SISTEMA DISTRIBUÍDO DE SERVICIOS DE TAQUILLA VIRTUAL

Documento de Diseño

PCCD - 2023/24

Aarón Riveiro Vilar
Adrián Cabaleiro Althoff
Iván Pillado Rodríguez
Manuel Martínez Vaamonde

INTRODUCCIÓN

El objetivo del proyecto es el desarrollo de un sistema distribuído que ofrezca servicios de taquilla virtual. En concreto, nos centraremos en el desarrollo de los nodos del sistema, encargados de recibir solicitudes de los clientes, procesarlas, y dirigirlas a una base de datos ficticia. Todo esto garantizando la exclusión mutua cuando sea necesario, y permitiendo la máxima concurrencia posible.

En cada nodo se podrán ejecutar uno, varios, o ninguno de los siguientes procesos:

- Proceso de Pagos
- Proceso de Anulaciones
- Proceso de Reservas
- Proceso de Administración
- Proceso de Consultas

REQUISITOS

- **REQUISITOS NO FUNCIONALES:**
    - ➢ La base de datos es monolítica, no se puede acceder solo a parte de ella
    - ➢ Será un sistema distribuído con un número dinámico de nodos
    - ➢ Los nodos se simularán en un único ordenador y se comunicarán mediante el paso de mensajes
    - ➢ Se supondrá que el sistema no sufrirá caídas y no será necesario recuperar el estado
    - ➢ Los procesos estarán ejecutando su sección crítica durante un período finito de tiempo
    - ➢ Las operaciones de mantenimiento conllevarán una parada total del sistema

- **REQUISITOS FUNCIONALES:**

  - ➢ Hay cinco tipos de procesos: Pagos, Anulaciones, Reservas, Administración y Consultas
  - ➢ Los procesos de Pagos se ejecutan en exclusión mutua con todos los demás procesos
  - ➢ Los procesos de Anulaciones se ejecutan en exclusión mutua con todos los demás procesos
  - ➢ Los procesos de Reservas se ejecutan en exclusión mutua con todos los demás procesos
  - ➢ Los procesos de Administración se ejecutan en exclusión mutua con todos los demás procesos
  - ➢ Debe existir un sistema de prioridades para los diferentes tipos de proceso

