Revista INGENIERÍA UC
ISSN: 1316-6832
revistaing@uc.edu.ve
Universidad de Carabobo
Venezuela

Castellanos D., Jorge A.; Dorta F., René
Implementación de la interfaz para programación paralela de Modula-3 bajo el sistema operativo Windows
Revista INGENIERÍA UC, vol. 12, núm. 1, abril, 2005, pp. 48-55
Universidad de Carabobo
Valencia, Venezuela

REVISTA INGENIERÍA UC. Vol. 12, Nº 1, 48-55, 2005

Implementación de la interfaz para programación paralela de Modula-3 bajo el sistema operativo Windows

Jorge A. Castellanos D. [(1)]***, René Dorta F.*** [(2)]

[(1)] *Departamento de Computación, Facultad Experimental de Ciencias y Tecnología, Universidad de Carabobo, Valencia, Venezuela*

[(2)] *Departamento de Sistemas y Automática, Escuela de Ingeniería Eléctrica, Universidad de Carabobo, Valencia, Venezuela*

Email: jcasteld@uc.edu.ve; rene@uc.edu.ve

Resumen

Con el fin de facilitar el estudio práctico de la programación concurrente (basada en hilos) se ha implementado una interfaz para programación paralela la cual agrega al lenguaje de programación Component Pascal la posibilidad de ejecutar programas con múltiples hilos. Se desarrolló la interfaz para el lenguaje de programación Component Pascal, por sus características de sencillez para el aprendizaje y flexibilidad en el desarrollo de pequeños programas. Con el uso de este lenguaje, el esfuerzo del estudiante se concentra mayormente en el código del algoritmo y no en la interfaz de usuario. La interfaz para programación multihilos se complementó con un conjunto de programas de prueba que además de servir para el desarrollo, prueba y evaluación de la interfaz objeto de este trabajo, le permitirán al estudiante interesado desde un primer momento familiarizarse con la programación en múltiples hilos. La implementación de la interfaz se fundamentó en la versión existente para Modula-3, porque además de su probada capacidad para la programación de aplicaciones multihilos, su uso resulta sencillo para el interesado en programación multihilos.

Palabras clave: Programación multihilos, concurrencia, sistemas operativos.

Implementation of the Modula-3 interface for parallel programming under Windows operating system

Abstract

In order to facilitate the practical study of the concurrent programming (based on threads) a parallel programming interface has been implemented that adds to the Component Pascal programming language the possibility of executing programs with multiple threads. The interface for the programming language was developed in Component Pascal, because of its characteristics of simplicity for learning and flexibility in the development of small programs. Using this language, the effort of the student is concentrated mainly in the code of the algorithm and not in the user interface. The interface for multithreads programming was complemented with a set of testing programs that besides serving for the development, test and evaluation of the interface object of this work, will allow the student that is interested to become familiar from the start with the programming based on multiple threads. The implementation of the interface is based on the existing one for Modula-3, because in addition to its proven capacity for programming multithread applications, its use is simple for those interested in multithread programming.

Keywords: Multithread programming, concurrence, operating systems.

1. INTRODUCCIÓN

En el programa de estudios de Licenciatura en Computación de la Facultad de Ciencias y Tecnología de la Universidad de Carabobo, el tratamiento del problema de la concurrencia se ha realizado tradicionalmente mediante el estudio teórico de los problemas clásicos de programación concurrentes en la asignatura Sistemas Operativos.

Debido a la importancia que ha tomado en los últimos tiempos la programación paralela y concurrente, el Departamento de Computación de la Facultad de Ciencias y Tecnología ofrece materias electivas como Programación Paralela y Sistemas de Tiempo Real, que permiten al estudiante de computación complementar su formación en esta área.

La dificultad que experimenta el estudiante que toma estos cursos sin ninguna experiencia práctica previa en programación concurrente, es conocida debido a que tradicionalmente se trabaja desde un comienzo con lenguajes de programación como C y C++, los cuales no ofrecen una interfaz amigable para la construcción y prueba de programas concurrentes y paralelos.

