métodos de análisis, con una probabilidad de 97,18 % . (ver Tabla 5).

TABLA 5. Resultados estadísticos según programa microstat para las medias en los tres métodos de análisis

Grupo (método)	Media
1	31,738
2	31,63
3	31,807
Media total	31,725

CONCLUSIONES

El método de análisis propuesto para la determinación de cromo en cromitas refractarias, resulta confiable, efectivo, simple y exacto; disminuye la peligrosidad en las operaciones; no requiere del uso de crisoles de hierro, y consume menor cantidad de reactivos, combustible y energía eléctrica.

BIBLIOGRAFÍA

ACEVEDO DEL MONTE, E.: "Determinación cuantitativa de cromo en mineral laterítico", *La Minería en Cuba*, 2 (3) :21-26, 1976.

DELAHAY, P.: *Análisis instrumental*, Instituto Cubano del Libro, C. Habana, 1992.

DICCIONARIO QUÍMICO. Tomo II, Editorial Universitaria, La Habana, 1986.

EDWARD, H. y W. FRED: *Mineralogía*, Instituto Cubano del Libro, C. Habana, 1970.

HOEL, P.: *Estadística elemental*, Editorial Pueblo y Educación, 400 p., C. Habana, 1979.

LEE, K. and D. LEE: "Selective separation of metal ions by a chelating", *Analytical Chemistry*, L (2) :255-258, 1978.

MILNER, A. and J. WRITESIDE: *Introduction of atomic absorption spectrophotometry*, Instituto Cubano del Libro, C. Habana, 1989.

ROSALES, E. y R. GUZMÁN: "Estudio de interferencia por absorción atómica del níquel, cobalto, cromo y manganeso en los minerales cubanos", Revista Cubana de *Química*, 3 (2), 1992.

SÁNCHEZ, C. y M. DURÁN: *Manual de operaciones de la nueva planta de beneficio de cromo*, EGMO, 1986.

SÁNCHEZ, M.: "Determinación de cromo y manganeso en aguas mediante espectrofotometría de A.A.", Revista Cubana de Química, 4 (1) :48-53, 1988.

SANTANA, O.: "Explotación de yacimientos de cromo refractario en la República de Cuba", Revista *Tecnológica*, 15 (1) :47-51, 1985.

SEMINOV, V.: "Yacimientos cromíticos de Cuba", Revista *Tecnológica*, 6 (3) y (4) :33-51, 1988.

FACULTAD DE MECÁNICA

El objetivo del trabajo de la carrera Ingeniería Mecánica lo constituyen las máquinas, equipos e instalaciones industriales presentes en las diferentes esferas de actuación del profesional con énfasis, por las condiciones particulares del territorio, en la industria minero-metalúrgica, la que de manera general puede estar vinculada a procesos industriales, procesos de producción de piezas y máquinas, procesos de utilización y transformación de la energía térmica y máquinas automotrices.

Las líneas fundamentales de investigación de la facultad están relacionadas con el proceso de soldadura y conformación de metales con el uso de explosivos; el desarrollo de nuevos materiales y el mantenimiento de equipos e instalaciones industriales.

El posgrado está estrechamente vinculado con el desarrollo de las líneas de investigación fundamentales de la facultad: la fabricación y recuperación de equipos y piezas para la industria (fabricación de recipientes a presión y de instrumentos de corte) y la formación académica en la maestría Electromecánica.

El claustro está formado por 14 profesores con categoría de auxiliares, asistentes e instructores.

Vol. XIII, No. 3, 1996

EMULADOR DE TIEMPO REAL PARA SISTEMAS DE SUPERVISIÓN Y CONTROL DE PROCESOS

Dr. Rafael Trujillo Codorniu

Departamento de Matemáticas. Instituto Superior Minero Metalúrgico

RESUMEN:

Se describen las principales características del emulador multitarea de tiempo real para DOS en modo protegido, KEROS, desarrollado por el autor y que ha servido de base para sistemas de supervisión en la industria del níquel y en algunos centrales azucareros de Cuba.

ABSTRACT:

An explanation of the KEROS real time multitasking emulator for DOS Protected Mode, which has been used as a key part of supervision systems on sugar and nickel industries in Cuba and its principal features are described in this paper.

Los sistemas computarizados de supervisión y control de procesos, son sistemas en los cuales una o varias computadoras monitorean los parámetros y variables de un proceso dado, y ejercen determinadas acciones de control que oscilan desde la simple información y advertencia al operador, hasta el accionamiento a distancia.

La necesidad de incrementar la eficiencia y la competitividad de la industria cubana ha motivado, entre otros factores, el creciente interés hacia tales sistemas en nuestro país.

