Sistemas Operativos – 4o Semestre – GII & GMI

Cuarto Parcial. Comunicación y Sincronización. 1 de Junio de 2015

Un equipo de diseñadores software tiene como tarea el desarrollo de un ficticio sistema informático de la Junta electoral central para la noche de las elecciones. Este sistema está compuesto por múltiples actividades concurrentes, que deben coordinarse en el acceso a varias fuentes de información.

En cada colegio electoral se lleva a cabo el recuento de los votos. El equipo de diseñadores decide que éste sea el primer subsistema (Subsistema A).

Cada colegio electoral envía al coordinador de distrito la información sobre los votos utilizando acceso web seguro (*https*). Dicho coordinador se encarga de la integración de toda la información. El coordinador de distrito será el segundo subsistema (Subsistema B).

Finalmente, cada distrito envía a la Junta electoral central la información agregada y a partir de esta información, la Junta electoral lleva a cabo el cómputo global de los votos, almacenándolo en una Base de Datos. La Junta electoral central constituye el tercer subsistema (Subsistema C).

Centrándonos en el procesamiento del Subsistema A y teniendo en cuenta que el servicio se centraliza en una única máquina, el equipo de diseñadores baraja varias opciones:

A1. Diseñar una solución basada en un único proceso formado por diferentes threads, cada uno de los cuáles procesará una parte de los votos.

A2. Diseñar una solución basada en varios procesos pesados emparentados, cada uno de los cuáles procesará una parte de los votos.

A3. Diseñar una solución basada en varios procesos locales independientes, cada uno de los cuáles procesará una parte de los votos.

1. Para el diseño A1, indicar cuál de los siguientes mecanismos de comunicación es el más apropiado:
    a. Variables globales + mutex + condiciones
    b. Variables globales + semáforos
    c. Tuberías sin nombre (PIPE)
    d. Tuberías con nombre (FIFO)

Solución: Variables globales + mutex + condiciones

Dado que se trata de un único proceso formado por diferentes threads, éstos comparten el acceso a las variables globales, que deberá ser controlado mediante el uso de un mecanismo de sincronización adecuado. Los semáforos no son apropiados. Las mutex y condiciones se adaptan perfectamente a este escenario. Las tuberías no son necesarias.

2. Para el diseño A2, indicar cuál de los siguientes mecanismos de comunicación es el más apropiado:
    a. Variables globales + mutex + condiciones
    b. Memoria proyectada compartida sin semáforos
    c. Tuberías sin nombre (PIPE)
    d. Sockets de tipo datagrama con dominio AF_INET

Solución: Tuberías sin nombre (PIPE)

Dado que se trata de varios procesos pesados emparentados, las variables globales no son compartidas. Tampoco es factible utilizar memoria proyectada compartida sin utilizar ningún mecanismo de sincronización. Los sockets con dominio AF_INET permiten comunicar procesos en máquinas diferentes, por lo que no son requeridos. Por último, las tuberías sin nombre se adaptan perfectamente.

3. Para el diseño A3, cuál de los siguientes mecanismos de comunicación **no** es factible:
    a. Tuberías sin nombre (PIPE)
    b. Fichero proyectado en memoria + semáforos con nombre
    c. Fichero + cerrojos sobre fichero
    d. Sistema de Gestor de BBDD con transacciones

Solución: Tuberías sin nombre (PIPE)

Las tuberías sin nombre no pueden ser utilizadas para comunicar procesos independientes. El resto de mecanismos son factibles en el escenario descrito.

A continuación, el equipo se centra en el diseño de la comunicación entre el Subsistema A y el Subsistema B.

4. Si el número de colegios electorales por distrito no es muy elevado, ¿Qué arquitectura es la más adecuada para soportar dicha comunicación?
    a. Cliente/servidor con servidor con funcionamiento serie (un solo proceso atiende de forma secuencial a los clientes)
    b. Cliente/servidor con servidores dedicados (un proceso atiende a cada cliente)
    c. Arquitectura P2P (Peer-to-peer), con red estructurada
    d. Arquitectura P2P (Peer-to-peer), con red no estructurada

Solución: Cliente/servidor con servidores dedicados (un proceso atiende a cada cliente)

Dado que el número de colegios electorales no es muy elevado, el servidor de la solución Cliente/Servidor no sería un cuello de botella. Por tanto, la solución P2P la podemos descartar frente a C/S, pues complicaría demasiado el diseño de la solución. La solución C/S con funcionamiento serie causaría una contención innecesaria en los clientes. Por tanto, la solución C/S con servidores dedicados es la más adecuada.

