

# **Simulación de un Sistema de Reservas**

**SISTEMAS OPERATIVOS**

**JOHN CORREDOR**

**17 – 11 – 2025**

**JUAN MANUEL SOLANO – YOSEFH STEVEN PEÑA**



## ***Introducción***

Este documento presenta una síntesis detallada y estructurada del proyecto correspondiente al curso de Sistemas Operativos para el periodo académico 2025-30. El propósito central de este trabajo es diseñar, implementar y evaluar una simulación funcional de un sistema de reservas para un parque recreativo, integrando conceptos fundamentales de la programación concurrente y de la comunicación entre procesos dentro de un entorno Linux. Para lograrlo, el proyecto hace uso intensivo de procesos independientes, hilos POSIX (pthread) y mecanismos de comunicación basados en pipes nominales (FIFOs), los cuales permiten coordinar la interacción entre múltiples agentes simulados y un controlador central encargado de gestionar el aforo y la disponibilidad horaria. Además, la propuesta integra el manejo explícito de señales para modelar el paso del tiempo dentro de la simulación, así como la validación de operaciones y estados internos propios de un sistema concurrente de propósito real. En conjunto, esta implementación busca no solo resolver el problema planteado, sino también ofrecer un escenario práctico que permita comprender, aplicar y demostrar la correcta utilización de las llamadas al sistema y los mecanismos de sincronización esenciales en el diseño de sistemas operativos modernos.

## ***Descripción General***

El proyecto se centra en el diseño e implementación de un sistema de reservas para un parque privado denominado *Parque Berlín*, un espacio que, debido a su reducida capacidad y al alto flujo de visitantes durante las temporadas vacacionales, requiere un mecanismo eficiente que permita regular la entrada de familias por horas y mantener un control estricto del aforo. Para resolver esta problemática, se propone la construcción de una arquitectura basada en el modelo clásico **cliente/servidor**, donde el *Controlador de Reserva* actúa como servidor centralizado encargado de gestionar las solicitudes, validar la disponibilidad y asignar los cupos por hora. Paralelamente, se implementan múltiples *Agentes de Reserva*, que funcionan como procesos cliente independientes, encargados de generar y enviar peticiones de reserva en representación de diferentes familias.

La comunicación entre estos componentes se lleva a cabo mediante **pipes nominales (FIFOs)**, un mecanismo de comunicación entre procesos propio de los sistemas Unix/Linux, que permite establecer canales de datos unidireccionales y confiables sin necesidad de conexiones complejas. Cada agente envía sus solicitudes a través del pipe indicado, mientras que el controlador recibe, procesa y responde dichas peticiones según el estado interno de la simulación. Esta estructura no solo permite simular un sistema real de gestión de reservas, sino que también integra conceptos esenciales de los sistemas operativos, como manejo de concurrencia, sincronización, procesamiento paralelo y comunicación interprocesos, convirtiéndose en un escenario práctico ideal para el aprendizaje y la aplicación de técnicas avanzadas de programación en C.

## ***Detalles del Proyecto***

<https://github.com/YosefhDyn/ProyectoOperativos>

### ***Objetivos Principales***

- Implementar procesos e hilos POSIX.
- Utilizar mecanismos de sincronización y comunicación mediante pipes.
- Emplear correctamente las llamadas al sistema para gestión de procesos e hilos.

### ***Controlador de Reserva (Servidor)***

- Recibe solicitudes de los agentes y evalúa disponibilidad por hora.
- Administra el aforo del parque según las horas de simulación.
- Simula el paso del tiempo y actualiza entrada y salida de familias.
- Genera un reporte final del día con estadísticas de ocupación.
- Se ejecuta con parámetros como hora inicial, final, duración de hora simulada, aforo y el pipe de recepción.

### ***Agente de Reserva (Cliente)***

- Cada agente representa un proceso que solicita reservas.
- Se comunica con el controlador a través de un pipe indicado.
- Envía solicitudes contenidas en un archivo CSV.
- Valida horas solicitadas y espera respuestas.
- Imprime resultados y finaliza al terminar su archivo.

