Universidad Francisco de Vitoria

Escuela Politécnica Superior

Grado en Ingeniería Informática

Práctica Final Parte II

Trabajo presentado por:	Daniel de Pablo Moreno, Claudia López Sanz y Jaime Serres Fernández
Profesor/a:	Marlon Cardenas Bonnet
Fecha:	24/3/2025

Resumen

En este trabajo presentamos la extensión de SecureBank, un simulador de sistema bancario concurrente avanzado, para incorporar gestión de memoria y sistemas de ficheros utilizando el lenguaje C y sistema operativo Linux con el que hemos trabajado en clase. Se gestiona la memoria para optimizar el acceso a los datos, mejorando el rendimiento, y, se gestiona también el sistema de ficheros para registrar las operaciones de los usuarios en directorios individuales. Además, el sistema gestionar las operaciones de entrada/salida de manera eficiente, utilizando buffers para minimizar el impacto de las operaciones frecuentes en disco.

Palabras clave: Sistema bancario, sincronización, gestión de memoria, sistemas de ficheros y operaciones de entrada/salida.

Abstract

In this work, we present the extension of SecureBank, an advanced concurrent banking system simulator, to incorporate memory management and file systems using the C language and the Linux operating system with which we have worked in class. Memory is managed to optimize data access, improving performance, and the file system is also managed to record user operations in individual directories. Also, the system manage intput/outpu operations efficiently, using buffers to minimize the impact of frequent disk operations.

Keywords: Banking system, synchronization, memory management, file systems, and input/output operations.

Índice de contenidos

Índice de figuras

Índice de ilustraciones

1. Introducción

Este documento describe la segunda parte del desarrollo de SecureBank, un simulador de sistema bancario concurrente. En esta fase, se extiende la funcionalidad crea anteriormente en la primera parte de la asignatura, para incorporar la gestión de memoria, sistemas de ficheros y gestión de operaciones de entrada/salida. Específicamente, se gestiona la memoria para optimizar el acceso a los datos mejorando así su rendimiento. El sistema de ficheros se emplea para registrar las operaciones de los usuarios de forma organizada en directorios individuales. Mientras que se gestión las operaciones de entrada/salida de manera eficiente mediante el uso de buffers, minimizando así el impacto de las operaciones frecuentes en disco. El objetivo es integrar estos componentes de manera funcional y coherente, aprovechando los recursos mencionados anteriormente.

1.1. Motivación

La motivación principal de esta segunda parte ha sido nuevamente la gran similitud con el mundo real que tiene esta práctica, ya que el problema de la gestión de memoria es uno de los principales en el día a día en cualquier ámbito y más respecto a la informática. Esta práctica nos ha permitido comprender mas a fondo los desafíos y posibles soluciones en el desarrollo de sistemas que manejan múltiples usuarios y operaciones simultáneamente, como los sistemas bancarios.

1.2. Planteamiento del trabajo

Queremos diseñar un sistema que además de gestionar usuarios y operaciones concurrentes, detectar anomalías y generar informes, utilizando técnicas avanzadas de Linux, también gestione memoria compartida, el sistema de ficheros y las operaciones de entrada/salida.

1.3. Estructura del trabajo

El documento se organiza en secciones que cubren el contexto, objetivos, metodología, desarrollo, resultados, conclusiones y limitaciones, detallando cada etapa del proyecto, además de proponer posibles mejoras en un futuro.

La estructura del trabajo ha sido diseñada para ofrecer una completa comprensión sobre la extensión de SecureBank. A través de este enfoque nos ha permitido contextualizar correctamente el trabajo a realizar, detallar las decisiones de diseño adoptadas y valorar las contribuciones de esta extensión al simulador bancario ya creado anteriormente.

2. Contexto y estado del arte

2.1.Contexto del problema

Los sistemas bancarios modernos requieren manejar transacciones simultáneas de forma segura y eficiente. El tener que gestionar grandes volúmenes de datos y transacciones de manera concurrente plantea desafíos importantes en términos de la gestión eficiente de la memoria, la organización de sistemas de ficheros, la gestión de operaciones de entrada/salida, entre otros problemas. La memoria compartida debe utilizarse de manera segura para evitar condiciones de carrera y garantizar la integridad de los datos. El sistema de ficheros debe ser capaz de almacenar y recuperar grandes cantidades de información de manera eficiente, y las operaciones de entrada/salida deben gestionarse utilizando buffers para minimizar el impacto de las operaciones frecuentes en disco.

2.2.Estado del arte

Existen diversas técnicas y enfoques para abordar los desafíos de la gestión de memoria, sistemas de ficheros y operaciones de entrada/salida en sistemas concurrentes. En cuanto a la gestión a la gestión de memoria, las técnicas de memoria compartida, segmentación y paginación son frecuentemente utilizadas. Para los sistemas de ficheros, las bases de datos y los sistemas de archivos distribuidos son soluciones comunes. Respecto a las operaciones de entrada/salida, el uso de buffers y técnicas de almacenamiento en caché son prácticas estándar para reducir la latencia y mejorar el rendimiento del sistema.

2.3. Conclusiones

El contexto del problema destaca la importancia de una gestión eficiente de los recursos del sistema operativo en aplicaciones concurrentes como los sistemas bancarios. El estado del arte muestra que existen diversas técnicas y herramientas disponibles para abordar estos desafíos.

3. Objetivos concretos y metodología de trabajo

3.1. Objetivo general

El objetivo general de esta segunda parte de la práctica es extender el simulador de sistema bancario concurrente SecureBank para incorporar la gestión avanzada de memoria, sistemas de ficheros y operaciones de entrada/salida, utilizando el lenguaje C y el sistema operativo Linux.

