Pontificia Universidad Javeriana

Facultad de Ingeniería de sistemas

Proyecto Sistemas Operativos

Integrantes:

María Fernanda Velandia Gracia
Danna Gabriela Rojas Bernal

Materia:
Sistemas Operativos

Profesor:
John Corredor Franco

18 de noviembre 2025

Bogotá D.C, Colombia

Índice

1. Introducción

En el presente documento se desarrolló una simulación de reservas por horas en un parque privado, el Parque Berlín, con el fin de diseñar e implementar un sistema concurrente de cliente/servidor que represente la reserva por horas, controlando el aforo y gestionando solicitudes de múltiples agentes mediante procesos, hilos y mecanismos de sincronización.

2. Objetivos

2.1 Objetivo General

Implementar un simulador concurrente que permita gestionar reservas por horas en un parque, utilizando proceso e hilos POSIX, mecanismo de sincronización y comunicación entre procesos

2.2 Objetivo Específico

- Diseñar un programa cliente/servidor donde un controlador de reserva actúe como servidor y varios agentes de reserva como cliente.
- Implementar comunicación entre procesos mediante FIFOs y validar llamadas al sistema.
- Gestionar concurrencia usando hilos y mecanismo de sincronización como mutex o semáforos.
- Simular el avance del tiempo y actualizar el estado del parque por hora para luego generar un reporte final.

3. Descripción general del proyecto

El proyecto consiste en el desarrollo de un Prototipo de Sistema de Reservas que simula el uso del Parque Berlín durante un día específico, para controlar el aforo máximo permitido y gestionar de manera eficiente la alta cantidad de familias en temporada vacacional. Con el propósito de evitar la saturación de los servicios dentro del parque mediante un sistema que procese múltiples solicitudes de reserva de forma concurrente, garantizando orden y equilibrio en la asignación de cupos.

Para esto, se implementa una arquitectura cliente/servidor, donde el Controlador de Reserva funciona como proceso servidor y los Agentes de Reserva operan como procesos cliente. La comunicación entre ambos se establece mediante pipes (FIFOs), permitiendo que varios agentes se conecten simultáneamente al controlador.

El Controlador de Reserva es responsable de recibir solicitudes de ingreso, validarlas y decidir si son aceptadas, negadas o reprogramadas, basándose en la capacidad

máxima del parque y en la disponibilidad de dos horas. Esto simula el paso del tiempo dentro de un rango entre las 7 y las 19 horas, avanzando el reloj cada vez que se cumpla una hora de simulación. En ese avance, el controlador debe retirar del parque a las familias cuya reserva haya terminado, admitir las nuevas reservas autorizadas y mantener un registro completo del estado del aforo a lo largo del día. Según el resultado de cada solicitud, el controlador responde al agente con mensajes como "Reserva ok", "Reserva garantizada para otras horas" o "Reserva negada" (por extemporaneidad o falta de cupo en todas las franjas posibles). Al finalizar el día, genera un reporte final con información clave como lo son las horas pico y de menor ocupación, total de personas admitidas, y número de solicitudes negadas o reprogramadas.

Por su parte, los Agentes de Reserva pueden iniciarse en cualquier momento después del controlador. Cada agente se registra enviando su nombre y luego procede a leer un archivo de entrada (fileSolicitud) que contiene las reservas solicitadas en formato (Familia, Hora, Número de personas). Antes de enviar una solicitud, el agente valida que la hora no sea inferior a la hora actual de simulación. Tras enviar la petición al controlador, espera la respuesta, la imprime en pantalla y luego envía la siguiente solicitud. El agente finaliza cuando agota todas las solicitudes de su archivo.

4. Marco teórico

4.1. Proceso

Un proceso es una instancia de un programa en ejecución que contiene instrucciones, el estado de ejecución, la memoria reservada y otros recursos asignados por el sistema operativo como la CPU, memoria y dispositivos de entrada/salida [1].

4.2. Hilos

Los hilos se utilizan para manejar la concurrencia dentro de un proceso. Estos permiten que un programa ejecute múltiples tareas de manera simultánea, y el desafío principal al utilizarlos es implementar restricciones para coordinar los hilos de diferentes maneras. Los dos principales desafíos que deben enfrentarse son:

- Condiciones de carrera(Race Condition): Esto ocurre cuando la salida de un programa depende de la secuencia o el momento en que se ejecutan los hilos.
- Interbloqueo (Deadlock): Esto puede causar que un conjunto de hilos quede en "starved" o sea eternamente inactivo.

Para gestionar lo anterior se emplean mecanismos de sincronización como:

- Muxetes: Estos se utilizan para marcar los límites de las regiones críticas y garantizar que haya a lo sumo un hilo presente dentro de ellas

- Semáforos: El interbloqueo se puede prevenir programáticamente implantando directivas para limitar el número de hilos que compiten por un recurso compartido. Un semáforo es un tipo de variable que representa un recuento de recursos finitos o un recuento de "permisos" o "tickets" disponibles. Los semáforos permiten la coordinación bidireccional de hilos. [2]

4.3. Concurrencia

La concurrencia se refiere a la capacidad de un sistema para manejar múltiples tareas o procesos que se ejecutan aparentemente al mismo tiempo. Es fundamental para mejorar el rendimiento, la capacidad de respuesta y la eficiencia de los sistemas informáticos.[3]

4.4. Pipes

Los pipes son un mecanismo de comunicación entre procesos (IPC) que permite el envío de datos entre un proceso emisor y un proceso receptor utilizando un canal unidireccional [4].

