



Adquisición y Presentación de Señales Biomédicas Utilizando Objetos de Sincronización entre Procesos Bajo Windows 98

M. Hernández S¹., C. Ramírez R.²

RESUMEN

En este artículo, se describe el desarrollo de un programa basado en la técnica de programación multitareas, utilizando objetos de sincronización entre procesos bajo el sistema operativo Windows98. Dicho programa, logra mantener una velocidad de muestreo óptima en un sistema de adquisición de señales, correspondiente en esta aplicación a información electrofisiológica. Sin embargo, la finalidad de utilizar objetos de sincronización entre procesos es, compartir en forma simultánea los datos que se van adquiriendo en el programa con una o más aplicaciones que se estén ejecutando al mismo tiempo, y que pueden ser programadas en otros lenguajes. Para ello se realiza un programa que consta de dos tareas fundamentales, cada una representada por una hebra de programación; la primera para la adquisición y almacenamiento en memoria principal de las muestras que provienen de módulos electrofisiológicos, y la segunda para la presentación de las señales obtenidas en la pantalla de la computadora. Para la sincronización de las hebras, se utiliza el algoritmo del productor – consumidor empleando semáforos, donde el almacén de las muestras adquiridas se implementa de manera tal que pueda ser compartido por diferentes procesos. Por último, para comprobar la efectividad del algoritmo desarrollado, se programó una aplicación adicional que utiliza los datos que se van guardando en el almacén. Todo el software ha sido desarrollado combinando la herramienta de programación Borland Builder C++ 5.0 con la API WIN32 de Windows98, la cual permite la implantación de aplicaciones a nivel del sistema operativo bajo un ambiente visual y orientado a objetos.

Palabras clave: *programación multitareas, adquisición de datos, muestreo, objetos, semáforos, procesos, hebras.*

Using Interprocess Synchronization Objects for Biomedical Signal Acquisition and Visualization.

ABSTRACT

This article describes the development of a program that follows a multithread approach using process synchronization objects under Windows98 operating system. This program is capable of providing a steady sampling frequency for acquisition of electrophysiology data. The aim of using synchronization objects is to share simultaneously the acquired data with other applications running at the same time and sometimes implemented in different languages. For this purpose, the program is divided into two stages, each one of them implemented by a thread; the first one for acquisition and storage of the data samples in the main memory, and the second one for painting the signals in the computer screen. The producer – consumer algorithm using semaphores is applied for thread synchronization, where the buffer of the acquired samples is implemented in such a way that it can be used by different process. Lastly, in order to corroborate the effectiveness of the proposed algorithm, an additional application was programmed that uses the data stored in the buffer. The algorithm has been implemented by combining the programming tool Builder C++ 5.0 and the API WIN32 of Window98, which allows the development of applications at the operating system level using a visual object oriented environment

Keywords: *Multitask programming, data acquisition, objects, process, semaphores*

¹ S. Av. Universidad, Sector Paramillo, San Cristóbal, Estado Táchira, Venezuela
E-mail: mahs@unet.edu.ve; Fax: (58) (76)-532454; Tel: (58) (76)-532454

² S. Av. Universidad, Sector Paramillo, San Cristóbal, Estado Táchira, Venezuela; E-mail: cram@unet.edu.ve; Fax: (58) (76)-532454; Tel: (58) (76)-532454



1. INTRODUCCIÓN

Es relevante destacar que la evolución de los sistemas operativos ha dado un gran aporte en las áreas de computación e instrumentación biomédica, específicamente en el diseño de monitores de parámetros fisiológicos. Gracias a los sistemas operativos de multitareas con interfaces gráficas amigables, se han logrado implantar algoritmos que aprovechan de una forma eficaz todos los recursos de la computadora [2]. Por una parte este hecho presenta una clara ventaja: la programación de la interfaz entre el usuario y el sistema es sencilla por ser orientada a objetos. Cada objeto se basa en propiedades y métodos fáciles de manejar. Por otra parte, el control de varios procesos que accedan recursos compartidos, sugieren técnicas de programación de mayor dificultad [2].

