UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
COLEGIO DE CIENCIAS Y HUMANIDADES
UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL
Y DE POSGRADO
INSTITUTO DE INVESTIGACIONES EN MATEMATICAS
APLICADAS Y EN SISTEMAS

DISEÑO Y SIMULACION DE UN SISTEMA OPERATIVO DISTRIBUIDO, RECONFIGURABLE Y TOLERANTE A FALLAS.

TESIS QUE PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS DE LA COMPUTACIÓN, PRESENTA:

YE QUNYING

1986 MÉXICO, D.F.

LIC. MANUEL MARQUEZ FUENTES
Directro
de la Unidad Académica de los
Ciclos Profesional y de Posgrado
del C.C.H.
U.N.A.M.

Después de haber leído la Tesis titulada: " DISEÑO Y SIMULACION DE UN SISTEMA OPERATIVO DISTRIBUIDO, RECONFIGURABLE Y TOLERANTE A FALLAS ", presentada por la alumna YE QUNYING, como requisito del Examen para obtener el grado de Maestro en Ciencias de la Computación, considero que reúne los méritos suficientes y no tengo inconveniente en formar parte del Jurado de Examen y dar mi voto Aprobatorio, a dicha tesis.

Sin otro particular, quedo de usted.

A t e n t a m e n t e

Cd. Universitaria, 4 de Febrero de 1986.

M. en C. LUIS HUGO PEÑARRIETA

Vocal

LIC. MANUEL MARQUEZ FUENTES
Director
de la Unidad Académica de los
Ciclos Profesional y de Posgrado
del C.C.H.
U.N.A.M.

Después de haber leído la Tesis titulada: " DISEÑO Y SIMULACION DE UN SISTEMA OPERATIVO DISTRIBUIDO, RECONFIGURABLE Y TOLERANTE A FALLAS " , presentada por la alumna YE QUNYING, como requisito del Examen para obtener el grado de Maestro en Ciencias de la Computación, considero que reúne los méritos suficientes y no tengo inconveniente en formar parte del Jurado de Examen y dar mi voto - Aprobatorio, a dicha tesis.

Sin otro particular, quedo de usted.

A t e n t a m e n t e
Cd. Universitaria, 4 de Febrero de 1986

Renato Barrera

DR. RENATO BARRERA RIVERA
Presidente.

LIC. MANUEL MARQUEZ FUENTES
Director
de la Unidad Académica de los
Ciclos Profesional y de Posgrado
del C.C.H.
U.N.A.M.

Después de haber leído la Tesis titulada: " DISEÑO Y SIMULACION DE UN SISTEMA OPERATIVO DISTRIBUIDO, RECONFIGURABLE Y TOLERANTE A FALLAS " , presentada por la alumna YE QUNYING, como requisito del Examen para obtener el grado de Maestro en Ciencias de la Computación, considero que reúne los méritos suficientes y no tengo inconveniente en formar parte del Jurado de Examen y dar mi voto - Aprobatorio, a dicha tesis.

Sin otro particular, quedo de usted.

A t e n t a m e n t e
Cd. Universitaria, 4 de Febrero de 1986.

G. Levine

M. en C. GUILLERMO LEVINE GUTIERREZ
Secretario

LIC. MANUEL MARQUEZ FUENTES
Director
de la Unidad Académica de los
Ciclos Profesional y de Posgrado
del C.C.H.
U.N.A.M.

Después de haber leído la Tesis titulada: " DISEÑO Y SIMULACION DE UN SISTEMA OPERATIVO DISTRIBUIDO, RECONFIGURABLE Y TOLERANTE A FALLAS " , presentada por la alumna YE QUNYING, como requisito del Examen para obtener el grado de Maestro en Ciencias de la Computación, considero que reúne los méritos suficientes y no tengo inconveniente en formar parte del Jurado de Examen y dar mi voto - Aprobatorio, a dicha tesis.

Sin otro particular, quedo de usted.

A t e n t a m e n t e
Cd. Universitaria, 4 de Febrero de 1986.

DR. ALEJANDRO BUCHMANN SAUTER
Suplente.

LIC. MANUEL MARQUEZ FUENTES
Director
de la Unidad Académica de los
Ciclos Profesional y de Posgrado
del C.C..H
U.N.A.M.

Después de haber leído la Tesis titulada: " DISEÑO Y SIMULACION DE UN SISTEMA OPERATIVO DISTRIBUIDO, RECONFIGURABLE Y TOLERANTE A FALLAS ", presentada por la alumna YE QUNYING, como requisito del Examen para obtener el grado de Maestro en Ciencias de la Computación, considero que reúne los méritos suficientes y no tengo inconveniente en formar parte del Jurado de Examen y dar mi voto - Aprobatorio, a dicha tesis.

Sin otro particular, quedo de usted.

A t e n t a m e n t e

Cd. Universitaria, 4 de Febrero de 1986.

DR. ALBERTO TUBILLA ESTEFAN
Suplente

México, 4 de Febrero de 1986.

Quiero expresar mi más sincero agradecimiento al Director de mi Tesis M.I. LUIS HUGO PEÑARRIETA ECHENIQUE, por sus valiosas sugerencias orientaciones y correcciones de esta Tesis, así como por su gran ayuda en mi formación profesional.

Agradezco a los profesores: DR. RENATO BARRERA RIVERA, M. en C. GUILLERMO LEVINE GUTIERREZ, DR. ALEJANDRO BUCHMANN SAUTER y DR. ALBERTO TUBILLA ESTEFAN, por sus comentarios revisiones y ayudas de mi Tesis.

Asimismo, agradezco al Departamento de Sistemas de Cómputo y la Biblioteca del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas , por las facilidades que me ofrecieron para desarrollar esta Tesis.

YE QUNYING

Tabla de Contenidos

Introducción
1. Sistemas distribuido 0

1.1 Introdución 0
1.2 Desarrollo del sistema de cómputo 1
1.3 Sistema distribuido 3
1.3.1 Definición 3
1.3.2 Ventajas del sistema distribuido 5
1.3.3 Red de las computadoras y sistema distribuido 6
1.3.4 Configuración 7
1.3.5 Distribución 9
1.3.6 Aplicaciones del sistema distribuido 12
1.4 Sistema Operativo Distribuido (SOD) 12
1.4.1 Definición 13
1.4.2 Modelo de sistema operativo distribuido 16
1.4.2.1 Modelo de objeto 16
1.4.2.2 Modelo de procesos 17
1.4.3 Los SOD Existentes 18

2. Diseño del sistema operativo distribuido 20

2.1 Introducción 20
2.2 Estructura física del sistema 20
2.2.1 Descripción del nodo 22
2.2.1.1 Computadora de tarea 22
2.2.1.2 Computadora de comunicación 23
2.2.2 Operación del nodo 23
2.2.2.1 Operación autónoma 25
2.2.2.2 Operación cooperativa 25
2.2.3 Rutamiento 26
2.2.4 Relación entre nodos y buses de la comunicación 26
2.2.5 Definiciones 27
2.2.6 Ejemplo 28
2.3 Estructura lógica del sistema 28
2.3.1 El nodo presidente 30
2.3.2 El nodo candidato 31
2.3.3 El nodo ministro 32
2.3.4 El nodo colaborador 32
2.4 Lenguaje de programación 32
2.4.1 Concurrencia y distribución 33
2.4.2 Procesos padre e hijos 36
2.4.3 Datos comunes 37
2.5 Características del sistema 40
2.5.1 Distribución de tareas 41
2.5.1.1 El protocolo de la distribución 41
2.5.1.2 Ejemplo de la distribución de las 43

tareas
2.5.1.3 Balanceo de las cargas 44
2.5.1.4 Procesos en modo background 45
2.5.1.5 Uso de recursos globales 45
2.5.2 Reconfiguración del sistema 46
2.5.2.1 Incremento de nodo 46
2.5.2.2 Despedida de un nodo 47
2.5.3 Tolerancia a las fallas 49
2.5.3.1 Falla recuperable 50
2.5.3.2 Falla grave 51
2.5.3.3 Falla catastrófica 53
2.6 Estructura de datos 53
2.6.1 En el nodo presidente 54
2.6.2 En el nodo ministro 54
2.7 Modelo del sistema 56

3. Simulación del sistema 60

3.1 Introduccion 60
3.2 Que es la simulación ? 60
3.2.1 El proceso de la simulación 61
3.2.1.1 Defincion de expermento 61
3.2.1.2 Modelación 62
3.2.1.3 Implementación de la computación 62
3.2.1.4 Validación 62
3.2.1.5 Datos de salida 63
3.2.2 Esquema general de la simulación 64
3.3 Simulación del SOD 65
3.3.1 Programa para la simulación 68
3.3.1.1 Sintaxis de programa 69
3.3.1.2 Representación estructurada 71
3.3.2 Reloj 71
3.3.2.1 Modo automático 74
3.3.2.2 Modo interactivo 74
3.3.3 Condicion inicial 74
3.3.4 Operaciones 76
3.3.4.1 Ejecución de los procesos 77
3.3.4.2 Transmisión y Recepción 78
3.3.5 Generador de eventos 80
3.3.5.1 Incremento del nodo 84
3.3.5.2 Despedida del nodo 85
3.3.5.3 Fallas 85
3.3.5.4 Recuperación de fallas 86
3.3.5.5 Entrada del programa 86
3.3.5.6 Modo de ejecución 86
3.3.5.7 Impresión de estados del sistema 86
3.3.5.8 Continuación de reloj 87
3.3.5.9 Terminación de la simulación 87
3.3.6 Impresión de resultados 88

3.3.6.1 Descripción del formato de resultados (salidas) 88
3.3.6.2 Ejemplo 89

4. Resultados 92

4.1 Pruebas simples 92
4.1.1 Ejecución de programa distribuido 92
4.1.1.1 Nodos colaboradores vs. tiempo de ejecución 93
4.1.1.2 Posición de colaboradores vs. tiempo de ejecución 95
4.1.1.3 Mensajes vs. tiempo de ejecución 101
4.1.2 Reconfiguración del sistema 103
4.1.2.1 Integración del nodo 9 104
4.1.2.2 Integración del nodo 4 104
4.1.2.3 Integración del nodo 1 105
4.1.2.4 Despedida del nodo 2 106
4.1.2.5 Despedida del nodo 1 106
4.1.2.6 Despedida del nodo 4 107
4.1.2.7 Falla de computadora de comunicación en el nodo 6 107
4.1.2.8 Falla de bus Y1 108
4.1.2.9 Falla de computadora de tarea en el nodo 9 109
4.1.2.10 Recuperación de la falla en bus Y1 109
4.1.2.11 Recuperación de falla en el 6 y el 9 110
4.2 Pruebas complicadas 111

5. Conclusión y trabajos futuros 124

5.1 Conclusión 124
5.2 Trabajos futuros 126
5.2.1 Sincronización de eventos 126
5.2.2 Interface 127
5.2.3 Programa real 127
5.2.4 Especifiacion de hardware 128

Bibliografía

Lista de Figuras

Figura 1-1: Sistema distribuido 4
Figura 1-2: Configuración de sistemas distribuidos 8
Figura 1-3: Modelo de CPU cache 11
Figura 2-1: Configuración 21
Figura 2-2: Componentes de un nodo 23
Figura 2-3: Computadora de tarea 24
Figura 2-4: Computadora de comunicación 24
Figura 2-5: Sistema general 27
Figura 2-6: Ejemplo de sistema 29
Figura 2-7: Configuración logica en una region 30
Figura 2-8: Ejemplo de proceso padre y proceso hijo 36
Figura 2-9: Protocolo de distribución de procesos 43
Figura 2-10: Ejemplo de distribución de tarea 44
Figura 2-11: Diagrama de flujo para incremento del sistema 48
Figura 2-12: Diagrama de flujo para fallas y recuperaciones 51
Figura 2-13: Comunicación entre procesos 57
Figura 2-14: Modelo del sistema 58
Figura 3-1: El esquema general de la simulación 64
Figura 3-2: Diagrama de flujo para la simulación 67
Figura 3-3: Un ejemplo de programa 70
Figura 3-4: Estructura del programa 72
Figura 3-5: Estructura interna del programa 73
Figura 3-6: Ejemplo 76
Figura 3-7: Buffer de transmisíon o de recepcion 79
Figura 3-8: Tabla de eventos 83
Figura 3-9: Resultado de la simulacion 91
Figura 3-10: Configuración del ejemplo 90
Figura 4-1: Representación del programa 93
Figura 4-2: Sistema de 2X2 94
Figura 4-3: Tiempos de ejecución del progarma 94
Figura 4-4: Sistema de 2X3 96
Figura 4-5: Tiempos del programa 97
Figura 4-6: El presidente en 0 y el programa en 1 99
Figura 4-7: El presidente en 1 y el programa en 0 0
Figura 4-8: Tiempos de ejecución caundo cambia tiempo de transmision 102
Figura 4-9: Estado inicial del sistema 103
Figura 4-10: Integración de nodo 9 104
Figura 4-11: Integración del nodo 4 105
Figura 4-12: Integración del nodo 1 105
Figura 4-13: Despedida del nodo 2 106
Figura 4-14: Despedida del nodo 1 106
Figura 4-15: Despedida del nodo 4 107
Figura 4-16: Nuevo configuración del sistema 108
Figura 4-17: Falla de computadora de comunicación del 108

nodo 6
Figura 4-18: Falla de bus Y1 109
Figura 4-19: Falla de la computadora de tarea en el nodo 9 110
Figura 4-20: Recuperación de la falla en bus Y1 110
Figura 4-21: Recuperación de los nodos 6 y 9 111
Figura 4-22: Trayectoria de la ejecución 113
Figura 4-23: Falla en el tiempo 2 en el nodo 1 114
Figura 4-24: Falla en el tiempo 13 en el nodo 1 115
Figura 4-25: Falla del nodo 0 en el tiempo 2 116
Figura 4-26: Falla del nodo 0 en el tiempo 7 117
Figura 4-27: Falla del nodo 0 en el tiempo 13 118
Figura 4-28: Falla del nodo 0 en el tiempo 22 119
Figura 4-29: Falla del noodo 0 en el tiempo 28 120
Figura 4-30: Ejecución de dos programas 121

Introducción

Un sistema distribuido es un sistema con elementos distribuidos geográficamente, los cuales se relacionan entre sí a través de líneas de interconección. Este sistema tiene machas ventajas tales como compartimiento natural de los recursos, distribución automática de las tareas, configuración flexible, y tolerancia a las fallas.

El sistema operativo es el núcleo de una computadora. Este controla las operaciones de un sistema de cómputo, con el fin de optimizar su funcionamiento y actuar como una interface entre los programas de usuario y el hardware de la computadora. El sistema operativo sobre un sistema distribuido se llama sistema operativo distribuido. Además de trabajar como un sistema operativo general, contempla la administración de los recursos que se encuentran ubicados en diferentes localidades. Esta administración significa compartir los recursos distribuidos por todos los procesos del sistema, tratar de balancear la carga entre los diferentes procesadores, operar sobre configuraciones dinámicas que pueden crecer o decrecer sin restrición alguna, y la posibilidad de tolerar fallas que pudiera presentarse en alguno de los recursos.

La presente tesis consiste en el diseño y la simulación de un sistema operativo distribuido (SOD) que se

aplica en un sistema distribuido de tipo matricial, en el cual se realiza automáticamente la distribución de tareas, se facilita la reconfiguración del sistema, y se permite la tolerancia a las fallas.

En el capítulo I, se trata sobre los concepto generales de sistemas distribuidos y se presentan algunas de sus características.

En el capítulo II, se presenta la estructura del sistema distribuido de tipo matricial y se procede a diseñar el sistema operativo distribuido con capacidades de distribución de las tareas, reconfiguración del sistema y tolerancia a fallas.

El capítulo III va a describir el trabajo de simulación del SOD basando en el sistema distribuido presentado en el capítulo II.

El capítulo IV presenta los problemas que se sometieron a la simulación y los resultados obtenidos, lo cual permite observar el comportamiento del SOD bajo diferentes tipos de cargas, asimismo se puede verificar los diferentes grados de recuperación del sistema cuando se simulan percanses o fallas imprevistas.

El último capítulo presenta las conclusiones a las

que se han llegado después de analizar los resultados de la simulación y se proponen trabajos futuros que se pueden seguir desarrollando sobre el sistema.

1. Sistemas distribuido

1.1 Introducion

La tecnología de los circuitos de gran escala de integración se ha desarrollado en forma muy acelerada en los últimos años. El uso de las microcomputadoras ha aumentado día a día tanto en las aplicaciones comerciales, industriales, en las organizaciones de gobierno como en las aplicaciones personales. Ciertas aplicaciones de las computadoras no son posibles llevarlas a cabo en microcomputadoras personales debido a la alta demanda de recursos, por ejemplo, alta velocidad de operación de procesador, necesidad de gran capacidad de memoria principal y masiva, manejo de diferentes tipos de periféricos, etc. Por lo que estas aplicaciones son atractivas para realizarse en computadoras grandes a un alto precio. Una buena solución para obtener un sistema de cómputo que tenga rendimientos semejantes a las grandes computadoras es aprovechar recursos de bajo costo integrandolos en un sistema único aunque éste se encuentre ubicado en distribuidas localidades geográficamente. Una ventaja adicional de este tipo de sistema es la facilidad que tiene sus procesadores (o recursos) de poder de trabajar independientemente o bien integrados cooperativamente dentro de sistema de mucho mayor capacidad.

Una de las partes más importantes de los sistemas distribuidos es el sistema operativo, el SOD es el encargado de integrar a todos los recursos en un solo sistema, el cual puede efectuar las operaciones en paralelo y tiene recursos redundantes que le permite tolerar ciertos tipos de fallas de algunos de estos recursos.

En este capítulo, se va a ver el desarrollo de los sistemas de cómputo, revisar los conceptos generales de los sistemas distribuidos y presentar el concepto del sistema operativo distriuido.

1.2 Desarrollo del sistema de computo

En la tercera generación del desarrollo de las computadoras, habían nacido muchas computadoras grandes que tienen mayor capacidad y veloccidad de procesamiento que las computadoras de primera y segunda generación; ocupan menos espacio porque sus fabricaciones están basadas en circuitos integrados; y tienen otras características que ahora son estándares [Levin 84]. Con las computadoras grandes se pueden obtener un sistema que centraliza datos y recursos para controlar usuarios en diferentes lugares por medio de líneas de comunicación. A este sistema se le llama sistema centralizado. Este sistema facilita el control de la seguridad y la integración de datos, pero tiene un costo alto.

En los últimos años, el desarrollo de la tecnología de circuitos integrados ha acelerado las producciones de las mini- y/o micro- computadoras, las cuales son baratas, pequeñas y no requiere condiciones especiales de operación. Por eso, estas computadoras son accesibles para que se utilizan en organizaciones pequeñas. El uso general de las computadoras en diferentes localidades geográficas que no tienen interacción entre sí, forma lo que se ha dado por llamar un sistema descentralizado, ya que las computadoras trabajan independientemente y no relacionan sus procesamientos entre sí.

Generalmente, los usuarios de las computadoras que están dispersas geográficamente accesan datos globales y a veces, a ellos les interesa el control de sus propios recursos. Con un sistema centralizado estas inquietudes se pueden resolver, pero el diseño es complicado y el costo es muy alto. Otra opción atractiva es instalar varios subsistemas interconectados entre sí, los cuales pueden ser computadoras baratas. Este sistema permite distribuir los recursos, los datos y las funciones en diferentes localidades geográficas y también permite tener control global de los subsistemas mediante teleproceso. Este sistema se llama sistema distribuido.

Por la breve descripción del desarrollo de los

sistemas de cómputo, se puede clasificar a estos en tres tipos principales: sistemas centralizados, descentralizados y distribuidos. Sus definiciones son las siguientes [Champine 80]:

Sistema Centralizado:
Este sistema tiene todos los recursos físicos y lógicos centralizados y puede tener usuarios remotos mediante teleproceso.

Sistema Descentralizado:
Este sistema tiene computadoras independientes en diferentes localidades geográficas, entre las cuales no existe ninguna interacción. Se toman decisiones en diferentes localidades físicas del sistema.

Sistema Distribuido:
Este sistemas tiene las computadoras en diferentes localidades geográficas, las cuales interaccionan entre sí y llava a cabo su procesamiento en forma cooperativa.

1.3 Sistema distribuido

En esta sección, se trata de describir el sistema distribuido con mayor detalle.

1.3.1 Definición

Generalmente, en un sistema distribuido cada localidad tiene un sistema de cómputo completo, en donde se tiene la unidad de procesamiento, la memoria, los periféricos de Entrada/Salida, los dispositivos de almacenamiento masivo y el software. A este sistema completo se le llama nodo. El nodo puede variar en tamaño

desde un gran sistema de multiprocesamiento hasta una microcomputadora, y en cualquier caso incluye software tanto de sistema como de aplicación.

Un sistema distribuido se compone de un grupo de nodos como los mencionados anteriormente y entre ellos se relacionan por medio de líneas de comunicación (ver figura 1-1). El término **sistema** en este caso, se usa para referir al conjunto de nodos que se comunican estre sí.

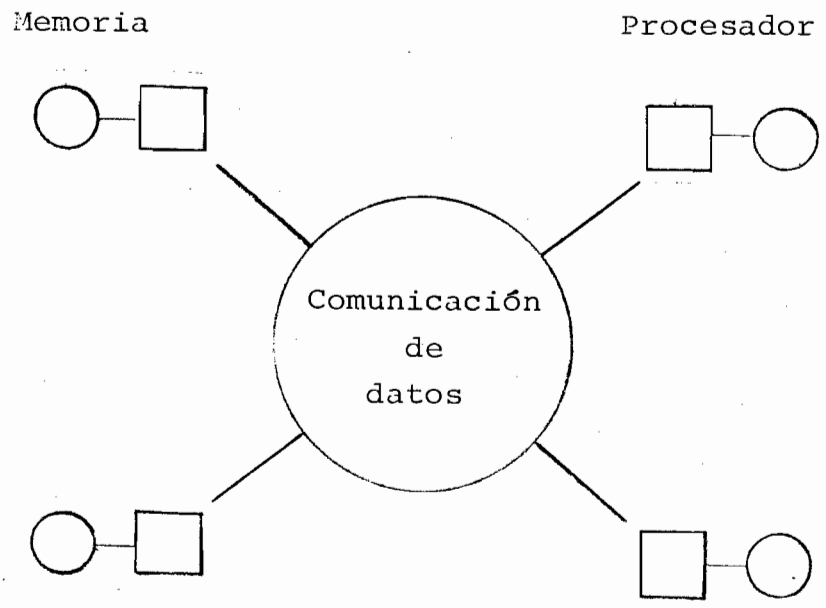


Figura 1-1: Sistema distribuido

Un sistema distribuido debe poseer por lo menos las siguientes características:

1. Los usuarios ven al sistema como un sistema centralizado o un sistema de uniprocesador.
2. La selección de recursos especificados por los usuarios es totalmente transperentes para ellos.
3. El sistema operativo distribuido puede distribuir y balancear automáticamente las cargas sobre los recursos disponibles.
4. El manejador de la base de datos distribuidos puede controlar los accesos concurrentes a los archivos por diferentes nodos y puede asegurar que los contenidos de copias redundantes de la base de datos sean consistentes todo el tiempo.
5. El sistema puede ser capaz de continuar las operaciones aunque haya fallado alguno de sus componentes.
6. El sistema tiene alta disponibilidad, es decir, busca de algún manera, que se puedan satisfacer los requerimientos de los procesos aunque no se logren cumplir con los recursos cercanos.

1.3.2 Ventajas del sistema distribuido

El sistema distribuido tiene muchas ventajas tanto para los usuarios como para el manejo de las operaciones internas. Algunas de ellas se mostrarán a continuación:

1. Capacidades de compartir automáticamente los recursos;
2. Distribución automática de la carga;
3. Facilidad de expansión en capacidades y funciones;
4. Flexibilidad de configuración;

5. Incrementar o decrementar modularmente el sistema.

1.3.3 Red de las computadoras y sistema distribuido

Hay confución en la literatura entre lo que es una red de computadoras y el concepto de sistema distribuido. La definición de Enslow [Tanenbaum 81] dice que un sistema distribuido tiene un sistema operativo con servicios que se requieren por nombre, y no por su localidad. En otras palabras, el usuario de un sistema distribuido ve el sistema como un uni-procesador. Ya que la asignación de tareas a procesadores, movimiento de archivos entre dos lugares y todas las funciones del sistema deben ser automáticas y transparentes.

Por otro lado, Liebowiz y Carson [Tanenbaum 81] han dicho: "Un sistema distribuido es aquel en el cual las funciones de computación están dispersas en varios elementos físicos." Obviamente esta definición incluye muchos sistemas que Enslow excluye.

En la vista de Andres S. Tanenbaum [Tanenbaum 81], un sistema distribuido es un caso especial de una red con un alto grado de coheción y transparencia. En realidad una red puede ser o no un sistema distribuido, dependiendo de su uso.

Por otra parte, el sistema distribuido es una consecuencia natural del desarrollo de las redes de las computadoras. Las redes de las computadoras se enfocan a obtener la comunicación entre los procesadores o microcomputadoras y la utilización de este sistema por los usuarios remotos. Se aplican los resultados del trabajo de la red de las computadoras al sistema distribuido para el diseño de sistemas de aplicación [Ralston 83].

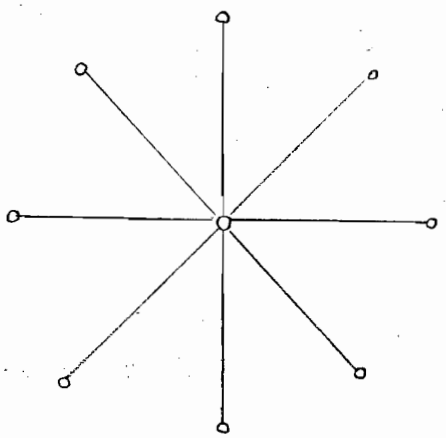
El sistema distribuido también se llama el sistema distribuido de información y a veces procesamiento distribuido se usa como sistema distribuido, sinónimamente.

1.3.4 Configuración

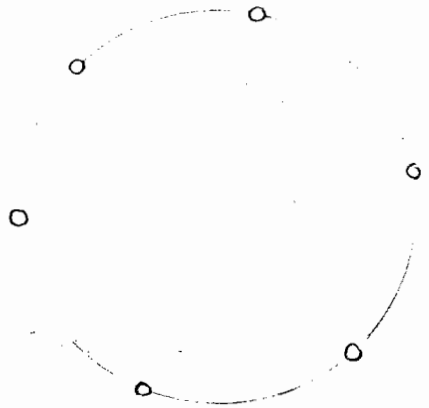
Existen varios tipos de configuraciones para interconectar los nodos en un sistema distribuido. Estos incluyen : (ver la sigueinte figura 1-2)

1. Estrella
2. Anillo
3. Conección completa
4. Jerarquía

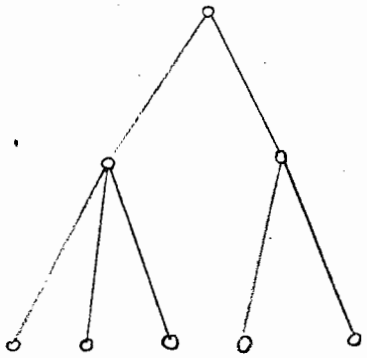
La configuración física de un sistema distribuido determina el diseño de las comunicaciones, tal como el costo y el retrazo de la transferencia de mensajes. La configuración lógica es tan importante como la física y



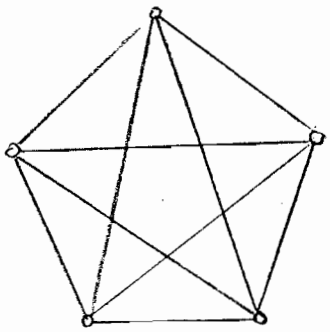
a. Estrella



b. Anillo



c. Jerarquía



d. Conección completa

Figura 1-2: Configuración de sistemas distribuidos

determina el manejo del sistema y de la base de datos. En un sistema distribuido las configuraciones físicas y lógicas

pueden ser diferentes. Por ejemplo, los nodos en una configuración física de anillo, pueden comunicarse lógicamente entre todos, como un sistema de tipo conección completa.

En algún sistema, las interconecciones físicas y lógicas son iguales.

1.3.5 Distribución

En un sistema distribuido lo que se puede distribuir, y cómo se puede distribuir es un problema interesante. Hay tres aspecto que se distribuye en este sistema:

1. Funciones de procesamiento;
2. Bases de datos;
3. Control de sistemas.

La distribución de funciones de procesamiento puede ser la distribución de computaciones, la cual consiste en dividir un problema de un usuario en varias piezas y correrlas en diferentes computadoras o procesadores. Hay varios modelos para hacer la distribución de computación; ellos son: modelo jerárquico, modelo de CPU cache, modelo de usuario-servidor, etc. [Tanenbaum 81]. La idea es tener una configuración lógica para distribuir las computaciones. Por ejemplo, en el modelo de CPU cache que se

describe en la figura 1-3, cada usuario tiene una minicomputadora que inserta entre su terminal y la computadora central de gran escala. Una parte de las computaciones va a estar en la computadora central y la otra parte en la minicomputadora. La decisión de ejecutar alguna de estas partes depende de la conveniencia de las dos máquinas, los costos relativos de las dos máquinas, el ancho de bandas de la comunicación entre las dos máquinas y las cargas actuales de las máquinas.

Una base de datos es distribuida si ésta puede ser dividida en distintas piezas para poner en diferentes lugares, así que para un usuario accesar a algún de estas piezas puede ser más lento que accesar a las demás. En el libro de C.J.Date [Date 83] habla más sobre este tema.

