Paso a Paso para Construir el Sistema Bancario Concurrente

Fase 1: Preparación y Configuración Inicial

1. **Comprender el enunciado y los objetivos**
   - Lee detenidamente el documento para entender los requisitos: concurrencia, sincronización, detección de anomalías, y uso de herramientas de Linux y C.
   - Identifica los componentes clave: banco.c, usuario.c, monitor.c, init_cuentas.c, y el archivo de configuración config.txt.
2. **Configurar el entorno de desarrollo**
   - Instala un entorno Linux (puede ser una máquina virtual o nativa) con un compilador GCC y bibliotecas necesarias (pthread, semaphore.h, etc.).
   - Crea un directorio para el proyecto (e.g., SecureBank) y organiza subcarpetas para el código fuente, archivos de datos (cuentas.dat, transacciones.log), y documentación.
3. **Diseñar la estructura del sistema**
   - Dibuja un esquema general del sistema (puedes incluirlo en la memoria):
     - Proceso principal (banco.c) como coordinador.
     - Procesos hijos (usuario.c) para cada usuario.
     - Hilos dentro de cada proceso hijo para operaciones.
     - Proceso independiente (monitor.c) para detección de anomalías.
     - Comunicación mediante tuberías y colas de mensajes.
     - Sincronización con semáforos y mutexes.
4. **Crear el archivo de configuración (config.txt)**
   - Define los parámetros iniciales según el ejemplo del documento:

```
# Limites de Operaciones
LIMITE_RETIRO = 5000
LIMITE_TRANSFERENCIA = 10000
# Umbrales de Detección de Anomalias
UMBRAL_RETIROS = 3
UMBRAL_TRANSFERENCIAS = 5
# Parámetros de Ejecución
NUM_HILOS = 4
ARCHIVO_CUENTAS = cuentas.dat
ARCHIVO_LOG = transacciones.log
```

   - Coloca este archivo en el directorio principal del proyecto.

Fase 2: Desarrollo de los Componentes del Sistema

5. **Implementar init_cuentas.c**
   - Crea un programa que inicialice el archivo binario cuentas.dat con datos de cuentas.
   - Define la estructura Cuenta:

```c
struct Cuenta {
    int numero_cuenta;
    char titular[50];
    float saldo;
    int num_transacciones;
};
```

- Escribe un código que genere cuentas iniciales (e.g., 1001 - John Doe, 5000.00) y las guarde en cuentas.dat en formato binario usando fwrite.

6. **Desarrollar banco.c (Programa Principal)**
   - **Inicialización:**
     - Implementa la función leer_configuracion (como en el documento) para leer config.txt y cargar los parámetros en una estructura Config.
     - Crea un semáforo nombrado con sem_open para controlar el acceso a cuentas.dat.
     - Abre tuberías para comunicación bidireccional con los procesos hijos.
   - **Gestión de usuarios:**
     - Usa un bucle para aceptar conexiones de usuarios (simula solicitudes con un número fijo o entrada por teclado).
     - Por cada usuario, usa fork() para crear un proceso hijo que ejecute usuario.c.
   - **Log:**
     - Abre el archivo transacciones.log y registra operaciones recibidas de los hijos.
7. **Implementar usuario.c (Procesos Hijos)**
   - **Menú interactivo:**
     - Diseña un menú en C con un bucle while que ofrezca: Depósito, Retiro, Transferencia, Consultar saldo, Salir.
     - Lee la opción del usuario con scanf.
   - **Hilos por operación:**
     - Para cada operación seleccionada, crea un hilo con pthread_create que ejecute la lógica correspondiente.
     - Usa mutexes para proteger variables locales y semáforos para acceder a cuentas.dat.
   - **Comunicación:**
     - Envía los detalles de la operación al proceso principal (banco.c) mediante una tubería.
8. **Implementar las operaciones bancarias**
   - **Depósito:** Aumenta el saldo de una cuenta en cuentas.dat.
   - **Retiro:** Verifica el límite (LIMITE_RETIRO) y reduce el saldo si hay fondos suficientes.
   - **Transferencia:** Mueve fondos entre cuentas, respetando LIMITE_TRANSFERENCIA.
   - **Consultar saldo:** Lee y muestra el saldo de la cuenta.
   - Usa sem_wait y sem_post para proteger las actualizaciones en el archivo binario.
9. **Desarrollar monitor.c (Detección de Anomalías)**
   - **Inicialización:**
     - Crea una cola de mensajes con msgget para recibir transacciones desde banco.c.
   - **Análisis:**

        - Lee transacciones en tiempo real y busca patrones sospechosos (e.g., retiros consecutivos > UMBRAL_RETIROS).
    - **Alertas:**
        - Envía alertas al proceso principal mediante una tubería cuando detecte anomalías.

Fase 3: Sincronización y Optimización

10. **Configurar la sincronización**
    - Usa semáforos nombrados para controlar el acceso concurrente a cuentas.dat.
    - Implementa mutexes (pthread_mutex_t) dentro de los hilos de usuario.c para operaciones locales.
    - Prueba con varios usuarios simultáneos para verificar exclusión mutua.
11. **Optimizar el código**
    - Elimina redundancias (e.g., funciones repetidas) y usa funciones modulares.
    - Asegúrate de que no haya bucles innecesarios ni condiciones mal definidas.
    - Compila con gcc -Wall para detectar y eliminar warnings.
12. **Documentar el código**
    - Agrega comentarios claros en cada función explicando su propósito.
    - Ejemplo:

```
// Actualiza el saldo de una cuenta en el archivo binario
void actualizar_saldo(int num_cuenta, float monto) { ... }
```

Fase 4: Pruebas y Entrega

13. **Probar el sistema**
    - Ejecuta init_cuentas.c para generar cuentas.dat.
    - Lanza banco.c y simula varios usuarios realizando operaciones concurrentes.
    - Verifica que monitor.c detecte anomalías y que el log (transacciones.log) se actualice correctamente.
    - Cambia parámetros en config.txt (e.g., LIMITE_RETIRO) y comprueba que el sistema se adapte.
14. **Preparar la memoria técnica**
    - Estructura el PDF con:
        - Portada, índice, introducción.
        - Explicación de la solución (cómo implementaste concurrencia, sincronización, etc.).
        - Esquema del sistema (puedes usar una captura del dibujo inicial).
        - Capturas de pantalla del menú, operaciones, y alertas del monitor.
15. **Preparar la presentación**
    - Ensaya la ejecución del código en vivo, explicando cada componente.
    - Prepárate para responder preguntas del profesor y modificar parámetros en tiempo real.
16. **Entregar los archivos**
    - Código fuente: banco.c, usuario.c, monitor.c, init_cuentas.c.
    - Archivos generados: cuentas.dat, transacciones.log, config.txt.

- Memoria en PDF.
- (Opcional) Script de compilación (Makefile) para facilitar la ejecución.

Consejos Adicionales

- **Modularidad:** Divide el trabajo en funciones pequeñas y reutilizables.
- **Depuración:** Usa printf o herramientas como gdb para rastrear errores.
- **Cronograma:** Asigna tiempos específicos para cada fase (e.g., 2 días para banco.c, 1 día para pruebas).
- **Interfaz:** Agrega mensajes claros en el menú para mejorar la experiencia del usuario.