4.5. Cliente/Servidor

El modelo cliente/servidor es una arquitectura donde un proceso servidor suministra servicios o recursos a uno o varios procesos cliente que los solicitan. El servidor controla el acceso al recurso, mantiene su estado y gestiona múltiples solicitudes, mientras los clientes únicamente piden operaciones específicas[5].

5. Desarrollo de proyecto

5.1. Explicacion de codigo

Lo primero que se encuentra en nuestro archivo de proyecto:

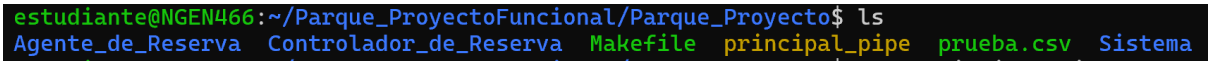

Imagen 1. Archivos de proyecto

donde:

- Agente_de_Reserva: Es el archivo donde se encuentran los programas para el proceso cliente que se comunica con el “Controlador_de_Reserva”
- Controlador_de_Reserva: Es el archivo donde se encuentran los programas para el proceso servidor, respodnable de administrar las reservas del parque
- Sistema:Es el módulo base del proyecto, en el que se alberga las definiciones comunes
- Makefile:s el archivo que automatiza la compilación del proyecto, es decir, permite compilar los ejecutables controlador y agente con un solo comando
- principal_pipe: Comprueba la creación de los pipes, es decir, la comunicación entre procesos.

- prueba.csv: Este archivo CSV contiene las solicitudes de reserva que los agentes deben procesar. El formato es el siguiente: nombreFamilia,Hora,Personas

5.1.1 Archivo Agente_de_Reserva

Para empezar tenemos dentro de nuestro proyecto una carpeta llamada “Agente_de reserva” el cual contiene los siguientes programas:

```
estudiante@NGEN466:~/Parque_ProyectoFuncional/Parque_Proyecto/Agente_de_Reserva$ ls
Agente.c  Agente.h  mainAgente.c
```

Imagen 2. Archivos de “Agente_de_Reserva”

La funcionalidad general del agente es leer los parámetros desde la línea de comandos (nombre del agente, archivo CSV con solicitudes, y pipe del controlador) este los guarda y crea su propio pipe para que el controlador responda. Luego, envía un mensaje de tipo registro mediante el pipe del controlador, le dice al controlador cuál es su pipe de respuesta, espera a que el controlador le confirme la conexión y entregue la hora actual de la simulación. Al momento de proceso las solicitudes del archivo CSV, lee linea por linea los datos; valida si la hora es válida, envía la solicitud al controlador, espera su respuesta e imprime lo que diga el controlador, espera 2 segundos; finaliza y borrar el pipe que creó anteriormente.

5.1.1.1.Agente.c

Este programa implementa el proceso cliente del sistema en el que se lee solicitudes desde el archivo CSV; envía estas solicitudes al controlador mediante pipe FIFO; espera la respuesta del controlador; muestra su la reserva fue aceptada, reprograma o negada; cierra y elimina el pipe de respuesta cuando termina. Tiene funciones como:

- inicializarAgente(): El cual configura el agentes antes de iniciar; este copia los parametros recibidos (nombre, CSV, pipe del controlador; genera un pipe de recepcion unico usando PID, crea un FIFO y muestra mensaje de inicio.
- ejcutarAgente(): Primero se registra ante el controlador y luego envía todas las solicitudes del archivo CSV.
- finalizarAgente():Elimina su pipe persona y muestra mensaje de cierre.
- regsitrarConControlador():Intenta abrir el pipe del controlador en modo escritura; envía estructura Solicitud de tipo MSG_REGISTRO; pone nombre del agente y el pipe donde quiere recibir respuestas; espera la respuesta del controlador y guarda la hora actual de simulación enviada por el servidor.
- procesarSolicitudes():Abre el archivo CSV; lee cada línea;obtiene familia, hora solicitada y número de personas ; llama a enviarSolicitud(); espera 2 segundos.
- enviarSolicitud():Abre el pipe del controlador en modo escritura; envía la estructura Solicitud por el pipe; abre su pipe de recepción; espera la respuesta del controlador; la recibe y llama a imprimirRespuesta().

- leerLineaCSV(): Lee una línea del archivo; evita líneas vacías o comentadas; usa sscanf() para separar familia, hora, personas.
- imprimirRespuesta():Muestra el mensaje enviado por el controlador:
    - RESP_OK: reserva aprobada.
    - RESP_REPROGRAMADA: se movió a otra hora
    - RESP_NEGADA_EXTEMPORANEA: hora ya pasó, pero se reprogramó
    - RESP_NEGADA_VOLVER: reserva rechazada sin alternativa

5.1.1.2. Agente.h

Es el archivo de cabecera donde se declaran la estructura ParametrosAgente que contiene: nombreAgente, archivoSolicitudes, pipeEnvio. Ademas de los prototipos de todas las funciones para que mainAgente.c y otros archivos puedan usar las funciones del agente.

