Facultad de Ingeniería
Departamento de Ingeniería de Sistemas

Proyecto Sistemas Operativos

Noviembre 2025

Integrantes:

Giovanny Andrés Durán Rentería
Christian Becerra Enciso

Asignatura: Sistemas Operativos
Docente: John Corredor Franco

Bogotá D.C., Colombia
19 de noviembre de 2025

Índice

1. Introducción **2**

2. Descripción del proyecto **2**

3. Marco Teórico **2**
3.1. Concurrencia, procesos y hilos 2
3.2. Comunicación entre procesos (IPC) 3
3.3. Sincronización y exclusión mutua 3

4. Desarrollo del proyecto **3**
4.1. Módulo Agente 3
4.2. Módulo controlador 4

5. Plan de pruebas **5**

6. Análisis y discusión **6**

7. Conclusiones **7**

8. Referencias **7**

1. Introducción

El proyecto consta de implementar un prototipo de sistema de reservas para la simulación del aforo de un parque durante un día de operación. Dado que el parque enfrenta problemas de congestión en épocas de demanda, se busca controlar la cantidad de personas que ingresan en cada hora. Por medio de un diseño tipo cliente/servidor donde cada proceso cliente, Agente de reserva, envía una solicitud para ingresar al parque, esta solicitud es administrada por el servidor Controlador, el cual verifica la disponibilidad por hora y capacidad, autorizando, re programando o rechazando las reservas.

2. Descripción del proyecto

El proyecto para el desarrollo del sistema de simulación de reservas, implementa una arquitectura cliente/servidor. De igual manera, implementa procesos recurrentes, hilos POSIX y pipes (FIFO) para comunicación interprocesos.

El Controlador actúa como proceso principal encargado de administrar la ocupación del parque, recibir solicitudes de múltiples Agentes de Reserva, validando según la hora solicitada, la disponibilidad de horas continuas y el aforo máximo permitido, y responderlas como aceptadas, re programadas o negadas. Este emplea hilos POSIX para gestionar simultáneamente el avance del tiempo simulado y la recepción de solicitudes. En cambio, el proceso Agente leer un archivo CSV con las reservas, envía sus peticiones dependiendo del tiempo definido y procesa las respuestas recibidas por medio de un FIFO propio. Al final, el sistema genera un reporte final con estadísticas de ocupación y solicitudes procesadas.

3. Marco Teórico

El desarrollo de sistemas con implementaciones concurrentes requiere comprender conceptos fundamentales, entre ellos la concurrencia, la comunicación entre procesos, la sincronización y la gestión de recursos.

3.1. Concurrencia, procesos y hilos

La concurrencia aparece cuando varias tareas parecen ejecutarse al mismo tiempo o se interponen entre sí, ya sea en uno o varios núcleos de CPU [2], además y como se abordó en las clases, la concurrencia puede ser de dos tipos explícita, que es cuando el programador define directamente los hilos o programas o implícita, que es cuando el mismo sistema maneja la concurrencia en automático. Por su parte, un proceso es una instancia de un programa en ejecución, con su propio espacio de direcciones, archivos abiertos y contexto de ejecución, mientras que un hilo es la unidad de ejecución más ligera que comparte el espacio de memoria con otros hilos del mismo proceso [2], [1]. En este proyecto, los Agentes de Reserva representan procesos separados, mientras que el Controlador de Reserva combina procesos e hilos para manejar eventos paralelos como la recepción de solicitudes y el avance del tiempo simulado. Los procesos son ficheros en ejecución que poseen su propio espacio de memoria y contexto, mientras que los hilos POSIX (pthreads) comparten el espacio de direcciones dentro de un mismo proceso, lo

cual facilita el acceso a estructuras compartidas pero exige mecanismos de sincronización para evitar condiciones de carrera.

3.2. Comunicación entre procesos (IPC)

La comunicación entre procesos (IPC) es un componente esencial en sistemas concurrentes. En particular, el uso de pipes nominales (FIFOs) permite el intercambio de información entre procesos independientes mediante flujos unidireccionales de datos. Un FIFO conserva la estructura de una cola, garantizando el orden de llegada de los mensajes, lo cual lo convierte en un mecanismo apropiado para interacciones cliente/servidor donde múltiples emisores pueden enviar solicitudes a un receptor central. No obstante, este tipo de comunicación requiere un manejo adecuado de aperturas, bloqueos y cierres para evitar deadlocks o pérdidas de información[4].

