ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

DISEÑO E IMPLEMENTACIÓN DE UN GESTOR DE PROCESOS DISTRIBUIDO EN PLAN 9

(Design and implementation of a distributed process manager in Plan 9)

Para acceder al Título de

INGENIERO TÉCNICO DE TELECOMUNICACIÓN

Autor: Eric Alvarado Salgueiro
Octubre - 2013

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Eric Alvarado Salgueiro
Director del PFC: Esteban Stafford Fernández
Título: "Diseño e implementación de un gestor de procesos distribuido en Plan 9 "
Title: "Design and implementation of a distributed process manager in Plan 9"

Presentado a examen el día:

para acceder al Título de

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SISTEMAS ELECTRÓNICOS

Composición del Tribunal:
Presidente (Apellidos, Nombre): Menéndez de Llano, Rafael
Secretario (Apellidos, Nombre): Stafford Fernández, Esteban
Vocal (Apellidos, Nombre): Prellezo Gutiérrez, José Miguel

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

Índice general

0.1. Estructura del documento 6

1. Introducción **7**

1.1. Sistemas distribuidos y sistemas operativos centralizados 7

1.2. Sistemas operativos distribuidos 9

1.2.1. Características de diseño de los sistemas operativos distribuidos 10

1.2.2. Comunicación entre procesos 14

1.2.3. Sincronización en sistemas distribuidos 14

1.2.4. Modelos de organización de los equipos 15

1.2.5. Sistemas distribuidos de archivos 15

1.2.6. Modelos de Sistemas Distribuidos 16

1.3. Sistema Operativo Plan 9 17

1.3.1. Historia 17

1.3.2. Herencia de Unix 18

1.3.3. Ideas propias del Sistema Operativo Plan 9 18

1.3.4. Características del Sistema Operativo 19

1.3.5. Planes futuros 21

1.4. Sistema de colas y planificación de procesos 22

1.4.1. Definición de proceso 22

1.4.2. Estados de un proceso 22

1.4.3. Planificación de procesos 23

1.4.4. Objetivos de la Planificación de procesos 23

1.4.5. Niveles de planificación 24

1.4.6. Algoritmos de planificación 25

1.5. Equilibrio de carga 27

1.5.1. Algoritmos de equilibrio de carga 27

2. Desarrollo **29**

2.1. Instalación del Plan 9 29

2.1.1. Instalación del Servidor 29

2.1.2. Añadir nodos al servidor 32

2.2. Herramientas del S.O Plan 9 utilizadas 38

2.2.1. Comando 'cpu' 38

2.2.2. Comando 'import' 38

2.2.3. Fichero '/dev/sysstat' 39

2.2.4. Llamada al sistema 'rfork' 39

2.2.5. Llamadas al sistema 'mount', 'bind' y 'unmount' 40

2.2.6. 'Señales' en Plan 9: 'notes' 41

2.3. Diseño e implementación de un sistema de colas 42

2.3.1. Funcionamiento 43

2.3.2. Implementación del programa encargado de repartir tareas . . 44

2.3.3. Preparación 45

2.3.4. Envío y ejecución de tareas 45

2.3.5. Algoritmos de planificación utilizados 46

3. Evaluación **49**

3.1. Descripción del entorno de pruebas 49

3.2. Descripción de las cargas 49

3.3. Descripción de las métricas de rendimiento 51

3.4. Resultados 51

4. Conclusión y líneas futuras **59**

4.1. Conclusiones 59

4.2. Líneas futuras 60

Prólogo

El objetivo de este proyecto es comprender el funcionamiento de un sistema operativo distribuido, en concreto del Plan 9. Para ello se llevará a cabo un plan de trabajo que consiste en lo siguiente:

Antes de comenzar con el Plan 9, será necesario entender el funcionamiento de los sistemas distribuidos. Para ello se realizará una labor de investigación y documentación sobre el sistema y sus aspectos más característicos. A continuación, se realizará la instalación del servidor y la conexión de otros computadores con dicho servidor.

Una vez se tenga el sistema instalado será el momento de aplicar el trabajo previo de documentación y empezar a comprender el funcionamiento del S.O, el sistema de ventanas, la shell, los editores de texto y las diferencias en las llamadas al sistema que serán necesarias al programar en este entorno.

La labor de investigación llevada a cabo no sólo valdrá para comenzar a utilizar el sistema sino que, será de suma utilidad para decidir que realizar en la elaboración del proyecto con las herramientas proporcionadas por el Plan 9, aprovechando su condición de sistema distribuido.

La elección final será la de realizar un programa encargado de repartir tareas entre los distintos computadores del sistema distribuido, ya que se considera que es una de las principales funciones de un sistema operativo distribuido. Para finalizar el proyecto se realizarán una serie de pruebas para comprobar el funcionamiento del algoritmo diseñado y se sacarán una serie de conclusiones.

0.1. Estructura del documento

El documento se va a basar en una serie de capítulos, dejando este prólogo como resumen de lo que se va a exponer a continuación. El resto de capítulos serán los siguientes:

- Capítulo 1, Introducción: se explicarán los fundamentos teóricos necesarios para comprender el funcionamiento de un sistema operativo distribuido, las característica del sistema Plan 9, la forma de actuar de un sistema de colas y la importancia del equilibrio de carga en los sistemas distribuidos.
- Capítulo 2, Desarrollo: se explicarán la instalación del Plan 9 en un servidor y como acceder a este desde otros ordenadores, las herramientas proporcionadas por el S.O utilizadas para la elaboración de este proyecto y el diseño y la realización del sistema de colas empleado para distribuir tareas.
- Capítulo 3, Evaluación: se mostrarán las pruebas realizadas, el entorno y las cargas de trabajo utilizadas y las conclusiones sacadas con los resultados obtenidos.
- Capítulo 4, Conclusiones y líneas futuras: se valorarán los resultados obtenidos en el capítulo anterior y se hará una valoración general de lo aprendido durante la elaboración del proyecto.
- Capítulo 5, Bibliografía: se indicará la bibliografía utilizada para la realización del trabajo.

Capítulo 1

Introducción

En este primer capítulo se van a explicar los conceptos necesarios para la realización del proyecto. Como se ha explicado en el prólogo este proyecto está basado en la utilización de un sistema operativo distribuido por lo que dicho concepto será definido a lo largo del capítulo. Se hará especial hincapié en el S.O Plan 9 al ser el elegido para la realización del proyecto.

Además, estos sistemas operativos distribuidos son una manera de explotar los sistemas distribuidos, los cuales son grupos de ordenadores que están conectados mediante una red y son capaces de trabajar juntos. Su objetivo es el de repartirse las tareas a realizar de forma equilibrada para maximizar el rendimiento [1].

Por último, para el diseño del programa encargado de repartir tareas en el sistema distribuido mencionado en el prólogo son necesarios una serie de conceptos adicionales. Se explicará la forma en la que un computador planifica las tareas que tiene ejecutar y en qué consiste un sistema de colas. Asimismo se definirá la importancia que tiene el equilibrio de carga en el sistema distribuido.

1.1. Sistemas distribuidos y sistemas operativos centralizados

Para entender el concepto de sistema operativo distribuido, primero es necesario comprender la función de un sistema operativo centralizado, comúnmente llamado sistema operativo (S.O). El S.O está compuesto por un núcleo y un conjunto de her-

ramientas que se encarga de controlar la ejecución de los programas de aplicación del usuario y de actuar como una interfaz entre el usuario y el hardware del computador. Además, sus dos objetivos primordiales son:

- Gestionar el hardware del computador, es decir, administrar de una forma más eficiente los recursos de la máquina.
- Facilitar el trabajo al usuario, lo que permite una comunicación con los dispositivos de la máquina [1].

Gracias al avance de la tecnología tanto en microprocesadores como en comunicación entre computadores y al abaratamiento de costes, se han hecho muy comunes los sistemas distribuidos o clusters.

Un sistema distribuido es un grupo de múltiples ordenadores (cada uno de ellos normalmente llamado *nodo*) unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador. Estos sistemas son empleados para mejorar el rendimiento y/o la disponibilidad que típicamente puede proporcionar un solo computador. Por lo que es más económico conseguir la misma rapidez con un sistema distribuido que con computadores individuales.

De este tipo de sistemas se espera que presente combinaciones de los siguientes servicios:

- Alto rendimiento
- Alta disponibilidad
- Balanceo de carga
- Escalabilidad

Además la construcción de los ordenadores del sistema distribuido es más fácil y económica debido a su *flexibilidad*, es decir, todos pueden tener la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (cluster semihomogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo).

Por otro lado para que un sistema distribuido funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del sistema distribuido. Dicho sistema se debe encargar de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.

Para maximizar los recursos de todos los computadores que forman dicha red, se diseñaron sistemas operativos específicos para dicho objetivo y recibieron el nombre de sistemas operativos distribuidos.

Por otra parte es necesario distinguir entre dos tipos de máquinas paralelas: las de memoria común, donde los procesadores comparten la memoria principal, como son los multicore, que no necesitan sistema operativo distribuido y las de memoria compartida, es decir, ordenadores independientes conectados por la red. Estos computadores independientes a su vez pueden ser máquinas de memoria común. En este último tipo si es útil el uso de sistemas operativos distribuidos.

1.2. Sistemas operativos distribuidos

Lo que caracteriza a este tipo de sistemas operativos es que permite que los usuarios accedan a todos los recursos del sistema distribuido independientemente del nodo o computador en el que se encuentren. Además el sistema operativo tiene un único conjunto de llamadas al sistema, un esquema de protección global y un sistema de ficheros y un mecanismo de comunicación entre procesos (IPC) también de forma global. Por último, cada computador controla sus propios recursos locales, pero estos son compartidos con el resto de máquinas [1].

En los sistemas distribuidos se suelen utilizar este tipo de S.O distribuidos ya que tienen una serie de ventajas respecto a los S.O centralizados. Algunas de esas ventajas son las siguientes: compartición de recursos y equilibrado de cargas, comunicación y compartición de información entre usuarios o procesos, mayor facilidad para la ampliación modular de nuevos recursos, mayor fiabilidad y mejor rendimiento, mayor seguridad porque en principio utilizar un S.O centralizado en una máquina distribuida es más inseguro que un S.O distribuido ya que este está concebido para ejecutarse en estas máquinas, e incluso una mejora en términos económicos.

Por otro lado también existen ciertas desventajas en la utilización de estos sistemas operativos como: la escasez de software propio, aunque esto es debido más a la inmadurez de este tipo de sistemas operativos que al hecho de ser distribuidos; mayor probabilidad de fallo debido a su complejidad, pero pueden realizar equilibrio de carga o delegación de tareas, de manera que cuando un equipo falla, otro puede asumir su cometido; y una mayor dificultad para proporcionar seguridad al sistema.

1.2.1. Características de diseño de los sistemas operativos distribuidos

Para entender mejor el comportamiento de un sistema operativo distribuido se van a explicar a continuación las características en las que se basa el diseño de dichos sistemas operativos.

Transparencia

Se dice que un sistema es transparente cuando funciona de forma similar en todos los puntos de la red, con independencia de la posición del usuario. Además no tiene que ser necesario establecer conexiones explícitas a máquinas remotas para utilizar sus recursos. Por lo que corresponde al S.O. establecer los mecanismos que ocultan la naturaleza distribuida del sistema y que permitan trabajar a los usuarios como si fuese un único equipo.

Un ejemplo claro de esto es cuando se hace uso de cualquier archivo. Por la naturaleza del sistema es necesario que haya varias copias del mismo en diferentes nodos. La transparencia significa que el S.O se encargue de que el usuario vea esas copias como un único archivo.

Se puede distinguir entre dos tipos o niveles de transparencia dentro de un S.O distribuido. A nivel de usuario, ocultando la distribución a los usuarios y a nivel de proceso, donde la interfaz de llamadas al sistema oculta la existencia de varios equipos. Por lo que el S.O tiene que controlar que tanto usuarios como programadores vean el sistema distribuido como un único equipo.

Además de los dos niveles de transparencia vistos anteriormente, existen también diferentes tipos de transparencia:

- De Acceso: Acceso a objetos locales o remotos de la misma manera.
- De Lugar: Acceso a objetos sin conocer dónde están.
- De Concurrencia: Varios procesos pueden operar concurrentemente usando objetos de información compartidos sin estorbarse.
- De Replicación: Diferentes réplicas de un mismo objeto de información sin enterarse a cuál se accede.
- De Fallo: Aislamiento de fallos, de forma que las aplicaciones puedan completar sus tareas.
- De Migración: Permite mover los objetos de información sin afectar a las aplicaciones.
- De Rendimiento: Redistribución de cargas en el sistema sin modificación en las aplicaciones.
- De Escalabilidad: Permite asumir cambios de tamaño del sistema y aplicaciones sin modificar la estructura del sistema ni los algoritmos de los programas.

Paralelismo

Algunos S.O distribuidos también se encargan de distribuir las tareas entre los distintos procesadores como en un sistema multiprocesador, pero con la dificultad añadida de que esta tarea tiene que realizarse a través de la red entre varios computadores.

Eficiencia

La idea principal de los sistemas distribuidos es la de obtener sistemas mucho más rápidos que los ordenadores actuales. Para ello es necesario conseguir una buena eficiencia. Si se quiere conseguir un sistema eficiente hay que descartar la idea de ejecutar un programa en un único procesador. Por ello hay que distribuir las tareas

a los procesadores libres más rápidos en cada momento, lo que lleva a la idea vista anteriormente del *paralelismo*.

