Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE COMPUTACIÓN

Título:
"Implantación de un servidor Linux sobre Mach".

Tesis para obtener el grado de Maestro en Ciencias,
en la Especialidad de Ingeniería Eléctrica

presenta:

Héctor Diez Rodríguez*

Director de tesis: Dr. Héctor Ruiz Barradas†

México, D.F. a 24 de febrero de 1998.

---

* Profesor Asistente del Departamento de Informática. Facultad de Ingeniería Eléctrica de la Universidad de Oriente. Santiago de Cuba. Cuba. Becario del Programa MUTIS de Intercambio Académico Iberoamericano.
† Profesor de la sección de computación y Jefe del área de Sistemas Digitales del Departamento de Ingeniería Electrónica. UAM Azcapotzalco.

DEDICATORIA

- A la memoria de mi padre.

- A mi madre, en su sesenta cumpleaños.

- A mi esposa Isabel Cristina,

al valor con que enfrentó una grave preeclamsia
que condujo a una inevitable interrupción de un embarazo deseado,
al empeño y voluntad para recuperarse de un traumático accidente,
a ella y con ella a nuestro presente, a la esperanza de *un sol y una luna*
y a nuestro futuro.

AGRADECIMIENTOS

Quiero agradecer a muchas personas, familiares, amigos e instituciones por el apoyo y confianza que me brindaron en todo momento, muy especialmente a los siguientes:

al Dr. Héctor Ruiz Barradas por su paciencia, asesoría y consejos en los momentos más oportunos. En su nombre a todos los profesores y personal de la sección de computación,

a Doña Carmen y Pilar por tanto amor,

a Nélida, Rafael, Lalo, Jesús, Sais, Heberto, en fin a todos mis compañeros de maestría por su ayuda y apoyo,

a la Secretaria de Educación Pública (SEP) del gobierno de México por haberme brindado el apoyo económico dentro del Programa Mutis de Colaboración Iberoamericana, y en especial a la Lic. Alicia Jeannetti Dávila por su fina atención y dedicación que me dispensó desde el primer momento,

a mis compañeros del Departamento de Informática de la Facultad de Ingeniería Eléctrica de la Universidad de Oriente y

a los que me ayudaron.

A todos, gracias.

Me disculpo si he omitido u olvidado a alguna persona.

Sinceramente,

Héctor Diez Rodríguez.

INTRODUCCIÓN

El proyecto de tesis: "Implantación de un servidor Linux sobre Mach", se enmarca dentro del área de Sistemas Distribuidos. Un sistema distribuido es una colección de computadoras independientes que proporcionan un acceso transparente a los recursos. Los usuarios de un sistema distribuido, además de tener los servicios que proporcionaría un sistema centralizado equivalente, cuentan con dos características importantes: el paralelismo y la tolerancia a fallas. Sin embargo, los sistemas distribuidos tienen ciertas desventajas, como es el hecho de tener una programación (*software*) más compleja, potenciales cuellos de botella en la comunicación y una débil seguridad en el acceso a la información. No obstante, existe un considerable interés mundial por su construcción e instalación, debido a las ventajas de costo y beneficio de estos sistemas sobre los sistemas tradicionales.

Un componente esencial de un sistema distribuido es el Sistema Operativo Distribuido (SOD), éste es un conjunto de programas que permiten administrar los recursos del sistema distribuido, dar transparencia al acceso de los recursos y proporcionar una interfaz entre el sistema y los usuarios. En los últimos años, el diseño de sistemas operativos distribuidos se ha movido hacia la variante de los micronúcleos, ya que ofrecen muchas ventajas en comparación con los sistemas monolíticos, entre las que se encuentran la portabilidad y la modularidad.

En la actualidad se han diseñado varios sistemas operativos distribuidos: Locus, Chorus, Mach, Amoeba, entre otros; sin embargo sólo dos de ellos han logrado transitar con éxito de los laboratorios de investigación al mundo comercial, éstos son Mach y Chorus.

Chorus fue diseñado en el instituto francés de investigaciones INRIA en 1980 y es comercializado actualmente por la empresa *CHORUS System*. Mach fue diseñado por investigadores de *Carnegie Mellon University* (CMU). Hay que resaltar que CMU decide hacer público el micronúcleo de Mach mientras que Chorus System comercializa y mantiene los derechos sobre Chorus, brindando sólo la posibilidad de explotar el sistema. Dado que

nuestro interés está en estudio del diseño e implantación de sistemas operativos distribuidos, hemos decidido trabajar con versiones públicas.

Algo en común de los sistemas Chorus y Mach es su diseño a través de un micronúcleo, que es la versión moderna de la estructura de los sistemas operativos. El diseño con micronúcleos conlleva a una migración de funciones fuera del núcleo, y esas funciones se implantan como servicios.

El desarrollo de servidores para el micronúcleo Mach ha sido una actividad importante tanto desde el punto de vista comercial como académico. Las propuestas comerciales sólo distribuyen los archivos binarios de tales servidores quedando excluida la posibilidad de realizar cualquier modificación. En el ámbito académico, como parte de los estudios que se realizan dentro del área de sistemas distribuidos, implantar servidores para diferentes sistemas operativos ha sido uno de los temas más atractivos que se están desarrollando y que han motivado investigar sobre los mismos.

En este contexto cobran gran interés los diferentes sistemas operativos que proporcionan funcionalidad similares a las de UNIX, dada la necesidad de contar con una interfaz conocida y de un sistema que permita la ejecución de aplicaciones ya existentes. Dentro de los sistemas de dominio público, Linux es, por el momento, el sistema con mayor popularidad. Linux es un sistema operativo de 32 bits que cumple con el estándar POSIX, desarrollado y mantenido mediante la colaboración de miles de usuarios en la Internet.

Debido a la importancia que desde el punto de vista de investigación y desarrollo ha despertado Mach, y a la popularidad de Linux que lo está convirtiendo en un estándar de facto, se ha impulsado la motivación de adecuar e implantar un servidor Linux para plataformas Intel sobre el micronúcleo de Mach, donde se permitan realizar estudios y una evaluación al micronúcleo y al servidor, así como realizar programas de aplicación.

Como disyuntiva para enfrentar el proyecto se analizaron dos propuestas tomando como base el micronúcleo Mach:

1. Programar un servidor, en su totalidad o
2. Poner en práctica un servidor ya existente.

Para la decisión, se estudiaron algunas variantes de servidores, entre ellos Lites BSD y Hurd, de los cuales se brindará información más adelante y que fueron relegados debido a la indisponibilidad que esos sistemas presentaron en el momento de selección, y porque tampoco proporcionaban la funcionalidad Linux que se deseaba.

Ante tal problemática y dada las siguientes razones:

1. el considerable tiempo de desarrollo que podría conllevar la realización del servidor, para obtener un sistema de igual funcionalidad a algunos ya existentes.
2. el trabajo no despreciable y enriquecedor que representa implantar y estudiar un servidor ya existente.
3. la disponibilidad de obtención de información y del sistema Mklinux, el cual se puede obtener gratuitamente a través de la red Internet.

decidimos poner en práctica un servidor ya existente, denominado Mklinux y desarrollado por el consorcio OSF (*Open Software Fundation*), el cual es un servidor Linux sobre el micronúcleo Mach 3.0

Mklinux (para plataforma Intel) en la actualidad ha reportado dificultades con su instalación, con la utilización de llamadas al sistema del micronúcleo Mach 3.0 de las cuales hay poca y dispersa documentación y con la compilación de programas en el sistema, lo cual brinda un marco de investigación que originaron el trabajo de tesis, cuyo objetivos son:

- instalación del sistema y documentación de la metodología de implantación.

- utilización de las llamadas al sistema del micronúcleo Mach 3.0.
- programación de aplicaciones y compilación de programas en el sistema Mklinux.

Este trabajo está presentado en cuatro capítulos y tres apéndices:

□ capítulo uno: Introducción a los sistemas distribuidos.

Hace una descripción de los sistemas distribuidos, dando una caracterización de los mismos, abordando sus ventajas y desventajas. Se describen aspectos de su diseño y se analizan los aspectos básicos de los sistemas operativos distribuidos, además se ejemplifican tres de ellos basados en micronúcleo.

□ capítulo dos: Mach.

Es una introducción al sistema operativo Mach, comenzando con su historia y describiendo las principales abstracciones del micronúcleo. Además se exponen los factores que favorecieron su paso de un proyecto de investigación universitaria a un producto comercial, así también se exponen algunas soluciones actuales de diseño e implantación de servidores sobre este micronúcleo.

□ capítulo tres: Linux.

Describe las características generales del sistema operativo Linux, historia y características. Además hace un análisis del sistema Mklinux, un servidor Linux sobre el micronúcleo Mach 3.0, desarrollado por la OSF.

□ capítulo cuatro: Utilización de las llamadas al micronúcleo Mach 3.0.

Aborda la programación de aplicaciones que utilizan las abstracciones del micronúcleo Mach 3.0 en el sistema Mklinux, y se presenta siguiendo la idea de

exponer los conceptos y documentar las llamadas al micronúcleo empleadas. Se describen los detalles necesarios para la programación como son: archivos de cabecera, bibliotecas y convenciones para escribir los programas. Los ejemplos abordan el manejo de hilos, tareas y la comunicación interproceso.

□ apéndice A: Requisitos mínimos para la instalación de Linux.

□ apéndice B: Implantación del sistema Mklinux.

Detalla la metodología de implantación del sistema Mklinux, cómo iniciar una sesión de trabajo con el sistema y conceptos básicos del ambiente de trabajo de OSF

□ apéndice C: Códigos fuente de los programas presentados.

ÍNDICE

CAPITULO I:
INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS.

Incluso si estamos en el camino correcto, seremos arrollados si nos quedamos sentados ahí.
- Pal Koppman.

I.1 Introducción

Los sistemas de cómputo están sufriendo una revolución. Desde 1945, cuando comenzó la era de la computadora moderna, hasta cerca de 1985, las computadoras eran grandes y caras. Incluso las minicomputadoras costaban por lo general cientos de miles de dólares cada una. Como resultado, la mayor parte de las organizaciones tenían tan sólo unas cuantas computadoras y, por carecer de una forma para conectarlas, éstas operaban por lo general de manera independiente entre sí.

Sin embargo, a partir de la mitad de la década de los 80', dos avances tecnológicos comenzaron a cambiar esta situación. El primero fue el desarrollo de poderosos microprocesadores. Al principio, se disponían de máquinas de 8 bits, pero pronto se volvieron comunes las unidades de procesamiento de 16, 32 e incluso 64 bits. Muchos de ellos tenían el poder de cómputo de una computadora *mainframe* de tamaño grande, pero por una fracción de su precio.

El segundo desarrollo fue la invención de las redes. Las redes de área local **(LANs Local Area Networks)** permiten conectar docenas, e incluso cientos de máquinas, de tal forma que se puedan transferir información entre ellas. Las redes de área amplia **(WANs Wide Area Networks)** permiten conectar millones de máquinas en todo el planeta con velocidades que varían de 64 kbps (kilobits por segundo) a gigabits por segundos para ciertas redes experimentales avanzadas.

El resultado neto de estas tecnologías es que hoy en día no sólo es posible, sino fácil, reunir sistemas de cómputo compuestos por un gran número de procesadores, conectados mediante una red de alta velocidad. Estos sistemas reciben el nombre genérico de **sistemas distribuidos**, en contraste con los **sistemas centralizados** que constan de un CPU, su memoria, sus periféricos y algunas terminales.

Sin embargo los sistemas distribuidos necesitan una programación radicalmente distinta a la de los sistemas centralizados. En particular, los sistemas operativos necesarios para estos sistemas distribuidos están apenas en una etapa de surgimiento.

En este capítulo se hace una descripción de los sistemas distribuidos, dando una caracterización de los mismos, abordando sus ventajas y desventajas. Se describen aspectos de su diseño, se analizan los aspectos básicos de los sistemas operativos distribuidos y se ejemplifican tres de ellos basados en micronúcleo.

I.2 Caracterización de un sistema distribuido

Una tendencia en los sistemas de cómputo es distribuir los cálculos entre varios procesadores físicos. Básicamente hay dos esquemas para construir estos sistemas: **sistemas multiprocesadores** en los cuales los procesadores comparten memoria y reloj, y la comunicación generalmente se lleva a cabo a través de la memoria compartida y **sistemas multicomputadoras** en los cuales los procesadores no comparten memoria ni reloj; en cambio cada uno tiene su propia memoria local. La comunicación se lleva a cabo por medio de intercambio de mensajes.

Un sistema distribuido es un conjunto de procesadores interconectados por medio de una red de comunicaciones; a tales sistemas se les conoce también como sistemas débilmente acoplados. Desde el punto de vista de un procesador específico en un sistema distribuido, los demás procesadores y sus recursos son *remotos*, mientras que sus propios recursos son

locales. La razón principal para construir sistemas distribuidos es poder compartir los recursos.

Un sistema distribuido proporciona a los usuarios acceso a los distintos recursos que ofrece el sistema, sin ser necesario que los usuarios sepan de la multiplicidad de máquinas y puedan accesar a los recursos remotos de la misma manera que lo hacen con los recursos locales. La migración de datos y procesos de un nodo (máquina, computadora) a otro queda bajo el control del sistema operativo distribuido.

Los sistemas distribuidos son implantados en plataformas que varían en tamaño desde pocas estaciones de trabajo interconectados por una red local a redes mundiales, donde se enlazan miles de computadoras. Las aplicaciones de sistemas distribuidos van desde proporcionar facilidades de cómputo de propósito general para grupos de usuarios hasta automatizar trabajos en bancos, líneas aéreas y sistemas de comunicación entre otros.

Definición

Un sistema distribuido es una colección de computadoras independientes, enlazadas por una red, que aparecen ante los usuarios del sistema como una única computadora.

Esta definición tiene dos aspectos. El primero se refiere al *hardware*: las máquinas son autónomas. El segundo a el *software*: los usuarios piensan que el sistema es como una única computadora.

Un aspecto básico es que el uso de múltiples procesadores debe ser invisible (transparente) para los usuarios; los cuales deben ver el sistema como un "procesador virtual" y no como una colección de máquinas diferentes. Los usuarios de un sistema distribuido no deben conocer en cual máquina (o máquinas) los programas están ejecutándose o donde están almacenados los archivos.

Ventajas de los sistemas distribuidos

Con la tecnología del microprocesador, por unos cuantos cientos de dólares, es posible comprar un microcircuito de CPU que puede ejecutar más instrucciones por segundo de las que realizaban una de las más grandes *mainframes* de la década de los 80' Si se está dispuesto a pagar el doble, se obtiene el mismo CPU, sólo que con una velocidad un poco mayor. Como resultado, la solución más eficaz en cuanto a costo es limitarse a un gran número de CPU baratos reunidos en un mismo sistema. Así la razón principal de la tendencia hacia los sistemas distribuidos, es que estos sistemas tienen en potencia una proporción precio-desempeño mucho mejor que la de un sistema centralizado. Las ventajas de un sistema distribuido se resumen en la tabla 1.1.

Tabla 1.1 Ventajas de los sistemas distribuidos sobre los sistemas centralizados.

Elemento	Descripción
Economía	**Los microprocesadores ofrecen mejor proporción precio-rendimiento que los mainframes**
Velocidad	**Un sistema distribuido puede tener mayor poder de cómputo que un mainframe**
Distribución Inherente	**Algunas aplicaciones utilizan máquinas que están separadas a ciertas distancias**
Confiabilidad	**Si la máquina se descompone, el sistema puede sobrevivir como un todo**
Crecimiento por incrementos	**Se puede añadir poder de cómputo en pequeños incrementos**

Una razón para la construcción de un sistema distribuido es que ciertas aplicaciones son distribuidas en forma inherente. Otra ventaja potencial de un sistema distribuido sobre uno centralizado es una mayor confiabilidad. Al distribuir la carga de trabajo en varias máquinas, la falla de un circuito descompondrá a lo más a una máquina y el resto seguirá intacto. Para el caso de aplicaciones críticas, el uso de un sistema distribuido para lograr mayor confiabilidad puede ser el factor dominante.

Por último, el crecimiento por incrementos es decir su gran escalabilidad también es una ventaja potencial. En un sistema distribuido podrían añadirse componentes al sistema según sea la demanda, lo que permite un desarrollo gradual conforme surjan las necesidades.

Aún cuando los microprocesadores representan una forma económica de trabajo, no es posible ofrecer a cada persona su propia computadora personal y permitirle trabajar independientemente, la razón esencial es la necesidad de muchos usuarios de compartir ciertos datos.

Los datos no son los únicos elementos que se pueden compartir. Otros recursos son los periféricos caros, como son las impresoras láser de color y los dispositivos de almacenamiento masivo.

Otra razón para la conexión de un grupo de computadoras aisladas en un sistema distribuido es lograr una mejor comunicación entre las personas, por ejemplo el correo electrónico cada vez tiene mayores ventajas respecto al correo de cartas. La tabla 1.2 presenta un resumen de las ventajas de los sistemas distribuidos sobre las computadoras personales aisladas.

Tabla 1.2 Ventajas de los sistemas distribuidos sobre las computadoras personales aisladas.

Elemento	Descripción
Datos compartidos	**Permiten que varios usuarios tengan acceso a una base de datos común**
Dispositivos compartidos	**Permiten que varios usuarios compartan periféricos caros**
Comunicación	**Facilita la comunicación de persona a persona**
Flexibilidad	**Distribuye la carga de trabajo entre las máquinas disponibles en la forma más eficaz en cuanto a los costos**

Desventajas de los sistemas distribuidos

Aunque los sistemas distribuidos tienen sus aspectos fuertes, también tiene sus desventajas. La mayor desventaja es la programación, ¿qué tipo de sistema operativo, lenguajes de

programación y aplicaciones son adecuadas para estos sistemas?, ¿cuánto deben saber los usuarios de la distribución?, ¿qué tanto debe hacer el sistema y qué tanto deben hacer los usuarios?.

Las redes de comunicación introducen otros problemas, como la pérdida de mensajes, lo cual requiere programas especiales para su manejo y puede verse sobrecargada.

Por último, el hecho de que los datos sean fáciles de compartir es una ventaja, pero también un problema. Si las personas pueden tener acceso a los datos en todo el sistema, entonces también pueden tener acceso a los datos con los que no tienen nada que ver, la seguridad puede ser con frecuencia un problema, además otro aspecto a considerar es el problema de consistencia en los datos.

A pesar de estos problemas potenciales, las ventajas que se obtienen tienen un mayor peso que las desventajas. De hecho, es probable que en poco tiempo, gran parte de las organizaciones conecten la mayoría de sus computadoras a extensos sistemas distribuidos, para proporcionar un mejor servicio, más barato y conveniente a sus usuarios.

I.3 Aspectos de diseño de los sistemas distribuidos

Las características fundamentales de un sistema distribuido son: soporte para compartir recursos, concurrencia, escalabilidad, tolerancia a fallas y transparencia [13]. A continuación se brindarán detalles de los aspectos que están involucrados en el diseño de un sistema distribuido con tales características.

I.3.1 Transparencia

El concepto de transparencia se puede aplicar a varios aspectos de un sistema distribuido y está asociado al diseño del sistema de forma que los usuarios piensen que la colección de máquinas es tan sólo un sistema de tiempo compartido con un procesador.

Existen distintos tipos de transparencia en un sistema distribuido, entre ellas están:

- transparencia en la localización

  Se refiere al hecho de que los usuarios no requieren indicar o conocer la localización de los recursos del *hardware* y *software* como CPU, impresoras, archivos y bases de datos para su localización.

- transparencia en la migración

  Significa que los recursos pueden moverse de una posición a otra sin que los usuarios se percaten de ello.

- transparencia de réplica

  Los usuarios no deben estar conscientes del número de copias existentes.

- transparencia de concurrencia

  Varios usuarios pueden compartir recursos de manera automática. Si el sistema es transparente respecto a la concurrencia, los usuarios no notarán la existencia de otros usuarios.

I.3.2 Confiabilidad

Uno de los objetivos originales de la construcción de sistemas distribuidos fue el hacerlos más confiables que los sistemas con un procesador. La idea es que si una máquina falla, alguna otra máquina se encargue del trabajo.