En este trabajo se ha desarrollado una interfaz sencilla y amigable de programación paralela para el Lenguaje de programación Component Pascal que le permitirá al estudiante de computación, desde el momento que cursa la materia Sistemas Operativos, obtener un conocimiento teórico y práctico de la programación concurrente y sus problemas más conocidos. En este estudio también se incluyen ejemplos clásicos completos de programación concurrente que le permitirán al estudiante que recién se inicia en este campo, conocer las principales dificultades y aprender a usar los recursos de la programación concurrente (regiones críticas y variables de condición) mediante el ejemplo.

2. MARCO TEÓRICO

2.1. Programación Multihilos

Cuando se propone el desarrollo de una librería para hacer paralelismo debemos hacer referencia a la unidad más básica para la ejecución de programas paralelos y/o concurrentes: esto es lo que denominamos un hilo, trenza de ejecución, proceso ligero ó por su nombre original en el idioma inglés “thread”.

Un “hilo” es un flujo secuencial simple de control dentro de un programa. Como se ve en la Figura 1, un programa puede estar conformado por un único “hilo” (programa secuencial).

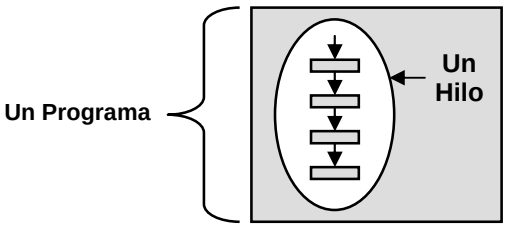

Figura 1. Programa secuencial ó de un solo hilo.

Pero si hablamos de paralelismo y/o concurrencia tenemos que pensar en un programa compuesto por al menos dos hilos (ver Figura 2). A un programa compuesto por varios hilos también se le denomina multihilos (multithread).

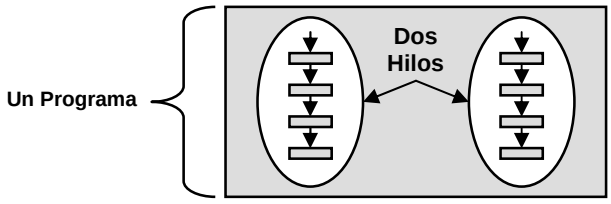

Figura 2. Programa paralelo con dos hilos.

2.2. Modula-3

Modula-3 es un moderno lenguaje de programación desarrollado como un proyecto conjunto entre Digital Equipment Corporation (DEC) y Olivetti. El concepto más importante en la familia de lenguajes Modula, es el concepto de Módulo (de ahí su nombre). Los módulos permiten construir jerarquías de abstracciones de datos. El módulo es a la vez una construcción que permite mejorar la estructura de los programas, en el se agrupan procedimientos, de la misma manera que los procedimientos agrupan instrucciones. Los procedimientos dentro de un mismo módulo pueden compartir datos, que son sin embargo privados para ese módulo.

Los módulos se separan en dos partes: la parte de la definición de la interfaz y la parte de la implementación. La parte de la interfaz es la única que necesita conocerse en detalle para aprovechar los servicios ofrecidos por la implementación [1].