La distribución del control del sistema se puede lograr dividiendo el sistema operativo en funciones disjuntas. El sistema operativo es un conjunto de programas que controlan automáticamente las operaciones de un sistema de cómputo. Cada función forma una parte del sistema operativo y se ejecuta en un medio ambiente privado que puede ser un procesador. En la Universidad de Carnegie-Mellon, un sistema operativo llamado Medusa se utiliza para la arquitectura Cm*, el cual usa el esquema que se ha mencionado anteriormente para la distribución del

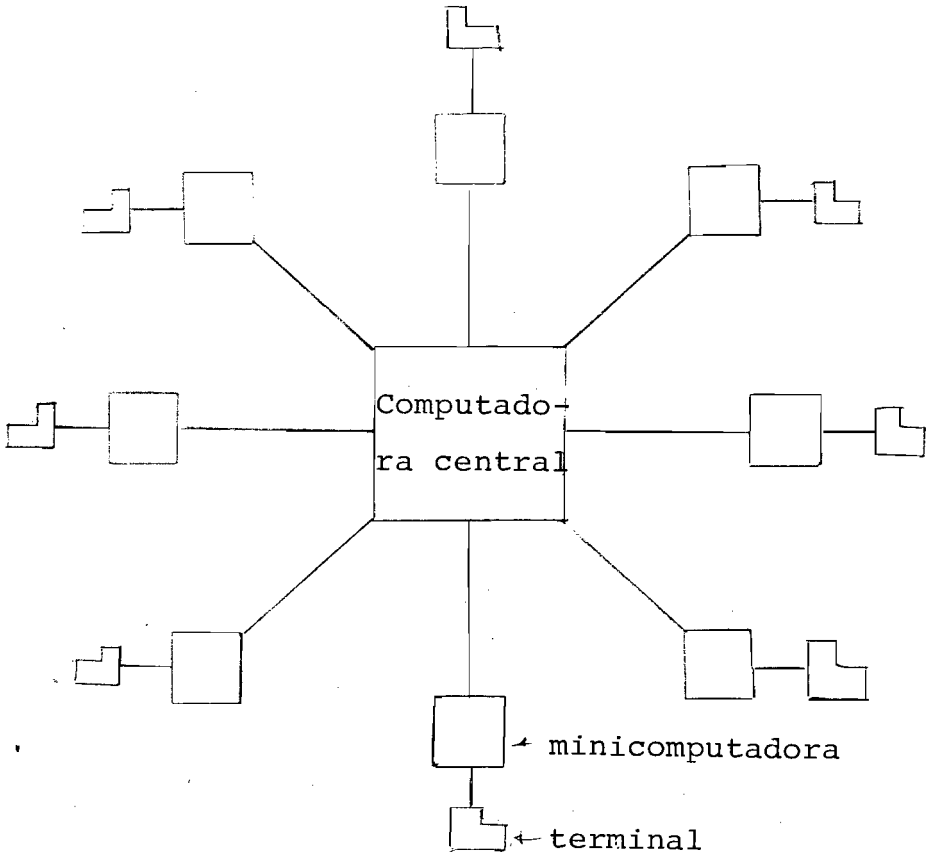


Figura 1-3: Modelo de CPU cache

control del sistema [Ousterhont 80].

1.3.6 Aplicaciones del sistema distribuido

Se puede aplicar el sistema distribuido en muchas areas tales como industrias, comercios, universidades, etc. Estas organizaciones son muy similares en las cuales tienen muchos usos individuales y necesitan también la coherencia y la comunicacion entre las unidades individuales.

Actualmente, el uso de las microcomputadoras se ha incrementado día por día. Pero en algunos casos, no se pueden satisfacer las necesidades de los usuarios con una sola de estas máquinas. Comparando con las computadoras grandes, la velocidad de ejecución es baja; la memoria es pequeña en algunos casos; y carece de los recursos de las computadoras grandes como discos de gran capacidad o cintas de 9 tracks. Si cada microcomputadora acompaña una impresora o cualquier otro recurso, el sistema puede ser descentralizado. Pero si se construye un sistema distribuido para interconectar las microcomputadoras individuales, se puede evitar los problemas de duplicación de los recursos, con los que va a obtener más eficiencia y una mejor utilidad de los recursos de las microcomputadoras.

1.4 Sistema Operativo Distribuido (SOD)

El hardware para implementar un sistema distribuido ya casi está disponible, pero el software aun está sometido a una amplia investigación. Hay problemas en diseño de

control de distribución y el manejo de los recursos dispersos, etc. Ellos son de parte del diseño del sistema operativo distribuido.

1.4.1 Definición

En una palabra, la implementación del sistema distribuido concluye en el problema de software , que es el Sistema Operativo Distribuido (SOD). El sistema operativo (generalmente se refiere sistema operativo centralizado) es la parte esencial de una computadora, ya que gobierna el control de los recursos tales como procesadores, memorias principales y secundarias, dispositivos de Entrada y Salida, y archivos; resuelve los conflictos; optimiza las ejecuciones y simplifica el uso efectivo del sistema de computadoras. Actúa como una interfase entre los programas de usuario y el hardware de la computadora. Un sistema operativo sofisticado incrementa la eficiencia y consecuentemente, disminuye el costo de uso de una computadora.

El SOD provee las mismas funciones lógicas de un sistema operativo centralizado en cada nodo [Booth 81]. Además, éste hace la distribución del procesamiento, ya sea de la base de datos y/o de control del sistema, y maneja las informaciones globales como usos de recursos dentro de los nodos del sistemas. Muchos SODs se basan en un sistema

operativo centralizado y desarrollan únicamente la parte del control de la distribución de la carga.

Tres problemas de investigación se distingue en el diseño del sistema operativo distribuido, ellos son:

1. Qué estructura lógica va a tener el sistema operativo distribuido.
2. Cómo recuperar el sistema cuando se presenta una falla.
3. Cómo separar las funciones en suficientes modulos para que una falla de cualquier no afecta al sistema completo.

Actualmente mucha gente adquirió microcomputadoras y cada quien quiere tener su propio software como compilador de PASCAL, compilador de C, manejador de base de datos, y una serie de paquetes de software. Pero la utilidad es limitada. Si existe un sistema distribuido para interconectar las microcomputadoras, se pueden guardar diferentes paquetes de softwares en diferentes computadoras, de esta manera cuando un usuario necesita alguno de ellos, aunque no exista en su propia computadora, el SOD lo busca automáticamente en todo el sistema sin que el usuario se de cuenta de esta actividad. Esto da el efecto de ampliar la memoria de una computadora, ya que se usa la memoria de otra computadora y la búsqueda del software es totalmente transparente para el usuario.

Generalmente, un grupo de microcomputadoras usa una impresora. Un usuario que no tiene la impresora en su propia computadora, debe de pasar su programa a la computadora que tiene la impresora. En caso de que esta computadora esté ocupada pero la impresora no, el usuario tiene que esperar. En un sistema distribuido, este ejemplo se resuelve de la siguiente manera: el SOD busca una impresora en todas las computadoras, si hay alguna, manda el archivo a imprimir en esta computadora aunque la impresora no está conectada a su proria computadora. Al terminar la impresión, el SOD manda al usuario un mensaje para decirle dónde se quedó su listado. En este sentido, la utilidad de los recursos en el sistema distribuido es alta porque el SOD aprovecha los recursos globales del sistema.

La capacidad de procesamiento de una microcomputadora es limitada. El SOD puede incrementar esta capacidad al doble o más, ya que esta micro está conectada a otras computadoras. Cada computadora tiene una unidad de procesamiento. Cuando una requiere ayuda de las otras, se pueden pasar trabajo a las otras para ejecutarse paralelamente. Entonces el tiempo de ejecución disminuye.

El presente trabajo es una investigación sobre un sistema operativo distribuido que realiza la distrbución de tareas automáticamente, permitir tolerancia a cierto tipo de

fallas y facilitar la reconfiguración del sistema.

1.4.2 Modelo de sistema operativo distribuido

La mayoría de los investigadores para el diseño del sistema operativo distribuido usa uno de los siguientes modelos:

-- Modelo de objetos.

-- Modelo de procesos.

1.4.2.1 Modelo de objeto

En el modelo de objeto, el sistema consiste de varios objetos, cada uno de ellos tiene un tipo, una representación y un conjunto de operaciones que puede ejecutar sobre el mismo objeto. Para llevar a cabo una operación sobre un objeto, por ejemplo, leer desde un archivo, un proceso de usuario debe poseer una capacidad (el permiso) del objeto. La tarea básica del sistema operativo es manejar la capacidad y permitir las operaciones. En un sistema centralizado, el sistema operativo mantiene las capacidades de los objetos dentro del mismo. Un objeto no puede operar sobre el otro si no tiene permiso. En un sistema distribuido las capacidades de los objetos deben pasarse a los distintos nodos según las necesidades.

1.4.2.2 Modelo de procesos

En el modelo de procesos, cada recurso (archivo, procesador, periférico, etc) se maneja por procesos y el sistema operativo debe manejar las comunicaciones entre los procesos. Existen dos diferentes esquemas fundamentales para implementar el mecanismo de la comunicación de entre-procesos (IPC), los cuales son: el esquema de llamadas de función y el esquema de pasar mensajes.

Cuando se usa el esquema de llamadas de función para IPC, el sistema completo consiste de una colección de funciones (o dice procedimientos) escritas en algun lenguaje. El código para las funciones es distribuido entre los procesadores o computadoras. Para lograr la comunicación entre las funciones, una función que está en una computadora puede llamar a otra función de otra computadora. Por ejemplo, en un sistema de multiprocesadores, un programa contiene la instrución **y=f(x)**. Las variables **x** y **y** están en el procesador 1 y la función **f** está en el procesador 2. Cuando el procesador 1 empieza a ejecutar esta instrucción, pasa el parametro **x** al procesador 2 y suspende la ejecución en el procesador 1 hasta que regrese el resultado de la funcion **f**. Para obtener la sincronización, puede usar semáforos o monitores [Colin 83]. Con este tipo de IPC, el sistema operativo se escribe como un programa grande, el

cual da la coherencia pero falta la flexibilidad [Tanenbaum 81].

El esquema de pasar mensajes, consiste de un conjunto de procesos que se comunican entre sí para intercambiar mensajes. El código de cada proceso es independientemente de los demás proecsos. Por ejemplo, los procesos se pueden escribir en diferentes lenguajes. Cada proceso tiene un puerto, cuando un proceso quiere comunicar con el otro, necesita saber la identificación del otro proceso o el nombre de su puerto. Como se pueden tener varios procesos que transfieren mensajes a un proceso al mismo tiempo, se necesita el mecanismo de sincronización en el puerto.

La presente tesis es un sistema operativo distribuido basado en el modelo de procesos con el mecanismo de comunicación de entre-procesos por el esquema de pasar mensajes.

1.4.3 Los SOD existentes

Actualmente ya existe varios SOD. SODS/OS es un SOD para IBM serie/1; es un proyecto de Universidad de Delaware [Sincoskie 80]. Medusa es un SOD para Cm* multiprocesadores en la Universidad de Carnegie-Mellon [Ousterhont 80]. El sistema DEMOS en nacional laboratorio de físico en Gran

Bretania es una red de anillo simple de 5-50 minicomputadoras [Dowson 78]. MICROS es un sistema operativo distribuido para la red de computadoras MICRONET que es reconfigurable y extendible de 16 nodos con switches para dos buses de alta velocidad [Whitlie 80].

2. Diseño del sistema operativo distribuido

2.1 Introducción

En este capítulo, se va a describir el diseño de un Sistema Operativo Distribuido (SOD), el cual se aplica en un sistema distribuido. Las características sobresalientes de SOD son las siguientes:

1. Capacidad de incremento modular
2. Distribución automática de tareas
3. Toleracia a las fallas
4. Reconfigurabilidad

2.2 Estructura física del sistema

SOD es un sistema operativo que podrá ser instalado en un sistema distribuido de mini- y/o microcomputadoras, que tenga una estructura matricial tal como se observa en la figura 2-1. El sistema se compone de un conjunto de buses (o canales) de comunicación de alta velocidad organizados en forma matricial, en la intersección de dos buses se encuentra ubicada una minicmputadora o dos microcomputadoras llamada nodo.

Cuando dos nodos requieren comunicarse entre sí, necesitan transmitir los datos por los buses de comunicación; en ocaciones tienen que utilizar nodos intermedios para lograr este objetivo. Por ejemplo, la

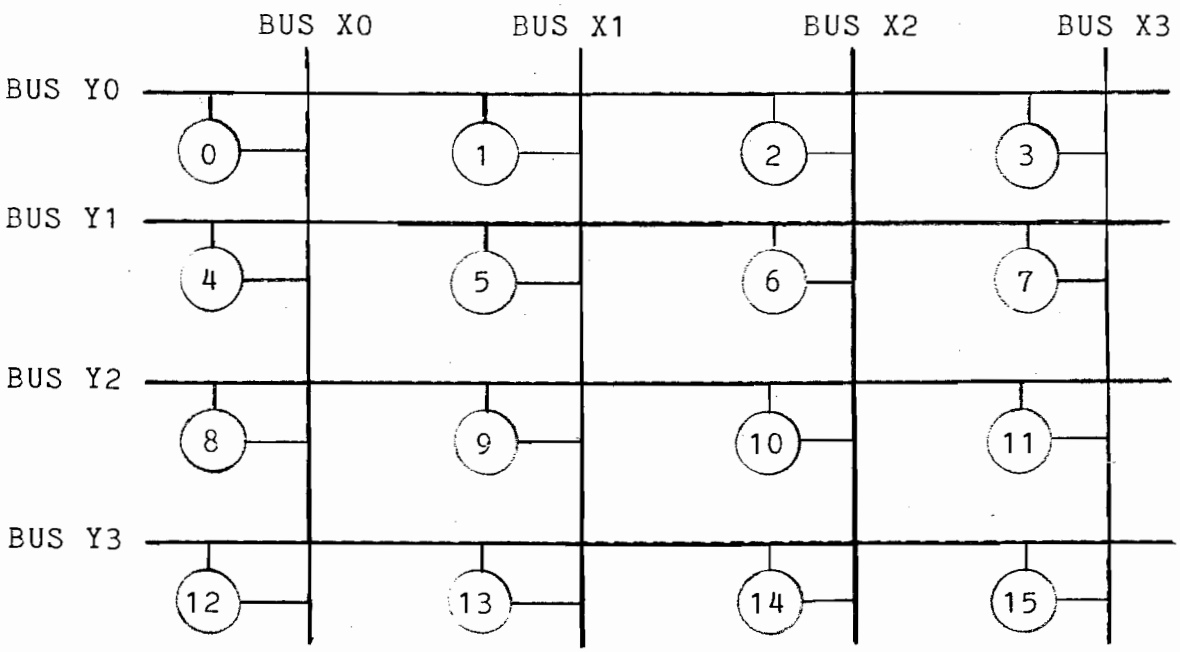


Figura 2-1: Configuración

comunicación entre el nodo 4 y el nodo 15 se puede realizar utilizando el BUS Y1 para transferir la información hasta el nodo 7, y luego el 7 las transfiere al nodo 15 a través del BUS X3. Las rutas para comunicación entre dos nodos son redundantes y existen varias posibilidades para transferir las informaciones. En el ejemplo anterior, otra ruta desde el nodo 4 hasta el nodo 15 es transferir al nodo 8 por el BUS X0, luego al nodo 11 por el BUS Y2 y finalmente al nodo 15 por el BUS X3.

2.2.1 Descripción del nodo

En el presente sistema, el nodo consiste de dos computadoras principales, una se denomina computadora de tarea, la cual funciona como una computadora convencional, y la otra se denomina computadora de comunicación que se encarga de los trabajos de transferencia de datos (ver la figura 2-2). La computadora de comunicación tiene dos puertos para comunicarse con los buses externos mencionados anteriormente. Las computadoras de tarea y de comunicación se ligan por un canal de acceso directo de memoria (DMA). Las dos computadoras comparten una memoria común, la cual puede ser utilizada por una a la vez.

2.2.1.1 Computadora de tarea

La figura 2-3 muestra los componentes de la computadora de tarea. Esta se compone de un CPU, la memoria, el controlador de periféricos de entrada y salida y el controlador de memoria masiva. En esta computadora, se realiza tareas de los usuarios y los programas del sistema. La computadora de tarea tambien tiene dos puertos para la comunicación con los buses, pero generalmente no se usan, solo en el momento en que fallan los dos puertos de computadora de comunicación simultáneamente, entonces la computadora de tarea manda las informaciones del sistema como mensajes urgentes a los buses. En este caso, como la

transferencia no pasa por la computadora de comunicación, no se pueden dividir mensajes largos.

2.2.1.2 Computadora de comunicación

Esta computadora hace el trabajo de la comunicación con los otros nodos, tales como división de mensajes largos en paquetes para la transmisión o viceversa, control de flujo, rutamiento, etc (ver la figura 2-4).

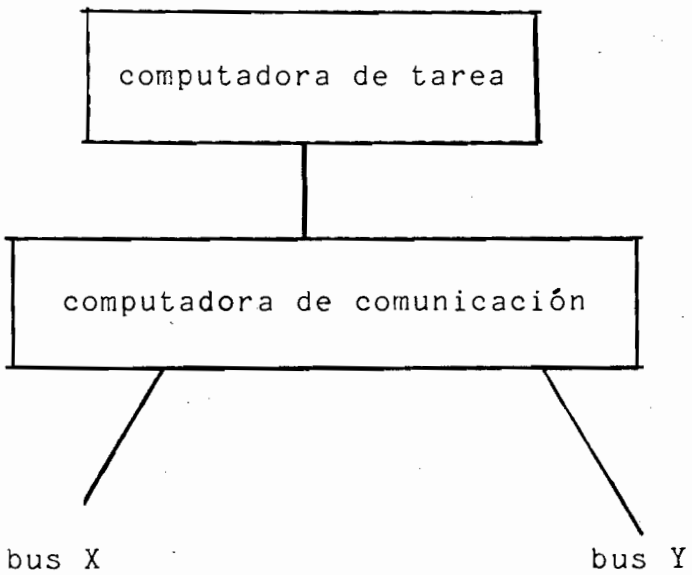


Figura 2-2: Componentes de un nodo

2.2.2 Operación del nodo

Cada nodo tiene un switch para prender y apagar la máquina. Eso significa que no todos los nodos estan trabajando para el sistema en todo el tiempo aunque cada uno

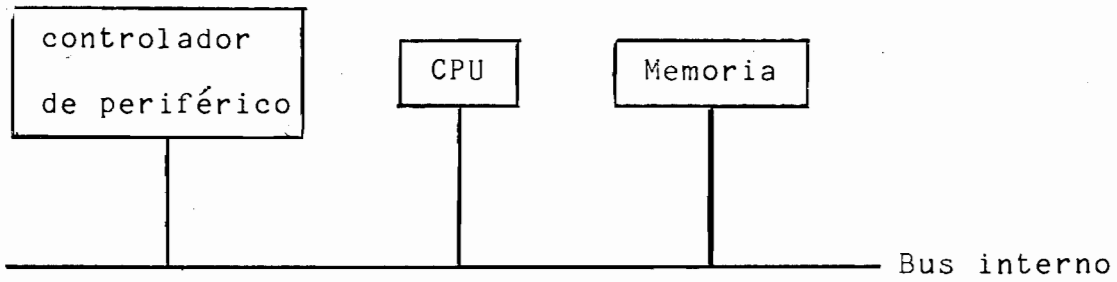


Figura 2-3: Computadora de tarea

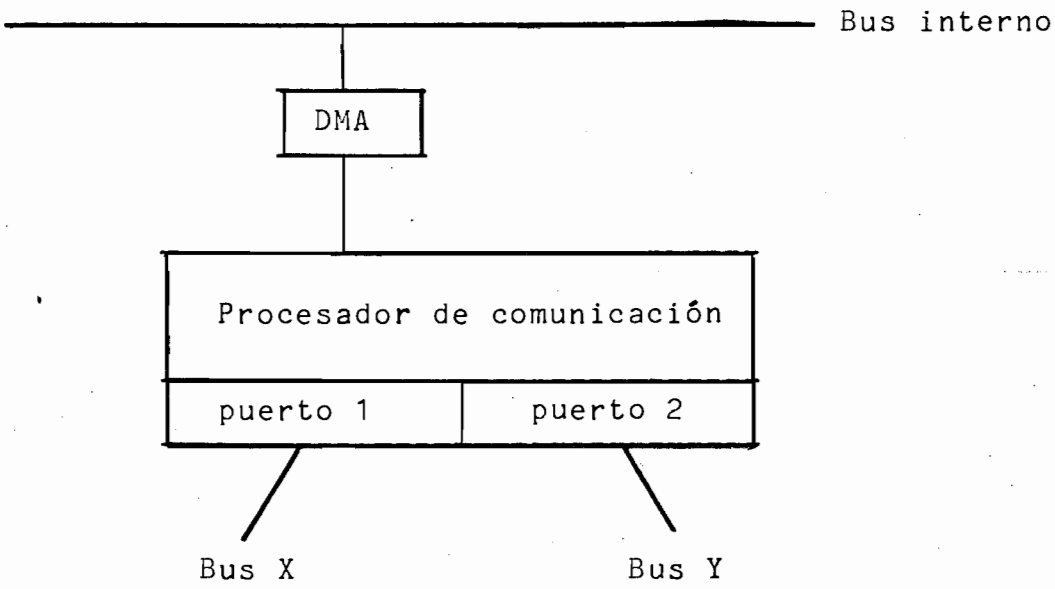


Figura 2-4: Computadora de comunicación

tiene su posición física en el sistema distribuido. El sistema es reconfigurable y extentible porque en cualquier momento se puede agregar nodos y expander el sistema sin

afectar la operación normal del sistema.

2.2.2.1 Operación autónoma

Un nodo puede trabajar autónomamente sin interaccionar con el resto de los nodos; aunque ocupa una posición física en la configuración, el SOD no considera este nodo como el nodo del sistema. Los trabajos se hacen sólo para el nodo mismo. A las operaciones que se hacen los trabajos de este tipo, se llaman operaciones autónomas.

2.2.2.2 Operación cooperativa

Una vez que el nodo pertenece al sistema (SOD trata este nodo como nodo del sistema), éste puede trabajar para otros nodos del sistema. A través de la computadora de comunicación, un nodo puede comunicarse con los demás nodos del sistema. Desde este momento, este nodo puede hacer trabajos en otros nodos o hacer trabajos de otro nodo. Por eso se llaman cooperativas a eate tipo de operaciones.

El nodo puede trabajar independientemente del sistema. Pero si el usuario quiere integrarlo al mismo, necesita llamar a SOD. Después de la integración de un nodo, el sistema tiene un nodo más, y el SOD lo considera para la cooperación con otros nodos. Por eso el sistema es autónomo y cooperativo.

2.2.3 Rutamiento

Cuando un nodo necesita comunicar con otro, primero checa si ya existe una ruta establecida entre ellos. Si no la hay, debe buscar una mediante un método de rutamiento [Tanenbaum 81]. A través de esta ruta se puede pasar el mensajes de petición de un recurso, transferir códigos de subrutina o proceso, y pasar variables comunes, etc.

2.2.4 Relación entre nodos y buses de la comunicación

Se debe definir el tamaño máximo del sistema para que SOD pueda manejarlo con facilidad. Los nodos y buses tienen sus identificaciones. Dado la identificación de un nodo, se lo puede localizar, o sea, con cuáles buses debería conectar este nodo. Al revés, dado las identificaciones de los buses X y Y, se puede saber qué identificación tiene el nodo. Las relaciones entre los nodos y los buses se muestran en las siguientes formulas (ver la figura 2-5):

i = identificación del nodo

N = número máximo de buses X

M = número máximo de buses Y

El tamaño del sistema = (N , M) = N X M

La posición de i = (a , b)

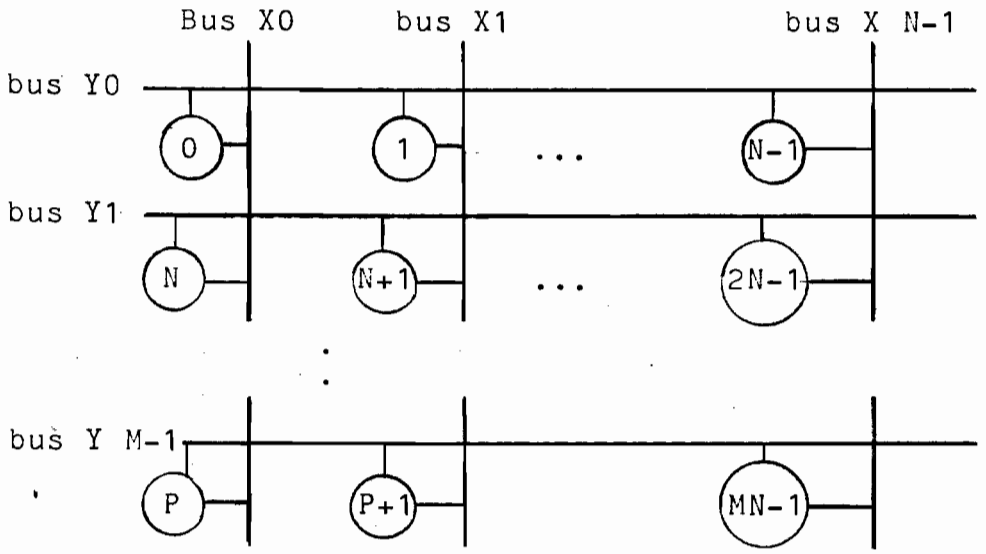
a = i mod N : donde a indica la posición en bus X.

b = i div N : donde b indica la posición en bus Y.

Nota: M puede ser no definido para expander el sistema en sentido vertical.

En caso contrario, dado la posición de un nodo (Bus Xa, Bus Yb), se puede saber la identificación:

$$i = b * N + a.$$



Nota: P= (M-1)N

Figura 2-5: Sistema general

2.2.5 Definiciones

En esta sección, se dará unas definiciones sobre el sistema. Ellos son:

Nodos activos: son aquellos que se consideran dentro de SOD. Pertenecen al sistema global y pueden ser nodos colaboradores de otro nodo del sistema. Generalmente, el nodo del sistema

se refiere al nodo activo.

Nodos pasivos: son los que no se integran al sistema y estan trabajando autónomamente. Las tareas de estos nodos no se pueden distribuir por el SOD.

Vecinos: son los nodos que se conectan al mismo bus.

Región: incluye un conjunto de nodos activos relacionados por los buses.

Regiones disjuntas:
significan que entre las regiones no pueden comunicarse.

Sistema: contiene una o varias regiones que son coordinadas por SOD.

2.2.6 Ejemplo

En la figura 2-6 se ilustra un ejemplo para entender bien los conceptos anteriores.

En este sistema de 4X4, los nodos achurados son nódos activos, los cuales son 1, 4, 5 y 11. Los restos nodos son pasivos. Los primeros nodos forman una región (la región 1). El nodo 11 es otra región (región 2) que solo consiste de sí mismo. El nodo 5 es vecino del nodo 1 y también es del nodo 4. Pero el nodo 1 no es vacino de 4. Las dos regiones son disjuntas.

2.3 Estructura lógica del sistema

No es suficiente tener una estructura física para que un sistema funcione, la estructura lógica siempre es el mecanismo mediante el cual opera el sistema. Una estructura

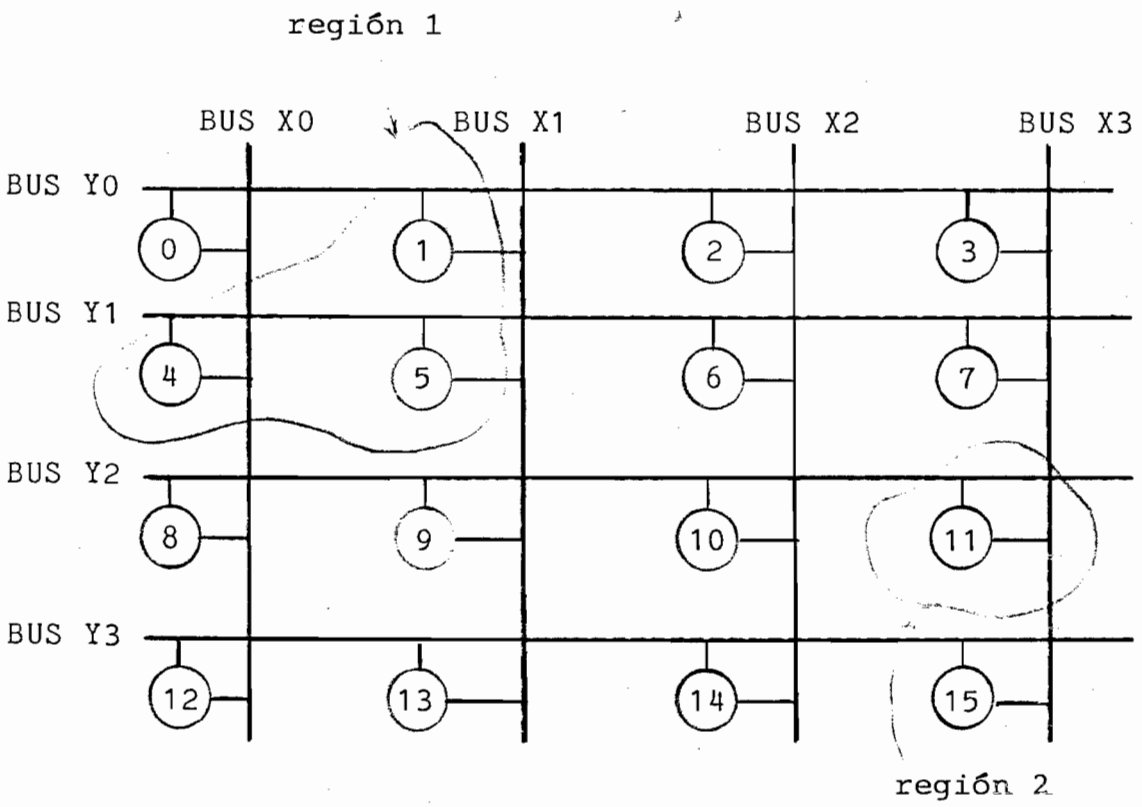


Figura 2-6: Ejemplo de sistema

lógica puede ser diferente a la estructura física.

En el presente trabajo, el SOD controla los nodos del sistema utilizando la estructura lógica con el modelo de jerarquía. En el sistema se divide en regiones. Cada región tiene un nodo presidente, uno o varios nodos ministros y nodos colaboradores (ver la figura 2-7).