3.3. Sincronización y exclusión mutua

Por otro lado, la sincronización es necesaria para coordinar el acceso concurrente a recursos compartidos. En este proyecto, el Controlador utiliza mutex para asegurar que el registro de por hora y la lista de reservas no sean modificadas simultáneamente por hilos que ejecutan operaciones. Un mutex es un mecanismo de exclusión mutua que garantiza que únicamente un hilo pueda acceder a una sección crítica en un momento dado, bloqueando el acceso a otros hilos hasta que el recurso sea liberado[3]. Este concepto es fundamental para prevenir condiciones de carrera, donde dos o más hilos podrían modificar datos compartidos y generar resultados incoherentes. Además, el uso de mutex permite mantener coherencia , ya que evita que un hilo lea información incompleta o que otro sobrescriba datos mientras están siendo utilizados. Por ende, se garantiza que operaciones como validar la capacidad actual, registrar nuevas reservas o actualizar la hora se realicen de forma segura, evitando inconsistencias como admitir más reservas de las permitidas, asignar horas incorrectas, etc.

En conjunto, este proyecto articula conceptos claves de los sistemas operativos: procesos, hilos, comunicación interprocesos, sincronización, exclusión mutua y administración de recursos, integrándolos en un entorno práctico que replica un escenario real de control de aforo y gestión de solicitudes concurrentes.

4. Desarrollo del proyecto

4.1. Módulo Agente

El fichero Agente se desarrolló con el propósito de representar a cada uno de los agentes que envían solicitudes al parque, Controlador. Cada agente opera como un proceso independiente cuya responsabilidad es leer un archivo CSV con solicitudes, registrarse ante el Controlador y conectarse a este mediante pipes FIFO. Por otro lado, tiene dos funciones principales: iniciarAgente() y enviarSolicitudes(), se dividió mediante programación modular, agente.c que tiene la implementación de las funciones, agente.h donde se declara sus funciones y por ultimo, mainAgente.c donde se maneja la lógica de los agentes para el envió de las solicitudes y la conexión con el controlador.

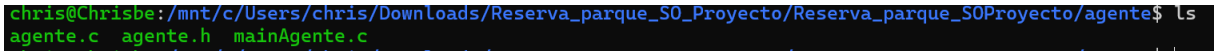

Figura 1: Ficheros Agente

Con la función iniciarAgente(), se implementó la creación del pipe FIFO del agente, definido dinámica mente como /tmp/pipe-<nombre>. Este pipe es el canal exclusivo donde el Controlador enviará las respuestas dirigidas a dicho agente. Posteriormente, la función genera un mensaje de tipo REGISTER, que incluye el nombre del agente y el nombre de su pipe personal, y lo envía al pipe central del Controlador. La función espera luego una respuesta de tipo TIME, la cual contiene la hora simulada inicial. Esta hora es necesaria para filtrar solicitudes desde el CSV y garantiza que cada agente esté sincronizado con el reloj del sistema.

La función enviarSolicitudes(), se diseñó para cargar y procesar todas las líneas del archivo CSV. Cada línea contiene el nombre de la familia, la hora solicitada y la cantidad de personas. Cada solicitud incluye el nombre de la familia, la hora de reserva, la cantidad de personas y el nombre del pipe de respuesta del agente. Si la hora solicitada es anterior a la hora recibida del Controlador, la solicitud se descarta.

Cada mensaje es enviado al pipe principal del Controlador y después el agente realiza una lectura en su pipe personal esperando respuesta. Las respuestas pueden ser de tipo OK, REPROGRAMADO, TARDE o RECHAZADO, y la función imprime el resultado correspondiente. Además, se implementó un retardo de 2 segundos entre cada envío.Finalmente, una vez procesadas todas las entradas, el módulo libera la memoria asignada dinámicamente y cierra la ejecución.

El archivo mainAgente.c complementa el desarrollo permitiendo ejecutar el agente mediante parámetros (-s, -a, -p), vandalizándolos y coordinando las llamadas a las funciones principales.

4.2. Módulo controlador

El fichero controlador encargado de administrar la ocupación del parque, coordinar la comunicación con los agentes y supervisar el avance del tiempo. Se implemento hilos POSIX (pthread), mutex para las condiciones de carrear y una comunicación pipe FIFO con los agentes.

Figura 2: Ficheros Controlador

En primer lugar, la función iniControlador() se encarga de definir la hora de inicio y fin, capacidad máxima del parque, duración de cada hora y el nombre del pipe FIFO por el cual se reciben todas las peticiones. También se inicializa el arreglo de ocupación del parque, que registra la cantidad de personas presentes en cada hora. Una vez configurado el entorno, se crea el FIFO general mediante mkfifo() y se lanzan los hilos hiloDeeReloj() y hiloDeSolicitudes(), que constituyen los dos pilares del sistema concurrente.

La función hiloDeSolicitudes() se encarga de leer de forma continua el FIFO, procesando mensajes enviados por los agentes.

La función gestioDeSolicitud() evalua todas las reglas necesarias para determinar si una reserva puede ser aceptada, reprogramada o negada. Para evitar condiciones de carrera,

esta función utiliza un mutex (lockParque) que protege el acceso a los datos compartidos, principalmente el arreglo ocupación y la hora del parque. Dependiendo del caso, el sistema responde con mensajes de tipo OK, REPROGRAMADO, TARDE o RECHAZADO, los cuales se envían directamente al pipe personal del agente.