2.3. Interfaz para Programación Paralela del lenguaje de programación Modula-3

Modula-3 ofrece un conjunto simple de primitivas para programación concurrente. Este conjunto incluye tipos y procedimientos para la creación de hilos y para sincronizar el acceso a memoria compartida (para los hilos dentro de un mismo proceso). Este conjunto de primitivas viene en un módulo llamado “Interfaz Estándar para Hilos”, esta interfaz (ver Figura 3) se estudiará con mas detalle en secciones posteriores [1].

```
INTERFACE Thread;      (* Copyright © 1989, Digital Equipment Corporation *)

CONST defaultStackSize = 3000;     (* en palabras *)
TYPE
T <: ROOT;                    (* Asidero (handle) para un hilo *)
Mutex = MUTEX;              (* Trancas (locks) para exclusión mutua *)
Condition <: ROOT;                        (* Cola de hilos en espera *)
Closure = BRANDED "Thread.Closure" OBJECT METHODS
Apply (): REFANY RAISES {} END;
(* También es posible indicar el tamaño del stack para un hilo: *)
SizedClosure = Closure BRANDED "Thread.SizedClosure" OBJECT
stackSize: CARDINAL:= DefautlStackSize END;

PROCEDURE Fork (cl: Closure): T;
(* Retorna el asidero a un hilo recién creado, que ejecuta el cl.apply () *)

PROCEDURE Join (t: T): REFANY;
(* Espera hasta que t termine y retorna su resultado. Es un error de
ejecución llamar más de una vez a este procedure para un mismo hilo t *)

PROCEDURE Wait (m: Mutex; c: Condition);
(* El hilo que llama debe poseer a m. Atómicamente, lo suelta, espera en
c, luego espera a que m esté libre, lo toma y retorna *)

PROCEDURE Acquire (m: Mutex); (* Espera a que m esté libre y lo toma *)

PROCEDURE Release (m: Mutex); (* Suelta m. El hilo que llama debe poseerlo *)

PROCEDURE Broadcast (c: Condition);
(* Todos los hilos que esperaban en c, dejan de esperar y son elegibles
para correr. Si no hay hilos esperando en c, se comporta como un no-op *)

PROCEDURE Signal (c: Condition);
(* Uno o mas hilos que esperaban en c, dejan de esperar y son elegibles
para correr. Si no hay hilos esperando en c, se comporta como un no-op *)

PROCEDURE Self (): T;         (* Retorna el asidero del hilo que llama *)

END Thread.
```

Figura 3. Interfaz Estándar para Hilos de Modula-3. Nelson (1991).

2.4. Component Pascal

Component Pascal es un lenguaje de programación desarrollado por Oberon Microsystems como un refinamiento del lenguaje Oberon-2. La extensión de tipos hace que Component Pascal sea un lenguaje orientado a objetos. Un objeto es una variable de tipo abstracto que consiste de datos privados (su estado) y procedimientos que operan sobre estos datos. Los tipos de datos abstractos se declaran como "records" extensibles. Component Pascal toma en consideración la mayoría de términos de los lenguajes orientados a objetos, estableciendo el vocabulario de lenguajes imperativos para minimizar el número de nociones de conceptos similares. La seguridad completa sobre tipos y la necesidad de un modelo de objeto dinámico hacen del Component Pascal un lenguaje orientado a componentes [2][3]. Este lenguaje de Programación puede descargarse libremente desde Internet.

2.5. Sincronización de hilos

Si dos o más hilos necesitan acceder a datos comunes, entonces se necesita usar alguna forma de sincronización para evitar que más de una de los múltiples hilos acceda los datos al mismo tiempo [5]. Es decir, un hilo debe esperar por otro para completar la utilización de los recursos compartidos antes de continuar su procesamiento. Debe controlarse así, el acceso a los datos compartidos y los recursos compartidos [6].

2.5.1. Objetos de exclusión

Los objetos de exclusión mantienen un mecanismo que permite a un solo hilo "poseer" un objeto de exclusión y por consiguiente cualquier código protegido por un objeto de exclusión sólo puede ejecutarse en su totalidad por un solo hilo. Una vez que un hilo posee un objeto de exclusión, el tiene la responsabilidad de liberar la propiedad del objeto. Sólo el hilo que posee el objeto de exclusión puede soltarlo. Hay varias formas de objetos, los más resaltantes son: la Sección Crítica, el Mutex, el Spinlock, el Semáforo, el Monitor, los bloqueos de lectura/escritura [7].

La Sección Crítica: es una secuencia de instrucciones con un comienzo y un final claramente marcados que, generalmente delimita la actualización de una o más variables compartidas. Cuando un hilo entra en una sección crítica, debe ejecutar todas las instrucciones incluidas en ella antes de que se pueda permitir a cualquier otro hilo entrar a la misma sección crítica.

Sólo el hilo que ejecuta la sección crítica tiene permitido el acceso a la variable compartida; los hilos restantes deben tenerlo prohibido hasta la terminación de la sección crítica. A esto se le suele denominar exclusión mutua, porque un solo hilo excluye temporalmente a todos los demás de utilizar un recurso compartido con el fin de asegurar la integridad del sistema [8].