2. PROCESOS Y HEBRAS

Un sistema operativo actual como Windows 98 esta basado en multitareas de procesos y hebras [3]. Un proceso es un programa completo cuya imagen esta en memoria, el cual se puede encontrar en varios estados: Ejecución, Bloqueado, Suspendido y Preparado [4]. A su vez, las hebras presentan las mismas características que los procesos, pero a diferencia de estos, cada una esta constituida por una unidad de código remitente que se puede ejecutar de forma independiente, es decir, una hebra puede ser una función dentro de un programa que se ejecuta sin depender de una lógica secuencial. Esto es, el planificador de procesos del sistema decide cuando se debe ejecutar dicha hebra y cuando debe dejar de hacerlo para dar paso a la ejecución de otra hebra u otro proceso. De acuerdo a esto, un programa se puede dividir en múltiples hebras para aumentar su efectividad [2].

Para el desarrollo de un software de adquisición y presentación de datos, el hecho de implementar hebras constituye una solución satisfactoria, ya que los algoritmos secuenciales emplean una gran cantidad del tiempo del procesador en presentar los datos en pantalla. Durante ese tiempo, es posible que se pierda una cantidad significativa de muestras. Esto depende de la frecuencia de las señales que se adquieren, el número de las mismas y la complejidad del hardware asociado a la adquisición [2].

Por otra parte, es posible que las señales que se estén obteniendo a través un proceso de adquisición, necesiten ser compartidas con otros programas adicionales que realicen cálculos pertinentes a estudios médicos, tales como la clasificación de arritmias cardíacas u otros más sencillos, como el simple calculo de la frecuencia a la que late el corazón. Para ello, se toman en cuenta dos aspectos: en primer lugar, se debe utilizar un espacio de memoria donde se almacenen los datos que se van adquiriendo, pero con la particularidad de que dicho espacio pueda ser accedido por aplicaciones diferentes. Los programas realizados bajo Windows 98, que están basadas en WIN32, solo pueden compartir memoria a través de la técnica de creación de memoria compartida etiquetada utilizando archivos mapeados a memoria [1].

En segundo lugar, solo un proceso o hebra puede utilizar la memoria compartida a la vez, por lo tanto el acceso a dicha región debe ser sincronizado entre los procesos. Windows 98 proporciona cuatro objetos de sincronización entre los cuales se destacan:

- Semáforos Clásicos.
- Semáforos Binarios o de Exclusión Mutua.

Los Semáforos Clásicos se utilizan para permitir que un número limitado de procesos o hebras tengan acceso a un recurso. Dichos semáforos son variables enteras compartidas entre procesos y hebras, las cuales se modifican a través de operaciones atómicas indivisibles [4]. La primera instrucción especial realiza una llamada al sistema (DOWN) verifica si el valor del semáforo que se ha pasado como parámetro es mayor que cero, de ser así, lo decrementa y continua la ejecución normal. Si el valor que toma la variable es cero, la hebra que llamo DOWN se va a la cola de procesos bloqueados y no accesa al recurso hasta que le corresponda un valor de semáforo mayor que uno. La segunda instrucción especial realiza una llamada al sistema (UP) la cual incrementa el valor del semáforo, con la finalidad de que si algún proceso dormía en ese semáforo, se le permitiera terminar un DOWN y así poder seguir ejecutándose para acceder al recurso compartido [4].

Otra clase de semáforos la constituyen los Semáforos Binarios (mutexes), los cuales son utilizados para garantizar que un solo proceso o hebra accese al recurso compartido durante un lapso de tiempo considerable. Estos semáforos se basan en la exclusión mutua sin espera ocupada, para proporcionar de esta manera una solución adecuada al problema de la sincronización. Los semáforos binarios solo pueden tener dos valores; uno o cero.