Específicamente en la Unión del Níquel, bajo la dirección del grupo EROS (grupo de desarrollo de SADPT de la empresa "René Ramos Latour" dirigido técnicamente por los ingenieros Iván Rodríguez y Carmen Mulet), se han desarrollado trabajos en este sentido. El sistema EROS v.1.0 desarrollado por este grupo ha sido aplicado en varias plantas de las empresas niquelíferas "Ernesto Che Guevara" de Moa y "René Ramos Latour" de Nicaro, y experimentalmente se prueba en algunos centrales azucareros.

Una de las principales características que distinguen a estos sistemas es que interactúan no sólo con el hombre, sino además, con una serie de dispositivos que están conectados en línea; también los mismos funcionan bajo estrictas limitaciones de tiempo. Aquí, a diferencia de otros sistemas, la computadora no puede "escoger" el ritmo de trabajo. Ella está obligada a reaccionar a tiempo ante los variados cambios que pueden ocurrir en el objeto de control. El ciclo de captación de la información, análisis de la misma y el pronóstico de la evolución futura del proceso, deben ser "suficientemente" rápidos como para permitir la retroalimentación. Por supuesto que el término "suficiente" depende del proceso en cuestión, aunque como regla el tiempo de respuesta de un sistema de este tipo debe estar en el orden de los segundos o fracciones de segundos. Es por esta limitación en el tiempo que se dice, que estos sistemas operan en tiempo real. La complejidad de los sistemas de control de procesos varía desde sistemas pequeños, basados en microprocesadores, hasta verdaderos gigantes como los sistemas de seguridad de plantas nucleares y de naves espaciales (ver, a tales efectos, Spector-Alfred, 1984, donde se discuten algunas facetas del sistema de control de los transbordadores espaciales norteamericanos).

Cuando el sistema es pequeño, puede organizarse el programa de manera tal que la captación de las mediciones, su procesamiento, el control y la atención al operador se efectúen de forma secuencial. Si el sistema es complejo o la captación de los datos es asincrónica, la organización secuencial puede hacer muy lenta la respuesta ante eventos de importancia y, generalmente, se estructura el programa como un conjunto de tareas que se ejecutan de forma concurrente de acuerdo con una prioridad dada. Esto implica que para el diseño de los Sistemas de Supervisión y Control se necesita generalmente un sistema operativo multitarea en tiempo real.

Desafortunadamente los entornos más difundidos en nuestro país, el Sistema Operativo DOS y el Windows 3.1, no son sistemas en tiempo real. El DOS no está diseñado para multitarea y el Windows 3.1 utiliza un mecanismo conocido como multitarea cooperativa que en ningún modo puede considerarse adecuado para el tiempo real (ver Zamora-Matamoros, 1995, donde se hace una excelente discusión de las diferentes versiones de Windows bajo la óptica del tiempo real). Sólo en las más recientes versiones del Windows; el Windows NT y el Windows 95, se implementa la multitarea no cooperativa y la posibilidad de asignar prioridades a las tareas. Sin embargo, la documentación de estos temas no está muy difundida en nuestro país, y la mayoría de los lenguajes de alto nivel no soportan interfaces hacia estas nuevas herramientas, lo que hace muy difícil su uso en el diseño de Sistemas de Supervisión y Control. Además, la adquisición de sistemas operativos profesionales adecuados al tiempo real, como por ejemplo el DOS Concurrente de la Digital Research, tiene como aspecto negativo la aparición de múltiples incompatibilidades con los sistemas de desarrollo más difundidos (C++, Borland Pascal, etc.). A fin de cuentas el "sueño" de un programador de Sistemas de Supervisión es poder usar el lenguaje de programación y los depuradores a los cuales está habituado.

Estas consideraciones motivaron la aparición de Sistemas en tiempo real "sobre" el DOS. Es decir, sistemas que tomaban del DOS el manejo de archivos y otros elementos, adicionándole primitivas para el con-

trol del tiempo real, tratando, en lo posible, de que el resultado fuera totalmente transparente para el usuario y minimizando los problemas de compatibilidad. El ejemplo más difundido en Cuba es el SOTRE, Sistema Operativo en tiempo real, diseñado por el ingeniero Reynaldo Mañalich Mazo y que ha servido de soporte para múltiples aplicaciones, entre las cuales se encuentra la versión 1.0 del Sistema EROS al que se hizo referencia anteriormente. No obstante, la evolución del hardware y la creciente complejidad de los sistemas de supervisión han desbordado las capacidades del SOTRE. En efecto, el SOTRE opera en el modo real teniendo acceso a sólo 640 Kb de memoria, y tiene conocidos problemas de compatibilidad con los depuradores más difundidos, por lo que la puesta a punto de los programas se hace realmente difícil.