El Mutex: es una bandera binaria asociada con una cola (posiblemente vacía) de hilos en espera. El Mutex puede estar bloqueado (lock) o libre (free). En el momento de la inicialización, el Mutex se pone en el estado libre (free). Los Mutex tienen dos operaciones:

- Adquisición: Si el Mutex está libre, mediante esta operación es bloqueado por un hilo que lo invoca, y el hilo continúa su ejecución. Si el Mutex ya ha sido adquirido (por otro hilo) en el momento que se invoca la operación, el hilo que lo trata de adquirir es bloqueado (suspendido) y es colocado en la cola de espera (esperando a que el Mutex esté libre).
- Liberación: Esta operación libera el Mutex, regresándolo a su estado inicial (libre). Solo puede liberar un Mutex, el mismo hilo que ya lo posee. Si en el momento de liberar el Mutex, hay hilos bloqueados esperando por el Mutex, solo se selecciona una de ellas (de la cola); el hilo liberado arranca y adquiere entonces el Mutex.

2.5.2. Mecanismos de sincronización

Los mecanismos de sincronización son objetos que permiten la sincronización entre hilos de forma que se puedan coordinar acciones que permitan sincronizar la ejecución de hilos en el tiempo. Entre ellos se encuentran: los eventos, las señales, las variables de condición y las barreras.

Los eventos: Los eventos proveen un mecanismo de señalar un "evento"'. Un evento tiene dos estados: señalado y no-señalado. Cuando un evento es señalado, a cualquier hilo suspendido que espera por el evento, se le permite continuar.

Las variables de condición son una combinación de una sección crítica y una clase de evento. Cuando se señala la condición, se pueden arrancar uno o todos los hilos que están esperando por la condición. Se debe poseer la sección crítica para entrar a esperar la condición. Cuando se entra a esperar la condición el hilo se bloquea y libera la sección crítica. Cuando se retorna de la espera por la condición, el hilo debe (de nuevo) adquirir la posesión de la sección crítica. Las variables de condición no son contadores. No acumulan señales para su uso posterior, como lo hacen los semáforos.

2.6. Hilos en Windows

En Windows 2000, un hilo es una tarea que puede correr independientemente de otras partes del programa. Un hilo pertenece a un proceso y obtiene su propio quantum (mínima cantidad de tiempo de procesador que el planificador del sistema asigna a un hilo) del procesador [9].

Las aplicaciones basadas en Windows 2000 pueden crear múltiples hilos dentro de un proceso dado. Mediante la creación de múltiples hilos, la aplicación puede lograr algún procesamiento en el trasfondo (backgroung), tales como cálculos para permitirle al programa correr más rápido. Mientras el hilo está en ejecución el usuario puede continuar interactuando con el programa [10].

2.6.1. Sincronización de la ejecución de múltiples hilos

Para evitar condiciones de carrera (race conditions) e inter-bloqueos (deadlocks), es necesario sincronizar adecuadamente el acceso a recursos compartidos por múltiples hilos. La sincronización también es necesaria para asegurar que el código interdependiente se ejecutan en la secuencia apropiada.

En Windows hay una diversidad de objetos cuyas asas (handles) pueden usarse para sincronizar múltiples hilos. Esos objetos incluyen: Buffers de entrada de consola, Eventos, Mutexes, Procesos, Semáforos, Hilos y Temporizadores (timers).

El estado actual (presente) de esos objetos es: Señalado ó No señalado. Cuando el usuario especifica un asa (handle) a cualquiera de esos objetos en una llamada a una función wait, se bloquea la ejecución del hilo que hace la llamada hasta que el estado del objeto especificado en la función wait sea de nuevo Señalado.

El estado de las asas (handles) de los objetos Process y Thread es Señalado cuando el objeto Process ó Thread termina. Esto permite a un hilo (padre) por ejemplo, crear un hilo hijo y luego bloquear su propia ejecución hasta que el nuevo hilo haya terminado.

Hay objetos que son útiles para la protección de recursos compartidos. Por ejemplo, si se quiere que un grupo de hilos puedan compartir el uso de un recurso entonces ellas deben conocer el asa (handle) de un objeto Mutex iniciado previamente. Así antes de acceder al recurso compartido, cada uno de los hilos que va a acceder el recurso debe llamar previamente una de las funciones wait, y de esta manera el hilo queda en espera hasta que el estado del Mutex sea Señalado. Cuando el estado del objeto Mutex es Señalado, solamente uno de los hilos en espera se libera para acceder al recurso. Entonces el estado del Mutex pasa inmediatamente a No señalado de forma que los demás hilos en espera permanecen bloqueados. Cuando el hilo ha finalizado el acceso al recurso compartido, el mismo hilo debe establecer el estado del Mutex en Señalado para permitir que otro hilo pueda acceder el recurso compartido. Este mecanismo es muy seguro para garantizar la exclusión a nivel de procesos, pero tiene la desventaja de ser lento para trabajar la sincronización a nivel de hilos, por ello no se usó para este trabajo.

