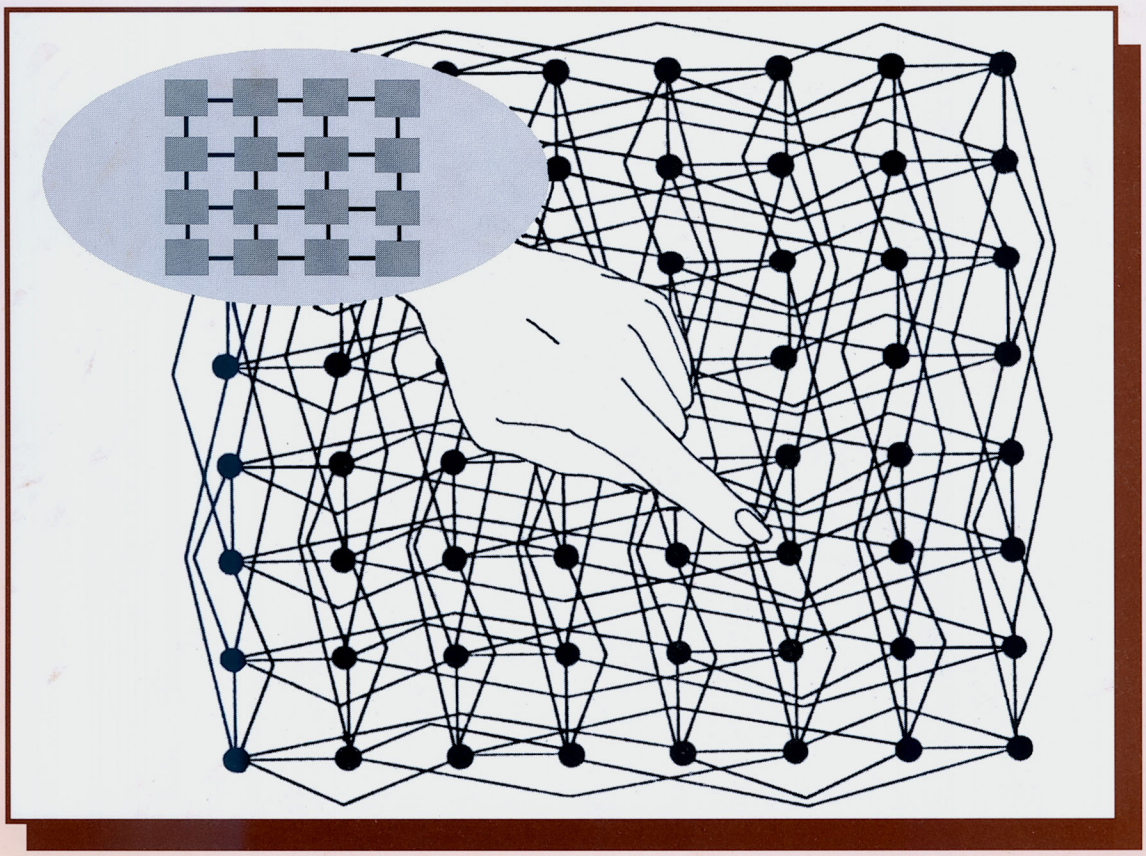
Ingeniería

Revista de la Universidad de Costa Rica
Julio/Diciembre 1996 VOLUMEN 6 Nº 2



INGENIERIA

Revista Semestral de la Universidad de Costa Rica
Volumen 6, Julio/Diciembre 1996 Número 2

CORRESPONDENCIA Y SUSCRIPCIONES

Editorial de la Universidad de Costa Rica
Apartado Postal 75
2060 Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

CANJES

Universidad de Costa Rica
Sistema de Bibliotecas, Documentación e Información
Unidad de Selección y Aquisiciones-CANJE
Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

Suscripción anual:
Costa Rica: ¢ 1 000,00
Otros países: US $ 25,00

Número suelto:
Costa Rica: ¢ 750,00
Otros países: $ 15,00

Edición aprobada por la Comisión Editorial de la Universidad de Costa Rica

Ciudad Universitaria Rodrigo Facio
San José, Costa Rica.

