SECRETARIA DE EDUCACIÓN PÚBLICA

DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA

INSTITUTO TECNOLÓGICO DE VERACRUZ

SIMULACION 1

UNIDAD 3 HERRAMIENTA DE PROGRAMACION DE MODELOS DE SIMULACION NUMERICA

3.9 MECANISMOS DE SINCRONIZACIÓN 2: TIMERS Y CRITICAL SECTIONS.

CATEDRÁTICO:
DR. JOSE ANTONIO GARRIDO NATAREN

H.VERACRUZ, VER. ENERO-JULIO DEL 2013

Timer

Como es lógico pensar, este es un punto muy importante en sistemas de tiempo real. Es fundamental que el sistema provea una base de tiempo adecuada para la sincronización interna de los procesos presentes. Debe aclararse aquí la diferencia entre *reloj* y *timer*.

- Reloj: es simplemente un contador que provee una base de tiempo
- Timer: es un contador que llegado a cierto estado, es capaz de notificar que esto ha sucedido. Para la implementación de un timer es necesario un reloj.

Un sistema de tiempo real debe ser capaz de medir internamente el tiempo con la resolución y precisión adecuada para el caso de aplicación. Es deseable también que el sistema sea capaz de reconocer cuándo un timer ha expirado varias veces (*timer overrun*) y que sea también capaz de generar una señal standard ante la expiración normal de un timer.

Un aspecto importante es la capacidad del sistema de sincronizar su reloj interno con el entorno, o sea con el tiempo real. De la misma manera, en un sistema distribuido, es necesaria una adecuada sincronización entre los diferentes nodos. Los principales problemas que se presentan son los drifts debido a la temperatura e incluso la falla de los relojes, problemas que deben ser subsanados para evitar la falla total del sistema. Uno de los métodos utilizados es el empleo de una base de tiempo global, a la cual tienen acceso todos los nodos.

Otro método es la implementación de un promedio entre los relojes presentes en el sistema. Para ello se utilizan

algoritmos que prevén los retardos debido a la comunicación y posibles relojes *maliciosos*. Uno de los algoritmos más utilizados es *Fault tolerant average algorithm* de Ochsenreiter y Kopetz, de 1987 [Buttazzo, 1997], que se encuentra disponible como un elemento de hardware que puede incluirse en el sistema.

Usos del Timer

- Proveen la hora, sirven de cronometro o de alarma.
- Se puede programa para que interrumpa cada 10 mseg (periodicamante).
- Al interrumpir se llama a una rutina de servicio (manejador de interrupcion).
- Se puede programa para atender eventos asincronos, que ocurren en cualquier momento y hay que darles atencion. P.ej. el teclado o el puerto serie.

Secciones Criticas

En programación concurrente, se define como a la porción de código de un programa de computador el cual accede a un recurso compartido (estructura de datos ó dispositivo) que no debe de ser accedido por mas de un hilo en ejecución (thread). La sección crítica por lo general termina en un tiempo determinado y el hilo, proceso ó tarea solo tendrá que esperar un período determinado de tiempo para entrar. Se necesita de un mecanismo de sincronización en la entrada y salida de la sección crítica para asegurar la utilización exclusiva del recurso, por ejemplo un semáforo.

Usos de la Sección Critica

La sección crítica se utiliza por lo general cuando un programa multihilo actualiza múltiples variables sin un hilo de

ejecución separado que lleve los cambios conflictivos a esos datos. Una situación similar, la sección crítica puede ser utilizada para asegurarse de que un recurso compartido, por ejemplo, una impresora, puede ser accedida por un solo proceso a la vez.

La manera en como se implementan las secciones puede variar dependiendo de los diversos sistemas operativos.
Sólo un proceso puede estar en una sección crítica a la vez.

Modelo de Sección Critica

El modelo de sección crítica que vamos a utilizar sigue el siguiente protocolo genérico:

Entrar_SC(esta_SC) /* Solicitud de ejecutar esta_SC */
/* código de esta_SC */
Dejar_SC(esta_SC) /* Otro proceso puede ejecutar esta_SC */
Es decir, cuando un proceso quiere entrar a la sección crítica:
(1) ejecuta Entrar_SC(), y si la sección crítica está ocupada el proceso espera;
(2) ejecuta la sección crítica;
(3) ejecuta Dejar_SC(), permitiendo que entre uno de los procesos en espera.

Propiedades del acceso Exclusivo a Secciones Criticas

Como criterios de validez de un mecanismo de sincronización nos referiremos al cumplimiento de las siguientes condiciones enunciadas por Dijkstra para el acceso exclusivo a una sección crítica.

Exclusión mutua. No puede haber más de un proceso simultáneamente en la SC.
No interbloqueo. Ningún proceso fuera de la SC puede impedir que otro entre a la SC.
No inanición. Un proceso no puede esperar por tiempo indefinido para entrar a la SC.
Independencia del hardware. No se pueden hacer suposiciones acerca del número de procesadores o de la velocidad relativa de los procesos.

Inhibición de interrupciones

Las secciones críticas protegidas por interrupciones se especifican de la siguiente forma:

```
s= inhibir() /* Inhibe interrupciones */
/* Sección crítica */
Desinhibir(s) /* Restaura estado anterior de interrupciones */
```

Este mecanismo lleva asociadas importantes restricciones. No cumple la condición de independencia del hardware, ya que en un multiprocesador sólo se inhiben las interrupciones en el procesador que ejecuta la inhibición, restringiéndose por tanto a las implementaciones monoprocesador. Por otra parte, en monoprocesadores, un proceso que utilizase la inhibición de interrupciones como forma de acceso exclusivo a secciones críticas de duración arbitrariamente larga impediría a los demás procesos ejecutar código alguno, aún ajeno a la sección crítica.

Cerrojos de espera activa

Un mecanismo más general que la inhibición de interrupciones es la utilización de una variable cerrojo para

proteger la sección crítica. El proceso que quiere entrar a la sección crítica consulta el cerrojo. Si está libre (cerrojo==0), el proceso lo echa (cerrojo=1) y entra a la sección crítica. Si está echado, ejecuta una espera activa consultando su valor hasta que esté libre. Cuando un proceso deja la sección crítica, libera el cerrojo (cerrojo=0). Este esquema tan sencillo presenta importantes problemas de implementación. Como se puede comprobar, la operación de consulta y modificación del cerrojo constituye a su vez una sección crítica que hay que resolver previamente; si no dos procesos podrían leer simultáneamente un valor cero y ambos entrar a la sección crítica, violando la condición exclusión mutua.