Es importante distinguir dos aspectos de la confiabilidad: la fiabilidad y la disponibilidad. La fiabilidad garantiza que el sistema no corrompa o pierda los datos, la disponibilidad, se refiere a la fracción de tiempo en que se puede utilizar el sistema. La disponibilidad se puede mejorar mediante un diseño que no exija el funcionamiento simultáneo de un número sustancial de componentes críticos. Otra herramienta para mejorar la disponibilidad es la redundancia: se pueden duplicar piezas claves del *hardware* y del *software*, de modo que si una de ellas falla, las otras puedan sustituirla.

Otros aspectos de la confiabilidad es la seguridad y la tolerancia a fallas. Los archivos y otros recursos deben ser protegidos contra el uso no autorizado. En general, los sistemas distribuidos se deben diseñar de forma que escondan las fallas, es decir, ocultarlos de los usuarios.

I.3.3 Desempeño

El problema del desempeño se complica por el hecho de que la comunicación, factor esencial en un sistema distribuido y que es "generalmente lenta" respecto a la velocidad de procesamiento. Así, para optimizar el desempeño, con frecuencia hay que minimizar el número de mensajes. La dificultad con esta estrategia es que la mejor forma de mejorar el desempeño es tener muchas actividades en ejecución paralela en distintos procesadores, pero esto requiere el envío de muchos mensajes.

I.3.4 Escalabilidad

La mayor parte de los sistemas distribuidos están diseñados para trabajar con unos cuantos cientos de CPU. Es posible que los sistemas futuros tengan mayores órdenes de magnitud. La cuestión es si los métodos de programación que se desarrollan en la actualidad podrán escalarse hacia tales grandes sistemas, evitando los componentes, tablas y algoritmos centralizados.

I.3.5 Primitivas de comunicación

Las computadoras que forman un sistema distribuido normalmente no comparten la memoria principal y por tanto la comunicación vía memoria compartida no son aplicadas generalmente. En su lugar, el paso de mensaje es utilizado.

Paso de mensajes

Un modelo utilizado por los investigadores en esta área es el modelo cliente-servidor, en el cual un grupo de procesos en cooperación llamados servidores ofrecen servicios a los usuarios llamados clientes. Las máquinas de los clientes y servidores ejecutan por lo general el mismo núcleo. Una máquina puede ejecutar varios clientes, varios servidores o combinación de ambos. Un esquema del modelo se muestra en la figura 1.1, en el mismo el cliente que desea algún servicio envía un mensaje al servidor y espera por el mensaje de respuesta.

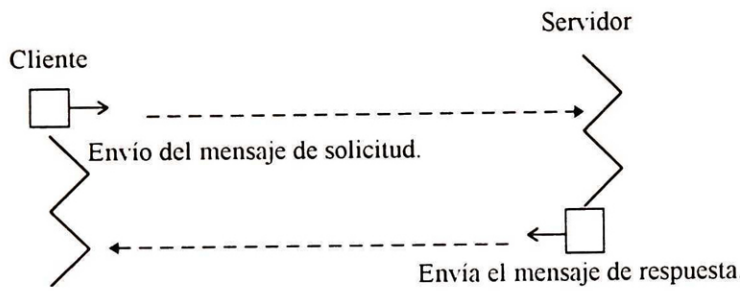


Figura 1.1 Modelo de comunicación cliente-servidor.

Para evitar un gasto excesivo en los protocolos orientados hacia la conexión [19] como OSI o TCP/IP, lo usual es que el modelo cliente-servidor se base en un protocolo solicitud/respuesta sencillo y sin conexión. El cliente envía un mensaje de solicitud al servidor para pedir cierto servicio (por ejemplo, la lectura de un archivo), el servidor hace el trabajo y regresa los datos solicitados o un código de error para indicar la razón por la cual una petición no se pudo llevar a cabo.

La mayoría de los sistemas proporcionan dos primitivas: SEND y RECEIVE, para el envío y recepción de los mensajes. Dos de las decisiones fundamentales están relacionadas con hacer las primitivas fiables versus no fiables y bloqueantes versus no bloqueantes. En un extremo, el SEND puede poner el mensaje en la red y desear tener suerte que éste llegue, sin garantizar que la entrega se efectúe y sin intentos de retransmisión automática por el

sistema. En el otro extremo, la primitiva SEND puede manipular los mensajes perdidos, retransmisiones y reconocimientos internamente y cuando su ejecución termine estará seguro que el mensaje ha sido recibido y reconocido.

I.3.6 Nombramiento

Los sistemas operativos soportan objetos como archivos, directorios, procesadores, servicios, nodos y dispositivos de entrada-salida. Cuando un proceso desea el acceso a uno de estos objetos, éste debe presentar algún tipo de nombre al sistema operativo para especificar a cuál de estos objetos se desea accesar. En algunos casos estos nombres son cadenas ASCII y en otros casos son identificadores.

En un sistema distribuido, un servidor de nombres es utilizado para mapear los nombres proporcionados por los usuarios en su localización. En la variante del modelo de un único servidor de nombres, para localizar un servicio se envía el nombre del servicio al servidor de nombres y éste responde en cuál nodo se encuentra dicho servicio.

I.4 Sistemas Operativos Distribuidos

Un sistema operativo es una parte importante de cualquier sistema de cómputo. Como su nombre lo indica, es un conjunto de programas el cual actúa como intermediario entre el usuario y la circuiteria de la computadora y llevan a cabo varias funciones como: comunicación con los periféricos, administración de memoria, supervisión, contabilidad y seguridad de recursos y la administración de programas y datos. Su propósito es proporcionar un entorno en el cual el usuario pueda ejecutar programas. El objetivo principal de un sistema operativo es lograr que el sistema de cómputo se use de manera cómoda y que la circuiteria (*hardware*) se emplee de manera eficiente.

Un componente esencial y una importante meta de investigación y desarrollo en el área de sistemas distribuidos ha sido el diseño de sistemas operativos distribuidos. Similar a un

sistema operativo convencional, éste es una colección de programas que simplifican la tarea de programación y soportan una amplia gama de aplicaciones. Sin embargo, éstos deben ser modulares y extensibles, permitiendo que nuevos componentes puedan ser adicionados a los sistemas operativos distribuidos en respuesta a las necesidades de las nuevas aplicaciones.

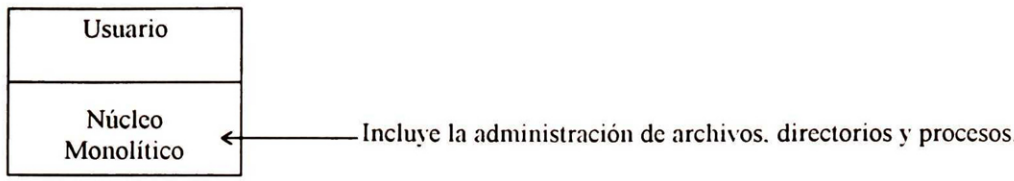
Los sistemas operativos distribuidos proporcionan una colección de mecanismos [3,4] sobre los cuales deben ser implantados las políticas de administración de los recursos. Esta infraestructura permite a los servidores encapsular y proteger los recursos, mientras permite a los clientes compartirlos concurrentemente. El sistema operativo distribuido proporciona los mecanismos necesarios: la resolución de nombres, comunicación y planificación, para que los clientes invoquen las operaciones sobre los recursos.

A continuación se hace una descripción de los aspectos más relevantes en el diseño de un sistema operativo distribuido: núcleo, protección de los recursos, comunicaciones y memoria virtual.

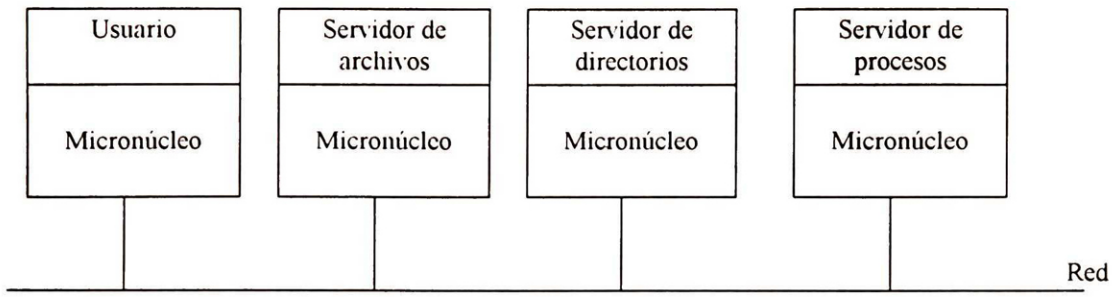
I.4.1 Núcleo

El núcleo es un programa que se distingue por el hecho que el código es ejecutado con los máximos privilegios para accesar a los recursos físicos en la computadora. En particular tiene el control del manejo de la unidad de memoria y de los registros del procesador.

Existen dos variantes para la arquitectura del núcleo: el núcleo monolítico y el micronúcleo. La primera plantea que cada máquina debe ejecutar un núcleo tradicional que proporcione la mayoría de los servicios. La segunda sostiene que el núcleo debe proporcionar lo menos posible y que el grueso de los servicios del sistema operativo se obtenga a partir de los servidores al nivel usuario. Estos modelos se ilustran en la figura 1.2.



a) Núcleo Monolítico.



b) Micronúcleo.

Fig 1.2 Variantes de la arquitectura del núcleo.

El núcleo monolítico está basado en el sistema operativo centralizado actual, aumentado con capacidades de red y la integración de servicios remotos. La mayoría de las llamadas al sistema se realizan mediante invocaciones al núcleo, en donde se realiza el trabajo, para que después el núcleo regrese el resultado deseado al proceso del usuario. Con este método la mayoría de las máquinas tienen discos y administra sus propios sistemas locales de archivos.

Muchos de los sistemas distribuidos que son extensiones de UNIX utilizan este método, puesto que el propio UNIX tiene un gran núcleo monolítico.

Todos los servicios del sistema operativo, no incluidos en el micronúcleo, se implantan por lo general como servidores a nivel usuario. Para buscar un nombre, leer un archivo u obtener algún otro servicio, el usuario envía un mensaje al servidor apropiado, el cual realiza el trabajo y regresa el resultado. La ventaja de este método es que es altamente modular: existe una interfaz bien definida con cada servicio y cada servicio es igual de accesible para todos los clientes, en forma independiente de la localización. Además, es fácil implantar, instalar y depurar nuevos servicios, puesto que la adición o modificación de un servicio no requiere el

alto total del sistema y el arranque de un nuevo núcleo, como es el caso de un núcleo monolítico.

Es precisamente esta capacidad de añadir, eliminar y modificar servicios lo que le da al micronúcleo su flexibilidad. Además, los usuarios que no estén satisfechos con alguno de los servicios oficiales son libres de escribir el suyo propio.

La única ventaja potencial del núcleo monolítico es el rendimiento porque las invocaciones al núcleo y la realización de todo el trabajo ahí puede ser más rápido que el envío de mensajes a los servidores remotos.

En los últimos años, el diseño de sistemas operativos distribuidos se ha movido hacía la variante de los micronúcleos, ya que ofrecen muchas ventajas en comparación con los sistemas monolíticos, entre las que se encuentran la portabilidad y la modularidad.

El diagrama de la arquitectura de un micronúcleo se muestra en la figura 1.3 y sus principales componentes son los siguientes:

- Administración de procesos.

  Manipula la creación y las operaciones al bajo nivel sobre los procesos. Un proceso consiste en un ambiente de ejecución con uno o más hilos.
- Administración de hilos.

  Un hilo es la abstracción de actividad de ejecución susceptibles a planificación. Esta unidad se encarga de la creación, sincronización y planificación de los hilos.
- Administración de comunicación.

  Se encarga de la comunicación entre hilos.
- Administración de la memoria.

  Manipula los accesos a la memoria física.
- Supervisor.

  Atiende las interrupciones y otras excepciones.

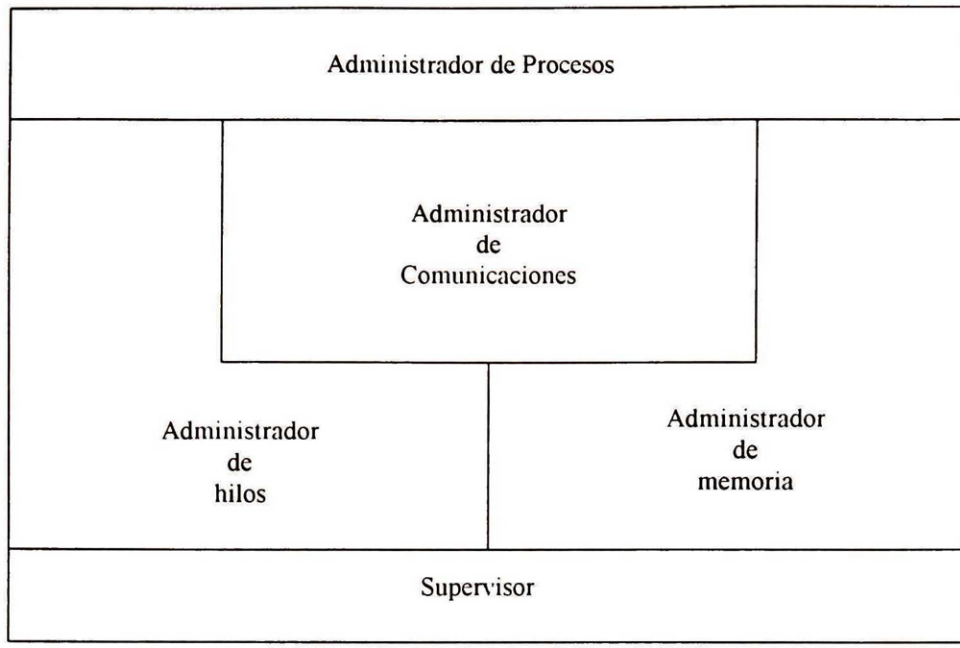


Figura 1.3 Arquitectura de un micronúcleo

La creación y administración de un ambiente de ejecución es normalmente caro, éste puede ser compartido por varios hilos. Los hilos pueden compartir todos los recursos y pueden ser creados y destruidos dinámicamente si es necesario.

La idea central de tener múltiples hilos en ejecución es maximizar el grado de concurrencia en la ejecución de operaciones poco relacionadas. Esto es particularmente útil en servidores, donde reduce la tendencia de convertirlos en cuellos de botella ante las solicitudes concurrentes de los usuarios.

El ambiente de ejecución proporciona protección a los datos y otros recursos contenidos en él, pero inaccesibles para los hilos residentes en otros ambientes de ejecución.

La tabla 1.3 muestra los nombres usados por algunos micronúcleos para las abstracciones de ambiente de ejecución e hilo.

Tabla 1.3 Abstracciones de ambiente de ejecución e hilo.

Núcleo del sistema operativo distribuido	Nombre del hilo	Nombre del ambiente de ejecución
Amoeba	hilo	proceso
Chorus	hilo	actor
Mach	hilo	tarea

I.4.2 Protección de los recursos

La idea del esquema de protección es asegurar que cada proceso pueda accesar sólo aquellas recursos para los cuales tiene permiso. El permiso especifica cuales operaciones se pueden realizar en el recurso.

El dominio de protección es una abstracción para proporcionar protección en ambientes compartidos. Dos alternativas de implantarlo comúnmente en sistemas operativos son las capacidades y las listas de control de acceso.

- Capacidades:

  Las capacidades han sido ampliamente utilizadas para la protección de recursos para autentificar los accesos en sistemas distribuidos. Una capacidad es una "*llave digital*", los accesos a un recurso son garantizados con sólo presentar la llave.

  Diferentes capacidades son utilizadas para diferentes combinaciones de derechos de acceso sobre el mismo recurso.

- Lista de control de acceso:

  Una lista es almacenada por cada recurso, proporcionando el dominio de los que tienen el acceso al recurso y las operaciones permitidas en cada dominio.

I.4.3 Comunicaciones

Una diferencia importante entre un sistema distribuido y un sistema con un procesador es la comunicación entre procesos. En un sistema con un procesador, la mayor parte de la

comunicación entre procesos supone de manera implícita la existencia de la memoria compartida.

Debido a la ausencia de la memoria compartida, toda la comunicación en los sistemas distribuidos se basa en la transferencia de mensajes.

El núcleo del sistema operativo distribuido proporciona normalmente las primitivas para el paso de mensaje en uno o en ambos sentidos. Además, la comunicación en grupos es suministrada en muchos sistemas operativos distribuidos, incluidos Amoeba y Chorus.

I.4.4 Memoria Virtual

La memoria virtual es una abstracción que consiste de dar la impresión al usuario de tener una memoria de un tamaño mucho mayor a la que realmente tiene de memoria física. Esta es implantada transparentemente por la combinación de memoria primaria como circuitos RAM y dispositivos de almacenamiento secundario, por ejemplo discos.

En un sistema distribuido, el diseño de una memoria virtual es de considerable interés, ya que su implantación puede necesitar utilizar almacenamiento secundario en una computadora separada de la que tiene la memoria primaria, además un dato puede ser compartido, y éste puede estar simultáneamente mapeado en espacios de memoria de diferentes procesos en diferentes computadoras.

En un sistema de memoria virtual son requeridas decisiones en dos áreas:

1. Política de localización del marco de página: Un algoritmo para decidir cúanta memoria principal debe ser utilizada para cada proceso que se este ejecutando.
2. Política de reemplazo de página: Es usado cuando debe ser buscada una página del almacenamiento secundario y no existe lugar disponible en memoria principal, entonces una página es seleccionada y debe ser reemplazada.

I.5 Ejemplos de sistemas operativos distribuidos

Se han diseñado varios sistemas operativos distribuidos, algunos como proyectos de investigación en algunas universidades; entre ellos se encuentran Amoeba, Chorus y Mach, todos basados en un micronúcleo para su uso en los sistemas distribuidos.

- Amoeba:

  Amoeba [13] se originó en Amsterdam, Holanda, en 1981, como proyecto de investigación en cómputo distribuido y paralelo. En el año de 1983, ya existía un prototipo inicial que tenía un nivel operacional. A partir de 1984, el proyecto de Amoeba se dividió y extendió su desarrollo a lugares de Inglaterra y Noruega con un proyecto de sistema distribuido de área amplia patrocinado por la Comunidad Europea. Este trabajo utilizó Amoeba 3.0, que, a diferencia de las versiones anteriores se basaba en las llamadas a procedimientos remotos.

  El sistema evolucionó durante algunos años, adquiriendo características como la emulación parcial de UNIX, la comunicación en grupo y un protocolo nuevo de bajo nivel.

  La arquitectura de Amoeba consiste de 4 componentes principales como se muestra en la figura 1.4. El primer componente refiere a las estaciones de trabajo (*wokstation*), una por cada usuario en las cuales los usuarios pueden llevar a cabo ediciones y otras tareas. El segundo componente son los grupos de procesadores (*processor pool*), los cuales pueden conectarse dinámicamente según sea necesario, utilizarse y retornarse luego al grupo.

  El tercer componente son los servidores especializados (*specialized servers*), tales como servidores de archivos, bases de datos, de arranque y otros servidores con funciones especializadas. El cuarto componente es la compuerta (*gateway*), la cual se utiliza para enlazar el sistema a diferentes sitios.

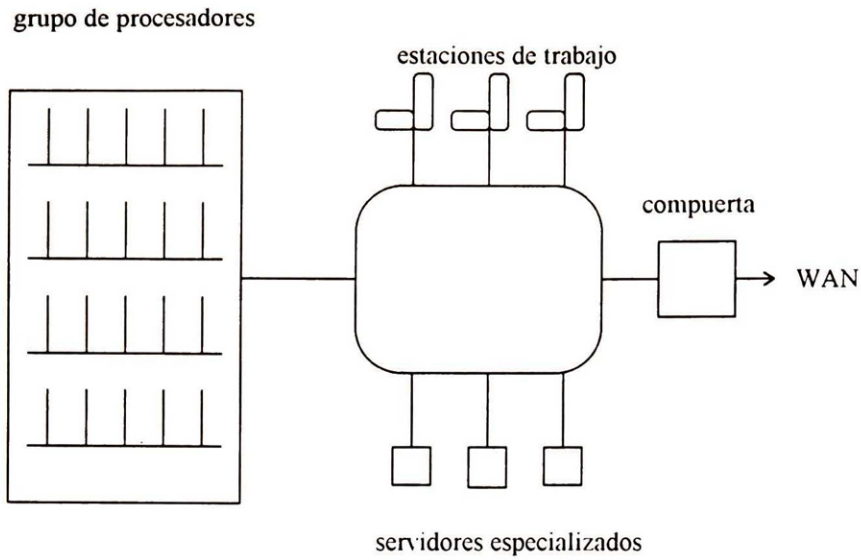


Figura 1.4 Arquitectura de Amoeba.