5.1.1.3. mainAgente.c

El archivo lee parámetros -s -a -p. Luego crea un ParametrosAgente params y llama a inicializarAgente(&params), ejecutarAgente() y finalizarAgente().

5.1.2 Archivo Controlador_de_Reserva

En este archivo Controlador_de_Reserva es el proceso en el que se administra el aforo del parque, recibe solicitudes enviadas por los agentes mediante pipes FIFO, gestiona reservas, reprograma, rechaza o acepta según la disponibilidad, mantiene un reloj interno que simula el paso de las horas, genera estadísticas y un reporte final del dia.

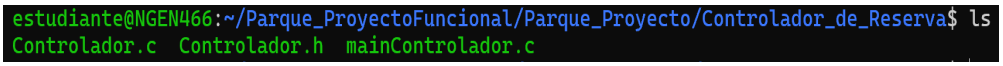

Imagen 3. Archivo “Controlador_de_Reserva”

5.1.2.1. Controlador.c

En este archivo se implementa el proceso principal el cual actúa como el servidor central encargado de recibir las solicitudes enviadas por los agentes a través de pipes FIFO; además de validar la hora, foro máximo y disponibilidad; en este se crea, reprograma o niega reservas según las reglas del sistema; lleva el control del tiempo simulado mediante thread; procesa las solicitudes mediante otro hilo; registra estadísticas; envía una respuesta al agente mediante un segundo pipe.

Se tiene funciones como:

- inicializarControlador(): Válida las horas iniciales y finales del controlador,carga los parámetros (horas, aforo, segundos por hora), crea el pipe FIFO donde los agentes enviarán sus solicitudes.
- finalizarControlador(): Cuando termina la jornada genera el reporte final, libera mutex, borra el pipe FIFO creado e imprime el mensaje final

- threadReloj(): Este cada cierto tiempo avanza la hora simulada, y cuando llega la hora final este termina.
- threadPeticiones(): este espera mensajes de agentes, lee estructuras solicitud, segun el tipo, si es MSG_REGISTRO llama procesarRegistro() y si es MSG_SOLICITUD llama a procesarSolicitud(); y escribe la respuesta en el pipe del agente.
- caculatPersonasEnHora(): Recorre el arreglo de reserva y calcula cuántas personas hay en una hora determinada
- buscarBloqueDisponible(): Busca el bloque disponible de dos horas seguridad con aforo disponible para programar las reservas.
- avanzarHora(): Cada vez que pasa una hora imprime quien sale, quien entra, ocupación actual y actualiza estadísticas por hora.
- manejarSenalFin(): si el usuario presiona CTRL+C: activa debeTerminar y el controlador realiza el cierre.
- procesarSolicitud():este contiene la logica principal del sistema en el que se valida:
    - Si el aforo es excedido, este se niega
    - Si la hora ya paso, reprograma si hay cupo
    - Si la hora está fuera del horario, se niega
    - Si hay disponibilidad, reserva aprobada
    - Si no hay cupo pero existe otra hora disponible, se reprograma
    - Si no hay ninguna disponibilidad, se niega

- ejecutarControlador(): Crea los dos hilos principales
    - Hilo 1 Reloj: maneja la funcion threadReloj(), este controla el paso del tiempo, cada segHoras avanza la simulación una hora y llama avanzarHora() para actualizar el estado del parque
    - Hilo 2 Peticiones: maneja la funcion threadPeticiones(), este abre el pipe de solicitudes; lee solicitudes de agentes,procesa los registros de agentes y las solicitudes de reserva; y envía la respuesta correspondiente por el pipe del agente.

5.1.2.2. Controlador.h

En este se define la estructura ParametrosControlador; expone las funciones públicas del controlador; declara hilos, funciones auxiliares y manejadores de señales
Este archivo le permite a otros módulos (como mainControlador.c) usar el controlador sin conocer sus detalles internos.

5.1.2.3. mainControlador.c

Este se encarga de leer parametros desde la linea de comandos; válida y copia dichos parámetros que coloca dentro de ParámetrosControlador; llama funciones del controlador como:

- inicializarControlador()
- ejecutarControlador()
- finalizarControlador()

5.1.3 Archivo Sistema

Este archivo de sistema funciona como la biblioteca central del proyecto ya que define todas las constantes globales del sistemas, por ejemplo, los horarios permitidos, duración de las reservas, tamaño maximo de nombre, tamaño máximo de buffer y número máximo de reservas; por otro lado, define las estructuras que usan el controlador y los agentes como:

- Solicitud que envía un agente
- Respuesta que envía el controlador
- Reserva que almacena el sistema
- Estadísticas que guarda los datos al final del dia

Y define funciones auxiliares como validar horas, validar rangos e imprimir mensajes de error