En el presente trabajo se exponen las características del Emulador de tiempo real sobre DOS, KEROS, diseñado por el autor, y las experiencias en casi 2 años de explotación del mismo.

Este emulador fue aplicado en la versión 2.0 del Sistema EROS, garantizando acceso a toda la memoria disponible, operando en modo protegido y con muchas facilidades para la puesta a punto, entre sus ventajas más significativas.

Características principales del Keros

El emulador de tiempo real KEROS es simplemente una librería de enlace dinámico (DLL), en la cual se exportan un conjunto de procedimientos y funciones que posibilitan la creación de sistemas multitarea operando en tiempo real. No posee, a diferencia del SOTRE, un intérprete de comandos, ni queda residente en memoria, por lo que sólo actúa durante el tiempo en que las aplicaciones hacen uso de la librería, ni se propone en modo alguno cargar y ejecutar programas diseñados para el DOS. De esta manera, a los efectos del programador, el sistema funciona como una extensión del lenguaje fuente. Es por ello que preferimos nombrarlo emulador y no sistema operativo. Esta filosofía no es nueva, y para microcomputadoras tiene antecedentes tan curiosos como el diseño de una extensión en tiempo real y multitarea para el BASIC (ver Dasiewicz, 1985). Aunque el KEROS se diseñó en Pascal y en lenguaje ensamblador, puede ser usado por cualquier sistema de desarrollo que permita la importación de procedimientos desde librerías de enlace dinámico. Sus características principales son:

- Trabaja en el modo protegido del micro 80 386 o 80 486. Esto permite acceso a toda la memoria instalada y la protección de los datos por hardware.
- Es altamente compatible con el SOTRE. Se trató de que las llamadas al Sistema fueran similares a las del SOTRE con el objetivo de facilitar el trabajo de los programadores acostumbrados a dicho sistema, y permitir la modernización de los programas ya desarrollados.
- Permite instalar hasta 64 tareas con diferentes niveles de prioridad. Las tareas pueden estar definidas en el módulo ejecutable (EXE) o en librerías de enlace dinámico (DLL) y además compartir el segmento de datos del programa principal o tener segmentos de datos independientes. Cada una de estas tiene su propia pila (stack) y el tamaño del mismo es configurable, con la única limitación de que no sobrepase los 64 Kb. Dichas facilidades permiten incrementar extraordinariamente la modularidad de los sistemas.
- El emulador tiene sólo 5 Kb de código y usa 11 Kb de memoria, pero tiene un segmento de datos independiente, por lo que no compite con el segmento de datos de las tareas que se definan. Es compatible con los depuradores profesionales (por ejemplo con el Turbo Debugger for Protected Mode de Borland International) lo que hace muy cómoda la puesta a punto de los sistemas.
- Permite cambiar la frecuencia del reloj para lograr animaciones suaves.
- Facilita cambiar la prioridad de las interrupciones de hardware con el objetivo de priorizar la captación de datos.
- Contribuye a la sincronización entre eventos y tareas.
- Asegura la ejecución de tareas periódicas y de tareas periódicas calendarias.
- Posee implementadas primitivas de alto nivel para el manejo de la impresora y la comunicación serie.
- Incorpora el tratamiento de semáforos para la comunicación entre procesos.

Manejo de los procesos o tareas

Cada tarea en el sistema es identificada con un número (entre 0 y 63) y tendrá asociada una prioridad (entre 0 y 127), siendo la representada por el cero la de más alta prioridad. Aunque en rigor para cualquier instante de tiempo la CPU está ejecutando sólo una tarea, en el curso de un segundo puede trabajar en varios procesos commutando entre ellos de forma rápida lo que da la ilusión de concurrencia. Los procesos o tareas pueden estar lógicamente en tres estados, en ejecución (que en realidad hace uso de la CPU en ese instante), bloqueado (incapaz de ejecutarse hasta que suceda algún evento externo) y listo (ejecutable; se detiene temporalmente para permitir que se realice una tarea de mayor prioridad). El planificador del KEROS realiza la commutación entre las tareas, de manera que la de más alta prioridad y que además esté lista para ejecutarse, siempre tenga el control del CPU. Antes de que un proceso pueda ser ejecutado, este debe ser "creado" y colocado en la cola de procesos listos. Esto se hace a través de las llamadas Install-Task y Execute-Task. La primitiva Install-Task alerta al KEROS de la existencia del proceso, el número que lo identifica, su prioridad, la dirección donde está su primera instrucción y la cantidad de espacio privado que necesita reservar para el Stack. Después de su creación, el proceso queda bloqueado hasta que sea "despertado" por una llamada a Execute-Task o se especifique como tarea periódica.

