SISTEMA DE RESERVAS PARA PARQUE ACUATICO

Samuel Montaña- Elier Ibarra

Pontificia Universidad Javeriana, Bogotá D.C, Colombia
Facultad de Ingeniería de Sistemas
Sistemas Operativos
17 de Noviembre de 2025

1. Introducción

En este proyecto desarrollamos en C un sistema de reservas para un parque acuático, aplicando varios temas de la materia de Sistemas Operativos: procesos, hilos (pthread) y comunicación entre procesos con **pipes nominales (FIFOs)** en Linux.

La idea general es que exista un programa servidor, llamado **controlador**, que maneja el aforo del parque y la hora de la simulación, y varios programas clientes, llamados **agentes**, que se encargan de leer solicitudes de familias desde un archivo y enviárselas al servidor. El servidor decide si la reserva se acepta en la hora pedida, se reprograma para otra hora o se rechaza, según las reglas del enunciado.

Más allá de que el programa compile, el objetivo fue aterrizar los conceptos de la clase en algo práctico y ver cómo se combinan hilos, pipes y estructuras de datos en un mismo ejercicio.

2. Objetivos

2.1 Objetivo general

Implementar un sistema de reservas para un parque acuático usando C en Linux, donde un proceso servidor y varios procesos clientes se comuniquen por pipes nominales y se use un hilo para simular el paso del tiempo, respetando aforo y horario.

2.2 Objetivos específicos

- Diseñar e implementar el **Controlador de Reserva** que:
    - Reciba parámetros por línea de comandos (hora inicial, hora final, segundos por hora, aforo y pipe).
    - Mantenga la hora de la simulación con un hilo tipo “reloj”.
    - Lea solicitudes desde un pipe nominal y decida si acepta, reprograma o niega.
    - Genere un reporte final con estadísticas de ocupación y de solicitudes.
- Diseñar e implementar el **Agente de Reserva** que:
    - Lea solicitudes desde un archivo CSV (familia, hora, personas).
    - Se conecte al pipe del servidor, lo salude y reciba la hora actual de la simulación.
    - Envíe las solicitudes al servidor y muestre por pantalla las respuestas.
- Organizar el proyecto en varios archivos (controlador.c, controlador.h, agente.c, agente.h, Makefile) para que sea más fácil de mantener y compilar con make.

3. Descripción del problema

El parque acuático tiene un horario de funcionamiento y un **aforo máximo** de personas. Cada familia quiere entrar a cierta hora y, si es aceptada, se queda **exactamente 2 horas** dentro del parque. El servidor debe controlar que:

- Nunca se supere el aforo máximo en ninguna hora.
- Las reservas solo se puedan hacer entre la hora inicial y la hora final de la simulación.
- Cuando una familia pide una hora que ya pasó, la reserva se considere **extemporánea** y solo se intente reprogramar si todavía hay huecos más adelante.

Reglas que aplicamos en el código:

1. Si en la hora pedida y en la siguiente hay suficiente cupo, la reserva se **acepta en la hora original**.
2. Si en la hora pedida no hay espacio, pero más adelante existe un bloque de 2 horas seguidas con cupo, se **reprograma** la reserva.
3. Si ya pasó la hora pedida o no se encuentra ningún bloque disponible, la reserva se **rechaza**.
4. Durante la simulación se cuenta:
    - Cuántas reservas fueron aceptadas.
    - Cuántas fueron reprogramadas.
    - Cuántas fueron negadas.

Al final, el sistema muestra un resumen con las horas de mayor ocupación y las de menor ocupación.

4. Descripción de la solución

4.1 Forma de ejecución

Servidor: controlador

El servidor se ejecuta así:

./controlador -i ini -f fin -s seg -t aforo -p pipeRecibe

Donde:

- -i ini → hora inicial de la simulación.
- -f fin → hora final de la simulación.
- -s seg → segundos reales que equivalen a una hora simulada.
- -t aforo → aforo total máximo del parque.
- -p pipeRecibe → nombre del FIFO por el cual llegan las solicitudes (por ejemplo /tmp/pipe_reservas).