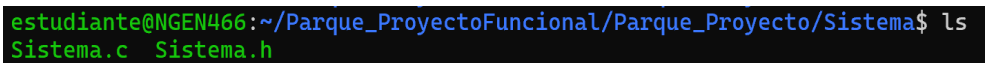

Imagen 4. Archivo “Sistema”

5.1.3.1. Sistema.c

Este program implementa funciones generales que son utilizadas por el controlador y los agentes; presenta funciones como:

- imprimirMensajeError():Imprime un mensaje por stderr (la salida estándar de errores), el cual se usa para mensajes críticos o avisos importantes.
- imprimirMensajeInfo(): Imprime un mensaje informativo, y este se usa para depuracion o mensajes de estado.
- validarHora(): Revisa si la hora está dentro del rango permitido (7 a 19 horas) y retorna 1 si es valido o 0 si no lo es. Esta función la utiliza el controlador antes de aceptar reservas.
- validarRangoHoras(): verifica que ambas horas esten en el rango y que horaInicio sea menor o igual a horaFin

5.1.3.2. Sistema.h

Este programa define estructuras; constantes como MAX_NOMBRE(Tamaño máximo de cadenas como nombres de familia o agentes),MAX_BUFFER(Tamaño de mensajes largos),HORAS_DIA(Número de horas simuladas (19 - 7 = 12 + 1)), DURACION_RESERVA(Las reservas duran 2 horas), MAX_RESERVAS(Máximo total de reservas que el sistema puede almacenar); Tipos de mensaje (enums) para permitir que ue agentes y controlador identifiquen qué tipo de mensaje están enviando y prototipos para que todo el proyecto comparta el mismo lenguaje

5.1.4 Makefile

El Makefile automatiza la compilacion del proyecto en el que se tienen tres modulos, Sistema (biblioteca Comun); Controlador_de_Reserva (proceso servidor); y Agente_de_Reserva (procesos cliente). En este se compila dos ejcutables, el controlador y el agente. Y tiene varias secciones:

- variable globales: en el que se define el compilador, los flags y el nombre de los programas finales
- Directorios
- includes: Que permite que cada .c pueda include headers de las otras carpetas sin errores
- El all: permite compilar ambos programas
- Clean: Elimina los ejecutables y los .o para limpiar el proyecto.
- Reglas para generar ejecutables: Cada ejecutable se arma combinando archivos objeto como controlador que usa Sistema.o, Controlador.o y mainControlador.o; o agente que usa Sistema.o, Agente.o y mainAgente.o**.**

5.2. Plan de Pruebas

5.2.1. Introducción

El propósito de este plan de pruebas es el correcto funcionamiento del Sistema de Reservas para el parque Berlín. Nuestros casos están enfocados en los tipos de reserva que se pueden y se deben realizar. Nuestro plan se estructuró con objetivo, parámetro de entrada en el controlador, parámetro de entrada en el agente, datos del archivo .csv y lo que se muestra por consola en ambas terminales(controlador y agente).

5.2.1.1. Casos de Prueba

5.2.1.1.1 Prueba #1

Objetivo: Verificar que reprograma bien en caso de tener una reserva antes de la hora de inicio del parque.

- Parámetro controlador para ejecutarlo: ./controlador -i 8 -f 14 -s 3 -t 30 -p principal_pipe

Donde: -i (hora inicio), -f(hora fin), -s(segundos por hora simulada), -t(aforo máximo), -p (pipe).

- Parámetro agente: ./agente -s A -a prueba1.csv -p principal_pipe
  Donde: -s(nombre agente) -a (archivo CSV), -p(pipe).
- Archivo.csv contiene:
  alba,7,4
  sanchez,9,8
  rodriguez,11,9

5.2.1.1.2 Resultados de Prueba #1

Controlador:

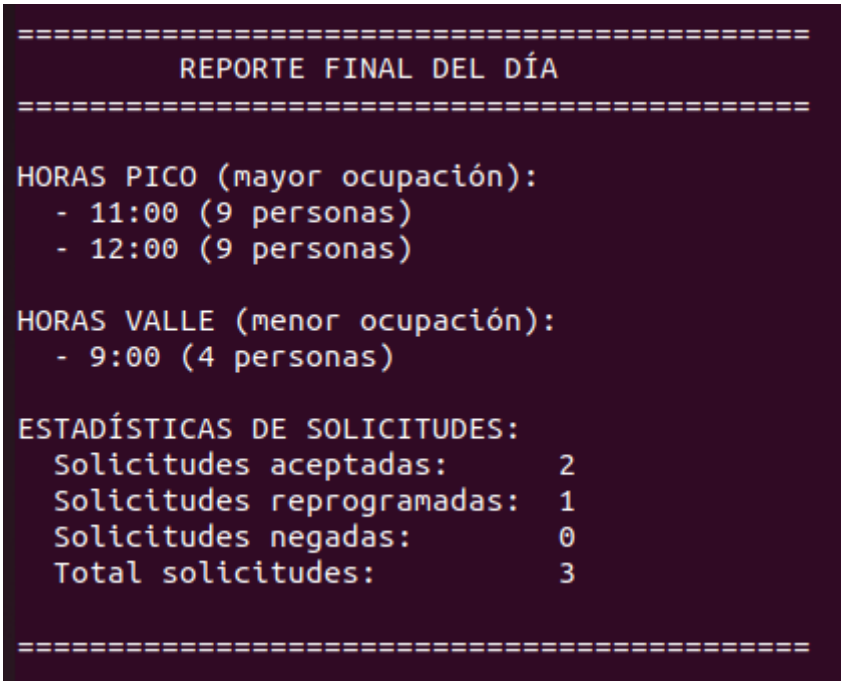

Imagen 5. Resultado de ejecución del Controlador para el primer caso

Agente:

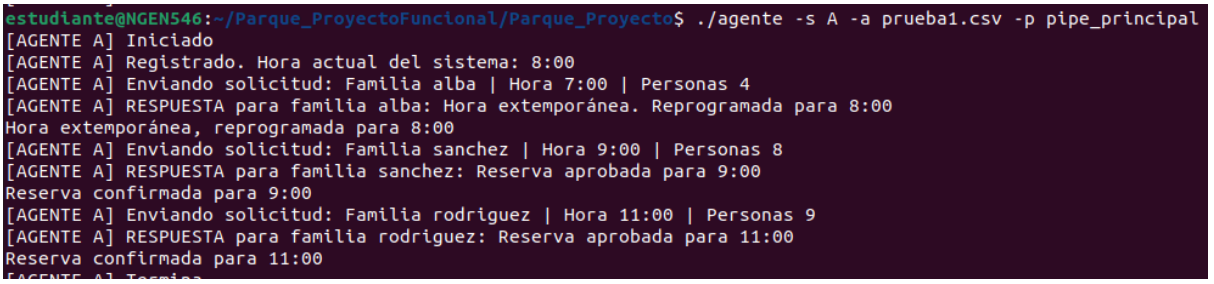

Imagen 6. Resultado de ejecución del Agente para el primer caso

5.2.1.1.3 Prueba #2

Objetivo: Verificar que programa en caso de tener una reserva a la misma hora de inicio del parque.

- Parámetro controlador para ejecutarlo: ./controlador -i 7 -f 14 -s 3 -t 30 -p principal_pipe
- Parámetro agente para ejecutarlo: ./agente -s A -a prueba2.csv -p principal_pipe
- Archivo.csv:
  alba,7,4
  sanchez,9,8
  rodriguez,11,9

5.2.1.1.4 Resultados de Prueba #2

Controlador:

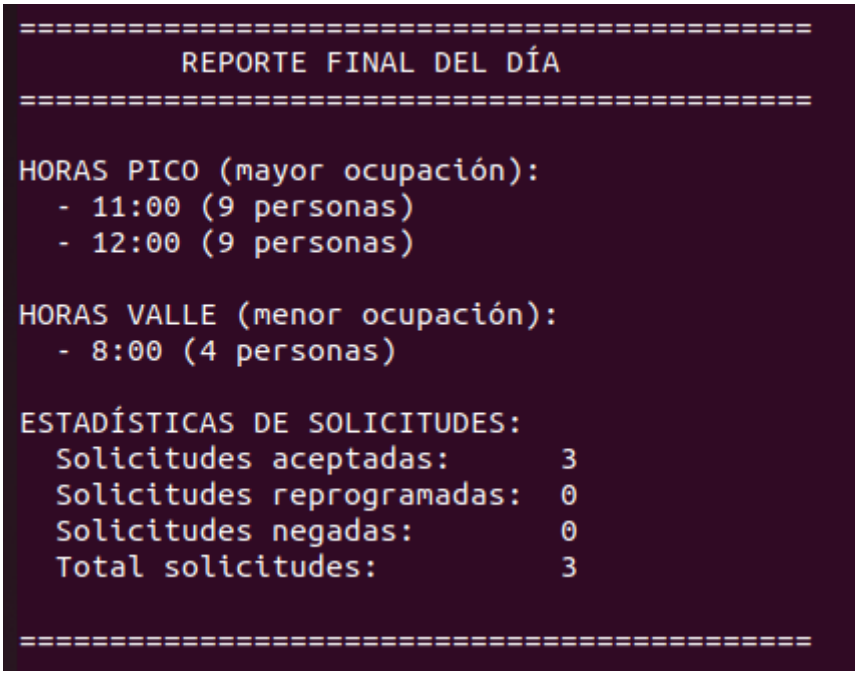

Imagen 7. Resultado de ejecución del Controlador para el segundo caso

Agente:

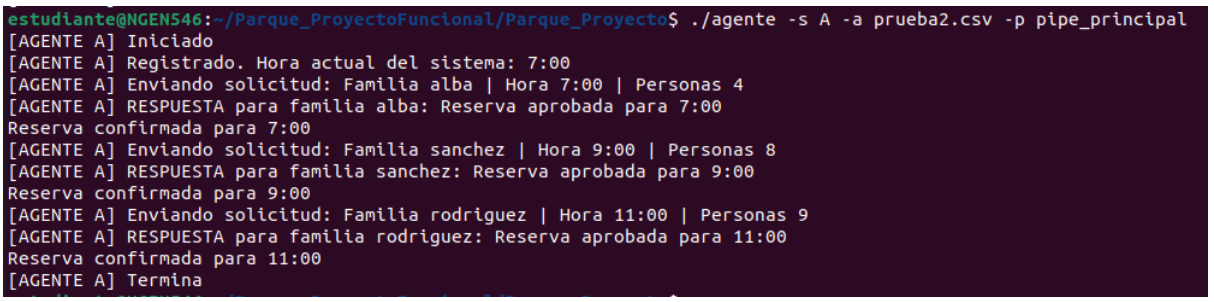

Imagen 8. Resultado de ejecución del agente para el segundo caso

5.2.1.1.5 Prueba #3

Objetivo: Verificar que programe bien si la reserva es posterior a la hora de apertura.