Por su parte, el hilo hiloDeeReloj() simula el tiempo del sistema. Cada cierto número de segundos (configurado mediante segundosHoras) incrementa la hora actual del parque y muestra el estado de ocupación de las horas activas. Cuando se alcanza la hora final, el hilo detiene la simulación y genera el reporte final del día, incluyendo horas pico, horas valle y estadísticas completas de reservas: aceptadas, re programadas, tardías y rechazadas. Tras finalizar, se cierra la ejecución del hilo de solicitudes para garantizar un cierre ordenado del sistema.

5. Plan de pruebas

El plan de pruebas se diseñó con el propósito de verificar el correcto funcionamiento del sistema de reservas bajo distintos escenarios, evaluando el Controlador como el comportamiento y los Agentes. Para ello, se definieron casos de prueba orientados a analizar, el registro de agentes, la comunicación mediante pipes FIFO, la correcta aplicación de las reglas de validación de reservas, la sincronización entre hilos y la actualización coherente de la ocupación del parque a lo largo del tiempo simulado.

Para el plan de pruebas en primera parte vamos a probar con un agente mandando solicitudes al controlador, y en el segundo caso, en vez de un solo agente serán dos.

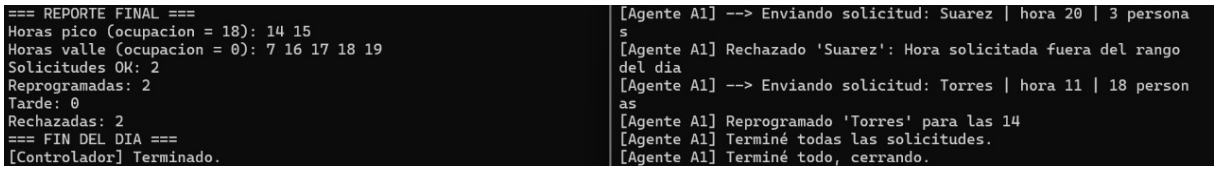

Figura 3: Un Agente

En el escenario con un solo agente, el programa permite evaluar el flujo del fichero sin la necesidad de condiciones de carrear al haber otro agente. El agente se registra ante el Controlador, envía sus solicitudes de forma secuencial y recibe las respuestas correspondientes, lo que facilita comprobar que la comunicación mediante pipes funciona correctamente y que el Controlador procesa adecuadamente las reservas y actualiza la ocupación .

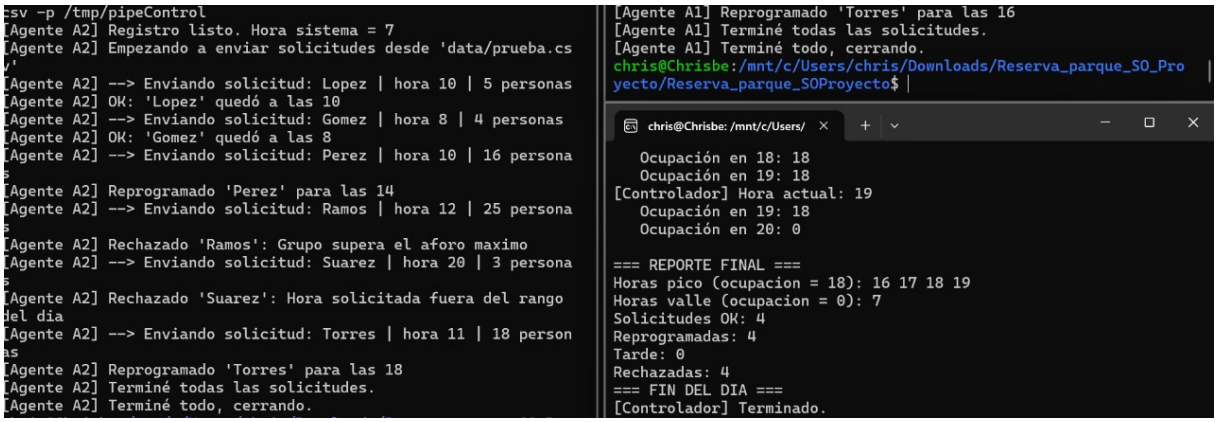

Figura 4: Dos agentes

En el escenario con dos agentes, el programa pone a prueba su capacidad de manejar concurrencia, ya que ambos procesos envían solicitudes al Controlador de manera independiente y casi simultánea. Esto permite verificar que el pipe principal recibe mensajes de múltiples emisores sin pérdida, que el Controlador procesa cada solicitud de forma ordenada y que los mecanismos de sincronización, como los mutex, garantizan la coherencia en la actualización del numero de personas actual. Es por eso que las estadísticas cambian al cambiar de horario elegido algunas peticiones dado que supera la capacidad.