3. ACCESO A LOS DATOS COMPARTIDOS

Para acceder a los datos compartidos entre procesos, se pueden implementar diversas técnicas de programación. Una de las más comunes esta basada en el algoritmo del Productor-Consumidor. Para dicho algoritmo, dos procesos comparten un área de memoria. El proceso Productor debe insertar elementos en dicha área mientras esta no este llena. En caso contrario, el proceso se bloqueara y será despertado cuando el consumidor ha eliminado por lo menos un elemento. El proceso Consumidor debe sacar datos del área uno a uno, se bloqueará cuando esta se vacíe y será despertado cuando el productor inserte al menos un elemento [4].

La implantación de esta solución debe contemplar el hecho de la sincronización entre procesos, es decir, el productor - consumidor probablemente funcionaría de manera adecuada si la velocidad de los procesos fuese muy similar. Pero en la mayoría de los casos, las velocidades de ejecución de cada proceso son distintas, y en lo que respecta a la adquisición y presentación de datos, los procesos que adquieren se ejecutan con una velocidad mucho mayor a la de los procesos que visualizan. En este caso, si no existe un acceso sincronizado a la región compartida, el proceso que adquiere, sobrescribirá datos que todavía no han sido presentados.

Se pueden combinar los semáforos clásicos y los binarios para sincronizar adecuadamente el acceso a almacenes de datos compartidos en memoria principal, evitando así, los problemas que se generan debido a las diferencias en la velocidad de ejecución. Una forma típica es implementando el algoritmo del productor y consumidor mediante semáforos [4]. Así, existirán dos semáforos clásicos que controlaran el estado del almacén y un semáforo binario que garantizará la exclusión mutua.

4. METODOLOGÍA

La aplicación propuesta ha sido diseñada bajo el esquema de programación multitareas, utilizando técnicas de diseño orientadas a objetos, además de los elementos de sincronización de procesos y de comunicación interprocesos (IPC) contenidos en la interfaz de programación de aplicaciones de 32 bits (API Win32) de Windows 98. Se utilizó Borland Builder C++ 5.0 como herramienta para el desarrollo del programa, ya que el mismo envuelve dentro de un ambiente visual y fácil de manejar, un compilador de C++ incremental de última generación, y permite el llamado a funciones API.

El programa adquiere y presenta por pantalla dos señales ECG (ver figura 1) que se obtienen de una interfaz que transforma dichas señales provenientes de dos módulos de electrofisiología a un formato digital. Esta interfaz se comunica con la computadora a través del puerto paralelo de capacidades extendidas, el cual se configura en modo bidireccional. Las transferencias de datos se hacen utilizando el método tradicional de encuesta (polling), es decir, sin emplear el acceso directo a memoria (DMA) y sin el apoyo de un manejador de interrupciones [2]. Esto es debido a que la interfaz está constituida por un circuito sencillo que no establece ningún tipo de negociación con el puerto ECP, según lo establecido en la norma IEEE 1284.4.

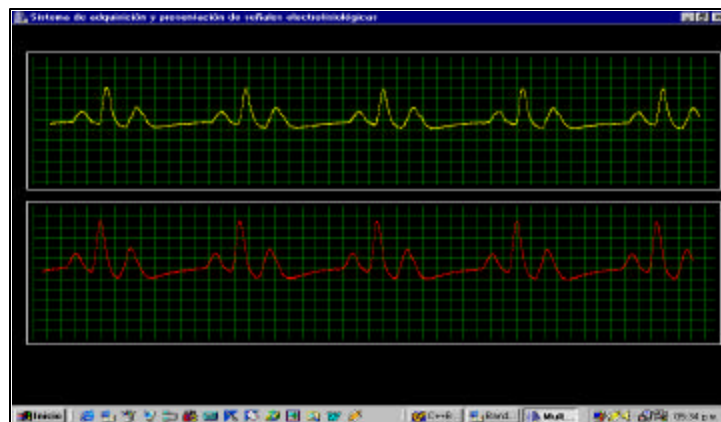


Fig.1. Pantalla de presentación de las señales ECG



El software esta constituido por una hebra principal y dos hebras adicionales; una de adquisición y otra de presentación de los datos. La hebra principal es la responsable de la distribución y programación de los objetos (componentes visuales) que proporcionan la interfaz sobre la cual se presentan las señales en la pantalla. Por otra parte, la inicialización de las variables globales, colas de almacenamiento, la construcción de las demás hebras, la creación o destrucción de los objetos de sincronización y la creación de un área de memoria compartida entre procesos son responsabilidad de esta misma hebra.