- Parámetro controlador: ./controlador -i 7 -f 14 -s 3 -t 30 -p principal_pipe
- Parámetro agente: ./agente -s A -a prueba3.csv -p principal_pipe

- Archivo.csv:
  alba,8,4
  sanchez,9,8
  rodriguez,11,9

5.2.1.1.6 Resultados de Prueba #3

Controlador:

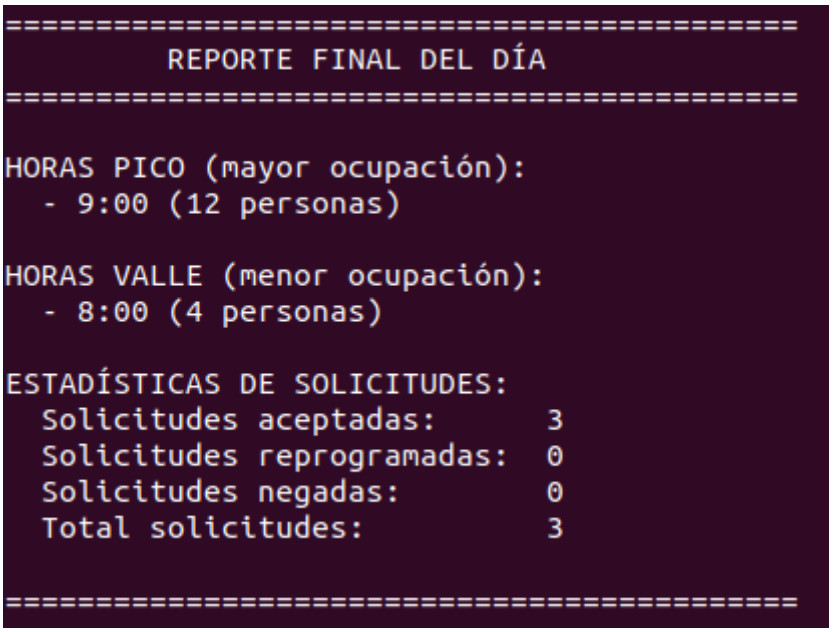

Imagen 9. Resultado de ejecución del controlador para el tercer caso

Agente:

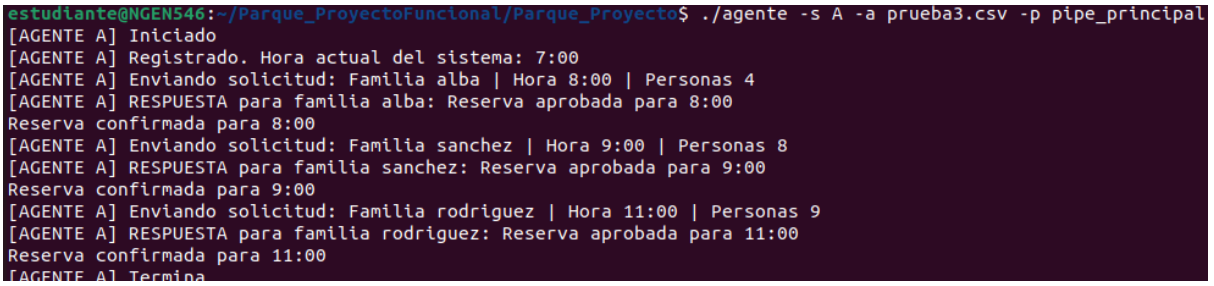

Imagen 10. Resultado de ejecución del agente para el tercer caso

5.2.1.1.7 Prueba #4

Objetivo: Verificar que niegue una reserva por aforo máximo;por hora posterior al cierre, que reprograme una reserva y que acepte una solicitud viable.

- Parámetro controlador: ./controlador -i 10 -f 18 -s 3 -t 30 -p principal_pipe
- Parámetro agente: ./agente -s A -a prueba4.csv -p principal_pipe
- Archivo.csv:
  lopez,12,35
  sanchez,8,5
  beltran,20,5
  duran,12,8

5.2.1.1.8 Resultados de Prueba #4

Controlador:

```
==========================================
         REPORTE FINAL DEL DÍA
==========================================

HORAS PICO (mayor ocupación):
  - 13:00 (8 personas)

HORAS VALLE (menor ocupación):
  - 12:00 (5 personas)

ESTADÍSTICAS DE SOLICITUDES:
  Solicitudes aceptadas:      1
  Solicitudes reprogramadas:  1
  Solicitudes negadas:        2
  Total solicitudes:          4

==========================================
```

