Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 1

Dado un SO en el que la interrupción de reloj se trata con la función:

```
Manejador_interrupción_reloj ():
        Ticks = Ticks +1;
}
```

donde Ticks es una variable global del sistema que se inicializa a 0.

La función planificador es la encargada de, utilizando las estructuras de datos con la información sobre los procesos, devolver el proceso que debe ejecutar a continuación.

Se dispone de una constante:
TICKS_POR_RODAJA declarada a nivel global en el SO que indica cuantos TICKS debe consumir un proceso antes de que se evalúe su sustitución por otro.

Indicar que habría que hacer para que la rodaja de tiempo asignada a cada proceso fuera el doble de la actual sin tocar el planificador modificando la función
Manejador_interrupción_reloj ():

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

Hay distintas estrategias, como por ejemplo:

- Modificar la constante que supone la rodaja.
- Modificar la tarea pendiente que se pone en la interrupción software para que cuente el doble de ticks por rodaja.

No obstante, nos piden realizar la modificación en la interrupción hardware. El código de la nueva interrupción hardware buscaría contar cada 2 ticks para la rodaja:

Manejador_interrupción_reloj ():

- static int swcontar=0;
- Ticks = Ticks +1;
- If (0 == swcontar) {
    - Ticks_rodaja = Ticks_rodaja + 1;
    - swcontar = 1 ;
- } else {
    - swcontar = 0 ;
- }

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 2

Se dispone de un sistema hardware que incluye un dispositivo de reloj, el cual genera una interrupción con cada *tick* del reloj.

Un sistema operativo multiproceso tiene prevista (a falta de la implementación) la llamada al sistema:

```
struct Fecha * ObtenerFecha ();
```

Que debe devolver la fecha y hora actuales. Para ello, se dispone además del código de la función:

```
struct Fecha * Convertir_Ticks_en_Fechas (int ticks);
```

Dicha función permite obtener la fecha y hora actuales a partir del número de *ticks* que han transcurrido.

Se pide:

Implementar en pseudocódigo la funcionalidad necesaria del kernel del sistema operativo para gestionar el reloj y ofrecer a los usuarios la funcionalidad de obtener la fecha y hora actuales cuando lo soliciten. Poner especial interés en indicar cuales son las estructuras de datos requeridas o modificadas, la interfaz de las funciones implementadas y los eventos utilizados.

Grupo: …… NIA: …… Nombre y apellidos: ……

SOLUCIÓN

a) Lo primero es enumerar los eventos involucrados:
- Interrupción del reloj
- Llamada al sistema ObtenerFecha

Después las funciones a ejecutar en cada evento.

Manejador_interrupción_reloj ():

- Ticks = Ticks +1;

Struct Fecha * LLamada_al_Sistema_ObtenerFecha ():

- Devolver (Convertir_Ticks_en_Fechas (Ticks));

Las Variables globales necesarias son:

- Ticks: Números de ticks de reloj desde el arranque de la máquina.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 3

Se dispone de un sistema operativo multiproceso dispone de temporizadores, de forma que es posible indicar para un tick de reloj específico en el futuro (Ti.ticks), que un proceso (Ti.proc) cambie de estado a listo para ejecutar y se inserte en la lista de listos para ejecutar.

Este sistema operativo dispone de una implementación de la rutina de tratamiento de interrupción de reloj que es en pseudocódigo:

- Ticks = Ticks +1;
- Recorrer la lista de temporizadores y por cada temporizador Ti
    - Si (Ti.ticks == Ticks)
        - Cambiar el estado de Ti.proc a listo.
        - Incluirlo en la lista de procesos listos para ejecutar.

La lista de procesos listos para ejecutar es una estructura de datos del sistema operativo a la que tienen acceso ciertas llamadas al sistema e interrupciones hardware.

Se pide:

a) Indicar los posibles problemas que puede presentar el pseudo-código anterior.
b) Si en el apartado anterior aparece algún problema, indicar posibles soluciones asociadas.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Los posibles problemas son:
- Las rutinas de tratamiento de interrupción hardware deben ser lo más rápidas posibles, si la lista es larga el procesamiento por tick de reloj puede hacer que se pierdan eventos.
- Problemas de concurrencia puesto que la interrupción de reloj puede interrumpir el tratamiento de otro evento que esté usando las estructuras de datos compartidas y dejarlas de forma incoherente.

b) La solución dependerá del tipo de sistema que se disponga:

- Si usamos un kernel no expulsivo las llamadas al sistema y las interrupciones software nunca se interferirán entre sí. Pero las interrupciones hardware y las excepciones siempre interferirán entre sí y con las llamadas al sistema e interrupciones software.

  En la solución anterior podemos tener una interrupción hardware y una llamada al sistema que accedan a la lista de procesos listos. Como la interrupción hardware siempre interferirá a la llamada al sistema tenemos problemas de concurrencia.

  La solución en un kernel no expulsivo es crear una interrupción software asociada a la interrupción hardware donde se realice las tareas que manipulan estructuras de datos compartidas:

  Pseudocódigo Manejador_interrucion_reloj()
  - Ticks=Ticks +1
  - Insertar_Interrupcion_Software(Despertar_entick)
  - Generar_Interrupcion_Software()

  Pseudocódigo Despertar_entick()
  - Recorrer la lista de temporizadores y por cada temporizador Ti
    - Si (Ti.ticks == Ticks)
      - Cambiar el estado de Ti.proc a listo.
      - Incluirlo en la lista de procesos listos para ejecutar.

- La solución en un kernel expulsivo implica que, además de lo anterior mínimo hay que bloquear las interrupciones (como mecanismo de control de concurrencia) en las interrupciones software y las llamadas a los sistemas al acceder a la lista de listos.

- Si el kernel es SMP y hay varias CPU entonces el bloqueo de interrupciones es insuficiente (son locales a cada CPU) y mínimo hay que usar spinlocks.