En el ejemplo anterior de la página 28, si se agrega el nodo 2, la estructura física cambia por este nuevo nodo; pero la estructura lógica sigue siendo una jerarquía en la

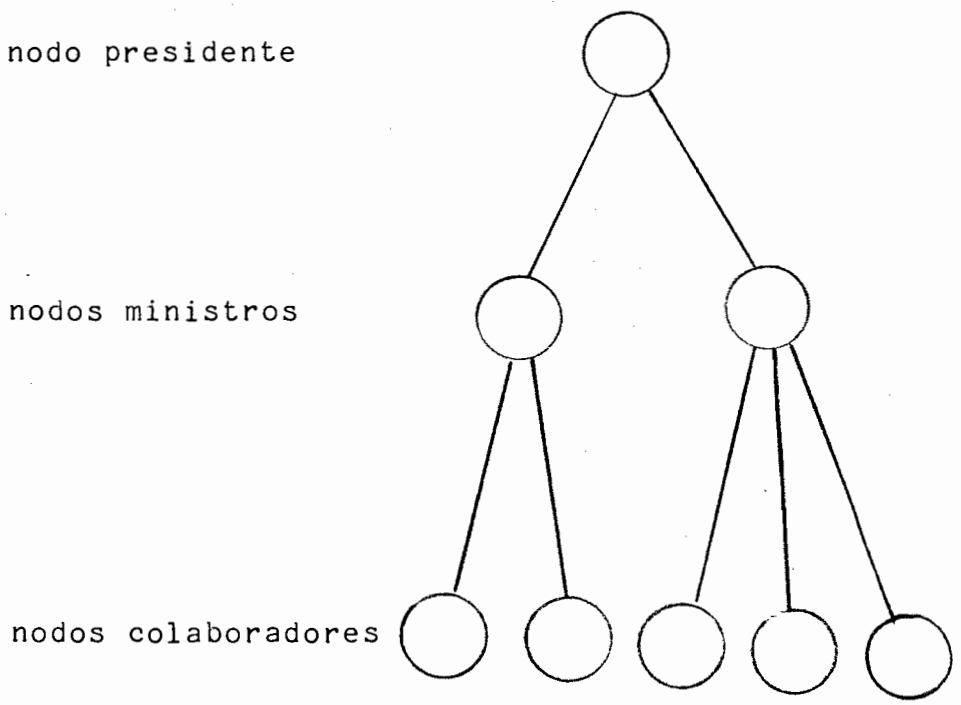


Figura 2-7: Configuración logica en una región

cual existe un presidente y varios ministros. A pesar de que todos los nodos del sistema están en un bus, o está en diagonal, la estructura lógica siempre es de tipo jerárquica.

2.3.1 El nodo presidente

El nodo presidente puede ser cualquier nodo en el sistema según la situación cuando cambia el sistema. Cuando un nodo requiere integrarse al sistema, primero el SOD verifica si este nodo tiene relación con los otros. Si con sus dos buses no puede comunicar con ningún nodo del

sistema, este nodo va a ser presidente de una nueva región donde sólo hay este nuevo nodo. Si este nodo tiene rutas para llegar a algún nodo de una región, el SOD lo integra a esta región y el presidente de este nodo es el presidente de esta región. El presidente tiene una lista de nodos ministros junto con sus porcentajes de cargas y sus recursos que posean. Estas informaciones son mandadas por los ministros periódicamente al presidente. Con los datos globales el presidente puede obtener colaborador y recursos para satisfacer la petición de algún proceso.

El puesto de presidente es dinámico. Los nodos en cualquier momento puede salir del sistema. Entonces la configuración física cambia, las rutas entre nodos activos cambian y a lo mejor forman nuevas regiones. El nodo presidente puede renunciar cuando pasa algo en el sistema, por ejemplo, las fallas del presidente, el despedido del presidente, o el cambio de configuración. Como restricción, el nodo presidente no puede ser colaborador.

2.3.2 El nodo candidato

En una región hay un candidato de presidente que puede sustituir el presidente cuando hay una falla de este mismo. El presidente manda sus informaciones al candidato periódicamente. Por eso, el candidato tiene casi todas las informaciones globales que el presidente tiene.

2.3.3 El nodo ministro

Los nodos ministros corresponden a los nodos que generan procesos por sus propias iniciativas. Ellos reciben comandos que mandan por el usuario y los ejecutan, generan procesos y piden recursos como impresora. Cada nodo activo es un nodo ministro, solo está en una región con su prorio presidente .

2.3.4 El nodo colaborador

Los nodos colaboradores son aquellos que son asignados por el presidente para auxiliar en la ejecución de una tarea a otro nodo. Si los nodos tienen poca carga, ellos pueden ser nodos colaboradores. Estos hacen trabajos para otros nodos y luego regresan resultados o señales de terminación al nodo iniciativo. Un nodo colaborador es ministro también.

Nota: Un nodo del sistema se denomina como un nodo ministro. Al mismo tiempo, éste puede ser nodo presidente o nodo colaborador.

2.4 Lenguaje de programación

Un programa distribuido se puede caracterizar como un algoritmo que requiere multiples procesadores para su ejecución, mientras un programa concurrente requiere multiples procesos para su implementación. Un programa

secuencial es un algoritmo que se implementa usando solo construcciones secuenciales [Cook 80].

Se puede aplicar cualquier tipo de estos tres programas en este sistema. El SOD encarga las distribuciones de procesos cuando es necesario.

2.4.1 Concurrencia y distribución

Cualquier programa se escribe por medio de un lenguaje. Un programa secuencial, concurrente y distribuido utiliza un lenguaje secuencial, concurrente o distribuido, respectivamente.

El programa concurrente que se ejecuta en sistema de un solo procesador provee instrucciones para empezar la ejecución de procesos concurrentes. Las instrucciones de **COBEGIN** y **COEND** sirven para agrupar los procesos concurrentes. Por ejemplo, un programa concurrente puede ser el siguiente:

```
S1
S2
S3
COBEGIN
  Proceso1
  Proceso2
  Proceso3
COEND
S4
S5
```

En este ejemplo, terminando la instrución S3 cuando se encuentra COBEGIN, los tres procesos 1, 2 y 3 van a hacer la ejecución concurrentemente. Hasta que el último proceso termine, se empieza a ejecutar la siguiente instrución S4. En el lenguaje OCCAM, usa la instrucción **PAR** para indicar el inicio de los procesos concurrentes.

En lugar de la ejecucion concurrente, el lenguaje distribuido ejecutan los procesos en varios procesadores paralelamente. *MOD es un lenguaje distribuido. Se define cuál procesador va a hacer cuál proceso. Por ejemplo, un programa en lenguaje *MOD puede ser la siguiente forma:

```
network module star=(centro,pr),(pr,centro);
  const noprocesador=4;
  processor module centro;
    define comunicator;
    process comunicator;
        .
        .
        .
      pr[i].comunicator(...); (*llama procesador i*)
        .
        .
        .
    end comunicator;
    begin  (* inicialización *)
    end  center;
  procesador module pr[noprocesador];
    define comunicator;
    import center;
    process comunicator(...);
        .
        .
        .
      center.comunicator(...);  (* llama central *)
        .
```

```
        .
        .
    end comunicator;
  begin  (* inicialización *)
  end pr;
end  star;
```

En este ejemplo, primero se definío que la configuración es de tipo estrella y luego los nombres de los procesadores en donde van a hacer procesos. Al empezar la ejecución de este programa, en los procesadores **centro** y los otros cuatro procesadores hacen diferentes procesos. Una de operación de los procesos es comunicar entre procesador **centro** y los esclavos pr[i].

Pero este lenguaje necesita aplicarse en un sistema perfecto (el sistema no sucede fallas durante la ejecución de los procesos) y las identificaciones de los procesadores están bien definidas. El usuario deben saber cuáles procesadores pueden usar y cuáles no. Si hay alguna falla de procesador durante la ejecución, no funcionará el programa.

Se desea aplicar con más flexibilidad los lenguajes tanto de concurrencia como de distribución a este sistema. Aunque el programa está hecho por lenguaje concurrente, si el compilador genera códigos para ejecutar los procesos en paralelo, el SOD distribuye los procesos en diferentes nodos del sistema. En la sección 2.5.1 se va a describir sobre la

distribución de las tareas.

2.4.2 Procesos padre e hijos

El proceso que indica el punto de los procesos paralelos se llama proceso padre, mientras los procesos paralelos se llaman procesos hijos. Cuando terminan las ejecuciones de los procesos hijos, ellos deben regresar al proceso padre las señales de terminación junto con resultados de estos procesos. En la figura 2-8 muestra una estructura de un programa. El proceso padre empieza a hacer la instrución S1. Cuando termina esta instrución, el proceso padre genera dos procesos hijos P1 y P2 en donde pueden contener varias instrucciones. Hasta que los dos procesos terminen sus ejecuciones, el proceso padre puede continuar la instrución Sn.

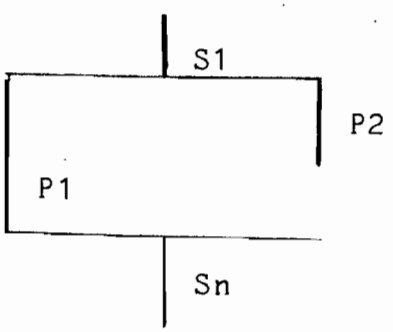


Figura 2-8: Ejemplo de proceso padre y proceso hijo

Cuando un proceso llama una subrutina que está en el

otro nodo, el SOD trae código de la subrutina. Las variables comunes entre los procesos hijos se guardan en el proceso padre. Cuando uno quiere accesar a estas variables, deben regresar al nodo padre (el nodo que contiene proceso padre). Se considera a estas variables como sección crítica y los procesos las accesan con exclusión mutua. Los códigos del programa deben ser reentrantes y relocalizable.

2.4.3 Datos comunes

Los procesos concurrentes o distribuidos pueden intercambiar informaciones o accesar variabbles comunes. Para tener este acceso de exclusión mutua a las variables comunes (considerando como sección crítica), se necesita tener control para la sincronización de los procesos. En el problema de productor-consumidor hay N procesos de productores y M procesos de consumidores. Los productores preparan los productos y los meten en un almacén; y los consumidores sacan desde el almacén los productos y los consumen. Los procesos necesitan meter los productos o sacarlos en la forma de exclusión mutua. Eso significa que no más de un productor puede depositar los productos al almacén al mismo tiempo, o no más de un consumidor puede sacar los productos al mismo tiempo. Para evitar este tipo de problema, una buena solución es usar semáforos: uno para productores, otro para consumidores y dos para almacén. El

semáforo de productor es para evitar dos o más productores meten productos al almacén al mismo tiempo. Una vez un productor empieza a hacer su operación, prende el semáforo, los otros productores no pueden hacer nada hasta que apague el semáforo. El semáforo de consumidor también es para evitar otros consumidores sacan productos al mismo tiempo. Los dos semáforos de almacén son para indicar si está lleno o vacío el almacén. El semáforo **lleno** controla los productores que no depositen más productos cuando el almacén está lleno mientras el semáforo **vacío** evita que los consumidores hagan operaciones cuando el almecén está vacío. Cuando se prende cualquier semáforo, el proceso correspondiente necesita formar en su propia cola. Hasta que el semáforo se apague el primer proceso en la cola puede lograr lo que quiera.

En el caso de procesos concurrentes o distribuidos se ponen las variables comunes en el nodo de proceso padre. Los procesos hijos pueden accesar las variables comunes con la exclusión mutua, mediante un mecanismo como semáforos, candados, monitores, etc [Holt 83], cuyas implementaciones pueden tener la construcción de **mutexbegin/mutexend**. **Mutex**begin y mutexand abarcan los procesos concurrentes o distribuidos para garantizar los accesos de las variables comunes en forma de exclusión mutua. A continuación se

ilustra un esquema:

```
proceso 1                    proceso 2

statment                     statment
mutexbegin                   mutexbegin
  acceso de                    acceso de
  variable comun               variable comun
mutexend                     mutexend
statment                     statment
```

La implementación de mutexbegin/end se realiza por la siguiente manera:

Mutexbegin: Se debe determinar si existe otro proceso en la sección crítica, es decir, se verifica si hay algun proceso está accesando la variable común y todavía no termina de usarla o modificarla. Si lo es, el proceso que quiere entrar la sección crítica debe esperar en la cola correspondiente. Al contario, el proceso accesa la variable poniendo un indicador para que el otro proceso espere cuando encuentra **mutexbegin.**

Mutexend: Se debe permitir el primer proceso que está esperando a entrar la sección crítica.

En el presente sistema, se puede poner variables comunes en el nodo padre. Las modificaciones de estas variables deberían realizarse en el nodo padre. Para cualquier tipo de acceso tanto para la modificación como para el usu de variables, necesita pasar mensajes desde el proceso hijo hasta el proceso padre. Si es uso de variable, el tipo de mensaje sólo indica al proceso padre que se

regrese esta variable con el mensaje de respuesta. Si es necesario cambiar la variable común desde un proceso hijo, éste debe pasar la fórmula y las variables locales necesarias al proceso padre. Cuando obtiene el nuevo valor de la variable común, regresa la viariable con el mensaje de repuesta hasta el proceso hijo.

Como resumen, el lenguaje que puede trabajar en el presente sistema necesita llevar las siguientes principales características, o sea el compilador del lenguaje podría tener las siguientes capacidades:

1. La concurrencia o distribución de procesos: Necesita tener la instruccione como COBEGIN para iniciar los procesos concurrentes o distribuidos.

2. Tener acceso de variables comunes en forma de exclusión mutua: Al empezar la distribución de procesos, necesita separar las variables comunes poniendose en el nodo padre. Además se debe utilizar un mecanismo para el acceso de las variables comunes en forma de exclusión mutua [Holt 83].

2.5 Características del sistema

Con la estructura lógica jerárquica, el SOD puede ejecutar las tareas en paralelo, balancear las cargas y ser transparente para usar los recursos del sistema tales como algunos archivos especiales (compilador, cargador, etc), bases de datos, impresoras, etc. El SOD también puede reconfigurar el sistema tanto físicamente como lógicamente y

tolerar las fallas.

2.5.1 Distribución de tareas

El compilador de un lenguaje puede tener llamadas de sistema operativo. En este sistema el compilador necesita decir numero de procesos que van a distribuir y el punto que va a partir los procesos, pero no es necesario saber en cuál procesador (o sea cuál nodo) va a hacer un proceso. La asignación de procesadores o nodos es transparente para el programador. Si un usuario ejecuta un programa y cuando llega el punto que hay procesos concurrentes o paralelos, el SOD avisa al presidente para obtener nodos colaboradores.

Se puede distribuir los procesos en los nodos del sistema dentro de una región. Un nodo de una región no puede ser colaborador de otra región, recuerda que entre ellos no pueden comunicarse. En caso de que no hay nodos disponibles, se ejecuta los procesos concurrentemente.

Como los procesos hacen las tareas, se refiere la distribución de tareas a la distribución de procesos.

2.5.1.1 El protocolo de la distribución

En la figura 2-9 se mostrará el protocolo de la distribución de los procesos.

Si al momento de solicitar un colaborador, el

presidente ve el sistema no es conveniente hacer las tareas en otro nodo; el programa se ejecutará concurrentemente en su nodo. Ser conveniente significa que el tiempo de la ejecución distriuido resulta menor que el de ejecución concurrente. Por los tiempos de comunicaciones y las cargas de los nodos colaboradores, la ejecucion distribuida puede ser más lenta o igual que la ejecución concurrente. Se hace la distribución de tareas sólo cuando el tiempo máximo de los procesos paralelos es menor que el tiempo total de los procesos si los hace concurrentemente. Si hay **n** procesos paralelos, SOD puede decidir que va a hacer la distribución checando si se cumple la siguiente desigualdad:

$$\text{Max } \{ T1, T2, \ldots Tn \} < \sum_{i=1}^{n} ti$$

donde Ti=tiempo de comunicación + tiempo de ejecución, ti = tiempo de ejecución.

Nota: El tiempo de comunicación incluye tiempo de pasar los códigos, variables comunes y respuestas (o dice resultados); mientras el tiempo de ejecución se considera el tiempo de la ejecución concurrente en el nodo procesos). Los tiempos son estimados.

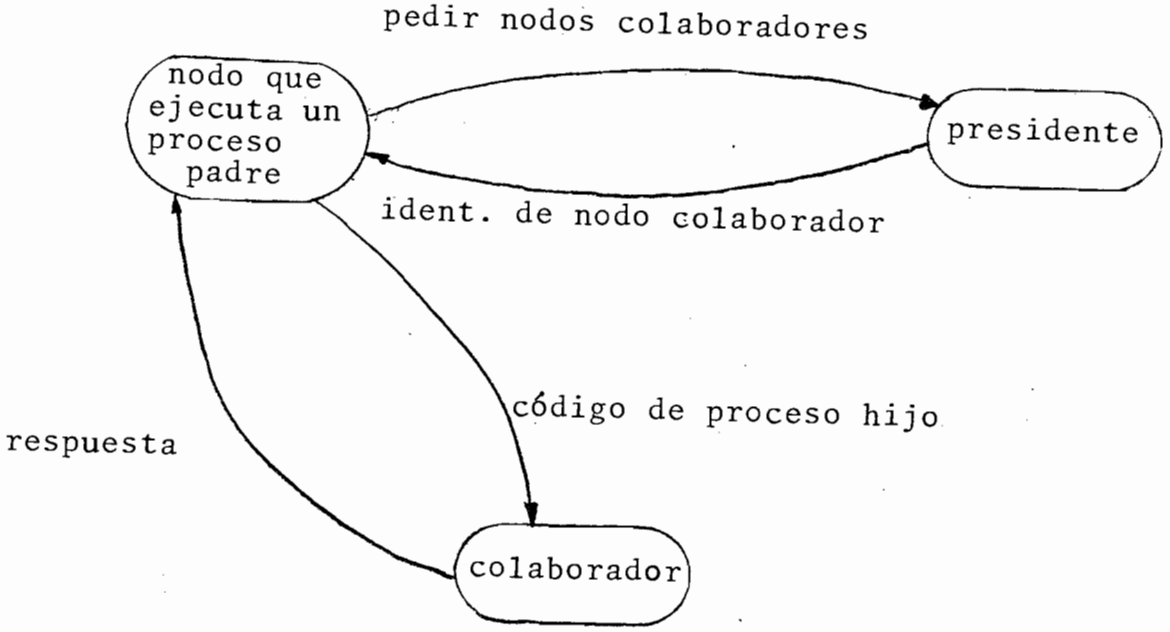


Figura 2-9: Protocolo de distribución de procesos

2.5.1.2 Ejemplo de la distribución de las tareas

En un sistema tiene cuatro nodos: 0, 1, 2 y 3 (ver la siguiente figura 2-10.a), y un programa tiene la estructura en la figuar 2-10.b. La distribución puede ser:

1. P0 se hace en el nodo 0;
2. P1 sigue estando en el nodo 0;
3. P2 está el nodo 2;

4. P3 se ejecuta hasta que los dos procesos P1 y P2 terminen.

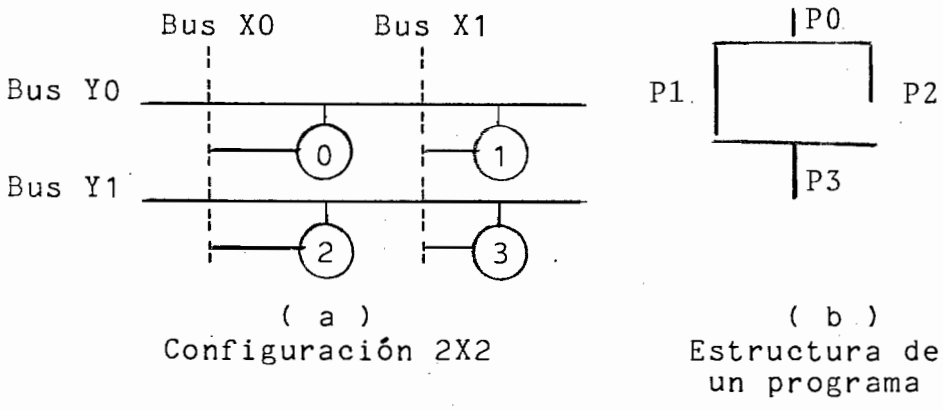


Figura 2-10: Ejemplo de distribución de tarea

Como el sistema es dinámico, el nodo colaborador no es fijo, depende de la carga en el momento que el presidente lo escoja. Cualquier proceso puede tener sus procesos hijos y la manera de obtener colaborador es igual. Los procesos hijos siempre regresan sus respuestas a su propio padre.

2.5.1.3 Balanceo de las cargas

Este sistema también puede hacer el balanceo de las cargas. Si un nodo está sobrecargado, el SOD pasa algunas tareas a los otros nodos. Si el nodo presidente tiene mucha carga, entonces es mejor que cambia otro presidente, o pasar sus cargas a otro nodo. Asi se puede hacer las operaciones

más rápido. El presidente siempre escoge colaborador al nodo con menos carga. Eso también es para lograr el balanceo de las cargas.

2.5.1.4 Procesos en modo background

Cuando se ejecuta una tarea sin intervenir por el usuario, esta tarea se llama **background.** Por ejemplo, un usauario quiere compilar un programa largo, pero no quería esperar el resultado sin hacer nada, entonces manda esta tarea como **background.** Ahora el usuario puede hacer otras operaciones mientras se ejecuta la compilación. Cuando la termina, se regresa el resultado al usuario o mientras manda algunos mensajes a la pantalla. En este sistema las tareas de **background** de un nodo pueden hacerse en otro nodo o concurrentemente en el mismo nodo. El resultado de la tarea **background** regresa al nodo iniciativo.

2.5.1.5 Uso de recursos globales

Para la petición de usar los recursos tal como impresora, solo puede escoger el nodo que tiene este recurso. Si un proceso solicita una impresora, el SOD checa primero si la tiene en su nodo. Si no, avisa al presidente para que le dice quién la tiene. Aunque la impresora está en el otro nodo, se manda a imprimir allá y regresa una señal de terminación de la impresión.

El otro recurso puede ser un compilador de lenguaje C, por ejemplo. Cuando el usuario quiere compilar un programa en C, aunque en este nodo no tiene el compilador, el SOD avisa al presidente y él lo busca en todos los nodos de la misma región. Si alguno de ellos tiene este compilador, el presidente avisa al nodo que quiere hacer la compilación y este nodo manda su programa al nodo que tiene el recurso para compilar. Si ningún nodo tiene el compilador de C, el presidente avisa al nodo que actualmente no puede compilar este programa.

2.5.2 Reconfiguración del sistema

Este sistema es reconfigurable. Dependiendo de los nodos en el sistema la configuración física y lógica cambia. Cuando un nuevo nodo quiere integrar al sistema o cuando un nodo quiere salir del sistema, el SOD reconfigura el sistema estableciendo nuevas rutas entre los nodos, dividiendo las regiones y obteniendo el presidente de cada región.

2.5.2.1 Incremento de nodo

Hay tres casos que necesita considerar para el incremento del sistema:

1. Cuando un nodo que **trabaja autónomamente requiere conectar con el sistema.**
2. **Generalmente, después de prender** la maquina se

necesita integrar el nuevo nodo con el sistema.

3. Despues de recuperar la falla, el nodo vuelva a funcionar para el sistemas.

Se va a utilizar un diagrama de flujo en la figura 2-11 para describir cómo incrementar el sistema.

Dependiendo de los estados anteriores se puede suceder las siguientes situaciones:

1. El nodo que quiere integrar al sistema es el único en su región, porque sus dos buses X y Y no pueden comunicar con los otros nodos del sistema. En el ejemplo de la figura 2-6 puede ser la integración del nodo 14.

2. El nuevo nodo tiene un vecino, por ejemplo, la integración del nodo 2, en el mismo ejemplo.

3. El nuevo nodo tiene dos vecinos en diferentes buses, pero con el mismo presidente; por ejemplo, el nuevo nodo puede ser nodo 0.

4. El nuevo nodo relaciona con dos regiones disjuntas. Eso puede ser la integración del nodo 3.

2.5.2.2 Despedida de un nodo

Hay dos casos que necesita llamar el modulo de despedida de nodo:

1. Cuando despide un nodo por usuario, o sea el usuario da un comando para salir del sistema.

2. Cuando falla la computadora de la tarea o la de comunicación. Aunque la despedida no es deseable, la situación para el sistema es igual que despedida de un nodo.

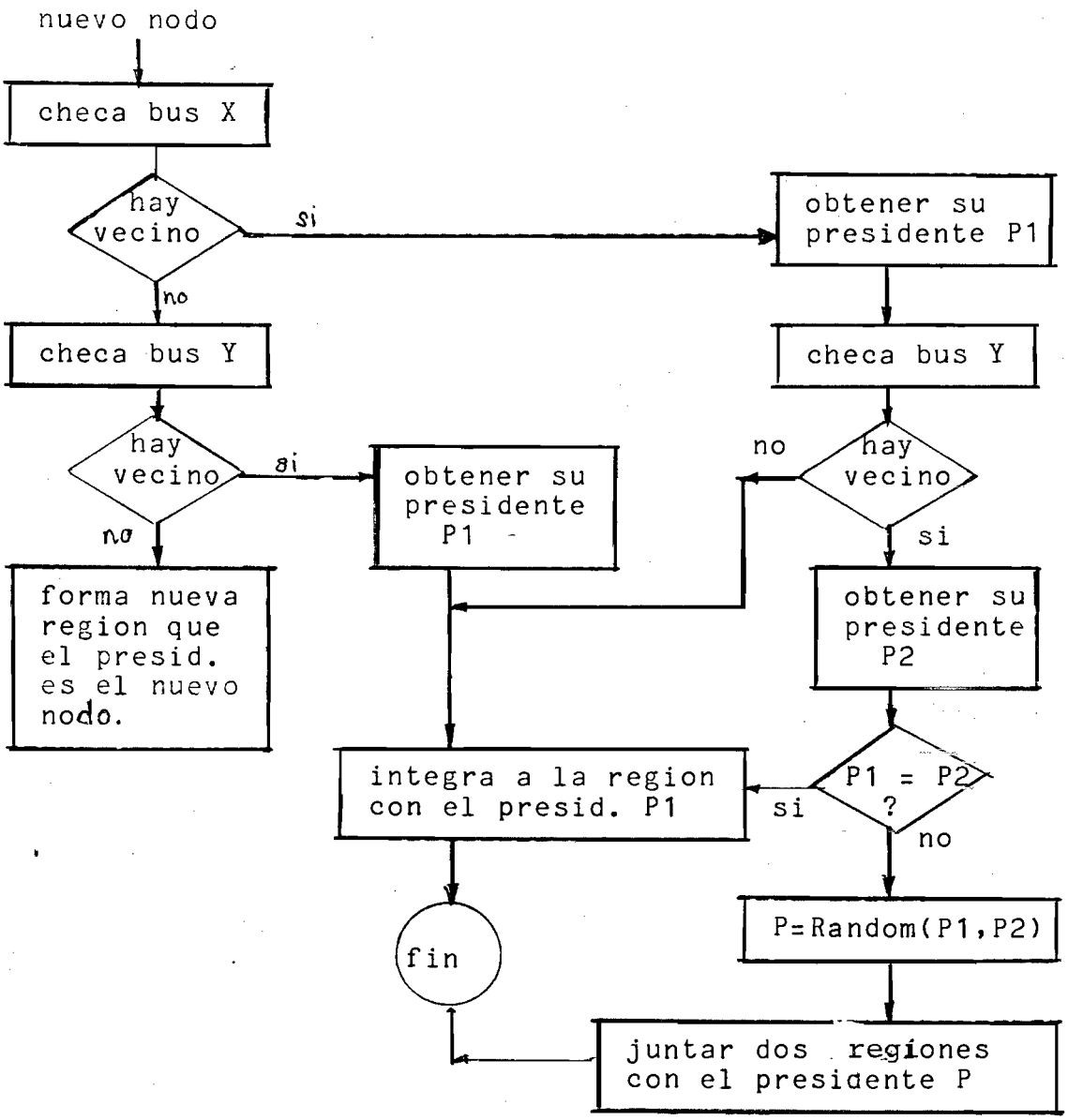


Figura 2-11: Diagrama de flujo para incremento del sistema

Antes de despidir un nodo, ademas de considerar a este nodo donde si esta trabajando algunos procesos, el SOD checa si el nodo es como presidente o como colaborador. Si

el nodo es un presidente, entonces se necesita pásar informaciones de presidente al candidato de presidente. Y este nodo obtiene nuevamente las rutas entre él y los ministros, y luego avisa a todos los ministros. En caso de despedida, si todavía hay algún proceso que está ejecutando, el SOD pregunta al usuario que si quiere terminar su trabajo o lo pasa al otro nodo para ejecutar. En caso de las fallas, se pasan códigos, variables, mensajes que están en el buffer de transmisión y recepción, etc; es decir todos los datos que se pasan al otro nodo que esta cerca.

Después de todos estos, el SOD quita este nodo desde el sistema y reconfigura el sistema. El SOD busca nuevas rutas entre nodos ministros y presidente, y entre el ministro y los colaboradores. El sistema forman nuevas regiones y tienen nuevos presidentes.

Falla de la computadora de tarea o de comunicación es desaparecer el nodo en el sistema excepto que la falla no es la intención de usuario.

2.5.3 Tolerancia a las fallas

Las posibilidades de fallas son inevitables y pueden suceder en cualquier momento. Se puede clasificar las fallas en la siguiente manera:

1. Falla recuperable

2. Falla grave
3. Falla catastrófica

2.5.3.1 Falla recuperable

Este tipo de falla ocurre en un recurso de un nodo o en un bus, por ejemplo, falla una computadora de tarea o la de comunicación, o el bus X1 del sistema.

El sistema puede tolerar a este tipo de falla; eso significa que el sistema puede funcionar bien aunque hay alguna falla recuperable, o sea tolerable. Cuando hay una falla de este tipo, el SOD reconfigura el sistema, pasa códigos, controles y datos necesarios a otro nodo. A continuación, en la figura 2-12 describirá por un diagrama de flujo para fallas y recuperaciones del sistema.

Cuando se encuentra este tipo de fallas, tiene manera para pasar informaciones del nodo que tiene la falla a otro nodo, mediante bus interno, alambre auxiliar, DMA, o sea utilizar algún remedio para pasar las informaciones aunque la forma es lenta. El sistema no pierde nada.

Después de la reconfiguración del sistema, el nodo que tiene falla, o el bus que falla, ya no existe en el sistema. Pero cuando recupera las fallas, el sistema reconsidera automáticamente a ellos como nodos del sistema, o buses del sistema.