3.2. Objetivos específicos

- Integrar las funcionalidades de la primera parte de la práctica con los nuevos componentes.
- Implementar la gestión compartida para la tabla de cuentas.
- Organizar el sistema de ficheros para el almacenamiento de las transacciones de cada usuario.
- Gestionar las operaciones de entrada/salida de manera eficiente, utilizando buffers.

3.3. Metodología del trabajo

El desarrollo se basa en la programación en C, utilizando gestión de memoria, sistema de ficheros y técnicas de manejo eficiente de operaciones de entrada/salida. Además, se realizaron pruebas exhaustivas para verificar el correcto funcionamiento de cada componente y la integración del sistema en su conjunto. Todo esto se documentó con el objetivo de poder hacer uso del sistema de manera sencilla y sin complicaciones.

4. Desarrollo de la solución

Para el desarrollo de nuestra práctica hemos creado los ficheros “init_cuentas.c”, “banco.c”, “usuario.c” y “monitor.c”, los cuales controlan el flujo de la práctica de la siguiente manera:

4.1. ”init_cuentas.c”

Este archivo es el responsable de inicializar los datos de la cuenta y configurar la estructura inicial. Esto permite que los componentes posteriores operen con un conjunto de datos predefinidos y consistente, lo que garantiza la disponibilidad del sistema para las posteriores interacciones de los usuarios y el procesamiento de transacciones.

4.2. “banco.c”

Este otro fichero es un bucle que espera conexiones de usuarios, actuando como coordinador del sistema bancario y generando procesos hijos para gestionar usuarios. Funciona como el núcleo del sistema y del modelo de concurrencia, garantizando la gestión de múltiples usuarios, manteniendo la integridad de los recursos compartidos, mediante una sincronización decente.

4.3. ”usuario.c”

Este fichero maneja las interacciones a nivel de usuario, encapsulando la lógica para gestionar las operaciones bancarias. Proporciona una interfaz de menú que permite a los usuarios realizar acciones como depósitos, retiros, transferencias y consultas de saldo. Dentro de este archivo, se emplean subprocesos para ejecutar estas operaciones simultáneamente, optimizando el rendimiento y cumpliendo estrictos protocolos de sincronización para evitar inconsistencias en los datos. Este componente conecta las entradas del usuario con los procesos subyacentes del sistema.

4.4. ”monitor.c”

Este último fichero es el encargado de identificar patrones sospechosos. Como pueden ser transferencias o retiros repetitivos.

Este último es el encargado de la vigilancia en tiempo real de las transacciones en todo el sistema. Analiza flujos de datos para detectar patrones sospechosos que indiquen anomalías.

4.5. Orden de compilación y ejecución de los ficheros

Para poder utilizar nuestra practica de SecureBank debemos seguir una secuencia específica a la hora de ejecutar los ficheros.

Antes de nada, debemos ejecutar el siguiente comando en nuestro entorno virtual: “sudo apt install gnome-terminal”, para poder abrir una nueva terminal por cada usuario autenticado. A continuación, debemos compilar “init_cuentas.c” para inicializar el sistema y preparar los datos fundamentales del sistema. Una vez compilado “init_cuentas.c”, debemos compilar “usuario.c” para permitir la interacción del usuario. Después debemos compilar “monitor” para activar el monitoreo en tiempo real, y, finalmente compilamos “banco.c” para activar el bucle de coordinación principal y habilitar las conexiones de los usuarios.

Para ver el funcionamiento solo necesitamos ejecutar “./banco” tras haber sido compilado previamente.

4.6. Notas

Para compilar y poder ejecutar todos los ficheros anteriormente mencionados, debemos hacerlo escribiendo en nuestra consola “gcc -o usuario usuario.c -pthread -lrt”.

- **-pthread:** Enlaza la biblioteca de hilos POSIX para las operaciones que usan hilos.
- **-lrt:** Para los semáforos POSIX.

Para poder iniciar sesión se debe registrar una nueva cuenta previamente o utilizando alguna de las que se muestran a continuación que han sido creadas por nosotros para las fases de prueba.

- Cuenta 8888, nombre: clau.
- Cuenta 7777, nombre: daniel.
- Cuenta 6666, nombre: jaime.
- Cuenta 1234, nombre: andrea.

Además, para poder ver el saldo correctamente y probar el funcionamiento de la práctica deberemos darle a la tecla “Intro” dos veces.

4.7. Diagramas

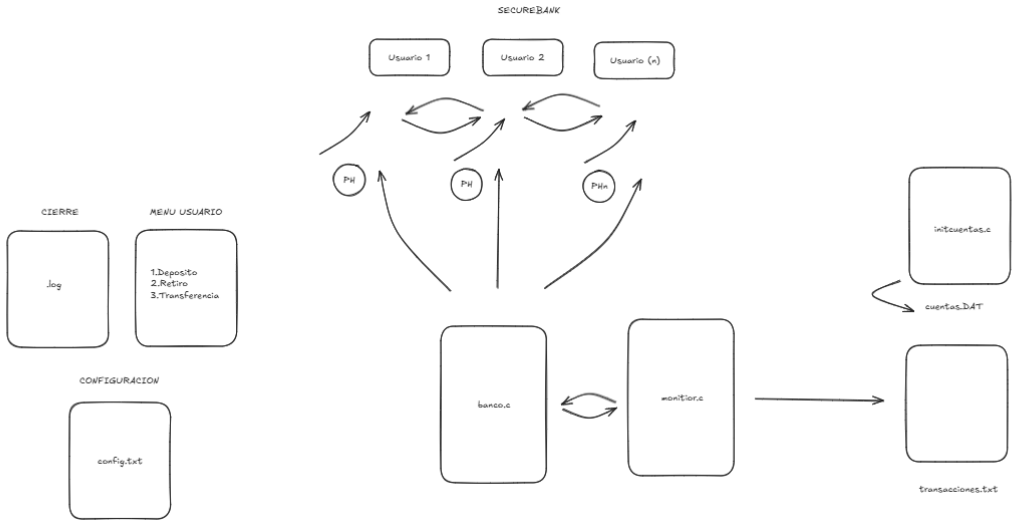

Figura 1: Ejemplo de figura de nuestro sistema

5. Código fuente y datos de prueba

5.1. Código fuente

https://github.com/Jugador9/PracticaSistemasOperativos.git

5.2.Plan de pruebas

Para garantizar la fiabilidad de SecureBank, se diseñó y ejecutó un plan de pruebas. Los siguientes casos de prueba fueron diseñados para cubrir las funcionalidades principales del sistema y los casos extremos.