Revisión Filológica: *Lorena Rodríguez*

Diagramación:
José R. Argüello V.

Control de Calidad:
Unidad Diseño Revistas. Oficina de Publicaciones

Impreso en la Oficina de Publicaciones
de la Universidad de Costa Rica

Revista
620.005
I-46i Ingeniería / Universidad de Costa Rica. ——
Vol. I, no. 1 (ene./jun. 1991)— . —— San José, C. R. : Editorial de la Universidad de Costa Rica, 1991— (Oficina de Publicaciones de la Universidad de Costa Rica)
v. : il

Semestral.

1. Ingeniería - Publicaciones periódicas.

CCC/BUCR—250

Ingeniería 6 (2): 107-114 1996 San José, Costa Rica

CREACIÓN DE UN PROTOTIPO PARA EL AMBIENTE DE PROGRAMACIÓN *C-LINDA* BAJO UNIX

Juan José Vargas[1]
Elpidio Calderón[2]

RESUMEN

Enmarcado en un proyecto de investigación, se llevó a cabo el trabajo de graduación de Elpidio Calderón, dirigido por el prof. Juan José Vargas, en el cual se programó un prototipo para la creación del ambiente de programación concurrente *C-LINDA*. Este ambiente consiste en una extensión del lenguaje C estándar, agregando los operadores *in*, *out*, *inp*, *rd*, *rdp*, y *eval*. Estos operadores se aplican a n-*tuples* ordenados de datos, de dimensión y tipos variables, los cuales se almacenan en un espacio compartido de *tuples* (*ET*). Este espacio es en realidad una memoria asociativa, puesto que los *tuples* se localizan por su contendido. Las operaciones sobre ellos son atómicas, propiedad que sirve para utilizarlos como semáforos. De esta manera, *C-LINDA* facilita y simplifica el uso de estructuras compartidas por procesos concurrentes. Como parte de los ejemplos de prueba, se programaron los problemas clásicos de coordinación de procesos: filósofos comensales, lectores/escritores, y multiplicación de matrices.

SUMMARY

Under a research project, the graduation work of Elpidio Calderón was carried out, guided by the professor Juan José Vargas, and in which a prototype was programmed to create the concurrent programming environment called *C-LINDA*. This environment consists of an extension of the language C standard, plus the operators *in*, *out*, *inp*, *rd*, *rdp*, y *eval*. Those operators are applied to ordered data n-*tuples*, with variable dimension and types., which are stored in a shared space of *tuples* (*ET*). This space is a real associative memory since the tuples are located by content. Operator over them are atomic, property that allows them to be use as semaphores. In such a way, *C-LINDA* facilitates and simplifies the use of shared structures by concurrent processes. As part of the testing cases, there were programmed classical problems of process synchronization: dining philosophers, readers-writers and matrix multiplication.

1.- INTRODUCCIÓN

La programación concurrente se puede definir como la escritura de programas que tienen varias partes en ejecución en un mismo período de tiempo. Para lograr esto no es necesaria la presencia de varios procesadores en la computadora. Un concepto relacionado con este tipo de programación es el procesamiento paralelo, el cual es una forma eficaz de procesamiento de información que favorece el buen uso de los sucesos concurrentes en el proceso de computación. Los sucesos paralelos son los que pueden producirse en diferentes recursos durante el mismo intervalo de tiempo. Existe paralelismo en áreas tales como: redes de computadoras, sistemas distribuidos y sistemas de tiempo real.