Imagen 11. Resultado de ejecución del controlador para el cuarto caso

Agente:

```
estudiante@NGEN546:~/Parque_ProyectoFuncional/Parque_Proyecto$ ./agente -s A -a prueba4.csv -p pipe_principal
[AGENTE A] Iniciado
[AGENTE A] Registrado. Hora actual del sistema: 10:00
[AGENTE A] Enviando solicitud: Familia lopez | Hora 12:00 | Personas 35
[AGENTE A] RESPUESTA para familia lopez: Reserva negada: Número de personas (35) supera aforo máximo (30). Vuelva otro día
Reserva negada
[AGENTE A] Enviando solicitud: Familia sanchez | Hora 8:00 | Personas 5
[AGENTE A] RESPUESTA para familia sanchez: Hora extemporánea. Reprogramada para 11:00
Hora extemporánea, reprogramada para 11:00
[AGENTE A] Enviando solicitud: Familia beltran | Hora 20:00 | Personas 5
[AGENTE A] RESPUESTA para familia beltran: Reserva negada: Hora fuera del período de simulación
Reserva negada
[AGENTE A] Enviando solicitud: Familia duran | Hora 12:00 | Personas 8
[AGENTE A] RESPUESTA para familia duran: Reserva aprobada para 12:00
Reserva confirmada para 12:00
[AGENTE A] Termina
```

Imagen 12. Resultado de ejecución del agente para el cuarto caso

5.2.1.1.9 Prueba #5

Objetivo: Verificar que reprograme una reserva por aforo máximo en un mismo horario.

- Parámetro controlador: ./controlador -i 10 -f 18 -s 3 -t 30 -p principal_pipe
- Parámetro agente: ./agente -s A -a prueba5.csv -p principal_pipe
- Archivo.csv:
  avila,16,28
  mendez,16,5

5.2.1.1.10 Resultados de Prueba #5

Controlador:

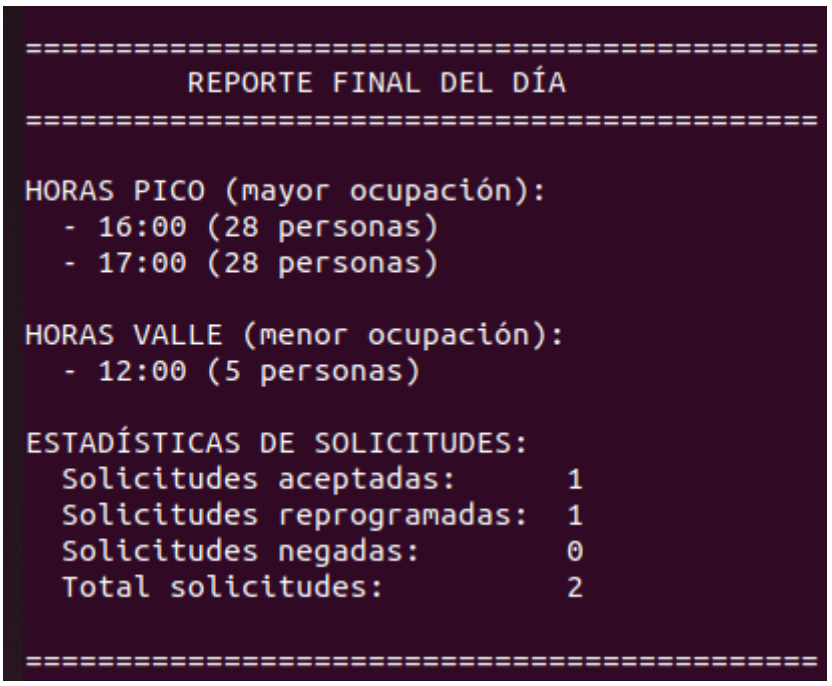

Imagen 13. Resultado de ejecución del controlador para el quinto caso

Agente:

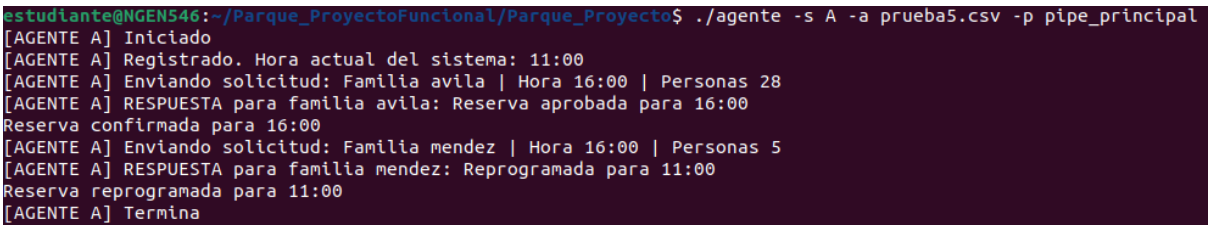

Imagen 14 Resultado de ejecución del agente para el quinto caso

5.2.1.1.11 Prueba #6

Objetivo: Verificar que reprograme una reserva por aforo máximo con horarios intercalados pues existe una hora de las dos que pueden estar que sobrepasa el aforo.