La idea de que un procesador vaya a realizar una tarea de forma rápida es bastante compleja. Depende de muchos aspectos concretos, como la propia velocidad del procesador pero también de la localización de ese procesador, el tipo de tarea que tenga que ejecutar, los dispositivos, etc [1]. Hay que evitar situaciones que puedan retrasar la ejecución de un trabajo concreto. Por ejemplo, enviar un trabajo de gran cómputo a un procesador que ya esté ejecutando otra tarea pesada, habiendo libre otro procesador.

Flexibilidad

Gracias a la flexibilidad se consigue una de las principales ventajas que ofrece un sistema distribuido, que su funcionamiento no tiene que estar ligado a ningún ordenador concreto de la red. Es decir, que cualquier ordenador puede suplir a otro en caso de avería o fallo.

En un proyecto como el diseño de un sistema operativo distribuido tiene que ser posible realizar cambios y actualizaciones que mejoren el funcionamiento del sistema. Esta necesidad provoca una diferencia en la arquitectura del núcleo del sistema, que da lugar a dos arquitecturas: núcleo monolítico ó micronúcleo. Mientras el monolítico ofrece todas las funciones básicas integradas en el núcleo, el micronúcleo solamente incorpora las fundamentales, por ejemplo: el control de los procesos y la comunicación entre ellos y la memoria. El resto de servicios se cargan dinámicamente a partir de servidores en el nivel de usuario.

Por lo general los S.O de núcleo monolítico son sistemas que tienen un núcleo grande y complejo que engloba todos los servicios del sistema. Está programado de forma no modular y tiene un rendimiento mayor que un micronúcleo. Sin embargo, cualquier cambio a realizar en cualquier servicio requiere la parada de todo el sistema y la 'recompilación' del núcleo. Un ejemplo de este tipo de S.O es UNIX.

Por el contrario, la arquitectura de micronúcleo se basa en una programación altamente modular. Gracias a esta característica tiene un tamaño mucho menor que el monolítico. Además, como consecuencia de esa programación, el refinamiento

y el control de errores son más rápidos y sencillos. También la actualización de los servicios es más sencilla y ágil, ya que sólo es necesaria la 'recompilación' del servicio y no de todo el núcleo. La desventaja es que el rendimiento se ve afectado negativamente.

En la actualidad, la mayoría de sistemas operativos distribuidos en desarrollo tienden a un diseño de micronúcleo. Los núcleos contienen menos errores y son más fáciles de implementar y de corregir. El sistema pierde ligeramente en rendimiento, pero a cambio consigue un gran aumento de la flexibilidad.

Escalabilidad

El concepto de escalabilidad da idea de si se mantiene el rendimiento a medida que el número de nodos aumenta. Un sistema que no tiene una buena escalabilidad, puede funcionar bien con diez nodos pero no con cien, o por lo menos no dar un rendimiento diez veces mayor como es de esperar. Esto es debido a que puede ocurrir que ciertas soluciones válidas para unos cuantos ordenadores, no sean aplicables para varios miles. Además con el tipo de red (ya sea LAN o WAN) puede pasar lo mismo y lo que funciona eficientemente en un sistema no lo haga en otro. Este concepto es un punto clave en los sistemas distribuidos y por lo tanto también en los S.O distribuidos.

Fiabilidad

Para conseguir la fiabilidad de todo el sistema se usa la redundancia, es decir, la información no tiene que estar almacenada en un solo servidor de archivos, por lo menos tiene que estarlo en dos máquinas. Mediante la redundancia de los principales archivos (o de todos) se evita que en caso de fallo del servidor se bloquee todo el sistema, ya que se tiene una copia idéntica de los archivos en otro ordenador.

También existe otro tipo de redundancia más compleja referida a los procesos. En este caso, consiste en que las tareas críticas pueden enviarse a varios procesadores independientes. De esta forma el primer procesador realiza la tarea normalmente, pero si por cualquier motivo ese procesador falla, dicha tarea pasa a ejecutarse a otro procesador.

1.2.2. Comunicación entre procesos

La diferencia principal entre un sistema distribuido y un sistema con un procesador es la comunicación entre procesos. En sistemas con un único procesador la comunicación se lleva a cabo mediante el uso de memoria compartida entre dichos procesos. Sin embargo, en los sistemas distribuidos, al no tener conexión física entre las memorias de los equipos, la comunicación se realiza mediante transferencia de mensajes.

Los sistemas centralizados pueden hacer IPC con procesos de otros equipos gracias al uso de librerías externas (MPI, RPC,...). Por su parte los sistemas distribuidos proporcionan en sus núcleos llamadas al sistema para resolver esto, de manera transparente. El cliente envía un mensaje al servidor y este envía de regreso una respuesta al cliente. Estos sistemas también utilizan llamadas a procedimientos remotos (RPC), en la que un proceso cliente que se ejecuta en una máquina, llama a un procedimiento que se ejecuta en otra máquina.

1.2.3. Sincronización en sistemas distribuidos

Una vez vista la forma de comunicarse entre los procesos en este tipo de sistemas, es también importante la forma en que los procesos cooperan y se sincronizan entre sí. Por ejemplo, la forma de implantar las regiones críticas o asignar recursos en un sistema distribuido.

En los sistemas con una sola CPU, los problemas relativos a regiones críticas, la exclusión mutua y la sincronización, se resuelven en general mediante métodos tales como los semáforos y monitores. Pero estos métodos no son válidos para sistemas distribuidos, ya que siempre se basan en la existencia de memoria compartida. Otro problema para realizar la sincronización es la falta de un reloj común o alguna fuente precisa de tiempo global.

La mayoría de S.O distribuidos tienen un paquete de hilos para realizar la sincronización y utilizan diversos algoritmos para sincronizar los relojes de todas las máquinas del sistema distribuido.

1.2.4. Modelos de organización de los equipos

Comúnmente se utilizan dos modelos de organización de los equipos: el modelo de estación de trabajo aislada y el modelo de estación de trabajo distribuida. En el primer modelo, cada usuario tiene su propia estación de trabajo y en ocasiones puede ejecutar procesos en las estaciones de trabajo inactivas. En el segundo, todas las instalaciones de cómputo son un recurso compartido. Los equipos se asignan de manera dinámica a los usuarios conforme sea necesario y regresan a un fondo al terminar el trabajo.

Además en los sistemas distribuidos se tiene una colección de equipos por lo que se necesita un algoritmo para asignar los procesos a dichas máquinas. Tales algoritmos pueden ser deterministas o heurísticos, centralizados o distribuidos, óptimos o subóptimos, locales o globales, iniciados por el emisor o por el receptor. Aunque los procesos se planifican por lo general de manera independiente, se puede mejorar el funcionamiento mediante la coplanificación, para garantizar que los procesos que deben comunicarse, se ejecutan al mismo tiempo. Una de las cuestiones en las que se centra este proyecto es en estos algoritmos encargados en asignar tareas.

Habitualmente cada procesador hace su planificación local sin preocuparse por lo que hacen los demás procesadores. También hay varios algoritmos de coplanificación que toman en cuenta los patrones de comunicación entre los procesos durante la planificación, para garantizar que todos los miembros de un grupo se ejecuten al mismo tiempo.

1.2.5. Sistemas distribuidos de archivos

Un componente fundamental dentro de un sistema operativo distribuido es el sistema de archivos. Su tarea es almacenar los programas y los datos, y tenerlos disponibles cuando sea necesario. También es necesario un servidor de ficheros cuya función es permitir el acceso remoto a archivos almacenados en él o directamente accesibles por este.

Además puede haber uno o varios servidores pero los clientes no deben necesitar conocer cuántos hay ni su posición o función. Lo ideal es que se vea como un sistema de ficheros en un sistema operativo centralizado (por ejemplo UNIX).

1.2.6. Modelos de Sistemas Distribuidos

Existen dos topologías principales a la hora de formar un sistema distribuido. Son las siguientes:

Grupos de Procesadores

En este modo, al abrir una sesión, el sistema asigna al usuario un procesador, pudiendo hacerlo en función de la sobrecarga, la proximidad o cualquier otro parámetro que fije el diseñador. Un método más sencillo y menos comprometido es hacer que el usuario elija el nodo al que quiere conectarse, pero se pierde en transparencia y puede dar lugar a fuertes desequilibrios en el sistema.

La topología se basa en varios ordenadores unidos a través de una red de comunicaciones, en la cual se pueden conectar los usuarios a través de sus ordenadores personales. Es un paso en la evolución de los sistemas distribuidos pero falta un sistema de ficheros transparente y global desapareciendo la idea de nodos.

Cliente-Servidor

En este modelo se cuenta con clientes que dan acceso a usuarios a los sistemas y servidores que contienen la información, prestan los servicios, etc. Una máquina puede ser cliente en un servicio y servidor en otro. Es el modelo más conocido y más ampliamente adoptado en la actualidad.

Se caracteriza por tener un conjunto de procesos servidores y una colección de procesos clientes. Los recursos compartidos son mantenidos y manejados por los procesos servidores, mientras que los procesos clientes llevan a cabo tareas que requieren acceso a esos recursos hardware o software compartidos.

Éste modelo se ha extendido y es utilizado en los sistemas actuales con servicios manejando diferentes tipos de recursos compartidos, pero no es posible que todos los recursos que existen en un sistema distribuido sean manejados y compartidos de esta manera. Algunos tipos de recursos deben permanecer locales a cada computadora de cara a una mayor eficiencia, como la RAM, el procesador o la interfaz de red local. Estos recursos clave son manejados separadamente por un sistema operativo en cada máquina.

Aunque el modelo cliente-servidor no satisface todos los requisitos necesarios para todas las aplicaciones distribuidos, es adecuado para muchas de las aplicaciones actuales y provee una base efectiva para los sistemas operativos distribuidos de propósito general.

1.3. Sistema Operativo Plan 9

Una vez explicado qué es un sistema operativo distribuido, sus ventajas e inconvenientes y sus principales características es el momento de entrar en detalle con el sistema usado para la realización de este proyecto, el sistema operativo Plan 9.

1.3.1. Historia

A finales de la década de los 80, algo se comenzó a gestar en el corazón de los laboratorios Bell, el núcleo de investigación y desarrollo de ATT, de donde surgió Unix. Se comenzó a trabajar en un nuevo proyecto de sistema operativo distribuido denominado Plan 9, en honor a la peor película de ciencia ficción de la historia: "Plan 9 del espacio exterior" de Ed Wood. Sus diseñadores y autores originales fueron Ken Thompson, Rob Pike, Dave Presotto y Phil Winterbottom [2].

Plan9 arrancó con la idea de solucionar algunos problemas de Unix que los ingenieros de Bell consideraban demasiado profundos para arreglarlos. Éste nuevo sistema operativo introdujo algunas innovaciones que tuvieron influencia en sistemas posteriores, pero hasta no hace muchos años Plan 9 estuvo bajo una licencia propietaria que no ayudó a popularizarlo entre los usuarios. Con el paso de los años la Licencia se fue liberando, y finalmente en 2003 Plan 9 se libera bajo una licencia de software libre reconocida. Durante ese tiempo Bell Labs y una pequeña comunidad de programadores han continuado perfeccionando y desarrollando Plan 9.

En la actualidad, Plan 9 es ofrecido a usuarios individuales e instituciones académicas libre de cargo, para uso no comercial. Es un S.O que constituye un sistema escalable, eficiente y con uso seguro en red y además se trata de un S.O distribuido. Se trata de un sistema en el cual coexisten por igual servidores de ficheros, de CPU, locales y de red.

1.3.2. Herencia de Unix

La idea original al diseñar Plan 9 consistió en realizar un moderno S.O partiendo de cero que respondiera a las necesidades tecnológicas actuales y que supliera todas las carencias respecto a la escalabilidad dinámica en red que tenía Unix [3]. La potencia del Plan 9 se ve cuando se instala en red, no en una máquina individual. Aunque el sistema se realizó desde cero comparte algunas características con Unix como:

- Se encuentra escrito en C, lenguaje adecuado para la programación de este tipo de sistemas.
- La interfaz de usuario es sencilla y potente, ofreciendo un refinamiento del shell de Unix, aunque compleja de aprender, ya que no fue pensada para uso hogareño.
- Dispone de un sistema de ficheros jerárquicos anclado a un superfichero raíz, permitiendo montar y desmontar volúmenes como en Unix.
- Al igual que en Unix, el sistema se encuentra constituido por ficheros, incluso los dispositivos de Entrada/Salida, pero con ideas propias que se verán más adelante.

1.3.3. Ideas propias del Sistema Operativo Plan 9

A parte de las ideas que conserva de Unix, Plan 9 tiene muchas ideas de creación propia. Por ejemplo, un terminal gráfico con sistemas de ventanas denominado *rio*. Este terminal puede ser cualquier computadora conectada a la red que tenga instalado Plan 9 con una CPU propia y con sistemas de ficheros propios. Pero se caracteriza como interfaz de acceso para usuarios humanos.

En el sistema de archivos se introduce el concepto de visibilidad de usuario. Es decir, como todos los S.O modernos es un sistema multiusuario, pero a diferencia de Unix no muestra un único sistema de ficheros a todos los usuarios y sólo distingue los permisos de acceso. Plan 9 muestra un mismo sistema de archivos en forma diferente

para cada usuario, ya que define la capa superior de visibilidad, y es llamada espacio de nombres.

La gestión de los recursos es tal que desaparece la separación entre recursos locales y remotos. Para ello se incorpora en el núcleo el protocolo '9P' (que reemplaza al TCP/IP).

1.3.4. Características del Sistema Operativo

El espacio de nombres

Como se ha mencionado antes, cada proceso dispone de una visibilidad particular del sistema de archivos denominado espacio de nombres. Es decir, dos procesos pueden ver sistemas de archivos completamente diferentes.