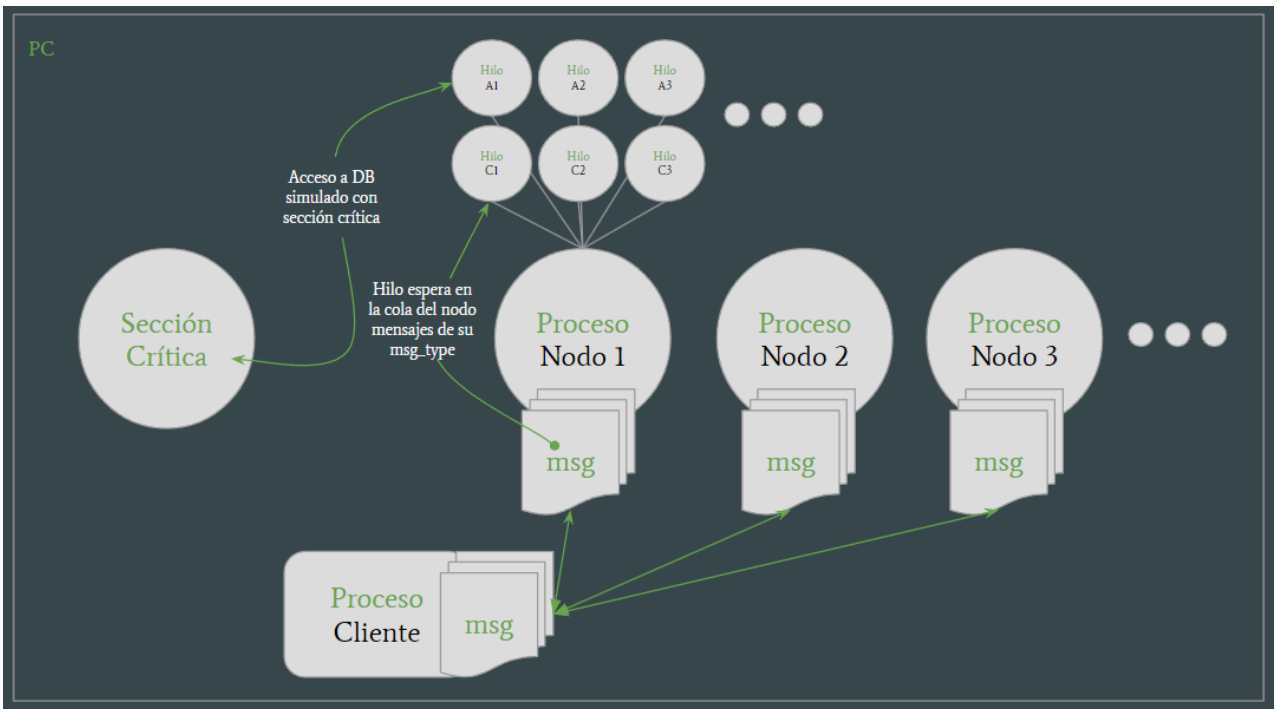
PRIORIDADES

Se ha implementado un sistema de tres niveles de prioridad para garantizar que aquellos procesos más importantes son atendidos primero.

- **Nivel de prioridad 0:** nivel de prioridad más alto, aquí están los procesos de PAGOS y ANULACIONES
- **Nivel de prioridad 1:** nivel de prioridad intermedio, aquí están los procesos de RESERVAS y ADMINISTRACIÓN
- **Nivel de prioridad 2:** nivel de prioridad más bajo, aquí están los procesos de CONSULTAS

Con este sistema de prioridades, los procesos de CONSULTAS podrían sufrir inanición. Este problema podría solucionarse implementando adelantamientos, de procesos CONSULTAS a procesos de prioridad uno.

TOPOLOGÍA



ALGORITMO

Se ha implementado el algoritmo de paso de testigo de Ricart-Agrawala, con las modificaciones necesarias para cumplir con los requisitos ya mencionados. Entre otras, se han hecho modificaciones para implementar lo siguiente:

- Sistema de prioridades
- Concurrencia de lectores
- Exclusión mutua de escritores

ESTRUCTURA

- **nodo.c**
  - Parámetros de entrada:
    - **`ID`**: identificador del nodo, empezando desde cero
    - **`procesos_pagos`**: número de procesos de tipo PAGOS que se ejecutarán en el nodo
    - **`procesos_anulaciones`**: número de procesos de tipo ANULACIONES que se ejecutarán en el nodo
    - **`procesos_reservas`**: número de procesos de tipo RESERVAS que se ejecutarán en el nodo
    - **`procesos_administracion`**: número de procesos de tipo ADMINISTRACIÓN que se ejecutarán en el nodo
    - **`procesos_consultas`**: número de procesos de tipo CONSULTAS que se ejecutarán en el nodo
  - Funcionamiento: este programa simula los diferentes nodos del sistema distribuído. Contiene toda la lógica del algoritmo, para gestionar el sistema de prioridades, concurrencia, exclusión mutua…
    Cada nodo creará tantos procesos como se indique en los parámetros de entrada, que esperarán a recibir el mensaje de un cliente en una cola de mensajes creada por el nodo.

- **utils.c**
  - Funcionamiento: este programa contiene las definiciones de varias funciones usadas en nodo.c, con el objetivo de reutilizar código y facilitar el mantenimiento.

- **utils.h**
  - Funcionamiento: fichero de cabecera de utils.c.