Hoy en día se hace necesario el diseño de aplicaciones que aprovechen la capacidad de procesamiento paralelo de las computadoras multiprocesadoras, así como el paralelismo existente en casi todos los algoritmos. Para el aprovechamiento del paralelismo se necesita que los lenguajes de programación tengan la capacidad de poder especificar la ejecución concurrente de dos o más actividades. Sin embargo, estos lenguajes, en su mayoría, ofrecen una ayuda limitada en cuanto a facilidades de concurrencia. Por ello, precisamente, es importante la investigación y el desarrollo en el campo de la programación concurrente.

[1] M.Sc.,. Prof. Comp. e Informática, Fac. Ing.; UCR.
[2] Lic. Comp. e Informática; UCR.

Existen varios enfoques para el establecimiento de lenguajes de programación concurrente; algunos de ellos son: la programación lógica concurrente, programación concurrente orientada a objetos, intercambio de mensajes, sistemas funcionales de programación y el enfoque de *LINDA*, cuya implantación fue el objeto de un proyecto de investigación y un trabajo final de graduación, del cual en este artículo se presenta un resumen.

Se espera que en el presente y en el futuro proliferen lenguajes de programación concurrente como *LINDA*, orientados al aprovechamiento del paralelismo en los algoritmos, especialmente cuando se ejecutan en máquinas multiprocesadoras. En Costa Rica, es la primera vez que este ambiente se implanta.

Las principales características de *LINDA* son su simplicidad y capacidad de proveer elegantes soluciones a problemas de programación concurrente.

1.1 Ventajas de *LINDA*

Según los autores Carriero y Gelernter [CG-89], *LINDA* es un lenguaje de programación más elegante, poderoso y simple que aquellos lenguajes basados en otros enfoques de programación paralela: intercambio de mensajes, pogramación concurrente orientada a objetos, lenguajes concurrentes de programación lógica y sistemas de programación funcionales. *LINDA* es un modelo de creación y coordinación de procesos ortogonal al lenguaje base al que se le agrega; sus ventajas se deben a la propiedad de comunicación generativa, descrita a continuación.

El modelo *LINDA* se basa en la comunicación generativa, esto es que si dos procesos necesitan comunicarse, en vez de intercambiar mensajes o compartir una variable, el problema lo resuelven mediante *tuples*. El proceso transmisor genera un tuple con los datos a enviar y lo pone en el *ET*; el proceso receptor lee del *ET* el tuple deseado. La creación de procesos es manejada de manera similar. Para crear un proceso concurrente, se genera un tuple activo en el *ET*.

Empleando comunicación generativa no se necesita que el proceso emisor conozca por anticipado cuántos o cuáles procesos destinatarios van a leer el mensaje. Contrariamente, en el sistema de intercambio de mensajes, un proceso no puede enviar mensajes si antes no conoce quiénes son sus receptores. Ver, por ejemplo, [CG-89] para más detalles sobre el sistema de intercambio de mensajes.

Los procesos concurrentes comúnmente utilizan el mecanismo que en sistemas operativos se conoce como "monitor" para la protección de sus estructuras contra accesos múltiples. Sólo un proceso puede estar activo dentro de las rutinas protegidas por el monitor en un tiempo dado. *LINDA* supera los monitores en simplicidad y flexibilidad, a la vez que unifica dentro del contexto de operaciones del *ET* la creación de estructuras, las variables de estado compartidas del monitor y los mecanismos de señales y condición de cola. Los monitores no son tan flexibles como los *tuples* para la construcción de estructuras de datos distribuidas.

El código generado en *LINDA* es más fácil de entender y más flexible que el de la programación lógica concurrente. Lo anterior es observable al programar el modelo cliente servidor y el conocido problema de los filósofos comensales.