Plan 9 lleva este concepto al extremo, ampliándolo a todo el sistema de archivos. Todo espacio de nombres en Plan 9 cuenta con una raíz, y en ella se halla el directorio '/bin' que resulta de la unión de todos los directorios que tienen ejecutables visibles para el proceso. De esta forma se elimina la variable de entorno 'PATH' de UNIX. Esta visibilidad es configurable tanto por el usuario como por el administrador.

La única semejanza que hay en Unix es el dispositivo '/dev/tty', el cual es un dispositivo que se halla implementado en el kernel por lo que cada proceso ve un '/dev/tty' diferente que corresponde al terminal en que se halla conectado.

Recursos como archivos

En Plan 9 todos los recursos son representados como archivos en un sistema de archivos jerárquicos. Este S.O amplía el concepto de archivo al de nombre para así superar las limitaciones prácticas surgidas en Unix a lo largo del tiempo (recursos incluidos fuera del sistema de archivos). El nombre corresponderá a una ruta única para cada objeto, ya sea éste un archivo, usuario, pantalla o computadora. Además, cada usuario podrá definir diferentes nombres para un mismo objeto "real", siendo posible crear un entorno personalizado a través de varios objetos en su espacio de nombres. Por eso se dice que Plan 9 es un sistema mucho más enfocado al contexto.

El Protocolo 9P

Como se ha dicho anteriormente la gestión de los recursos en Plan 9 es tal que desaparece la separación entre recursos locales y remotos y para ello se incorpora en el núcleo el protocolo '9P'.

Este protocolo es el que permite tratar a todos los recursos como ficheros, ya sean locales o remotos, con lo cual no es necesario utilizar el NFS, ya que se permite el acceso a dispositivos de forma remota.

Por otra parte el protocolo 9P ofrece una semántica única de accesos a archivos que implementa procedimientos RPC sobre el protocolo IL (Intenet Link), un nuevo protocolo fiable que se monta sobre una IP.

Mecanismos de Autenticación

En los sistemas operativos distribuidos es muy importante la seguridad en las comunicaciones entre las distintas máquinas del sistema distribuido. El mecanismo de autenticación de Plan 9 sigue el esquema 'Kerberos', es decir, no se envía el 'password' por la red. En su lugar, se emplea un mecanismo de tickets encriptados facilitados por un servidor de autenticación.

En Plan 9 no existe el concepto de 'superusuario' ni tampoco la idea de programas con 'setuid' (en Unix, permisos de acceso). Por otro lado, el servidor de archivos de Plan 9 no corre programas de usuario, solo se limita a dar acceso a archivos autorizados.

Sistema de Archivos "/proc"

Algunos Unix como el SV y Linux emplean un sistema de archivos virtual denominado '/proc'. Con él se puede acceder a procesos, entornos de procesos, dispositivos y recursos. En Plan 9 se incorpora este mecanismo. Así bajo el directorio '/proc' (visible por todos los procesos) existen otros subdirectorios asociados con cada proceso del sistema y que toman como nombre el número del proceso correspondiente. Dentro de esos directorios hay otros archivos como ctl, mem, status, etc. que permiten acceder a través de las funciones E/S de la API. Los mensajes de lectura y escritura son de texto, y se puede perder algo de prestaciones aunque se gana en

claridad.

De esta forma, el comando 'ps' de Unix se puede implementar leyendo las variables 'status' de todos los procesos como 'cat /proc/*/status', aunque también existe como 'ps'.

Entorno de Creación de Procesos

Otra de las nuevas características de Plan 9 es la forma de acceder al entorno de un proceso, ya que también se realiza a través del espacio de nombres, en el directorio '/env'. El espacio de nombres, como se ha visto anteriormente, será diferente para cada proceso. Una forma de ver todas las variables asociadas a un proceso podría ser: 'cd env; echo *'.

Para la creación de procesos hijos, el proceso padre hace uso de la primitiva 'rfork'. Esta llamada acepta un mapa de bits como argumento que describe cómo se compartirá el espacio de nombres entre padre e hijo. Más adelante se explicará con más detenimiento.

1.3.5. Planes futuros

Plan 9 nunca ha sobrepasado a Unix en popularidad, y continúa como una herramienta de investigación. La siguiente cita de Eric S. Raymond resume de forma adecuada por qué Plan 9 no superó a Unix: "Plan 9 falló simplemente porque se quedó corto de ser una mejora lo suficientemente convincente de Unix como para desplazar a su ancestro. Comparado a Plan 9, Unix tiene obvios puntos herrumbrados, pero hace el trabajo lo suficientemente bien como para mantener su posición. Esta es una lección para los arquitectos de sistemas ambiciosos: un código base que es lo suficientemente bueno es el peor enemigo de una mejor solución."

1.4. Sistema de colas y planificación de procesos

1.4.1. Definición de proceso

Un proceso es una instancia de ejecución de un programa y está caracterizado por su contador de programa, su palabra de estado, sus registros del procesador, su segmento de texto, pila y datos, etc. A su vez, un proceso es un concepto dinámico, mientras que un programa es un concepto estático. Es posible que un programa sea ejecutado por varios usuarios en un sistema multiusuario. Por cada una de estas ejecuciones existirá un proceso, con su contador de programa, registros, etc.

El sistema operativo necesita el concepto de proceso para poder gestionar el procesador mediante la técnica de multiprogramación o de tiempo compartido. De hecho, el proceso es la unidad planificable o de asignación de la CPU.

1.4.2. Estados de un proceso

Durante su vida, un proceso puede pasar por una serie de estados (ver figura 1.1). A continuación, se explican dichos estados, en donde se hace referencia únicamente a sistemas monoprocesadores.

- *Preparado o listo:* este estado involucra la creación del proceso y suele ser el primer estado a adoptar por él. En este estado el S.O ya reconoce al proceso como tal. La característica principal de este estado es que el proceso posee todos los recursos para su ejecución menos la CPU. Es decir, está preparado para ejecutarse pero no se está ejecutando [2]. Cuando la CPU sea liberada todos los procesos listos competirán por ella. Esto se realizará a través del Algoritmo de planificación existente.

- *En ejecución:* en este estado el proceso ya posee todos los recursos, incluido el procesador. Aquí es donde un proceso realiza todas las operaciones que implementa.

- *Suspendido o bloqueado:* el proceso no dispone del procesador y además carece de algún otro recurso. Por lo que no sólo no se está ejecutando, si no que no

compite por ello. Normalmente un proceso es bloqueado cuando necesita de algún dato de E/S para proseguir con la ejecución. Debido a que el tiempo de la operación de E/S es impredecible y habitualmente largo, hablando en términos computacionales, el S.O prefiere bloquear el proceso y planificar otro. El proceso permanece bloqueado hasta que se produce la operación de E/S y cuando esta llega a su fin el proceso continúa con su ciclo.

(Diagrama de transición de estados)

1.4.3. Planificación de procesos

Una vez definido lo que es un proceso y sus posibles estados hay que explicar cómo se determina el orden con el que se van a ejecutar las tareas en un procesador. Por ello surge la planificación de procesos, que hace referencia a un conjunto de políticas y mecanismos incorporados al sistema operativo, para determinar el orden en que deben ser ejecutados los trabajos. El objetivo de la planificación es optimizar el rendimiento del sistema [2].

1.4.4. Objetivos de la Planificación de procesos

La planificación de procesos tiene como principales objetivos los siguientes:

- *Equidad:* Todos los procesos deben ser atendidos.
- *Eficacia:* El procesador debe estar ocupado el 100 % del tiempo.
- *Tiempo de respuesta:* El tiempo empleado en dar respuesta a las solicitudes del usuario debe ser el menor posible.
- *Tiempo de regreso:* Reducir al mínimo el tiempo de espera de los resultados esperados por los usuarios por lotes.
- *Rendimiento:* Maximizar el número de tareas que se procesan por cada hora.

1.4.5. Niveles de planificación

La planificación del procesador se clasifica según la escala relativa de tiempo en que es realizada. Existen tres tipos: planificador a largo plazo, a medio plazo y a corto plazo.

Planificador a largo plazo

Determina qué nuevos programas son aceptados para ser procesados por el sistema, o sea determina el grado de multiprogramación. Una vez admitidos, se convierten en procesos que son agregados a la cola de Planificación a corto plazo. En algunos sistemas, son agregados a la cola de Planificación a medio plazo, ya que los procesos creados recientemente comienzan en una condición de suspendidos.

El objetivo primordial es el de dar al planificador de la CPU una mezcla equilibrada de trabajos, tales como los limitados por la CPU (debido a que utiliza mucho la CPU) o la E/S. Por ejemplo, cuando la utilización de la CPU es baja, el planificador puede admitir más trabajos para aumentar el número de procesos listos y, con ello, la probabilidad de tener algún trabajo útil en espera de que se le asigne la CPU. A la inversa, cuando la utilización de la CPU llega a ser alta y el tiempo de respuesta comienza a reflejarlo, el planificador a largo plazo puede optar por reducir la frecuencia de admisión de trabajos.

Normalmente, se invoca al planificador a largo plazo siempre que un proceso termina. La frecuencia de invocación depende de la carga del sistema, pero generalmente es mucho menor que la de los otros dos planificadores. Esta baja frecuencia de uso hace que este planificador pueda permitirse utilizar algoritmos complejos, basados en las estimaciones de los nuevos trabajos.

Este tipo de planificador también se puede ver como si fuese un sistema de colas. Los algoritmos que se van a realizar en este proyecto y que serán explicados en el capítulo 2 serán de este tipo.

Planificador a medio plazo

Este planificador es el encargado de regir las transiciones de procesos entre memoria principal y secundaria. Además, actúa intentando maximizar la utilización de los

recursos. Por ejemplo, transfiriendo siempre a memoria secundaria procesos bloqueados, o transfiriendo a memoria principal procesos bloqueados únicamente por no tener memoria. El planificador a medio plazo se ejecuta un poco más frecuente que la planificación a largo plazo.

Planificador a corto plazo

Determina cuál es el próximo proceso a ejecutar. Es invocado cada vez que ocurre un evento que pueda causar una suspensión (interrupciones del reloj, interrupciones de entrada/salida, llamadas al sistema operativo, señales) o que pueda asegurarle una mayor prioridad a un proceso a favor de otro aunque se esté ejecutando. También conocida como *dispatcher*, es la que se ejecuta más frecuentemente.

1.4.6. Algoritmos de planificación

Vistos los niveles de la planificación, se van a explicar algunos de los algoritmos utilizados para planificar las tareas en un procesador. Hay dos aspectos importantes a contemplar en los diferentes algoritmos de planificación: la función de selección y el modo de decisión.

La función de selección determina qué proceso, entre los procesos listos, es seleccionado para ejecutarse a continuación; puede estar basada en prioridad, en los requerimientos de los recursos, o en las características de ejecución del proceso.

El modo de decisión especifica los instantes en el tiempo en los cuales la función de selección es aplicada, y puede ser sin preferencia o con preferencia. Si es con preferencia, el proceso que se está ejecutando actualmente puede ser interrumpido y movido al estado de 'listo' por el sistema operativo. En el caso contrario, el proceso seleccionado pasará a ejecutarse cuando el actual termine o se bloquee. A continuación, se explican algunos de los principales algoritmos de planificación:

Primero en llegar primero en ser servido

Conocido como FCFS (First Come First Served). Este algoritmo emplea una cola de procesos, asignando un lugar a cada proceso por el orden de llegada. Cuando el proceso llega es puesto en la cola después del que llegó antes que él y pasa a estado

'listo'. Además, si un proceso comienza a ejecutarse no se le interrumpe sino que se le deja terminar.

Prioridad al más corto

Este planificador también es llamado SJF (Shortest Job First). Consiste en que el proceso que se encuentra en ejecución cambiará de estado voluntariamente, es decir, no tendrá un tiempo de ejecución determinado para el proceso. A cada proceso se le asigna el tiempo que usará cuando vuelva a estar en ejecución, y se irá ejecutando el que tenga un menor tiempo asignado. Si se da el caso de que dos procesos tengan igual valor en ese aspecto, se emplea el algoritmo FCFS.

Round Robin

En este caso a cada proceso se le asigna un tiempo determinado para su ejecución, el mismo tiempo para todos. En caso de que un proceso no pueda ser ejecutado completamente en ese tiempo se continuará su ejecución después de que todos los procesos restantes sean ejecutados durante el tiempo establecido. Es un algoritmo basado en FCFS que trata la cola de procesos en estado de listos como una cola circular.

Planificación por prioridad

En este tipo de planificación a cada proceso se le asigna una prioridad siguiendo un criterio determinado, y de acuerdo con esa prioridad se establece el orden en que se atiende cada proceso.

Planificación garantizada

Para realizar esta planificación el sistema tiene en cuenta el número de usuarios que deben ser atendidos. Para un número "n"de usuarios se asignará a cada uno un tiempo de ejecución igual a 1/n.

Planificación de Colas Múltiples

El nombre proviene de MQS (Multilevel Queue Schedulling). En este algoritmo la cola con los procesos que se encuentran en estado de listos es dividida en un número determinado de colas más pequeñas. Los procesos son clasificados mediante un criterio, para determinar en qué cola será colocado cada uno cuando quede en estado de listo. Cada cola puede manejar un algoritmo de planificación diferente a las demás.

1.5. Equilibrio de carga

En el punto 1.1 se explicó en qué consiste un sistema distribuido o cluster, de forma rápida, es un grupo de ordenadores unidos mediante una red de alta velocidad con el objetivo de mejorar el rendimiento y la disponibilidad.