El programa funciona de la siguiente manera: basado en el algoritmo de Productor – Consumidor, La hebra de adquisición (Productora), se encarga de tomar muestra por muestra de las señales digitalizadas que llegan al puerto paralelo y las transfiere a una de las dos colas de almacenamiento, donde cada una de ellas corresponde a uno de los canales de adquisición. Sin embargo, estas colas pueden ser compartidas solamente entre cualquier hebra que pertenezca a la propia aplicación, es decir, no pueden ser manipuladas por otros programas. Para permitir que los datos sean compartibles entre diversas aplicaciones, se utiliza la técnica de mapeo de archivos a memoria (*File Mapping*) para crear un área que pueda ser compartida con otros programas, a través de un manejador (handle), un puntero y un nombre. Para aplicaciones Win32, no existe otra manera de compartir memoria entre procesos distintos [1].

El mapeo de archivos a memoria, consiste en la asociación del contenido de un archivo con una porción de la memoria virtual de un proceso, de manera tal que el mismo pueda ser escrito o leído a través de paginación [1]. De esta manera, cualquier proceso puede acceder de forma parcial o total al archivo a través de un puntero a un bloque de paginas (ver figura 2).

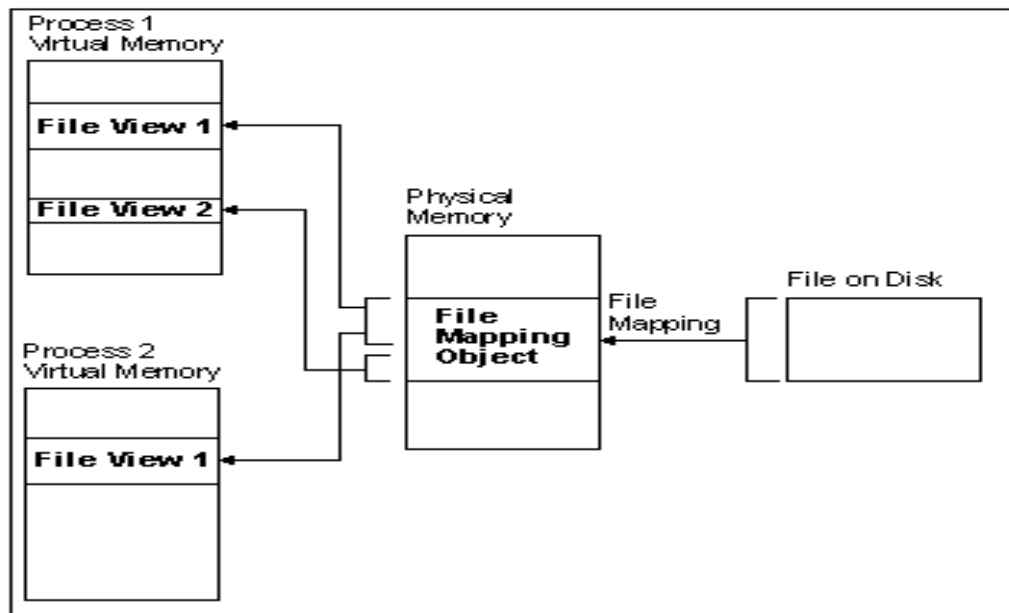


Fig. 2. Mapeo de un archivo en memoria. El archivo es cargado a través de un objeto a memoria física, y cada proceso crea una “vista” total o parcial del mismo, en su propio espacio de direcciones virtuales.

Tomando en cuenta, que la memoria se comparte entre hebras y procesos, que trabajan a velocidades diferentes, el acceso a la misma es controlado a través de semáforos, ya que estos pueden ser manejados tanto por hebras como por procesos desarrollados en Win32 [3][1], es decir, se emplean tres semáforos para aplicar el algoritmo de productor consumidor sincronizado, basándose en la exclusión mutua y sin espera ocupada [4].