El procedimiento Execute-Task inserta la tarea que se especifique como parámetro en la cola de procesos listos y llama al planificador. Por supuesto que el momento en que realmente tomará el control del CPU esa tarea, depende de su prioridad y de la del resto de las tareas que estén listas. El comienzo de la emulación del tiempo real se especifica con el procedimiento InitRealTime. Aquí se redefinen todos los vectores de interrupción de hardware y se le da el control a la tarea de prioridad más alta que esté lista. Esto presupone que, previo al llamado a este procedimiento, exista al menos una tarea instalada mediante la invocación a Install-Task, y que alguna de ellas tenga la posibilidad de tomar el control, porque sea periódica o porque haya sido mandada a ejecutar a través de Execute-Task. El retorno a la instrucción que sigue a InitRealTime se efectúa cuando en alguna tarea se hace un llamado al procedimiento EndRealTime. Esta primitiva finaliza la ejecución de todas las tareas, restaura los vectores de interrupción tomados y salta a la instrucción del programa que sigue al procedimiento InitRealTime.

De esta forma EndRealTime no es un procedimiento propiamente dicho, ya que no retorna nunca al punto donde se llama, más bien puede ser visto como la secuencia de retorno de InitRealTime.

Una tarea puede bloquearse voluntariamente llamando al procedimiento WaitState, estado del cual sólo puede salir si otra tarea la despierta invocando a Clr-Wait. Estos procedimientos, que se soportan con el fin de garantizar compatibilidad con el SOTRE, pueden ser usados para la sincronización de procesos, aunque a estos efectos se sugiere utilizar semáforos.

Las tareas pueden cambiar dinámicamente su prioridad a través de Change-Prior. Esto implica la reinserción en la cola en que actualmente se encuentre la tarea de acuerdo con su nueva prioridad.

En el KEROS puede declararse una tarea de fondo. Por esta se entiende un proceso al cual se le da el control sólo cuando no existen otros que estén listos en ese instante. La tarea de fondo no puede bloquearse ni hacer uso de semáforos o recursos que potencialmente la pudieran bloquear. El stack asociado a la misma siempre es el del programa principal y por ello el mismo puede modificarse con la directiva de compilación {$M} si se utiliza el Borland Pascal para el desarrollo del programa principal. En general, el tamaño del Stack que debe declararse asociado a una tarea, depende de las llamadas anidadas a procedimientos, del uso en los mismos de variables locales, de la recursividad y además, debe asimilar las interrupciones de Hardware, pues las mismas usan como regla, el Stack de la tarea que esté en ejecución. Es por ello que es potencialmente peligroso declarar espacios muy pequeños para el Stack, y se sugiere que en ningún caso sea inferior a 1 Kb. Usualmente 8 Kb es suficiente.

Sincronización entre procesos. Manejo de los recursos

Algunos dispositivos, como la impresora, el coprocesador matemático y otros, si se utilizan por más de un proceso, en un momento dado pueden crear graves conflictos. Del mismo modo cuando dos o más procesos leen o escriben datos en zonas de memoria compartida, el resultado final depende de cuál se ejecuta en un momento preciso. Situaciones como estas se denominan condiciones de concurso. La depuración de programas que contienen condiciones de concurso suele ser difícil, pues los problemas aparecen sólo de vez en vez, como algo misterioso e inexplicable. Se necesita una adecuada sincronización de los procesos que necesitan acceder a recursos compartidos.

En la mayoría de los textos de Sistemas Operativos se examinan las diferentes primitivas que se han propuesto para la solución de este problema (ver por ejemplo, Peterson y Silberschatz, 1986). El KEROS utiliza la solución propuesta por Dijkstra (1968), que consiste en la introducción de semáforos. Un semáforo es una variable entera sobre la cual se pueden efectuar 2 operaciones básicas; la operación Down y la operación Up. La operación Down con un semáforo verifica si el valor del mismo es mayor que cero. Si es así, disminuye el valor y simplemente retorna. Si el valor era menor o igual a cero, la tarea que efectúa el llamado se bloquea, colocándose la misma en una lista de espera asociada al semáforo. La verificación del valor, su alteración y el bloqueo se realizan en una sola acción atómica (indivisible).

Usualmente, si el procesador no implementa instrucciones especiales para este fin, se deshabilitan las interrupciones en el cuerpo de este procedimiento para lograr su indivisibilidad. Por su parte la operación Up incrementa el valor del semáforo. Si uno o más procesos estaban bloqueados en ese semáforo, incapaces de completar la operación Down anterior, se escoje uno de ellos al azar (en el KEROS se escoge el que lleva más tiempo bloqueado, es decir de acuerdo al principio FIFO) y se le permite completar la operación Down. Por tanto, después de una operación Up con un semáforo con procesos bloqueados en él, el semáforo continuará estando en cero aunque con un proceso bloqueado menos. Al igual que en Down, la acción de incrementar el semáforo y de desbloquear un proceso es indivisible. Los semáforos cuyos valores iniciales son iguales a la unidad pueden ser utilizados por uno o más procesos para controlar el acceso a recursos compartidos reusables. En efecto, luego del primer Down que se efectúe sobre el semáforo, cualquier otro proceso que intente realizar la operación Down quedará bloqueado hasta que se "libere" el recurso a través de un Up.