- **ejecutar_nodo.sh**
  - Parámetros de entrada:
    - **`N`**: número de nodos a inicializar
    - **`procesos_pagos`**: número de procesos de tipo PAGOS que se ejecutarán en cada nodo
    - **`procesos_anulaciones`**: número de procesos de tipo ANULACIONES que se ejecutarán en cada nodo
    - **`procesos_reservas`**: número de procesos de tipo RESERVAS que se ejecutarán en cada nodo
    - **`procesos_administracion`**: número de procesos de tipo ADMINISTRACIÓN que se ejecutarán en cada nodo
    - **`procesos_consultas`**: número de procesos de tipo CONSULTAS que se ejecutarán en cada nodo
    - **`timeout`**: el tiempo máximo, en segundos, que estarán ejecutándose los nodos
  - Funcionamiento: este script permite automatizar la inicialización de varios nodos en una sola ejecución. Tras pasar la cantidad de tiempo indicada por el parámetro timeout, todos los nodos terminarán automáticamente.

- **cliente.c**
  - Parámetros de entrada:
    - **`N`**: número de nodos del sistema
  - Funcionamiento: este programa muestra un menú interactivo que permite enviar una solicitud a un nodo. Primero se pregunta con qué nodo se desea comunicar, y a continuación qué tipo de solicitud. Por último, el programa se encarga de enviar el mensaje, y vuelve a mostrar el menú.

- **cliente_rand.c**
  - Parámetros de entrada:
    - **N**: número de nodos del sistema
    - **`cantidad_de_solicitudes`**: el número total de solicitudes a enviar
    - **`tiempo_maximo_entre_solicitud`**: el máximo tiempo, en milisegundos, que puede transcurrir entre el envío de dos solicitudes
    - **`solicitudes_pagos`**: si se enviarán o no (1 o 0) solicitudes de tipo PAGOS
    - **`solicitudes_anulaciones`**: si se enviarán o no (1 o 0) solicitudes de tipo ANULACIONES
    - **`solicitudes_reservas`**: si se enviarán o no (1 o 0) solicitudes de tipo RESERVAS
    - **`solicitudes_administración`**: si se enviarán o no (1 o 0) solicitudes de tipo ADMINISTRACIÓN
    - **`solicitudes_consultas`**: si se enviarán o no (1 o 0) solicitudes de tipo CONSULTAS
  - Funcionamiento: este programa permite enviar múltiples solicitudes a múltiples nodos en una sola ejecución, y de forma aleatorizada. Entre cada envío, se esperará aleatoriamente entre cero milisegundos, y el valor especificado por tiempo_maximo_entre_solicitud. Para cada envío, se seleccionará aleatoriamente un nodo del sistema, y un tipo de solicitud de entre los permitidos por los parámetros de entrada.

- **kill.c**
  - Parámetros de entrada:
    - **N**: número de nodos del sistema
  - Funcionamiento: este programa envía un tipo de mensaje especial a todos los nodos del sistema, para indicarles que deben finalizar su ejecución.

PSEUDOCÓDIGO

- Procesos prioridad 0:

```
LOOP:
    esperar_peticion()
    quiere[0]++
    IF (NOT token OR NOT lista_vacia()) AND NOT peticion_activa(0):
       broadcast_peticion(0)
    IF nodo_activo:
       cola_t0++
       wait(cola_t0_sem)
    ELSE:
       nodo_activo = 1
    IF NOT token:
       token = recv(TOKEN)
       actualizar_atendidas(token.vec_atendidas)
       token = 1
    wait(mutex_sc)
    sc = 1
    //SC
    sc = 0
    signal(mutex_sc)
    quiere[0]--
    IF quiere[0] == 0:
       vector_atendidas[0][id_nodo] = vector_peticiones[0][id_nodo]
    nodo_siguiente = buscar_nodo_siguiente()
    IF nodo_siguiente > 0:
       token = 0
       enviar_token(nodo_siguiente)
    IF procesos_quieren():
       IF nodo_siguiente > 0:
             hacer_peticiones()
       despertar_siguiente()
    ELSE:
       nodo_activo = 0
```