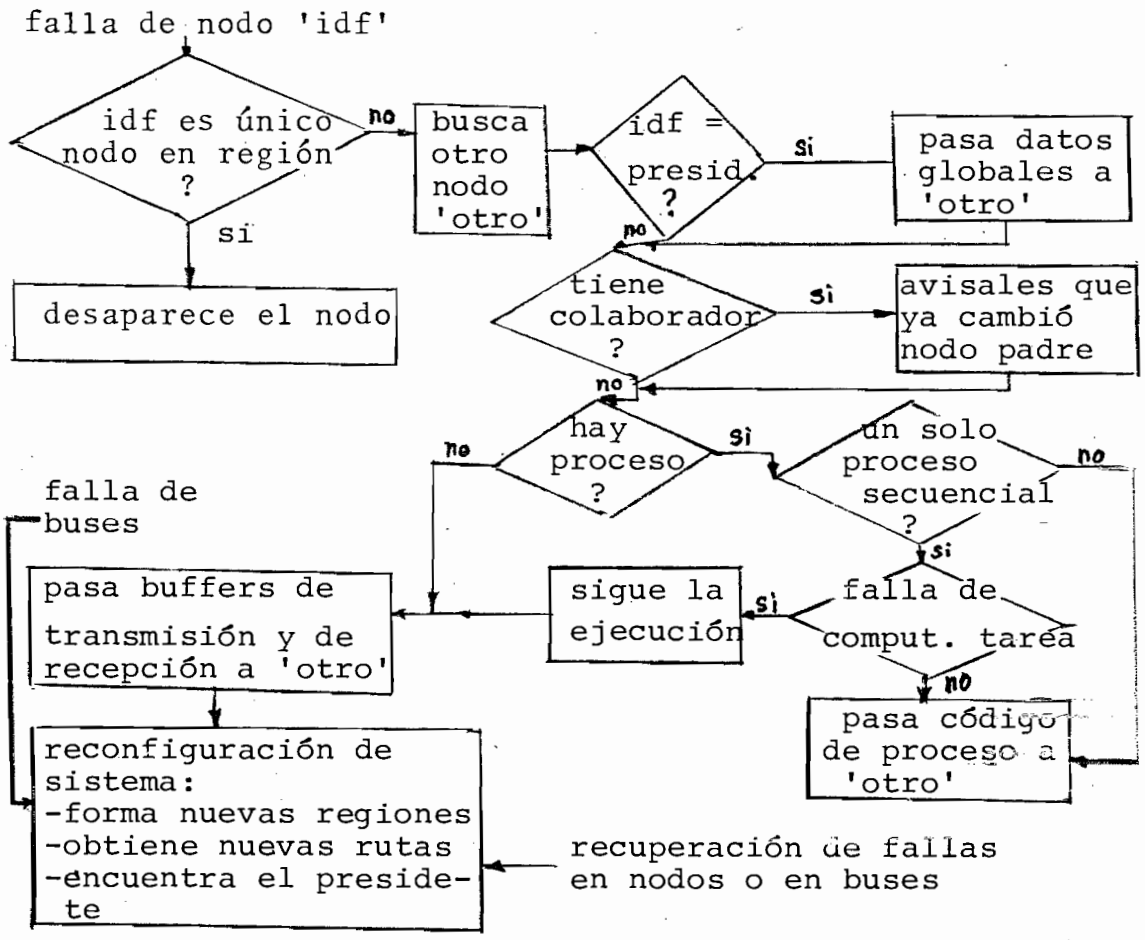


Figura 2-12: Diagrama de flujo para fallas y recuperaciones

2.5.3.2 Falla grave

Hay muchas fallas se consideran como fallas graves porque solo se puede recuperar una parte de las informaciones y puede dejar inconsistencia para le ejecución de algún proceso. Cuando varias fallas recuperables ocurren simultáneamente, se puede tener una falla grave. Por

ejemplo, falla de dos computadoras del nodo, causa la aislación del nodo con otros nodos del sistema. Entonces no hay manera de pasar las informaciones necesarias a otro nodo. Si este nodo hay un proceso de otro nodo, el otro necesita reinicializar su ejecución. El presidente en el SOD puede saber que el sistema perdió un nodo y necesita reconfigurar el sistema. Los nodos periódicamente manda las informaciones globales a su presidente. Si el presidentes no recibe la señal de algún nodo durante un intervalo de tiempo, se considera que este nodo hay falla. Como la noticia de falla no llega al presidente inmediatamente de la falla, se puede hacer que el sistema entre una situación inconsistente. Eso es grave! Generalmente las informaciones globales en el nodo que tiene falla ya no puede recuperar, pero el sistema sigue funcionando con las informaciones de estado anterior.

Otro ejemplo de falla grave puede ser falla de dos buses conectados a un nodo presidente. Esta falla puede detectarse por un nodo ministro, el cual no recibe la respuesta de presidente después de mandar informaciones globales. Entonces este nodo avisa al nodo candidato. Desde este momento, el candidato vuelva a ser presidentes, asigna un nuevo candidato y reconfigura el sistema. Como el candidato recibe informaciones globales periódicamente desde

el presidente, el candidato no tiene las informaciones identicas que el presidente. El candidato solo tiene informaciones de estado anterior y el sistema reconfigura con estos datos. El sistema pierda algunas informaciones.

2.5.3.3 Falla catastrófica

Para cualquier sistema puede suceder fallas catastróficas; y es muy difícil de recuperar este tipo de falla. Por ejemplo, falla de presidente y candidato de presidente en el mismo tiempo, o falla un nodo que es el único punto de relacionar dos regiones, hace el sistema entre la situación inconsistente para muchos nodos, y la confunción de relaciones entre nodos, y entre nodos y buses, porque se destruyó la estructura lógica.

2.6 Estructura de datos

Para cada nodo del sistema tiene sus estructuras de datos. Aunque un nodo no es presidente, debe tener tablas reservadas para ser presidente, porque el SOD es un sistema dinámica y no se sabe cuando uno va a ser el presidente. Por lo menos un nodo es nodo ministro y puede ser presidente o colaborador. A continuación se describirá las estructuras de datos para el ministro, el colaborador y el presidente.

2.6.1 En el nodo presidente

Hay una tabla de presidente que se contiene los recursos de los ministros bajo su control. Por ejemplo en la figura 2-6, si el presidente 4 tiene los ministros 1, 4, y 5, su tabla de presidente puede ser la siguiente:

Tabla de Presidente

lista de nodos ministros	cargas (%)	impresora	memoria	disk disp.
1	10	si	20k	100k
4	35	no	10k	38k
5	20	no	50k	500k

2.6.2 En el nodo ministro

Cualquier nodo del sistema es un nodo ministro, se necesita tener las siguientes datos: la identificación de su presidente, la de candidato, las de colaboradores, la ruta entre el nodo y el presidente, y las rutas entre el presidente y los colaboradores. Además hay un dato contiene la carga del nodo. La siguiente tabla es un ejemplo para la tabla de ruta.

Tabla de Rutas

(para el nodo 1)

nodo destino	nodos intermedios
1	-1
4	5
5	-1

Los restos datos son los siguientes:

carga	10 %
nodo presidente	1
nodo candidato	4
nodos colaboradores	5

Nota:

1. "Nodo intermenio" es -1 significa que no hay nodo intermedio, y se puede pasar informaciones directamente por el bus;
2. "Nodos colaboradores" es -1, significa que no tiene colaborador y supuestamente no hay rutas entre ellos.

Esencialmente, se conserva las rutas con los colaboradores, aunque estos ya no comparten los trabajos. En el mismo ejemplo, la tabla de ruta aparece la ruta entre 1 y 4. Pero el nodo 4 ya no es colaborador del nodo 1 y solo el nodo 5 es.

2.7 Modelo del sistema

Para implementar el SOD que ha descrito anteriormente, se utiliza el modelo de procesos con comunicación entre procesos por pasar mensajes (Eso ha mencionado en el capítulo I).

Cada proceso tiene uno o mas puertos que son creados por si mismo tanto para recibir como para transferir mensajes. Los puertos son estructuras de datos como monitores. Los puertos se residen fuera de procesos. Las comunicaciones se realizan por pasar mensajes desde el puerto de transmisión en un proceso hasta el de recepción del otro. Cuando un proceso manda un mensaje a un puerto lleno, se bloquea el proceso hasta algun mensaje salir desde el puerto. Lo mismo, cuando un proceso quiere recibir un mensaje desde un puerto vacío de recepción, el proceso bloquea hasta algun mensaje llega al puerto. Cuando dos procesos no estan en el mismo nodo, un proceso debe mandar mensajes a su puerto y pasarlos al puerto del otro proceso, luego este proceso recibe mensajes desde su puerto (ver la figura 2-13). Si dos procesos están en el mismo nodo, se puede usar un puerto común para el puerto de transmisión de un proceso y el de recepción del otro. Este puerto se llama canal.

Con este modelo, el sistema será más modularizado y

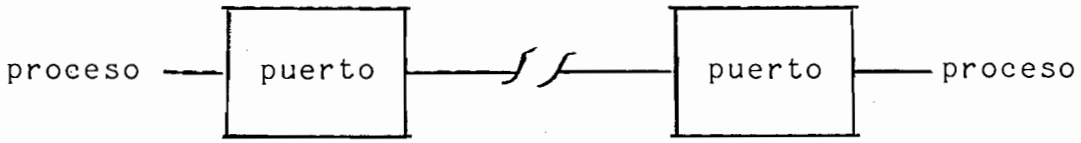


Figure 2-13: Comunicación entre procesos

fácil de manejar los datos y controles del sistema. Sobre todo, para un sistema distribuido es muy flexible y útil. En cualquier momento, se puede agregar un proceso y se establece puertos para las comunicaciones con los otros procesos. Similar, se puede quitar un proceso sin afectar o modificar funcionamiento de otros procesos.

En la figura 2-14, se mostrará los flujos de datos y de controles entre principales procesos residentes en cada nodo.

Los circulos representan procesos, las flechas indican sentidos de accesos. El proceso CRT maneja un terminal de usuario y CLI actúa como interprete de lenguaje de comandos para generar nuevos trabajos. CLI lee lineas de entrada desde CRT mediante un buffer de linea obtener comandos.

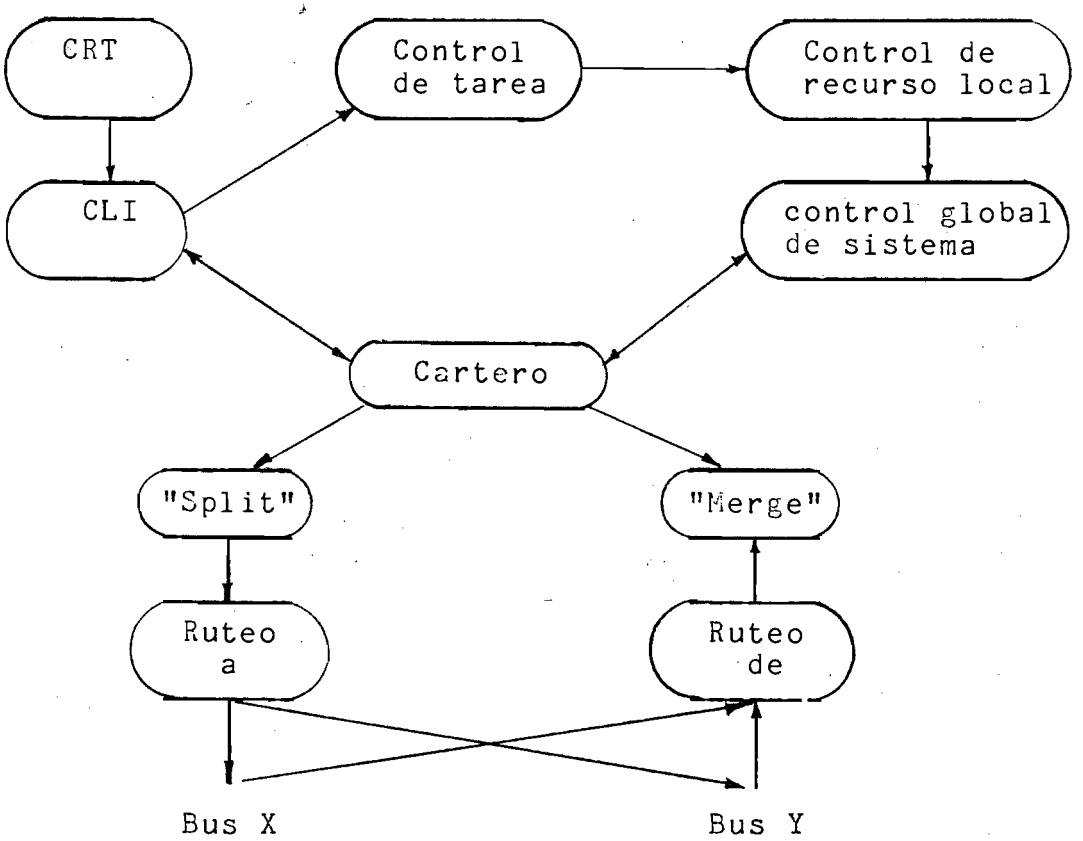


Figure 2-14: Modelo del sistema

Muchos comandos causan CLI cargar uno o más programas desde el sistema de archivo a la memoria disponible. Una vez cargado programa, las tareas corren hasta que se termine por terminación normal, requerimiento de usuario, o intervención del sistema. Las tareas pueden generar otras tareas y puede transmitir otros comandos del sistema. CLI también pueden recibir comandos desde **cartero.**

Cuando una tarea necesita producir otra tarea, o sea

hay otros procesos concurrentes o paralelos, el proceso de **control de tareas** debe mandar un mensaje al proceso de **control** de **recursos** para checar si existe suficientes recursos tales como espacio para poner códigos, para datos, para mensajes (o puertos); dispositivos de E/S; y ver si llega el límite de tareas que pueden soportar por un nodo, o sea ver si está sobrecargado el nodo. Para cualquier de estos casos no cumplidos, se necesita ayuda del proceso de **control global del sistema** para buscar colaborador en el sistema.

Este proceso pregunta al presidente cuál nodo puede hacer la tarea producida mediante un mensaje desde el nodo de trabajo al nodo de presidente. El formato de mensaje debe incluir tipo de tarea, el cual se puede distinguir por el presidente cuál recurso se requiere la tarea. El presidente hace estos trabajos en el proceso de **control global de sistema.**

El **cartero** es accesible desde cualquier proceso dentro del nodo. Este proceso recibe mensajes desde diferentes procesos y los pasa a donde quieren según indicación de mensaje recibido. Se transfiere mensajes desde puertos al proceso split o merge. Los procesos split y merge hace repartición y conjunción de mensaje respectivamente, solo cuando el mensaje es muy largo.

3. Simulación del sistema

3.1 Introducción

Antes de construir un sistema tan grande y tan complicado como el sistema distribuido descrito en el capítulo II, es deseable tener la simulación del mismo. Por los resultados de la simulación, se puede entender mejor el funcionamiento del sistema y se puede generar buenas sugerencias para el diseño del sistema real.

El sistema operativo distribuido (SOD) es la parte esencial para controlar el sistema distribuido. Así que la simulación del SOD es el fundamental para la construcción del sistema distribuido.

En este capítulo, va a describir la simulación del SOD con el diseño descrito en el capítulo anterior.

Se pretende que la simulacíon del SOD pronorcione datos más claros sobre el funcionamiento del SOD en el sistema distribuido de tipo matricial.

3.2 Qué es la simulación ?

La simulacíon es la representacíon de ciertas características del comportamiento de un sistema abstracto o físico a través del comportamiento de otro sistema. El objeto de la simulación es obtener la medida experimental y

predecir el comportamiento del sistema. Gracias a la simulacion, se puede obtener muchas experiencias y conocimientos del sistema real y descubrir los problemas que pueda suceder en el mundo real. La simulacíon es primer paso para que funcione bien un sistema grande. Pero debe recordar que los resultados de simulacíon serán aproximados más que los resultados precisos de un sistema real.

3.2.1 El proceso de la simulacion

Generalmente, el proceso de la simulación incluye los siguientes pasos:

1. Definicíon de experimento,
2. Modelacion,
3. Implementacíon de computacion,
4. Validacíon y
5. Datos de salida.

3.2.1.1 Definicion de expermento

La simulación es aplicable a sistemas dinámicos cuyos estados cambian con el tiempo. A este sistema se llama sistema simulable. Los datos disponibles de entrada y los datos deseables de salida son ingredientes importantes para definir la simulación.

3.2.1.2 Modelación

Existen muchos tipos de modelos para la simulación. Se clasifican ellos por las características de los sistemas. Pero el criterio de clasificación más usado es de la propiedad del cambio de las variables en el sistema: continuas o discretas.

El modelo de variable-continua representa los sistemas con el cambio de variables continuamente. Estos sistemas se describen generalmente por las expresiones matemáticas. El modelo de variable-discreta es para los sistemas cuyos variables cambian disretamente. Por ejemplo, un sistema cambia sus estados por asignación y desasignación de algún recurso.

3.2.1.3 Implementación de la computación

Una vez que se tiene la definición y el modelo de la simulación, se necesita implementarla. Eso puede realizarse por algún lenguaje de programación en la computadora.

3.2.1.4 Validación

La validación es el aspecto más complejo del proceso de la simulación. Generalmente, la validación refiere a la estimación sobre el grado de validad de los resultados de la simulación. El proceso de la simulación requiere un criterio de validad, porque los resultados de la simulación son

aproximados a los resultados precisos reales y pueden ser validos para unos propósitos y ser invalidos para otros.

3.2.1.5 Datos de salida

Los datos obtenidos desde la simulación en modelo de variable-discreta caen en tres clasificaciones básicas: datos de tiempo, datos de utilización de recursos y de cola, y datos históricos.

Los datos de tiempo incluye las estadísticas del sistema o tiempo de evento; por ejemplo, tiempo para ejecutar procesos o números de usuarios en el sistema por cada unidad de tiempo. Esto es para medir la ejecución dinámica del sistema.

Los datos de utilización de recursos y de cola incluye números de llamadas, tiempo usado de un recurso, tiempo de espera, longitud de una cola, etc. Estos se usan para determinar el flujo del sistema y balancear los recursos.

Generalmente, los datos históricos representan la trayectoria de la simulación completa o parcial. Eso es para analizar el modelo y examinar las condiciones transitorias en el sistema de la simulación [Ralston 83].

3.2.2 Esquema general de la simulación

En la figura 3-1 se puede ver el esquema general de la simulación.

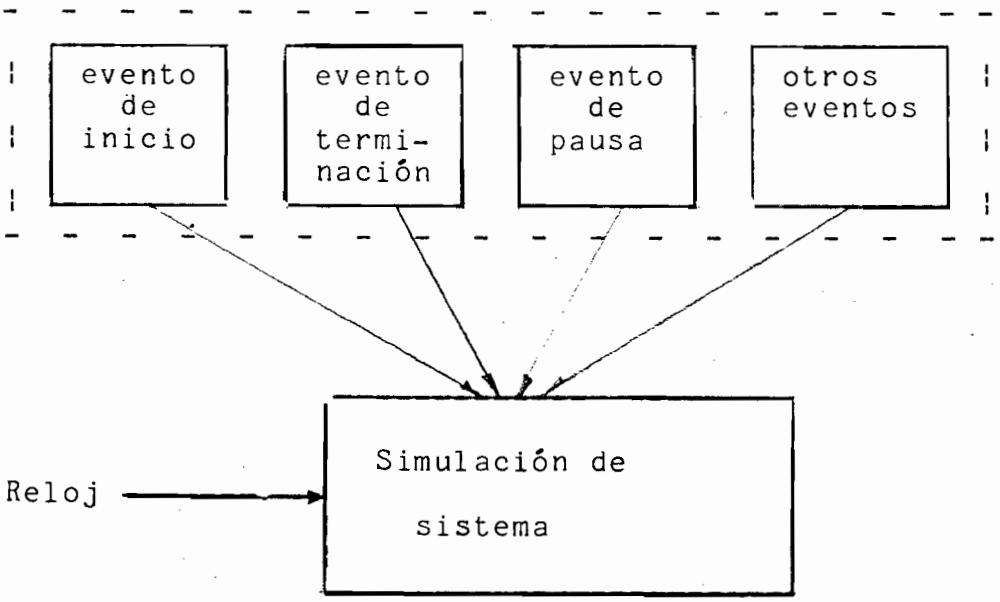


Figura 3-1: El esquema general de la simulación

Una simulación empieza con los datos iniciales y luego opera sobre el modelo de la misma. Mientras el tiempo pasa, el sistema cambia de acuerdo con su definición. El generador de eventos hace que el sistema cambie el estado que pueda inicializar la simulación, detenerla, terminarla o generar otros eventos.

3.3 Simulación del SOD

Un sistema de cómputo es dinámico. A veces el funcionamiento del sistema es muy complicado. La simulación de este sistema provee una idea experimental para la investigación y es una forma económica de estimar el comportamiento del sistema. Sobre todo, para un sistema distribuido es necesario tenerla. El sistema distribuido es simulable. Por ejemplo, en cualquier momento, puede integrar un nuevo nodo, desaparecer otro, ejecutar un procesos, etc.

El SOD es una parte para controlar el sistema distribuido. La simulación del SOD debe tener primero el modelo de sistema distribuido. Sobre éste mismo, simula los comportamientos del SOD. El SOD es un programa para controlar el sistema; cualquier incremento de un nodo, despedida de un nodo, inicio de un proceso, etc. hace el sistema distribuido cambiar la configuración física y/o lógica, o los estados del sistema.

La simulación de este sistema usa el modelo de variable-discreta, porque el cambio de estados es discreto. Por ejemplo, se generan los procesos aleatoriamente. Durante un intervalo de tiempo puede inicializar muchos procesos. O a veces, después de un rato, no aparece ningún proceso. Las fallas también ocurren discretamente.

Esta simulación se realizó en la minicomputadora ONYX utilizando el lenguaje C.

Como el sistema operativo distribuido puede construirse basando en un sistema operativo centralizado y agregando el manejo de controles de distribución, la presente simulacíon del SOD se esfuerza sólo en la parte de la distribucíon de las tareas, tolerancia a fallas y sus recuperaciones en el sistema distribuido de tipo matricial como ha mencionado en el capítulo II. Por ejemplo, se trata de describir cómo distribuir procesos, cómo tolerar a las fallas, y cómo reconfigurar el sistema. Todos los movimientos del sistema son transparentes para los usuarios. Los recursos que están en el sistema son como si fuera de cada usuario. El usuario no nota que se falta un archivo ni sabe que su capacidad de procesamiento se incrementó o no.

Como otras simulaciones, ésta también tiene el esquema de la simulación tal como ha descrito en la figura 3-1. En la figura 3-2 se mostrará un diagrama de flujo para esta simulación. La simulacíon tiene la iniciacíon de estados, el generador de eventos, las operaciones (ejecución de procesos, transmisión y recepción de mensajes), modificacíon de estados, y impresión de reportes para presentar los resultados.

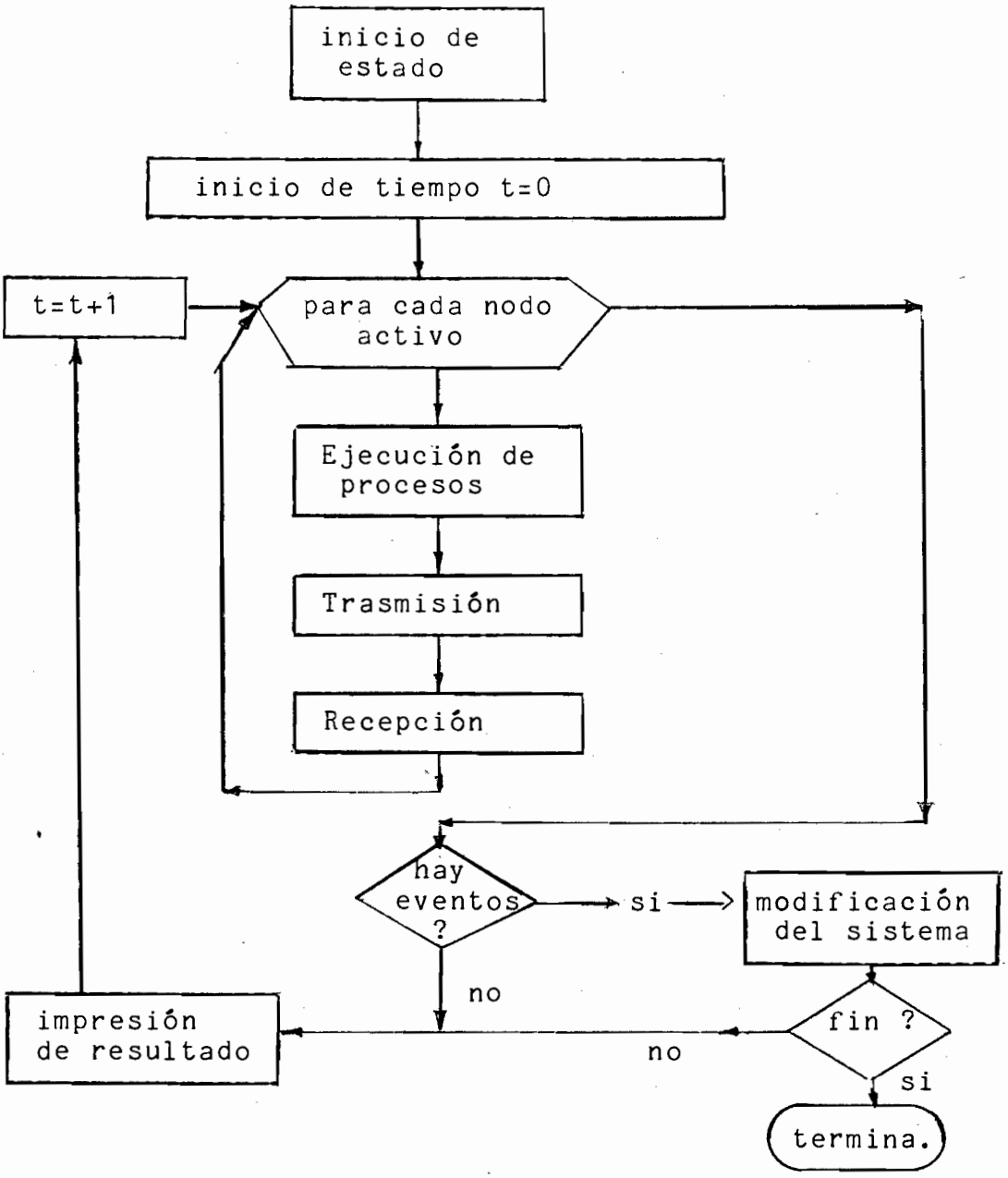


Figura 3-2: Diagrama de flujo para la simulación

Viendo la figura 3-2, después de inicializar las

variables del sistema y el tiempo, en cada nodo activo del sistema distribuido se empieza a hacer los tres procedimientos en cada unidad de tiempo:

Ejecucíon de procesos;

Transmisíon de mensaje;

Recepcíon de mensaje.

El usuario puede en cualquier momento cambiar estado del sistema, tales como incremento de un nuevo nodo al sistema, causar una falla de computadora de comunicación, ó entrar un programa en un nodo, etc. Sólo se puede terminar la simulacion entrando la seccíon de cambio de estados. Antes de incrementar una unidad de tiempo, se imprime en un archivo los resultados de esta unidad del sistema, en donde está la trayectoria del comportamiento del sistema desde el tiempo inicial hasta el tiempo determinado.

3.3.1 Programa para la simulación

Una acción que va a simular es ejecutar un programa. Para entender un programa por la simulacíon se necesita tener una representacíon sintaxis y semántica, y una representacion estructurada. En las siguientes subsecciones se va explicar sobre este programa.

3.3.1.1 Sintaxis de programa

El sintaxis del programa tiene la siguiente forma:

programa = (p proceso)

proceso = (1 tiempo) | (proceso^{n}) | (m proceso^{m})

p = un caracter indica que evento es un programa

tiempo = número enteros positivo > 0

m = número entero positivo > 1

= numero de procesos paralelos

n = número entero positivo > 1

= número de procesos secuenciales

$$\text{proceso}^{n} = \underbrace{\text{proceso proceso} \ldots \text{proceso}}_{n}$$

= procesos secuenciales

$$(\text{m proceso}^{m}) = (\text{m } \underbrace{\text{proceso proceso} \quad \ldots \quad \text{proceso}}_{m})$$

= procesos paralelos

(1 tiempo) = proceso simple

A la simulacíon sólo interesa el tiempo de la ejecucíon de un proceso y el punto en que empieza los procesos paralelos. En el siguiente ejemplo mostrará un programa y su significado del programa estará en la figura

3-3.

(p ((1 2)(2 ((1 4)(3 (1 9)(1 15)(1 8))(1 2)) (1 14))(1 4)))

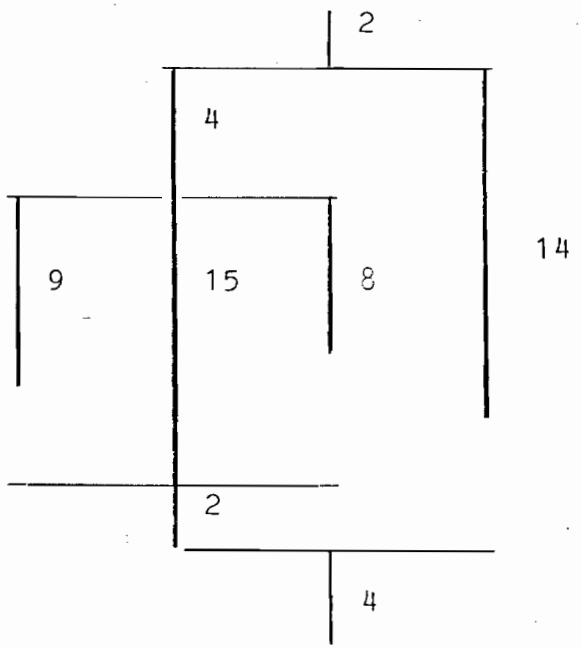


Figura 3-3: Un ejemplo de programa

Los números antes de parentesis derecha son los tiempos de procesos. Se puede ver este programa en tres partes. El primer parte ejecuta las instrucciones en 2 unidades de tiempo del proceso (Es diferente entre la unidad de tiempo de proceso y unidad de tiempo de la simulación que va a mencionar en 3.3.2). La tercera parte ejecuta en 4 unidades de tiempo de proceso. En la segunda parte hay dos

procesos paralelos. Uno necesita 14 unidades; y el otro en donde después de 4 unidades de tiempo, genera tres procesos paralelos que usan 9, 15 y 8 unidades de tiempo, respectivamente; y al final ejecuta 2 unidades. Si este programa ejecuta concurrentemente, va necesitar 58 unidades de tiempo de proceso. Pero si se realiza la ejecución paralelamenta, el tiempo de ejecución será 33.

3.3.1.2 Representación estructurada

Internamente se construye un arbol para representar el programa. Con respecto a la sintaxis del programa, hay siguientes definiciones:

Proceso simple (1 tiempo):
se representa por una celda simple (Ver la figura 3-4.a)

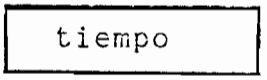
Procesos secuenciales ($proceso^{n}$):
se muestra en la figura 3-4.b.

Procesos paralelos (M proceso):
tienen la estructura descrita en la figura 3-4.c.