En el KEROS se permiten hasta 64 semáforos, de los cuales los primeros 32 se inicializan con 1 y pueden ser usados para regular el acceso a recursos. Los restantes 32 se inicializan con cero, lo que les permite ser utilizados para la regulación de recursos que necesitan ser "producidos" por una tarea, y que usualmente son "consumidos" por otras. Los ejemplos más típicos de este tipo de recursos son los buffers de entrada-salida y los mensajes. La "producción" del recurso equivale a la operación Up con el semáforo correspondiente, mientras que el "consumo" equivale al Down. Nótese que esto garantiza que la tarea que solicita un recurso que aún no se haya producido, quede bloqueada hasta tanto otro proceso no produzca el mismo. Para el tratamiento de recursos reutilizables y por razones de compatibilidad con el SOTRE, se definieron en el KEROS las primitivas GetResource y FreeResource. GetResource solicita al sistema un recurso. Si este está libre, es asignado a la tarea que hace el llamado. Si el recurso está tomado, la tarea se pone en la cola del semáforo asociado al recurso, en espera de que el recurso se libere por un FreeResource. El número de recurso válido que puede pasarse

a este procedimiento está entre 0 y 30, puesto que el recurso 31 se reserva para el DOS. Por su parte Free Resource libera un recurso al sistema tomado previamente por un GetResource. Si en la cola de espera del mismo habían tareas bloqueadas, se despierta la primera (principio FIFO, First In, First Out). Si el recurso no pertenece a la tarea que hace el llamado la petición se ignora.

En ocasiones un proceso no puede esperar indefinidamente por la liberación o la producción de un recurso. Para estos procesos la primitiva Down es inaplicable. En previsión de estos casos se definió en el KEROS la primitiva DownTime.

La función lógica DownTime con un semáforo, verifica si el valor del mismo es mayor que cero. Si es así, disminuye el valor y simplemente retorna. Si el valor era menor o igual a cero la tarea que efectúa el llamado se bloquea, colocándose en una lista de espera asociada al semáforo, de la cual sale cuando algún proceso ejecuta la operación Up correspondiente, o cuando se agota un tiempo definido como parámetro en DownTime. Al igual que en Down y Up la verificación del valor, su alteración y el bloqueo se realizan en una sola acción atómica (indivisible). La función devuelve verdadero si hubo "TimeOut", es decir si la tarea se despertó por haber agotado el tiempo máximo de espera, en caso contrario devuelve el valor falso. Ejemplos clásicos del uso de esta primitiva son la espera por la liberación de la impresora, esperas por teclas en determinados casos y otros.

Debe señalarse que en el KEROS no está implementado ningún mecanismo de mensajes entre procesos. Sin embargo, es bien conocido que existe una equivalencia esencial entre sistemas basados en mensajes y sistemas basados en semáforos.

Comunicación por el puerto serie

En muchos sistemas de supervisión y control (como por ejemplo el EROS) la captación de la información se efectúa a través de la interface RS-232. El KEROS provee procedimientos de alto nivel tanto para la configuración de los parámetros de los puertos de comunicación, como para el envío y la recepción de información. La función OpenCom permite definir para un puerto dado, la velocidad de trasmisión, el chequeo de paridad, el número de bits de parada y la longitud de palabra.

La función devuelve el valor lógico verdadero si el puerto especificado existe físicamente, es decir, si está reportado en la lista de equipamiento de la ROM BIOS y la inicialización del mismo fue exitosa.

En caso contrario devuelve falso. Esta función puede usarse tanto dentro del tiempo real como fuera de él, pero como en el proceso de inicialización se modifican temporalmente parámetros del controlador de interrupciones, es preferible usarlo antes del llamado a InitRealTime. Después de la inicialización del puerto la recepción y trasmisión de datos por dicho canal, se hará por interrupciones a través de las primitivas SendCom y ReceiveCom.

La inicialización del puerto puede interferir con el MOUSE si éste está instalado en el mismo puerto, serie que se pretende usar para la comunicación. Si por el contrario (como sucede en la mayoría de los equipos modernos) el MOUSE es del tipo PS/2, la recepción o trasmisión por los puertos serie no entra en conflicto con el MOUSE.