Los objetos Critical Section (sección crítica) ofrecen una forma más eficiente de sincronización que los objetos Mutex [11]. Una sección crítica se usa similar a un objeto Mutex para permitir que solo un hilo pueda usar un recurso protegido al mismo tiempo. Un hilo puede usar la función Enter Critical Section para solicitar la propiedad de una sección crítica. Si la sección crítica ya es propiedad de otro hilo, el hilo que la solicita se bloquea. Si se quiere solicitar la propiedad de una sección crítica sin bloquear el hilo (en el caso de fallar en su obtención) se puede usar la función TryEnterCriticalSection. La ejecución de otros hilos no se afecta a menos que ellas intenten entrar a la misma sección crítica.

3. METODOLOGÍA

La investigación del objeto de estudio es del tipo tecnológico. En este trabajo se estudian tecnologías en Sistemas Operativos e Ingeniería de Software y se desarrolla una nueva implementación de una interfaz de software, bajo el sistema operativo Windows y para el lenguaje de programación Component Pascal.

La metodología usada es como sigue: se revisa la literatura existente, se realiza un estudio previo de factibilidad, se implementa la nueva interfaz, se realizan pruebas y los resultados se comparan contra otras interfaces similares; con esto se proponen cambios, se implementan los cambios y se realizan nuevas pruebas.

3.1. La Interfaz Multihilos Para Component Pascal

El lenguaje de programación Component Pascal, no posee una interfaz para programación paralela

```
DEFINITION UcThreads;
    IMPORT UcQueues, Dialog;
    CONST
        max = 32;
        version = "3.4";
    TYPE
        Thread = POINTER TO EXTENSIBLE RECORD
            slot-, id-: INTEGER;
            par-: INTEGER
        END;
        Body = PROCEDURE (i: INTEGER);
        Condition = EXTENSIBLE RECORD (UcQueues.Queue)
            (VAR c: Condition) Broadcast, NEW;
            (VAR c: Condition) Signal, NEW;
            (VAR c: Condition) Wait (VAR m: Mutex), NEW
        END;
        Mutex = EXTENSIBLE RECORD
            r: POINTER TO RECORD
            held-: BOOLEAN    END;
            (VAR m: Mutex) Acquire, NEW;
            (VAR m: Mutex) Release, NEW
        END;

    VAR n-: INTEGER; trace: BOOLEAN;

    PROCEDURE Assert (b: BOOLEAN; IN msg: ARRAY OF CHAR);
    PROCEDURE Fork (proc: Body; par: INTEGER);
    PROCEDURE Info;
    PROCEDURE Kill;
    PROCEDURE Pause (ms: INTEGER);
    PROCEDURE Self (): Thread;
    PROCEDURE State;

END UcThreads.
```

Figura 4. Interfaz UcThreads para Component Pascal.

propia que permita al usuario tener facilidades de concurrencia a través de la creación y manipulación de hilos. En su lugar posee un soporte para la ejecución de tareas de segundo plano (background) que brinda al usuario la posibilidad de ejecutar tareas en forma programada. El soporte de ejecución de tareas de segundo plano está disponible importando el componente de biblioteca llamado Services.

Con el trabajo actual, las facilidades de concurrencia mediante hilos estarán disponibles al usuario a través de un módulo de biblioteca que denominaremos UcThreads (Uc: Universidad de Carabobo; Threads: término en inglés universalmente aceptado para referirse a hilos de programa). Este módulo (ver Figura 4) se desarrolló en conjunto con un paquete de software compuesto por módulos de soporte (UcQueues, UcBuffers, UcGlobalLock, UcIOx), módulos complementarios (UcSemaphores, UcVectores) y módulos de aplicaciones de prueba (Reloj, UcFern, UcTSP, UcSuma, UcTSPbbc, UcHesselink, UcPqueues). Todos ellos van precedidos del prefijo Uc que se escogió como nombre del subsistema, un requisito de Component Pascal para alojar un conjunto de módulos de un mismo proyecto.