Las operaciones de *LINDA* ya han sido agregadas en otros países a lenguajes de programación como *C* y *FORTRAN*. *LINDA* corre en un amplio rango de máquinas paralelas y ha tenido éxito en computadoras de memoria compartida y de memoria distribuida. Algunas de estas máquinas de memoria distribuida *MIMD* son *Intel IPS/2*, *S/net*, sistemas distribuidos basados en *Vax/VMS* y sistemas basados en *transputers*, como se mencionan en [SS-93]. Es debido a que el modelo *LINDA* se ajusta bien a la arquitectura de

multicomputadoras de memoria compartida que sus implantaciones en dichas computadoras han sido altamente exitosas. Las aplicaciones desarrolladas han mostrado un buen rendimiento.

2.- DESCRIPCIÓN DE *C-LINDA*

LINDA es un lenguaje desarrollado en la Universidad de Yale y marca registrada de Scientific Computing Associates, Inc. Es un lenguaje de programación concurrente basado en un espacio de *tuples* (*ET*). Tiene operadores que proveen facilidades de programación concurrente y se agrega a lenguajes de programación huéspedes produciendo nuevos ambientes de programación, tales como *PASCAL-LINDA*, *FORTRAN-LINDA*, etc.

C-LINDA es un ambiente de programación resultante de la mezcla de los lenguajes de programación *C* y *LINDA*, que conjuga el poder del lenguaje *C* con la capacidad de programación concurrente de *LINDA*. Para una comprensión de *C-LINDA*, es necesario definir los siguientes conceptos generales.

2.1.- *Tuples*, operadores, espacio de *tuples* (*ET*)

Se define tuple como una colección ordenada de campos. Cada uno de los n campos de un n-tuple tiene un tipo asociado que debe ser alguno de los tipos definidos en el lenguaje base. De esta manera en *C-LINDA* se trabaja con los campos de tipo elemental de C: *int*, *float*, *char* y *double*. Los operadores de *LINDA* son: *out*, *in*, *rd*, *inp*, *rdp* y *eval*, y sus significados son los siguientes:

out: inserta un tuple en el *ET*, no bloquea el proceso solicitante.

in: lee y remueve el tuple del *ET*, bloquea el proceso solicitante en caso de que el tuple no se encuentre disponible.

rd: lee el tuple del *ET*, bloquea el proceso solicitante en caso de que el tuple no se encuentre disponible.

inp: lee y remueve el tuple del *ET*, no espera a que el tuple esté disponible, es decir, no bloquea el proceso solicitante.

rdp: lee el tuple del *ET*, no bloquea el proceso solicitante en caso de que el tuple pedido no se encuentre disponible.

eval: crea un nuevo proceso que se ejecuta en forma concurrente con el actual, evaluando las funciones que se encuentran en el n-tuple. De esta manera, *eval* crea un n-tuple activo que pasa al estado pasivo una vez que hayan sido evaluadas las expresiones del caso.

Un tuple se encuentra en estado activo cuando tiene alguno de sus componentes sin calcular. Contrariamente, un tuple está pasivo cuando todos sus componentes ya han sido calculados. Los operadores *in*, *rd*, *inp* y *rdp* sólo pueden leer *tuples* pasivos.

El *ET* es el lugar donde se almacenan y recuperan los *tuples*. Es una memoria asociativa compartida, en la que cualquier tuple puede estar en estado activo o pasivo. Los *tuples* pueden usarse fácilmente para construir semáforos, comunicar procesos, almacenar estructuras de datos, etc.

Se le llama memoria asociativa porque sus *tuples* son accesados por contenido y no por dirección. Para recuperar un tuple, se utiliza un marco o plantilla, indicando de esa manera el número de campos, sus tipos y tal vez algunos cuantos valores. Este marco es comparado contra todos los *tuples* pasivos del *ET*, y si alguno coincide con la descripción dada, el tuple es recuperado. La correspondencia entre el marco y el tuple del *ET* se da si concuerdan en tipo, aridad de campos, y valores especificados en el marco.