La primitiva SendCom permite enviar un mensaje vía puerto serie. Esta función devuelve cero si el mensaje se envió satisfactoriamente o un código de error en caso contrario. El mensaje que se trasmite se divide en tres campos: Longitud, Datos y CRC. Esta estructura de mensajes garantiza una alta confiabilidad. El campo longitud tiene 2 bytes y almacena la longitud total del mensaje (longitud de los datos + 4). El campo CRC (Chequeo de Redundancia Cíclica) es de 2 bytes y almacena el CRC del mensaje. La función SendCom calcula automáticamente el CRC del búffer y lo deposita en el campo adecuado. Luego envía el primer caracter del búffer y bloquea la tarea que hace el llamado hasta que el manejador de interrupciones del canal haya terminado la trasmisión completa del búffer, después de lo cual es despertada la tarea. En prevención de que pueda perderse una interrupción por el canal y que como consecuencia de ello la tarea quede indefinidamente bloqueada en espera de que se complete la trasmisión, se lleva un monitoreo constante de cuantos tics del reloj transcurren entre la trasmisión de un caracter y el otro. Si el mismo excede un valor prefijado, la tarea es despertada y la función devuelve el código de error de mensaje truncado.

Por su parte, la primitiva ReceiveCom permite recibir mensajes desde un puerto serie. La función retorna cero si se recibió satisfactoriamente el mensaje o un código de error en caso contrario. El mensaje que se recibe tiene la misma estructura que para SendCom. Si el arribo del primer caracter demora demasiado, la función devuelve el código de error entre mensajes. En caso contrario se van depositando los caracteres en el búffer hasta que el tiempo de arribo entre dos de ellos sea mayor que el tiempo máximo entre caracteres. El mensaje recibido se somete a chequeo de longitud y de CRC, y luego se transfiere hacia el búffer del usuario.

El procedimiento SetTimeOut establece los tiempos máximos entre caracteres y mensajes para la comunicación por el RS232 a los que se hizo referencia anteriormente. El tiempo máximo entre caracteres es el tiempo después del cual el sistema asume que el mensaje que se estaba recibiendo ha terminado, mientras que el tiempo máximo entre mensajes establece el intervalo máximo de tiempo en que quedará bloqueada una tarea que invocó a ReceiveCom en espera del primer caracter del mensaje, y al término del cual la tarea se despierta.

Tratamiento de la impresión

Al igual que en el SOTRE, la impresión se realiza como parte de la tarea de fondo del sistema. La primitiva OpenPrn inicializa un puerto para la impresión y crea un búffer de tamaño definido. Después de esto, todos los procedimientos de impresión depositan los caracteres que se desean imprimir en ese búffer, y luego en fondo (y en cada tic del reloj) se extraen del mismo y se pasan físicamente a la impresora. Es posible definir diferentes tamaños de búffer para cada puerto, en dependencia de la velocidad de la impresora asociada a él y de la frecuencia de uso. Del mismo modo, si se abren los dos puertos es posible imprimir concurrentemente con el resto de las tareas por dos impresoras a la vez. Las funciones PrnWrite, PrnWriteln y PrnBinWrite permiten la salida tanto de cadenas de caracteres como de datos binarios. En todos los casos si el búffer de impresión está casi lleno y no puede alojar la secuencia de datos completa, se deposita la parte que cabe devolviéndose como resultado de la función el número de caracteres realmente transferidos. La función PrnAvail permite monitorear el espacio disponible en el búffer de impresión.

CONCLUSIONES

La experiencia de trabajo con el emulador ha mostrado que tiene una alta fiabilidad, y permite crear sistemas de tamaño profesional con uso profuso de la memoria extendida, tanto para alojar código como datos, además, puede considerarse una buena alternativa para los programadores no familiarizados con las nuevas herramientas de tiempo real del Windows95 que deseen diseñar sistemas de supervisión y control de procesos, juegos o cualquier otro programa que tenga estrictas limitaciones de tiempo. En el anexo se expone la interface de la Unit KEROS para los programadores que desarrollen sus programas en Borland Pascal. Herramientas análogas se diseñaron para C.

BIBLIOGRAFÍA

DASIEWICZ, P.: "A Soft Real-Time Multi-Tasking Basic Interpreter", SIGPLAN Notices, 20 (6), 1985.

DIJKSTRA, E.W.: *Cooperating Sequential Processes, Programing Languages*, Academic Press, Londres, 1968.

MANUAL DE OPERACIÓN DEL SISTEMA DE SUPERVISIÓN Y CONTROL: EROS, 1993.

MAÑALICH MAZO, R.: *Manual del Sistema Operativo de tiempo real*, EDSAD, MINBAS, Ciudad de La Habana, 1989.

PETERSON, J.L y A. SILBERSCHATZ: *Operating System Concepts*, ENPES, Ciudad de La Habana, 1986.