### ***Detalles Importantes de Implementación***

- Comunicación controlada mediante pipes nominales.
- Manejo de hilos en el controlador: uno para recibir solicitudes y otro para gestionar el reloj.
- Validación de horas dentro del rango 7–19.
- Manejo de cupos por hora y de reservas alternativas.
- Respuestas a solicitudes: aprobada, reprogramada, negada extemporánea o negada sin alternativas.

### ***Desarrollo***

El desarrollo del proyecto implica la creación y coordinación de procesos mediante fork() e hilos POSIX (pthread), los cuales permiten simular múltiples agentes actuando de manera paralela. Se requiere una sincronización adecuada para evitar condiciones de carrera, garantizando que el aforo y las reservas se actualicen de forma consistente. Además, el sistema utiliza manejadores de señales para simular el paso de las horas y activar eventos temporales dentro de la simulación. También incluye lectura estructurada de archivos para cargar configuraciones iniciales y validar parámetros como aforo, duración y número de agentes. La implementación debe asegurar concurrencia controlada, consistencia de datos, y una comunicación confiable entre procesos mediante pipes nominales (FIFOs), que transportan solicitudes y respuestas de forma ordenada incluso bajo alta carga de peticiones.

### ***Plan de Pruebas***

#### ***1. PRUEBAS FUNCIONALES BÁSICAS***

##### ***CP-01: Reserva Simple Aceptada***

Objetivo: Verificar que el sistema acepta una reserva con disponibilidad

Entrada:

Familia: Zuluaga

Hora: 8

Personas: 10

Aforo disponible: 50

Ejecución:

```
cd ProyectoSO  
./ejecutar.sh
```

Resultado:

```
OK ACEPTADA|Zuluaga|8|10|Reserva OK 8-10
```

Criterios de Éxito:

Mensaje de aceptación recibido

Familia registrada en hora 8 y 9 (2 horas)

Ocupación incrementa en 10 personas

Reporte final muestra la reserva

### ***CP-02: Múltiples Reservas Concurrentes***

Objetivo: Verificar que el sistema maneja múltiples agentes simultáneamente

Entrada:

Agente1: Zuluaga (8, 10 personas)

Agente2: Perez (9, 8 personas)

Agente3: Diaz (12, 6 personas)

Resultado:

```
OK Todas las reservas aceptadas  
OK Hora 9: 18 personas (10 de Zuluaga + 8 de Perez)  
OK Sin conflictos entre agentes
```

Criterios de Éxito:

- Los 3 agentes se registran correctamente
- Todas las solicitudes se procesan
- No hay condiciones de carrera
- Ocupación calculada correctamente

### ***CP-03: Reprogramación Automática***

Objetivo: Verificar que el sistema reprograma cuando no hay espacio

Entrada:

Primera reserva: Zuluaga (8, 10 personas) → ACEPTADA

Segunda reserva: Dominguez (8, 4 personas) → Debe reprogramarse

Preparación: Modificar aforo a valor bajo para forzar reprogramación

Resultado:

```
** REPROGRAMADA|Dominguez|9|4|Reprogramada a hora 9
```

Criterios de Éxito:

- Sistema detecta falta de espacio en hora 8
- Busca siguiente hora disponible
- Reprograma a hora 9
- Estadísticas: solicitudes reprogramadas = 1

#### ***CP-04: Simulación de Tiempo***

Objetivo: Verificar que el reloj avanza y gestiona entradas/salidas

Entrada:

Reserva: Zuluaga (8, 10 personas)

SegHoras: 2 (2 segundos = 1 hora)

Resultado:

```
** Hora actual: 8:00
Familias que entran: + Zuluaga (10 personas)
Ocupación actual: 10

** Hora actual: 9:00
Familias que entran: (ninguna)
Ocupación actual: 10

** Hora actual: 10:00
Familias que salen: - Zuluaga (10 personas)
Ocupación actual: 0
```

Criterios de Éxito:

- Reloj avanza cada 2 segundos
- Familias entran en hora reservada
- Familias salen después de 2 horas
- Ocupación se actualiza correctamente

## **2. PRUEBAS DE CASOS LÍMITE**

### ***CP-05: Aforo Máximo***

Objetivo: Verificar rechazo cuando grupo excede aforo

Entrada:

Familia: Grande (8, 60 personas)

Aforo: 50

Resultado:

```
** NEGADA_AFORO|Grande|8|60|Grupo excede aforo maximo
```

Criterios de Éxito:

- Solicitud rechazada inmediatamente
- No se registra la reserva
- Estadísticas: solicitudes\_negadas\_aforo = 1

### ***CP-06: Hora Fuera de Rango***

Objetivo: Verificar rechazo de horas inválidas

Entrada:

Familia: Temprano (6, 5 personas) → Antes de apertura

Familia: Tarde (20, 5 personas) → Despues de cierre

Rango válido: 8-19

Resultado:

```
** NEGADA_CUPO|Temprano|6|5|Hora fuera de rango
** NEGADA_CUPO|Tarde|20|5|Hora fuera de rango
```

Criterios de Éxito:

- Horas < 8 rechazadas
- Horas > 17 rechazadas (considerando 2 horas de duración)
- Estadísticas: solicitudes\_negadas\_cupo aumenta

### ***CP-07: Hora Extemporánea***

Objetivo: Verificar que agentes no envían solicitudes de horas pasadas

Entrada:

Hora actual: 9

Solicitud: Lopez (7, 5 personas) → Hora ya pasó

Resultado:

```
** Solicitud no enviada (hora 7 < hora actual 9) para familia Lopez
```

Criterios de Éxito:

- Agente detecta hora extemporánea
- Solicitud NO se envía al controlador
- Mensaje mostrado en consola del agente

#### **CP-08: Parque Lleno (Sin Cupo)**

Objetivo: Verificar rechazo cuando no hay horas disponibles

Entrada:

Aforo: 20

Reservas que llenan todas las horas 8-17

Nueva solicitud: Final (8, 10 personas)

Resultado:

```
** NEGADA_CUPO|Final|8|10|No hay cupo disponible
```

Criterios de Éxito:

- Sistema busca en todas las horas
- No encuentra espacio disponible
- Rechaza con NEGADA\_CUPO
- Estadísticas: solicitudes\_negadas\_cupo = 1

### **3. PRUEBAS DE CONCURRENCIA**

#### **CP-09: Sincronización con Mutex**

Objetivo: Verificar que no hay condiciones de carrera

Entrada:

3 agentes enviando solicitudes simultáneamente

Pausas de 2 segundos entre solicitudes

Resultado:

**OK Todas las solicitudes procesadas en orden correcto**  
**OK Ocupación sin errores de cálculo**  
**OK Estadísticas consistentes**

Criterios de Éxito:

- No hay sobrescritura de datos
- Ocupación = suma de todas las familias activas
- Reporte final con números correctos
- Sin errores de segmentación

### ***CP-10: Comunicación por Pipes***

Objetivo: Verificar comunicación bidireccional correcta

Entrada:

Registro de agente

Envío de solicitud

Recepción de respuesta

Resultado:

```
** Enviada solicitud: agente=Agente1, familia=Zuluaga...
** Respuesta controlador: RESP|ACEPTADA|Zuluaga|8|10|...
```

Criterios de Éxito:

- Pipe de entrada funciona (agente → controlador)
- Pipe de respuesta funciona (controlador → agente)
- Mensajes no se pierden
- No hay bloqueos (deadlocks)

## ***4. PRUEBAS DE MANEJO DE ERRORES***

### ***CP-11: Terminación Controlada (Ctrl+C)***

Objetivo: Verificar cierre limpio del sistema

Entrada:

Sistema en ejecución

Presionar Ctrl+C

Resultado:

```
^C
Señal de terminacion recibida. Cerrando sistema...
===== REPORTE FINAL =====
[... reporte completo ...]
Controlador finalizado.
```

Criterios de Éxito:

- Sistema captura señal SIGINT
- Genera reporte final antes de terminar
- Cierra pipes correctamente
- Libera memoria (sin memory leaks)

#### ***CP-12: Archivo CSV Vacío***

Objetivo: Verificar manejo de agente sin solicitudes

Entrada:

Archivo CSV con solo línea "Fin,0,0"