5. ¿Cuál de los siguientes mecanismos de comunicación es adecuado para este escenario?
    a. Sockets de tipo stream con dominio AF_INET
    b. Sockets de tipo datagrama con dominio AF_INET
    c. Tuberías con nombre (FIFO)
    d. Sockets de tipo stream con dominio AF_UNIX

Solución: Sockets de tipo stream con dominio AF_INET

Las tuberías con nombre no permiten la comunicación entre procesos en diferentes máquinas. Tampoco lo permiten los sockets que utilizan dominio AF_UNIX. Por último, este escenario requiere fiabilidad, por lo que se requiere un socket de tipo stream con dominio AF_INET.

Para el desarrollo del Subsistema B, los diseñadores optan por una solución basada en procesos pesados. Uno de los diseñadores nos muestra la siguiente función utilizada en el subsistema B, que contiene errores:

```
void actualizar_votos (int votos, int id_colegio)
{

      struct flock fl;
      int fd;
      int val;
      int start = id_colegio*sizeof(int);

      fd = open("BD", O_RDWR | O_CREAT | O_TRUNC, 0644);
      fl.l_whence = SEEK_SET;
      fl.l_start = start;
      fl.l_len = sizeof(int);
      fl l_pid = getpid();
      fl.l_type = F_RDLCK;
      fcntl(fd, F_SETLK, &fl);
      lseek(fd, start, SEEK SET);
      read(fd, &val, sizeof(int));
      val += votos
      lseek(fd, start, SEEK SET);
      write(fd, &val, sizeof(int));
      fl.l_type = F_UNLCK;
      fcntl(fd, F SETLK, &fl);
      close(fd);
}
```

6. Indicar cuál de las siguientes acciones **no** es correcta de cara a resolver los problemas del código:
    a. El flag O_TRUNC debería eliminarse del código
    b. El cerrojo debería ser exclusivo
    c. La primera llamada a `fcntl` debería ser síncrona
    d. La segunda llamada a `lseek` debería eliminarse del código

Solución: La segunda llamada a lseek debería eliminarse del código

Si se elimina la segunda llamada a lseek, no actualizaremos el dato en la zona del fichero donde se guarda la información del colegio correspondiente, dado que la llamada read avanza el puntero de posición los bytes que corresponden al tamaño de un entero. Por otro lado, el flag O_TRUNC debería eliminarse, para evitar que el fichero se trunque cuando se lleva a cabo la apertura. El cerrojo tiene que ser exclusivo pues vamos a modificar la zona del fichero y la primera llamada a fcntl debe ser síncrona, para evitar que un proceso pueda progresar si no consigue el cerrojo exclusivo.

Respecto a la comunicación entre el subsistema B y el subsistema C, los diseñadores se plantean la siguiente cuestión:

7. Para que los subsistemas B y C puedan intercambiar mensajes de texto, ¿qué se requiere en ambos extremos de la comunicación?
    a. Convertir cada carácter del texto a enviar de formato host a formato de red en el subsistema B y de formato red a formato host en el subsistema C
    b. Convertir cada carácter del texto a enviar de formato de red a formato de host en el subsistema B y de formato de host a formato red en el subsistema C
    c. Transformar cada carácter del texto a enviar en coma flotante y utilizar un formato de conversión no estándar antes de que el subsistema B lo envíe al subsistema C.
    d. No llevar a cabo ninguna operación de conversión sobre los caracteres enviados.

Solución: No llevar a cabo ninguna operación de conversión sobre los caracteres enviados

Dado que la representación de un carácter no depende de la arquitectura de los subsistemas B y C, no es necesario llevar a cabo ninguna operación de conversión.

Los diseñadores deciden implementar el subsistema C como un servidor concurrente orientado a conexión, que espera las peticiones de los posibles clientes (procesos de diferentes subsistemas B). Uno de los diseñadores nos muestra el siguiente trozo de código del servidor, que contiene errores:

```
int sd, cd, size;
struct sockaddr_in s_ain, c_ain;
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
bzero((char *)&s_ain, sizeof(s_ain));
s_ain.sin_family = AF_INET;
s_ain.sin_addr.s_addr = INADDR_ANY;
s_ain.sin_port = 150;
bind(sd, (struct sockaddr *)&s_ain, sizeof(s_ain));
listen(sd, 100);

….
```

8. Indicar cuál de las siguientes respuestas corresponde a todos los errores del código:
    a. El número de puerto debería convertirse de formato host a formato red.
    b. Al campo `s_ain.sin_addr.s_addr` no se le puede asignar el valor `INADDR_ANY` , ya que debería tener asignado una dirección IP.
    c. El número de puerto debería convertirse de formato red a formato host.
    d. El número de puerto debería convertirse de formato host a formato red y al campo `s_ain.sin_addr.s_addr` no se le puede asignar el valor `INADDR_ANY` , ya que debería tener asignado una dirección IP.

Solución: El número de puerto debería convertirse de formato host a formato red

El único error viene dado por la no conversión de formato host a formato red (y no a la inversa) del número de puerto, que es un número entero. Al campo s_ain.sin_addr.s_addr se le puede asignar el valor INADDR_ANY, cuando queremos que el servidor acepte conexiones de cualquier cliente.