El *ET* es compartido porque varios procesos pueden estar trabajando a la vez en él. Si varios *tuples* coinciden para una operación de lectura, cualquiera de ellos puede ser recuperado en forma no determinística. Es importante destacar

que las operaciones de lectura *in* e *inp* en el *ET* poseen exclusividad mutua para el tuple seleccionado, esto es, si varios procesos ejecutan *in* o *inp* a la vez hacia el mismo tuple, únicamente uno de ellos podrá leerlo. Si hubiesen varios *tuples* idénticos, cada operación extraería uno cualquiera de ellos. Todas las operaciones dentro del *ET* son concurrentes.

El diseño del *ET* mereció una especial atención. Si no se consideran aspectos de eficiencia, cualquier medio de almacenamiento, ya sea en disco o en memoria RAM, sería válido para albergar el *ET*. Como los *tuples* son de longitud variable, una estructura de archivo plano no sería adecuada; sin embargo, por simplicidad y sencillez, esta realización fue escogida para un primer prototipo. Para localizar un tuple, se pueden utilizar varios mecanismos de búsqueda, entre ellos el binario y el secuencial. De ellos, el de menor rendimiento es el secuencial. Sin embargo, éste fue el mecanismo elegido porque para efectos del primer prototipo, la eficiencia no era un objetivo principal.

2.2.- Sintaxis de *LINDA*

Los distintos operadores de *LINDA* utilizan como parámetro un n-tuple, cuyos componentes pueden ser valores constantes, variables instanciadas, y variables precedidas por un signo "?" para representar parámetros variables. Por ejemplo, si se invocan los siguientes comandos:

out("Puerto", 10.2, "A");
in("Puerto", ?valor, ?letra);

se obtendrán las variables "valor" y "letra" instanciadas con los valores 10.2 y "A", respectivamente.

La sintaxis de *C-LINDA* hace necesaria la utilización de un pre-procesador para el lenguaje fuente. Debido a que los llamados a los operadores de *LINDA* contienen un número variable de argumentos de diferentes tipos, así como argumentos de la forma "?variable", es necesario definir cuáles acciones son realizadas mediante llamados a rutinas de una biblioteca de códigos fuentes, cuáles mediante llamados a macros, y cuáles otras mediante un pre-procesamiento del programa.

Los operadores de *C-LINDA* deben tener la capacidad de recibir *tuples* con un número variable de argumentos, característica que no presenta el lenguaje C en el que las funciones declaradas deben tener un número fijo de argumentos. Fue necesario diseñar un mecanismo que proveyera esta facilidad. Después de un análisis, se decidió que los operadores *in, out, inp, rd y rdp* se podían programar mediante llamados a subrutinas, mientras que *eval* se programaría como una macro.

2.3.- Creación de procesos mediante *eval*

Eval es un operador difícil de construir debido a su capacidad para crear nuevos procesos, e incorporar *tuples* al *ET*. Sin embargo, UNIX ofrece llamadas al sistema como *fork* y *exec* para el manejo de procesos. Así, la llamada *fork* crea un nuevo proceso que hereda de su padre el ambiente existente, incluyendo las variables de ambiente definidas, descriptores de archivos abiertos, etc. Con ayuda de estas llamadas, se logró construir el operador *eval*.

2.4.- Tipos de datos de parámetros

Todo operador de *C-LINDA* debe poder reconocer el tipo de dato de cada componente del tuple a procesar. C, por su parte, no provee una macro *typeof(x)* que explore el dato *x* y determine su tipo, similar a *sizeof(x)* que sí existe. Esto obligó a modificar ligeramente la sintaxis de todos los operadores de *LINDA*, al requerir que el programador envíe una hilera indicadora de tipos dentro de cada llamado a un operador.

2.5.- Manejo de secciones críticas de código

La sección crítica de código es aquel código encargado del acceso y actualización del *ET*. Se diseñó un mecanismo que permita estas acciones, sin el peligro de llegar a un *ET* inconsistente. Para ello, se recurrió a las facilidades que ofrece UNIX en cuanto a manejo de semáforos, señales y memoria compartida.