Los principios de diseño usados para este desarrollo fueron: seguridad, facilidad de uso y bajo consumo de recursos. Debido a: afinidad entre Component Pascal y Modula-3, requisitos de seguridad, bajo consumo de recursos y experiencia previa con la interfaz de modula-3 adquirida por el tutor del trabajo [1], se tomó como base la interfaz para programación paralela de Modula-3, desarrollada previamente para el sistema operativo Mach que trabajos de investigación anteriores [12] del autor del trabajo, y motivado por

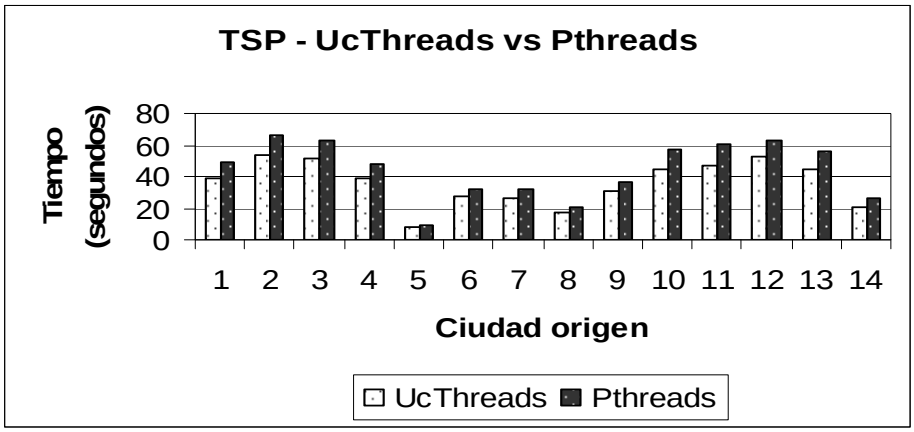

Figura 5 Tiempos de ejecución para el programa TSP.

búsqueda de facilidades de utilización, se incorporaron ó previeron para futuros trabajos algunas características nativas en las interfaces de programación paralela Pthreads de "C", Multithread de Java y clase Thread de Borland C++ Builder y Borland Delphi.

Del mismo modo que Modula-3 [13], todos los hilos en un programa multihilos de Component Pascal usando la interfaz UcThreads residirán en un mismo espacio de dirección, lo cual significa que compartirán el mismo código y las variables globales.

Todos los tipos y las operaciones que soportarán la programación concurrente estarán disponibles a través de la nueva interfaz UcThreads.

4. RESULTADOS

Los resultados que se presentan se obtuvieron con un computador tipo PC con procesador Pentium 4, 1.6GHz, 512Mb de memoria ram. Para poder corroborar los resultados se usaron como base el sistema operativo Windows 2000 (service pack 4) para las pruebas en Component Pascal y el sistema operativo Debian/GNU Linux 3.1 para las pruebas con lenguaje "C" y la interfaz Pthreads.

Aún cuando se desarrollaron un conjunto de problemas de prueba que ayudan al interesado a conocer el funcionamiento y forma de programar usando esta nueva interfaz, se presenta como referencia un programa de agente viajero que permite obtener resultados en tiempo de ejecución con el objeto de conocer el desempeño de la interfaz. Se advierte que debido a la disimilitud de las comparaciones (diferentes lenguajes de programación sobre diferentes sistemas operativos) estos resultados solo se pueden tomar como una referencia.

Este problema se trató ampliamente en otro trabajo anterior [12] y sus resultados sirven para verificar la movilidad (liveness) de la aplicación multihilo ante la presencia de los mecanismos de sincronización. Es de resaltar que los resultados obtenidos en Component Pascal son similares a los obtenidos en Lenguaje "C" sin usar optimizaciones de código; esto significa que esta aplicación multihilos del agente viajero en Component Pascal se ejecuta con buenos tiempos.