1. Prueba de detección de anomalías.

Activamos patrones sospechosos para probar la funcionalidad de “monitor.c”, realizando 4 retiros consecutivos (excediendo el umbral establecido de retiros, en este caso, 3 creado en “config.txt”) desde la cuenta 1234. Como resultado, el proceso genera una alerta.

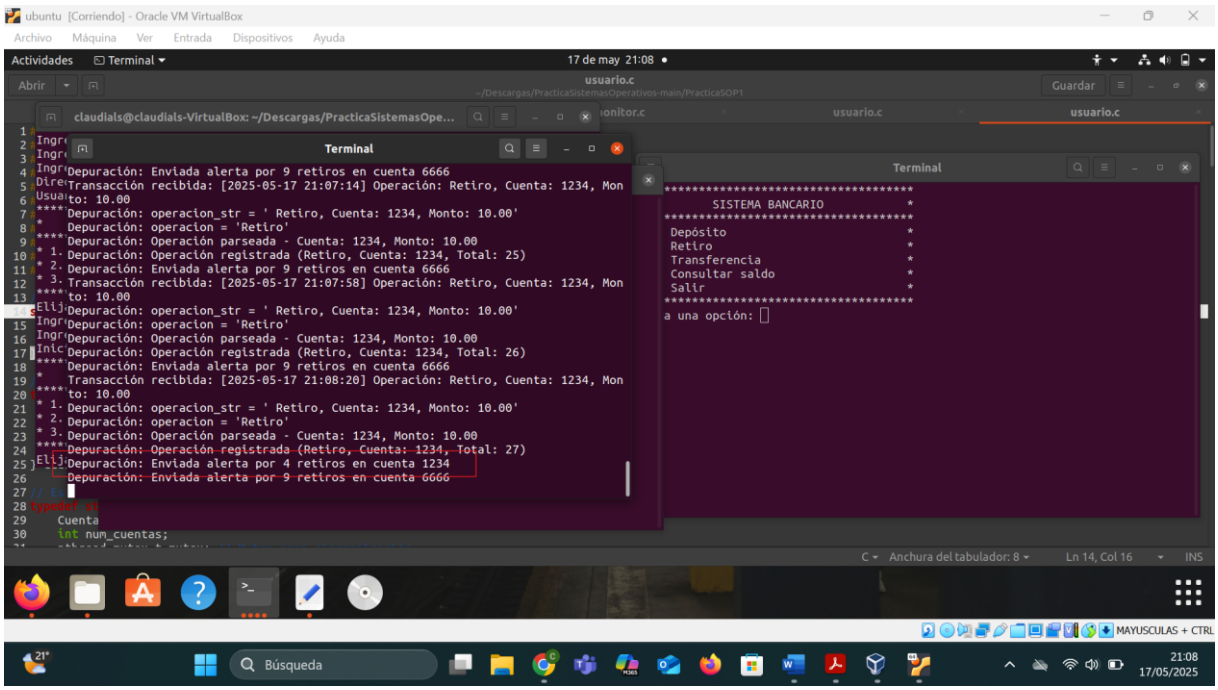

Ilustración 1: Plan de pruebas

2. Prueba de límites de configuración.

Validamos el cumplimiento de los límites operativos, intentando un retiro de 9000 euros (excediendo el límite establecido de retiros, en este caso, 5000 euros, creado en “config.txt”). Como resultado, el sistema rechaza la operación con un mensaje de error.