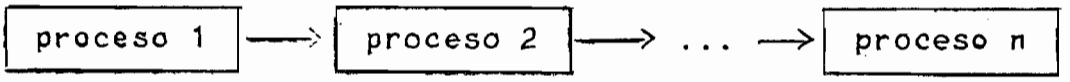
La estructura interna del program en el ejemplo anterior se muestra en la figura 3-5.

3.3.2 Reloj

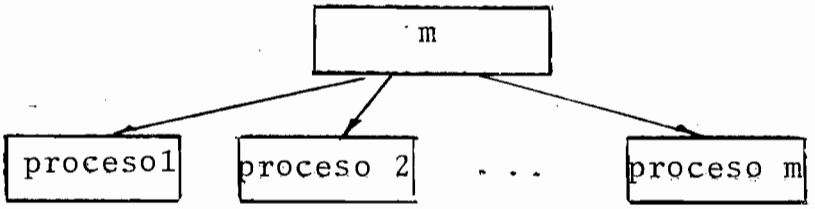
El tiempo incrementa desde 0 hasta un tiempo determinado. Se considera una unidad de tiempo como la medida mínima individible. En esta simulación se toma el



a. Proceso simple



b. Procesos secuenciales



c. Procesos paralelos

Figura 3-4: Estructura del programa

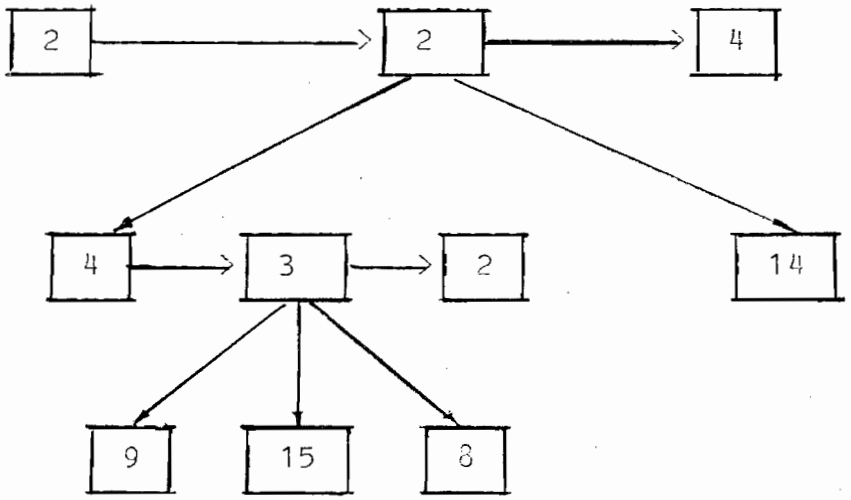


Figura 3-5: Estructura interna del programa

tiempo de transferencia de un mensaje como una unidad de tiempo. Se supone que el tiempo de transferencia de un mensaje entre cualquier par de nodos es igual. Se avanza el reloj por cada unidad de tiempo para la simulación.

El cambio de sistema por algún evento sólo ocurre en el principio de la unidad de tiempo, o sea antes de hacer las operaciones en el primer nodo activo modifica el estado del sistema. En la vida real, en los nodos activos se hace las operaciones paralelamente. Pero en la simulación con un lenguaje secuencial necesita una secuencia para pasar nodo por nodo. Como una unidad de tiempo es individible, debería aceptar los eventos después de pasar todos los nodos activos.

La simulación puede ejecutar en dos modos: automático o interactivo.

3.3.2.1 Modo automático

En el modo automático se avanza el reloj autimáticamente, hasta que pare por una interrupción, el reloj se detenga. Si se genera el evento en un nodo que no es el último (de acuerdo que las identificaciones de los nodos tienen un orden) para incrementar el reloj, no se hace caso a la interrupción.

3.3.2.2 Modo interactivo

El modo interactivo es continuar el reloj por teclear <Return>. Después de pasar todos los nodos, se avanza una unidad de tiempo.

3.3.3 Condición inicial

Se define la persona que corre el programa de la simulacíon como el usuario. Cuando un usuario quiere ejecutar la simulacion, primero necesita definir los siguientes datos:

1. El tamaño del sistema distribuido de tipo matricial: el número máximo de los nodos horizontales, y el número máximo de los nodos verticales.
2. El número de nodos activos.
3. Si inicialmente existe nodos activos, indica el nodo presidente.

4. El número máximo de los procesos que pueden ejecutar concurrentemente en un nodo.

5. El factor entre la unidad de tiempo de proceso y la de tiempo de la simulacion que es tiempo de transferencia. O sea una unidad de tiempo para el proceso es cuantas veces de tiempo de la transferencia.

6. El periodo para transferir informaciones globales desde los nodos ministros al nodo presidente.

Después de meter los datos iniciales, aparece en la pantalla la configuración del sistema.

Por ejemplo, hay siguientes datos de entrada:

Tamaño horizontal : 4

Tamaño vertical : 2

Nodos activos : 5

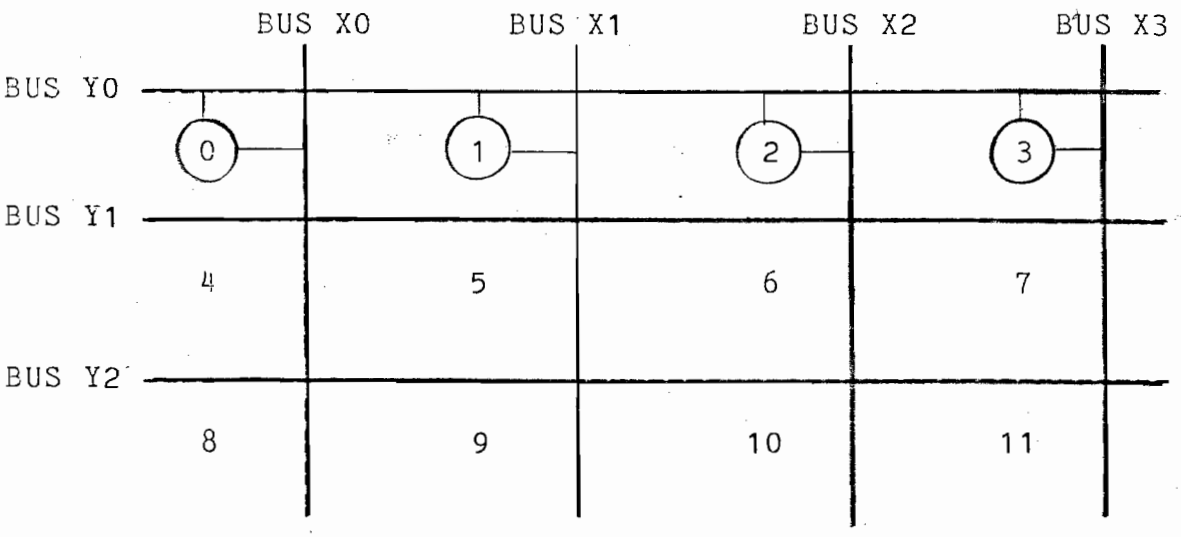
Nodo presidente : 3

Factor de tiempo : 4

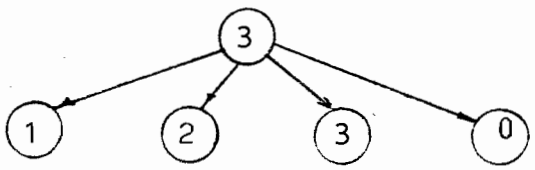
Periodo : 20

Entonces en la pantalla aparece la siguiente figura 3-6.a y la estructura lógica está en la figura 3-6.b.

El factor de tiempo es 4, significa que 4 veces de tiempo de la transferencia de un mensaje es un tiempo unitario del proceso. En otra palabra, la unidad de tiempo de procesos es 4 veces la unidad de tiempo de la simulación.



a. La congiguración física del ejemplo.



b. La estructura lógica.

Figura 3-6: Ejemplo

En el programa para la simulación, los tiempos de los procesos cuentan por la unidad de tiempo de procesos. En este ejemplo si un proceso cuenta por nanosegundo y el tiempo de transferencia es 1/4 ns, o sea 0.25 ns es para transferir un mensaje.

3.3.4 Operaciones

Como la descripción del diagrama de flujo en la figura 3-2, en todos los nodos activos por cada unidad de

tiempo hace tres operaciones: ejecución de procesos, transmisión y recepción de mensajes. Ahora se va a describir uno por uno de estas operaciones.

3.3.4.1 Ejecución de los procesos

En cada unidad de tiempo, neceista checar los procesos que mete por el programa de la simulación (ver la 3.3.1) si ya llegan los tiempos de terminación o si deben generar procesos paralelos. Cuando se empieza a ejecutar un proceso, se pone el tiempo de terminación del proceso según las cargas actuales del nodo. Se puede expresar el tiempo de terminación en la siguiente manera:

$$T_{fin} = T_{actual} + T_{necesario} \times (\text{número de procesos})$$

Si en el tiempo T_i ya están ejecutando dos procesos en un nodo y va a empezar el otro que necesita T_n para la ejecución, entonces el tiempo de terminación correspondiente al nuevo proceso será: $T_{fin} = T_i + T_n \times 3$. Si no terminan los primeros dos procesos, este nuevo va a terminar en el tiempo T_{fin}. Pero si antes de llegar a este tiempo, se acaba el primer proceso o el segundo, necesita modificar el tiempo de terminación de los proceso quedados; porque ahora sólo existe dos procesos en el nodo. Entonces el nuevo tiempo de terminación del tercer proceso es igual a

$$T_{actual} + (T_{fin} - T_{actual})/3 \times 2.$$

Todos los procesos necesistan modificar los tiempos de terminación cuando viene o sale un proceso.

La ejecución de procesos que ha mencionado en la figura 3-2 es comparar el tiempo que se indica en el reloj con los tiempos de teriminación de los procesos.

El SOD hace la distribución cuando encuentra el punto que va a generar procesos paralelos.

3.3.4.2 Transmisión y Recepción

En cada unidad de tiempo, se hace transmisión y recepción de mensajes. Cada vez, transmite un mensaje o recibe uno en caso de que haya mensajes en buffers correspondientes. Los buffers de mensajes son como colas circulares (vea la siguiente figura 3-7). Si necesita transferir un mensaje, pero ya llega el fin de buffer y hay lugares libres en el principio, entonces se pone mensaje en el principio. En el caso de que no haya lugar disponible para un mensaje, se bloquea el proceso.

Todos los mensajes tiene el siguiente formato:

Destino Fuente Ruta Tipo Origen Contenido

donde

Destino: es el nodo al cual quiere mandar un mensaje

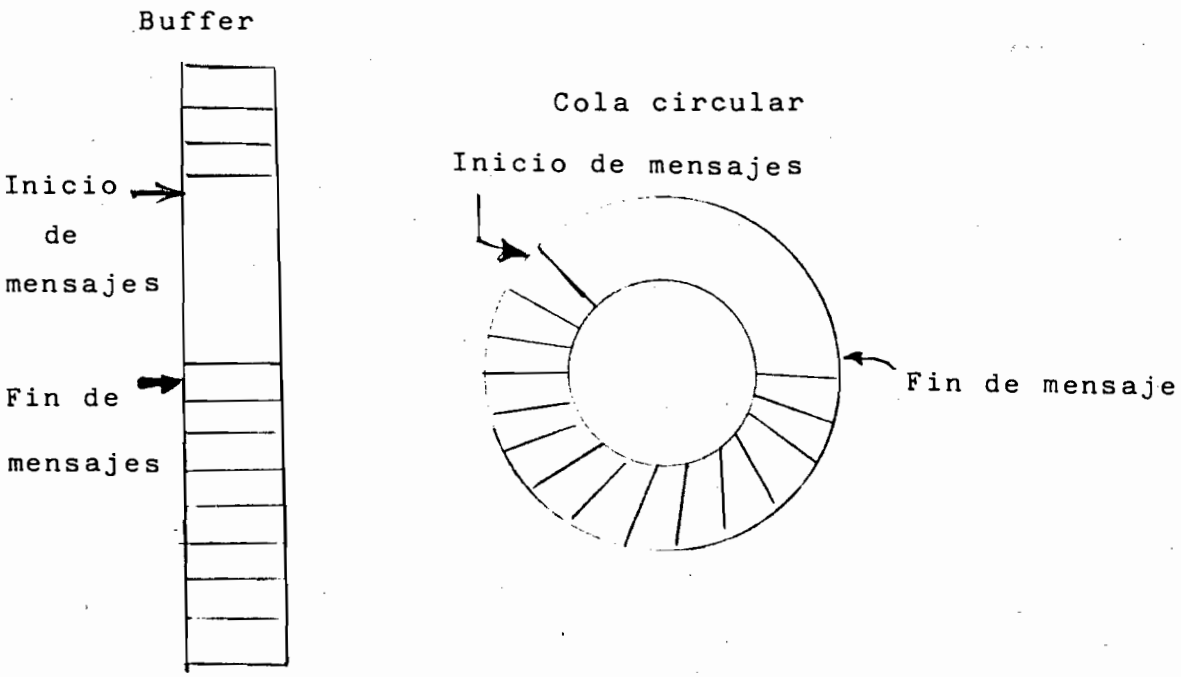


Figura 3-7: Buffer de transmisíon o de recepción

el nodo origen.

Fuente: es el nodo que manda un mensaje en una transferencia. No necesariamente es el nodo origen, porque puede ser un nodo intermedio.

Original: es el nodo que manda el mensaje.

Ruta: contiene nodos intermedios para llegar al nodo destino. Pueden ser los siguientes casos:

1. No hay nodos intermedios: -
2. Solo hay un nodo intermedio i1: i1,-
3. Hay n nodos intermedios: i1, i2, ... in,-

El signo - es para indicar el fin de nodos intermedios.

Tipo y Contenido:

1. Pedir al presidente el número de colaboradores que se requiere.

2. Es una respuesta de peticíon de mensaje de tipo 1. Solo se lo manda por el presidente para avisar al nodo cuál va a ser colaborador. Es posible que no haya nodo disponible como colaborador, en tal caso indica con el signo -.

3. Mandar codigos de proceso distribuido al nodo colaborador. Puede ser un proceso con más procesos hijos. Necesita indicar apuntador de proceso padre para que se pueda regresar a su propio proceso padre con su posición.

4. Regresar la respuesta del proceso hijo al proceso padre.

5. Este mensaje se manda periódicamente al presidente para reportar las informaciones del nodo, por ejemplo, la carga total en el nodo.

Si no se puede transmitir un mensaje a un nodo destino, lo transmite a los nodos intermedio; y despues lo pasa hasta el nodo destino.

3.3.5 Generador de eventos

Se puede generar eventos en dos maneras: por programa o interactivamente. Para esta simulación las dos maneras son de forma secuencial. La situación de generar eventos simultaneamente es un caso más complicado.

En el programa de eventos contienen todos los

eventos que van a pasar por el tiempo. La sintaxis tiene la siguiente forma:

El programa de eventos tiene la sigueinte sintaxis:

programa_de_evento = (evento ^{N})

evento = incremento_de_nodo ¦
despedida_de_nodo ¦
falla ¦
recuperacion_de_falla ¦
programa ¦
nada

incremento_de_nodo = (i id_nodo)

despedida_de_nodo = (d id_nodo)

falla = (f tipo)

recuperación_de_falla = (r tipo)

programa = (p proceso)

nada = (n tiempo)

i, d, f, r, p, n = caracteres reservados para indicar tipo de evento correspondiente.

tipo = (1 id_nodo) ¦ (2 id_nodo) ¦
(3 id_busX) ¦ (4 id_busY)

Nota: tipo 1: falla de computadora de tarea
tipo 2: falla de computadora de comunicación

tipo 3: falla de bus X

tipo 4: falla de bus Y

id_nodo = identificación de nodo

id_busX = identificación de bus X

id_busY = identificación de bus Y

tiempo = número para indicar intervalo de tiempo que no hace nada.

proceso = proceso simple ¦ procesos secuenciales ¦ procesos paralelos

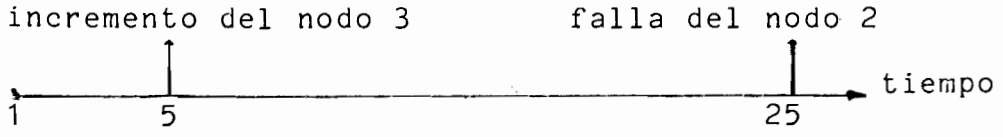
N = número > 1

Un ejemplo del programa de eventos puede ser lo siguiente:

```
((n 5) (i 3) (n 20) (f (1 2)))
```

Se puede ejecutar este programa de evento antes de avanzar una unidad de tiempo. El programa va a generar una tabla de eventos en donde guarda todas las informaciones de eventos junto con el tiempo inicio de ese evento. En la figura 3-8 muestra la tabla para el ejemplo anterior.

Esta tabla puede explicar con la siguiente forma:



t	eventos
1	nada, 5
5	incremento del nodo 3
5	nada, 20
25	falla del nodo 2

Figura 3-8: Tabla de eventos

Los dos eventos (incremento y falla) suceden en el tiempo 5 y 25 respectivamente y en los otros tiempos no pasan nada.

El otro modo de generar eventos es interactivo. Con una interrupción hace el sistema entrar un menú de eventos para cambiar el estado del sistema. Aquí genera los eventos por medio interactivo. Sólo se modifica el estado del sistema hasta el siguiente unidad de tiempo.

El menú consiste de las siguientes opciones:

1. Incremento de nodo
2. Despedida de nodo
3. Falla
4. Recuperación de falla
5. Entrada de un programa
6. Cambio de modo de ejecución

7. Impresión de estado del sistema
8. Continuación de reloj
9. Terminación de la simulación

Durante el cambio de estados, la simulación no acepta la interrupción. Después de modificar el estado del sistema se aparece otra vez el menú para que pueda modificar otros estados; así sucesivamente hasta que teclee continuación del reloj o terminación de la simulación.

En el menú anterior, los primeros cinco eventos van a afectar al SOD. El SOD hace la reconfiguración del sistema cuando encuentra el evento del incremento del nodo o el de la despedida del nodo; el SOD reajusta el sistema cuando hay fallas o recuperación en el sistema; y el SOD distribuye las tareas palarelas cuando hay un program con paralelismo.

Los últimos cuatro eventos en el menú son para la simulación.

A continuación, explicará cada uno de ellos.

3.3.5.1 Incremento del nodo

Cuando uno selecciona este evento, debe decir a la simulador la identificacíon del nodo que quiere integrar al sistema. No se acepta el nodo que ya existe en el sistema

para el incremento.

Luego el SOD empieza a reconfigurar el sistema según lo que describe en el capítulo II.

3.3.5.2 Despedida del nodo

Igual que el caso anterior, el usuarios necesita dar la identificación del nodo que va a despedir. No se puede ser el nodo que no existe en los nodos activos del sistema. Con la identificación del nodo, el SOD quita el nodo desde su región correspondiente y luego acomoda los otros nodos del sistema.

3.3.5.3 Fallas

Cuando selecciona el evento tres, se aparece otro menú que es:

Tipos de Falla:

1. Nodo de la computadora de comunicación
2. Nodo de la computadora de tarea
3. Bus X
4. Bus Y

Al seleccionar el tipo de la falla, necesita indicar la identificación del nodo o del bus. No se consideran las fallas en nodos pasivos ni nodos que todavía no han recuperado.

3.3.5.4 Recuperación de fallas

El menú que va a mostrar después de selecionar el evento 4 es semejante al menú de las fallas. En vez de la falla aparece el menú con los mismos datos, pero para la recupearación de falla. La recuperacíon de una falla en un nodo o en un bus debe coincidir con la falla que ha sucedido.

3.3.5.5 Entrada del programa

En esta opción hay que indicar el nodo que va a ejecutar el programa y el modo de leer el programa: por la pantalla, o por un archivo. El programa ha mencionado en 3.3.1.

3.3.5.6 Modo de ejecución

Durante la ejecucíon de simulacíon puede cambiar el modo de ejecucion al modo interactivo o automático.

3.3.5.7 Impresión de estados del sistema

Se puede imprimir el estado global del sistema o de los nodos con un menú que se describe en la siguiente forma:

Menú de Impreción

1: Tiempo de la simulación

2: Estado de buses

3: Regiones

4: Nodo

5: Configuración

Se puede ver el tiempo de la simulación con la opción 1. El segundo opción muestra el estado de los buses, el cual indica cuáles buses están funcionado. En la opción 3, se ilustra todas regiones que existen actualmente en el sistema, en las cuales se indican el nodo presidente, los nodos ministros, y rutas entre ellos, o sea la configuración lógica. La opción es 5 para graficar la configuración actual. Generalmente, después de la reconfiguración del sistema, requiere ver la configuración que es la opción 5. En la opción 4 se necesita indicar el nodo que quiere ver. Aquí se muestra la carga del nodo, los nodos colaboradores junto con las rutas, y mensajes en el buffer de transmisión y de recepción.

3.3.5.8 Continuación de reloj

Este evento reactiva el reloj para continuar la simulación. Generalmente, después de selecionar los primeros siete eventos, se necesita generar este evento excepto que el usuario quiere terminar la simulación.

3.3.5.9 Terminación de la simulación

Este evento es para terminar la simulación. Después de eso, ya se puede imprimir el archivo de la trayectoria de

la simulación.

3.3.6 Impresión de resultados

Los resultados de la simulación guardan en un archivo que contienen todos los comportaminetos de la simulación. Se muestra cuándo falló un nodo junto con el tipo de la falla, cuándo la recuperó, cuándo entró un programa y distribuyó los procesos, las cargas en cada momento, y cuánto duró el programa, etc.

3.3.6.1 Descripción del formato de resultados (salidas)

El primer atributo del resultado es el tiempo que se indica dos medidas: uno incrementa por la unidad de tiempo de procesos, y el otro es por la unidad de tiempo de la simulación (el tiempo de transferencia de un mensaje para esta simulación). Luego vienen los resultados de los nodos del sistema que tiene los siguientes atributos para cada nodo:

Cargas: (CARGAS) local (L) -- número de procesos que trabajan para este nodo.

remoto (R)-- número de procesos que trabajan para otros nodos.

total (T)-- suma de estos dos números anteriores.

Colaborador (COLBS): En este campo indica el nodo que está colaborando.

Mensaje : (MENSAJES) destino (D) -- el nodo que va a trasmitir el mensaje junto con su tipo (T).

fuente (F) -- el nodo que mandó el mensaje

junto con su tipo (T).

Evento (ENT): indica entrada y salida de un programa, inicio de la falla y de la recuperación de falla junto con su tipo.

Cuando pasa un mensaje en un bus, debe indicar en el campo de mensaje desde cuál nodo empieza hasta cuál otro termina y el tipo de mensaje. Si entre los dos nodos hay un nodo intermedio, entonces el mensaje que va al nodo intermedio se distingue con el tipo de mensaje más el tipo correspondiente con una constante.

3.3.6.2 Ejemplo

El ejmplo que se va a mostrar en la figura 3-9 es la trayectoria de la simulación del programa que está en la página 70 utilizando la configuración física de la figura 3-10.

Ahora, las dos unidades de tiempos son iguales. En el tiempo 7, el nodo 0 tiene un proceso local y el nodo 1 tiene un proceso remoto que es para el nodo 0. En el tiempo 5, pasa un mensaje de tipo 3 desde el nodo 0 hasta el nodo 1. En el tiempo 32, desde el nodo 1 transfiere un mensaje tipo 5 hasta el nodo intermedio 0 para que en la siguiente unidad de tiempo lo pase hasta el nodo destino 2. Ahora la

constante es 20, entonces el tipo de mensaje para el nodo intermedio es 25.

En el tiempo 0, en el nodo 0 entró un programa indicado por un asterisco, y en el tiempo 32 se lo abacó por la misma manera. En el tiempo 7 falló la computadora de tarea del nodo 3. F2 indica la falla de tipo 2 (ver el menú en la página 85).

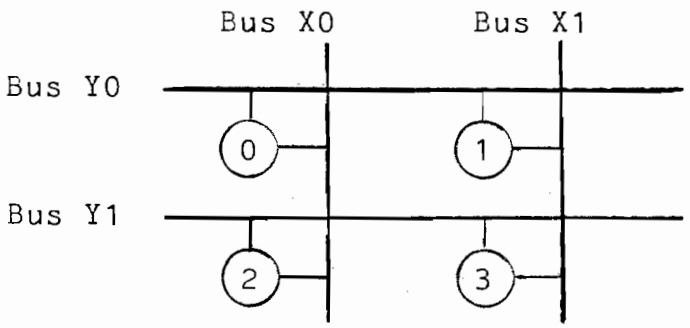


Figura 3-10: Configuración del ejemplo

Las rutinas para simular el sistema distribuido son apartes de las rutinas del SOD. Entonces la simulación no es difícil para aplicar en otro tipo de sistema distribuido.

Figura 3-9: Resultado de la simulacion

4. Resultados

En este capítulo, se va a describir unos ejemplos que se obtuvo con la simulación. Dado el sistema inicial, por los eventos que se ocurren, se puede ver el sistema cómo se cambia. Con el mismo simulador, también se puede obtener unos resultados como sugerencias para construir un sistema de cómputo. Hay muchas cosas que se puede probar, pero aquí sólo se va a mostrar:

Prueba simple: es la ejecución de un programa distribuido sin suceder otros eventos; la falla de un nodo o de un bus; el incremento o despedida de un nodo cuando no se afectan los procesos.

Prueba complicado:
es el caso cuando en un nodo está ejecutando un programa, falla un nodo o ejecuta otro programa.

4.1 Pruebas simples

4.1.1 Ejecución de programa distribuido

A continuación, va a ver los resultados del programa mencionado en el capitulo III. Este programa distribuido para la simulación tiene la siguiente expresión:

(p ((1 2)(2 ((1 4)(3 (1 9)(1 15)(1 8))(1 2))(1 14)) (1 4)))

y su representación se mostrará en la figura 4-1.

Este programa va a correr en varios sistemas para ver como cambian los tiempos de ejecucion. Hay muchos

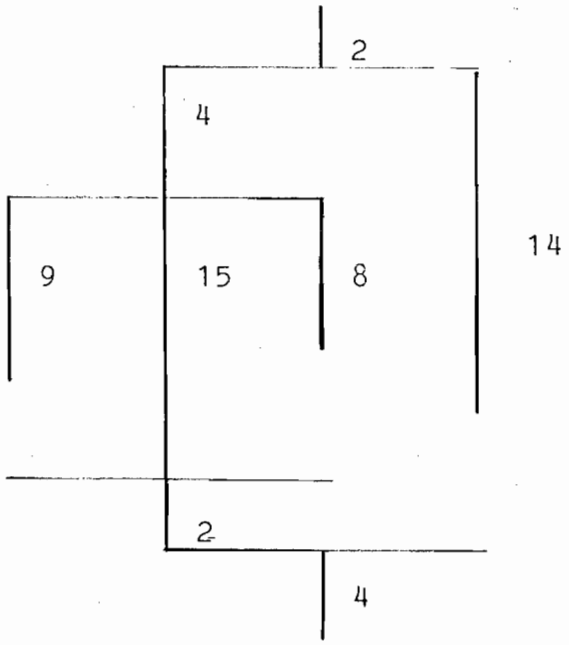


Figura 4-1: Representación del programa

factores que afectan al tiempo de ejecución tanto la posición del presidente, y de los colaboradores como la del programa mismo. Si un programa no tiene paralelismo, las tareas no se pueden distribuir aunque haya muchos nodos disponibles para ser colaboradores.

4.1.1.1 Nodos colaboradores vs. tiempo de ejecución

La primera prueba va a hacer en un sistema de 2X2 (ver la siguiente figura 4-2). El presidente es el nodo 0 y también se va a correr este programa en el 0. Las relaciones de nodos activos y tiempos necesarios para la ejecución se

mostrará en la figura 4-3.

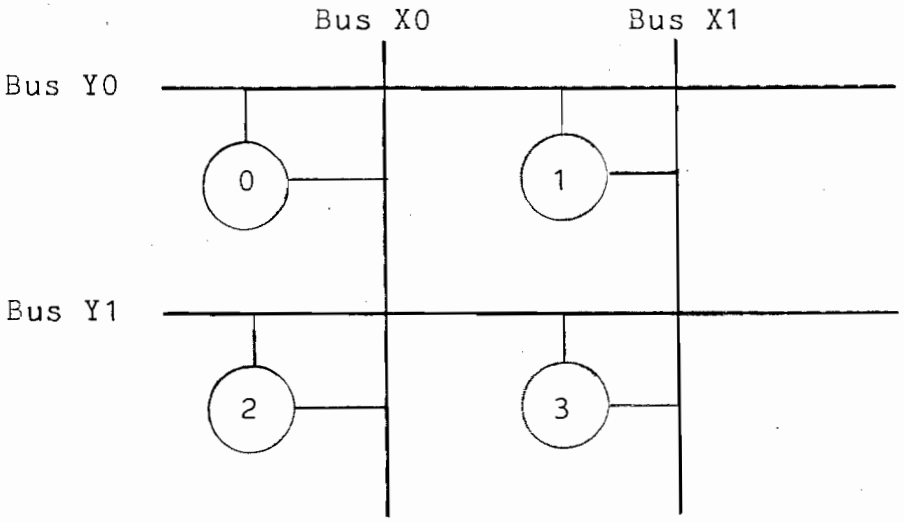


Figura 4-2: Sistema de 2X2

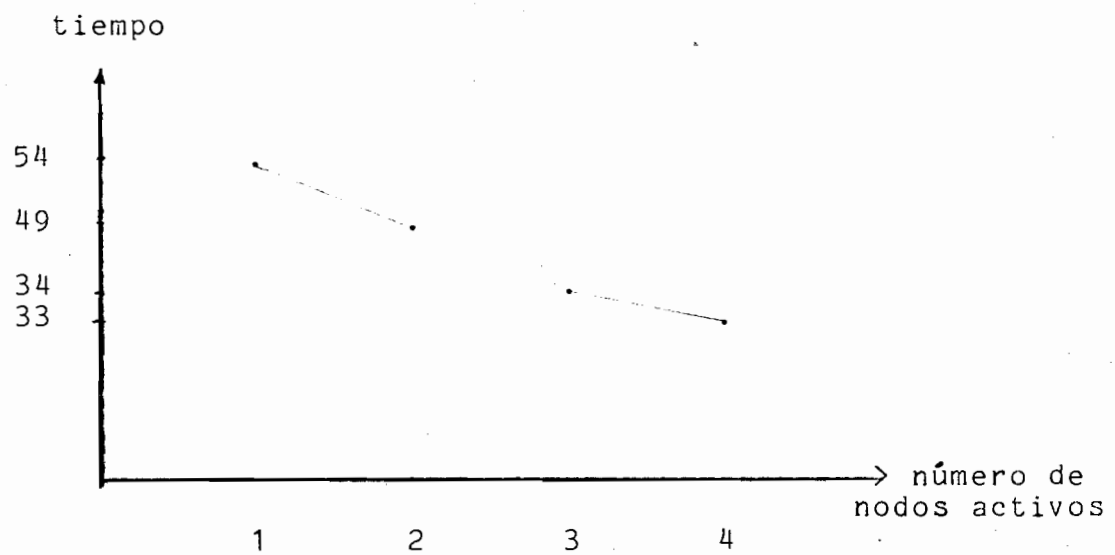


Figura 4-3: Tiempos de ejecución del progarma

Cuando está ejecutando el programa en un solo nodo, el tiempo es 54. Cuando está en 4 nodos, el tiempo es 33. El tiempo de ejecución disminuye cuando se incrementa nodos colaboradores en el sistema. Pero si hay más nodos activos en este sistema y no cambia los nodos colaboradores, no se va a disminuir el tiempo porque el número máximo de procesos paralelos es 4. Claro, si hay más de 4 nodos que pueden colaborar los trabajos, a la hora de solicitar los colaboradores, el SOD escogen los que más conveniente a su ejecución.