1.5.1. Algoritmos de equilibrio de carga

Una de las principales ventajas que tiene un sistema distribuido es que se puede formar con ordenadores poco potentes y conseguir muy buenos resultados. La parte negativa es que no siempre los ordenadores tendrán el mismo hardware, generando con ello sistemas heterogéneos que requieren herramientas que corrijan los desequilibrios. Por ejemplo, tener nodos menos potentes con un exceso de carga que ralentiza la ejecución de las tareas, mientras existen nodos con potencia excedente que están siendo infrautilizados [4]. Además de estos desequilibrios, cada nodo puede tener un sistema operativo distinto, lo cual complica las cosas a la hora de intentar trabajar en conjunto.

Estos problemas pueden ser solucionados mediante algoritmos de equilibrio de carga, los cuales consisten en encontrar una solución asignando tareas a los nodos, para que aproximadamente cada nodo tenga la misma cantidad de trabajo. Uno de los objetivos del proyecto es realizar un programa que se encargue de esto mismo. Este programa recibirá una lista de tareas y tendrá que repartirlas en los diferentes nodos disponibles.

Existen dos tipos de algoritmos de equilibrio de carga:

- Algoritmo estático: este tipo de algoritmo se basa en repartir las tareas antes de ser ejecutadas, es decir, en el tiempo de compilación. Esto tiene la ventaja de que no existe sobrecarga durante la ejecución pero no tiene en cuenta ningún factor que produzca un cambio en dicha ejecución.

- Algoritmo dinámico: en este caso las tareas si son repartidas durante la ejecución del proceso. Por lo que son más adaptables a cambios que puedan producirse en la ejecución. La parte negativa es que se crea una sobrecarga en los nodos. Dentro de estos algoritmos se pueden distinguir dos tipos: centralizados o distribuidos.

Los algoritmos realizados para este proyecto son todos de tipo estático. En el siguiente capítulo se verán en detalle.

Capítulo 2

Desarrollo

Como se ha explicado en el prólogo el objetivo de este proyecto es conocer el funcionamiento y las características de los sistemas operativos distribuidos, siendo el escogido para ello el Plan 9. Para conseguir lo dicho, se van a desarrollar una serie de algoritmos de planificación para distribuir tareas en un sistema distribuido, por lo que será necesario el conocimiento de las herramientas que el S.O Plan 9 proporciona al usuario/programador.

En este capítulo se va a abordar el desarrollo realizado durante la elaboración del presente proyecto.

2.1. Instalación del Plan 9

2.1.1. Instalación del Servidor

En este primer punto se va a explicar la instalación necesaria para poner en funcionamiento el servidor de la 4º edición del S.O Plan9. La instalación no es tan trivial como la de un S.O moderno y hay ciertos pasos que tienen algo de complicación, de ahí que se explique todo el proceso paso a paso.

Antes de comenzar con el proceso de instalación hay que comprobar que el ordenador cumple los requisitos mínimos que necesita el S.O Plan9:

- Más de 32MB de memoria RAM [3].
- Como mínimo disponer de 300MB de memoria en disco.

Como se puede apreciar son unos requisitos muy básicos para cualquier ordenador que se utiliza hoy en día. Además, hay que tener cuidado con los controladores SATA ya que Plan 9 tan solo soporta algunos de ellos, pero si se trata de controladores IDE no debería haber ningún problema.

Para comenzar con la instalación, se obtiene la distribución de Plan 9 de la web oficial. Al estar disponible bajo una licencia de software libre, esto se hace sin coste alguno. Para realizar la instalación, se puede grabar la distribución en un CD. Después se arranca el ordenador desde el sitio en el que se tenga la imagen del Plan 9, por ejemplo desde el lector de CDs. El sistema preguntará donde se encuentran los archivos de la distribución("boot from:. . . "). En el caso de encontrarse grabados en un CD, sería: "boot from: sdC1!cdboot!9pcflop.gz", donde sdC1 es el lector de CDs y 9pcflop.gz es el kernel utilizado para cargar desde un disquete de arranque. Si lo encuentra sin problemas preguntará sobre la DMA (Direct Memory Access), el puerto del ratón y la configuración VGA.

A continuación, si la tarjeta de video es soportada, el sistema arrancará en modo gráfico con el sistema de ventanas *rio*. Si no, saldrá una 'prompt' para seguir en modo texto, en la que habría que poner: "inst/textonly" para continuar la instalación.

A partir de este momento el sistema pedirá que se vayan realizando una serie de tareas para ir avanzando en la instalación, las cuales se explican a continuación:

La primera llamada *'Configfs'* sirve para elegir el tipo de sistema de ficheros a instalar. Se da la opción al usuario de instalar solamente el fossil o conjuntamente con el venti. Fossil es el sistema de archivos por defecto del Plan 9. Lo que lo hace diferente de otros sistemas de archivos es que puede tomar instantáneas de todo el sistema de ficheros ejecutando una orden o con un intervalo de tiempo prefijado [5]. Por otra parte el venti es un servidor de almacenamiento de archivos en red que almacena permanentemente bloques de datos y que está pensado para almacenar de forma permanente instantáneas del sistema fossil. La opción elegida en este proyecto es únicamente usar el servidor de ficheros fossil. El sistema venti es muy útil para almacenar copias de seguridad del sistema pero no es necesario para la elaboración de este proyecto.

La siguiente tarea es *'Partdisk'* en la cual se elige el disco duro donde se quiere

instalar el Plan 9, por lo que el sistema mostrará una tabla con las particiones de los discos del ordenador. Como durante este proyecto solo se quiere tener el Plan 9 en funcionamiento en los equipos, se borrarán todas las demás particiones con el comando "d nombre partición". También es necesario instalar un cargador compatible con Plan9 en el MBR (Master Boot Record) del equipo. Plan9 suministra uno llamado 9load. El sistema automáticamente crea una partición para el Plan 9, en el caso de que se esté de acuerdo, se escribe en disco dicha partición con el comando "w" y se sale del programa de particionado mediante el comando "q".

En el paso anterior se crea una única partición FAT para Plan9 en la que se crea un sistema de ficheros. Por defecto a esta partición se la llama y se la conoce como 9fat. Sin embargo, Plan9 subdivide ésta partición para albergar una serie de sistemas de ficheros especiales. La tarea *'Prepdisk'* realiza esta tarea de subdivisión. El sistema muestra donde ha encontrado la partición creada anteriormente e indicando dónde está dicha partición, muestra una tabla con las subparticiones que va a crear (9fat, nvram, fossil y swap). Para escribirlas en disco se usa el comando "w" y para salir "q", como se ha hecho en anteriores ocasiones.

A continuación, la tarea *'fmtfossil'* da formato a la subpartición creada para el servidor fossil. Una vez la partición tiene el sistema de ficheros correctamente inicializado, la tarea *'Mountfs'* 'monta' el sistema de ficheros fossil en el sistema. Es decir, habilita la subpartición creada con su sistema de ficheros para poder acceder a los datos que tiene, agregándola a la estructura de directorios del sistema.

En este momento el equipo está correctamente particionado y con todos los sistemas de ficheros creados. Por lo que es el momento de copiar los archivos de la distribución del CD al equipo. Para ello, con la tarea *'Configdist'* hay que especificar al sistema donde se encuentra la distribución a instalar. En este caso sería de manera 'local', al estar en el CD, pero también es posible instalar desde la red.

La tarea *'Mountdist'* montará en el disco la distribución escogida que en este proyecto está ubicada en '/dev/sdD0/data'. A continuación, se le indica la ruta exacta de la distribución o con la ayuda de la shell se va al directorio en el que se encuentre, usando los comandos 'cd' (accede a una carpeta), 'lc' (muestra el contenido de un directorio) y 'exit'.

Por último, la tarea *'Copydist'* copia los archivos que se han localizado anteriormente al sistema de ficheros y la tarea *'Bootsetup'* configura la forma de arranque, dando a elegir varias formas (crear un floppy disk, arrancar desde la partición del plan 9,...). Se elige 'plan9' para que ponga la partición de este como activa (partición que arranca por defecto) y por lo tanto queda finalizada la instalación del servidor del Plan 9.

2.1.2. Añadir nodos al servidor

Una vez arrancado el servidor con el plan 9 se pueden añadir nodos para poder formar un sistema distribuido. En este proyecto se ha decidido que los nodos arranquen vía PXE a través de dicho servidor para no tener que instalar de nuevo el S.O en ellos. Conviene mencionar algunos aspectos de este modo de arranque.

Preboot Execution Envinment (PXE)

El protocolo PXE (Preboot eXecution Environment) consiste en una combinación de los protocolos DHCP (Dynamic Host Configuration Protocol) y TFTP (Trivial File Transfer Protocol) con pequeñas modificaciones en ambos. El DHCP es utilizado para configurar los parámetros de red y localizar el servidor de arranque apropiado, mientras que TFTP permite descargar ficheros como el programa inicial de arranque (bootloader).

Cuando un equipo se inicia a través de PXE envía, a través de la tarjeta de red, un paquete broadcast del protocolo DHCP con algunas opciones específicas de PXE al puerto 67/UDP. Estas opciones indican que el firmware es capaz de manejar PXE, pero serán ignoradas por los servidores DHCP estándar. Si en el segmento de red existe un servidor PXE, este recibirá la petición broadcast y a través del protocolo DHCP se configurará la dirección IP del equipo.

Además las opciones especiales de PXE especificarán la ubicación de los archivos necesarios para iniciar el arranque. El equipo utilizará el protocolo TFTP para descargar estos archivos y los ejecutará.

Conociendo en qué consiste el protocolo PXE se procede a explicar cómo añadir nodos al servidor.

En primer lugar es necesario que el servidor y el nuevo equipo (se le llamará cliente) se encuentren en la misma subred. Por ejemplo, conectándolos a través de un 'switch'. El objetivo es que el cliente arranque desde el servidor con la 4º edición del Plan 9 instalada y en funcionamiento. Como se ha mencionado anteriormente, el cliente no tendrá ningún S.O instalado, únicamente la configuración necesaria para arrancar vía PXE.

Para conseguir que el cliente arranque desde el servidor es necesario realizar una serie de tareas, estas son algunas de ellas: compilación e instalación del kernel del servidor de cpu, autentificación y ficheros, configuración de la red de trabajo, configuración de la forma de arranque de los nodos, creación y autentificación de usuarios [6].

Una vez conectados ambos ordenadores, hay que compilar e instalar el kernel 'pccpuf' en el servidor para tener un servidor de cpu, de autentificación y de ficheros.

```
cd /sys/src/9/pc
mk 'CONF=pccpuf' install
9fat:
cp 9pccpuf.gz /n/9fat/
acme /n/9fat/plan9.ini #set the bootfile to 9pccpuf.gz
```

Figura 2.1: Instalación del kernel.

Como se muestra en la figura 2.1, dentro de la carpeta '/sys/src/9/pc' se realiza una orden 'mk' que compila el kernel pccpuf. '9fat:' es un script que monta la partición FAT del disco en el directorio '/n/9fat'. A continuación, se copia el fichero de kernel creado al directorio anterior y se edita el fichero de configuración 'plan9.ini' para que el ordenador arranque con dicho kernel, 9pccpuf.gz.

También es necesario editar el proceso de arranque del sistema en el fichero '/rc/bin/cpurc', para que el servidor 'escuche' a otros computadores. Para ello hay que descomentar las líneas de código necesarias para el arranque de otro computador y las correspondientes al servidor de autentificación. En el fichero 'cpurc' vienen explicadas las líneas necesarias así que al ser un fichero de gran tamaño no se van a citar cada una de las líneas.

Instalado el kernel y editado el proceso de arranque se necesita un usuario para administrar el servidor. Por convenio a este usuario se le llama 'bootes'. También es necesario añadirle al grupo de usuarios y de administrador para que tenga suficientes permisos.

Como se puede observar en la figura 2.2, el usuario 'bootes' se crea a través de la consola del 'fileserver', llamada fscons, y se le dan los permisos dichos anteriormente. De esta forma se está creando el usuario en el servidor de archivos, no en el servidor de autentificación. Que se realizará más adelante.

```
con /srv/fscons
prompt: fsys main
main: uname bootes bootes
main: uname sys +bootes
main: uname adm +bootes
```

Figura 2.2: Creación de usuario.

Cada vez que se arranque el servidor o alguno de los nodos el sistema pedirá una información para autentificarse. Para cambiar la información por defecto por una elegida por el usuario, hay que escribir cualquier contenido en el 'nvram' del servidor, de manera que cuando el sistema lo lea sea erróneo y pida la nueva información para identificarse. 2.3

```
echo loquesea  /dev/sdC0/nvram
```

Figura 2.3: Ejemplo de invalidación de 'nvram'.

De esta forma al reiniciar el sistema se pedirá la información necesaria para entrar, esto sería un ejemplo:

Como se ve en la figura 2.4 el sistema pedirá: Nombre del usuario y del dominio, y contraseña del sector y del usuario. Esta información se necesitará cada vez que se quiera arrancar tanto el servidor como los clientes.

Cuando el usuario se conecta por primera vez, se debe ejecutar el comando '/sys/lib/bootes' (en este caso sería bootes) para configurar el directorio de dicho

```
authid: bootes
authdom: red.com
secstore: secstore password
password: bootes password
```

Figura 2.4: Información de autentificación.

usuario. Como se dijo al crear el usuario anteriormente, también es necesario crear dicho usuario en el servidor de autentificación. Eso se hace de la siguiente forma: 'auth/changeuser bootes'.

El servidor de cpu, de autentificación y de ficheros está configurado. Ahora es necesario configurar la red en la que van a trabajar los nodos que se quieren añadir.