Cliente: agente

Cada agente se ejecuta así:

./agente -s nombre -a archivo.csv -p pipe_servidor

Donde:

- -s nombre → nombre del agente (por ejemplo Agente1).
- -a archivo.csv → archivo con las solicitudes de la forma Familia,hora,personas.

- -p pipe_servidor → nombre del pipe del servidor (debe ser el mismo que se usó en -p del controlador).

4.2 Estructuras y archivos principales

4.2.1 controlador.h

En este archivo se definen las estructuras principales del servidor:

- config_t: guarda la configuración de la simulación:

```
typedef struct {
   int start_hour;        // Hora inicial de simulación
   int end_hour;          // Hora final de simulación
   int seg_por_hora;      // Segundos por hora simulada
   int aforo_total;       // Aforo máximo del parque
   char pipe_recibe[128]; // Nombre del pipe donde llegan las peticiones
} config_t;
```

- hora_t: lleva la ocupación por hora:

```
typedef struct {
   int hour;       // Número de la hora
   int ocupacion;  // Personas reservadas en esa hora
} hora_t;
```

También se declara sim_hour para la hora actual simulada y un arreglo horas[] con la información de cada hora.

4.2.2 agente.h

En el agente usamos:

```
typedef struct {
   char nombre[MAX_NAME];      // Nombre del agente
   char archivo[128];          // Archivo CSV con solicitudes
   char pipe_servidor[128];    // Pipe donde envía las solicitudes
   char pipe_local[128];       // Pipe propio donde recibe respuestas
} agente_cfg_t;
```

Esta estructura (acfg) concentra todo lo que el agente necesita para funcionar.

5. Funcionamiento del servidor (controlador.c)

5.1 Configuración inicial

En el main del controlador se leen los argumentos de línea de comandos (-i, -f, -s, -t, -p). Si faltan parámetros, el programa muestra un mensaje de uso y termina.

Luego se llena la estructura cfg con los valores leídos y se inicializa el arreglo de horas, asignando para cada una su número y ocupación

inicial 0. Después se crea el FIFO principal con mkfifo(cfg.pipe_recibe, 0666) si aún no existe.

5.2 Hilos utilizados

El servidor crea dos hilos:

1. **Hilo reloj (thread_reloj)**
    - Se encarga de simular el paso del tiempo.
    - Usa sleep(cfg.seg_por_hora); para pasar de una hora a la siguiente.
    - Va aumentando sim_hour desde cfg.start_hour hasta cfg.end_hour.
    - Dentro de este hilo se imprimen mensajes que indican la hora simulada actual y se podrían listar las entradas y salidas de familias.
2. **Hilo escucha (thread_escucha)**
    - Abre el FIFO cfg.pipe_recibe.
    - Lee líneas de texto que llegan desde los agentes.
    - Cada línea se analiza con strtok para saber si es un comando REGISTER (registro de agente) o REQUEST (solicitud de reserva).
    - Llama a funciones para registrar el agente o procesar la solicitud.

Como los dos hilos acceden a variables globales (como horas[] y contadores de solicitudes), se usa un **mutex** para proteger esos accesos y evitar condiciones de carrera.

5.3 Registro de agentes

Cuando llega un comando REGISTER, el servidor:

- Lee el nombre del agente y el nombre de su pipe local.
- Envía un mensaje de respuesta a ese pipe con la hora actual de la simulación (sim_hour), por ejemplo: HORA_ACTUAL,8

De esta forma, el agente sabe desde qué hora va el sistema en ese momento y puede decidir qué solicitudes tiene sentido mandar.

5.4 Procesamiento de reservas

Cuando llega un comando REQUEST, el servidor extrae:

- Nombre de la familia.
- Hora solicitada.
- Número de personas.
- Pipe del agente para responder.