4.1.1.2 Posición de colaboradores vs. tiempo de ejecución

En otro sistema de 3X2 hay 4 nodos 0,1,3,5; el tiempo de ejecución del mismo programa resulta 35 (ver la siguiente figura 4-4), se necesita dos unidades de tiempo más que el caso anterior (cuando hay 4 nodo 0,1,2,3 en el sistema 2X2).

Las posiciones de los colaboradores también afectan el tiempo de la ejecución. Si el nodo colaborador tienen muchos nodos intermedios para comunicar con su nodo ministro, el tiempo total de la comunicación aumentará considerando que los tiempos de comunicaciones entre dos nodos siempre van a ser iguales. Para este ejemplo dos nodos colaboradores (3 y 5) tienen el nodo intermedios que es 1. En cambio el ejemplo anterior solo el nodo 3 tiene nodo

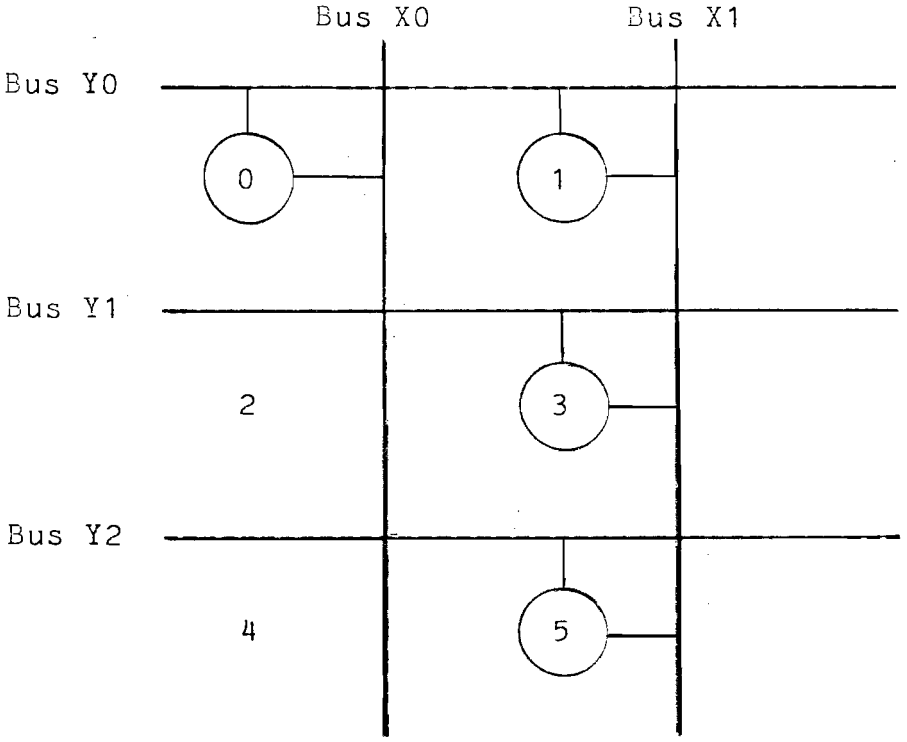


Figura 4-4: Sistema de 2X3

intermedio 1.

A continuación (ver la figura 4-5) se mostrará los resultados de otras pruebas. Se puede ver los tiempos de ejecución como se comportan cuando cambian la posición de presidente y la del nodo que corre el programa.

La asignación de los ejemplos anteriores usan el siguiente algoritmo:

1. Busca un nodo con menor carga que el nodo que

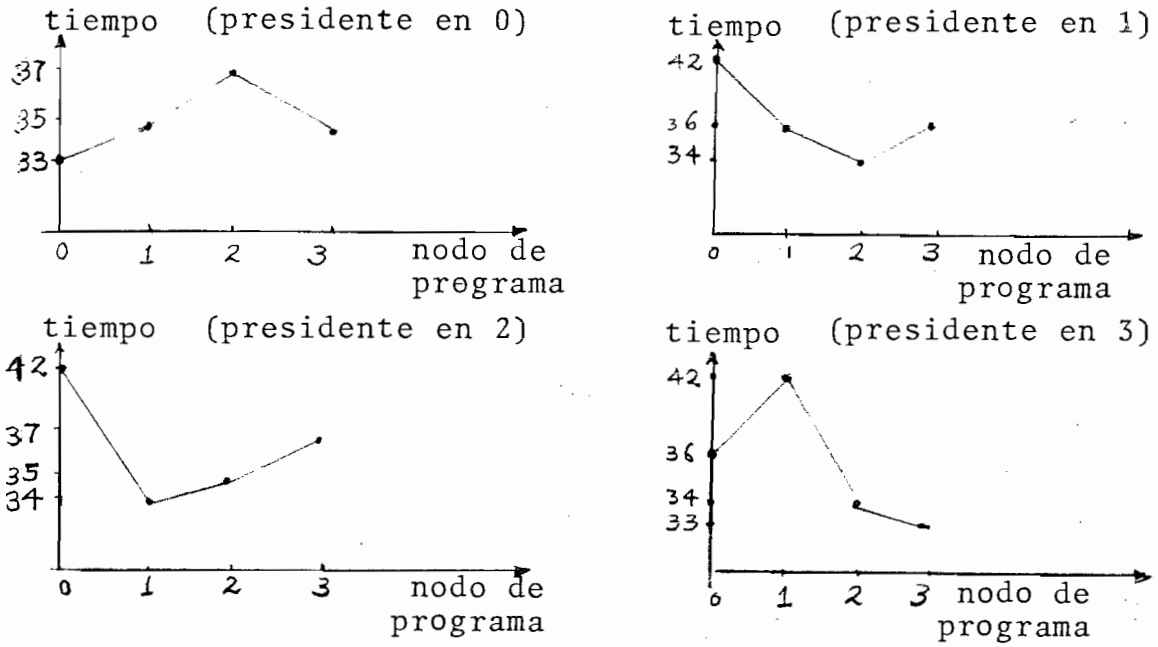


Figura 4-5: Tiempos del programa

solicita colaborador.

2. Si el total de el tiempo estimado de la ejecución en otro nodo y el tiempo de la comunicación es menor que el tiempo de la ejecución concurrente en el mismo nodo, va a hacer el 3. Al contrario, va al 4.

3. Si existe más de un nodo con iguales cargas, el nodo con la identificación menor va a ser este nodo colaborador.

4. No existe nodo colaborador.

Por este algoritmo, aunque el nodo que corre el **programa y el presidente tiene la misma relación (nodo** de programa es 1 y presidente es 0 contra nodo de programa es 0 y el presidente es 1), con la diferencia de orden de asignación, el tiempo resulta diferente. Para el primer caso

tiempo es 35 y el segundo es 42. En la página 98 y 98 ilustrarán las trayectorias de estados cuando están ejecutndo el programa en estos dos casos.

Con este resultado, se puede obtener otro algoritmo de asignación de nodos colaboradores. Si un proceso pide n colaboradores, una vez obtenido los nodos asigna el proceso con más tiempo de ejecucion estimado a un nodo con menos procesos concurrentes (o sea con menor carga) y con menos nodos intermedios para la comunicación. Por ejemplo, cuando el nodo que inicializa el programa es 0 y el presidente es 1 en el ejemplo anterior; si obtuvo los nodos colaboradores por el primer algoritmo, el tiempo de ejecución es 42. Por el nuevo algoritmo el proceso que usa 15 unidad de tiempos va a ejecutar en el nodo 3 en vez del nodo 2 y el procesos con 8 va a ejecutar en el nodo 2, porque este proceso con 15 necesita más tiempo que el otro proceso con 8, y el nodo 3 usa menos tiempo de comunicación con el nodo 1 que el nodo 2 (el nodo 2 necesita nodo intermedio 3 para comunicar con el nodo 1). Entonces, el tiempo de ejecución de este programa va a resultar menor que 42. Ahora es 34 unidades de tiempo.

Ya existen dos algoritmos de asignar los nodos colaboradores, pero no se puede decir cuál es mejor que el otro. Los tiempo de ejecución de procesos son estimados y las situaciones se pueden cambiar en cualquier momento.

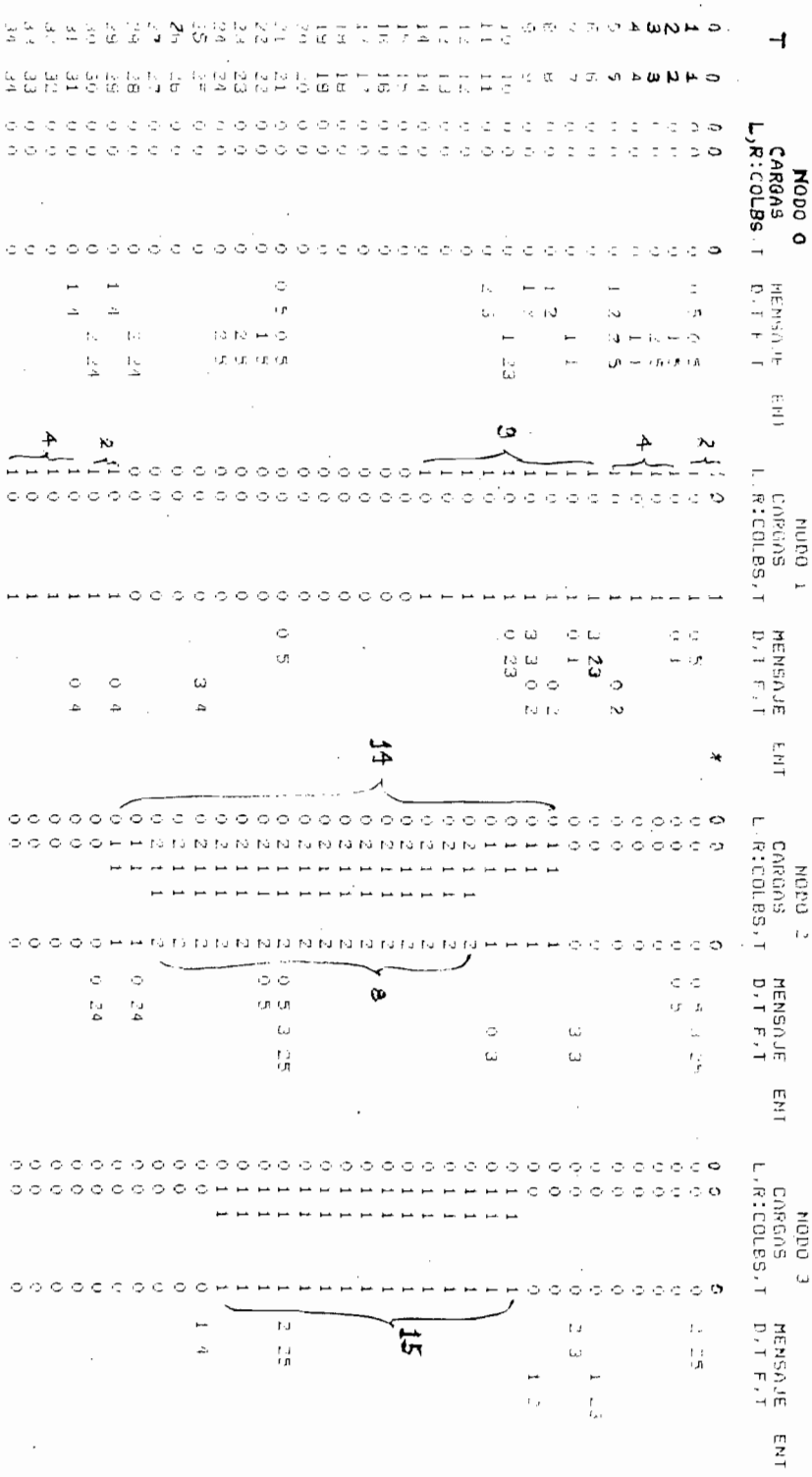


Figura 4-6: El presidente en 0 y el programa en 1

Figura 4-7: El presidente en 1 y el programa en 0

Entonces con el segundo algoritmo que se parece mas sofisticado puede resultar con más tiempo de ejecución. Por ejemplo, si despues de obtener los colaboradores, aumento la carga en algun colaborador, entonces causa el retraso de respuestas del proceso hijo. Como la situación cambia después de hacer la decisión para la asignación , solamente se puede decir un algoritmo es mejor que el otro relativamente.

4.1.1.3 Mensajes vs. tiempo de ejecución

Por otra parte, los mensajes en buffers de transmisión y de recepción en un nodo también afecta al tiempo de ejecución de un proceso. Como en una unidad de tiempo solamente transmite o recibe un mensaje, los otros mensajes se ponen en cola, el tiempo de espera para sdlicitar nodos colaboradores o para recibir códigos de procesos hijos en colaboradores, puede tardar el tiempo de ejecución. Número de mensajes en los buffers es aleatorio. Para los ejemplos anteriores, cada 20 unidad de tiempo manda mensajes globales desde cada nodo al presidente como se ha descripto en el capitulo anterior. Si este tiempo disminuye, el tiempo de ejecución va a incrementar porque aumenta tiempo de espera para los mensajes.

Para los ejemplos anteriores, se considera que el tiempo de transferir mensajes desde cuelquier nodo al otro

es igual que el tiempo que se cuenta para los procesos, o sea la unidad del tiempo de transmisión es igual a unidad de tiempo de procesos. En la siguiente figura mostrará los tiempos de ejecución del mismo programa cuando el tiempo de transmisión es 1/2, 1/4, 1/6, 1/8 y 1/10 de la unidad de tiempo de procesos. El programa se corre en el nodo 0, el presidente es 0 y el sistema es de 2X2 con nodos activos 0, 1, 2 y 3.

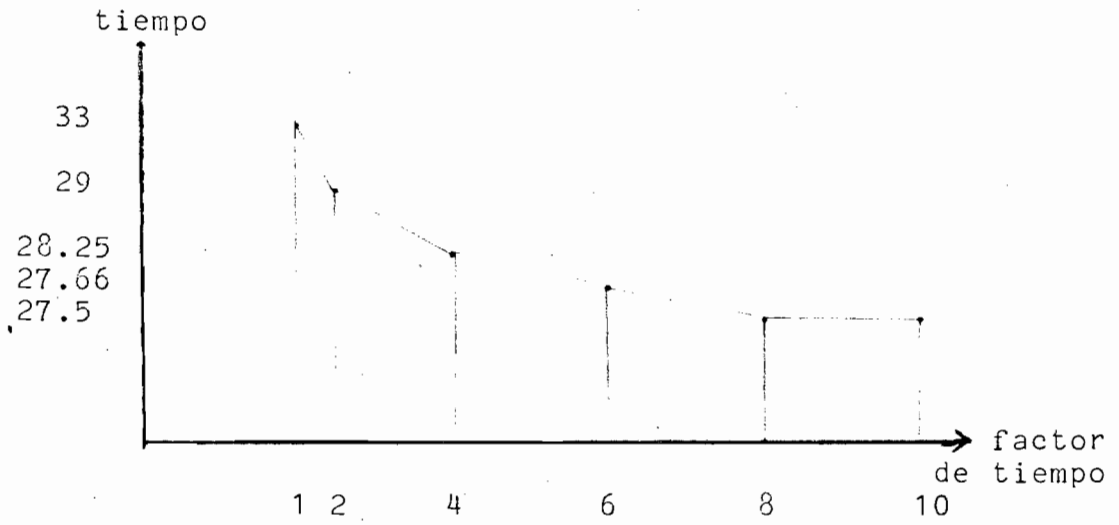


Figura 4-8: Tiempos de ejecución caundo cambia tiempo de transmisión

El tiempo de comunicación también es un factor importante para el tiempo de ejecución.

4.1.2 Reconfiguración del sistema

En esta sección, se va a describir la reconfigguración del sistema. Se probará en un sistema de 3X4. Desde el simulador, dado los eventos como integración de nodos, despedida de nodos, falla de nodos y falla de buses; el sistema se reconfigura encontrando nuevas rutas entre nodos y obteniendo nuevas estructuras lógicas.

Se supone que el sistema inicial solo contiene 4 nodos activos: 0, 2, 6 y 7 que forma una sola región. Su configuración y la estructura lógica se mostrara en la figura 4-9. Más adelante dando los eventos secuencialmente, se cambiará el sistema.

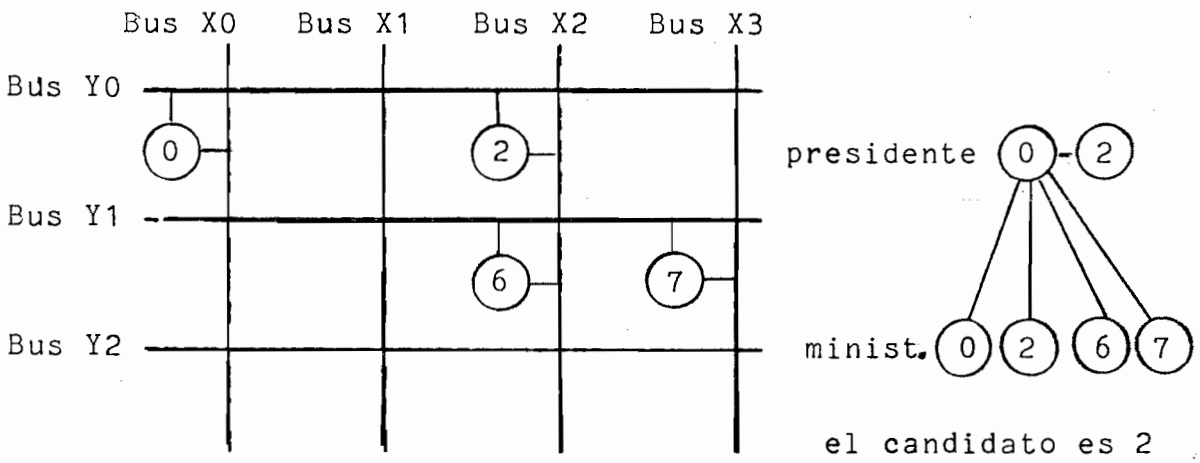


Figura 4-9: Estado inicial del sistema

4.1.2.1 Integración del nodo 9

Despues de hacer la integración el sistema quedara como se muestra a continuación (la figura 4-10).

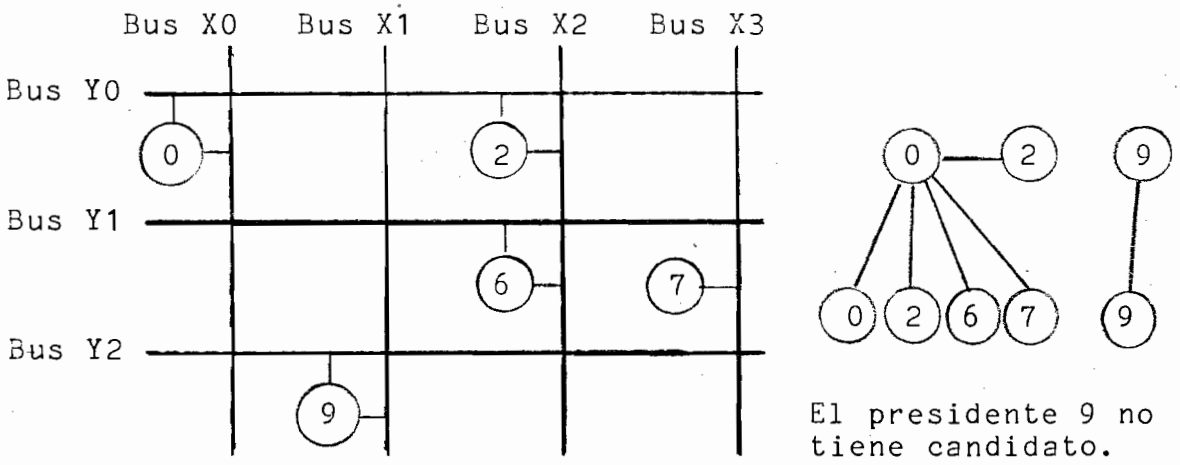


Figura 4-10: Integración de nodo 9

Como el nodo 9 no tiene bus comun con ningun nodo de la region anterior, entonces este mismo forma su propia región cuya presidente y ministro es un solo nodo que es el 9.

4.1.2.2 Integración del nodo 4

El nodo 4 se integro a la región que tiene presidente 0, porque este nodo no solo comparte el bus X0 con el nodo 0, sino también comparte el bus Y1 con el nodo 6. Pero con el nodo 9 no puede pasar mensajes directamente ni por ningun nodo como intermedio (ver la figura 4-11).

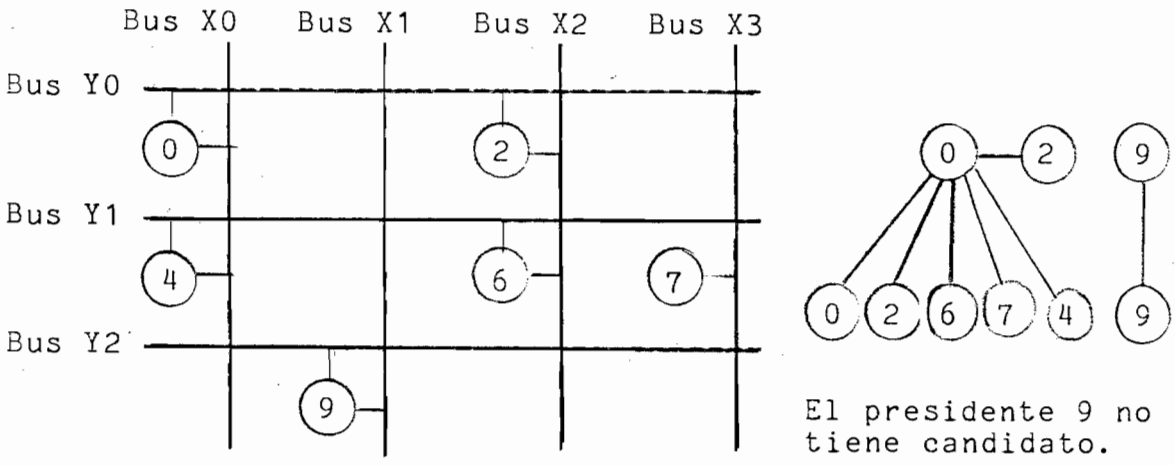


Figura 4-11: Integración del nodo 4

4.1.2.3 Integración del nodo 1

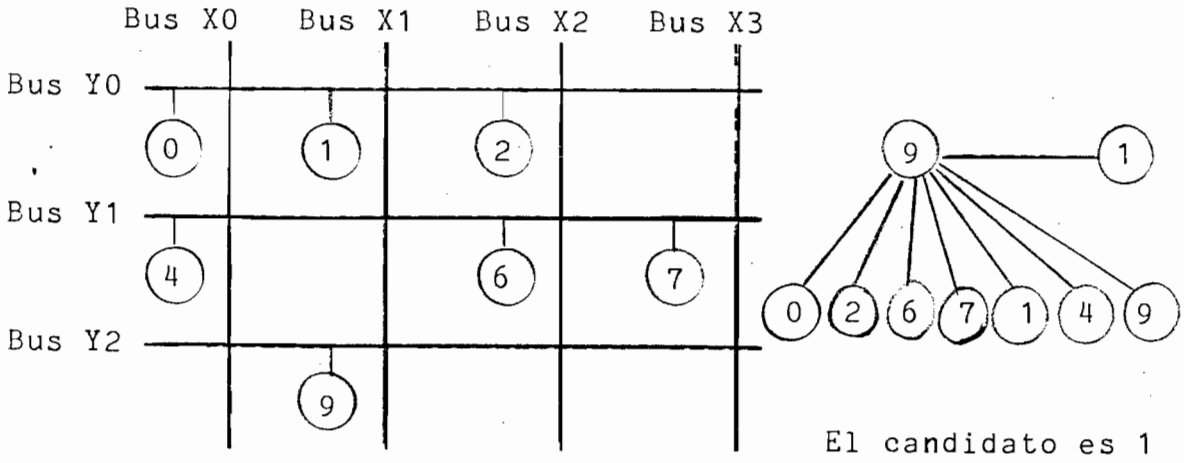


Figura 4-12: Integración del nodo 1

Ver la figura 4-12 el nodo 1 relaciona con las dos regiones. Entonces se junto estas dos a una sola región cuya presidente es 9 y el candidato es 1.

4.1.2.4 Despedida del nodo 2

Como el nodo 2 no es un nodo importante, se desaparece sin afactar los otros nodos (ver la figura 4-13).

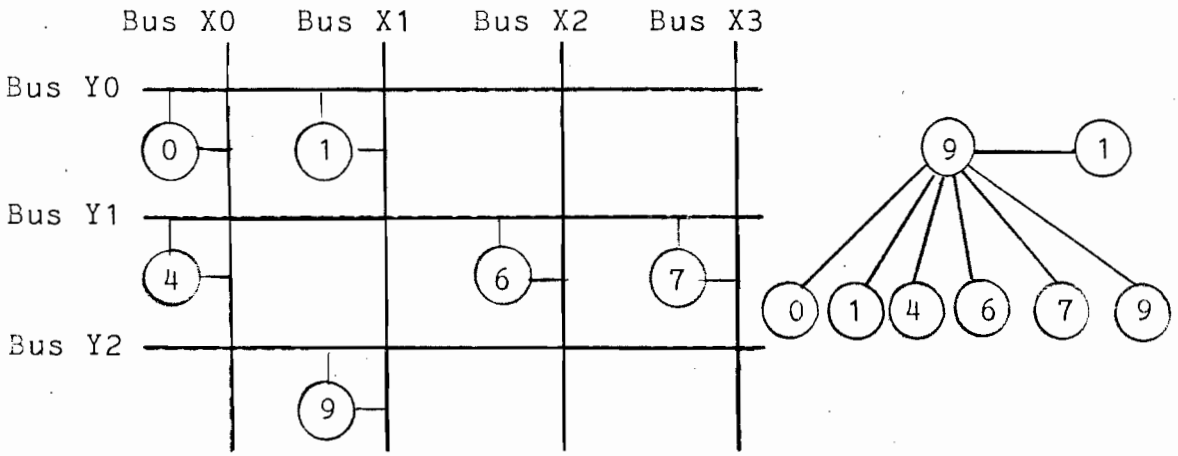


Figura 4-13: Despedida del nodo 2

4.1.2.5 Despedida del nodo 1

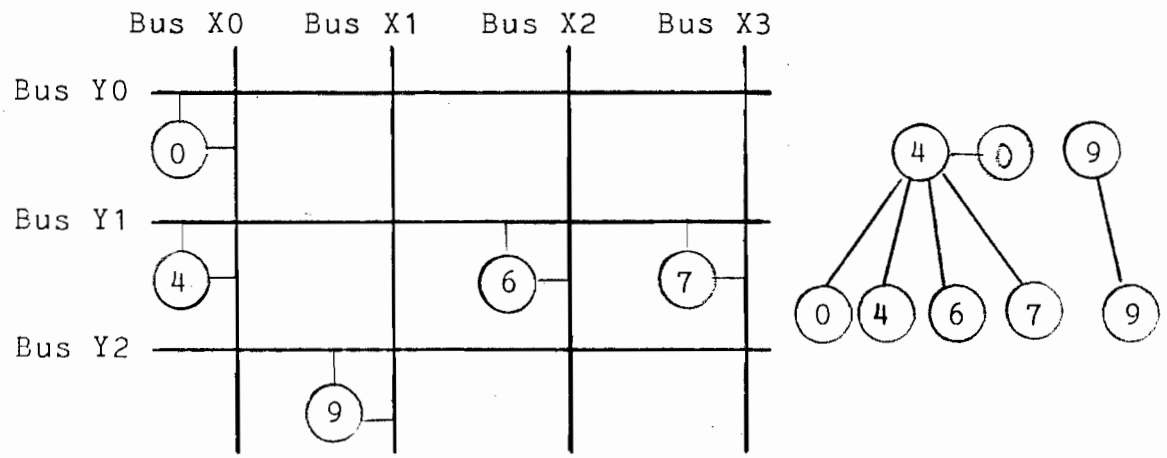


Figura 4-14: Despedida del nodo 1

El nodo 1 ocupa una posición muy importante. Por su ausencia, la región anterior separó en dos (ver la figura 4-14).

4.1.2.6 Despedida del nodo 4

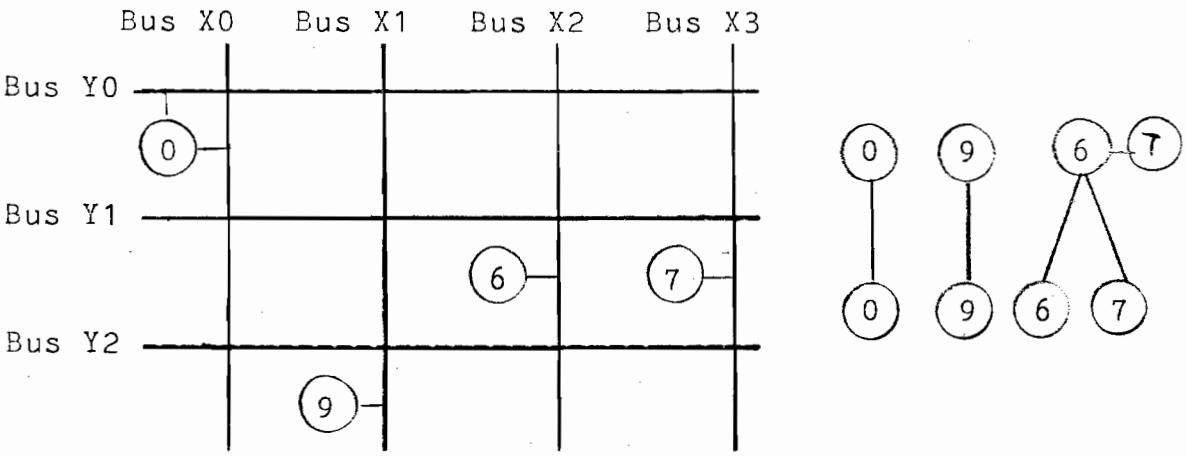


Figura 4-15: Despedida del nodo 4

Por la misma razon, el nodo 4 separó otra región. Ahora el sistema queda en tres regiones como la figura 4-15.