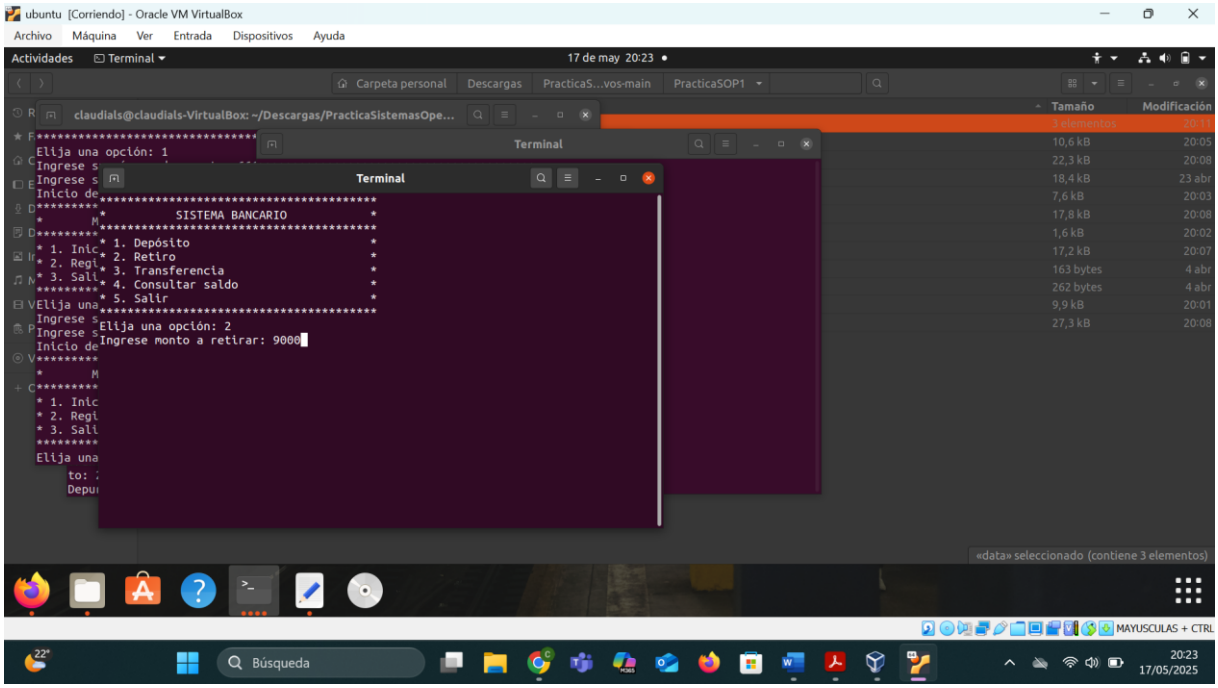

Ilustración 2: Plan de pruebas

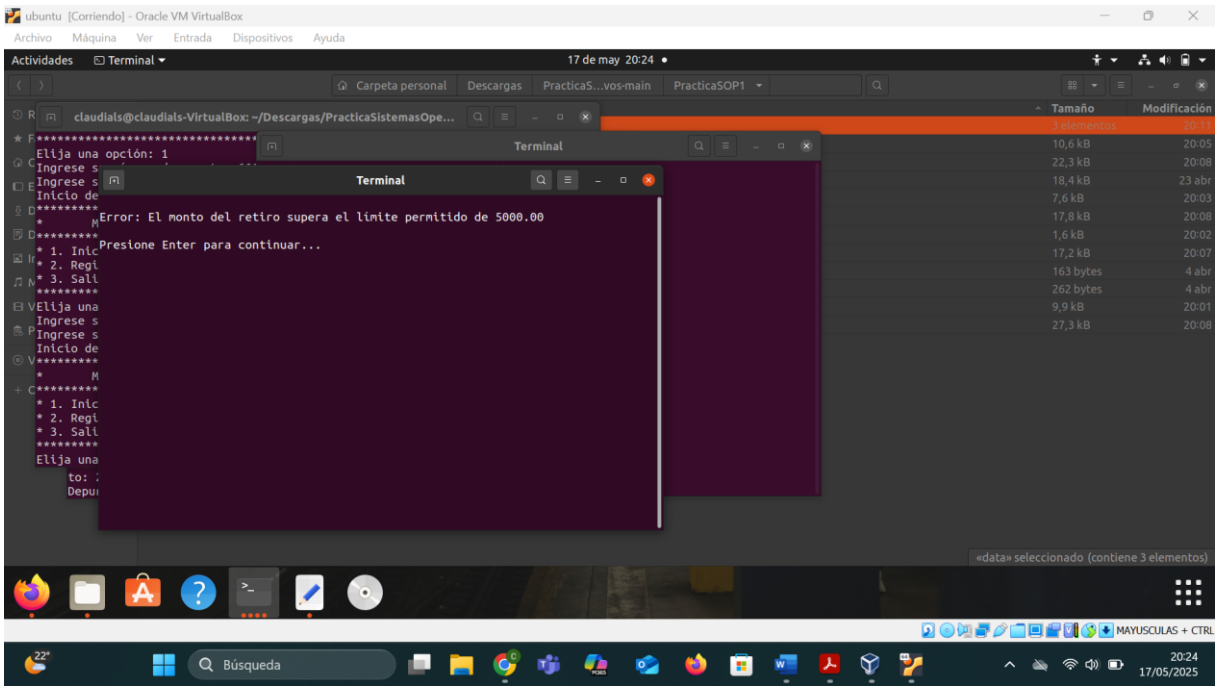

Ilustración 3: Plan de pruebas

3. Prueba de transferencias entre cuentas.

Validamos la corrección de las transferencias entre cuentas en un entorno concurrente. Para ello, dos usuarios inician transferencias simultáneamente: el usuario 1, con número de cuenta 8888, transfiere 2000 euros a la cuenta 6666, mientras que el usuario 2, con número de cuenta 6666 transfiere 152 euros a la cuenta 8888. Como resultado, el saldo de la cuenta 8888 disminuye 1848 euros y el de la cuenta 6666 aumenta en 1848 euros.