En la Figura 5 hay un gráfico de barras que permite apreciar que los tiempos son comparables. En esta figura las barras claras representan los tiempos obtenidos para Component Pascal y las barras oscuras son los tiempos para la aplicación en lenguaje "C" (Pthreads).

5. CONCLUSIONES

Se cumple con el propósito principal del trabajo al terminar con una primera versión de una interfaz para programación paralela con todas las funciones básicas necesarias que permitirán a estudiante la implementación de problemas clásicos de programación

concurrente basada en hilos.

Los resultados obtenidos en tiempo de ejecución para los programas de prueba multihilos en Component Pascal usando la interfaz para programación paralela, fueron similares a los registrados para versiones de los programas de prueba codificadas en el lenguaje de programación "C", usando la interfaz Pthreads.

Aunque los resultados son la experiencia de varios años trabajando en el desarrollo de la presente interfaz, se deja abierta la posibilidad de descubrir nuevas condiciones "excepcionales", porque para el momento de la presentación de este trabajo aún no se contaba con el código fuente de algunos módulos del compilador de Component Pascal, que permitieran incluir mecanismos de "debug" (prueba de validez de sentencias del programa multihilos) para prever la aparición de dichas condiciones.

6. RECOMENDACIONES

Los ejemplos presentados son sólo una muestra de un conjunto de problemas basados en hilos que se desarrollaron en este trabajo y servirán para el lector interesado en utilizar esta interfaz ó de continuar su desarrollo. Se recomiendan como un punto de partida para futuros trabajos en esta área de investigación.

Para el desarrollo de una interfaz de este tipo es importante contar con la información necesaria de operación del sistema de "debug" (procesamiento interno de condiciones de error del compilador) y el "garbage collector" (recolector de basura), ó en su lugar con los códigos fuente de forma que sea posible el manejo de condiciones excepcionales por parte del soporte multihilo de la interfaz.

Aún cuando el código de la interfaz y de los ejemplos desarrollados no se presenta de forma explícita (por limitaciones de espacio), se pueden solicitar a los autores de este trabajo, vía correo electrónico.

REFERENCIAS

[1] DORTA, René. (1998). Implementación de la interfaz para programación paralela de Modula-3 bajo el sistema operativo Mach. Universidad de Carabobo. Escuela de Ingeniería Eléctrica. Trabajo de Ascenso.

[2] Oberon Mycrosystems. (1997). Component Pascal Language Report. http://www.oberon.ch/docu/language_report.html

[3] WARFORD, J. Stanley. (1996). Programming with BlackBox. Pepperdine University. Revised: Aug 10,2000.

[4] NELSON, Greg. (1991). System Programming with Modula-3. Prentice Hall.

[5] BLACK, Normal. (2002). Thread synchronization. http://www.home.ix.netcom.com/~stonybrk/env/index33.htm.

[6] FOWLKES, Edward y SHIDA, Takashi. (1996). Multithread programming. http://www.userpages.umbc.edu/~schmitt/331F96/tshida1/thread.html

[7] TANENBAUM, Andrew S. (1995). Sistemas Operativos Distribuidos. Primera edición. Prentice Hall Hispanoamericana S. A.

[8] SILBERSCHATZ, A, PETERSON, J. y GALVIN, P. (1994). Sistemas Operativos - Conceptos Fundamentales. Tercera Edición. Addison-Wesley Iberoamericana.

[9] GORDANA, Dodig - Cmkovic. (2001). Threads in Windows NT.
e-mail: gordana.dodig-cmkovic@mdh.se

[10] Microsoft Developer Network. (2004). MSDN Library. Processes and Threads. Microsoft Corporation. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/processes_and_threads.asp

[11] PIETREK, Matt y OESTERLUND, Russ. (2003). Break free of Code Deadlocks in Critical Sections Under Windows. MSDN Magazine. http://msdn.microsoft.com/msdnmag/issues/03/12/CriticalSections/print.asp

[12] CASTELLANOS, Jorge. (2003). Estudio comparativo de la interfaz Pthreads del lenguaje de programación "C" y el soporte para programación multihilo de Java. FACYT-UC. Departamento de Computación. Trabajo de Ascenso.

[13] HARBISON, Samuel P. (1992). Modula-3. Prentice Hall, Englewood Clips, New Yerse