Para ver casos de fallas, primero se reconfigura el sistema a un estado que hay siguientes nodos activos 0, 1, 4, 6, 7, y 9 que se mostrara en la siguiente figura 4-16.

4.1.2.7 Falla de computadora de comunicación en el nodo 6

Si sucede este tipo de falla, no se afectara a ningun nodo ni los buses, porque el nodo 6 no es nodo intermedio de ningun nodo, entonces el sistema no cambia

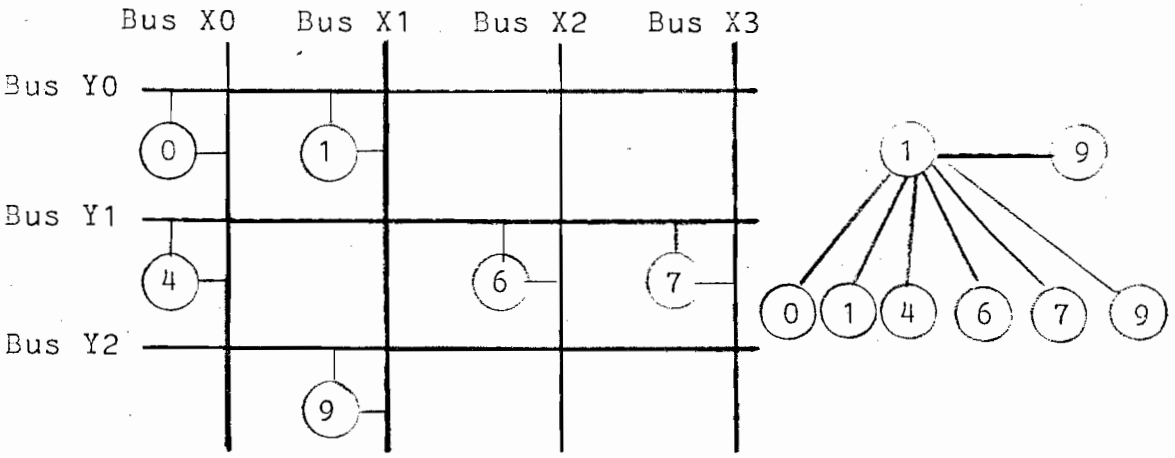


Figura 4-16: Nuevo configuración del sistema

bruscamente (ver la figura 4-17).

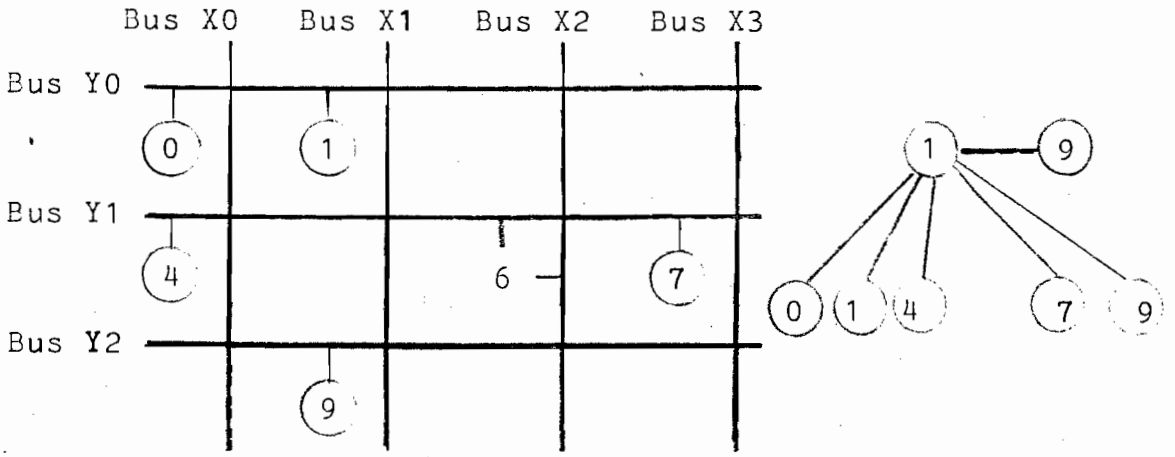


Figura 4-17: Falla de computadora de comunicación del nodo 6

4.1.2.8 Falla de bus Y1

Por esta falla interrumpió la relación entre nodos 4 y 7, entonces el nodo 7 ya no puede comunicar con ningun

nodo del sistema y se forma otra región (ver la figura 4-18).

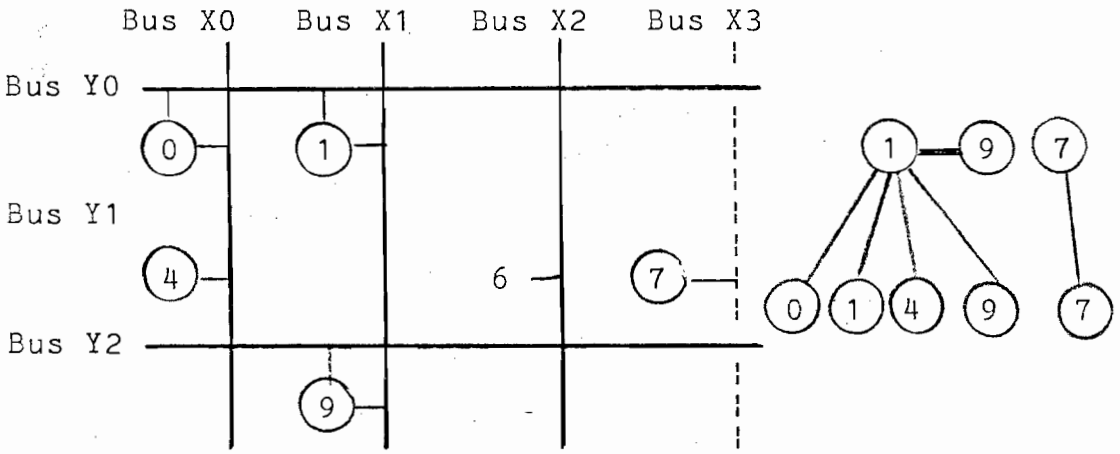


Figura 4-18: Falla de bus Y1

4.1.2.9 Falla de computadora de tarea en el nodo 9

El nodo 9 era el candidato de presidente 1. Por su ausencia, el candidato cambió al nodo 4 (ver la figura 4-19).

Ver el sistema mostrado anteriormente, aunque nodos 6 y 9 no se considera como nodos activos del sistema, pero cuando se recupera las fallas, se les integran al sistema inmediatamente.

4.1.2.10 Recuperación de la falla en bus Y1

Por la recuperación de la falla, juntó las dos regiones otra vez, porque el nodo 7 ya puede comunicar con

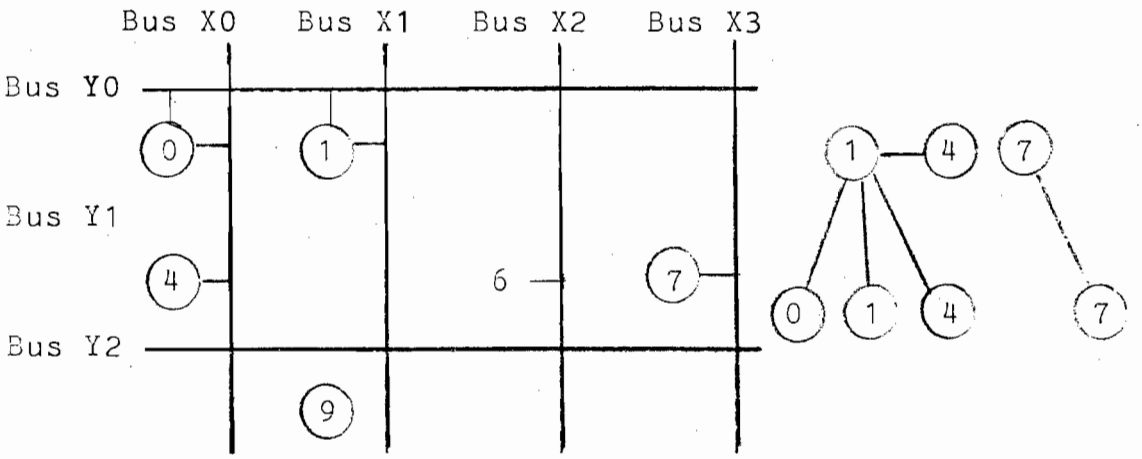


Figura 4-19: Falla de la computadora de tarea en el nodo 9

los otros (ver la figura 4-20.

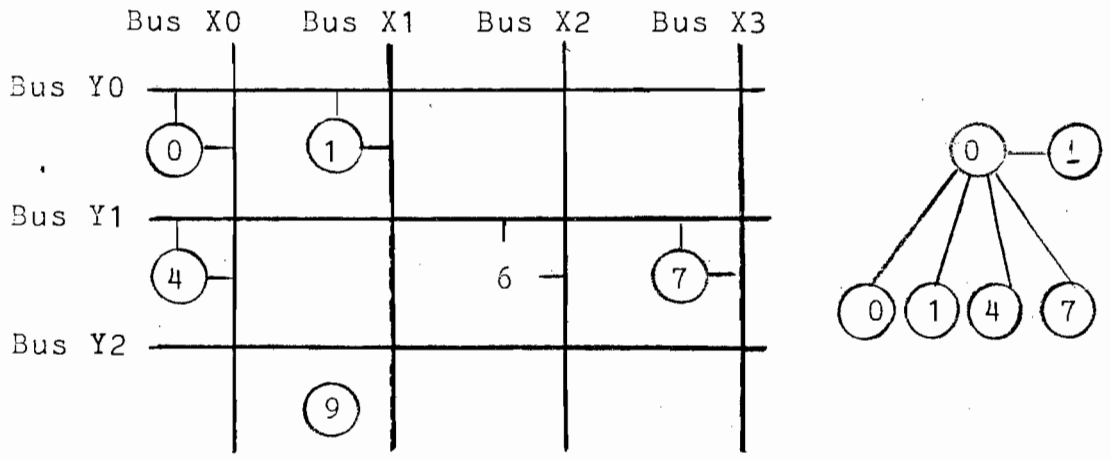


Figura 4-20: Recuperación de la falla en bus Y1

4.1.2.11 Recuperación de falla en el 6 y el 9

Ahora el sistema queda como la figura 4-21.

Aunque el sistema físico es igual que es el anterior

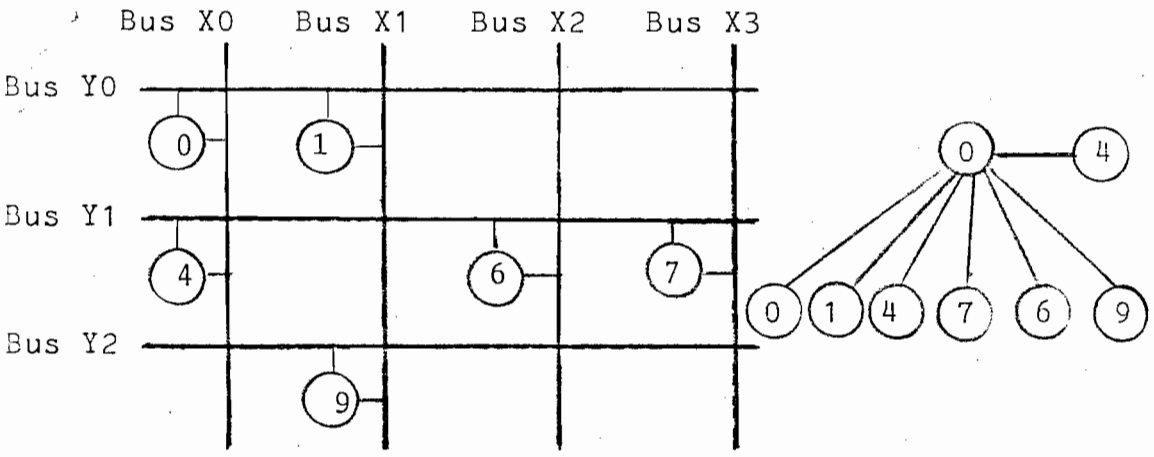


Figura 4-21: Recuperación de los nodos 6 y 9

(ver la figura 4-16), por una serie de fallas y recuperaciones la estructura lógica cambió.

4.2 Pruebas complicadas

Una gran ventaja de este sistema es tolerante a fallas. Aunque se encuentra las fallas durante la ejecución de un programa, se puede pasarlo al otro nodo para continuar la ejecución sin darse cuenta por el usuario. A continuación, vienen unas pruebas sobre la ejecucion cuando hay fallas en el sistema. También se va a mostrar las trayectorias de ejcución para estas pruebas.

Primero, se mostrará en la página **113** la trayectoria de ejecución del programa anterior. Esta ejecución se ha realizado en el nodo 0 de un sistema de 2X2. El presidente es el 0 también. Durante toda la ejecución no

tiene ninguna interferencia sobre ella, o sea no se ha reconfigurado el sistema ni ha fallado el nodo de ejecución y colaboradores o algún bus (ver la figura 4-22).

La figura 4-23 en la página 114 trae la trayectoria durante la ejecución del programa. En el tiempo 2, se falló la computadora de comunicación del nodo 1. Comparando con la figura 4-22, el nodo 1 ya no puede ser colaborador y el trabajo correspondiente se realizó en el nodo 2. El tiempo aumentó en 2, el cual es 35 ahora.

La figura 4-24 mostró el caso de que cuando falla el nodo colaborador (el nodo 1) en tiempo 13, se pasaron los códigos al nodo 2 para terminar el trabajo. El tiempo es 41.

Cuando falla el nodo que esta ejecutando un proceso por su propio iniciativo, también pasa los códigos a otro nodo disponible. Cuando termina la ejecución de este proceso, guarda sus resultados en el nodo y avisa al presidente. Una vez recuperado el nodo, manda los resultados a este nodo. Desde la figura 4-25 hasta la figura 4-29 son trayectorias de ejecuciones del programa cuando se encuentran fallas en el nodo 0 en diferentes momentos. La figura 4-25 muestra el caso de que falla la computadora de tarea en el nodo 0 antes de tener procesos paralelos. La figura 4-26 muestra el caso cuando está pasando los códigos