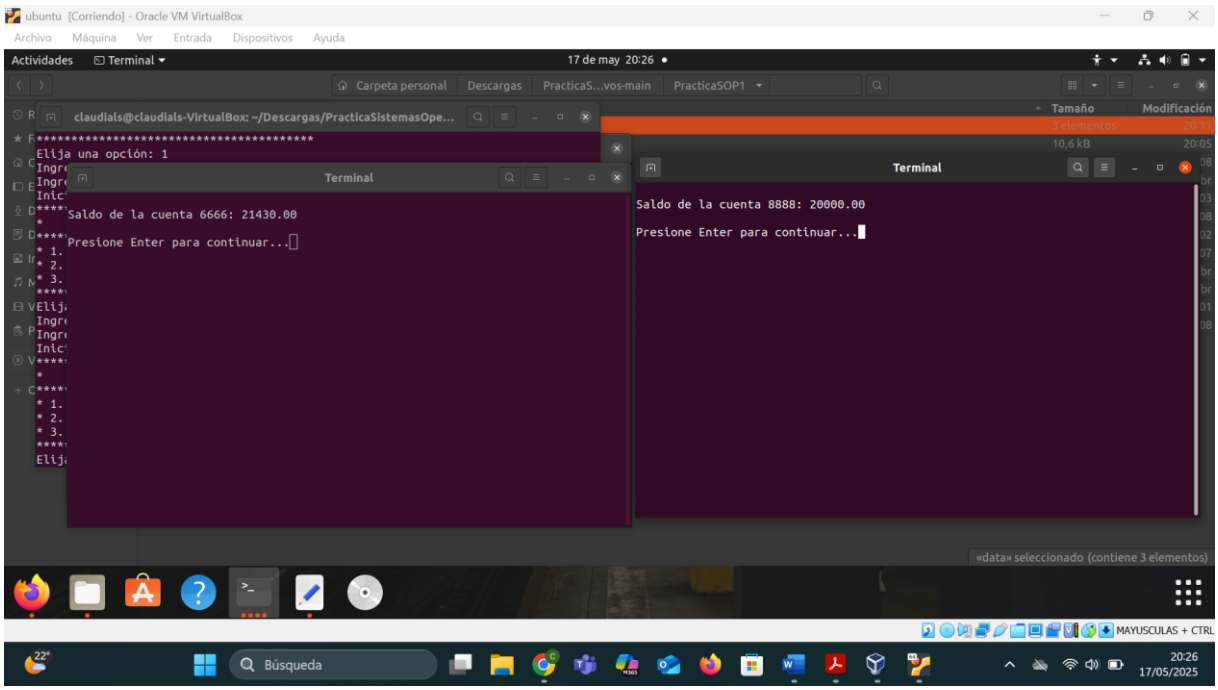

Ilustración 4: Plan de pruebas

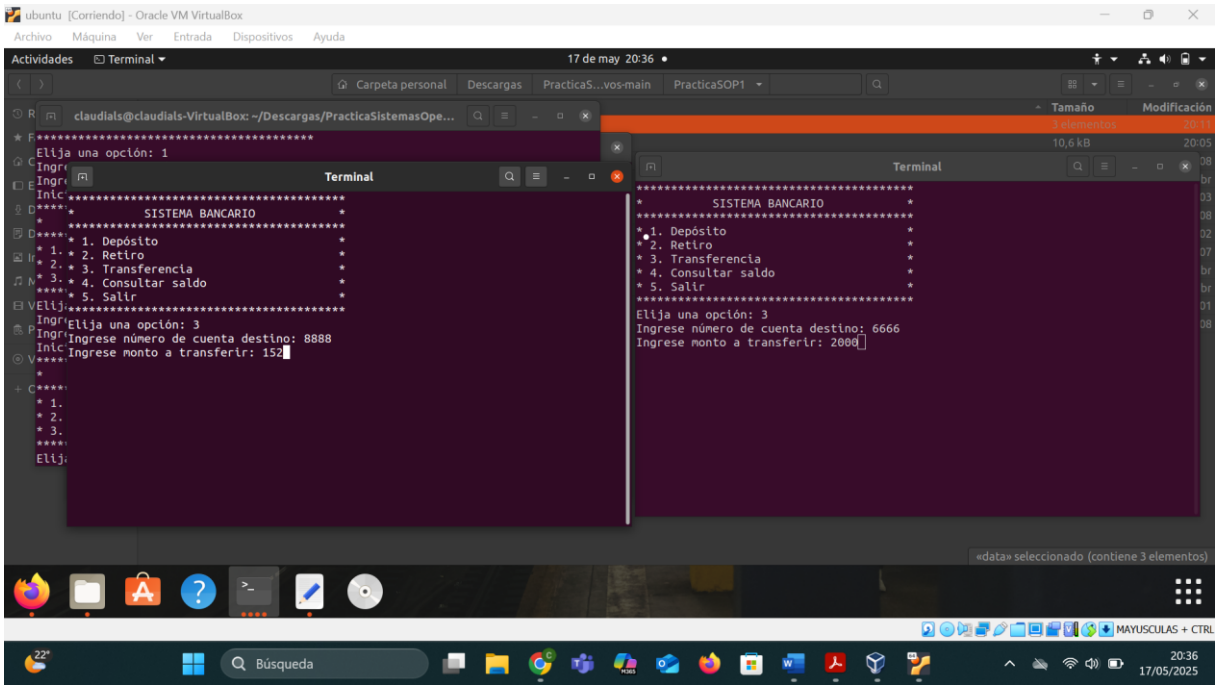

Ilustración 5: Plan de pruebas

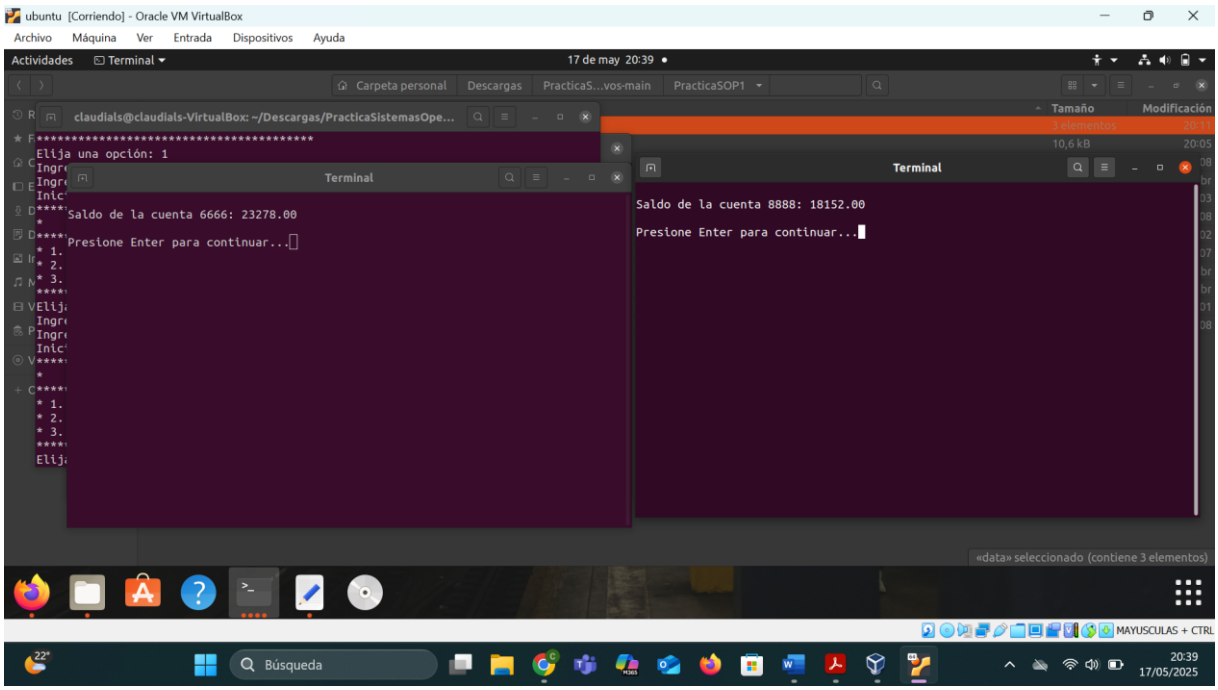

Ilustración 6: Plan de pruebas

4. Prueba de operación no válida.

Verificamos la respuesta del sistema ante entradas de usuario no válidas, intentando realizar una transferencia a una cuenta inexistente, en nuestra prueba utilizamos 4321. Como resultado, el sistema rechaza la operación con un mensaje genérico de error.