El siguiente diagrama de flujo ilustra detalladamente la lógica de la hebra de adquisición:

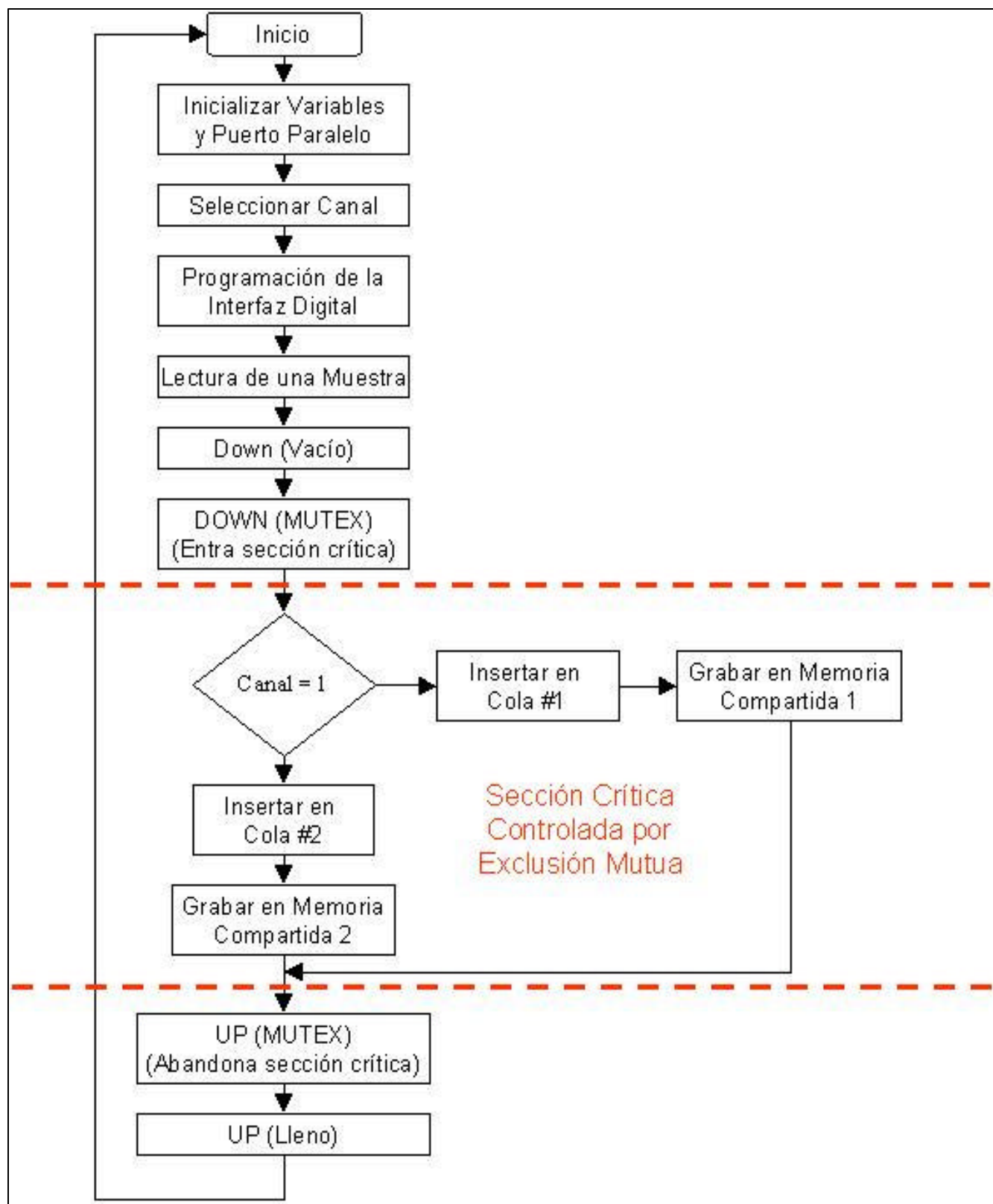


Fig. 3. Flujograma de la hebra de adquisición (PRODUCTORA).