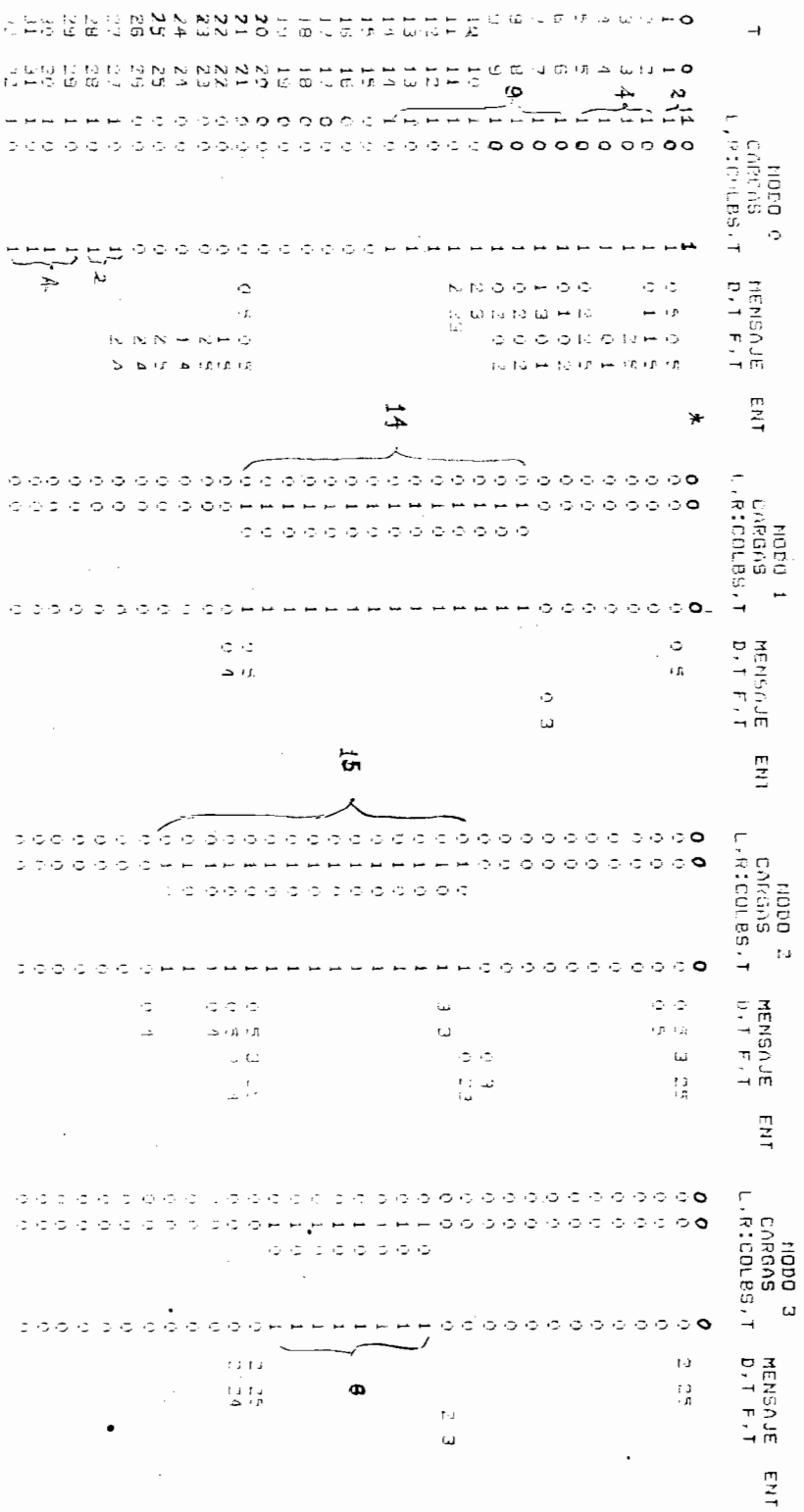


Figura 4-22: Trayectoria de la ejecución normal

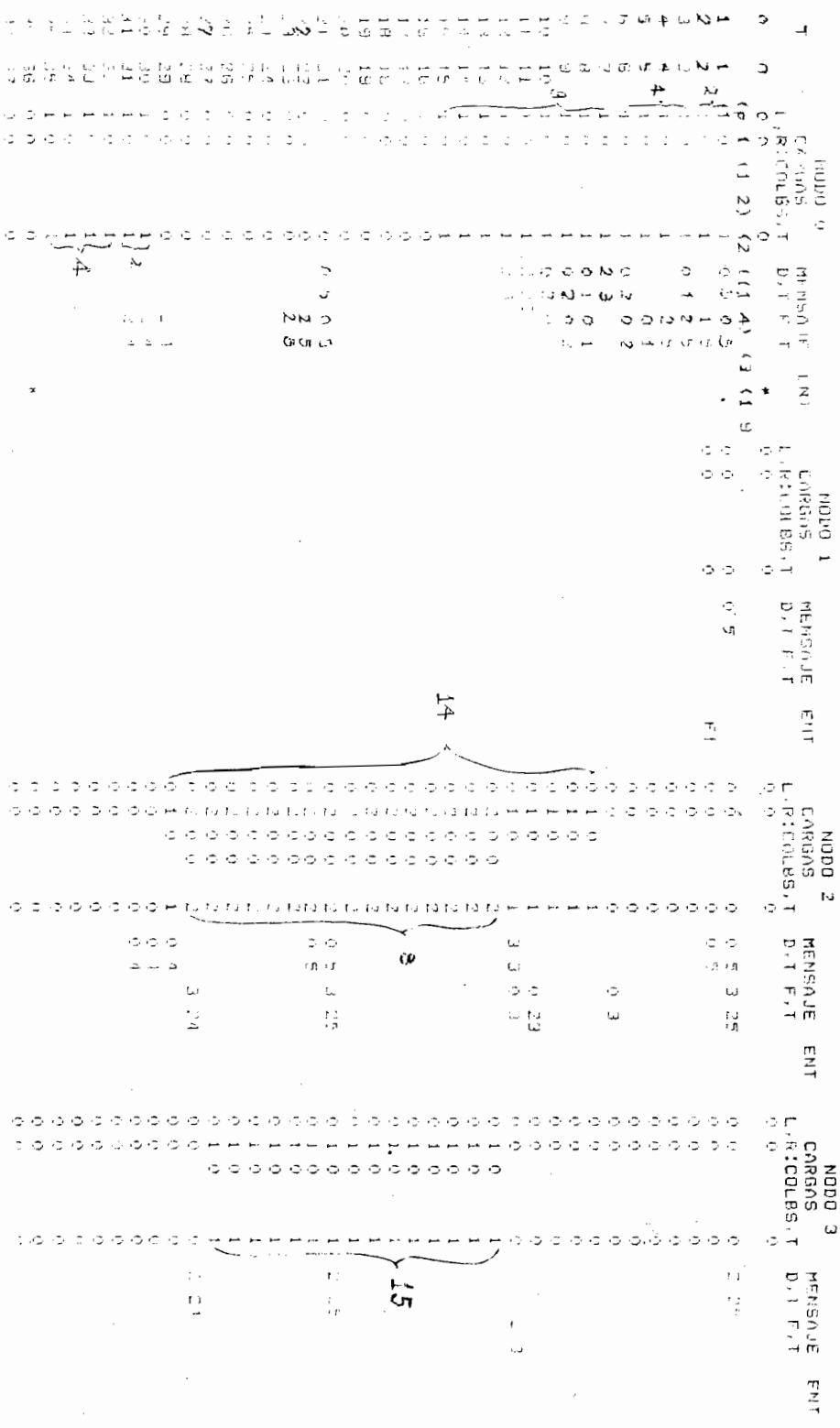


Figura 4-23: Falla en el tiempo 2 en el nodo 1

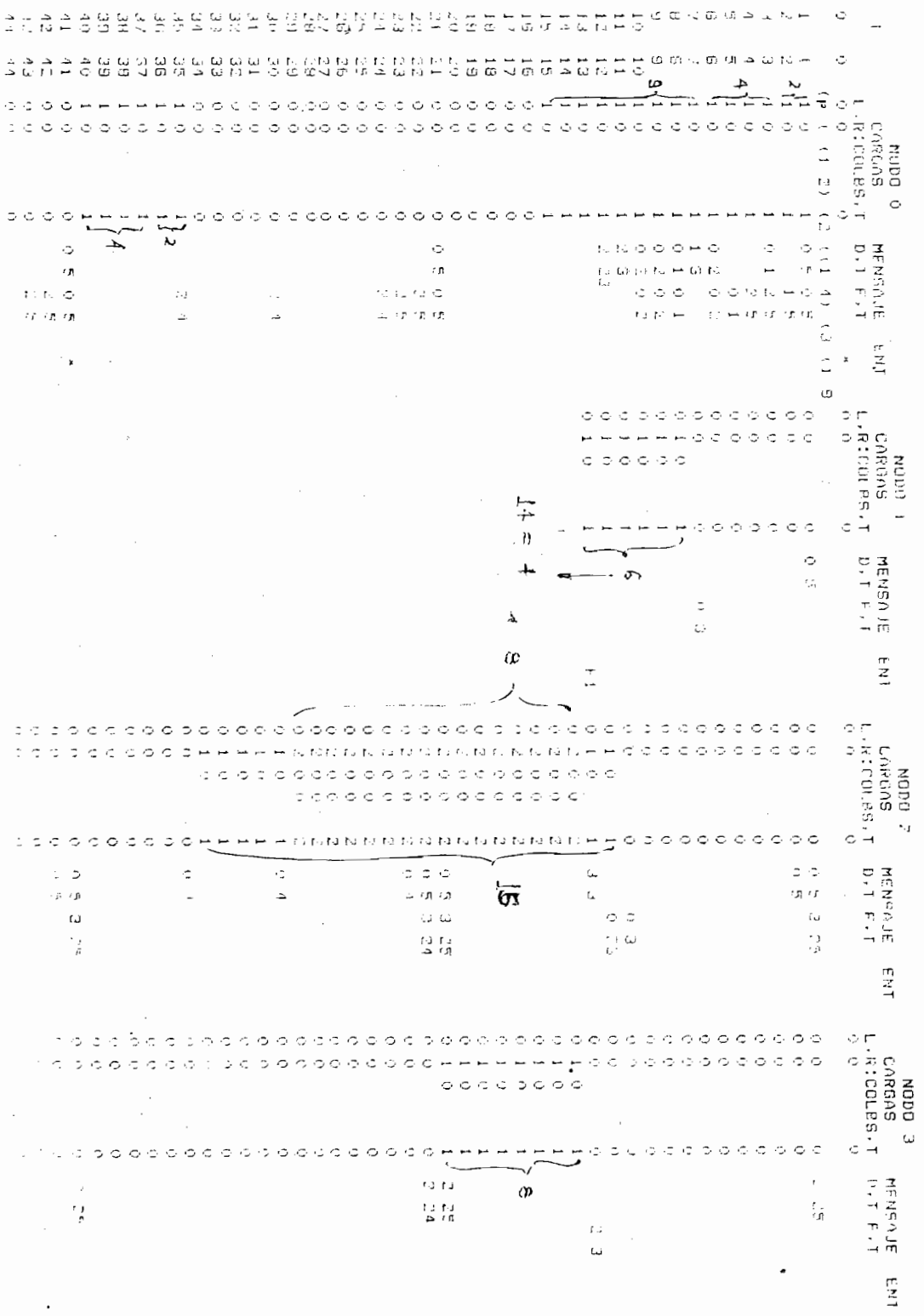


Figura 4-24: Falla en el tiempo 13 en el nodo 1

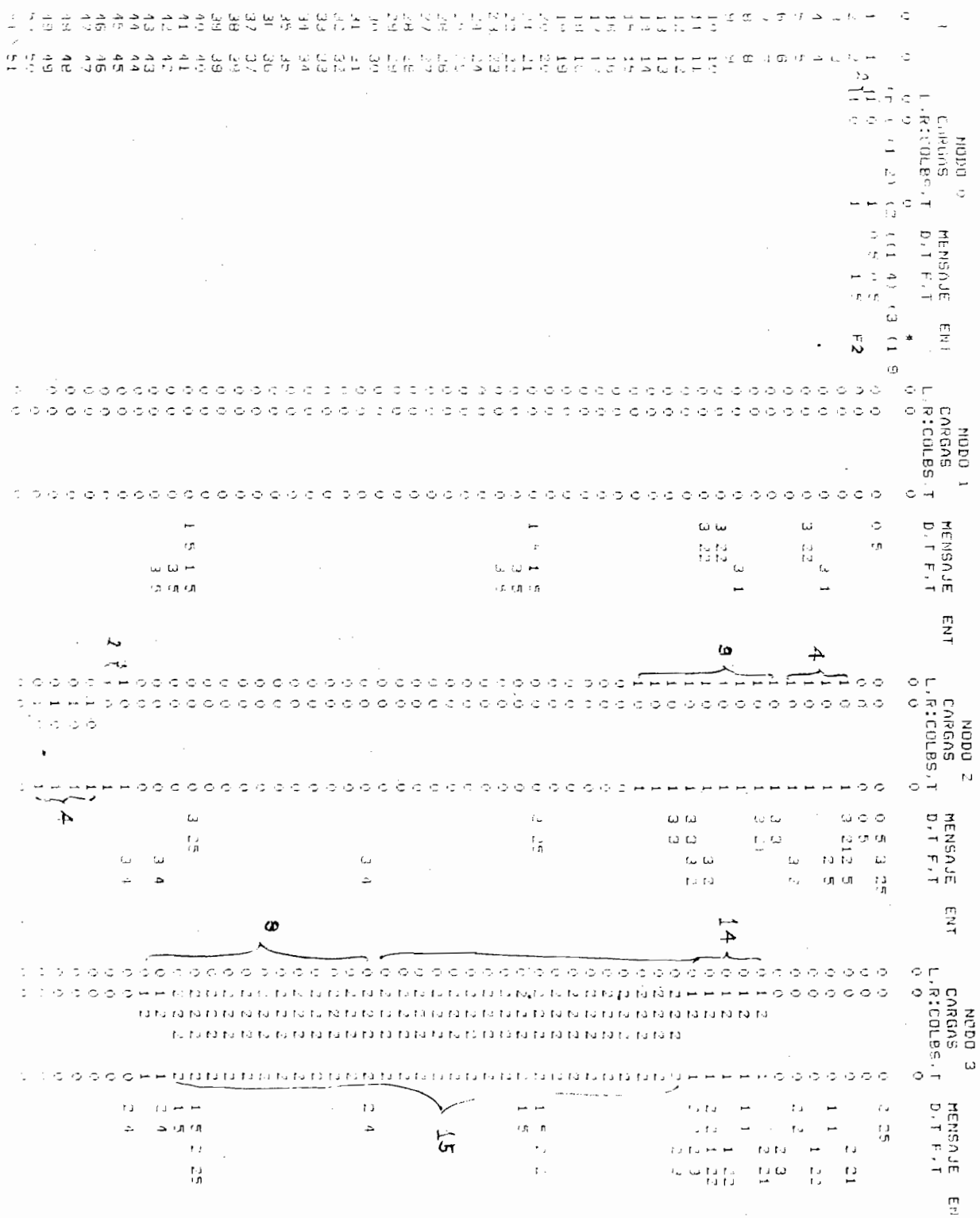


Figura 4-25: Falla en el tiempo 2 en el nodo 0

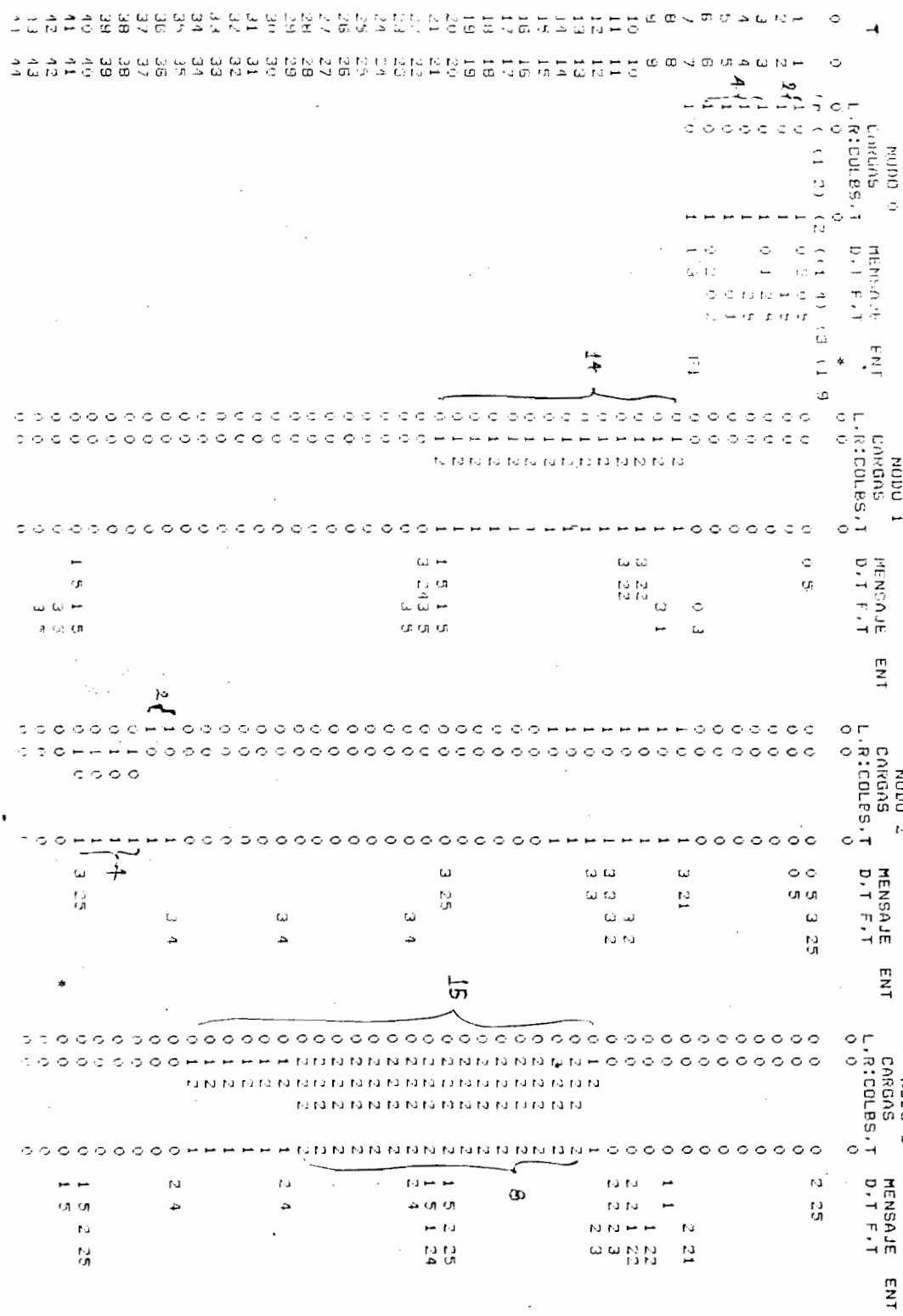


Figura 4-26: Falla en el tiempo 7 en el nodo 0

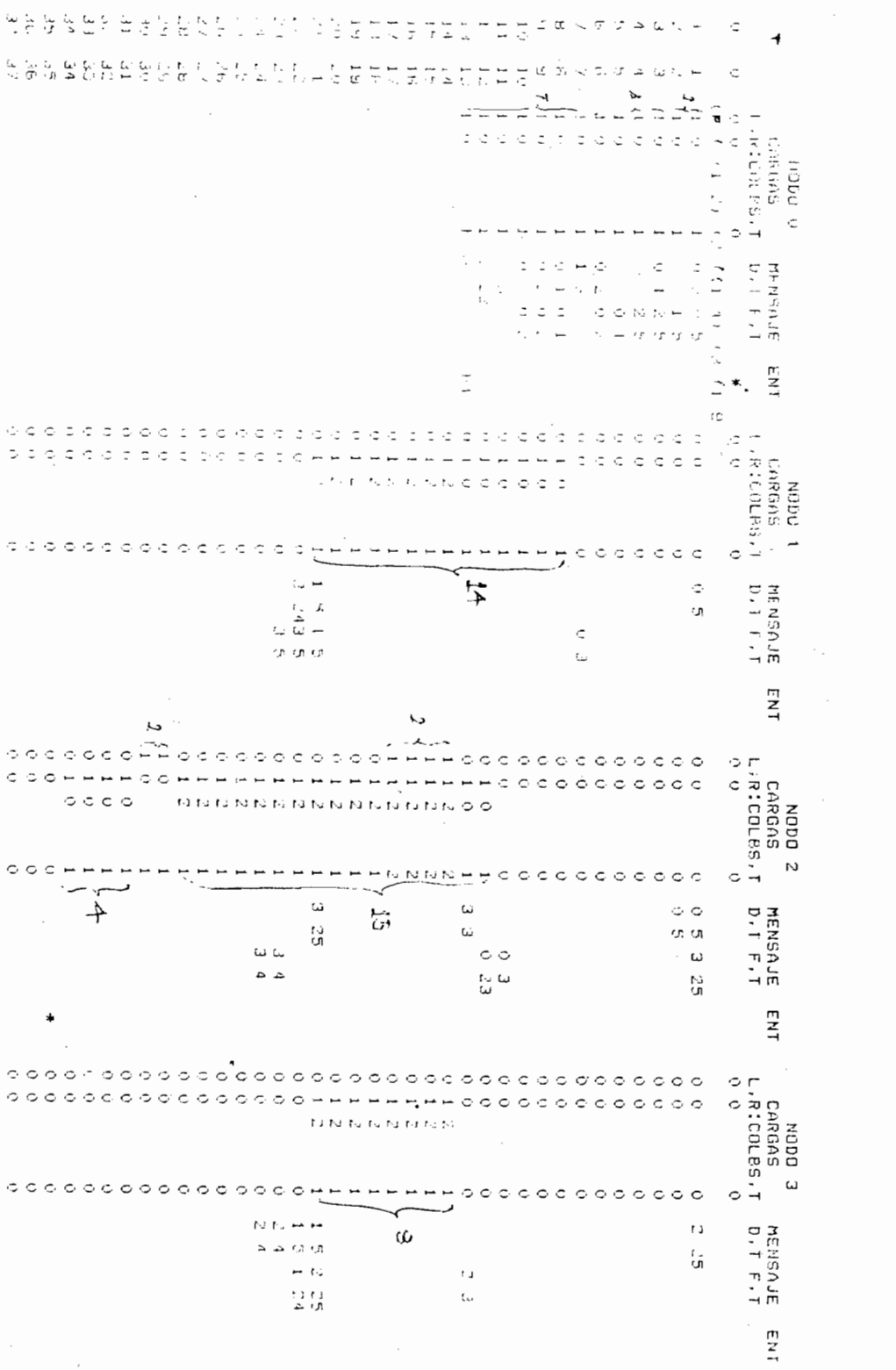


Figura 4-27: Falla en el tiempo 13 en el nodo 0

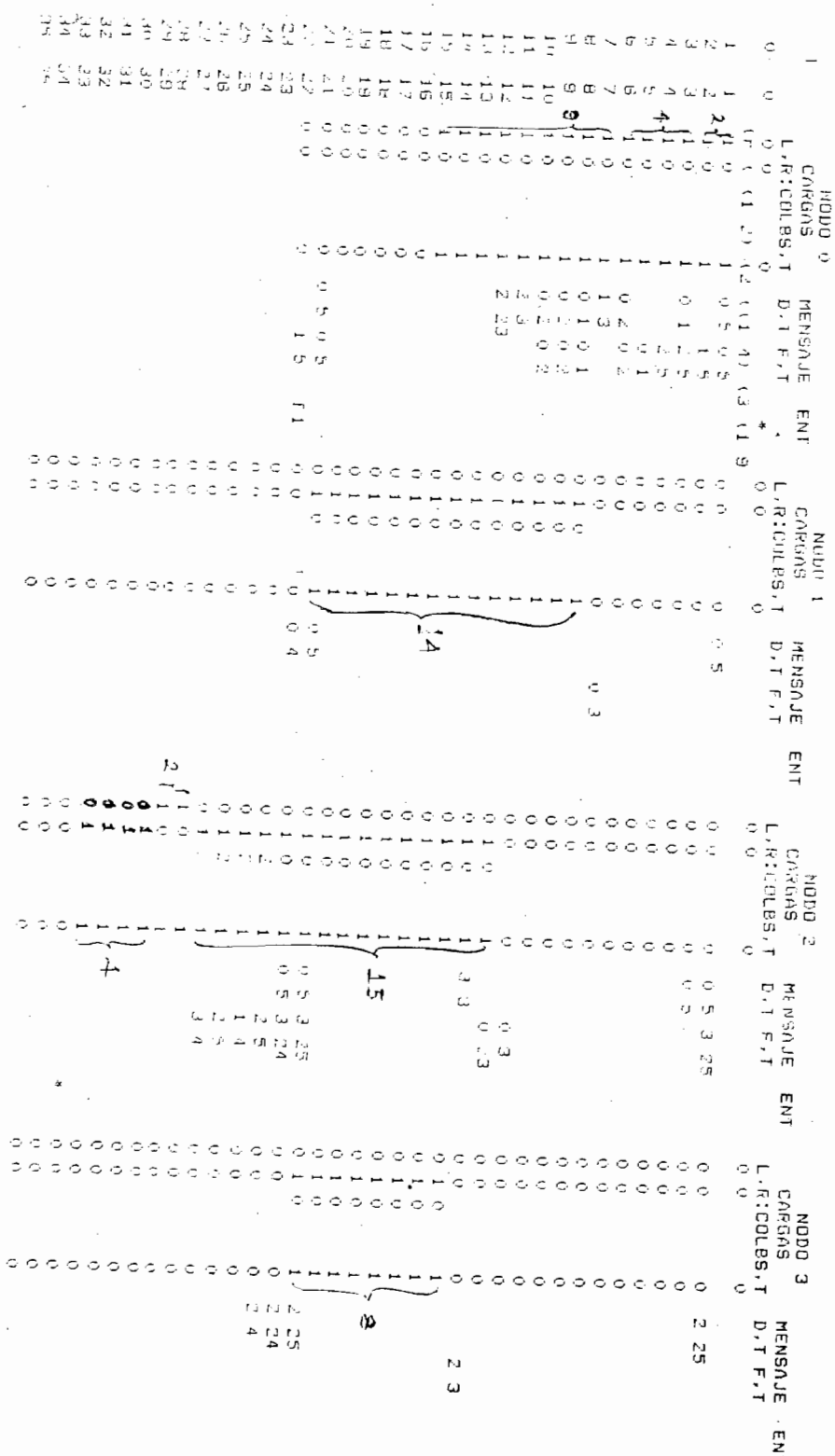


Figura 4-28: Falla en el tiempo 22 en el nodo 0

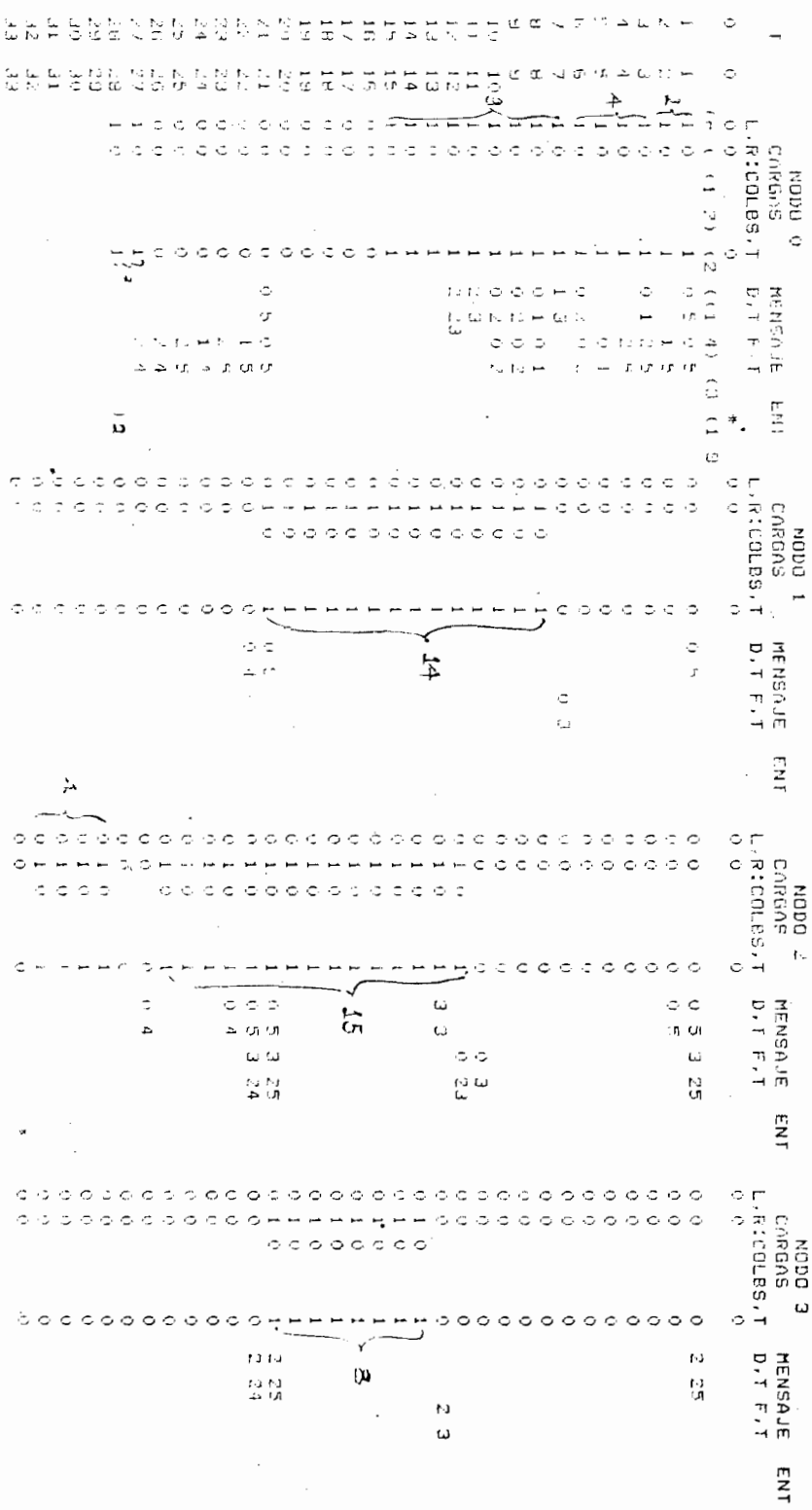


Figura 4-29: Falla en el tiempo 28 en el nodo 0

Figura 4-30: Ejecución de dos programas

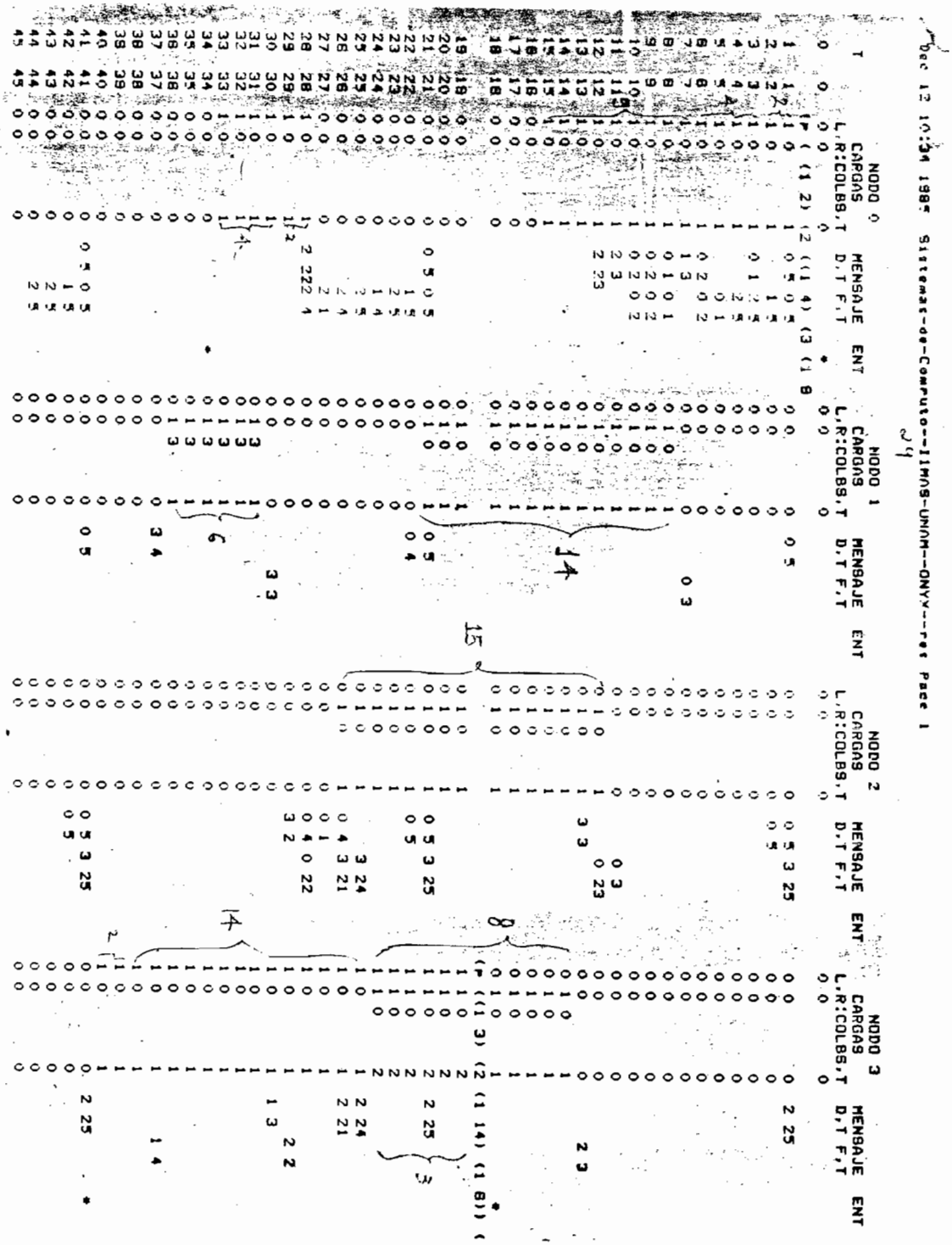


Figura 4-30: La ejecución de dos programas

del proceso hijo. La figura 4-27 tiene el caso cuando está haciendo la ejecución de un proceso hijo en el nodo 0. La figura 4-28 es cuando el nodo está esperando los resultados de otros procesos hijos que estan ejecutdo en otros nodos, falló el nodo. El ultimo caso en la figura 4-29 es cuando va a empezar la ejecución del ultimo proceso secuencial, falló la computadora de tarea en nodo 0. Si falla la computadora de comunicación en este momento, la ejecución se puede seguir haciendo en el mismo nodo, porque la computadora de tarea todavia esta funcionando y ademas, este nodo no tiene nodos colaboradores ni es colaborador de otro nodo. Para los ultimos 6 casos, el proceso en nodo 0 se ha pasado al nodo 2. Los tiempos de ejecución del mismo programa son 50, 41, 34, 32 y 32 respectivamente.

Otro ejemplo para prueba complicada puede ser siguiente: tener dos programas en dos nodos diferentes en un periodo de tiempo. La figura 4-30 muestra la trayectoria de comportamiento para este caso. En el tiempo 0, metió el programa del ejemplo anterior en el nodo 0, y en el tiempo 18 vino otro en nodo 3, el cual es

(p ((1 3) (2 (1 14) (1 6)) (1 2)))

En ese momento, las cargas del nodo 3 se aumentó. Entonces se tardó de terminar el primer programa que inició

en nodo 0. Ahora el tiempo de primer programa es 34 en ves de 33 y el tiempo de segundo es 22 en ves de 25. Si el segundo programa no ha ejecutado en paralelo sino concurrente, el tiempo de la ejecución va a ser 28.

Por los ejemplos anteriores se puede tener una idea más clara sobre el simulador y el funcionamiento del sistema operativo distribuido. Pero el presente trabajo todavia es muy limitado, y todavia hay muchos trabajos para completar el sistema. Eso se va a mencionar en el siguiente capítulo.

5. Conclusión y trabajos futuros

5.1 Conclusión

En la presente tesis se ha definido el diseño del sistema operativo distribuido (SOD) que se aplica en el sistema distribuido de tipo matricial. Este SOD tiene capacidad para distribuir las tareas, balancear las cargas, reconfigurar el sistema, tolerar cierto tipo de las fallas y controlar los recursos del sistema.

Los nodos pueden incluirse en el sistema o separarse del mismo con mucha facilidad. SOD siempre organiza los nodos activos en regiones, donde cada una de ellas tiene un presidente, varios ministros, y a veces varios colaboradores. Este esquema facilita la distribución de las tareas, el balanceo de las cargas y el control de los recursos con el fin de minimizar el tiempo de la ejecución de procesos. En este sistema se clasificaron las fallas en tres tipos, las cuales tienen diferentes grados de recuperación.

La segunda parte de la tesis presenta la simulación del sistema, la cual proporciona un más claro funcionamiento de SOD y facilita investigar el sistema con más profundidad y más detalle. De las pruebas que se han hecho en el capítulo IV, se puede obtener las siguientes conclusiones:

1. El tiempo de la ejecución de un programa paralelo disminuye cuando aumenta el número de colaboradores. Pero el número de colaboradores debe ser menor o igual que el número de procesos paralelos (ver 4.1.1.1).

2. El tiempo de la ejecución depende de las posiciones de los colaboradores (ver 4.1.1.2).

3. El tiempo de transferencia es un factor importante para la ejecución. En la figura 4-8 resulta que a medida que el tiempo de transferencia de un mensaje disminuye, el tiempo de la ejecución también disminuye. Sin embargo, llega un momento en que el tiempo de ejecución se estabiliza aunque se siga disminuyiendo el tiempo de transferencia.

4. En la subsección 4.1.2 se puede ver claramente que cuando la configuración física cambia, la lógica siempre forma una jerarquía con un presidente y varios ministros en una región, tal vez hay colaboradores.

5. Las fallas recuperables no afectan el funcionamiento del sistema y desde luego, retrazan la ejecución de un programa (ver las figuras desde la página 114 hasta 120).

El simulador se divide en dos modulos: uno es para el funcionamiento del sistema y el otro es la simulación del SOD. El funcionamiento de estos dos módulos es independiente. Si no cambia la estructura física del sistema, pero se modifica las políticas de operaciones de SOD, los cambios en la programación se refieren exclusivamente al módulo de la simulación de SOD. En caso contrario, los cambios se refieren exclusivamente al funcionamiento del sistema.

El sistema distribuido es un sistema grande y complicado. La presente tesis sólo es una parte de todos los trabajos para la construcción. Falta todavía muchos detalles que necesita completar. Pero basado en la simulación será más fácil desarrollar el sistema.

5.2 Trabajos futuros

5.2.1 Sincronización de eventos

Actualmente, la simulación no toma en cuenta el tiempo de cambio de estados del sistema. Pero este cambio en realidad necesita un intervalo de tiempo, en donde pueden suceder muchas cosas. Generalmente algunos cambios de estados tales como incremento de nodo, despedida de nodo y fallas de nodo o buses hacen que el sistema reorganize su estructura física y lógica. Si no hay control de los cambios de estado se pueden destruir la estructura lógica, ya que en un intervalo de tiempo se generan varios eventos que pueden reconfigurar el sistema. Por ejemplo, mientras un nodo se está despidiendo del sistema, otro puede integrarse al mismo tiempo. Antes de desaparecer el nodo de despedida desde la lista de ministros en su presidente, el nodo de incremento toma los datos del presidente considerando que el nodo de despedida es un ministro, y luego inserta el mismo en la lista de ministros. El nodo de incremento no sabe que otro nodo va a salir del sistema. Como la actualización de la

lista de ministros para el incremento viene despues de la de despedida, la lista de ministros tiene los datos equivocados. El nodo de despedida ya no debe estar en la lista.

Los datos del presidente o las informacioones globales se consideran como sección crítica, donde solo se accesan en forma de exclusión mutua. Cuando se está integrando un nuevo nodo al sistema y en ese momento otro nodo también desea ser integrado en SOD, a este último no se lo debe dejar de entrar hasta que termine de integrarse totalmente el primero. El mecanismo de sincronización puede utilizar candados, semáforos o monitores. Pero este trabajo se considera com trabajo futuro.

5.2.2 Interface

Más adelante se puede hacer la interface entre la parte del sistema operativo de proposito general y la parte de distribución de la carga. En el capítulo II se ha mencionado que un sistema operativo distribuido se compone de dos partes en lo referente al manejo de recursos: locales y globales. En la presente tesis se enfoca al manejo de recursos globales, posteriormente se puede conectar con la parte de manejo de recursos locales.

5.2.3 Programa real

Para mejorar la simulación se puede sustituir el programa distribuido que se ha descrito en el capítulo III por medio de un programa real utilizando un lenguaje concurrente como OCCAM o un lenguaje distribuido como *MOD. Entonces se puede probar el acceso a variables comunes, la transmisión de las subrutinas con llamadas en otro nodo.

5.2.4 Especifiación de hardware

Para que la simulación sea más cercana al sistema real, es necesario especificar con más detalle el hardware del sistema distribuido como por ejemplo, en la parte de comunicación. El tiempo de transferencia, el tiempo de retraso de mensajes, el tamaño de paquete de mensaje, etc. son factores importantes para la simulación.

Algoritmo de balancear las cargas, programa para la comunicación y otros trabajos tanto de software como de hardware, se necesitan considerar para construir un sistema disitrbuido tan grande y tan complicado como el presente.

BIBLIOGRAFIA

[Algirdas 71] Algirdas Avizienis, George C. Gilley.
The STAR (Self-Testing And Repairing) computer: An investigation of the theory and practice of fault-tolerance computer design .
IEEE Transaction on Computer C-20(11), 1971.

[Bernstein 81] Philip A. Bernstein and Nathan Goodman.
Concurrency control in ditributed database systems.
Computer Surveys 13(2), June, 1981.

[Booth 81] Booth, Grayce M.
The distributed system environment.
McGraw-Hill, 1981.

[Borg 83] Anita Borg.
A message system supporting fault-tolerance.
ACM , 1983.

[Drett 81] Brett D. Fleisch.
An architecture for Pup services on a distributed operating system.
Operating system review , 1981.

[Enslow 78] Enslow, P.H
What is a distributed data processing system?.
Computer 11, Jan, 1978.

[Champine 80] George A. Champoine.
Distributed computer systems.
North-holland, 1980.

[Cheriton 85] David R. Cheriton.
Distributed process groups in the V kernel.
ACM Transaction on Computer 3(2), May, 1985.

[Colin 83] Colin J. Theaker & Grahanm R. Brookes.
A practical course on operating systems.
The Macmillan Press LTP, 1983.

[Cook 80] Robert P. Cook.
*MOD: A language for distributed programming.
IEEE Transaction on software engineering Se-6, Nov., 1980.

[Danning 78] Peter J. Denning.
Operating system principle for data flow networks.
IEEE Computer , July, 1978.

[Date 83] C. J. Date.
An introduction to Database System.
Addison-Wesley Publishing Company, Inc., 1983.

[Dowson 78] M. Dowson.
The DENOS multiple processor technical summary.
National Phiysical Lab., Great Britain, NPL Rep. Com 102 , Apr, 1978.

[Fisher 83] P. S. Fisher, Jacob Slonim and E. A. Unger.
Advances in distributed processing management.
John Wiley & Sons, 1983.

[Holt 83] R. C. Holt.
Concurrent Euclid, The UNIX system and TUNIS.
Addison-Wesley, 1983.

[Kartasev 81] Svetlana P. Kartasev .
Reconfiguration of dynamic architecture into multicomputer network.
Proceedings of the 1981 international conference on parallelo processing , 1981.

[Levin 84] Guillermo Levin Gutierrez.
Introducción a la computación y a la programación estructurada.
McGraw-Hill, 1984.

[Lo. 81] Virginia Lo. & Jane W. S. Liu.
Task assigment in distributed multiprocessor system.
Proceedings of 1981 internacional conference on parallelo processing , 1981.

[Ni. 81] Lionel M. Ni. .
Optimal load balancing: strategies for a multiple processor system.
Proceedings of the 1981 international conference on parallelo processing , 1981.

[Ousterhont 80]

John K. Ousterhont, Donald A. Scelza, Pradeep S. Sindhu.
Medusa: an experience in distributed operating system.
Communication of the ACM 23(2), Feb., 1980.

[Ralston 83] Anthony Ralston, Endwin D. Reilly, JR.
Encyclopedia of Computer Science and Engineering.
Van Nostrand Reinhold Company, 1983.

[Rashid 85] Richard F. Rashid.
Accent: A communication oriented network operating system kernel.
Proceedings of the eighth symposium on operating systems principles 15(5), Dec., 1985.

[Rennels 80] David A. Rennels.
Distributed fault-tolerance computer system.
Computer , March, 1980.

[Sincoskie 80] W. David Sincoskie and David J. Farber.
SODS/OS: A distributed operating system for ibm serie/1.
Operating system review , 1980.

[Tanenbaum 81] Andrew S. Tanenbaum.
Computer Network.
Prentice-Hall, 1981.

[Taylor 82] Richard Taylor & Pete Wilson .
Process-Oriented Language meets demands of distributed processing .
Electronic , Nov., 1982.

[Taylor 82] Richard Taylor and Pete Wolson.
Process-oriented languaje meets demands of distributed processing.
Electronics , Nov., 1982.

[Tsujino 84] Y. Tsujino.
Concurrent C: A programming language for distributed multiprocessor system.
Software-Practice & Experience 14(11), Nov., 1984.

[Valdorf 84] C. Valdorl.
Dedicated, distributed and protable operating

system: a structuring concept.
Software-practice & experience 14(11), Nov., 1984.

[Whitlie 80] Larry D. Whitlie.
Micros: a distributed operating system for micronet, a reconfigurable network computer.
IEEE Transaction on computer c-29(12), Dec., 1980.

[Wulf 74] W. Wulf.
Hydra: The kernel of a multiprocessor operating system.
Communication of the ACM 17(6), June, 1974.