Primero hay que configurar la dirección ip que le asignará el servidor al cliente cuando, al arrancar, éste detecte su dirección MAC. Por otra parte, hay que decirle al cliente el fichero que tiene que usar para el arranque. Todo esto se realiza en el archivo '/lib/ndb/local' del servidor.

A continuación se muestra cómo quedaría el fichero 'local' para conseguir lo dicho anteriormente:

En esta figura 2.5 se configura la dirección ip tanto del cliente como del servidor. También es necesario indicar la dirección MAC de cada ordenador y el nombre que se quiera que éste reciba. En el caso del cliente, 'bootf=/383/9pxeload' indica el fichero que se tiene que ejecutar al arrancar vía PXE. Por último, también se configuran cosas como: la ip de la puerta de enlace de la red, la máscara, las DNSs y el ordenador que se vaya a utilizar como servidor de autentificación (auth), cpuserver y fileserver, que en nuestro caso todos los servidores están en el mismo ordenador.

Con el archivo 'local' configurado, el servidor ya sabe que dirección ip tiene que darle al cliente cuando este arranque y además, el cliente sabe que fichero tiene descargarse para arrancar.

A continuación, hay que crear el fichero '/cfg/pxe/000bdb4f1b06' (la MAC del cliente en este caso) en el que se indica al cliente la configuración con la que debe arrancar. Para la configuración descrita en el fichero 'local', este fichero quedaría de la siguiente forma:

```
database=
 file=/lib/ndb/local
 file=/lib/ndb/common
# cliente
ip=192.168.0.3 sys=pc2 ether=000bdb4f1b06 bootf=/383/9pxeload
ipnet=red ip=192.168.0.0 ipmask=255.255.255.0
 ipgw=192.168.0.1
 dns=130.206.5.234
 dns=130.206.5.4
 auth=192.168.0.2
 cpu=192.168.0.2
 fs=192.168.0.2
 gw=192.168.0.2
# maestro
ip=192.168.0.2 sys=pc1 ether=000bdb808c4
ip=127.0.0.10 sys=localhost dom=localhost
```

Figura 2.5: Fichero de configuración 'local'.

En esta figura 2.6 se ve tanto la forma de arranque del cliente, que será por red, como el kernel que va a arrancar: 'bootfile=ether0!/386/9pcf'. También se observa que el cliente va a coger dicha información, a través del puerto 564, de la dirección 192.168.0.2: 'nobootprompt=tcp!192.168.0.2!564'. El resto son cosas de configuración como por ejemplo: la tarjeta de red, el puerto del ratón, la pantalla y su resolución, etc.

Una vez configurado el servidor para proporcionarle al cliente lo que éste necesita, hay que ejecutar los demonios dhcpd y tftpd en dicho servidor para 'escuchar' las posibles peticiones que reciba. Estos programas una vez son activados, están en continuo funcionamiento contestando peticiones de DHCP y TFTP respectivamente.

Por último, hay que crear un fichero normalmente llamado 'flproto', que atenderá las conexiones de la red a través del puerto 564 del cliente (como se ha visto en la figura 2.6). El puerto 564 será por donde se comunicarán el servidor y el cliente,

```
bootfile=ether0!/386/9pcf
nobootprompt=tcp!192.168.0.2!564
*debugload=1
*nobiosload=1
*nodumpstack=1
console=0
debugboot=1
dmamode=ask
partition=new
ether0=i82598
mouseport=ps2
monitor=xga
vgasize=1280x1024x8
```

Figura 2.6: Fichero de configuración de arranque.

ya que es el puerto que utiliza el sistema de archivos fossil.

```
cat flproto
fsys main config /dev/sdC0/fossil
fsys main open -c 3000
fsys main snaptime -s 60 -a 0500 -t 2880
listen tcp!*!564
cat flproto | fossil/conf -w /dev/sdC0/fossil
```

Figura 2.7: Fichero 'flproto'.

En esta figura 2.7 se ve como primero se crea el fichero 'flproto' en el que se 'escucha' el puerto 564: 'listen tcp!*!564' y a continuación, se sustituye el fichero antiguo por el nuevo.

De esta forma esta todo preparado para que el cliente pueda arrancar a través del servidor vía PXE.

2.2. Herramientas del S.O Plan 9 utilizadas

Una vez instalado el servidor del Plan9 y añadido clientes a éste, tal como se ha explicado en el punto anterior, se van a explicar las herramientas características de Plan 9 necesarias para la realización del programa implementado para distribuir tareas en el sistema distribuido.

2.2.1. Comando 'cpu'

Este comando arranca una shell 'rc' en la máquina que se le pasa como argumento. La entrada, salida y error estándar de la shell será el fichero '/dev/cons' en el espacio de nombres donde fue invocado el comando 'cpu'. Normalmente este comando se ejecuta en una ventana *rio* en un terminal, por lo que la salida irá a dicha ventana y la entrada vendrá desde el teclado. Por ello el comando 'cpu' es muy útil para ejecutar comandos o tareas en una máquina remota.

El espacio de nombres para la nueva shell es análogo al espacio de nombres donde se invoca el comando 'cpu', excepto para directorios dependientes de la arquitectura como /bin [7]. Además, el espacio de nombres es construido ejecutando '/usr/$user/lib/profile' y su raíz 'montada' en '/mnt/term'.

Por último, si en este comando se utiliza el argumento '-R', se ejecuta un programa que hace de demonio, es decir, dicho programa actúa en segundo plano permanentemente 'escuchando' conexiones. Por lo que es necesario ejecutarlo en la máquina remota si se quiere 'escuchar' por un puerto concreto (tcp17010) al comando 'cpu'. Por ejemplo, si se quiere ejecutar un programa utilizando el comando 'cpu' es necesario que se esté ejecutando el comando 'cpu -R' en la máquina donde se quiera ejecutar el programa. De esta forma el demonio 'escuchará' al comando 'cpu' y se podrá realizar la conexión.

2.2.2. Comando 'import'

El comando 'import' permite que cualquier fichero de un computador remoto sea importado al espacio de nombres local. Normalmente dicho fichero suele ser un directorio, por lo que el árbol de archivos completo dentro de ese directorio se

encuentra disponible.

Su funcionamiento es el siguiente: al ejecutar el comando, un proceso comienza en la máquina remota con la autorización del usuario, que hace el 'import' para realizar un trabajo local usando el servicio 'exportfs'. Este es un servidor de archivos a nivel de usuario que permite a los servidores de cómputo de Plan 9 exportar partes de un espacio de nombres a través de redes. Entonces 'Exportfs' actúa como un servidor de archivos: las operaciones en el árbol de archivos importados son ejecutadas en el servidor remoto y éste devuelve los resultados.

El puerto usado por defecto es el TCP 17007. Si el punto de montaje es omitido al ejecutar el 'import', éste usa el nombre del fichero (o directorio) que tiene en la máquina remota.

2.2.3. Fichero '/dev/sysstat'

Este fichero contiene diez datos estadísticos del ordenador, de los cuales algunos son utilizados en este proyecto. Algunos de ellos son los siguientes: número de procesadores, llamadas al sistema, carga media, idle time (tiempo no productivo) y tiempo de interrupciones. La *carga media* está en unidades de mili-CPUs. La información de esta carga se obtiene en función de la longitud de la cola de la CPU. De esta forma se consigue más información que midiendo el uso de la CPU. Esto es debido a que cuando un servidor está muy cargado, su utilización es probable que sea cercana al 100 % pero no es capaz de reflejar el nivel de carga exacta de utilización. Sin embargo, midiendo las longitudes de la cola de CPU se pueden reflejar la cantidad de carga en una CPU. Por ejemplo, dos sistemas, uno con 3 y el otro con 6 procesos en la cola, probablemente tendrán utilizaciones cerca del 100 % a pesar de que, obviamente, difieren. Es por esto que en Plan9 se miden los procesos que hay a la espera de entrar en el procesador y multiplicados por mil, al ser mili-CPUs. El *idle time* y el *tiempo de interrupciones* vienen dados en porcentaje.

2.2.4. Llamada al sistema 'rfork'

En Plan 9, los procesos se crean usando la llamada del sistema 'rfork'. Esta llamada toma un solo argumento, un vector de bits que especifica que recursos del

proceso padre deben ser compartidos, copiados o creados de nuevo en el proceso hijo. Los recursos controlados por 'rfork' incluyen el espacio de nombres, el entorno, la tabla de descriptores de ficheros, segmentos de memoria, y notas (equivalentes a señales en Unix).

Uno de los bits controla si 'rfork' creará un proceso nuevo; si el bit está en off, las modificaciones resultantes de los recursos ocurrirán en el proceso que hace la llamada. Por ejemplo, un proceso llama 'rfork(RFNAMEG) ' para desconectar su espacio de nombres del de su padre.

Algo que demuestra que 'rfork' es el modelo correcto es la variedad de formas en que es utilizado. Aparte del uso canónico de la rutina de librería 'fork', es difícil encontrar dos llamadas con el mismo conjunto de bits; los programas lo usan para crear muchas formas distintas de compartición y de asignación de recursos. Un sistema con solo dos tipos de procesos, procesos normales e hilos, no puede manejar esta variedad.

Hay dos formas de compartir memoria. Una es usando, un flag con 'rfork' que hace que todos los segmentos de memoria del padre sean compartidos con el hijo (excepto la pila que hay que hacerlo de forma independiente). La otra forma es enlazando un segmento nuevo de memoria usando la llamada del sistema 'segattach'. Este segmento será siempre compartido entre padre e hijo.

2.2.5. Llamadas al sistema 'mount', 'bind' y 'unmount'

Los procesos de usuario construyen espacios de nombres usando tres llamadas del sistema: 'mount', 'bind' y 'unmount'. La llamada 'mount' 'engancha' un árbol dado por un servidor de ficheros al espacio de nombres actual. Antes de llamar a 'mount', el cliente debe (por medios externos) adquirir una conexión con el servidor en forma de un descriptor de fichero que puede ser leído y escrito para transmitir mensajes 9P. Este descriptor de fichero representa una 'pipe' o una conexión de red.

La llamada 'bind' por otra parte, duplica una parte del espacio de nombres en otro punto del mismo espacio. Y por último, la llamada 'unmount' permite retirar componentes.

Con el uso de 'bind' o 'mount', pueden apilarse múltiples directorios en un solo

punto del espacio de nombres [7]. En la terminología de Plan 9, esto es una unión de directorios y se comporta como la concatenación de los directorios que la constituyen. Un argumento de 'bind' y de 'mount' especifica la posición del nuevo directorio en la unión, permitiendo a los nuevos elementos agregarse al inicio de esta o bien reemplazar la jerarquía completa. Cuando se realiza una búsqueda de un fichero en una unión de directorios, cada componente de dicho conjunto se busca por turno, y se toma el primero que coincida. Del mismo modo, cuando se lee una unión de directorios, los contenidos de cada uno de estos son leídos por turnos.

La unión de directorios es una de las características de organización más utilizadas de los espacios de nombres de Plan 9. Por ejemplo, el directorio '/bin' se crea como unión de '/$cputype/bin' (binarios de programas), '/rc/bin' (scripts de la shell), y quizá más directorios proporcionados por el usuario. Esta construcción hace innecesaria la variable '$PATH', como ya se había comentado en el apartado 1.3.4.

Cuando un directorio es agregado a una unión, una opción en 'bind' o en 'mount' habilita el permiso de creación (una propiedad del espacio de nombres) en ese directorio. Por otro lado cuando un fichero está siendo creado con un nombre nuevo, se crea en el primer directorio de la unión con permiso de creación. Si esa creación falla, el 'create' falla. Este esquema permite el uso común de colocar un directorio privado en alguna parte de una unión de directorios públicos, sin embargo solo permite la creación en el directorio privado.

2.2.6. 'Señales' en Plan 9: 'notes'

Cuando un proceso produce una condición excepcional, como la división por cero o escribir en una 'pipe' cerrada, una nota se publica para comunicar dicha excepción. También puede ser enviada por otro proceso a través del comando 'postnote'. En Unix, las notas se implementan como señales [8].

Las medidas adoptadas al recibir una nota dependen de dicha nota. Además, las acciones por defecto pueden ser anuladas. La función 'notify' registra una notificación que es llamada en el proceso cuando se recibe una nota.

Una notificación debe terminar ya sea saliendo del programa o siendo llamada

por la función 'noted'. El argumento que se pasa a 'noted' define la acción a tomar: 'NDFLT' ejecuta la acción predeterminada; 'NCONT' reanuda el proceso en el punto en que fue notificado. Si la nota interrumpe una llamada al sistema, devuelve un error (error con la cadena interrumpida) después de que se reanude el proceso.

Las notas más comunes son las siguientes:

- Interrupt: el usuario ha interrumpido el proceso (tecla Del).
- Hangup: se ha cerrado la conexión de E/S.
- Alarm: ha saltado el temporizador puesto previamente.
- Sys: bad address: se está usando una dirección de memoria inválida y el kernel 'mata' al programa.

En general, las notas son peligrosas y es preferible evitar su uso ya que pueden interrumpir ciertas llamadas al sistema, como las bloqueantes: 'read' o 'write'. Además, al usarlas se rompe con la idea de un único flujo de control secuencial.

2.3. Diseño e implementación de un sistema de colas

En esta sección se va a explicar el funcionamiento del sistema de colas elegido para el proyecto. Como se vio en el apartado 1.3, un sistema de colas consiste en un conjunto de políticas y mecanismos incorporados al S.O para determinar el orden en que se ejecutan las tareas [9] [10].

