TRABAJO PRÁCTICO N° 2

Temas: Interrupciones. Timers.

Objetivos:

- *Técnica de programación por interrupciones.*
- *Programación de un productor – consumidor (buffer limitado).*
- *Programación de timers.*

Material de soporte:

- *Para resolver el práctico se propone consultar el **"Material para clases prácticas"**, como así también la sección correspondiente a este práctico, disponible en el Aula Virtual.*
- *Para el primer ejercicio (Productor/Consumidor) utilizar el proyecto base **proyectobase2a.X** disponible en el Aula Virtual.*
- *Para el segundo ejercicio (Multitask) utilizar el proyecto base **proyectobase2b.X** disponible en el Aula Virtual.*

Ejercicio 1: Productor/Consumidor

Instrumentar una rutina de interrupción para almacenar en una tabla de memoria caracteres que recibe por el puerto A (PORTA) en orden de aparición. La estructura del programa debe corresponder al paradigma Productor/Consumidor conforme a lo siguiente:

- **Productor**: el cuerpo de la rutina de atención para la interrupción debe resolver el ingreso de caracteres en un arreglo. Este arreglo (buffer) se utilizará de manera circular (se volverá al inicio una vez que se lo termine de recorrer). El ingreso de un caracter se detecta por un flanco descendente en una de las Interrupciones Externas (INTx), y se extraerán del puerto A (PORTA)
- **Consumidor**: el consumidor leerá los caracteres nuevos (los que no haya leído ya) del buffer y los depositará en otra tabla.

Configurar un timer para que cada cierto tiempo el consumidor lea los caracteres pendientes; si no hay ningún caracter nuevo para leer, el tiempo de espera del consumidor se duplicará. Tener en cuenta que puede haber uno o varios caracteres nuevos para leer.

Por ejemplo, al inicio del programa, el timer se configurará para que espere 100us (microsegundos). Si hay caracteres nuevos para leer, serán leídos y el programa volverá a esperar 100us. Si no hay caracter nuevo, el timer ahora esperará 200us. Si sigue sin haber caracteres nuevos, serán 400us. Este valor será un tope. Si no llegan caracteres, se seguirá esperando 400us. Cuando lleguen caracteres, el timer volverá a esperar 100us.

Ejercicio 2: Multitask

Escribir un programa que simule la ejecución de tres procesos independientes (procesoA, procesoB y procesoC) y concurrentes. El programa debe constar de los tres procesos, una rutina de atención para la interrupción de un reloj y un planificador que alterna el uso de la CPU entre los tres procesos.

Para resolver este ejercicio tomen como base el **proyectobase2b.X** entregado por la cátedra en el Aula Virtual. En este proyecto encontrarán un esqueleto de solución. El código de los tres procesos ya está presente y se recomienda NO modificarlos. El código de la función boot() en kernel.c también

está completo. Si ejecutan el proyecto tal cual está, sólo se ejecutará el procesoA. Lo que deben lograr es que se ejecuten en forma concurrente los tres procesos. Para esto deberán completar las funciones incompletas en kernel.c, además de crear las estructuras de datos necesarias.

A los procesos se les asignará idéntico quantum (tiempo de ejecución). Cuando la rutina del reloj detecta que expiró el quantum del proceso actual, invoca al planificador. El planificador debe resguardar los valores de los registros del proceso actual (estado del proceso), recuperar el estado del proceso siguiente (previamente resguardado) y restablecer el quantum.

El planificador debe constar de una estructura de datos apropiada para contener el estado de los procesos. Para implementar el reloj se utilizará un Timer. Una vez que se configura el timer, este generará interrupciones en forma regular. A estas les denominamos “interrupciones de reloj en este ejercicio. El quantum será igual a 2 (dos) interrupciones de reloj. Esto quiere decir que cada dos interrupciones del timer, se considera que se acabó el quantum del proceso actual y por lo tanto hay que cambiar la ejecución al proceso siguiente.

Nota: Para manipular la dirección de ingreso a una rutina se puede asignar directamente el nombre de una función a una variable.
Ejemplo:

```
int direccion;
…
direccion = main;
```