2.6.- Bloqueo en los operadores *rd* e *in*

Los procesos que llamen a los operadores *rd* e *in* eventualmente podrán quedar bloqueados cuando los *tuples* buscados no estén disponibles. El bloqueo debe persistir mientras no se consiga el tuple. En otras palabras, se necesita que los operadores *rd* e *in* tengan la capacidad de dormir y que los operadores *out* y *eval* tengan la capacidad de despertarlos. Esto se realizó utilizando el mecanismo de señales que ofrece UNIX.

3.- EL PRE-PROCESADOR *PRECLI*.C

Precli.c es el corazón del proyecto. Es un programa escrito en C que se encarga de leer programas en lenguaje fuente de *C-LINDA*, para convertirlos en programas en lenguaje fuente C estándar, los cuales deben ser luego compilados bajo el compilador usual cc de UNIX, para producir finalmente el código ejecutable. Este pre-procesador se encarga de 4 acciones principales:

1. la preparación del ambiente de *C-LINDA*
2. la expansión del código de *eval*
3. la adaptación a la sintaxis de parámetros variables
4. el control de la finalización de procesos creados por *eval*

3.1.- Expansión del código de *eval*

El operador *eval*, debido a su funcionamiento, debe ejecutar llamadas a UNIX para creación de procesos (*fork*), manejo de estructuras de memoria compartida (*shmat, shmctl, shmdt, shmget*), control de la finalización de procesos (*wait* y macros de biblioteca <sys/wait.h>) y escritura de *tuples* (operador *out*). Por tanto, un *eval* de la forma *eval(f1(), f2(), f3())* se expande en:

- una llamada *fork* para cada función *f1, f2, f3* (procesos hijos),
- una llamada *fork* para crear un proceso padre de los *fork* funcionales, y
- una llamada *out(t1, t2, t3)* por parte del proceso padre, una vez que todos los procesos hijos han finalizado su trabajo y han asignado sus resultados a las variables *t1, t2* y *t3*.

En otras palabras, una llamada a *eval* con *n* parámetros crea *n+1* procesos concurrentes (uno para el padre y *n* para los hijos).

3.2.- Adaptación a la sintaxis de parámetros variables

Según la sintaxis de *C-LINDA*, el programador puede realizar un llamado a un operador con parámetros variables por ejemplo *rdp("cd", 'a', ?i)* donde *i* debe ser una variable de tipo *int*. *Precli* toma dicho llamado y lo traduce en *rdp("c?d", 'a', &i)*. Este proceso lo realiza para todos los operadores que tienen la capacidad de utilizar parámetros variables (*rdp, rd, in, inp*).

3.3.- Cuidados en el uso de *C-LINDA*

El uso del ambiente creado se basa en la aplicación del pre-procesador *precli* sobre el programa desarrollado en *C-LINDA*. Este programa debe tener un *include* de la biblioteca clinda.h.; es decir debe llevar como primera línea:

```
# include "clinda.h"
```

El uso de parámetros variables se da en los operadores *rd, rdp, in* e *inp*. Por ejemplo, para leer un tuple (con espera) de dimensión tres con los siguientes elementos: el entero 6, el *char* 'a' y un flotante cualquiera se puede utilizar la instrucción *rd("dcf", 6, 'a', ?flot)*, donde *flot* es una variable de tipo *float*. El número de parámetros variables cambia y recibe un pre-procesamiento de *precli* transparente para el programador.

En *eval* pueden ir como parámetros funciones que invocan a otros operadores de *C-LINDA*, exceptuando la llamada a *eval*. En otras palabras, no puede utilizarse a *eval* en forma recursiva. Tampoco se puede pasar directamente como parámetro, ninguno de los operadores.

lectores escritores en [SO-91]. La solución programada está descrita en [VA-93].