El modo de funcionamiento elegido del sistema de colas está basado en el funcionamiento de los centros de cálculo o investigación, en el que el programa encargado de repartir las tareas recibe uno o varios ficheros de entrada con las tareas y con cualquier otro dato necesario para realizarlas, devolviendo uno o varios ficheros de salida con los resultados obtenidos y ciertos datos estadísticos.

Se ha decidido realizar el sistema de colas porque es una buena forma de trabajar con los espacios de nombres del S.O Plan9 ya que son una parte interesante de este sistema distribuido. Dejando al margen la parte académica, el objetivo del sistema

de colas es mejorar el rendimiento del sistema distribuido, es decir, que las tareas se ejecuten de manera más rápida que lo que lo harían sin él.

2.3.1. Funcionamiento

En éste proyecto los ficheros enviados al distribuidor de tareas serán dos, uno con las tareas a realizar, que tienen que ser órdenes válidas de la shell de Plan9 o algún programa compatible a ejecutar. Y otro fichero con los nodos disponibles del sistema distribuido, que vendrán indicados con su dirección ip correspondiente. Estos nodos serán los utilizados para ejecutar las tareas.

El fichero de nodos será proporcionado por un pequeño script, que ejecutará el comando 'ip/ping' en todas las direcciones ip que encuentre en el archivo de configuración 'local', visto en la figura 2.5. Este comando, equivalente al 'ping' de Unix, es una herramienta de diagnóstico en redes de computadores, que comprueba el estado de la conexión del host local con uno o varios equipos remotos de una red TCP/IP. Esto se realiza por medio del envío de paquetes de solicitud y de respuesta. En función de lo que devuelva dicho comando, se sabrá si el nodo está operativo o no. Con lo que el script devolverá un fichero con los nodos operativos.

Al finalizar la ejecución el distribuidor creará una serie de ficheros de salida. El número total de ficheros de salida dependerá del número de tareas.

Por una parte habrá un fichero por cada tarea realizada con estadísticas de dicha tarea como por ejemplo el tiempo de ejecución, que será proporcionado por el programa 'stress', utilizado para realizar las simulaciones, y que será explicado más adelante.

Además habrá otro fichero que contiene información de todas las tareas, tales como el PID del proceso, el nombre de la tarea, el nodo en el que ha sido ejecutada la tarea, el tiempo inicial y final, etc.

También existirá un fichero que mostrará la carga media y las interrupciones de cada nodo con datos recogidos cada dos segundos, y por último un fichero con el tiempo de ejecución total. Estos ficheros servirán para la elaboración de las gráficas mostradas en el siguiente capítulo.

2.3.2. Implementación del programa encargado de repartir tareas

Una vez explicada la forma de enviar ficheros al programa y cómo los devuelve éste, se va a explicar cómo se ha implementado el programa que se encarga de la repartición de las tareas y de la obtención de los resultados.

El programa se podría dividir en tres partes que realizan distintas funciones. La primera (preparación) sería la encargada de preparar todo lo necesario para la realización de las tareas y de las estadísticas. La segunda (envío y ejecución) sería la que se encarga de enviar y ejecutar tareas a los distintos nodos del sistema distribuido y de la obtención de los resultados una vez finalizadas dichas tareas. Y por último, la tercera parte (algoritmos) está compuesta por los diversos tipos de algoritmos utilizados para repartir las tareas en los nodos disponibles. En la imagen 2.8 se muestra un esquema de la implementación realizada.

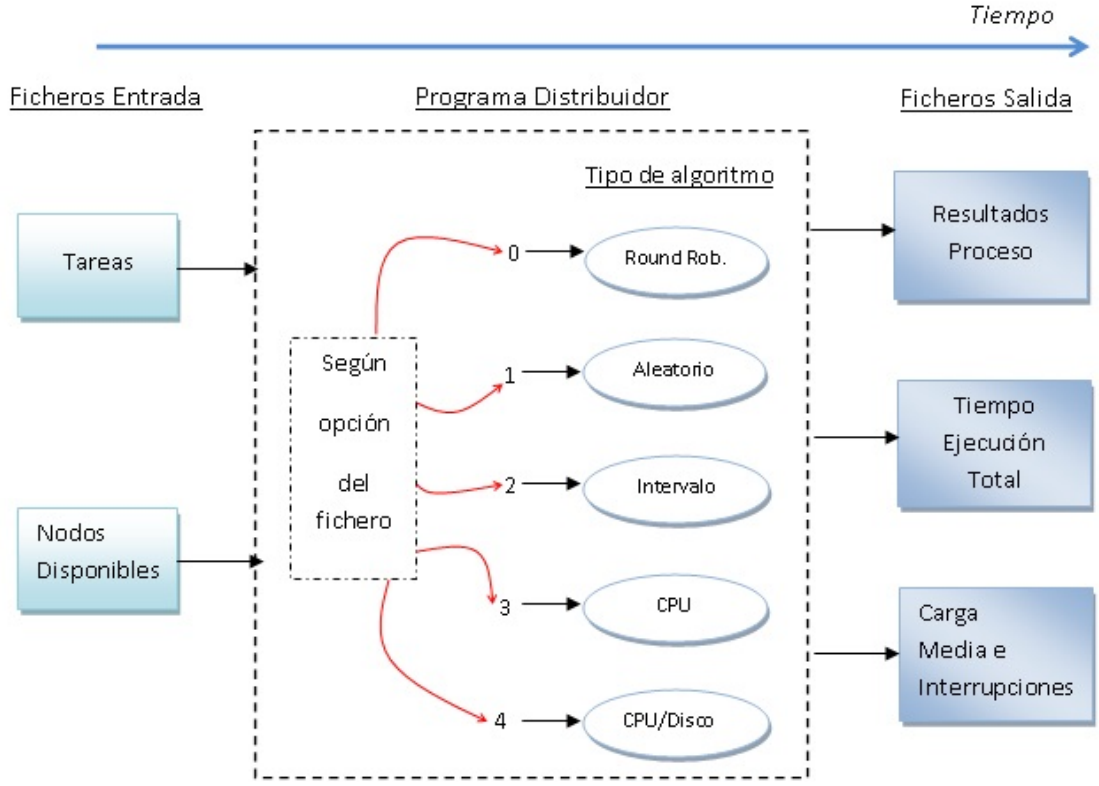

Figura 2.8: Diagrama implementación de los algoritmos.

2.3.3. Preparación

En esta parte de preparación se realizan una serie de tareas básicas para la posterior ejecución del programa. Para comenzar se leen los nodos disponibles del fichero que ha sido recibido y utilizando el comando 'import', explicado en el apartado 2.2, se importa el directorio '/dev' de todos los nodos leídos al directorio de trabajo.

Para ejecutar los comandos propios del Plan 9 se utiliza la llamada al sistema 'execl' que no retorna al programa original, por lo tanto hay que crear un hijo para importar cada directorio utilizando la llamada al sistema 'rfork', explicada en el apartado 2.2.

Una vez importados los directorios '/dev', se abren los ficheros '/dev/sysstat' y sus descriptores son metidos en un 'array' de descriptores. También hay que tener cuidado con dejar el fichero de entrada leído anteriormente, en el mismo punto en el que se encontró, ya que posteriormente se volverá a leer. Para ello se usa el comando 'fseek'.

Por último, se creará un proceso que monitoriza dos variables, la carga media y las interrupciones. Estas variables se encuentran en el fichero '/dev/sysstat', explicado en el apartado 2.2, que ha sido importado al directorio de trabajo anteriormente. Este programa recogerá esas variables cada dos segundos y las irá metiendo en un fichero hasta que termine el programa principal, que se cerrará utilizando una 'nota' (una señal en Unix).

2.3.4. Envío y ejecución de tareas

Otra parte importante dentro del programa es la del envío y ejecución de las tareas. Para ello se usa el comando 'cpu', explicado en el apartado 2.2, utilizando la llamada 'execl' que sirve para ejecutar programas en un nodo dentro del sistema distribuido.

En este proyecto como se verá en el apartado 3.1, las tareas serán trabajos de un programa, llamado 'stress'. Por eso hay que colocar dicho programa en el directorio unión '/bin' para que pueda ser utilizado con el comando 'cpu'.

Por último, se quiere que el resultado que proporciona 'stress' sea metido en un fichero, por lo que hay que hacer un 'bind' del fichero '/dev/cons', entrada y salida

estándar de la shell como se explicó en el apartado 2.2, con el fichero en el que se quiera tener la salida del 'stress'.

2.3.5. Algoritmos de planificación utilizados

Para finalizar falta la forma en la que las tareas serán repartidas por los nodos del sistema distribuido. Como se vio en el apartado 1.4.6 existen diversos tipos de algoritmos de planificación para la distribución de tareas. En este proyecto se han usado los siguientes tipos: Round Robin, Aleatorio, Intervalo, Umbral CPU y Umbral CPU/Disco. Estos algoritmos serán explicados a continuación.

Round Robin

Este algoritmo explicado en el apartado 1.4.6 es un caso particular de Round Robin con una Q infinita, para que la tarea no salga hasta que finalice. Funciona de la siguiente forma: teniendo dos listas, una de tareas y otra de nodos, envía en orden las tareas a los nodos ordenados hasta que se agotan los nodos disponibles. Cuando esto último ocurre vuelve a comenzar por el primer nodo.

Aleatorio

Algoritmo que elige el nodo al que enviar la tarea de forma aleatoria utilizando el comando 'nrand', el cual devuelve un número aleatorio entre 0 y el valor que se le pase.

Intervalo

El nodo elegido es el que menos carga media tenga en el momento en que se vaya a distribuir una tarea. El intervalo de distribución se ha fijado en 100 ms. En este algoritmo es necesario por tanto leer la variable de carga de cada nodo mientras se están ejecutando las tareas. Con este algoritmo se pretende distribuir las tareas de una forma más eficiente que con el algoritmo Round Robin ya que las tareas se van a enviar al nodo que tenga menos carga.

Umbral CPU

La tarea es enviada al primer nodo que se encuentre con una carga media menor que una constante proporcionada por el usuario. Con ello se pretende no saturar ningún nodo ya que cuando la carga media supere ese umbral el nodo no recibirá nuevas tareas hasta que su carga media baje por debajo del umbral. Para conseguir esto hay que monitorizar la variable de la carga media de cada nodo durante toda la ejecución de las tareas.

Umbral CPU/Disco

Este algoritmo tiene una diferencia respecto a los otros. Las tareas son diferenciadas entre tareas de CPU y de disco. Para ello en el fichero de entrada que recibe el distribuidor hay que añadirle a cada tarea el tipo al que pertenece (cpu o hdd).

Para las de CPU se mira, como en el algoritmo anterior, la carga media y la tarea es enviada al primer nodo con dicha carga menor que una constante. A su vez, para las tareas de disco se busca un nodo con unas interrupciones menores que otra constante distinta, también proporcionada por el usuario. De esta forma se intenta conseguir que los nodos además de estar equilibrados en cuanto a carga media lo estén también en cuanto al tipo de tarea que ejecutan. Ya que mientras en un nodo se ejecuta una tarea de disco también puede ir ejecutando otra de cpu sin sobrecargarse demasiado.

Capítulo 3

Evaluación

3.1. Descripción del entorno de pruebas

El sistema distribuido (Capítulo 3) utilizado para la realización de las pruebas durante este proyecto ha sido de tipo heterogéneo, para observar las diferencias en la distribución de la carga debido a la diferente capacidad de cómputo de los computadores. El sistema se ha compuesto por cuatro nodos encargados de realizar las tareas y por un computador que se utilizará como servidor de ficheros, de autentificación y de cpu, y se encargará de repartir las tareas entre los nodos.

Estos computadores están conectados en una subred de ethernet sencilla, es decir, todos los computadores se conectan mediante cables de ethernet a través de un switch.

Como se ha explicado en el apartado 2.1 en el servidor se ha instalado la cuarta edición del Plan 9, mientras que el resto de nodos arrancan a través de dicho servidor mediante PXE. Este método de arranque fue explicado en el apartado 2.2.

En la figura 3.1 se muestra un esquema del entorno de trabajo.