- Chorus:

  Chorus [3,4] surgió del instituto francés de investigación INRIA en 1980, como proyecto de investigación en sistemas distribuidos. La versión 3 se inició en 1987, y marcó la transición de un sistema de investigación a un proyecto comercial, ya que los diseñadores de Chorus salieron de INRIA y formaron una compañía para seguir su desarrollo e iniciar su comercialización.

  Se hicieron numerosos cambios técnicos en la versión 3, incluyendo un refinamiento del micronúcleo y de su relación con el resto del sistema, se introdujo las llamadas a procedimientos remotos (RPC) como el modelo de comunicación empleado.

  Para que Chorus fuese un producto comercial viable, se aumentó la capacidad de emulación de UNIX. Se agregó compatibilidad binaria, de modo que los programas en UNIX se ejecutaran sin compilarse de nuevo.

  El diseño de Chorus consta de 3 capas conceptuales: la capa del núcleo, los subsistemas y los procesos usuarios.

La capa del núcleo contiene al micronúcleo propiamente dicho, así como algunos procesos que comparten su espacio de direcciones. La capa intermedia contiene los subsistemas, que se utilizan para proporcionar el soporte del sistema operativo a los programas usuarios, que residan en la capa superior.

El micronúcleo proporciona seis abstracciones fundamentales: los procesos, los hilos, las regiones, los mensajes, los puertos, los grupos de puertos y los identificadores únicos. Los procesos proporcionan una forma de reunir y controlar los recursos. Los hilos son los entes activos del sistema y son planificados por el núcleo mediante un planificador basado en prioridades. Las regiones son áreas del espacio de direcciones virtuales que pueden tener segmentos asociados a ellos. Los puertos son buffers que se utilizan para guardar los mensajes recibidos que no han sido leídos. Los identificadores únicos son nombres binarios utilizados para la identificación de los recursos.

El micronúcleo y los subsistemas proporcionan juntos tres construcciones adicionales: las posibilidades, los identificadores de protección y los segmentos. Los primeros dos se utilizan para nombrar y proteger a los recursos del subsistema. La tercera es la base para la asignación de memoria, tanto dentro de un proceso en ejecución como en el disco.

- Mach:

  Mach [1,3] es un sistema operativo basado en un micronúcleo. Se diseñó con el fin de proporcionar una base para la construcción de nuevos sistemas operativos y la emulación de los ya existentes.

  Desde el inicio del proyecto Mach, éste estuvo destinado a tener un impacto en la industria y en el desarrollo de investigaciones de sistemas operativos para arquitecturas paralelas. De hecho el proyecto estuvo planeado con la colaboración

entre Digital Equipment Corporation (DEC) y CMU, financiado con fondos de un programa de alto rendimiento en el cómputo de DARPA[1]

Mach fue escogido como base para la investigación en sistemas operativos y como ambiente para la programación en una amplia gama de plataformas.

I.5.1 Comparación de los sistemas operativos distribuidos Amoeba, Mach y Chorus

En la sección anterior se describieron tres sistemas operativos distribuidos basados en micronúcleo: Amoeba, Mach y Chorus. Aunque tienen ciertos puntos en común, Amoeba y Mach difieren en muchos aspectos técnicos. Chorus ha retomado muchas ideas de Amoeba y Mach, y por tanto tiene una posición intermedia entre ambos.

Amoeba, Mach y Chorus tienen una historia y filosofía diferentes. Amoeba se diseñó como un sistema distribuido para su uso en una colección de CPU conectados entre sí mediante una LAN. Mach se inició como un sistema operativo en un procesador y después se añadieron los multiprocesadores y las LAN. Chorus se inició como un proyecto de investigación de sistemas operativos distribuidos, un tanto lejano del mundo de UNIX, y ha pasado revisiones, acercándose cada vez más a UNIX.

Amoeba sigue la filosofía de tener un micronúcleo mínimo, donde la mayor parte del código se encuentra en los servidores a nivel usuario. Por el contrario, los diseñadores de Mach deseaban proporcionar la funcionalidad suficiente en el núcleo como para controlar el rango más amplio posible de aplicaciones. En muchas áreas, Amoeba tiene una forma de hacer algo y Mach tiene dos o más, cada una de las cuales es más eficaz o conveniente en distintas circunstancias. En consecuencia, el núcleo de Mach es mucho mayor y tiene cinco veces el número de llamadas al sistema que Amoeba. Chorus ocupa una posición intermedia entre Amoeba y Mach.

[1] *DARPA, Defense Advanced Research Projects Agency.*

Amoeba tiene una implantación de llamada a procedimiento remoto (RPC) rápida a través de la red, mientras que el mecanismo de copia durante la escritura de Mach proporciona una comunicación de alta velocidad en un nodo. Chorus ha enfatizado principalmente en los sistemas con un solo procesador y multiprocesadores, pero como la administración de la comunicación se realiza en el núcleo (como en Amoeba) y no en un proceso usuario (como en Mach), en potencia tiene un buen desempeño en cuanto al RPC.

Amoeba y Mach tiene muchos aspectos en común, pero también varias diferencias. Ambos tienen procesos e hilos y se basan en la transferencia de mensajes.

Un componente esencial de Mach, Chorus y Amoeba es el micronúcleo, cuya función es proporcionar las abstracciones básicas del sistema como son: hilos, procesos, memoria e IPCs. Sobre el micronúcleo de estos sistemas se construyen los diferentes servicios que proporciona el sistema operativo. Un subconjunto de los servicios ofrecidos por Mach y Chorus emulan los servicios de UNIX, lo que los hace particularmente interesantes, pues no sólo presentan una interfaz conocida, sino que ofrecen capacidades que van más allá de los tradicionales sistemas UNIX.

Los sistemas operativos distribuidos deben interactuar actualmente con los núcleos de los sistemas operativos tradicionales, tales como UNIX, ya que se ejecutan en muchas estaciones de trabajo y para los cuales existen muchas aplicaciones.

En este capítulo se han abordado, de forma general, a los sistemas distribuidos enfatizando en los aspectos básicos de los sistemas operativos distribuidos y ejemplificando tres de ellos. Por su importancia dentro de nuestro proyecto e interés en desarrollar investigaciones sobre el micronúcleo Mach, se dedica el segundo capítulo a la explicación de sus características y a presentar diferentes propuestas de servidores desarrollados sobre el mismo.

CAPITULO II:
Mach

Si miramos hacia el futuro,podemos ver cómo elegir las operaciones correctas
- Doug Engelbart

II.1 Introducción

Mach está diseñado para incorporar muchas innovaciones recientes en el campo de los sistemas operativos para producir un sistema operativo completamente funcional y de tecnología avanzada. Uno de los objetivos claves de Mach es ser un sistema operativo distribuido capaz de funcionar en plataformas heterogéneas.

En este capítulo se dará una descripción de la historia de Mach y de los principales conceptos del micronúcleo, además se expone qué factores favorecieron a que pasara de un proyecto de investigación universitaria a un producto comercial.

II.2 Historia

Las primeras raíces de Mach van hasta un sistema llamado RIG[1] [10,11] que se inició en la Universidad de Rochester en 1975. RIG fue escrito para una minicomputadora de 16 bits y su principal objetivo de investigación era demostrar que se podían estructurar los sistemas operativos de una manera modular, como una colección de procesos que se comunican entre si mediante la transferencia de mensajes.

En 1979 investigadores de la Universidad Carnegie Mellon (CMU) continuaron desarrollando sistemas operativos con transferencia de mensaje, pero con una circuitería *(hardware)* más moderna. El nuevo sistema operativo se llamó Accent [12] y superó a RIG al añadir protección, además se podía operar de manera transparente a través de la red y soportaba una memoria virtual de 32 bits entre otras características.

[1] *Rochester Intelligent Gateway.*

RIG y Accent ofrecieron importantes lecciones en la transferencia de tecnología. Ambos RIG y Accent fueron implantados para arquitecturas de máquinas específicas: la Data General Eclipse y la Perq Systems PERQ, respectivamente y ambas proporcionaban un único ambiente de programación. Tampoco tuvieron un desarrollo con éxito en la comunidad de usuarios y ambos murieron con el declive de la plataforma de hardware para la cual fueron desarrollados.

Fue evidente de la experiencia de RIG y Accent que el éxito de Mach debía depender de la capacidad de permitir nuevas tecnologías de *hardware* y *software* y desarrollar sin ignorar las inversiones efectuadas en sistemas ya existentes.

El programa de RIG y Accent puso de manifiesto un factor sintomático en la manufactura e investigación de la década de los 80', como fue forzar para acomodar las nuevas y viejas tendencias de los sistemas de hardware y software, tales como:

- Arquitectura de las unidades centrales de procesamiento (CPU): sistemas CISC (computadoras con un conjunto complejo de instrucciones) y sistemas RISC (computadoras con un reducido conjunto de instrucciones).

- Arquitectura de la memoria para sistemas uniprocesadores y arquitectura de la memoria para sistemas multiprocesadores.

- Organización de entrada/salida: buses y redes.

Mach era una versión modificada de 4.1BSD[2], con características adicionales insertadas para la comunicación y la administración de la memoria.

Cuando se dispuso de 4.2BSD y 4.3BSD, el código de Mach se combinó con ellos para producir versiones más actualizadas. Aunque este método produjo un núcleo de gran

[2] *Berkeley Software Distribution.*

tamaño, garantizó la absoluta compatibilidad con el código de Berkeley, un objetivo importante para DARPA y para que Mach se convirtiera en un exitoso sistema operativo.

La primera versión de Mach apareció en 1986, para la VAX 11/784, un multiprocesador de 4 CPU (Unidad Central de Proceso). Poco después se realizaron versiones para la IBM PC/RT y la Sun 3. Mach se puso a la vanguardia de las actividades de la industria cuando en 1989 la *Open Software Fundation* (OSF), un consorcio de vendedores de computadoras dirigidas por IBM, DEC y Hewlett-Packart, anunció que usaría Mach como base para su nuevo diseño y eligió a Mach 2.5 para su sistema operativo OSF/1. En 1988, CMU eliminó todo el código de Berkeley del núcleo quedando un micronúcleo sólo con Mach.

El micronúcleo de Mach se diseñó como base para emular UNIX y otros sistemas operativos. La emulación se lleva a cabo mediante una capa de programas que se ejecutan fuera del núcleo. Cada emulador consta de una parte que está presente en el espacio de direcciones de los programas de aplicación, así como uno o más servidores que se ejecutan de manera independiente a los programas de aplicación. Hay que observar que se pueden ejecutar varios emuladores al mismo tiempo, por lo que es posible ejecutar programas en 4.3 BSD y MS DOS en la misma máquina al mismo tiempo por ejemplo.

Existen varias versiones de Mach. La primera, versión 1.0 fue distribuida por CMU para permitir a otros investigadores que constataran el progreso de Mach. Esta versión soportaba memoria virtual y subsistemas IPC[3], pero no soportaba tareas ni hilos.

La versión 2.0 contenía un soporte completo para tareas e hilos. Versiones posteriores adicionan funcionalidad y compatibilidad con 4.3BSD, incluyendo la mayor parte de éste en el núcleo. La versión 2.5 implanta el sistema de archivo distribuido, el *Sun Network File System* (NFS) [3] y además contiene soporte para administración de memoria externa. Mach está disponible para una amplia gama de máquinas, incluyendo algunas Sun, Intel, IBM y DEC de un solo procesador y sistemas multiprocesadores DEC, Sequent y Encore.

---

[3] *Interprocess Communication.*

En la presentación de la versión de Mach 3.0 se ha reducido el tamaño del núcleo. La figura 2.1 muestra la estructura del sistema operativo Mach 3.0. Todo el código BSD fue eliminado del núcleo y se colocó a nivel de usuario, y el núcleo sólo contiene las abstracciones necesarias para apoyar a BSD y a otros sistemas operativos.

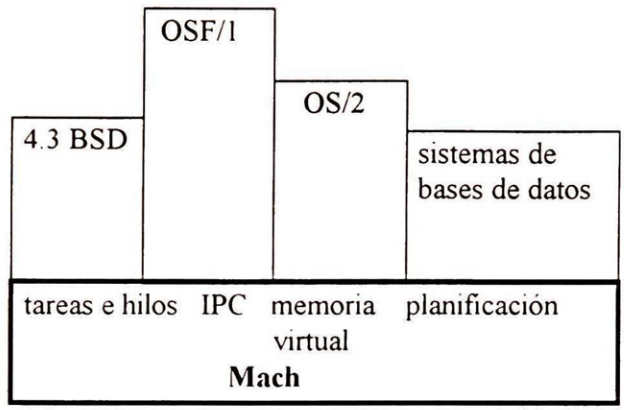


Figura 2.1 Estructura del Sistema Operativo Mach 3.0

En marzo de 1990, miembros de OSF anunciaron los planes de llevar productos al mercado basados en este sistema operativo. Este anuncio siguió al efectuado en septiembre de 1988 por NeXT que basaría su nuevo ambiente para las estaciones de trabajo en una versión del sistema operativo Mach. A la fecha se han escrito varios sistemas basados en Mach, para uniprocesadores y multiprocesadores. Mach ha sido usado en grandes números de investigaciones y proyectos de desarrollo en universidades y compañías, tomando alcance mundial.

Han existido un número de factores que fueron responsables de la rápida transición [15] de Mach, de un proyecto de investigación universitaria a un proyecto industrial:

1.- Los diseñadores comenzaron su trabajo con una sólida base de experiencia en trabajos de investigaciones previa.

Al inicio del proyecto, sus miembros ya tenían alrededor de 10 años de experiencia en el diseño e implantación de sistemas operativos. Esta experiencia condujo a interfaces más simples, más rápidas y a implantaciones más efectivas.

2.- La transferencia de tecnología a otros grupos de investigación y a la industria como una meta del proyecto.

Esto se debió a dos razones básicas: la primera, el proyecto comenzó como colaboración con el gobierno y la industria, y segundo, la experiencia de sus diseñadores condujo a la creencia de que la validación exitosa de las ideas de investigación, debían ser a través de la programación de aplicaciones de tipo industrial.

3.- Importantes necesidades fueron satisfechas al tiempo que las otros sistemas no lo hacían.

Debido al énfasis de la independencia al *hardware* en el diseño, manejo e implantación de Mach en diferentes plataformas, éste fue capaz de satisfacer las necesidades de muchos grupos de investigación dentro y fuera de CMU.

Además Mach proporciona soporte para una variedad de sistemas multiprocesadores, que suministran a los grupos de investigadores un ambiente UNIX para los sistemas multiprocesadores no disponible en el mercado comercial.

4.- Compatibilidad binaria con otros sistemas.

El costo de la transferencia de tecnología se redujo proporcionando compatibilidad binaria con los sistemas UNIX existentes, lo cual atrae a la construcción de nuevas aplicaciones utilizando Mach y experimentando con él las nuevas funciones y facilidades.

5.- Rendimiento.

Desde el inicio se determinó que el rendimiento debía ser igual o superior a los sistemas existentes, ya que la experiencia muestra que los mejores planes para la transferencia de tecnología pueden desviarse por el peor rendimiento de la nueva tecnología.

6.- Dedicación activa a la distribución del *software*.

Se desarrollaron herramientas y procedimientos para hacer Mach fácilmente disponible para las organizaciones externas.

La transferencia de tecnología de Mach a la industria fue fuertemente ayudada por los cambios en el mercado de los sistemas operativos.

Durante la década de 1980, los sistemas operativos llegaron a convertirse en un debate por el estándar de los sistemas abiertos, esto fue debido en gran medida al éxito de algunas compañías como Sun Microsystem, la cual utilizó a UNIX como su sistema operativo.

Varias compañías internacionales y grupos industriales intentaron definir, implantar y convenser a los usuarios de comprar un nuevo ambiente de computación abierta. Muchos de esos esfuerzos se centraron alrededor de versiones del sistema operativo UNIX, pero no hubo consenso de cual versión debía ser la base del estándar de los sistemas operativos abiertos. Sistemas no compatibles con UNIX como Macintosh OS, MS DOS y OS/2 fueron también vistos como estándares para varios fabricantes de *hardware* y *software*.

Toda esta actividad abrió las puertas para Mach y otros núcleos de sistemas operativos. Sistemas operativos tradicionales como UNIX fueron históricamente implantados "todo en una pieza", esto lo hace deficiente para adaptarse como soporte compatible para múltiples ambientes de sistemas operativos.

Existen algunas características que permitieron reducir el costo de la transferencia de tecnología, entre ellas se pueden mencionar las siguientes: el positivo impacto del rendimiento del sistema, poco costo de actualización, licencia gratuita del micronúcleo y distribución masiva.

- Costo de actualización.

  Durante la ejecución del proyecto fue adoptado el principio de proporcionar compatibilidad binaria para los sistemas que fueron implantados. Esto significa que el usuario no tiene inconveniente de intercambiar UNIX con Mach y así el usuario tomar las ventajas de nuevas aplicaciones e interfaces de programación.

- Licencia gratuita del micronúcleo.

  Todos los programas desarrollados por el proyecto Mach fueron proporcionados de forma gratuita y libre para su distribución a universidades, compañías y otras organizaciones. Una licencia ha sido utilizada para garantizar que los cambios y modificaciones efectuados al sistema sean retornados a CMU.

- Distribución masiva.

  En el proyecto no sólo se decidió que todo el sistema creado sea libre, sino que también fue seguida una política de distribución masiva.

  Comenzando desde 1987, la versión 0 de Mach fue distribuida a grupos dentro de CMU. En 1988 la versión 2 fue distribuida en más de 100 lugares fuera de CMU. Desde ahí, la distribución de Mach se ha caracterizado por la capacidad de actualización en sitios externos con las modificaciones y cambios recientes.

II.3 Conceptos fundamentales

Mach ofrece cinco abstracciones básicas [1,8,17], las cuales se muestran en la figura 2.2 y constituyen los bloques básicos para la construcción de los sistemas.

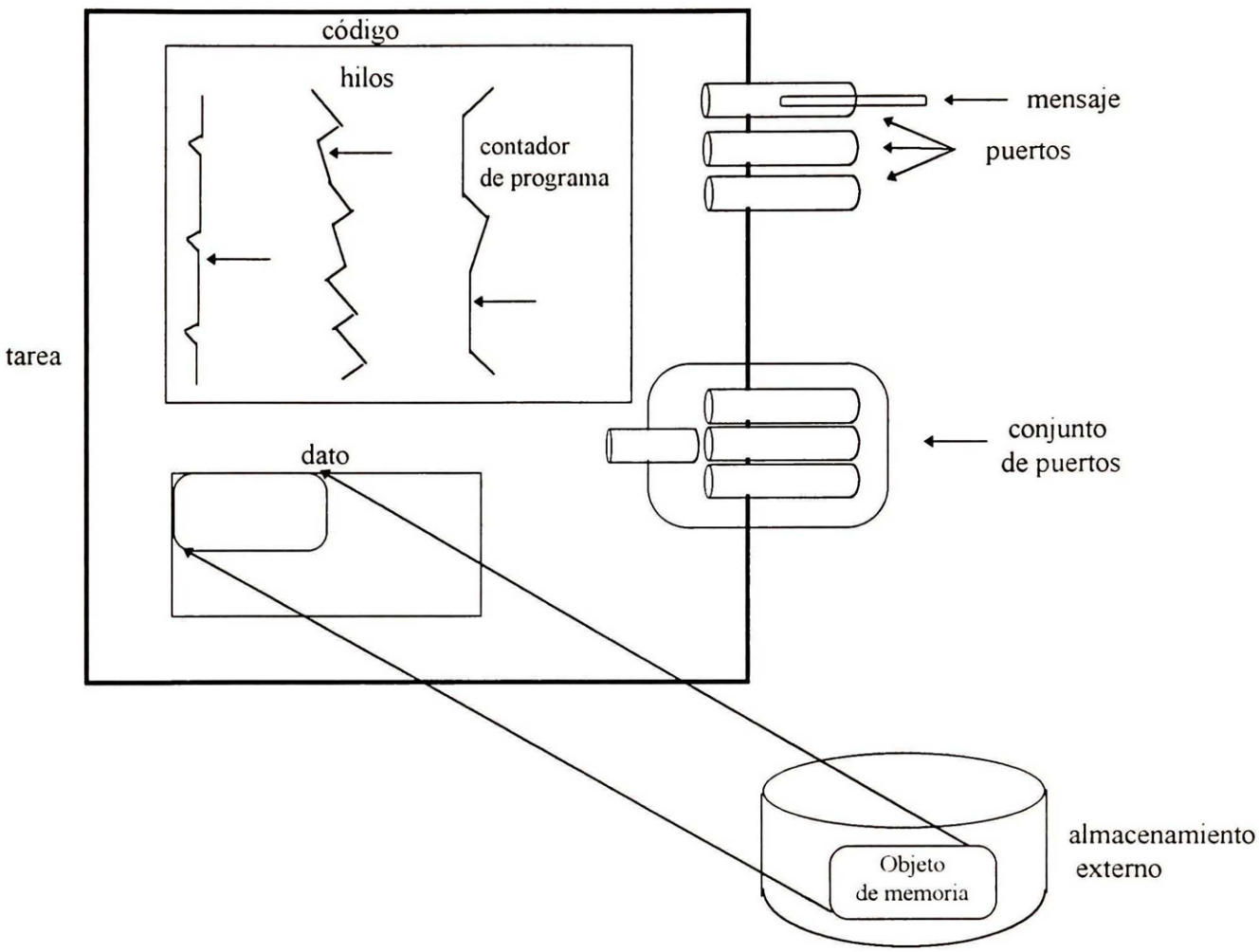


Figura. 2.2 Abstracciones del sistema operativo Mach.

