Universidad
Rey Juan Carlos

Práctica 2: Relojes de Lamport

Sistema distribuidos y concurrentes

En esta segunda práctica pondremos en práctica los conocimientos adquiridos en clase de teoría sobre paso de mensajes y relojes de Lamport**.**
Desarrollaremos un mecanismo de sincronización basado en relojes de Lamport para sincronizar el apagado de las máquinas.

Paso de Mensajes.
Para esta práctica todos los mensajes intercambiados mediante sockets entre los procesos utilizarán la siguiente estructura:

```
enum operations {
      READY_TO_SHUTDOWN = 0,
      SHUTDOWN_NOW,
      SHUTDOWN_ACK
};

struct message {
   char          origin[20];
   enum operations    action;
   unsigned int  clock_lamport;
};
```

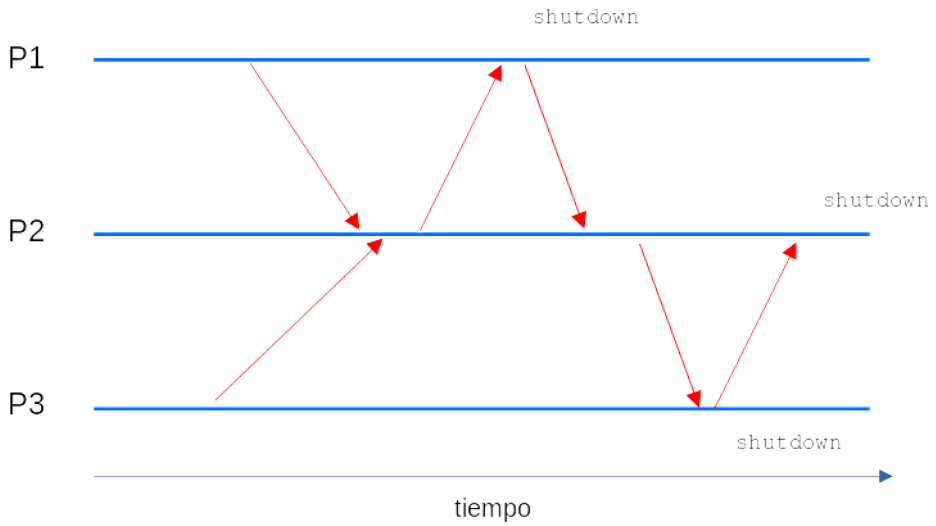
El campo ***origin*** contendrá el nombre del proceso que envía el mensaje.
El campo ***action*** podrá contener valores del enumerado operations:
· READY_TO_SHUTDOWN: la máquina notifica que está lista para apagarse.
· SHUTDOWN_NOW: Al recibir este mensaje la máquina sabe que debe apagarse.
· SHUTDOWN_ACK: La máquina manda este mensaje antes justo de realizar el shutdown.
El campo ***clock_lamport*** es utilizado para enviar el contador de lamport.

Relojes de Lamport
La sincronización entre los procesos debe realizarse mediante relojes de Lamport. Toda la funcionalidad de los relojes debe incluirse dentro del stub. Los procesos deben ejecutar eventos exclusivamente teniendo en cuenta el valor de su reloj lamport.

Implementación

Desarrolla 3 procesos independientes (P1, P2, P3) que hacen uso del stub con las características mencionadas anteriormente. El funcionamiento del sistema es el siguiente:

1. P1 y P3 notifican a P2 que están listos para apagarse (READY_TO_SHUTDOWN)
2. P2 recibe los mensajes y envía a P1 la orden de apagarse. (SHUTDOWN_NOW)
3. P1 recibe el mensaje y envía a P2 el ACK de apagado. (SHUTDOWN_ACK)
4. P2 recibe el mensaje y envía a P3 la orden de apagarse. (SHUTDOWN_NOW)
5. P3 recibe el mensaje y envía a P2 el ACK de apagado. (SHUTDOWN_ACK)
6. P2 recibe el mensaje.



Cada vez que un evento (envío/recepción) ocurra en un proceso, debe imprimir la siguiente traza por la salida estándar

PX, contador_lamport, SEND, operations
PX, contador_lamport, RECV (PY), operations

Como ejemplo en el caso inicial del proceso P1, debería imprimir

```
P1, 1, SEND, READY_TO_SHUTDOWN
```

Y el proceso P2, debería imprimir

```
P2, 2, RECV (P1), READY_TO_SHUTDOWN
```

Además el proceso P2 debe escribir siempre lo siguiente cuando sepa seguro que los otros 2 procesos han realizado el shutdown (este paso se debe realizar en el código de p2 y no en el stub).

```
Los clientes fueron correctamente apagados en t(lamport) = XX
```

Encapsulación en el stub
Toda la funcionalidad de comunicación a través de sockets y sincronización de relojes lamport debe quedar abstraída y encapsulada en el *stub.* Puedes implementar toda la funcionalidad en los ficheros stub.c / stub.h.

La librería stub debe implementar obligatoriamente el acceso al reloj lamport. El código de P1, P2 y P3 no puede acceder a la variable del reloj directamente, ni modificarlo. Para el resto de funcionalidad puedes definir las funciones que considere necesarias.

```
// Obtiene el valor del reloj de lamport.
// Utilízalo cada vez que necesites consultar el tiempo.
// Esta función NO puede realizar ningún tiempo de comunicación (sockets)
int get_clock_lamport();
```

El funcionamiento de las aplicaciones P1,P2,P3 debe quedar claro en el código de dichas aplicaciones, esto quiere decir, que el código debe mostrar claramente que espera a que el contador tenga un valor para ejecutar las funciones. Por ejemplo:

```
while (get_clock_lamport() < VALUE) {
      continue;
      …
}

funcion_a_ejecutar();
```

Consideraciones
- La sincronización debe cumplir el diagrama temporal de la página anterior excepto para los eventos concurrentes (si existieran)
- Evita espera activa, y que tus programas no consuman el 100% CPU (usa htop).
- Os recomendamos que lo primero que hagan los procesos es abrir los canales de comunicación con los procesos que vayan a intercambiar los mensajes.
- Toda la lógica de comunicación y relojes de lamport deben quedar encapsuladas dentro del stub.
- Los recv() en cualquiera de los procesos, se deben ejecutar siempre en un thread distinto al hilo principal del proceso (asume que no hay problemas de concurrencia)
- El envío y recepción de mensajes debe ser controlado y sincronizado dependiendo del contador del reloj lógico de lamport.
- Puedes realizar espera activa en tu programa principal para detectar cuando cambia el contador. Pero esa espera activa no puede dejar bloqueado las comunicaciones de ese proceso.
- Puedes utilizar pthreads y sockets con la configuración que creas más conveniente (bloqueante, no-bloqueante).
- Todos los procesos deben aceptar IP y PUERTO por argumento.
    - ./P1 127.0.0.1 8000

- Sigue el estilo de código definido para esta asignatura.
- Utiliza el sistema de compilación Makefile
- La práctica debe compilar correctamente y ejecutar en los ordenadores del laboratorio.
- Ejecuta cada proceso en una máquina física distinta.