3.2. Descripción de las cargas

Para la realización de todas las pruebas necesarias durante la elaboración del proyecto se ha usado un programa para Unix llamado 'stress', el cual simula cargas de trabajo de diferentes tipos. Las simulaciones que genera el programa son ciclos

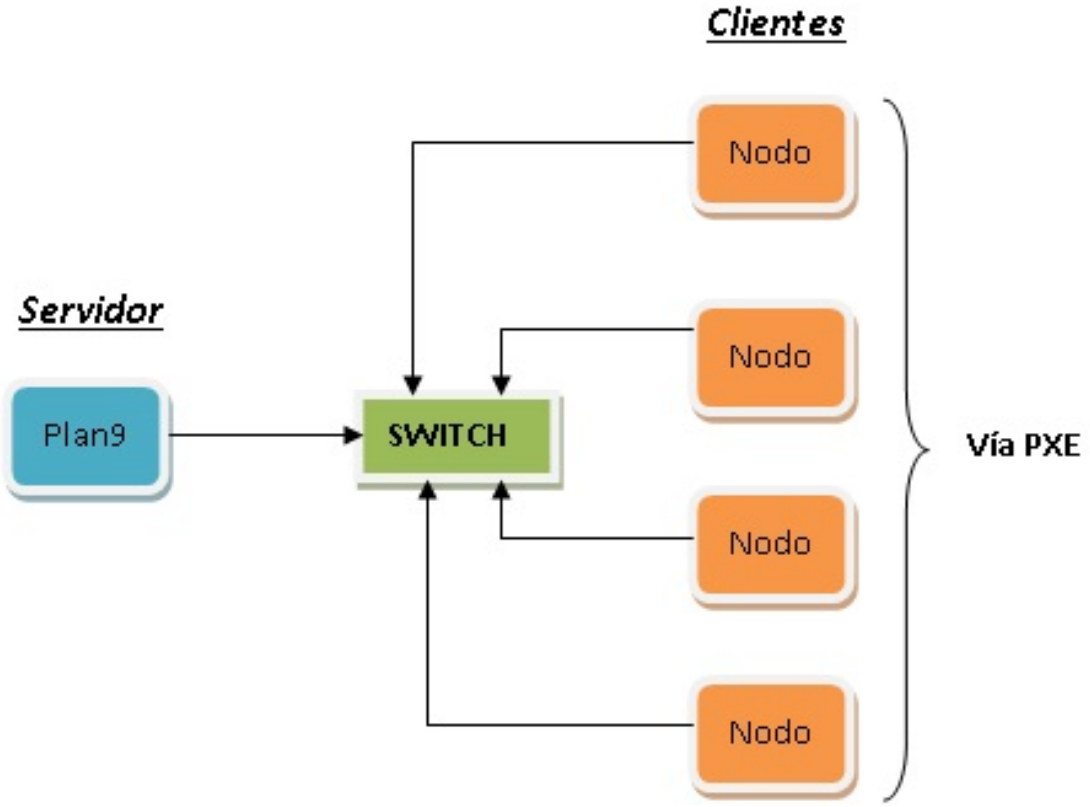

Figura 3.1: Diagrama entorno de trabajo.

de operaciones que necesitan de recursos de cpu, de entrada/salida, de disco... Para las pruebas en este proyecto se han utilizado cargas de cpu y de disco y, además, el programa 'stress' ha sido modificado para ser posible su uso en Plan 9 [11].

Las pruebas realizadas para cada uno de los diferentes algoritmos, que se mostrarán a continuación, han sido hechas con 35 tareas tanto de cpu como de disco. Dentro de esas tareas hay algunas con gran exigencia de cpu/disco mientras que hay otras más livianas. El número de tareas se ha escogido para que la ejecución de todas las tareas tarde un tiempo significativo pero sin ser muy grande debido a la cantidad de veces que van a ser ejecutadas (5 veces los algoritmos Round Robin, Aleatorio e Intervalo, 30 veces el algoritmo Umbral CPU debido a que se van a simular distintas constantes de cpu y 120 veces el algoritmo Umbral CPU/Disco por la misma razón que el anterior pero con dos constantes).

3.3. Descripción de las métricas de rendimiento

Para la clasificación de los diferentes tipos de algoritmos, en función de los resultados obtenidos, se ha utilizado el tiempo de conclusión de la última tarea a realizar. Esto se ha escogido de dicha manera porque se considera que un algoritmo es mejor que otro si ejecuta todas las tareas de forma más rápida.

Además ha sido necesario medir una serie de tiempos para posteriormente elaborar las tablas y gráficas que se mostrarán en el siguiente apartado. A continuación se describen los tiempos necesarios:

- Tiempo de inicio: es el tiempo en el que es creado el proceso con la tarea a ejecutar.
- Tiempo final: es el tiempo en el que finaliza el proceso creado, o lo que es lo mismo, cuando la tarea ha finalizado.
- Tiempo de ejecución: es el tiempo que tarda en ejecutarse una tarea desde que el planificador la envía a un nodo hasta que finaliza.
- Tiempo de ejecución total: es el tiempo que tardan en ejecutarse todas las tareas, es decir, la suma de los tiempos de ejecución de todas las tareas.
- Tiempo medio de espera: es la media del tiempo que tardan todas las tarea en ser atendidas desde que están listas hasta que son ejecutadas.

3.4. Resultados

Como se ha comentado anteriormente se ha utilizado el tiempo de ejecución total para clasificar a los diferentes algoritmos. En la gráfica 3.2 se van a mostrar todos los algoritmos utilizados en el eje de abscisas (alguno de ellos con variaciones en las constantes necesarias) y el tiempo que han tardado en ejecutarse las tareas de prueba en el eje de ordenadas.

Los tiempos medios de los diferentes algoritmos han sido calculados realizando la media de tiempos de 5 ejecuciones de las tareas a realizar.

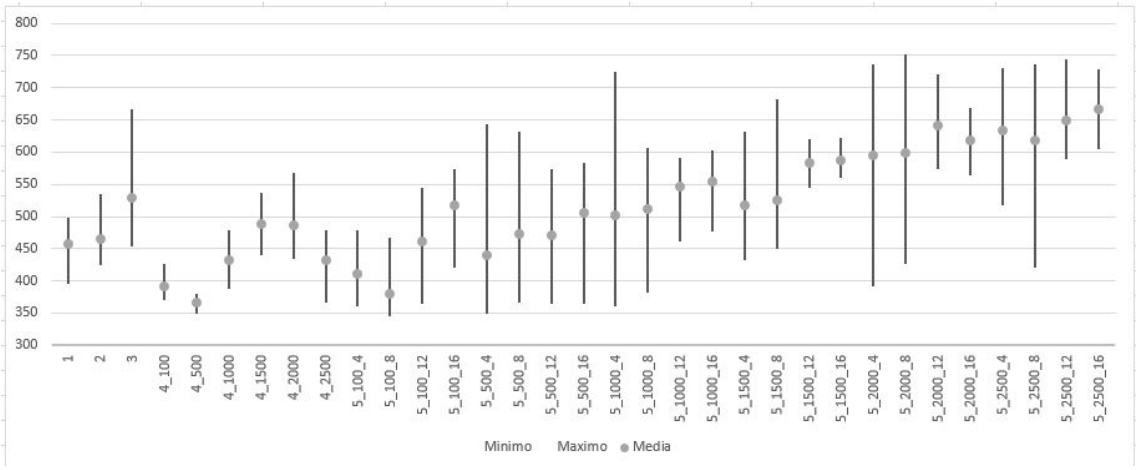

Figura 3.2: Comparativa entre algoritmos.

Como se puede apreciar los algoritmos de ejecución más rápidos son los del algoritmo Umbral CPU con constantes 100 y 500 y además se puede observar que la variación existente entre el mínimo y el máximo de ambos es muy pequeña, aspecto importante en la ejecución de tareas.

Por contra en los distintos tipos del algoritmo Umbral CPU/Disco se observa que tanto el tiempo de ejecución como la variación son más elevados. En un principio se pensaba que este iba a ser el mejor algoritmo al tener en cuenta tanto la carga de cpu como las interrupciones, pero vistos los resultados se puede concluir en que, en este caso, no compensa el leer ese dato de más a la hora de repartir las tareas. Aunque también puede ser debido a que las tareas de disco suponen un pequeño aumento en la carga media del nodo y no solamente en las interrupciones.

Además en los algoritmos con una constante de disco de 4 o de 8 se observa un mayor incremento en la variación. Esto es debido a que el planificador no envía tareas de disco al nodo hasta que está sin ninguna otra tarea de disco. Por ello se puede retrasar bastante el tiempo total de ejecución.

Los algoritmos Round Robin, Aleatorio e Intervalo no tienen un mal tiempo de ejecución pero no llegan al nivel de algunos tipos del algoritmo Umbral CPU. El Round Robin, como se explicó en el apartado 2.3.5, consiste en ir repartiendo las tareas cíclicamente en los nodos. Por ello tiene gran dependencia de si las tareas más pesadas recaen en los nodos con menos poder de computación. De no ser así es un algoritmo rápido. El problema está en que con las tareas seleccionadas para realizar

las pruebas (35 tareas de diferente tipo) no se acaba de apreciar correctamente. Con un mayor número de tareas y con más ejecuciones aleatorias de las mismas se podría llegar a apreciar mejor. El algoritmo Intervalo tiene grandes variaciones en el tiempo de ejecución. El problema en este algoritmo es que se van enviando tareas cada segundo al nodo que menor carga tenga y aunque con ello se consigue ir equilibrando la carga de los nodos va a llegar un momento en el que se van a acabar sobrecargando y el tiempo de ejecución total va a aumentar.

Una vez analizados los resultados de los diferentes algoritmos se van a mostrar tanto la carga media como el porcentaje de interrupciones de algunos casos para apreciar mejor el funcionamiento de dichos algoritmos.

Cada una de las gráficas que aparecen a continuacion consta de dos tipos de gráficas. La que aparece en la parte superior representa la carga media de la cpu durante el tiempo de ejecución y en la de parte inferior se plasma el porcentaje de interrupciones durante ese mismo tiempo. En ambas gráficas se muestran, con diferente color, los cuatro nodos encargados de realizar las tareas. En la gráfica de la carga media el eje de ordenadas indica la carga media (concepto explicado en el apartado 2.2.3) de la cpu, mientras que en la gráfica de las interrupciones el eje de ordenadas indica el porcentaje de interrupciones. Los ejes de abscisas de las dos gráficas representan el tiempo.

En la figura 3.3 se muestra el planificador Round Robin, en el que se puede apreciar como la carga de los nodos va aumentando escalonadamente al ir entrando nuevas tareas. Además se observa como cuando un nodo tiene gran porcentaje de interrupciones está más descargado de cpu o a la inversa. Esta es una simulación de las cinco pruebas realizadas para este planificador. Si se observan los resultados de todas ellas se puede ver la diferencia en las cargas de los nodos en función de si a alguno de los nodos le entran varias tareas de las más exigentes.

En el algoritmo de la figura 3.4 existen grandes variaciones (como es lógico debido a la aleatoriedad) entre unas ejecuciones y otras con las mismas tareas a realizar. En este caso en concreto se ve como el nodo 1 (verde) y el 3 (rosa) han recibido varias tareas de disco de gran exigencia ya que los demás nodos han finalizado de ejecutarse

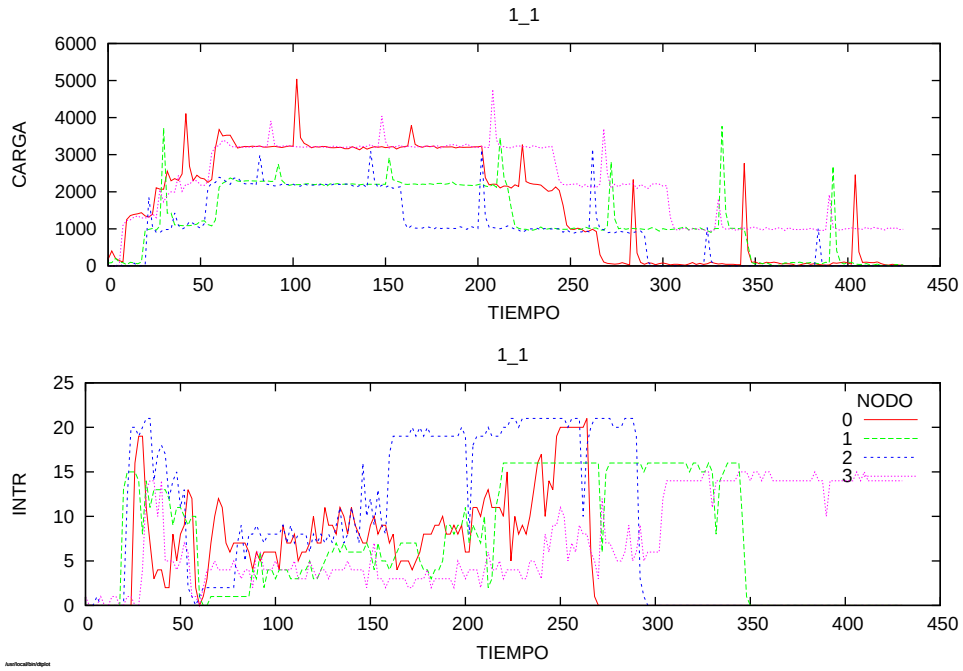

Figura 3.3: Round Robin.

todas sus tareas, mientras que el nodo 1 continúa ejecutando tareas aproximadamente 200s más (desde t=200 a t=400) y el nodo 3 continúa 400s más (desde t=200 a t=600).

Con el planificador de la figura 3.5 se pretende ir equilibrando los nodos enviando tareas al que más descargado se encuentre. Pero como se puede observar el nodo 0 (rojo) continúa ejecutando tareas unos 200s más después de que algunos nodos ya hayan finalizado sus tareas. Sin embargo el nodo 2 (azul) finaliza sus tareas a los 200s como se puede observar en la gráfica de la carga media.