Las abstracciones básicas son:

TAREA (*task)*: Es la unidad básica de asignación de recursos y cuenta con un espacio virtual de direcciones y acceso protegido a los recursos del sistema. La tarea contiene todos los recursos, se trata de un entorno de ejecución.

HILOS (*thread*): El hilo es la unidad básica de ejecución, así como el objeto de trabajo más pequeño y se ejecuta en el contexto de una tarea. Todos los hilos de una tarea comparten el espacio de direcciones y todos los recursos con otros hilos de la misma tarea.

El concepto tradicional de 'proceso' de UNIX; éste se implanta en Mach como una tarea con un solo hilo de ejecución. El concepto de proceso es dividido en dos componentes separados: tarea e hilo, como se muestra en la figura 2.3.

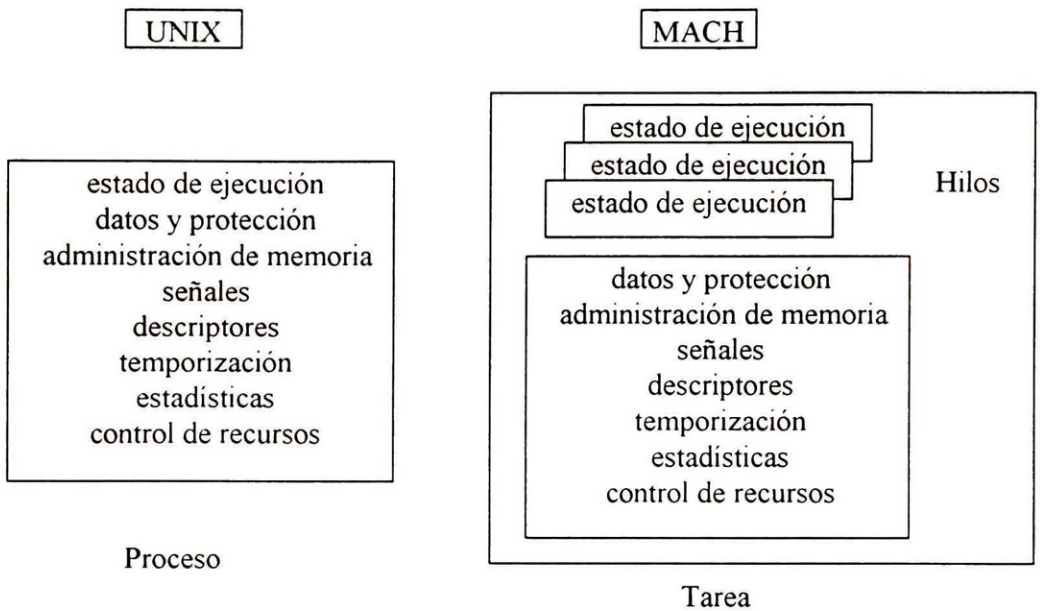


Figura 2.3 Diagrama de las abstracciones de proceso del sistema operativo UNIX y hilos y tarea del sistema operativo Mach.

Los hilos de Mach son controlados por el núcleo. El núcleo crea y destruye los hilos mediante la actualización de estructuras de datos.

En un sistema con una CPU, los hilos se ejecutan en forma concurrente gracias al entrelazamiento e sus acciones, primero se ejecuta uno y después otro. En un multiprocesador pueden existir varios hilos activos al mismo tiempo.

PUERTOS (*Port):* El medio de comunicación usado para la transferencia de los datos se denomina puerto. Toda la comunicación se lleva a cabo a través de los puertos, canales de comunicación unidireccionales administrados y protegidos por el núcleo. Funcionalmente éstos son colas de mensajes. Tanto el emisor como el receptor deben tener *derechos* para tener acceso al puerto. Por ejemplo, un emisor debe tener el derecho para enviar un mensaje a un puerto, de la misma manera que un receptor requiere el derecho de recepción para extraer mensajes de la cola. El puerto es un recurso y es propiedad de una tarea. Los hilos de una tarea pueden usar los puertos que pertenecen a la misma.

Los puertos se pueden agrupar en *conjuntos de puertos,* si así se desea. Un conjunto de puertos se define como un grupo de puertos que comparten una misma cola de mensajes. Un puerto puede pertenecer a lo más a un conjunto de puertos.

Cuando un hilo crea un puerto, obtiene un valor entero que identifica a éste, similar a un descriptor de archivo. Este valor se utiliza en las llamadas posteriores que envían mensajes al puerto o reciben mensajes de él, con el fin de identificar el puerto por utilizar.

La figura 2.4 muestra una situación en la que dos hilos, *A* y *B*, tienen acceso al mismo puerto. El hilo *A* envía un mensaje al puerto y *B* lee dicho mensaje. El encabezado y cuerpo del mensaje se copian de manera física de *A* al puerto y más adelante, del puerto a *B*.

MENSAJES (*Message*): El método básico de comunicación entre hilos en Mach es la transferencia de mensajes; éste es la unidad básica de transferencia, que puede definirse como una colección de datos, y puede contener datos en sí, apuntadores o referencias a los datos o capacidades de puertos (para transferir los derechos de acceso a puertos a otro hilo).

Hilos en diferentes tareas se comunican entre sí mediante el intercambio de mensajes y los que estan en la misma tarea por memoria compartida.

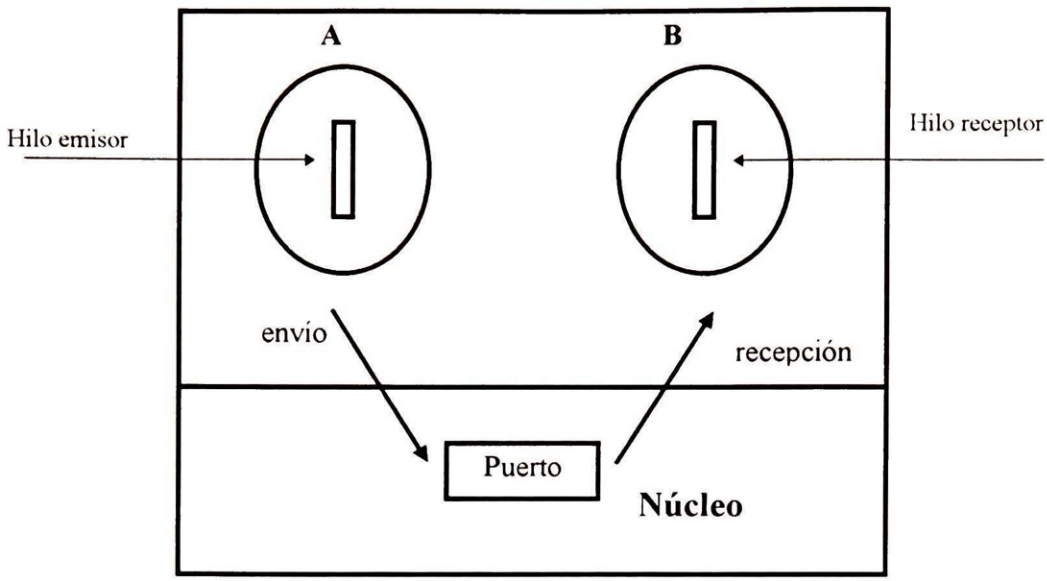


Figura 2.4 Transferencia de mensaje

OBJETOS DE MEMORIA *(Memory Object)*: Es una abstracción para soportar la administración de memoria y es un concepto fundamental relacionado con el uso del espacio de direcciones virtuales.

Mach tiene un poderoso sistema para la administración de la memoria, muy elaborado y muy flexible, basado en la paginación. El sistema para la administración de la memoria interactúa de manera muy cercana con los mecanismos de comunicación.

Gracias a la posibilidad de disponer un manejador de memoria, que comunica con el núcleo mediante un protocolo bien definido, los usuarios pueden implantar su propio paginador externo, que controle la parte lógica del sistema para la administración de la memoria, principalmente el manejo de los espacios de respaldo (disco).

Todos estos conceptos son parte fundamental de la programación en Mach y son utilizados en el micronúcleo. Una característica poco usual de Mach, y una clave para su eficiencia, es su combinación de características de memoria y comunicación.

II.4 Soluciones actuales de implantación de servidores sobre el micronúcleo Mach

Existen sistemas operativos comerciales como NeXTStep y OSF/1 donde se pueden ejecutar aplicaciones Mach 2.5, pero no es posible modificar en forma alguna dichos sistemas dado que no ofrecen los archivos fuente del sistema. Además ninguno de los dos ofrece las llamadas que brinda el micronúcleo Mach para que el programador de aplicaciones pueda controlar las aplicaciones de paginación de memoria virtual.

Por otro lado, existen servidores UNIX, sobre el micronúcleo Mach que no son servidores Linux o no tienen la documentación adecuada para ser modificadas y/o para crear nuevas aplicaciones y servicios.

Existen dos enfoques básicos en el diseño y construcción de servidores: el primero integra todos los servicios que ofrece y el segundo que se basa en el diseño de múltiples servidores individuales para proporcionar los servicios. A continuación se presentan algunos sistemas desarrollados sobre el micronúcleo Mach:

GNU HURD

GNU Hurd [7] es el proyecto de GNU para reemplazar el núcleo de UNIX. Hurd es una colección de servidores que corren sobre el micronúcleo de Mach para implantar el sistema de archivos, los protocolos de red, el control de acceso a los archivos y otras características y funciones que están implantadas por el núcleo de UNIX o núcleos similares (como lo es Linux).

La primera versión de prueba de Hurd se liberó el 6 de agosto de 1996. GNU HURD[4] presenta una gran innovación y un diseño interesante. Debe convertirse en el primer multiservidor completo que corra sobre Mach.

---

[4] Información sobre el sistema puede encontrarse en los siguientes lugares:
http://www.funet.fi/pub/OS/Hurd
ftp://ftp.gnu.ai.mit.edu/pub/gnu

Hurd está construido sobre el núcleo de Mach 3.0 de CMU y utiliza las facilidades de paso de mensajes y administración de memoria virtual. La versión actual de GNU HURD ya está disponible.

El servidor Lites

Lites [2,7] es un servidor basado en 4.4 BSD Lite y tiene una biblioteca de emulación que proporciona funcionalidad UNIX a los sistemas basados en Mach.

El servidor Lites fue desarrollado en la Facultad de Tecnología de la Información, Departamento de Ciencias de la Computación en la Universidad de Helsinki[5]

Lites proporciona compatibilidad binaria con 4.4 BSD, NetBSD (0.8, 0.9 y 1.0), FreeBSD (1, 1.5 y 2), 386BSD, UX (4.3 BSD) y Linux sobre plataformas i386 de Intel. También ha sido portado a PA-RISC y R3000. La biblioteca de emulación de Lites es completamente nueva, la cual permite una rápida extensibilidad y nuevas formas de implantar servicios.

El servidor Lites maneja multihilos, paginación y soporta llamadas al sistema concurrentes desde cualquier proceso. El sistema resultante es rápido y suficientemente estable para su uso.

El sistema Lites consiste de dos principales componentes: un servidor y un emulador. El emulador proporciona todas las interfaces de aplicación e implementa algunos servicios directamente. El servidor proporciona todos los servicios protegidos y el estado global del sistema: registros generales del procesador, contador de programa y puntero a la pila. Algunos servicios como los relativos a recursos físicos son proporcionados por el núcleo de Mach.

[5] http://www.cs.hut.fi/~jvh/lites-osdi-bof.html

Las aplicaciones se comunican con el servidor a través del emulador. El emulador utiliza diferentes formas de comunicación con el servidor. Estas incluyen RPC y memoria compartida.

El rol de la aplicación, y de hecho del emulador, es el de un cliente. Una de las metas de Lites fue crear un modelo e interfaces cliente-servidor limpias, bien separadas. De este modo el papel de los servidores es ser una entidad pasiva que responda a las peticiones.

El servidor Lites mapea la biblioteca de emulación en el espacio de direcciones de la tarea del usuario, y la manipulación de hilos en forma similar a *c-threads*.

Mklinux

Mklinux [7] es básicamente Linux ejecutándose como un servidor Linux en el micronúcleo Mach 3.0 de OSF

El micronúcleo OSF MK, originalmente desarrollado en *Carnegie Mellon* sobre Mach 3.0, ha sido subsecuentemente mejorado por el Instituto de Investigaciones de OSF[6]

Junto a Mklinux se pueden estar ejecutando otros sistemas operativos, tanto como el espacio del disco y la memoria RAM lo permitan. Existen versiones de Mklinux para Intel y Power PC. A la descripción de este sistema se le dedica una sección posterior.

[6] http://www.mklinux.apple.com/info/faq/Mach.html

Masix

MASIX[7] [14] es un sistema operativo distribuido multiservidor, actualmente bajo desarrollo en MASI Laboratory y está basado en el micronúcleo Mach 3.0, con múltiples soporte de *"personalidad"* Una 'personalidad' es la implantación de servicios para el sistema operativo. Su principal característica es la capa distribuida genérica[8] (DGL), la cual ofrece los servicios distribuidos a las personalidades. La figura 2.5 muestra la estructura del sistema.

El sistema tiene una estructura en dos capas, que son las siguientes:

1. La Capa Genérica Distribuida, la cual proporciona los servicios distribuidos (comunicación, memoria compartida, administración de procesos, tolerancia a fallas, entre otros) implantados como servidores.
2. Las personalidades, que implantan la semántica de los sistemas operativos tradicionales. Estos son construidos por varios servidores cooperativos.

El sistema está implantado en PCs, ofreciendo soporte de personalidades para Linux 1.2, FreeBSD 2.0 y Lites 1.1.

La arquitectura multiservidor distribuida de MASIX garantiza una alta modularidad. Su principal meta es la ejecución simultánea de múltiples personalidades, para ejecutar concurrentemente, en la misma estación de trabajo (*workstation*), aplicaciones de UNIX, DOS, OS/2 y Win32. MASIX también proporciona servicios distribuidos a las personalidades.

[7] http://www-masi.ibp.fr
[8] *Distributed Generic Layer*

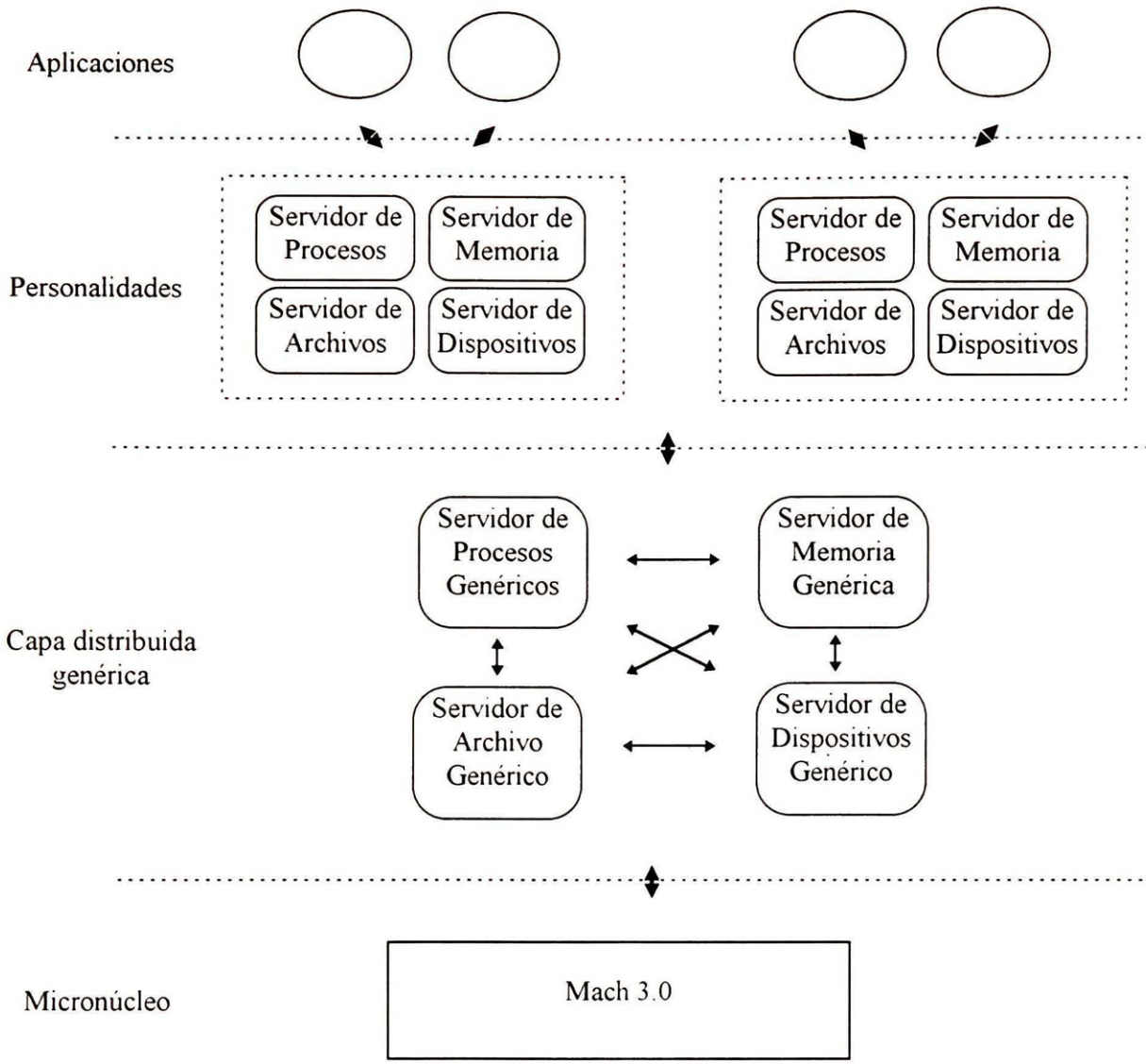


Figura 2.5 Estructura del sistema MASIX

Mklinux y Lites son ejemplos de sistemas que integran todos los servicios en un servidor y GNU Hurd y Masix ejemplos de sistemas multiservidor.

En este capítulo se explicaron las abstracciones básicas del micronúcleo Mach y se expusieron algunos trabajos actuales relacionados con la construcción e implantación de servidores. En este entorno, nuestra decisión fue poner en práctica el servidor Mklinux que es un servidor Linux sobre el micronúcleo Mach 3.0. Por la relevancia dentro de nuestro

trabajo se le dedica el capítulo 3 a explicar los conceptos básicos relacionados con el sistema operativo Linux y al análisis del servidor Mklinux.

CAPITULO III:
Linux

Somos yo y mi máquina el resto de la mañana,
el resto de la tarde, el resto de mi vida
- James Taylor

III.1 Introducción

Linux es un sistema operativo, que aunque no es un producto comercial respaldado por una gran corporación, es fruto de la creación de un equipo heterogéneo de entusiastas de la computación de todo el mundo. Este equipo se valió de los recursos de Internet para comunicarse, diseñar y construir este sistema operativo. Este proyecto aún no está finalizado, ya que prácticamente cientos de personas diseminadas en todo el mundo actualizan y amplían de manera constante a Linux.

