:

Implementa un servidor que controla los semáforos en una calle.

Utiliza ServerSocket para aceptar conexiones de sensores de vehículos y semáforos.

Usa threads para gestionar las solicitudes de control concurrentes.

Sensores de Vehículos:

Los sensores de vehículos son clientes que detectan la presencia de vehículos en la calle.

Envían información al servidor sobre las detecciones de vehículos y solicitan cambios en la señal de semáforo cuando sea necesario.

Utiliza sockets para la comunicación entre los sensores de vehículos y el servidor.

Semáforos:

Los semáforos cambian su señal en respuesta a las solicitudes del servidor.

Deben sincronizarse para garantizar transiciones seguras de las señales.

Gestión de Secuencias de Semáforos:

El servidor programa secuencias de semáforos basadas en las detecciones de vehículos y las reglas de tráfico en una sola calle.

9- Sistema de Control de Ascensor:

Diseña un sistema de control para un edificio con un solo ascensor. El objetivo es gestionar las solicitudes de diferentes pisos y garantizar un funcionamiento eficiente del ascensor. El sistema constará de los siguientes componentes:

Controlador de Ascensor:

Implementa un controlador que gestiona las solicitudes de pisos y el movimiento del ascensor.

Utiliza ServerSocket para aceptar solicitudes de los pasajeros.

Utiliza threads para gestionar las solicitudes concurrentes y el movimiento del ascensor.

Pasajeros:

Cada pasajero es un cliente que puede solicitar una dirección y un piso de destino.

Envían sus solicitudes al controlador del ascensor y esperan la llegada del ascensor.

Utiliza sockets para la comunicación entre los pasajeros y el controlador del ascensor.

Ascensor:

El ascensor debe moverse entre los pisos para recoger y dejar a los pasajeros.

Cambia de dirección según las solicitudes de los pasajeros.

Utiliza semáforos para garantizar un acceso seguro a los pisos y la gestión de las solicitudes.

Planificación de Rutas:

El controlador debe planificar rutas eficientes para recoger y dejar a los pasajeros en diferentes pisos.

Utiliza algoritmos simples de planificación de rutas para lograr un funcionamiento eficiente.