En los algoritmos de tipo Umbral CPU (en esta gráfica con constante 500, figura 3.6) se consigue no elevar demasiado la carga de los nodos, dejando que la carga baje hasta una constante fijada y sólo en ese momento es cuando se le envía una nueva tarea a ese nodo. Se han realizado simulaciones con 6 constantes diferentes (desde 100 hasta 2500) y la mejor simulación se ha obtenido con 500 (la segunda

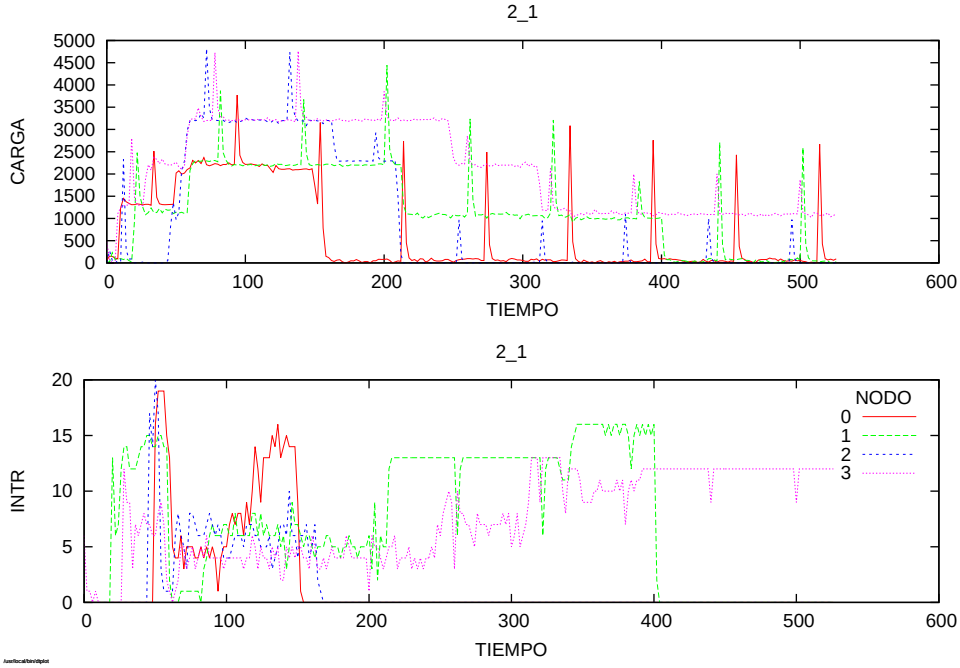

Figura 3.4: Aleatorio.

mejor con 100). Esto es debido a que de esta forma se consigue no sobrecargar los nodos, es decir, se evita meter al mismo nodo tareas de gran exigencia y gracias a esto se consigue que no haya ningún nodo que esté ejecutando tareas durante mucho tiempo mientras otros nodos estén descargados.

En el algoritmo Umbral CPU/Disco el proceso que mejores resultados ha obtenido es el de constante de cpu de 100 y constante de interrupciones de 8. El tiempo medio de este algoritmo es el segundo mejor de todos los algoritmos probados, tan sólo por detrás del Umbral CPU de constante 500. Aunque la variación entre el menor y el mejor tiempo es algo mayor que los algoritmos de constante de CPU de 100 y 500.

Como se puede observar en la figura 3.7 los nodos 0, 1 y 3 terminan casi al mismo tiempo, mientras que el nodo 2 (azul) finaliza con algo de antelación. Comparandolo con el algoritmo de CPU del apartado anterior se aprecia la diferencia ya que en ese caso las últimas tareas prácticamente finalizaron al mismo tiempo.

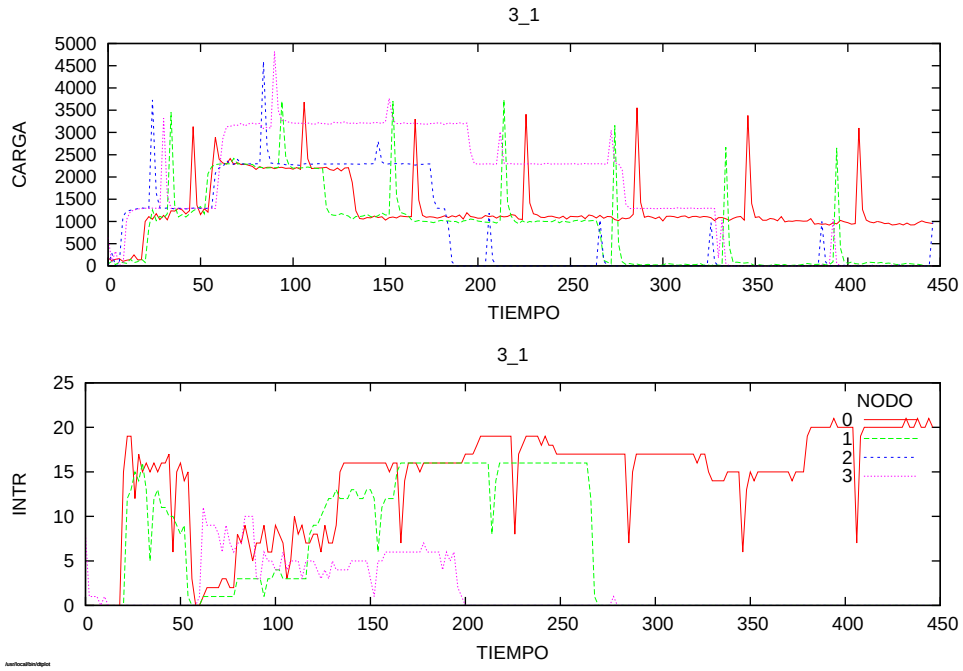

Figura 3.5: Intervalo.

Observando todas las gráficas se puede concluir que el algoritmo Umbral CPU de constante 500 es el mejor de todos y que además tiene poca variación entre el mejor y el peor caso.

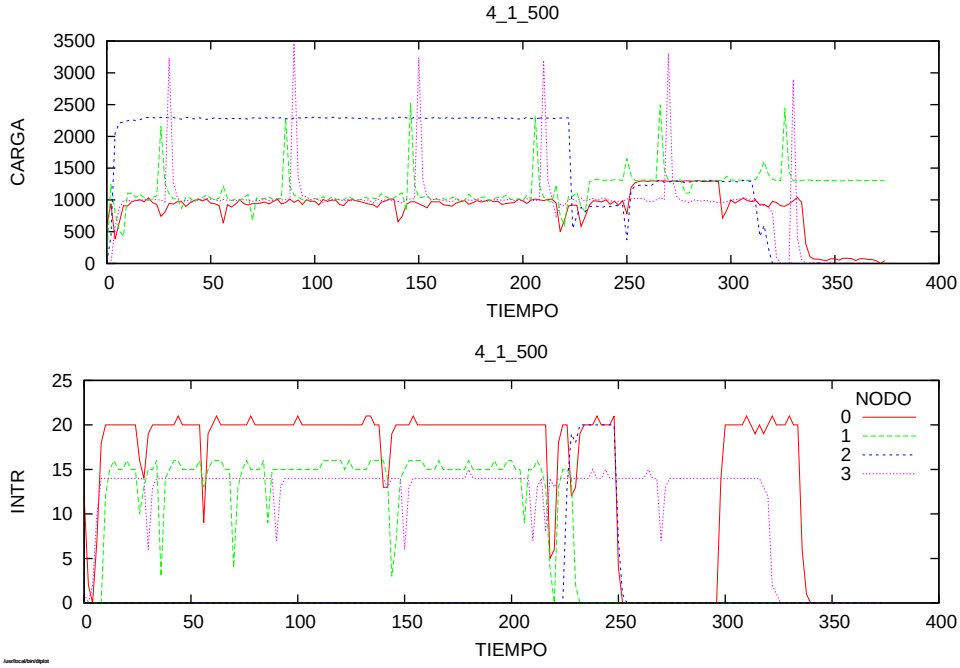

Figura 3.6: Umbral CPU con constante 500.

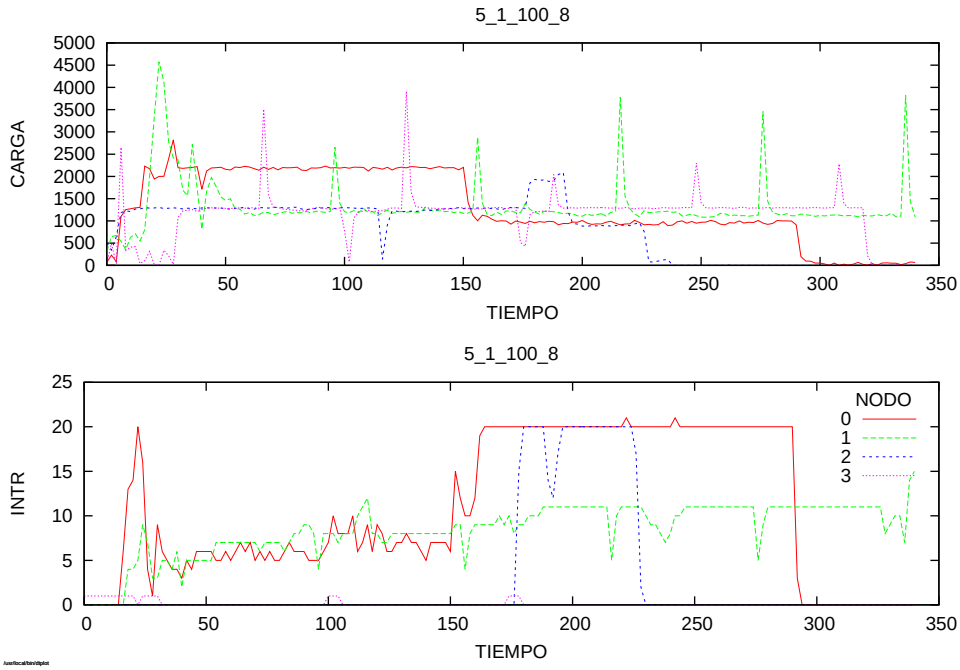

Figura 3.7: Umbral CPU/Disco con constantes 100 y 8.

Capítulo 4

Conclusión y líneas futuras

4.1. Conclusiones

En este capítulo se comentarán las distintas conclusiones que se han podido extraer de la realización del proyecto, tanto a nivel personal como técnico.

A nivel personal se ha aprendido el funcionamiento de un nuevo Sistema Operativo Distribuido el cual no se había utilizado en la titulación. Para ello se ha tenido que utilizar un sistema distribuido de cinco computadores, por lo que ha sido necesario el aprendizaje de estos sistemas. Al trabajar con el Sistema Operativo Plan9 ha sido necesario aprender la mayoría de comandos que aunque similares a otros sistemas Unix, tienen sus pequeñas diferencias. Además, a la hora de realizar el programa en C encargado de distribuir las tareas en el sistema distribuido, se ha tenido que tener en cuenta las diferencias en las llamadas al sistema utilizadas, así como las diferencias en algunas funciones de algunas librerías.

Todo ello ha aportado más soltura en cuestiones relativas a la programación y desempeño en distintos Sistemas Operativos.

Por otro lado, desde un punto de vista técnico es importante destacar que con este proyecto se han cumplido con todos los objetivos propuestos en el prólogo de esta memoria como son los siguientes: comprender el funcionamiento de un sistema operativo distribuido para lo que ha sido necesario una labor previa de investigación y documentación, la instalación del sistema y la preparación del cluster. También se ha aprendido a utilizar las herramientras proporcionadas por el sistema, en concreto

las necesarias para la realización del programa encargado de distribuir tareas.

Además con la finalización de este proyecto se cree haber alcanzado un dominio suficiente en el manejo de un Sistema Operativo Distribuido, lo que puede permitir en un futuro inmediato abordar otros proyectos similares.

Finalmente se presentan una serie de conclusiones generales extraídas de los resultados obtenidos en la experimentación. Estas conclusiones se obtienen de los experimentos basándose en cómo varía el funcionamiento del algoritmo en función del tiempo de ejecución total. Por ello se puede concluir que el mejor algoritmo ha sido el Umbral CPU y siendo más preciso con constante definida por el usuario de 500. También se puede concluir que este algoritmo es el que menor variación entre mejor y peor caso tiene. Esto se ve claramente al compararlo con el algoritmo Umbral CPU/Disco.

4.2. Líneas futuras

Una vez sacadas las conclusiones de este proyecto surgen otras líneas de estudio. Antes de enumerar algunos ejemplos que se pueden realizar es necesario recordar que el principal objetivo de este proyecto era el de comprender el funcionamiento de un sistema operativo distribuido en concreto el del Plan9 y que este sistema es usado de manera experimental. Por lo que el atractivo de realizar algo para este sistema es básicamente educacional.

Teniendo esto en cuenta por ejemplo se puede seguir completando el estudio de los distintos algoritmos, añadiendo modificaciones en algunos de los ya probados o añadiendo nuevos algoritmos que tengan en cuenta nuevos parámetros a la hora de distribuir las tareas.

También se podría aumentar el número de nodos a los que enviar tareas para ver si cambia el rendimiento de los algoritmos probados. Por otra parte sería posible utilizar más de un ordenador como distribuidor de tareas.

Otra opción sería utilizar un sistema operativo distribuido diferente y comparar los resultados entre ambos.

Bibliografía

[1] M. S. Ajay D.Kshemkalyani, *Distributed Computing.*, 2011.

[2] F. J. Ballesteros, *Introduction to Operating Systems Abstractions Using Plan 9 from Bell Labs.*, 2006.

[3] ——, *Notes on the Plan 9 3rd edition Kernel Source.*, 2007.

[4] F. C.Xu, *Load balancing in parallel computers: Theory and Practice.*, 1997, kluwer Academic Publishers Boston.

[5] P. bell labs, "Plan 9 - the documents (volume 2)." http://plan9.bell-labs.com/sys/doc/, 2012.

[6] cat v.org, "Plan 9 manual sections." http://man.cat-v.org/plan_9/, 2012.

[7] M. Hill, *Plan 9 from Bell Labs Programmer's Manual.*, 1993, vol. First Edition, computing Science Research Center AT&T Bell Laboratories.

[8] A. C. Martín, *Procesado de eventos de ratón para el sistema operativo Plan 9.*, 2004, proyecto fin de carrera Universidad Carlos III Madrid.

[9] A. A. Mirtchosvki, *Grid Computing with Plan 9 - an Alternative Solution for Grid Computing.*, 2005, tesis Doctoral University of Calgary.

[10] 9grid.fr, "The plan 9 cpu/auth server installation and basic setup howto." http://mirror.9grid.fr/mirror.9grid.fr/plan9-cpu-auth-server-howto.html, 2012.

[11] R. Pike, "How to use the plan 9 c compiler." *plan9.bell-lasbs.com*, 2012.