- Parámetro controlador: ./controlador -i 10 -f 18 -s 3 -t 30 -p principal_pipe
- Parámetro agente: ./agente -s A -a prueba6.csv -p principal_pipe
- Archivo.csv:
  avila,12,28
  mendez,13,5

5.2.1.1.12 Resultados de Prueba #6

Controlador:

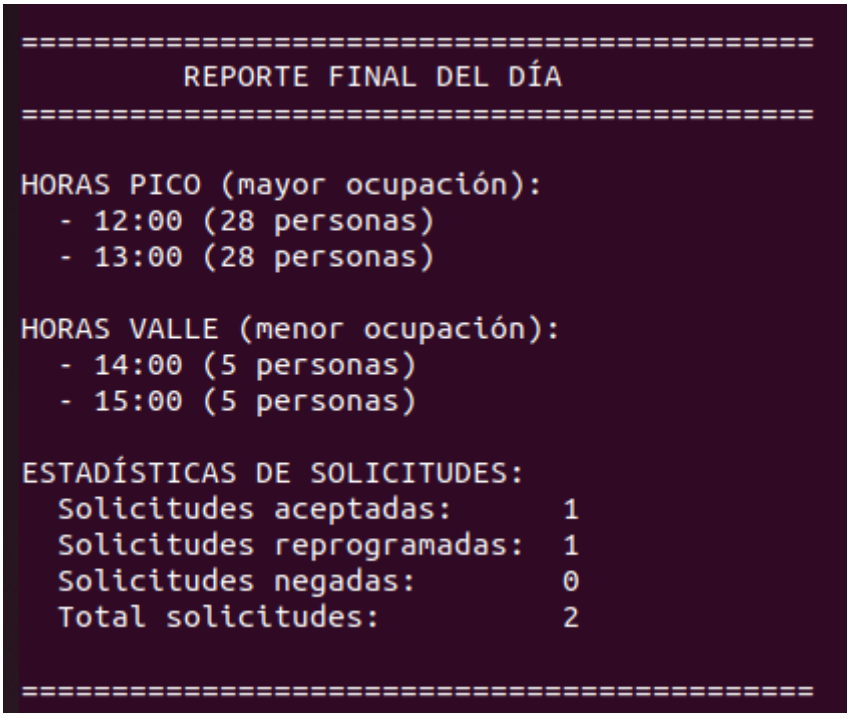

Imagen 15 Resultado de ejecución del controlador para el sexto caso

Agente:

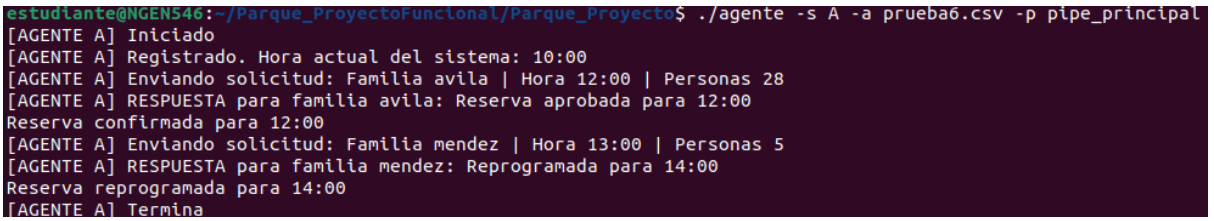

Imagen 16 Resultado de ejecución del agente para el sexto caso

6. Conclusiones

A partir del proyecto realizado acerca de un sistema de reservas, nos permitió comprender y aplicar de manera práctica la funcionalidad de un programa cliente/servidor, cómo se comunican y coordinan los diferente procesos dentro del sistema mediante pipes FIFO; los hilos para manejar tareas concurrentes con pthread, y la importancia de proteger los recursos compartidos utilizando mutex, evitando así condiciones de carrera y garantizando que los datos sean coherentes entre sí.

Además, permitió entender cómo cada componente del sistema cumple un rol específico dentro del flujo, por ejemplo, el controlador actúa como el componente central de coordinación gestionando horarios, verificando la disponibilidad de la reserva para luego darle respuesta al agente; mientras que los agentes funcionan como clientes que envían solicitudes y reaccionan según lo determinado por el controlador.

7. Referencias

- [1] Misistema-operativo, “Unidad 2: Procesos,” Blogspot. [En línea]. Disponible en: https://misistema-operativo.blogspot.com/p/unidad-2.html.
- [2] C. Gregg y N. Troccoli, Excerpts from "Lectura_Patrones_Semaforos (1).pdf", CS110 Lecture 11: Semaphores and Multithreading Patterns, Principles of Computer Systems, Winter 2021.
- [3] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.
- [4]IONOS Inc., “Pipes de Linux: explicamos el comando con ejemplos,” *Digital Guide*, 11/23/2022. [En línea]. Disponible: https://www.ionos.com/es-us/digitalguide/servidores/configuracion/pipes-linux
- [5]Misistema-operativo, “Unidad 2: Procesos,” Blogspot. Disponible en: https://misistema-operativo.blogspot.com/p/unidad-2.html