A menudo se dice que Linux es un clon de UNIX, pero de hecho es un sistema operativo multiusuario y multitareas que cumple con POSIX. POSIX especifica la interfaz entre el sistema operativo y el usuario, de modo que los programas de aplicaciones sean transportables a través de distintas plataformas. Los estándares POSIX también son conocidos como IEEE Std 1003.

El objetivo, del desarrollo de Linux, ha sido crear un clon de UNIX libre de cualquier programa con derecho de autor comercialmente registrado - que cualquier persona pueda usar. Linux surgió como una idea de Linus Torvalds, mientras era estudiante en la Universidad de Helsinki, en Finlandia.

El sistema operativo UNIX, distribuido por diversos proveedores, viene en distintas presentaciones incluidas las versiones para plataformas Intel para PC; sin embargo, casi

todas son muy costosas. A ese respecto Linux ofrece una solución gratuita si usted tiene acceso a Internet[1]

Este capítulo describe las características generales del sistema operativo Linux, su historia y requerimientos para su instalación. Además hace un análisis del sistema Mklinux, un servidor Linux sobre el micronúcleo Mach 3.0, desarrollado por OSF.

III.2 Historia de LINUX

La historia de Linux [5] está ligada a la de UNIX y, en menor grado, a un programa llamado Minix. Hasta la versión 6 de UNIX, el código fuente se disponía en todas partes con autorización de AT&T[2], pero cuando ésta entregó UNIX V7 la emitió con una licencia que prohibía el estudio del código fuente. Para proporcionar el código fuente de un sistema operativo que pudiera ser estudiado en las universidades A.S. Tanenbaum [18] escribió un nuevo sistema operativo que sería compatible con UNIX desde el punto de vista del usuario, pero sin usar una sola línea de código de AT&T. A éste sistema se le llamó Minix.

Aunque AT&T fue el creador del sistema operativo UNIX, muchas otras compañías e individuos han tratado de mejorar durante años la idea básica. A continuación se examinan algunas de las principales variantes que se usan en la actualidad.

AT&T

> En 1969, un grupo de personas que trabajaba bajo las órdenes de Ken Thompson, desarrolló un sistema operativo que era flexible y totalmente compatible con las diversas necesidades de los programadores. Este nuevo sistema fue nombrado UNIX y se ha convertido en el estándar industrial de facto para los sistemas operativos multiusuarios y multitareas.

---

[1] Los archivos fuente se pueden obtener en ftp://ftp.funet.fi en /pub/Linux/PEOPLE/Linus o en otros servidores. Típicamente tienen el nombre linux-x.y.z.tar.gz, donde x.y.z es el número de la versión.

[2] *American Telephone and Telegraph Corporation*

BSD

Berkeley Software Distribution (BSD), en la Universidad de California en Berkeley, en 1978 dio a conocer su primera versión de UNIX, basada en la versión 7 de AT&T. El BSD UNIX, como se le conoce en toda la industria, contiene las mejoras desarrolladas para hacer de UNIX un sistema más amigable.

A pesar de no ser cien por ciento compatible con el UNIX original de AT&T, BSD logró su objetivo de tentar a los usuarios comunes de utilizar UNIX. BSD se ha convertido en el estándar académico.

Coherent

Coherent [5,6], otro clon de UNIX, lo distribuye *Mark William's Company*. Durante algún tiempo fue el clon más económico de UNIX de los que había comercialmente disponible en el mercado. Coherent aún tiene bastantes seguidores y varios grupos de noticias de USENET que siguen la evolución del producto.

USL

USL [5,6] o *UNIX System Laboratories* fue subsidiaria de la organización AT&T. USL produjo el código fuente para todos los derivados del System V de UNIX de la industria, sin poder vender ningún producto terminado en aquel entonces.

El último lanzamiento de UNIX de USL fue la versión 4.2 del System V de UNIX (SVR4.2). Con la compra de USL por parte de Novell, se ha cambiado el enfoque de USL (ahora parte de UNIX System Group de Novell) de productor de código fuente a productor de UNIX Ware.

XENIX, SunOS y AIX

A finales de los setenta y principios de los ochenta, Microsoft desarrolló su versión de UNIX, a la que llamó XENIX. Microsoft y AT&T fusionaron XENIX y UNIX en un solo sistema operativo llamado System V/386, versión 3.2, el que opera en casi cualquier *hardware* común. XENIX todavía puede conseguirse con *Santa Cruz Operation* (SCO), desarrollador conjunto de Microsoft, cuyo esfuerzo por promover XENIX en el mercado de las PCs ha hecho que esta versión de UNIX sea una de las más exitosas en el ámbito comercial.

Sun MicroSystems ha contribuido en gran medida al potencial de venta de UNIX mediante la promoción de SunOS y sus estaciones de trabajo correspondiente. El trabajo de Sun con UNIX produjo una versión basada en BSD.

La incursión de IBM en el mundo de UNIX dio como resultado un producto llamado AIX. AIX se desempeña bien, tal vez el viejo concepto de que cualquier versión de UNIX es un sistema poco amigable es lo que ha impedido que AIX tenga una mayor aceptación en el mercado.

Linux

Linux cobró vida como un proyecto de entretenimiento de Linus Torvalds, quien esperaba crear una versión más completa de UNIX para los usuarios de Minix, proporcionando una mejor plataforma que pudiera correr en las ampliamente accesibles PCs de IBM.

Linux ofrece una oportunidad de aprendizaje sin paralelo en la sociedad actual. Con él tiene un sistema operativo completo y funcional, que incluye el código fuente, en el cual puede probar y conocer el mecanismo que lo hace funcionar.

El uso de Linux proporciona muchas ventajas. De los numerosos sistemas operativos disponibles en la actualidad, Linux es el sistema gratuito más popular y de amplio acceso. Para las PCs de IBM, Linux proporciona un sistema completo con capacidades integradas para multiusuarios y multitareas que aprovechan toda la potencia de procesamiento de los sistemas de computo 386 y superiores. Linux viene con una adaptación de protocolo para red TCP/IP. Hoy día, Linux cuenta con miles de aplicaciones disponibles.

En la actualidad Linux se distribuye en varias formas: CD-ROM, cintas, a través de Internet e incluso discos flexibles. Linux es, de hecho, el núcleo del sistema operativo; pero una distribución Linux también contiene las utilerías y otros programas que usted espera encontrar en un sistema. Los requisitos mínimos para instalar Linux se describe en el apéndice A.

III.3 Características generales de LINUX

Los beneficios derivados del uso del sistema operativo UNIX [16], y por tanto de Linux, provienen de su potencia y flexibilidad. Estos son resultados de numerosas características integradas al sistema, las que están disponible tan pronto como se inicia el sistema.

A continuación se explican algunas de las características más importantes:

Multitarea:

La palabra multitarea describe la habilidad de ejecutar, aparentemente al mismo tiempo, numerosos programas sin obstaculizar le ejecución de cada aplicación. Cada programa tiene la posibilidad de ejecutarse. Linux realiza el procesamiento mediante el monitoreo, tanto de los procesos que están en espera de ejecución como de los que se están ejecutando. El sistema planifica cada proceso para que tenga la oportunidad de acceso al microprocesador.

Multiusuario:

La capacidad de Linux para asignar tiempo del microprocesador a numerosas aplicaciones simultáneas se prestó como causa para servir a numerosas personas al mismo tiempo, cada una ejecutando una o más aplicaciones.

***Shell* programables:**

El *shell* programable es otra característica que hace de UNIX y en consecuencia de Linux, lo que es: un sistema operativo flexible. Un shell hace el papel de intérprete o procesador de órdenes entre el usuario y el núcleo. Existen diferentes *shell* disponibles, pero la principal diferencia entre los tres shell que exponemos a continuación radica en la sintaxis de la línea de comandos, éstos son:

1. *Shell* de Bourne: Es el procesador de órdenes fundamental en UNIX, ofreciéndose en la mayoría de los sistemas. El programa se denomina `sh`.

2. C *Shell*: El C shell es el preferido por los programadores de UNIX, y aunque permite utilizar las mismas órdenes UNIX que el *shell de Bourne*, aporta las siguientes ventajas:
   - historial de órdenes,
   - control de tareas y
   - creación de alias.

3. *Bourne Again Shell*: *Shell* de *Free Software Fundation*. Fundación con fines no lucrativos cuyo objetivo es desarrollar y suministrar a todo el mundo UNIX gratuito. Su *shell* `bash` emplea las capacidades del *shell* de Bourne, y es utilizado por Linux como *shell* por defecto.

Independencia de los dispositivos bajo Linux:

Linux comparte muchos beneficios de la independencia de dispositivos. Por desgracia, una de las mejores características de Linux es al mismo tiempo una de sus mayores desventajas: su independencia del mundo comercial. De hecho, Linux no corre en algunas PCs de IBM, en especial las que utilizan el bus microcanal.

Durante los últimos años, Linux ha sido desarrollado por programadores de todo el mundo, quienes no tienen acceso a todos los equipos creados para las PCs de IBM y compatibles. Sin embargo, como cada vez son más los programadores que se unen al proyecto, son más los dispositivos (*hardware*) que se suman a los distintos núcleos y distribuciones de Linux.

III.4 MKLINUX: Un servidor Linux sobre el micronúcleo Mach 3.0

Desde la selección de Mach 2.5 como la base del sistema operativo OSF/1, OSF intentó basar sus desarrollos de sistemas operativos en el micronúcleo Mach 3.0 el cual proporciona un conjunto de abstracciones neutras, escalables y extensibles. El instituto de investigaciones de OSF ha desarrollado significativas mejoras y extensiones al núcleo original Mach 3.0 de CMU y el resultado llamado OSF MK, ya está liberado y disponible. Las últimas versiones de OSF/1 están basadas sobre OSF MK pero están protegidas por licencias comerciales.

Actualmente OSF proporciona dos versiones de OSF/1, cada versión de OSF/1 está instalada sobre el micronúcleo: OSF/1 1.3, un sistema apropiado para estaciones de trabajo y microcomputadoras con rendimiento muy competitivo comparado con otras versiones de UNIX y OSF/1 AD, disponible para usuarios de supercomputadoras de procesamiento paralelo masivo.

El núcleo OSF MK, al derivarse del micronúcleo de CMU es libre, pero el servidor OSF/1 está protegido por licencias comerciales incluyendo a *System V Release 2* (SVR2). En un

esfuerzo por eliminar estos obstáculos, investigadores del Instituto de Investigación de OSF (Grenoble y Cambridge), desarrollaron un servidor UNIX gratuito, para proporcionar a los investigadores y a la industria un sistema completamente sin licencia, apropiado para las necesidades de los programadores Mach.

El Instituto de Investigación decidió utilizar a Linux por varias razones, entre otras:

1. Linux, es la más popular de las implementaciones *libres* de UNIX,
2. Existen muchos proyectos de investigación basados en Linux, que podrían beneficiarse del micronúcleo,
3. Disposición del código fuente.

La combinación de Mach y Linux es referida como MKLINUX. Mklinux[3] consiste de dos conjuntos de programas:

- la versión del servidor Linux y
- la versión del micronúcleo Mach 3.0.

El micronúcleo Mach 3.0 proporciona las siguientes funciones: administración de memoria virtual, comunicación interproceso, primitivas de calendarización y manejadores de dispositivos.

El servidor Linux está disponible para las siguientes plataformas:

- Intel x86 (386 y superiores)
- Apple Power Mac[4]
- HP PA-RISC[5]

[3] http://www.gr.osf.org/mklinux/flyer.htm
[4] http://www.mklinux.apple.com
[5] http://www.osf.org/mall/OS/pa-mklinux/

La figura 3.1 muestra un diagrama de la estructura del sistema.

Figura 3.1 Estructura del sistema Mklinux.

La primera presentación sobre la liberación del sistema para plataformas Intel fue presentada en Febrero de 1996[6]

Para instalar el micronúcleo Mach y el servidor Linux para plataformas Intel, es necesario seguir un conjunto de instrucciones las cuales son detalladas en el apéndice B, además en el mismo se aborda y documenta los aspectos relacionado con el sistema Mklinux para plataformas Intel x86, detallando dónde obtener los archivos fuentes, y la metodología de implantación del sistema.

III.4.1 Problemas reportados

Mucho ha sido el interés de instalar y estudiar Mklinux a nivel mundial. Sin embargo este proceso no ha sido satisfactorio en todos los casos.

[6] http://www.gr.osf.org/mklinux/fs96.htm

Se han reportado investigaciones al respecto en España, Venezuela (*Unidad de Redes Telemáticas, Universidad de Carabobo*), Estados Unidos, y Australia (*University of Western, Australia*) por sólo citar algunos lugares.

OSF mantiene una lista de discusión (**mklinux-intel@gr.osf.org**) donde se solicita información, se intercambian criterios y se reportan problemas con el sistema Mklinux. Los temas abordados van desde cómo pronunciar Mklinux, cuál distribución de Linux a utilizar, hasta detalles específicos de la implantación en una máquina determinada, el estado actual del sistema y ambiente de trabajo para la compilación de aplicaciones.

Entre los aspectos más reportados están:

- Carga de Mach.

  El proceso de configuración y carga del sistema permite establecer en qué partición instalar Mklinux, los nombres de los dispositivos y la actualización de archivos de configuración necesarios, que deben ser utilizados, para su correcto funcionamiento. No obstante se han reportados problemas si la partición a utilizar no es la primera.

  Las soluciones recomendadas ante esta situación son:

  - Intercambiar las particiones y tener Linux en la primera partición.
  - Investigar que tarjeta controladora tiene su disco, ya que tarjetas como por ejemplo *aha-2940* no es soportada.

- Distribución de Linux a utilizar.

  Se han notificado problemas con la instalación de Mklinux en máquinas Linux. Cualquier distribución de Linux (*Slackware, Debian, Redhat*) es soportada. Sin embargo, deben tenerse en cuenta los requerimientos mínimos para instalar Linux en su máquina. Se recomienda por ejemplo tener procesador 486 o superior, con 16 MB de RAM y 200 MB para la versión de Slackware.

- Actualización de las variables de ambiente.

  El sistema puede fallar al intentar recompilar cualquier parte del sistema, si no se encuentran los archivos correctos para realizarlos.

  La solución para ello es actualizar los PATH adecuadamente y asegurarse que está trabajando en el *sandbox* correcto. En el apéndice A se describen los conceptos básicos del ambiente de desarrollo de OSF (ODE).

CAPITULO IV:
Utilización de las llamadas al micronúcleo Mach 3.0

Los caminos de la vida no son como yo pensaba,
no son como imaginaba,
son muy difíciles de andar.
- dominio público

IV.1 Introducción

Mach es un buen ejemplo de un interesante, popular y relativamente nuevo sistema operativo distribuido. Desde su selección como base del sistema operativo OSF/1 de Open Software Fundation (OSF), su uso ha crecido en aceptación.

Sistemas como Mach o similares, están permitiendo sacar la computación distribuida de los laboratorios de investigación al mundo comercial; de ahí la importancia del dominio y estudio de la programación en estos sistemas.

Las aplicaciones desarrolladas, sobre el sistema Mklinux instalado, estuvieron inspiradas en "Programming Under Mach" [1], en el cual se hace una presentación detallada de las abstracciones básicas del micronúcleo Mach, explicando las llamadas al sistema y ejemplificando su uso.

Los programas presentados allí están soportados en Mach 2.5, por tanto era necesario adecuar estos programas con las llamadas al sistema, las estructuras y tipos de datos para que pudieran ejecutarse sobre el micronúcleo Mach 3.0. Lamentablemente la información disponible sobre las llamadas al micronúcleo Mach 3.0 está dispersa y no se cuenta con la documentación suficiente que ejemplifiquen los cambios que están presentes en este micronúcleo.

Este capítulo aborda la programación sobre Mach 3.0 y se presenta siguiendo la idea de exponer los conceptos y documentar las llamadas al micronúcleo empleadas. Se describen los detalles necesarios para la programación como son: archivos de cabecera, bibliotecas y

convenciones para escribir los programas Mach 3.0 sobre el sistema Mklinux. Los ejemplos abordan el manejo de hilos, tareas y la comunicación interproceso.

IV.2 Consideraciones para la programación

Esta sección aborda las consideraciones relacionadas con la escritura y compilación de programas Mach. Afortunadamente los mecanismos de compilación en Mach son simples, pero como todos, deben seguir algunas convenciones las cuales presentamos a continuación.

Archivos de cabecera (*header*)

Existe un archivo muy importante, que todos los programas Mach deben incluir. Este archivo se llama **mach.h** y comprende todas las definiciones de tipos e interfaces necesarias para programar en Mach.

Este archivo se encuentra en el subdirectorio:
/usr/src/mklinux-1.0b2/osf/export/at386/usr/include y en él también se encuentran todas las bibliotecas del sistema.

Compilación y bibliotecas

Para la compilación de los programas Mach, se necesitan el uso de bibliotecas apropiadas para brindar las funciones que usan. El archivo **libmach.a** contiene muchas de las funciones que se utilizan.

El proceso de compilación debe ser invocado de la siguiente forma:

```
gcc -o prog prog.c -lmach -Idir_include -Ldir_bibliotecas
```

donde prog simboliza el nombre del programa ejecutable a obtener y prog.c el nombre del archivo fuente.

El compilador gcc utiliza las opciones -I para indicar dónde buscar los archivos de inclusión y -L para indicar el directorio donde se encuentran las bibliotecas.

Para compilar programas que utilicen llamadas al sistema de Mach, debe indicarse la opción -lmach. Si se compilan programas que utilicen hilos c (*c threads*), se debe incluir la opción -lthreads y para la compilación de los programas que utilizan la biblioteca de hilos p (*p threads*) debe utilizarse -lpthreads.

Dentro del subdirectorio:
/usr/src/mklinux-1.0b2/osfmk/obj/at386/mach_services/lib se pueden localizar bibliotecas que se necesitan.

mach_init()

Un pequeño código de inicialización se necesita al inicio de la función **main()** en los programas Linux. La función **mach_init()** debe ser incluida en todos los programas que sobre Mklinux se realicen e invoquen llamadas del núcleo de Mach.

Si la función no es incluida, es muy probable que no se obtengan errores de compilación, pero sí en tiempo de ejecución, en la invocación de la primera llamada de una función al micronúcleo.

El error señalado es: **(ipc/send) invalid destination port**.

IV.3 Programación de aplicaciones

En esta sección se presentan y se describen los programas que fueron probados sobre el micronúcleo Mach 3.0, se detallan las funciones al micronúcleo y se exponen las diferencias con las llamadas al sistema de Mach 2.5.

IV.3.1 Aplicaciones con tareas e hilos

Multiplicación de matrices con un solo hilo de ejecución

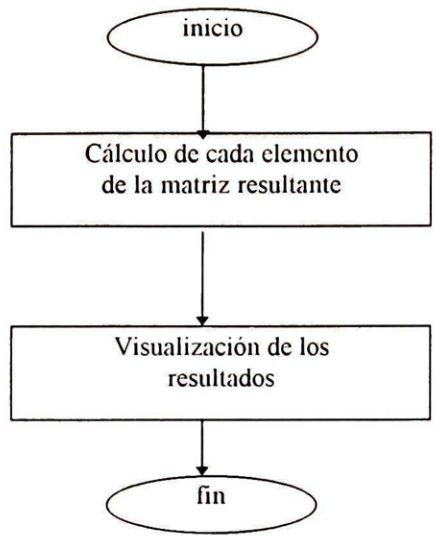
Este programa toma dos matrices mat_1 y mat_2, las multiplica y deja el resultado en la matriz result. Las tres matrices están implantadas estáticamente en memoria. Este programa nos va ha permitir examinar la versión de la multiplicación de matrices sin el uso de múltiples hilos.

Este ejemplo fue escogido por dos razones: la primera, ejemplificar que sobre éste sistema se pueden realizar programas que no invoquen las llamadas al micronúcleo de Mach 3.0; la segunda presentar un problema que es posible resolver utilizando las ventajas de la programación multihilo que está presente en el sistema operativo Mach.