4.3.- Cálculo en paralelo de la multiplicación de matrices

Se trata de calcular una matriz cuadrada C producto de la multiplicación de una matriz A de dimensión *nxn* por una matriz B de igual dimensión, aprovechando el paralelismo inherente a dicho algoritmo, de modo que cada operación de multiplicar una fila de A por una columna de B se realiza en forma concurrente con todas las demás.

4.4.- Prueba de *inp*, *rdp* y *eval*

En los programas de prueba anteriores faltan algunos aspectos de *C-LINDA*, tales como: las llamadas al operador *inp* y *rdp*, tanto bajo comportamiento de éxito como de fracaso, así como la llamada a *eval* con más de un parámetro.

Con el fin de presentar un programa que incluya los aspectos anteriormente mencionados, se desarrolló un programa llamado prueba.cli, el cual crea un tuple de dimensión 3 y tipos "fdc". Con este tuple se realiza un *inp* fallido, y un *inp* exitoso. Luego, escribe otro tuple , y ejecuta un *rdp* exitoso. Por último, ejecuta un *eval* de un tuple de dimensión tres y de tipos "dgg", y evalúa su existencia mediante 2 *inp* con parámetros variables, indicando si localizó o no el tuple buscado. En medio de los 2 *inp* se realizan unas asignaciones, con el fin de dar tiempo a que *eval* termine, si es que aún no ha terminado.

5.- CONCLUSIONES

A continuación, se describen los mayores logros del proyecto. Debe tomarse en cuenta que el pre-procesador construido *precli*.c es un prototipo, por lo cual es admisible una serie de mejoras, algunas de las cuales se mencionan más adelante.

Se creó el pre-procesador *precli* que adecua los programas escritos en ambiente *C-LINDA* a la sintaxis del lenguaje de programación C estándar. Las pruebas realizadas funcionaron correctamente, lo cual es un indicativo de que, hasta este momento, *precli*.c está funcionando bien..

El *ET* fue diseñado de manera sencilla y completa. Tiene una estructura que permite su adaptación a otros medios de almacenamiento sin mayores problemas. Maneja en su totalidad los conceptos teóricos de *C-LINDA*, a saber, la variabilidad en la dimensión de los *tuples*, la diversidad de tipos de datos, su acceso concurrente y el mantenimiento de la integridad.

Los operadores tienen procedimientos y funciones modularizadas que permiten modificaciones con el mínimo de inconvenientes. Cada uno de ellos cumple con su descripción teórica y al ser utilizados en conjunto consiguen la elegancia y sencillez de que hace gala el ambiente *C-LINDA*.

La programación realizada es producto de una investigación de alternativas, algunas de la cuales, por diversas razones, fallaron. Esto plantea un interés especial en la investigación, particularmente en cuanto al manejo de *pipes* y señales, así como con el mejoramiento de las estructuras de almacenamiento y las políticas de búsqueda de *tuples*.

6.- BIBLIOGRAFIA

1. [CG-89]Carriero, Nicholas y Gelernter, David: "*LINDA in Context*" Comunications of the ACM 32 (4): 444-458, April 1989.

2. [KP-87]Kernighan, Brian y Pike, Rob: El Entorno de Programación UNIX, 1era edición, Prentice-Hall Hispanoamericana, México, 1987.

3. [SS-93]Shekhar, K. H. y Srikant, Y. N: "*LINDA* Sub System on Transputers" Comput Lang 18 (2), pp 125-136, 1993.

4. [SO-91]Silberschartz, Abraham y otros: Operating System Concepts, 3era edición,

Addison-Wesley Publishing Company Inc., USA, 1991.

5. [TA-88]Tanenbaum, Andrew S: Sistemas Operativos Diseño e Implementación, 1era edición, Prentice-Hall Hispanoamericana, México, 1988.

6. [VA-93]Vargas Morales, Juan José: "*LINDA*: Un ambiente para Programación Concurrente" Fercómputo 2do Congreso de Informática y Computación, Costa Rica, 1993.