La primera labor que realiza la hebra, consiste en inicializar las variables locales, el puerto paralelo y la selección del canal de adquisición. Posteriormente se programa el módulo de adquisición MAX197. Luego se lee la muestra que ha sido colocada en el puerto paralelo. Inmediatamente, es decrementado un semáforo que cuenta el número de espacios vacíos. Si el valor de este es cero, la hebra se va a dormir debido a que no hay espacio en la cola para guardar esa muestra. Si existe espacio, se decrementa un semáforo binario que garantiza la exclusión mutua. Si el valor del semáforo es cero, la hebra se va a dormir, ya que la cola esta siendo utilizada por la hebra de presentación (Consumidora). Si el valor es uno, la muestra es colocada en la cola respectiva y grabada en una de las dos áreas de memoria compartida (cada una corresponde a una cola), para ponerla a la orden de cualquier proceso consumidor. Después de almacenarse la muestra, Se incrementa el semáforo binario para dar permiso a otros consumidores de tomar datos. Luego, se incrementa el semáforo contador de espacios llenos, lo cual garantiza a cualquier consumidor que existe por lo menos un dato en el almacén.

La hebra de presentación (Consumidora), toma las muestras almacenadas en cada una de las colas para ir dibujándolas en la pantalla. En el siguiente flujograma se ilustra la lógica de funcionamiento:

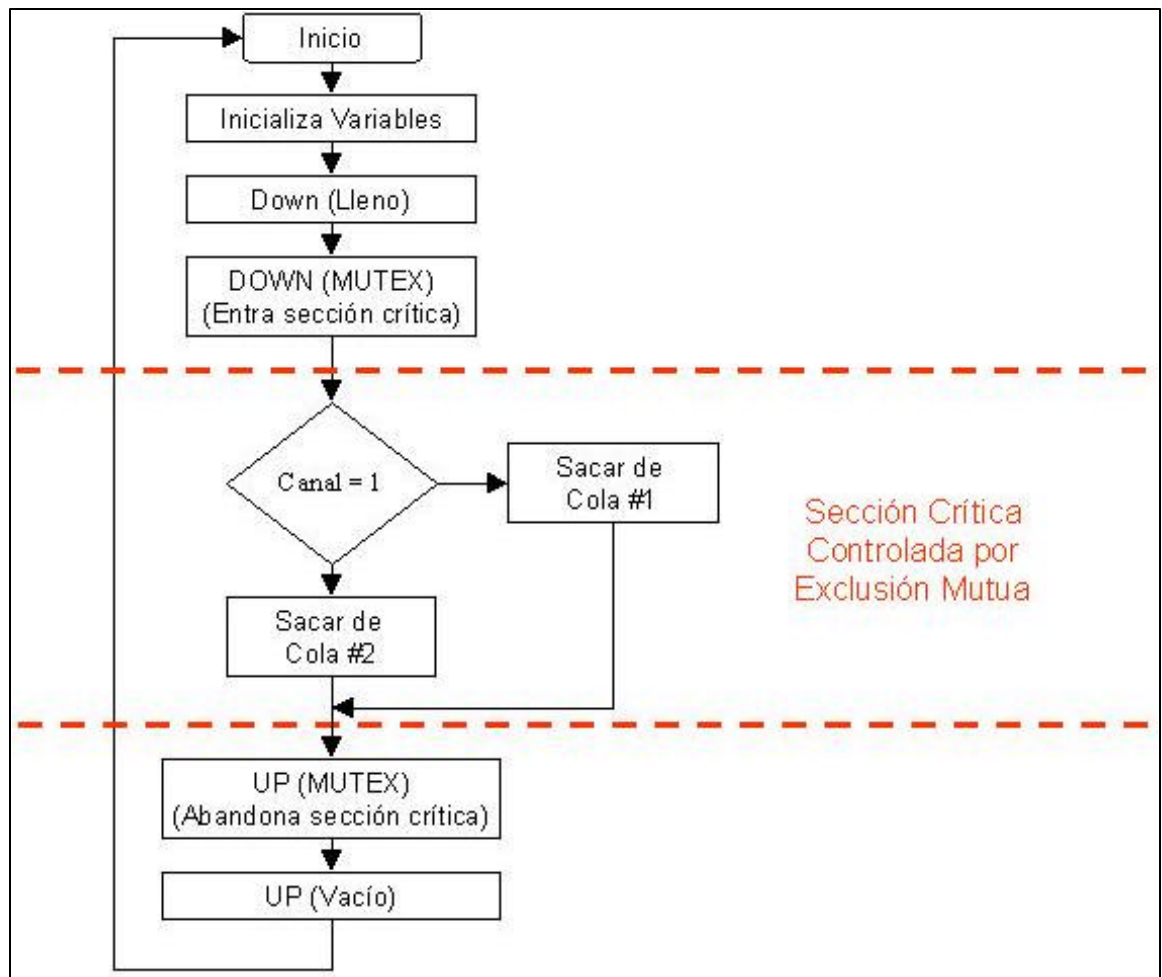


Fig. 4. Flujograma de la hebra de presentación (CONSUMIDORA).

La hebra consumidora, verifica el semáforo de espacios llenos para comprobar la existencia de algún dato en la cola, de ser así, este se decrementa para indicar que se ha liberado un espacio del almacén. Posteriormente, se verifica el semáforo binario para poder hacer uso del almacén. Al obtener el permiso, inmediatamente retira una muestra y cede de nuevo el recurso, incrementando el mutex. Por último, incrementa el contador de espacios vacíos, para garantizar al productor que el buffer no esta lleno. La muestra toma papel como parte de la señal en la pantalla.

5. RESULTADOS

Los objetivos principales de esta investigación están enmarcados en el diseño de un programa que adquiere señales electrofisiológicas a una velocidad de muestreo óptima, donde dichas señales pueden ser compartidas con otras aplicaciones de diagnóstico, esto es, que mientras un paciente es monitoreado con esta aplicación, otras aplicaciones pueden emitir un diagnóstico a tiempo real sobre los datos adquiridos (ver figura 5).

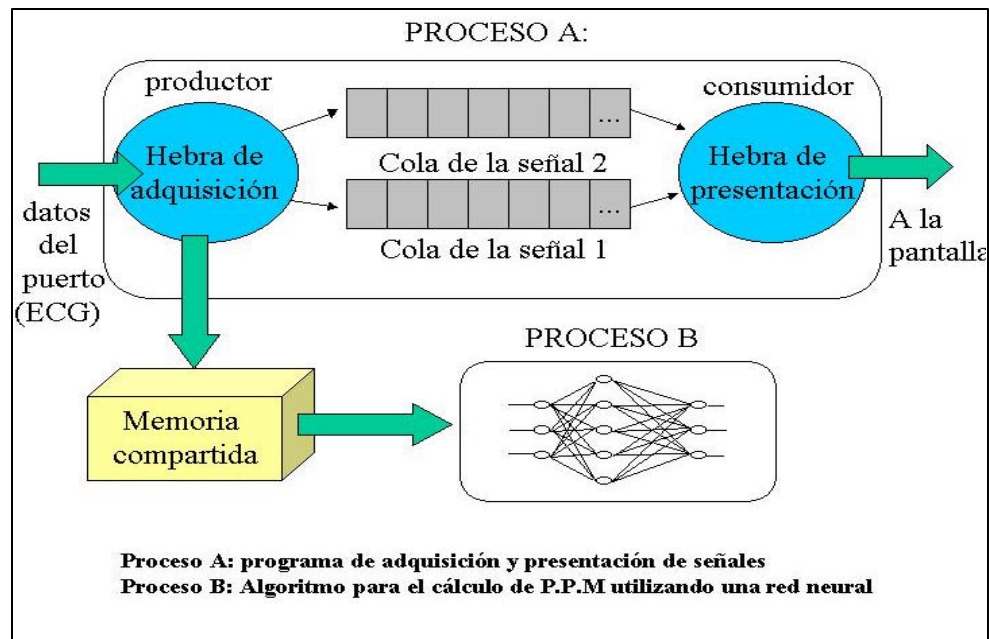


Fig. 5. Descripción gráfica del funcionamiento ideal del programa.