- Procesos Prioridad 1:

```
LOOP:
    esperar_peticion()
    quiere[1]++
    DO:
       proceso_despertado = 0
       IF (NOT token OR NOT lista_vacia()) AND NOT peticion_activa(1):
             broadcast_peticion(1)
       IF nodo_activo:
             cola_t1++
             wait(cola_t1_sem)
       ELSE:
             nodo_activo = 1
       IF NOT token:
             token = recv(TOKEN)
             actualizar_atendidas(token.vec_atendidas)
             token = 1
       IF quiere[0] > 0:
             proceso_despertado = 1
             despertar_siguiente();
    WHILE proceso_despertado
    wait(mutex_sc)
    sc = 1
    //SC
    sc = 0
    signal(mutex_sc)
    quiere[1]--
    IF quiere[1] == 0:
       vector_atendidas[1][id_nodo] = vector_peticiones[1][id_nodo]
    nodo_siguiente = buscar_nodo_siguiente()
    IF nodo_siguiente > 0:
       token = 0
       enviar_token(nodo_siguiente)
    IF procesos_quieren():
       IF nodo_siguiente > 0:
             hacer_peticiones()
       despertar_siguiente()
    ELSE:
       nodo_activo = 0
```

- Procesos Prioridad 2:

```
LOOP:
    esperar_peticion()
    quiere[2]++
    DO:
       proceso_despertado = 0
       IF NOT (token OR token_consulta) AND NOT peticion_activa(2):
             broadcast_peticion(2)
       IF nodo_activo:
             IF NOT paso_consultas:
                   cola_t2++
                   wait(cola_t2_sem)
       ELSE:
             nodo_activo = 1
       IF NOT (token OR token_consulta):
             token = recv(TOKEN)
             actualizar_atendidas(token.vec_atendidas)
             IF token.consulta:
                   token_consulta = 1
                   token_consulta_origen = token.id_nodo_origen
             ELSE:
                   token = 1
       IF quiere[0] > 0 OR quiere[1] > 0:
             IF token_consulta:
                   devolver_token_consulta(token_consulta_origen)
             proceso_despertado = 1
             despertar_siguiente()
    WHILE proceso_despertado
    IF primera_consulta:
       primera_consulta = 0
       wait(mutex_sc)
       sc = 1
       paso_consultas = 1
       despertar_todos_t2()
    consultas_sc++
    // SC
    consultas_sc--
    quiere[2]--
    IF quiere[2] == 0:
       vector_atendidas[2][id_nodo] = vector_peticiones[2][id_nodo]
    nodo_siguiente = buscar_nodo_siguiente()
    IF nodo_siguiente > 0 OR quiere[0] OR quiere[1]:
       paso_consultas = 0
       IF consultas_sc == 0:
             IF token_consultas:
```

```
                    token_consulta = 0
                    devolver_token_consulta(token_consulta_origen)
              ELSE:
                    IF NOT lista_vacia():
                          wait(lista_vacia_sem)
              sc = 0
              signal(mutex_sc)
              primera_consulta = 1
              nodo_siguiente = buscar_nodo_siguiente()
              IF token:
                    IF nodo_siguiente > 0:
                          token = 0
                          enviar_token(nodo_siguiente)
              IF procesos_quieren():
                    IF nodo_siguiente > 0:
                          hacer_peticiones()
                    despertar_siguiente()
              ELSE:
                    nodo_activo = 0
```

- Procesos Receptor:

```
LOOP:
      peticion = recv(PETICION)
      IF peticion.devolucion:
            quitar_lista(peticion.id_nodo)
            IF lista_vacia():
                  signal(lista_vacia_sem)
      ELSE:
            vector_peticiones[peticion.id_nodo] =
max(vector_peticiones[peticion.id_nodo], peticion.num_peticion)
            IF token AND NOT sc AND prioridad_superior(peticion.prioridad)
AND vector_peticiones[peticion.id_nodo] >
vector_atendidas[peticion.id_nodo]:
                  token = 0
                  enviar_token(peticion.id_nodo)
            ELSE IF paso_consultas AND peticion.prioridad == 2 AND
vector_peticiones[2][peticion.id_nodo] >
vector_atendidas[2][peticion.id_nodo]:
                  enviar_token_consulta(peticion.id_nodo)
```