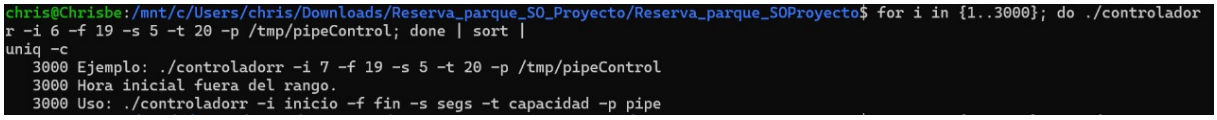

Figura 5: Hora fuera de lo permitido, caso Controlador

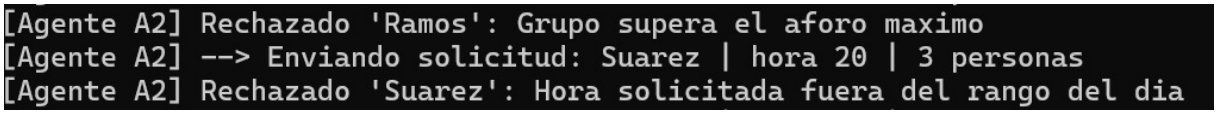

Figura 6: Hora fuera de lo permitido, caso respuesta al Agente

En el escenario donde una solicitud supera la hora del rango del parque, el objetivo es verificar que el Controlador identifique correctamente las peticiones fuera del horario definido. Cuando un agente o el mismo controlador da como parámetro una reserva para una hora mayor a la hora final configurada, el Controlador clasifica la solicitud como inválida y responde con un mensaje de rechazo, evitando que esta afecte la ocupación o el flujo de la ejecución. Este caso permite confirmar que el sistema respeta estrictamente los límites del horario de funcionamiento y que maneja adecuadamente las solicitudes que no cumplen con las restricciones.

6. Análisis y discusión

El sistema integro los conceptos de concurrencia, comunicación entre procesos y sincronización, logrando el objetivo de controlar y administrar las reservas de un parque.

En primer lugar, el uso de pipes FIFO como forma de comunicación entre procesos demostró ser efectivo al momento en el que el fichero agente mandaba las solicitudes y el controlador las recibía. Los agentes pueden enviar solicitudes de manera simultánea sin que el Controlador pierda mensajes, gracias a la naturaleza del FIFO y al hilo implementado. Sin embargo, se evidenció que los FIFO presentan ciertas limitaciones en cuanto a multiplexación y capacidad de identificación de emisores, lo cual fue necesario que cada agente contara con su propio pipe privado para recibir respuestas.

Por otro lado en la concurrencia, al declarar en dos hilos uno para el reloj y otro para la recepción de solicitudes , demostró ser una solución efectiva. El hilo del reloj pudo avanzar de manera independiente, evitando bloqueo. Gracias al uso de mutex, se pudo actualizar la ocupación, la lectura de la hora y el registro de reservas. Se comprobó que sin dichos mecanismos podrían haberse producido condiciones de carrera, especialmente cuando varios agentes enviaban solicitudes simultáneas.

Durante la ejecución Controlador logro manejar las excepciones como la saturación de aforo o necesidad de re programación. Puede que en contextos de alta cantidad de solicitudes sea menos eficiente dado que se debe recorrer múltiples horas consecutivas.

El proceso Agente funciono adecuadamente al validar solicitudes y esperar respuestas del Controlador.

Finalmente, al terminar la ejecución se genero un reporte con estadísticas las estadísticas del parque. La información como horas pico, horas valle y número de solicitudes aceptadas, negadas o re programadas demostró que el sistema administra correctamente las reservas.

7. Conclusiones

El desarrollo del proyecto permitió integrar de forma los conceptos de concurrencia, sincronización y comunicación entre procesos. A través de la arquitectura cliente/servidor implementada mediante procesos Agente y un Controlador, de manera efectiva se dio la gestión de reservas.

El proyecto cumplió con los objetivos planteados gracias al manejo de hilos, mutex y pipes.

En conclusión, el proyecto logró integrar de manera efectiva los conceptos abordados, mostrando un resultados coherentes dentro de los parámetros planteados. Aunque es posible que existen algoritmos mas eficientes, el programa cumple con los requerimientos y demuestra un correcto entendimiento de la programación concurrente.

8. Referencias

[1] Bradford Nichols, Dick Buttlar, and Jacqueline Farrell. *Pthreads Programming.* O'Reilly Media, 1996.

[2] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts.* Wiley, 9 edition, 2018.

[3] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts.* Wiley, 9 edition, 2018.

[4] W. Richard Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley, 1 edition, 1992.