Resultado:

```
OK Agente registrado: Agente1...
** Solicitud no enviada (hora 0 < hora actual 8) para familia Fin
Agente Agente1 termina.
```

Criterios de Éxito:

- Agente se registra correctamente
- No envía solicitudes inválidas
- Termina limpiamente
- Sin errores de lectura de archivo

### **5. PRUEBAS DEL REPORTE FINAL**

#### ***CP-13: Cálculo de Estadísticas***

Objetivo: Verificar que el reporte final es correcto

Entrada:

5 solicitudes aceptadas

1 solicitud reprogramada

1 solicitud negada por aforo

1 solicitud negada sin cupo

Resultado:

```
===== REPORTE FINAL =====
Ocupacion por hora:
  Hora 8: 10 personas
  Hora 9: 22 personas
  ...
Horas pico: 9 10
Horas de menor ocupacion: 14 15 16 17 18
Solicitudes aceptadas: 5
Solicitudes reprogramadas: 1
Solicitudes negadas por aforo: 1
Solicitudes negadas sin cupo: 1
```

Criterios de Éxito:

- Ocupación histórica correcta (suma acumulada)
- Horas pico correctamente identificadas
- Horas valle correctamente identificadas
- Contadores de estadísticas precisos

## 6. MATRIZ DE CASOS DE PRUEBA

ID	Caso de Prueba	Prioridad	Estado	Resultado
CP-01	Reserva Simple Aceptada	Alta	OK	PASA
CP-02	Múltiples Reservas Concurrentes	Alta	OK	PASA
CP-03	Reprogramación Automática	Media	Esperando	Pendiente

CP-04	Simulación de Tiempo	Alta	OK	PASA
CP-05	Aforo Máximo	Alta	Esperando	Pendiente
CP-06	Hora Fuera de Rango	Media	Esperando	Pendiente
CP-07	Hora Extemporánea	Media	OK	PASA
CP-08	Parque Lleno	Baja	☒	Pendiente
CP-09	Sincronización con Mutex	Alta	Esperando	PASA
CP-10	Comunicación por Pipes	Alta	OK	PASA
CP-11	Terminación Controlada	Media	OK	PASA
CP-12	Archivo CSV Vacío	Baja	Esperando	Pendiente
CP-13	Cálculo de Estadísticas	Alta	OK	PASA

### **RESULTADOS ESPERADOS - EJEMPLO COMPLETO**

Configuración:

Rango: 8-19

Aforo: 50

SegHoras: 2

3 agentes con archivos CSV actuales

Salida Esperada Final:

```
===== REPORTE FINAL =====
Ocupacion por hora:
  Hora 8: 10 personas
  Hora 9: 22 personas
  Hora 10: 22 personas
  Hora 11: 13 personas
  Hora 12: 9 personas
  Hora 13: 6 personas
  Hora 14: 0 personas
  Hora 15: 0 personas
  Hora 16: 0 personas
  Hora 17: 0 personas
  Hora 18: 0 personas

Horas pico: 9 10
Horas de menor ocupacion: 14 15 16 17 18

Solicitudes aceptadas: 5
Solicitudes reprogramadas: 1
Solicitudes negadas por aforo: 0
Solicitudes negadas sin cupo: 0
```

### Referencias

- Documentación POSIX Threads (pthreads)
- [https://www.hostinger.com/tutorials/how-to-run-sh-file-in-linux?utm\\_campaign=Generic-Tutorials-DSA-t2|NT:Se|Lang:EN|LO:CO&utm\\_medium=ppc&gad\\_source=1&gad\\_campaignid=23](https://www.hostinger.com/tutorials/how-to-run-sh-file-in-linux?utm_campaign=Generic-Tutorials-DSA-t2|NT:Se|Lang:EN|LO:CO&utm_medium=ppc&gad_source=1&gad_campaignid=23)

208937126&gclid=Cj0KCQiArOvIBhDLARIIsAPwJXOYnR4Zu7n23Yu73NiAVEF  
MOy7MsyMW4Dkg4Iv4ZDdd-Lsk3VL\_yo-QaAuTQEALw\_wcB