El código fuente del programa se muestra en el apéndice C.1 y si asumimos que el programa fuente tiene nombre mul.c, su compilación debe invocarse de la siguiente manera para obtener un archivo ejecutable de nombre mul:

```
gcc -o mul mul.c -lmach -I/usr/src/mklinux-1.0b2/osfmk/export/at386/usr/include
    -L/usr/src/mklinux-1.0b2/osfmk/obj/at386/mach_services/lib/libmach
```

El diagrama en bloques de éste programa, para la multiplicación de matrices utilizando un único hilo de ejecución se muestra a continuación:



La técnica de programación utilizada es modularizar el programa. La función matrix_multiply() lleva a cabo la multiplicación de las dos matrices mat_1 y mat_2. Los resultados son visualizados en pantalla mediante la función matrix_print(). Además su ejecución es en forma secuencial.

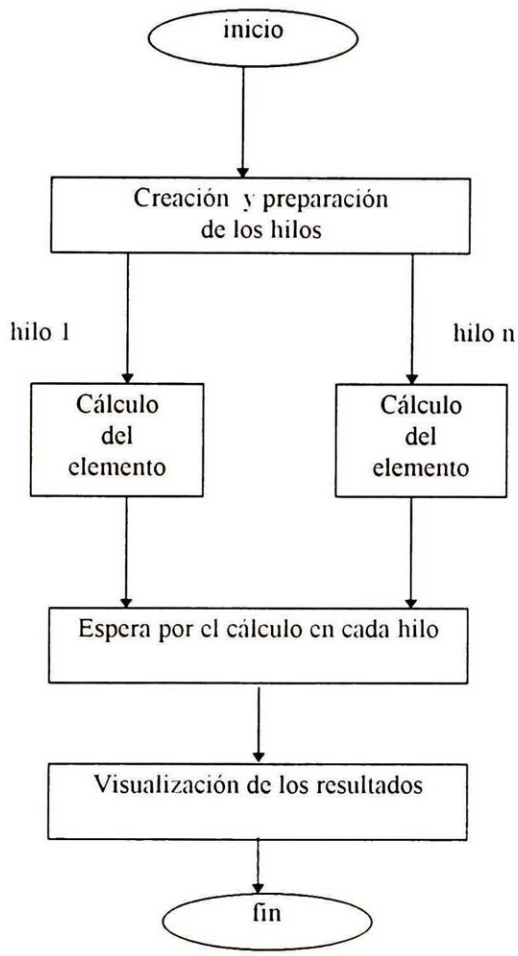
Multiplicación de matrices usando múltiples hilos

El ejemplo anterior presentó el programa de multiplicación de matrices utilizando un solo hilo, sin embargo como el resultado del cálculo de cada elemento de la matriz resultante es independiente del otro, este problema puede ser descompuesto en múltiples hilos que aprovechen la capacidad del sistema y que se ejecuten independientemente. Esta independencia debe entenderse que los hilos no requieren interactuar entre ellos para realizar su ejecución.

La variante multihilo requerirá compartir los datos, y deben ser accesibles para todos los hilos. Recordemos que todos los hilos en Mach, que pertenecen a una tarea comparten los recursos de ella, entre ellos la memoria.

La versión de la multiplicación de matrices con un simple hilo, es sencilla; sin embargo aunque en la versión multihilos se sigue esa misma estructura básica, se requerirá introducir algunos cambios para utilizar la programación multihilos, ya que para trabajar con hilos, estos deben ser creados, prepararlos para su ejecución e inicializarlos.

El diagrama en bloques de la versión de multiplicación de matrices utilizando múltiples hilos se muestra a continuación:



Comenzaremos analizando los detalles relativos de las llamadas al sistema para el trabajo con hilos que son necesarios utilizar, estas son: la creación, preparación y ejecución de los hilos.

Creación de hilos

Mach proporciona la llamada **thread_create()** para la creación de un nuevo hilo. Su sintaxis es la siguiente:

```
kern_return_t thread_create(
        mach_port_t parent_task,
        thread_port_t child_thread
        );
```

donde

parent_task es la tarea donde se ubicará al nuevo hilo y

child_thread es el nombre asignado por el núcleo para el nuevo hilo

La función **thread_create()** crea un nuevo hilo en la tarea con identificador **parent_task**. Si la llamada se completa sin error un identificador para el hilo es devuelto en el argumento **chid_thread** y el núcleo retorna la constante **KERN_SUCCESS**. La llamada puede fallar si **parent_task** no es una tarea válida o si el núcleo no tiene suficientes recursos para asignar al hilo. El nuevo hilo creado está en estado suspendido.

La función **mach_task_self()** puede ser usada, y de hecho lo es muy frecuentemente, como el primer argumento de la función **thread_create()**, ya que la misma devuelve un apuntador a la tarea donde actualmente se está ejecutando el hilo.

Su sintaxis es:

```
#include mach/mach_traps.h

mach_port_t mach_task_self(
        void
        );
```

La función **mach_task_self()** no requiere parámetros de entrada. Las llamadas proporcionadas para estos fines en el micronúcleo Mach 2.5 son:

- para la creación del hilo

```
kern_return_t status;
task_t parent_task;
thread_t new_thread;

status=thread_create(parent_task, &new_thread);
```

- para la obtención de un identificador a la tarea donde se está ejecutando el hilo

```
task_t actual_task;

actual_task=task_self();
```

La construcción en el micronúcleo Mach 3.0

```
kern_return_t status;
thread_port_t th;

status=thread_create(mach_task_self(), &th);
```

permite crear un nuevo hilo en la tarea donde esta invocación se realice, por tanto el hilo **th** pertenecerá a la tarea del hilo que ejecuta la llamada **thread_create()**.

Los tipos de datos *task_t* y *thread_act_t* en Mach 2.5 son compatibles con los tipos **mach_port_t** y **thread_port_t** en Mach 3.0 respectivamente.

Preparación del hilo para correr

El nuevo hilo creado por la función **thread_create()** no está preparado para ejecutarse. Primero deben ser asignados al procesador algunos valores relativos al estado del procesador; estos datos corresponden con la información de estado y en

ella se actualiza el puntero de la pila (*stack*), la dirección de la primera instrucción y el estado de las banderas. Típicamente, el estado de un hilo está compuesto por los registros del microprocesador y cada arquitectura requiere inicializar el estado para que el hilo se ejecute.

La llamada **thread_set_state()** es utilizada por el micronúcleo Mach 3.0 para especificar el estado de ejecución. Su sintaxis es la siguiente:

```
kern_return_t thread_set_state(
        thread_port_t target_thread,
        thread_state_flavor_t flavor,
        thread_state_t new_state,
        target_thread new_state_count
        );
```

donde

target_thread es el identificador del hilo para el cual se actualizaran los parámetros,

flavor identifica el tipo de arquitectura que será utilizada,

new_state la información de estado que se actualizará y

new_state_count es el tamaño del *buffer* con la información de estado.

Para los sistemas basados en la arquitectura del Intel 386, la estructura **thread_state_t** tiene el tipo **i386_thread_state**. El formato de los valores que debe actualizarse es específico para cada procesador y está definido en el archivo: **mach/thread_status.h.**

En Mach 2.5 la llamada al sistema *thread_set_state()* realiza la misma función que en el micronúcleo Mach 3.0, pero tiene que invocarse de la siguiente forma:

```
kern_return_t status;
thread_t th;
int flavor;
```

thread_state_t new_state;
unsigned int count;

status=thread_set_state(th, flavor, new_state, count);

La llamada **thread_set_state()**, no es portable entre múltiples arquitecturas y la diferencia entre Mach 2.5 y Mach 3.0 para esta llamada radica esencialmente en los cambios de tipos de datos.

Ambos micronúcleos ofrecen otras llamadas relativas a la manipulación del estado de un hilo, que no son explicadas aquí.

Ejecución del hilo

Cuando el hilo es creado, Mach inicializa su estado como *suspendido* (contador=1), en esta condición el hilo no se ejecuta. Para que un hilo pueda ejecutarse debe invocar a la llamada **thread_resume()**. La sintaxis de esta llamada es:

```
kern_return_t thread_resume(
        thread_port_t th
        );
```

donde

th es un identificador que especifica el hilo que se ejecutará.

Como resultado de la llamada se decrementa el contador del hilo y cuando este llegue a 0, el hilo podrá ejecutarse. Mach no garantiza que el hilo se ejecute inmediatemente. El núcleo de Mach determina qué hilo debe ejecutarse en base a las prioridades del hilo y a los recursos del procesador.

Eliminación del hilo

Cuando un hilo ha completado su ejecución, este debe deshacerse. La llamada **thread_terminate()** realiza esta función.

La sintaxis de la llamada **thread_terminate()** es la siguiente:

```
kern_return_t thread_terminate(
        thread_port_t target_thread
        );
```

donde

target_thread especifica el hilo a ser eliminado.

Como resultado de su ejecución destruye el hilo especificado.

Si se utiliza como argumento de la función **thread_terminate()** la llamada a la función **mach_thread_self()**, provoca que una vez finalizado su trabajo, el hilo se autodestruya.

La función **mach_thread_self()** es una trampa al sistema y devuelve una referencia al hilo que lo invoca. No requiere parámetros de entrada y se debe incluir en el programa que la utilice el archivo de biblioteca **mach_traps.h**.

La sintaxis de la función **mach_thread_self()** es la siguiente:

```
#include <mach/mach_traps.h>

thread_port_t mach_thread_self(
        void
        );
```

El tipo de dato **thread_port_t** en Mach 3.0 es compatible con el tipo thread_act_t en Mach 2.5.

Descripción del programa

En la versión del programa de multiplicación usando múltiples hilos, cada hilo realiza los cálculos de una porción de la matriz. La función matrix_multiply() no realizará realmente los cálculos, en su lugar ésta llamará a la función thread_fork() para crear diferentes hilos, prepararlos para la ejecución y comenzar su ejecución.

La función matrix_multiply() crea e inicia NTHREAD hilos para multiplicar las matrices mat_1 y mat_2. La nueva versión de matrix_multiply() se programa como sigue:

```
matrix_multiply()
{
int thread_num;

  for (thread_num=0; thread_num<NTHREADS; thread_num++) {
    threads[thread_num]=THREAD_NOT_DONE;
    thread_fork(thread_num);
  }
  wait_for_threads();
}
```

La función thread_fork() realiza tres acciones básicas:

1. Crear el nuevo hilo.
2. Actualizar el estado de la información.
3. Activar la ejecución del hilo.

Su código es como sigue:

```
thread_fork(int thread_num)
{
kern_return_t status;
thread_port_t th;

  status=thread_create(mach_task_self(), &th);
  if (status!=KERN_SUCCESS) {
    mach_error("thread_create: ", status);
    exit(1);
  }
  setup_thread(th, thread_num);
  status=thread_resume(th);
```

```
  if (status!=KERN_SUCCESS) {
    mach_error("thread_resume: ", status);
    exit(2);
  }
}
```

La función **setup_thread()** prepara al nuevo hilo para la ejecución. Antes que un hilo creado pueda ejecutarse la información de estado debe inicializarse. El código para preparar un hilo para su ejecución comprende:

1. Ubicar espacio para la pila del hilo e inicializar los punteros de inicio y fin de la misma.
2. Limpiar y actualizar la estructura de estado del nuevo hilo.
3. Almacenar el nuevo estado en el núcleo Mach.

El código de esta función se presenta a continuación.

```
setup_thread(mach_port_t th, int arg)
{
char *stack;
register int *top;
struct i386_thread_state state, *ts;
kern_return_t status;

  stack=(char *) &stacks[arg];
  top=(int *) (stack+STACKSIZE-sizeof(void));
  bzero( (char *) &state, sizeof(state));
  ts=(struct i386_thread_state *) &state;
  ts->eip=(int)(int(*)())thread_multiply;
  *--top=(int) arg;
  *--top=0;
  ts->uesp=(int) top;
  status=thread_set_state(th, i386_THREAD_STATE, (thread_state_t) &state,
                  i386_THREAD_STATE_COUNT);
  if (status!=KERN_SUCCESS)
    mach_error("setup_thread: ", status);
}
```

Todos los hilos creados utilizan la misma rutina, **thread_multiply()**. Esta función determina cuales elementos de la matriz resultante serán calculados. Su código es como sigue:

```
thread_multiply(int thread_num)
{
kern_return_t status;
int i,j;

  for (i=0; i<N; i++)
    for (j=thread_num; j<M; j+=NTHREADS)
      result[i][j]=compute_element(i,j);
  threads[thread_num]=THREAD_DONE;
  status=thread_terminate(mach_thread_self());
  if (status!=KERN_SUCCESS)
    mach_error("thread_terminate: ", status);
}
```

Aunque en el programa no existe una llamada explícita a la función **thread_multiply()**, esta es designada como la rutina de comienzo de cada hilo. Para cada hilo creado, la función **setup_thread()** actualiza la dirección de memoria de la función thread_multiply() como el valor del contador de programa asociado que debe iniciar cada hilo.

El código fuente para el ejemplo de la multiplicación multihilos se muestra en el apéndice C.2, y asumiendo que el nombre del programa fuente es mulh.c la compilación debe realizarse de la siguiente forma:

```
gcc -o mulh mulh.c -lmach
-I/usr/src/mklinux-1.0b2/osfmk/export/at386/usr/include
-L/usr/src/mklinux-1.0b2/osfmk/obj/at386/mach_services/lib/libmach
```

Alternativamente puede emplearse la función **thread_create_running(),** la cual realiza de forma automática la secuencia necesaria para obtener un nuevo hilo corriendo.

Información sobre los hilos

El siguiente ejemplo ilustra el empleo de la información que sobre el hilo maneja el núcleo Mach 3.0. El programa solicita la información para el hilo que se está ejecutando y para otro que es creado perteneciendo a la misma tarea, para ello realiza las siguientes operaciones:

- reserva espacio en memoria para almacenar la información.
- realiza la solicitud de la información básica al núcleo, para el hilo que se está ejecutando.
- visualizar la información solicitada.
- creación de un nuevo hilo.
- solicitud de la información básica al núcleo para este nuevo hilo creado.
- visualizar la información asociada al nuevo hilo.

La información que sobre los hilos maneja el núcleo, se puede obtener utilizando la llamada al sistema **thread_info()**. Mach retorna la información solicitada por el hilo en una estructura del tipo **thread_basic_info_t**.

Los siguientes elementos están comprendidos en la estructura que es retornada por la llamada **thread_info()**:

```
struct thread_basic_info {
        time_value_t user_time;
        time_value_t system_time;
        integer_t cpu_usage;
        policy_t policy;
        integer_t run_state;
        integer_t flags;
        integer_t suspend_count;
        integer_t sleep_time;
        }
```

donde

user_time es el tiempo de ejecución de usuario empleado por el hilo.

system_time es el tiempo de ejecución del sistema para el hilo.

cpu_usage devuelve el porciento de utilización de la cpu por el hilo.

policy identifica la política de planificación en uso.

run_state es el estado del hilo:

TH_STATE_RUNNING: El hilo se está ejecutando normalmente

TH_STATE_STOPPED: La ejecución del hilo está detenida

TH_STATE_WAITING: El hilo está en espera

suspend_count devuelve el *conteo del tiempo suspendido del hilo.*

La llamada al núcleo que para este fin proporciona Mach 2.5 es similar en nombre y cantidad de parámetros que el micronúcleo Mach 3.0, pero deben ser adecuados sus tipos correspondientes tanto de la función como algunos de los datos que la estructura devuelve.

El código fuente del programa se muestra en el apéndice C.3 y su compilación, si asumimos como nombre del programa fuente pthinfo.c, debe invocarse de la siguiente forma y se obtendrá un archivo ejecutable con nombre pthinfo:

```
gcc -o pthinfo pthinfo.c -lmach
-I/usr/src/mklinux-1.0b2/osfmk/export/at386/usr/include
-L/usr/src/mklinux-1.0b2/osfmk/obj/at386/mach_services/lib/libmach
```

La ejecución de este programa muestra la información del tiempo que llevan suspendido dos hilos y su ejecución es secuencial.

IV.3.2 Comunicación interprocesos

Quizás los mayores cambios observados en el micronúcleo Mach 3.0, en comparación con Mach 2.5, es la forma en que se implementa la comunicación interproceso. En esta sección se ilustra cómo implementa el micronúcleo este mecanismo, detallando los cambios reportados en las llamadas al sistema y en las definiciones de tipo de datos utilizados.

El ejemplo utiliza el paradigma cliente-servidor y está conformado por dos programas: uno que funciona como servidor y se ejecuta como tarea de fondo, respondiendo las solicitudes que se realizan.

El otro programa realiza la función del cliente y realiza la solicitud de información.

Mach utiliza los puertos para la comunicación. Los puertos son implantados como una cola de mensajes que el núcleo maneja. Los mensajes contienen información de control y datos.

Cada puerto es identificado por un único nombre, él cual es implantado como un valor entero.

Los sistemas basados en Mach están provistos de un servidor de nombre, el cual es conocido como servidor de mensajes en red (***netmsgserver***). Mach dota automáticamente a las nuevas tareas con los derechos de envío al *netmsgserver*, solucionando así los problemas de adquirir los derechos de envío.

Referencias al servidor de nombres

La función para registrar el nuevo puerto es **netname_check_in()**, la misma tiene cuatro parámetros: la referencia al servidor de nombre, el nombre del servicio que se desea registrar, la tarea a la que pertenecerá el puerto, y el identificador del nuevo puerto registrado.

Un fragmento de código que hace uso de esta llamada es el siguiente:

```
        char *servicename="Nada";
        mach_port_t port;
        kern_return_t kr;

(inst 1) kr=netname_check_in(
                name_server_port, /* identificador del servidor de nombre */
                servicename, /* nombre del servicio */
                mach_task_self(), /* tarea que contendrá el puerto */
                port /* identificador de puerto */
                );
(inst 2)  if (kr=KERN_SUCCESS) {
                fprintf(stderr, "servidor: la llamada a la funcion netname_check_in ()
                        falló con el error %s /n", mach_error_string(kr));
                exit(1);
                }
```

En el paso (inst 1) se registra un servicio de nombre "Nada" y que pertenece a la tarea donde se está ejecutando el hilo que la invoca. En el paso (inst 2) se verifica el resultado de la ejecución, si falló se visualiza un mensaje de error y se aborta la ejecución del programa.

La función mach_error_string(), devuelve una cadena de caracteres que corresponde al error indicado. La constante **KERN_SUCCESS** es el valor genérico que devuelven las funciones para indicar que su ejecución terminó exitosamente.

La función **netname_look_up()** permite obtener el puerto por el cual el servidor atiende las solicitudes, la cual tiene cuatro parámetros: el primero es el identificador del puerto del servidor de nombres que será consultado. El segundo es la máquina en la cual será buscado.

Si el nombre de una máquina es proporcionado en el segundo argumento, sólo en esta máquina se buscará; una cadena nula identifica la máquina local y un asterisco especifica que serán consultadas todas las máquinas que estén conectadas a la red. Esto proporciona una mayor flexibilidad, ya que no es necesario conocer en cual máquina se está ejecutando el servidor, además el servidor puede migrar de una máquina a otra. Sin embargo el uso de un servidor de otra máquina (remota) en la red siempre será menos eficiente que el uso de un servidor corriendo en la máquina local.

El tercer parámetro es el nombre (ASCII) que identifica el servicio. El cliente debe usar el mismo nombre con que el servidor lo registró. El último parámetro es un puntero a una variable que devolverá el puerto por el cual se atenderá el servicio, si la llamada es exitosa.

Un fragmento de código que ilustra el uso de la función se muestra a continuación:

```
char *servicename="Nada";
mach_port_t port;
kern_return_t kr;

(inst 3) kr!=netname_look_up(name_server_port, /* servidor de nombre */
                             "*", /* búsqueda en todas las máquinas */
```

```
                        servicename, /* nombre del servicio */
                        &port /* identificador del puerto */
                        );
(inst 4) if (kr!=KERN_SUCCESS) {
            fprintf(stderr, "cliente: machipc -> netname_look_up: %s \n",
                    mach_error_string(kr));
            exit(1);
            }
```

En el paso señalado como (inst 3) se hace la solicitud de búsqueda del puerto con el que fue registrado el servicio. Si la función es completada exitosamente, el paso (inst 4) lo verifica, la constante **KERN_SUCCESS** es retornada, y la variable del cuarto parámetro de la llamada del paso (inst 3) contendrá el identificador del puerto por el cual el servidor atenderá las solicitudes.