Por estas razones, el criterio de evaluación de este trabajo está basado en la observación de dos aspectos fundamentales: 1) el desempeño del sistema en términos de velocidad. 2) La capacidad de compartir datos con otras aplicaciones. Para este fin fueron utilizados los siguientes recursos:

EQUIPO	DESCRIPCION
Computadora personal	Pentium II 450 Mhz. 64 MB RAM.
Osciloscopio Digital	Tektronix TDS220 100Mhz
Módulos de electrofisiología	Hewlett Packard 8811 A, con capacidad de monitoreo de ECG y EEG.
Interfaz de PC – Módulos	Diseñada y manufacturada en el Laboratorio de Bioingeniería UNET. Tiempo óptimo de adquisición: 21 μ Seg.

Tabla 1.

Equipos utilizados en las pruebas

Las pruebas fueron realizadas en dos fases. En la primera se observó la velocidad de adquisición en un osciloscopio digital, tomando la medida en la patilla de fin de conversión del MAX197. La medida obtenida fue de 6452 Khz., tomando en cuenta dos canales de adquisición entonces, la velocidad medida es igual a 3226 muestras por segundo. La segunda prueba consistió en verificar que no se produjeran conflictos al compartir las señales adquiridas con otros procesos. Para ello se desarrolló un programa que calcula la frecuencia cardíaca utilizando una red neural



Este programa obtiene las señales del área compartida de memoria. Para ello utiliza instrucciones especiales para crear una interfaz que le permita trasladar los datos desde la memoria hacia su propio espacio de direcciones virtuales. Además, utiliza los semáforos creados anteriormente por el programa que es objeto de evaluación. Para realizar estas dos tareas, este programa obtiene los manejadores y los nombres de cada uno de los semáforos pertinentes.

Posteriormente se ajusto al simulador de señales ECG para que generara señales de con valores de frecuencia de 70,80,110 p.p.m. y seleccionados arbitrariamente. Al ejecutar las dos aplicaciones simultáneamente, se observó que el programa de adquisición y presentación visualizó las señales sin ninguna deformidad aparente, y el programa de calculo de frecuencia arrojó los valores correspondientes a cada uno de los ritmos cardíacos ajustados en el simulador.

6. CONCLUSIONES

- Este trabajo es un aporte al área de la bioingeniería ya que las señales que este adquiere, pueden ser utilizadas por otras aplicaciones de diagnóstico e investigación que sean desarrolladas posteriormente.
- Todas las aplicaciones utilicen los datos adquiridos por este programa, deberán utilizar las funciones de comunicación entre procesos (IPC), mapeo de archivos a memoria y objetos de sincronización que proporciona la API Win32 de Windows 98.
- El funcionamiento adecuado en términos de velocidad y sincronización es dependiente del número de procesos consumidores y la complejidad de los mismos
- Haciendo algunas modificaciones, este programa podrá usarse en sistemas distribuidos a través de la filosofía de cliente – servidor, para de esta manera acceder a la información desde lugares remotos al monitor.

AGRADECIMIENTOS

Este trabajo fue realizado gracias al apoyo del Decanato de Investigación de la Universidad Nacional del Táchira.

REFERENCIAS

- [1] Inprise Corporation. “Borland Builder C++ 5.0, Microsoft Win32 reference” . California USA. 2000
- [2] M. Hernández, O. Fernández. “Adquisición y presentación de señales biomédicas utilizando programación basada en multihebras”. pp. 130 – 134 Tendencias Actuales en Bioingeniería. 2000.
- [3] H Schildt. “Programación en C/C++ bajo Windows 95”. Editorial McGraw Hill: Barcelona, España. 1996.
- [4] A. Tanenbaum. “Sistemas Operativos Modernos”. Editorial Prentice Hall: Madrid, Segunda Edición. 1992.