SPECTOR, ALFRED Z.: *Computer Software for Process Control, Scientific American*, 1984.

ZAMORA MATAMOROS, R.: *Herramientas para la Programación de Sistemas Automatizados de Control de Procesos*, Tesis de maestría, Ciudad de La Habana, 1995.

ANEXO

```
unit Keros;
interface
uses Objects, Dos;
{************ Constantes **************}
const
      RecursoDOS   =31;
      RecursoMem   =30;

{Constantes de la bandera de estado}
const
      scRealTime   =1;
      scDebug            =2;
      scTimeSlice  =4;
      scTimeReq    =8;
      scExitReq    =16;

{************Constantes relacionadas con el tratamiento del teclado************}
const
      kbEnaReset   =1;
      kbEnaBreak   =2;
      kbEnaPause   =4;
      kbEnaPtrScr  =8;
      kbEnaCtrlAltEsc      =16;
      kbSemaforo   =63;

const {**** Para uso en OpenCom ****}
      cNoParity    =0;
      cParityOdd   =1;
      cParityEven  =3;
      cStickParity =4;

{*************** Procedimientos ***************}
procedure Timer(dh, dm, eh, em, NoTask: integer);
function  IsInReadyList(Task:word): boolean;
procedure GetResource(NoRecurso: word);
procedure FreeResource(NoRecurso: word);
procedure Down(Semaforo: word);
function DownTime(Semaforo, TimeInSeg: word): boolean;
procedure Up(Semaforo: word);
procedure WaitState;
procedure Change_Prior(NoTask,NewPrior: word);
function  Get_Prior(NoTask:word): word;
procedure Install_Task(NoTask,Prior: word;
   NAddress:pointer; StackSize: word);
procedure Execute_Task(NoTask: word);
procedure Terminate_Task(NoTask: word);
procedure Clr_Wait(NoTask: word);
procedure InitRealTime;
procedure Put_Backgnd(BackAddr: pointer);
procedure SetIRQPriority(IRQBottom: word);
procedure Act_period(DinCount, EstCount,
   NoTask: integer);
procedure Act_periodT(DinCount, EstCount,
   NoTask: integer);
procedure Act_PeriodM(DinCount, EstCount,
   NoTask: integer);
procedure Delay_task(time: integer);
procedure Delay_taskT(time: integer);
procedure Change_Frec( NewFrec: integer);
function  Clock_Frec: integer;
procedure Susp_period(NoTask: integer);
procedure Susp_periodT(NoTask: integer);
procedure Susp_periodM(NoTask: integer);
procedure Timer_Off(NoTask: integer);
procedure GetShortTime(var h, m, s : word);
procedure GetShortDate(var a, m, d : word);
function  GetTimeStr: string;
function  GetDateStr: string;
procedure GetDateTime(var D: DateTime);
procedure SetDateTime(var D: DateTime);
procedure EndRealTime;
function  GetKbEnaFlag: byte;
procedure SetKbEnaFlag(NewFlag: word);
function  GetKey: word;
function  GetKeyTime(var TimeOut:boolean;
   TimeInSeg:word): word;
```

```
procedure PutKey(KeyCode: word);
function  KeyAvail: boolean;
procedure SetKbdTask(Task: word);
procedure ClearKbdTask;
procedure GetMem(var P; Size: Word);
procedure FreeMem(var P; Size: Word);
function  SendCom(var Buff; CommPort:word): integer;
function  ReceiveCom(var Buff; CommPort: word;
   Flush:boolean): integer;
function  OpenCom (Puerto, Divisor, Paridad, StopBits,
   WordSize: word): boolean;
procedure CloseCom(Puerto: word);
function  PortComAvail:integer;
procedure SetTimeOut(TimeChar, TimeMess: word);
function  CRC(var buffer): word;
function PrnWrite(S:string; Puerto:word): word;
function PrnWriteln(S:string; Puerto:word): word;
function PrnBinWrite(var Buf; NoBytes,
   Puerto:word): word;
function PrnAvail(Puerto:word): word;
procedure OpenPrn(BufferSize, Puerto: word);
procedure ClosePrn(Puerto: word);
procedure SetPrnTimeOut(TimeOut, Puerto: word);
function  DinSStack(Task:word): pointer;
implementation
```

LABORATORIOS

Se cuenta con laboratorios especializados para el desarrollo de la docencia, las investigaciones y el posgrado, entre ellos están: Difracción de Rayos X, Análisis Espectral de Emisión y Absorción, Mineragrafía, Petrografía, Fotogeología, Propiedades Físicas y Granulométricas, Mineralogía, Análisis de Agua, Análisis Fotocolorimétrico, Preparación de Muestras, Preparación Mecánica y Beneficio de Minerales, Hidrometalurgia, Pirometalurgia, Análisis Químico, Análisis Físico-Químico, Hidráulica, Automatización, Electrotecnia, Electrónica, Metalografía, Resistencia de Materiales, Mecánica de Rocas, Fotogrametría, Física de las Rocas, Modelación, Ventilación y Protección del Trabajo y Servicios de Computación.