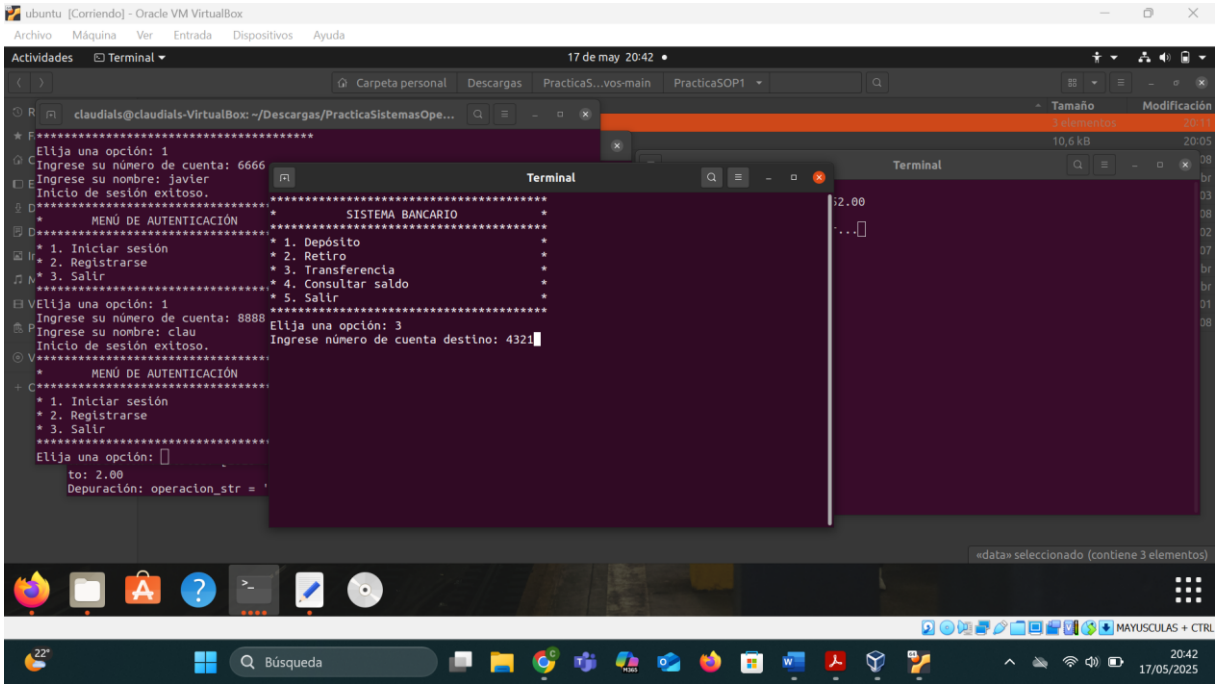

Ilustración 7: Plan de pruebas

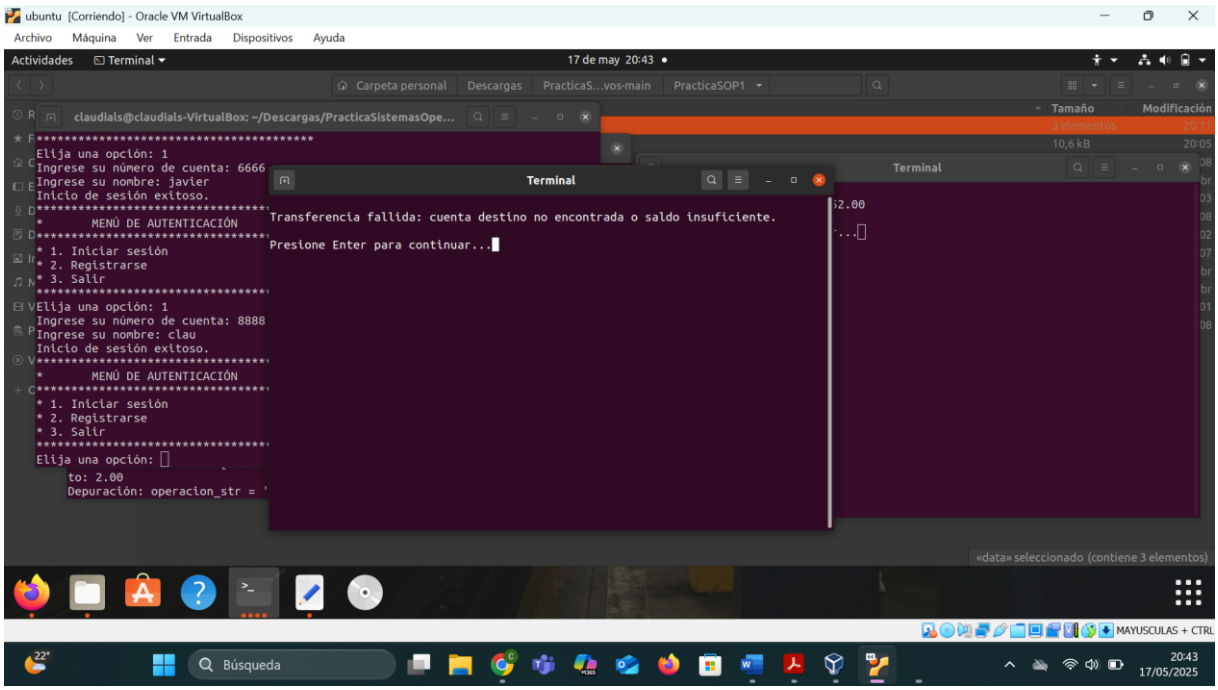

Ilustración 8: Plan de pruebas

5. Prueba de creación individual de directorios.

Comprobamos que se han creado correctamente los directorios individuales de las cuentas 8888,6666 y 1234 utilizadas anteriormente para las fases de pruebas previas.

