PROGRAMACIÓN DE SISTEMAS CONCURRENTES Y DISTRIBUIDOS
Dorian Boleslaw Wozniak 817570
Jaime Velasco Gimeno 816818

Práctica 5

DISEÑO:

En esta práctica, se vuelve a realizar una simulación de un locutorio, con clientes, cabinas y un proceso de limpieza periódico, pero esta vez se requiere una implementación distribuida síncrona a través de un modelo cliente-servidor.

Por un lado, va a haber dos tipos de procesos cliente: “usuario” y “limpiador”. Estos realizarán peticiones al servidor enviando cadenas de caracteres, que el servidor deberá interpretar y contestar adecuadamente.

Por otro lado, habrá un servidor que se mantendrá a la escucha de peticiones de ambos clientes, y realizará acciones conforme a la información recibida. Esta gestión de peticiones se realizará de forma concurrente, creando en el caso específico de los clientes “usuario” nuevos threads a medida que se va realizando nuevas conexiones con dichos clientes.

La comunicación entre clientes y servidor se realizará de forma síncrona mediante el uso de sockets, configurados para usar el protocolo TCP, que permite intercambios con conexión (el servidor debe recibir una petición de conexión explicita del cliente) fiables. Los clientes deberán crear, cada uno, un socket con la dirección y puerto del servidor al que contactar. El servidor creará dos sockets en puertos diferentes y dirección local que se mantendrán a la escucha de peticiones de conexión. Cada vez que acepte una conexión, obtendrá un descriptor de fichero para el cliente que realizó la petición.

El sistema, una vez establecidas las conexiones, tendrá la siguiente estructura:

<table>
<tr><td colspan="2">PETICIONES Y RESPUESTAS EXPLÍCITAS (EN ORDEN DE ENVIO/RECEPCIÓN)</td></tr>
<tr><td>Usuario</td><td>Respuesta del servidor</td></tr>
<tr><td>“RESERVAR_CABINA,i”
“LIBERAR_CABINA,c”
“FIN”</td><td>“CONCEDIDA,c”
No responde
Información de uso</td></tr>
<tr><td>Limpiador</td><td>Respuesta del servidor</td></tr>
<tr><td>“LIMPIEZA_ENTRAR”
“LIMPIEZA_SALIR”
“FIN”</td><td>“OK_LIMPIEZA”
No responde
Información de uso</td></tr>
<tr><td colspan="2">ACCIONES DEL SERVIDOR AL RECIBIR PETICIONES</td></tr>
<tr><td colspan="2">“RESERVAR_CABINA,i”: Comprueba si puede entrar en el locutorio. Si hay al menos una cabina libre y no está ocupada, la encuentra y devuelve su número (c)
“LIBERAR_CABINA,c”: Libera la cabina c utilizada por el usuario
“LIMPIEZA_ENTRAR”: Avisa que se va a limpiar y espera hasta que el locutorio esté vacío, restringiendo mientras el acceso a nuevos clientes.
“LIMPIEZA_SALIR”: Indica que se ha dejado de limpiar, permitiendo nuevamente el acceso a otros usuarios
“FIN”: Devuelve, una vez finalizado el cuerpo del proceso cliente, la información de uso de la conexión realizada</td></tr>
</table>

El proceso clientes usuario y limpiador realizarán varias veces su acción antes de pedir la información de uso, con el tiempo total de uso de la cabina, y acabar. El servidor cerrará una vez reciba el mensaje “FIN” de todas sus conexiones activas. Las acciones de control de las cabinas se realizarán con las mismas consideraciones de concurrencia de las prácticas anteriores.

PSEUDOCÓDIGO:

```
channel of character array ch_user, ch_cleaner

--- PROCESOS CLIENTE ---

Process usuario(i := 1..N_USER)::
   for j := 1..N_TIMES_USER
      ch_user <= "RESERVAR_CABINA,i"    - i es el id de cliente
      ch_user => "CONCEDIDA,c"          - c es el n° de cabina
      espera(aleatorio(10, 40))
      ch_user <= "LIBERAR_CABINA,c"
      - No espera recibir respuesta
      espera(aleatorio(20,50))
   end

   ch_user <= "FIN"
   ch_user => "Uso total de la cabina: xx.xx" -xx.xx es el tiempo de uso
end

Process limpiador::
      for j := 1..N_TIMES_CLEANING
      espera(PER_CLEANING)
      ch_cleaner <= "LIMPIEZA_ENTRAR"
      ch_cleaner => "OK_LIMPIEZA"
      espera(aleatorio(80, 120))
      ch_cleaner <= "LIMPIEZA_SALIR"
      - No espera recibir respuesta
   end

   ch_cleaner <= "FIN"
   ch_cleaner => "Tiempo total limpiando: xx.xx"
end

--- PROCESOS SERVIDOR ---

Process servir_usuario(i := 1..N_USER)::
   boolean fin := false
   integer c, i, timer := 0
   character array s

   while not fin
      ch_user => s

      switch s
         case "RESERVAR_CABINA,i"
            ControlCabinas.entraUsuario(c)
            timer_init()                - Función de medición de tiempo
            ch_user <= "CONCEDIDA,c"
         case "LIBERAR_CABINA,c"
            ControlCabinas.saleUsuario(c)
            timer := timer + timer_end()
         case "FIN"
            ch_user <= "Uso total de la cabina: timer"
            fin := true
   end
   Close()

end

Process servir_limpiador::
   boolean fin := false
   integer c, i, timer := 0
   character array s

   while not fin
      ch_user => s

      switch s
         case "LIMPIEZA_ENTRAR"
            ControlCabinas.entraLimpieza()
            timer_init()            - Función de medición de tiempo
            ch_user <= "OK_LIMPIEZA"
         case "LIMPIEZA_SALIR"
            ControlCabinas.saleLimpieza(
            timer := timer + timer_end()
         case "FIN"
            ch_user <= "Tiempo total limpiando: timer"
            fin := true
   end

   Close()
end
```

```
--- CLASE MONITOR PARA EL CONTROL DE CABINAS HEREDADA DE LA PRÁCTICA 4 ---

Monitor ControlCabinas

   integer dentro := 0
   boolean array cabina[N_CAB]
   boolean limpiando

   condition vacio, acceder

   operation entraUsuario(REF integer cab)
      while not (not limpiando and dentro < N_CAB)
         waitC(acceder)
      end

      boolean encontrado := false
      integer k := 0

      while not encontrado
         if cabina[k] := false
            cabina[k] := true
            cab := k
            dentro := dentro + 1
            encontrado := true
         else
            k := k + 1
         end
      end
   end

   operation saleUsuario(REF cab)
      cabina[cab] := false
      dentro := dentro - 1

      if dentro = 0
         signalC(vacio)
      end

      if not limpiando
         signalC(acceder)
      end
   end

   operation entraLimpieza()
      limpiando := true

      while not dentro = 0
         waitC(vacio)
      end
   end

   operation saleLimpieza()
      limpiando := false
      signalC_all(acceder)
   end
end
```