Luego aplica las reglas:

1. Verifica que el grupo no sobrepase el aforo por sí solo.
2. Verifica que la hora esté entre start_hour y end_hour.
3. Si la hora solicitada ya es menor que sim_hour, se considera una solicitud extemporánea y solo se intenta reprogramar si hay huecos más adelante.
4. Si la hora aún no ha pasado:
    - Primero se intenta reservar en esa misma hora (horas h y h+1).
    - Si no hay espacio, se busca otro bloque de 2 horas seguidas disponible.

5. Si se encuentra una hora adecuada:
    - Se actualiza el arreglo horas[] sumando el número de personas en las 2 horas.
    - Se incrementan los contadores de solicitudes aceptadas o reprogramadas.
    - Se envía una respuesta al agente indicando si fue en la hora original o en una nueva hora.
6. Si no se encuentra ningún hueco:
    - La solicitud se niega, se incrementa el contador de negadas y se envía la respuesta correspondiente.

5.5 Reporte final

Al finalizar la simulación (cuando se supera la hora final y se cierran los hilos), el servidor:

- Recorre el arreglo de horas para determinar:
    - La ocupación máxima.
    - La ocupación mínima.
- Imprime:
    - Horas con ocupación máxima (horas pico).
    - Horas con ocupación mínima.
    - Número de solicitudes negadas, aceptadas y reprogramadas.

Esto permite ver rápidamente cómo se comportó el sistema durante el “día” simulado.

6. Funcionamiento del agente (agente.c)

6.1 Parámetros y pipes

En el main del agente se leen los parámetros:

- -s nombre → se guarda en acfg.nombre.
- -a archivo.csv → se guarda en acfg.archivo.
- -p pipe_servidor → se guarda en acfg.pipe_servidor.

Luego se construye el nombre del pipe local:

snprintf(acfg.pipe_local, sizeof(acfg.pipe_local), "/tmp/pipe_%s", acfg.nombre);

Si ese pipe no existe, se crea con mkfifo.

6.2 Registro con el servidor

En registrar_agente():

- Se abre el pipe del servidor en modo escritura.
- Se arma un mensaje REGISTER,<nombre>,<pipe_local>\n.
- Se envía por el pipe del servidor.
- Luego el agente abre su propio pipe local en modo lectura y espera una respuesta del tipo HORA_ACTUAL,<sim_hour>, que indica la hora actual de la simulación.

6.3 Envío de solicitudes

En ejecutar_agente():

- Se abre el archivo CSV con las solicitudes.
- Cada línea se separa en familia, hora y personas.
- Para cada solicitud válida se arma un mensaje de tipo REQUEST con los datos y se envía al servidor por el pipe.
- Después se leen las respuestas desde el pipe local del agente, donde el controlador va diciendo si la reserva se aceptó, se reprogramó o se negó.

Al final, el agente cierra sus descriptores y borra su pipe local con unlink(acfg.pipe_local);.

7. Compilación y prueba

El proyecto incluye un Makefile que facilita la compilación.

Para compilar:

make

Esto genera los ejecutables:

- controlador
- agente

Ejemplo de prueba:

En una terminal (servidor):

./controlador -i 8 -f 12 -s 5 -t 30 -p /tmp/pipe_reservas

1. En otra terminal (agente):

./agente -s Agente1 -a solicitudes.csv -p /tmp/pipe_reservas

2. Con esto se puede observar cómo el servidor acepta, reprograma o niega reservas según el aforo y las horas.

8. Conclusiones

Como equipo pudimos implementar un sistema de reservas sencillo que cumple con lo que se pedía en el enunciado y, al mismo tiempo, nos permitió practicar varios temas importantes de Sistemas Operativos.

- Entendimos mejor cómo dos programas distintos (servidor y cliente) se pueden comunicar usando **pipes nominales**.
- Vimos en la práctica el uso de **hilos** para separar la simulación del tiempo de la lógica de atención de solicitudes.
- Usamos **mutex** para proteger variables globales y evitar problemas de concurrencia.
- Modelamos el tema del aforo con una estructura de ocupación por hora, lo que nos obligó a pensar en términos de estado del sistema y no solo en entradas y salidas.