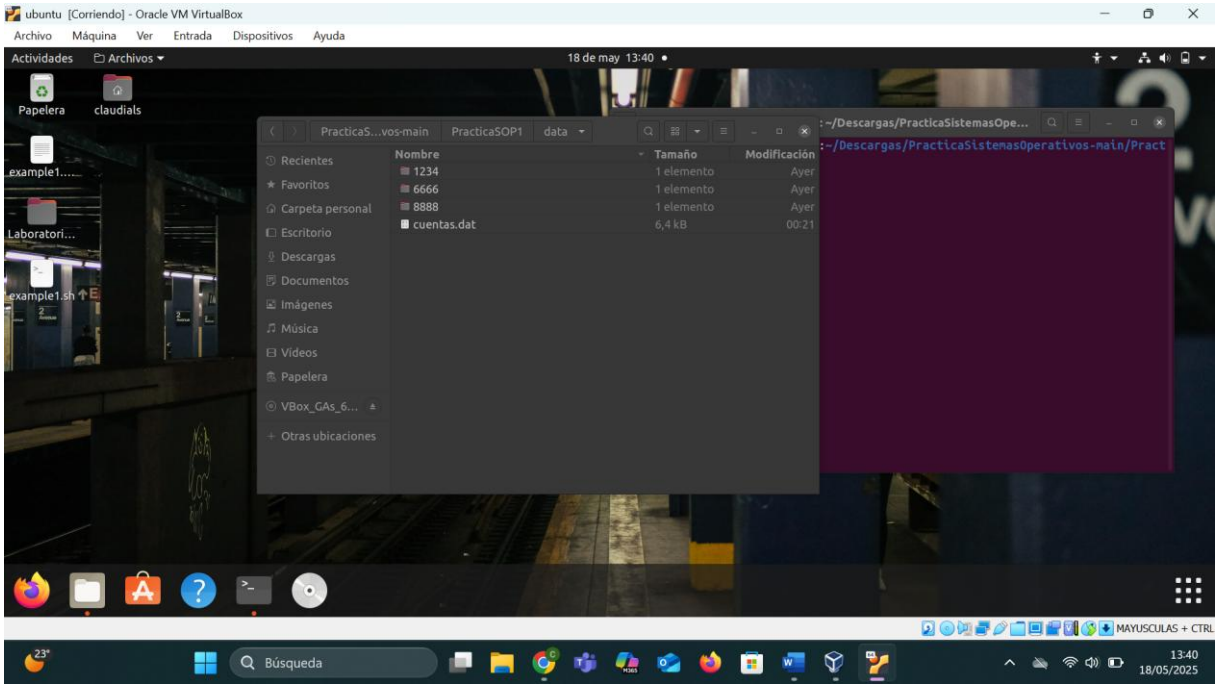

Ilustración 9: Plan de pruebas

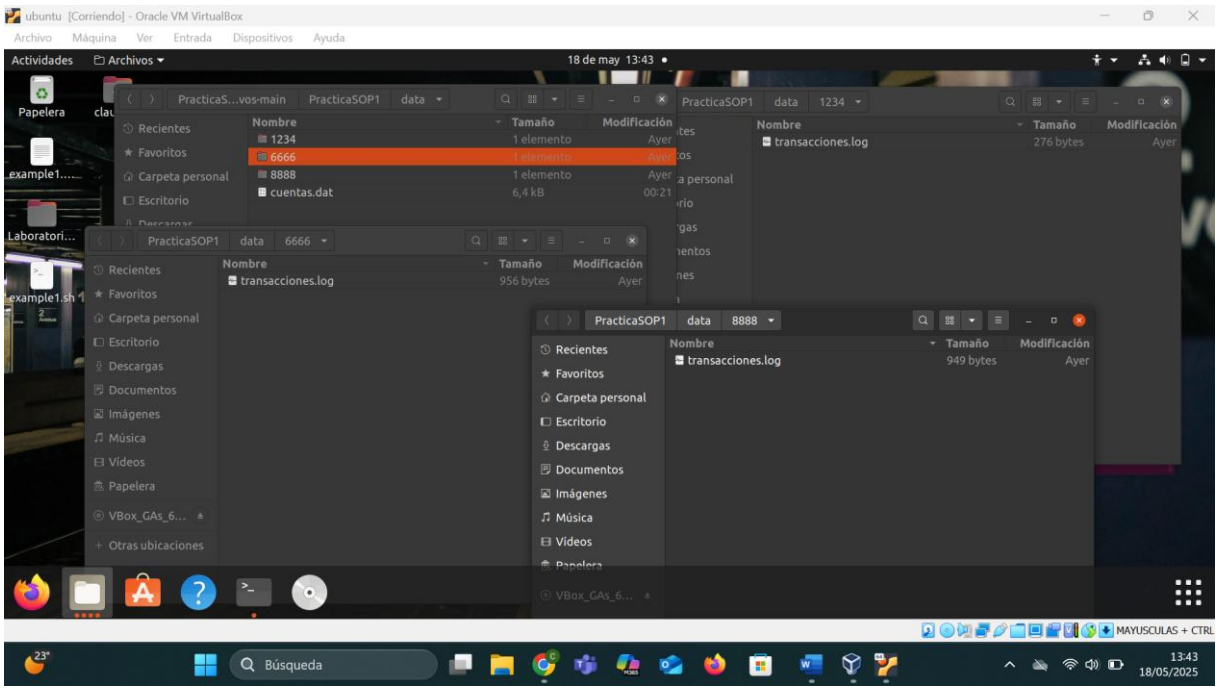

Ilustración 10: Plan de pruebas

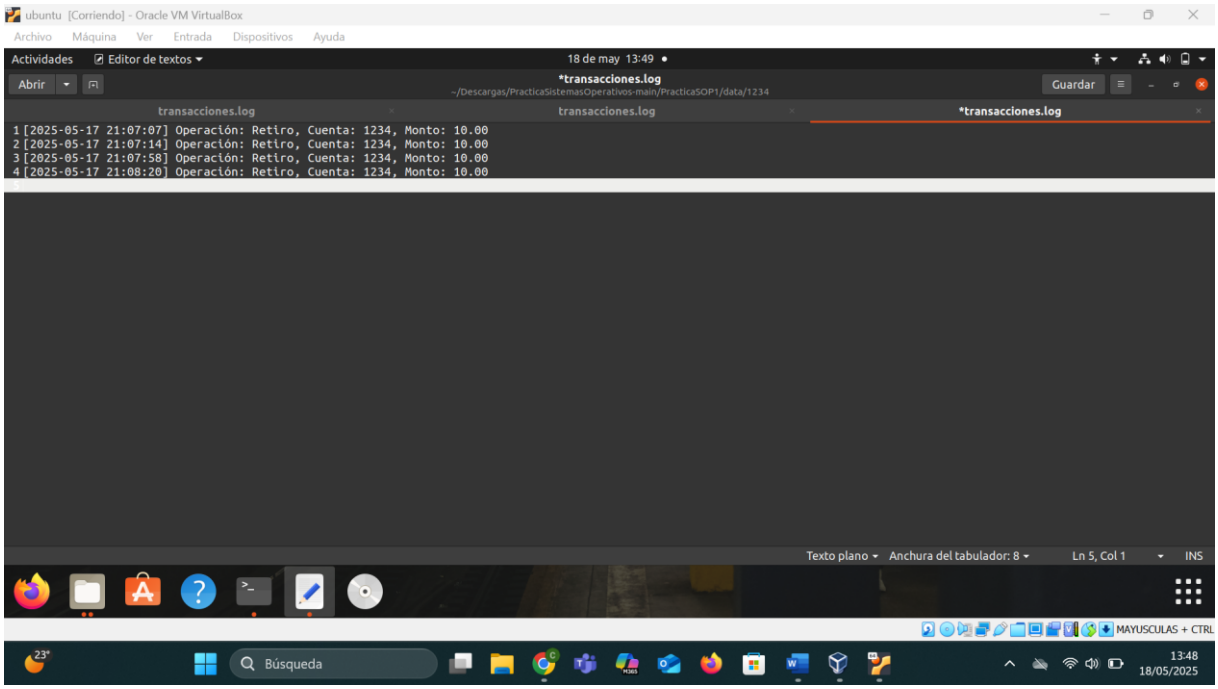

Ilustración 11: Plan de pruebas

6. Conclusiones

Con SecureBank henos obtenido una mejor comprensión de la complejidad que lleva la gestión de transacciones concurrentes en un sistema bancario, así como la importancia de implementar soluciones seguras y eficientes. Inicialmente encontramos dificultades para lograr una comunicación efectiva mediante tuberías, pero conseguimos superar estos obstáculos y desarrollar un sistema capaz de gestionar transacciones concurrentes de manera segura y eficiente, para poder implementar posteriormente la gestión de memoria, sistemas de ficheros y operaciones de entrada/salida. Establecimos la memoria compartida para una comunicación eficiente entre procesos, organizamos el sistema de ficheros para registrar las transacciones de cada usuario en directorios individuales y gestionamos las operaciones de entrada/salida utilizando buffers para minimizar el impacto de las operaciones frecuentes en disco. Con esta práctica no solo hemos aprendido a aplicar conceptos complejos, sino que también nos hemos aprendido a solucionar los desafíos encontrados y lo importante que es el trabajo en equipo para logarlo.

7. Limitaciones

El sistema implementado en esta segunda parte de la práctica presenta la limitación de la escalabilidad del sistema ante un gran número de usuarios. En un trabajo futuro se podría abordar esta limitación para mejorar la funcionalidad y la robustez del sistema.