MUSEO

El museo de Geología está estructurado en tres secciones principales: Mineralogía, Petrología y Paleontología.

La sección de Mineralogía expone muestras representativas de los principales grupos de minerales, entre ellos elementos nativos, sulfuros y compuestos similares, óxidos e hidróxidos, halogenuros, carbonatos, sulfatos, fosfatos, boratos y nitratos, vanadatos, arseniatos, walframatos y silicatos de diversos países y en una se exponen minerales de la región de Moa.

La sección de Petrología expone los principales grupos de rocas: igneas, sedimentarias y metamórficas. Una muestra importante lo constituye la asociación ofiolítica de la región Moa-Baracoa, por ser éste uno de los macizos ofiolíticos más grandes del mundo por el grado de aflorabilidad de las rocas ultramáficas. A partir de este sustrato rocoso se originan las cortezas de intemperismo ferroniquelíferas.

La sección de Paleontología expone un conjunto de fósiles que se disponen de forma consecutiva, de las épocas geológicas más viejas hasta el presente, apoyados con ilustraciones gráficas de algunos momentos que caracterizan un determinado período del desarrollo evolutivo, tanto de la flora como de la fauna existentes en el mismo.

Vol. XIII, No. 3, 1996

CONTROLADOR AUTOMÁTICO ON-LINE DE LA COMPOSICIÓN DE LICORES Y PULPAS CARBONATO AMONIACALES EN EL PROCESO DE SEPARACIÓN DE NÍQUEL Y COBALTO

Dr. Antonio Muñoz Moner
Ing. Angel O. Columbié Navarro
Ing. Daniel Guzmán del Río

Departamento de Electromecánica. Instituto Superior Minero Metalúrgico

RESUMEN:

Se muestran las particularidades de un controlador automático on-line para el análisis instrumental en líneas de flujos continuos de licores y pulpas carbonato-amoniacales que utiliza un sistema de toma de muestras especial acoplado a la línea tecnológica. El mismo se aplica en la industria del níquel y puede generalizarse a las industrias de materiales de construcción y biotecnológica.

ABSTRACT:

This paper shows the particularities of an automatic controller for instrumental analysis on-line in continuos stream flows of carbonate-ammonia liquors and pulps, which use a special sampler joined into the technological line. The controller was designed and built to be used in the nickel industry, but it also may be used in the construction material and biotechnological industries.

La separación de níquel y cobalto se lleva a cabo, mediante la tecnología basada en la precipitación a partir de licores carbonato-amoniacales del sulfuro de níquel y cobalto, proveniente de la planta de lixiviación y cuyo esquema de flujo tecnológico se muestra en la Figura 1, también, se puede observar la vista general de una instalación en la Foto 1. En el proceso se utiliza el hidrosulfuro de amonio NH_4HS como reactivo precipitador.

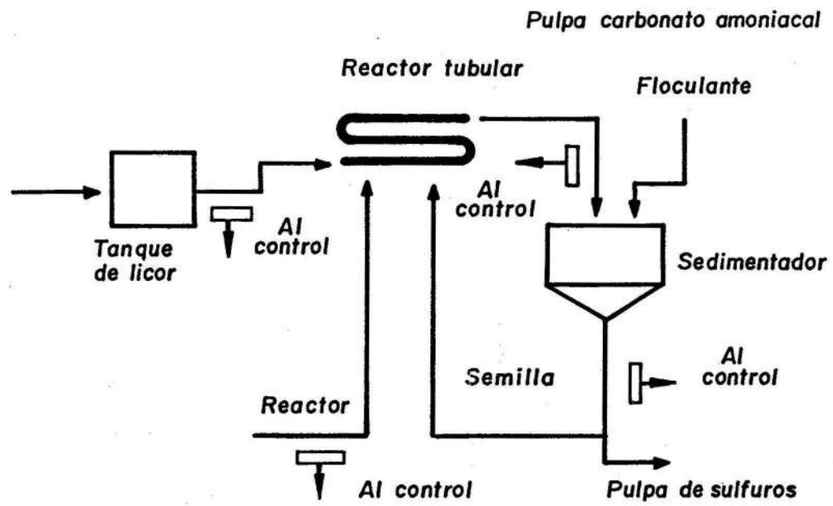

FIGURA 1. Esquema de flujo corto del proceso de separación de níquel y cobalto.