Funciones para el manejo de puertos

La primera operación que debe realizar el servidor es asignar el nuevo puerto para recibir las solicitudes.

Las llamadas al sistema **mach_port_allocate()** y **mach_port_deallocate()** asignan y liberan un puerto respectivamente.

La llamada **mach_port_allocate()** crea un nuevo puerto. A continuación se detallan los parámetros de la llamada **mach_port_allocate()**.

```
kern_return_t mach_port_allocate(
        ipc_space_t task,
        mach_port_right_t right,
        mach_prt_name_t *name
        );
```

donde

task es un parámetro de entrada que especifica la tarea a la cual se asigna el puerto.

right es un parámetro de entrada que especifica el tipo de entidad que será creada, puede ser uno de los siguientes valores:

- MACH_PORT_RIGHT_RECEIVE: La función crea un puerto. El nuevo puerto creado no pertenece a ningún conjunto de puerto. Este puerto sólo mantiene los derechos de envío y la cantidad de mensajes está limitada a la constante
- MACH_PORT_QLIMIT_DEFAULT. Las funciones **mach_port_extract_righ()** y **mach_port_insert_right()** pueden ser utilizadas para convertir el derecho de recepción en derechos combinados de envío y recepción.
- MACH_PORT_RIGHT_PORT_SET: La función crea un conjunto de puerto. El nuevo conjunto de puerto inicialmente no contiene ningún miembro.
- MACH_PORT_RIGHT_DEAD_NAME: Indica que el puerto sólo será utilizado como referencia.

name es el identificador para el puerto. Será cualquiera que no esté en uso.

Si es exitosa la invocación de la función **mach_port_allocate()**, un nombre de puerto es devuelto. De lo contrario, si no es posible crear el puerto, la función devuelve **KERN_NO_SPACE** especificando que no hubo espacio dentro de la tarea para ubicar el puerto solicitado.

El siguiente fragmento de código ejemplifica el uso de esta función. En el mismo se crea un conjunto de puertos y luego se especifica que el puerto por el cual se solicitará el servicio pertenece a este conjunto de puertos. Si las ejecuciones no son satisfactorias se aborta la ejecución del programa y se hacen las indicaciones correspondientes. De lo contrario, si es correcta, se devuelve un identificador de conjunto de puerto y un puerto que pertenece al mismo.

```
kern_return_t ret;
mach_port_t port, pset;

ret=mach_port_allocate(
        mach_task_self(),
```

```
        MACH_PORT_RIGHT_PORT_SET,
        &pset);
if (ret!=KERN_SUCCESS) {
        fprintf(stderr, "servidor: machipc -> mach_port_allocate(pset): %s \n",
                mach_error_string(ret));
        exit(1);
        }
ret=mach_port_move_member(
        mach_task_self(),
        port,
        pset);
if (ret!=KERN_SUCCESS) {
        fprintf(stderr, "servidor: machipc -> mach_port_move_member: %s
        \n", mach_error_string(ret));
        exit(1);
        }
```

Estructura del mensaje

Cada mensaje comienza con un encabezado, seguido del cual pueden haber cero o más parámetros. Este puede incluir derechos de los puertos y direcciones de grandes regiones de memoria.

Mostramos a continuación la declaración de un mensaje, el cual sólo tiene un campo que se corresponde con el encabezado del mensaje:

```
struct {
  mach_msg_header_t h;
  } msg;
```

La estructura del encabezado del mensaje está formado por varios campos que cumplen una función específica dentro del mismo, su estructura es la siguiente:

```
typedef struct {
        mach_msg_bits_t msgh_bits;
        mach_msg_size_t msgh_size;
        mach_port_t msgh_remote_port;
```

```
mach_port_t msgh_local_port;
mach_msg_size_t msgh_reserved;
mach_msg_id_t msgh_id;
} mach_msg_header_t
```

donde

msgh_bits es información para control del mensaje.

msgh_size es el tamaño del mensaje. El tamaño del mensaje debe ser especificado en *bytes* e incluye el encabezado, los descriptores de tipos, los datos incluidos en el mensaje (*in_line data*) y los punteros para los datos fuera del mensaje (*out_of_line data*).

msgh_remote_port identifica al puerto destino del mensaje y debe especificar un puerto con derechos válidos para asimilar la acción a desarrollar..

msgh_local_port identifica al puerto local.

msgh_reserved está reservado.

msgh_id no es interpretado por las primitivas del mensaje. Esta información normalmente especifica el formato o significado del mensaje..

Llamada al sistema para el soporte de IPC

La llamada al sistema **mach_msg()** se emplea para enviar y recibir los mensajes. La función utiliza la misma área de almacenamiento (*buffer*) para el envío y la recepción del mensaje.

La sintaxis de la llamada al sistema **mach_send()** es:

```
mach_msg_return_t mach_msg(
        mach_msg_header_t msg,
        mach_msg_option_t option,
        mach_msg_size_t send_size,
        mach_msg_size_t receive_limit,
        mach_port_t receive_name,
        mach_msg_time_out_t timeout,
        mach_port_t notify
        );
```

donde

msg es el *buffer* del mensaje utilizado por la función para las operaciones de envío y recepción.

option especifica las acciones a realizar. Uno o ambos de los siguientes valores debe ser especificado MACH_SEND_MSG o MACH_RCV_MSG. Otras opciones no son tomadas en cuenta.

send_size Cuando un mensaje será enviado, especifica el tamaño del mensaje en bytes

receive_limit Si el mensaje será recibido, especifica el tamaño en bytes del mismo. En otro caso debe suministrarse el valor 0.

receive_name En la recepción de un mensaje especifica el conjunto de puerto o el puerto. En otros casos debe suministrase la constante MACH_PORT_NULL.

timeout Cuando se utilizan las opciones MACH_SEND_TIME_OUT y MACH_RCV_TIMEOUT, este parámetro especifica el tiempo en milisegundos que debe esperar antes de abortar la ejecución (giving up). En otros casos debe suministrarse MACH_PORT_NULL.

notify Cuando se utilizan las opciones MACH_SEND_CANCEL y MACH_RCV_NOTIFY, este parámetro especifica el puerto utilizado para la notificación. En otro caso la constante MACH_PORT_NULL debe ser indicada.

Si el argumento **option** contiene el valor **MACH_SEND_MSG,** la llamada envía el mensaje. En el parámetro **send_size** se especifica el tamaño del buffer del mensaje (incluyendo encabezado y cuerpo del mensaje) a enviar.

Si el argumento **option** contiene el valor **MACH_RCV_MSG**, la llamada recibe un mensaje. El parámetro **receive_limit** especifica el tamaño del buffer que será recibido. Mensajes que excedan esta área (tamaño) no serán recibidos.

Si el argumento **option** contiene ambas constantes **MACH_SEND_MSG** y **MACH_RCV_MSG**, la llamada **mach_msg()** realiza ambas operaciones: envío y recepción en ese orden. Si la operación de envío encuentra un error (cualquier código de retorno

diferente a **MACH_MSG_SUCCESS**), la llamada retorna (aborta) inmediatamente sin intentar la operación de recepción.

Semánticamente la combinación de las opciones de envío y recepción es equivalente a realizar por separado las operaciones.

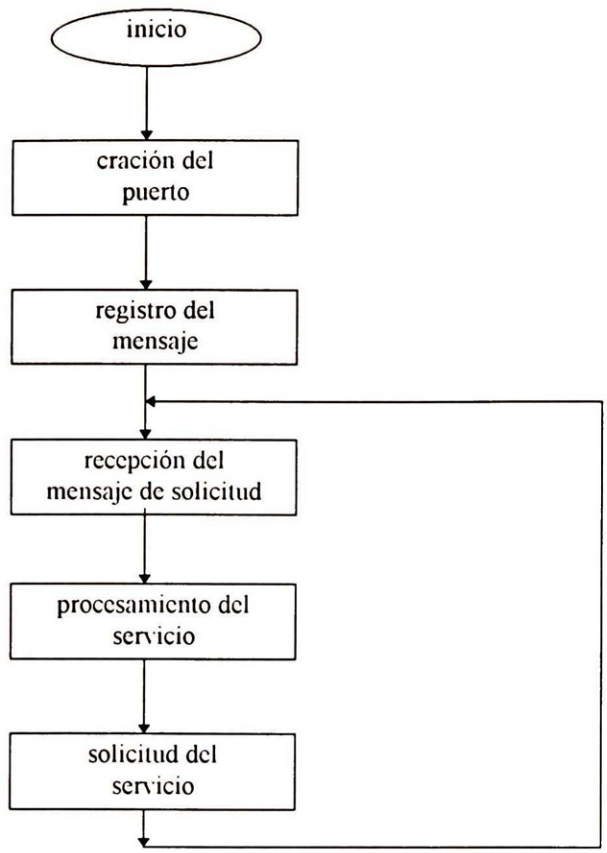
Si el argumento **option** no especifica ni **MACH_SEND_MSG** o **MACH_RCV_MSG**, la función **mach_msg()** no realiza ninguna acción.

Descripción de los programas

Servidor:

Este ejemplo define un servidor que recibirá una solicitud y devolverá el mismo mensaje. Inicialmente se reserva espacio en memoria para el puerto y lo registra en el servidor de nombres. Su ejecución está a la espera permanentemente de solicitudes de servicio.

El diagrama en bloques del programa se muestra a continuación:



La compilación y ejecución del programa debe realizarse de la siguiente forma:

- compilación:

```
gcc -o servidor servidor.c -lmach -lnetname
-I/usr/src/mklinux-1.0b2/osfmk/export/at386/usr/include
-L/usr/src/mklinux-1.0b2osfmk/obj/at386/mach_services/lib/libmach
-L/usr/src/mklinux-1.0b2/osfmk/obj/at386/mach_services/lib/libnetname
```

La opción **-lnetname** debe indicarse para que las funciones relacionadas con el servidor de nombre sean resueltas satisfactoriamente.

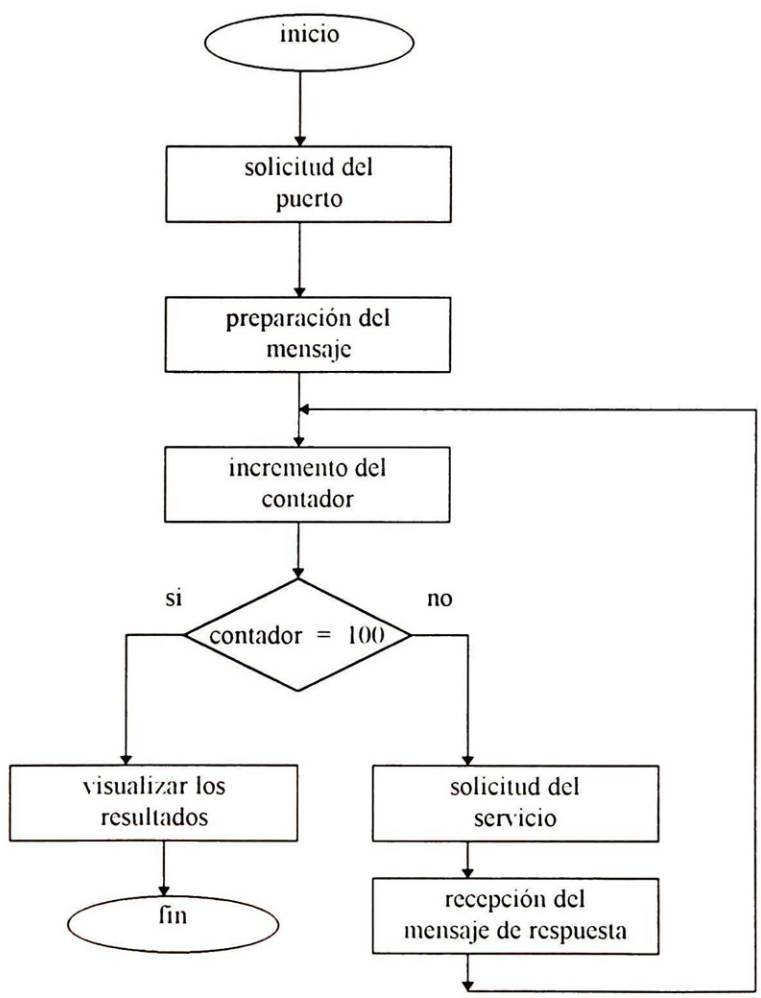
□ invocación para que su ejecución se realice como tarea de fondo:

servidor &

Cliente

Este programa hace solicitudes al servidor creado. Inicialmente verifica si el servicio ha sido registrado y si lo está invoca sus servicios 100 veces para calcular el tiempo promedio empleado para la realización de la comunicación interproceso.

El diagrama en bloques del programa se muestra a continuación:



La compilación del programa debe realizarse de la misma forma que el programa servidor.

El código fuente de los programas del cliente y el servidor se muestran en los apéndices C.4 y C.5 respectivamente.

Para obtener una información detallada de todas las llamadas al sistema del micronúcleo Mach 3.0, puede consultarse la página de Internet:
http://www.osf.org/RI/MC/AD3/KI_REF/HTML.

CONCLUSIONES

Este proyecto de tesis se enmarca dentro del área de sistemas operativos distribuidos y consistió en la implantación de un servidor Linux sobre el micronúcleo Mach. El tema tiene gran relevancia ya que en los últimos años el diseño de los sistemas operativos distribuidos se ha movido hacía la variante de los micronúcleos, lo que conlleva a una migración de funciones fuera del núcleo y las funciones que el núcleo no brinda son implantadas como servicios.

En la actualidad son muchas y variadas las investigaciones que se están desarrollando, y que utilizan ampliamente al sistema operativo Mach para su construcción. En el ámbito académico, el desarrollo de servidores para el micronúcleo Mach ha sido una actividad de gran interés tanto desde el punto de vista de investigación como comercial. A este momento se han desarrollado varios intentos de construcción de servidores de funcionalidad UNIX, por ser una interfaz conocida y con amplia facilidad de uso para ejecutar aplicaciones. Algunos de ellos son centralizados, con funcionalidad similar a UNIX, tal como Lites y otros distribuidos como Sprite[1] y Guide[2]

El micronúcleo Mach 3.0 actualmente corre entre otras, en las siguientes plataformas: Vax, Sun 3, Intel 386 (superiores y compatibles). En este entorno algunas líneas que se destacan por su impacto e importancia son:

- Mejoras al micronúcleo Mach. Ya que aunque Mach es un sistema operativo moderno, con abstracciones muy poderosas y diseñado para incorporar muchas de las innovaciones recientes en el campo de los sistemas distribuidos, con el tiempo ha sufrido del mismo mal que la mayoría de sus similares: ha crecido demasiado y de forma poco ordenada. Actualmente investigadores de la Universidad de Utah

[1] ftp://ftp.cs.berkeley.edu/ucb/sprite/sprite.html
[2] http://www-bi.imag.fr/GUIDE/presguide.html

están trabajando en Mach 4 y OSF trabaja sobre un sistema llamado MK6, dos versiones diferentes de Mach.

- Desarrollos e implantación de sistemas en este paradigma, destacando:
  - la implantación de servidores Linux para diferentes plataformas entre ellas Intel y Macintosh, en esta última con una amplia aceptación y mayor información disponible.
  - implementación de una memoria compartida sobre Mach.
  - diseño y construcción de un servidor DOS sobre el micronúcleo de Mach.

- Trabajos relacionados con la migración y tolerancia a fallas de estos sistemas en diferentes plataformas.

Tampoco debe despreciarse otros resultados obtenidos, de desarrollos de servidores con funcionalidad UNIX sobre Mach, y que emergen con mucha fuerza dentro de la comunidad científica, como es el caso del servidor GNU Hurd y que es importante evaluar. Además están apareciendo otras alternativas, como el micronúcleo Gneiss[3] que está basado en un lenguaje conocido como Oberon2.

El servidor Mklinux ha reportado problemas con su instalación y uso para plataformas Intel. Aunque con los fuentes del sistema se brinda una pequeña documentación para su instalación, ésta es incompleta y no describe aspectos esenciales para que el sistema sea utilizado eficientemente.

Como parte del proyecto de tesis se realizaron varios intentos para instalar el sistema Mklinux. Al respecto se pueden resaltar algunos aspectos importantes derivados de esta experiencia:

[3] http://umfi.cs.sun.ac.za/

1) Siguiendo la metodología de implantación, el servidor pudo ser implantado en algunas computadoras satisfactoriamente.

2) Desafortunadamente, en otras máquinas no pudo ser posible llevar a cabo el proceso de implantación, debido a que el *hardware* de las computadoras no lo soportaba y se presentaron problemas con los requerimientos mínimos de memoria RAM para que el sistema pudiera ser soportado.

3) Aunque la metodología propuesta y distribuida por los diseñadores, no contempla explícitamente el paso *build -here mach_services/lib/libnetname*, tal como se describe en el apéndice A para la construcción del servidor de nombres, éste es necesario que se realice para obtener la biblioteca que el servidor de nombre utiliza y que los programas que invoquen sus llamadas requiere.

4) El proceso de construcción, de algunas partes del sistema, puede fallar si no se utilizan las herramientas de ODE, ante la presencia de alguna falla es necesario comprobar que se esté invocando el comando **make** adecuado.

Transportar hacia el micronúcleo Mach 3.0 los programas ejemplos desarrollados para Mach 2.5 no ha sido un proceso lineal, ni mera copia; ha conllevado un estudio minucioso de las llamadas ofrecidas por este micronúcleo. Además es necesario resaltar que son necesarias algunas consideraciones en la programación de aplicaciones, que la documentación no señala explícitamente, para el empleo de las llamadas al micronúcleo y que se ejecuten en el servidor Mklinux.

No todos los programas pueden ejecutarse sin volverse a compilar y adecuar los parámetros de las funciones para asegurar su correcto funcionamiento. El más grave problema enfrentado es la falta de documentación la cual es limitada y está dispersa, y de ejemplos que muestren la forma de utilizar estas llamadas.

Este proyecto no pretende dar una guía de usuario de las llamadas al micronúcleo Mach 3.0, sino brindar una detallada explicación y ejemplos de la utilización de las abstracciones básicas y las funciones que ofrece el sistema operativo Mach para el manejo de hilos, tareas y comunicación interproceso, utilizando como plataforma el sistema Mklinux instalado.

Un aspecto muy importante y que debe ser tenido en cuenta por todos los programadores que sobre Mklinux desean realizar sus aplicaciones es el hecho de la inclusión de la llamada a la función **mach_init()** al inicio del programa. La función **mach_init()** debe ser incluida en todos los programas que sobre Mklinux se realicen e invoquen a llamadas del núcleo de Mach. Si la función no es incluida, es muy probable que no obtenga errores de compilación, pero si en tiempo de ejecución en la invocación de la primera llamada de una función al micronúcleo. El error señalado es: **(ipc/send) invalid destination port**.

Las llamadas al sistema del micronúcleo Mach 3.0, entre las estudiadas y empleadas, que más cambio muestran en relación a Mach 2.5 son las relacionadas al manejo de la comunicación interproceso. En Mach 2.5 se tenían dos llamadas independientes y el micronúcleo Mach 3.0 utiliza solamente una, para realizar las funciones de envío y recepción de los mensajes. Todos los programas descritos funcionan correctamente y sus códigos fuentes se entregan.

Este no es un trabajo finalizado, en esta misma línea quedan abiertas muchas interrogantes:

- extensión de la comunicación interproceso a varias máquinas.
- programación con c-threads.
- utilización de este sistema como base para el desarrollo de otras aplicaciones distribuidas.

aumento del rendimiento del sistema.

por sólo citar algunas que serán investigadas y continuadas.

La presentación del documento de tesis está realizada en cuatro capítulos, en el primero se hace una descripción de los principales aspectos de los sistemas operativos distribuidos,

haciendo énfasis en la variante que se basa en micronúcleos. En el segundo capítulo se presentan las abstracciones del micronúcleo Mach y se describen algunos proyectos de servidores con funcionalidad UNIX que se han desarrollado tomando como base el micronúcleo Mach.

En el capítulo tres se hace una descripción del sistema operativo Linux y se describe al servidor Mklinux, un servidor de funcionalidad Linux sobre el micronúcleo Mach 3.0 desarrollado por OSF El capítulo cuatro se describen las llamadas al micronúcleo Mach 3.0 para el manejo de hilos, tareas y comunicación interproceso.

Los apéndices muestran los requisitos mínimos que son necesarios para instalar el sistema operativo Linux, la documentación detallada de la metodología de implantación del sistema Mklinux y los códigos fuentes de los programas realizados.

BIBLIOGRAFÍA

[1] J.Boykin, D. Kirschen, A. Langerman, S. Loverso. Programming under Mach. Addison-Wesley Publishing Company. 1993.

[2] J. Helander. Unix under Mach. The Lites Server. Master´s thesis. Helsinki University of Technology. Faculty of Information Technology. Departament of Computer Science. Helsinki. 1994.

[3] A. S. Tanenbaum. Sistemas Operativos Distribuidos. Prentice Hall. 1996.

[4] G. Colouris, J. Dollimore, T. Kindberg. Distributed Systems: Concepts and Design. Addison-Wesley, Second Edition. 1994.

[5] J. Tackett. Linux Edición Especial. Prentice-Hall Hispanoamericana. 1996.

[6] V. J. Blanco. Linux Instalación, administración y uso del sistema. RA-MA. 1996.

[7] F. Barbou des Places, N. Stephen, F. D. Reynolds. Linux on the OSF Mach 3.0 microkernel. Technical report. OSF Research Institute, Grenoble and Cambridge.

[8] A. Silberschatz, J. Peterson, P Galvin. Sistemas Operativos, Conceptos Fundamentales. Addison-Wesley, Tercera Edición. 1994.

[9] OSF Development Environment. User´s Guide. ODE Release 2.3.4 A. Open Software Founfation.

[10] J. E. Ball, J. A. Feldman, J. R. Low, R. F. Rashid, P D. Rovner. Rig, rochester´s intelligent gateway: system overview. IEEE Transaction on Software Engineering. 1976.

[11] K. A. Lantz, K. D. Gradischning, J. A. Felman, R. F. Rashid. Rochester's Intelligent Gateway. Computer. 1982.

[12] R. F Rashid, G. Robertson. Accent: A communication oriented network operating system kernel. In Proc. 8th Symposium on Operating Systems Principles, pp 64-67. ACM. 1981.

[13] A. S. Tanenbaum, R. Van Renesse. Distributed Operating Systems. Computing Surveys. Volume 17 Number 4, pp 419-470. ACM. December 1985.

[14] R. Card. Masix - Un Systéme d'Exploitation Multi-Environnements utilisant le micro-noyau Mach - Conception et Réalisation. Thése de Doctorat. Mai 1993.

[15] R. F Rashid. Mach: A Case Study in Technology Transfer. CMU Computer Science: 25th Anniversary Commerorative. Edited by Richard F. Rashid. Carnegie Mellon University. ACM Press Anthology Series. Addison-Wesley Publishing Company.

[16] F M. Márquez. UNIX Programación Avanzada. Addison-Wesley Iberoamericana. 1994.

[17] K. Loepere. Mach 3 Kernel Principles. Open Software Foundation and Carnegie Mellon University.

[18] A. S. Tanenbaum. Sistemas Operativos. Diseño e Implantación. Prentice-Hall Hispanoamericana. 1988.

[19] A. S. Tanenbaum. Redes de Ordenadores. Segunda Edición. Prentice-Hall Hispanoamericana. 1991.

APÉNDICES.

A: Requerimientos mínimos para la instalación de LINUX

Linux [6] soporta en la actualidad sistemas basados en procesadores Intel 80386, 80486 o superiores. Eso incluye todas las variantes sobre esta CPU, tales como 386SX, 486SX, 486DX, 486DX2 y 486DX4. El procesador Pentium de Intel y procesadores de otras marcas, como AMD y Cyrix, también están soportados.

Una característica fundamental de Linux es la poca memoria que requiere en comparación con otros sistemas operativos de 32 bits de iguales características. Si necesita utilizar X Windows, serán indispensables 8 Mb para el correcto funcionamiento.

En teoría, Linux no es muy exigente con el tipo de disco duro. Cualquier disco soportado por la BIOS de su PC es adecuado. Linux soporta interfaces para discos IDE, EIDE y SCSI y códigos de grabación MFM y RLL si la interfaz es SISOG. Para una instalación completa de Linux, necesitará al menos 257 Mb en su disco duro, pero podrá realizar la instalación básica con un mínimo de 100 Mb.

En modo texto, Linux acepta cualquier adaptador de video: MDA/Hercules, CGA, EGA, VGA y SVGA. En modo gráfico, necesitará una tarjeta de video compatible con XFree86 para ejecutar el sistema X Window.

Linux puede ser ejecutado en la mayoría de las arquitecturas actuales de bus, como ISA y EISA. La arquitectura Micro Channel (MCA) no se soporta en modelos IBM PS/2.

Las siguientes tarjetas de red están soportadas por Linux:

- 3Com, 3C503, 3C505/16.
- Novell EN1000, EN2000.
- Western Digital WD8003, WD8013.
- Hewlett Packard HP27245, HP27247, HP27250.

- D-Link DE-600

La tabla A.1 muestra los requerimientos necesarios para la instalación de *Software Slackware*[1] LINUX, con su código fuente, versiones del núcleo 1.2.13, editores y herramientas de desarrollo, programas de comunicación y muchos archivos de todo tipo.

Tabla A.1 Configuración de los componentes de la computadora.

Componente	Básico	Recomendado
Procesador	Intel 80386 o equivalente	Intel 80486 Pentium Intel AMD o Cyrix
RAM	4 MB	16 MB
Tarjeta gráfica	VGA	SVGA soportada por Xfree86
Disco Duro	100 MB	275 MB
Bus	Modelo soportado	Modelo Soportado
Ratón (*mouse*)	Microsoft o compatible	Microsoft o compatible
Tarjeta de red	Ninguna	Modelo soportado

[1] Slackware es una marca registrada de Patrick Volkerding y Walnut Creek CDROM.

Archivos fuentes del sistema y proceso de instalación

Los archivos fuentes, necesarios para la instalación del sistema Mklinux para plataformas Intel puede obtenerse vía Internet[2] o *ftp anonymous*, desde el directorio **http://www.osf.org/mall/OS/SW_mklinux-1.0b2/** que contiene la segunda versión del sistema.

Este directorio contiene los siguientes archivos:

- mklinux-10b2-bin.tgz: Incluye los archivos binarios y archivos de configuración necesarios para instalar el sistema. Se recomienda desempaquetar este archivo en el directorio raíz y seguir las instrucciones orientadas en el archivo MKLINUX_BOOT.README.

- mklinux-10b2-bin-extra.tgz: Incluye otros archivos binarios y también debe desempaquetarse en el directorio raíz.

- mklinux-10b2-mklinux.tgz: Incluye los archivos fuentes del servidor Linux. Se recomienda desempaquetar en el directorio **/usr/src**[3] y seguir las instrucciones suministradas en los archivos MKLINUX_BUILD.README. Los archivos fuentes y herramientas suministradas deben ser desempaquetadas en el mismo subdirectorio. Si los archivos no son desempaquetados en el directorio **/usr/src** se debe sustituir la cadena "**/usr/src** " con el camino correcto, que identifique de manera correcta el lugar donde han sido ubicados los archivos.

[2] http://www.osf.org/mall/OS
[3] Esta recomendación es una buena opción, pero no obligatoria.

- mklinux-10b2-osfmk.tgz: Incluye los archivos fuentes del micronúcleo de OSF Se deben desempaquetar en el directorio **/usr/src** y seguir las instrucciones dadas en el archivo OSFMK_BUILD.README.

- mklinux-10b2-tools.tgz: Incluye la herramienta ODE[4], ambiente de desarrollo de OSF.

Proceso de instalación del micronúcleo Mach y del servidor Linux en una máquina Linux

Para instalar el micronúcleo Mach y el servidor Linux, en una máquina Linux, es necesario seguir las siguientes pasos:

1.- Desempaquetar el archivo **mklinux-10b2-bin.tgz** en el nodo raíz del sistema de archivo[5]

El sistema de archivos de UNIX (o de similar funcionalidad a él, como Linux), está organizado a nivel lógico en forma de árbol invertido, con un nodo principal conocido como nodo raíz ("/"). Cada nodo dentro del árbol es un directorio y puede contener a su vez a otros nodos (subdirectorios), archivos normales y archivos de dispositivos.

Los nombres de los archivos describen como localizar un archivo dentro de la jerarquía del sistema, puede ser absoluto (referido al nodo raíz) o relativo (referido al directorio de trabajo actual[6]).

Ejecute:

```
# cd /
```

[4] *OSF Development Environment*
[5] Deben tenerse derechos de superusuario
[6] *CWD Current Work Directory*

```
# tar -xvzf mklinux-10b2-bin.tgz
```

2.- Actualizar el archivo **/mach_boot.env** para señalar la partición raíz de Linux.

Se deben usar los nombres de dispositivos de Mach.
Linux utiliza el estilo Minix para los dispositivos de discos (hda1, hda2, ..., hdb1, ... para los discos IDE o sda1, sda2, ..., sdb1, ... para los discos SCSI).

Mach usa el estilo BSD para los nombres de los dispositivos (hd0a, hd0b, ... , hd1a, ..., para los discos IDE o sd0a, sd0b, ..., sd1a, ... para los discos SCSI.

La tabla B.1 ilustra los nombres para dispositivos Mach y Linux.

En nuestro caso la partición raíz "**sda1**" necesita ser escrita como "**sd(0,a)**"

Tabla B.1 Nombre de dispositivos en los sistemas operativos Mach y Linux

	Linux	Mach
1ra. partición del 1er. disco IDE	hda1	hd0a
2da. partición del 1er. disco IDE	hda2	hd0b
3ra. partición del 1er. disco IDE	hda3	hd0c
4ta. partición del 1er. disco IDE	hda4	hd0d
1ra. partición del 2do. disco IDE	hdb1	hd1a
2da. partición del 2do. disco IDE	hdb2	hd1b
3ra. partición del 2do disco IDE	hdb3	hd1c
4ta. partición del 2do. disco IDE	hdb4	hd1d
1ra. partición del 1er. disco SCSI	sda1	sd0a
2da. partición del 1er. disco SCSI	sda2	sd0b
3ra. partición del 1er. disco SCSI	sda3	sd0c
4ta. partición del 1er. disco SCSI	sda4	sd0d
1ra. partición del 2do. disco SCSI	sdb1	sd1a
2da. partición del 2do. disco SCSI	sdb2	ad1b
3ra. partición del 2do. disco SCSI	sdb3	sd1c
4ta. partición del 2do. disco SCSI	sdb4	sd1d

También se puede seleccionar la configuración del micronúcleo que se desea cargar. Este puede ser uno de las siguientes opciones:

- mach_kernel.DEBUG: Micronúcleo para monoprocesadores con inclusión de opciones para la depuración, pero de un tamaño mucho mayor y lento que la configuración del micronúcleo "FAST"
- mach_kernel.DEBUG+MP: Micronúcleo para multiprocesadores Intel MPv1 con opciones de depuración.

- mach_kernel.FAST+MP: Micronúcleo para procesadores Intel MPv1.

- mach_kernel.FAST: Micronúcleo optimizado, para sistemas monoprocesadores y cargado por omisión en la inicialización.

3.- Actualizar el archivo **/mach_servers/bootstrap.conf**

a) Si es necesario seleccionar que servidor Linux se desea iniciar, actualizar la información correspondiente en la línea que comienza con "**startup**"

b) Mencionar la partición raíz de Linux, en el formato Linux "dev/sda1", después de "**root=**" en la línea que comienza con "**startup**"

4.- Actualizar el archivo **/etc/lilo.conf**

Adicione el contenido del archivo **/mach_servers/lilo.conf.add-on** al archivo **/etc/lilo.conf**, después editar en él los cambios de la partición raíz.

Ejecute:

```
# cat /mach_servers/ >> /etc/lilo.conf
```

Ejecute **lilo** para considerar los cambios efectuados:

```
# /sbin/lilo
```

La salida debe mencionar "**mklinux**" como respuesta de que el proceso funcionó correctamente. Si no ocurre así, algo se efectuó mal; es necesario verificar nuevamente el archivo /etc/lilo.conf

5.- Reinicie el sistema.

De preferencia, y por la propia seguridad del sistema, se debe ejecutar la secuencia:

- # logout
- oprimir las teclas:

Ctrl **Alt** **Del**

simultáneamente.

6.- En el indicador "**lilo**", teclear mklinux.

7.- En el indicador "**boot:**", indicar la partición.

Como ejemplo, si la partición raíz de Linux es hda1 y el la variante del núcleo a cargar es la "FAST", teclee:

```
boot:hd(0,a)/mach_kernel.FAST
```

Una vez realizado esto, Mach debe comenzar la inicialización, cargando el servidor Linux e inicializando el sistema.

Conceptos básicos de ODE

El ambiente de desarrollo de OSF (ODE) [9], está diseñado para permitir desarrollos simultáneos, que pueden ser compilados en una variedad de diferentes y esencialmente incompatibles plataformas de *hardware*. El mismo es un conjunto de herramientas, las cuales soportan áreas de trabajo individual y en grupos, control de los archivos fuentes y compilación en ambientes distribuidos.

Sandbox

El ambiente de desarrollo básico soportado por ODE (OSF Developed Environment) es el sandbox. Los *sandbox* son áreas de trabajo individuales y privadas, que disponen y controlan los usuarios para desarrollar y probar los programas, sin interferir entre ellos, aún cuando esten utilizando los mismos archivos.

Componentes de un *sandbox*

Hay tres componentes esenciales de un *sandbox* de ODE: el área de edición , el área para construir los módulos objetos y el área para mantener los archivos de encabezados y las bibliotecas.

Los archivos fuentes son ubicados en el directorio **src**, y los archivos objetos en el directorio **obj**, sin que el usuario deba especificar los diferentes directorios, éstos son separados automáticamente por las herramientas. Los componentes de un sistema que son usados para construir otros sistemas deben ser colocados en el subdirectorio **export**.

Archivo sandboxrc

Cada usuario que trabaje con ODE, tiene un archivo que contiene la información sobre el *sandbox* que él utilice. Esta información incluye: lista de los *sandboxes* del usuario, directorio base de cada *sandbox*, y el *sandbox* por omisión de cada usuario. El archivo es usualmente localizado en **${HOME}/.sandboxrc**. Si la ubicación del archivo no es en éste subdirectorio, el usuario deberá suministrar la ruta para localizar al mismo.

La información contendida en el archivo **.sandboxrc**, para los comandos de ODE tienen el siguiente formato:

nombre_de_comando opciones

Operaciones con los *sandbox*

Las operaciones con los *sandbox* soportados por ODE incluyen entre otras las opciones de crear y destruir los *sandbox*.

El comando para crear un *sandbox* es **mksb**. Este comando es el primer paso en la colocación de un ambiente de desarrollo bajo ODE. **Mksb** crea la estructura del *sandbox*, la cual incluye los subdirectorios **src, obj, export y rc_file**.

Proceso de compilación

El comando para compilar programas con ODE es **build**. El proceso de compilación ODE (*build*) está basado en el comando UNIX **make**. ODE utiliza una versión mejorada de éste comando, la cual tiene funcionalidad para apoyar el desarrollo de la compilación.

La diferencia entre **build** y **make** está en que **build** puede operar fuera del alcance de un *sandbox*, **make** requiere estar dentro de un *sandbox* antes de ejecutarse. **Build** primero busca en el directorio **src** del *sandbox* y una vez localizadas las referencias proporciona al compilador la localización de las bibliotecas y los archivos de cabeceras.

Los siguientes elementos son fundamentales de la compilación en ODE:

- **build** y **make** son utilizados en todas las compilaciones.
- Los archivos fuentes y objetos son localizados en directorios separados.
- Las reglas de compilación, frecuentemente utilizadas son agrupadas en un archivo común.

Compilación del micronúcleo

Una vez realizada la instalación del sistema es necesario llevar a cabo la instalación de algunas partes adicionales. A continuación se describen los pasos necesarios para la compilación del micronúcleo.

1.- Inclusión de las herramientas de Mach y de ODE en su PATH.

Se necesitaran las herramientas ODE para la ejecución del proceso de instalación del sistema Mklinux. En el subdirectorio: **/usr/src/mklinux-1.0b2/ode-bin/** se encuentran todos los archivos binarios, que son necesarios. Cuando se construye el micronúcleo se debe usar el comando "***make***" de ODE, si ocurren problemas con la sintaxis de Makefile, probablemente se está ejecutando erróneamente el comando "***make***", verifique que se han incluidos en la variable de ambiente PATH los caminos para localizar las herramientas correctas.

El archivo **set_ode_path.sh** actualizará automáticamente la variable PATH con el valor correcto.

Ejecute:

```
# . /usr/src/mklinux-1.0b2/set_ode_path.sh
```

2.- Editar e instalar el archivo **~/.sandboxrc**.

Si nunca se ha utilizado el ambiente de desarrollo de OSF, utilice el archivo **sandboxrc.template**. Instale este archivo como **~/.sandboxrc** en el directorio HOME.

Ejecute:

```
# cp /usr/src/sandboxrc.template ~/.sandboxrc
```

3.- Adicionar la variable USER al ambiente.

Ejecute:

```
# export USER = $LOGNAME
```

4.- Activar el sandbox del micronúcleo de OSF

Ejecute:

```
# workon -sb osfmk
```

Como resultado de ello debe aparecer:

coding to sandbox source directory: /usr/src/mklinux-1.0b2/osfmk/src

project: osc

starting new shell: /bin/bash

#

Este comando debe iniciar un nuevo shell, y el directorio actual debe ser ahora: **/usr/src/mklinux-1.0b2/osfmk/src**.

5.- Construir los archivos cabeceras de OSF MK.

Ejecutar:

```
# build MAKEFILE_PAST=FIRST
```

6.- Construir los archivos binarios de OSF MK.

Ejecutar:

```
# build -here mach_services/lib/libcthreads
# build -here mach_services/lib/libsa_mach
# build -here mach_services/lib/libmach
# build -here mach_services/lib/libmach_maxonstack
```

7.- Otros aspectos que deben construirse son:

- para la biblioteca de archivos del sistema[7] ejecutar:

```
# build -here file_system
```

[7] *File System Library*

- para el servidor de nombres ejecutar:

```
# build -here mach_services/lib/libservice
# build -here mach_services/servers/netname
# build -here mach_services/lib/libnetname
```

Inicio de una sección de trabajo

Para iniciar una sección de trabajo con el sistema Mklinux, en una máquina Linux, se deben realizar los siguientes pasos:

1.- Reinicializar el sistema.

2.- En el indicador "**lilo**", teclear: mklinux.

3.- En el indicador "**boot**", teclear: hd(0,a)/mach_kernel.FAST o la indicación correcta para suministrar la partición raíz de Linux y el núcleo que se desea iniciar.

4.- Adicionar la variable USER al ambiente.

5.- Activar el sandbox del micronúcleo de OSF

C: Códigos fuente de los programas.

C.1 Multiplicación de matrices con un solo hilo de ejecución

```
;archivos de cabeceras utilizados.
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <mach.h>

;definiciones empleadas en el programa para el orden de la matriz.
#define N 3
#define P 4
#define M 5

int mat_1 [N][P] = {{1,2,3,4},{5,6,7,8},{9,0,1,2}};
int mat_2 [P][M] = {{3,4,5,6,7},{8,9,0,1,2},{3,4,5,6,7},{8,9,0,1,2}};
int result [N][M];

main()
{
  matrix_multiply(); /* multiplica las matrices */
  matrix_print(); /* visualiza los resultados */
  exit(0);
}

matrix_multiply()
{
/* multiplica las matrices mat_1 y mat_2 */
int i,j,l;

  for (i=0; i<N; i++) {
    for (j=0; j<M; j++) {
      result[i][j]=0;
      for (l=0; l<P; l++) {
        result[i][j]+=mat_1[i][l]*mat_2[l][j];
      }
    }
  }
}

matrix_print()
{
int i,j;

  printf("mat_1:\n"); /* visualiza matriz mat_1 */
  for (i=0; i<N; i++) {
    for (j=0; j<P; j++)
      printf("%4d  ", mat_1[i][j]);
    putchar('\n');
  }
  printf("mat_2:\n"); /* visualiza matriz mat_2 */
  for (i=0; i<P; i++) {
    for (j=0; j<M; j++)
```

```
            printf("%4d   ", mat_2[i][j]);
        putchar('\n');
    }
    printf("result:\n"); /* visualiza matriz resultante */
    for (i=0; i<N; i++) {
        for (j=0; j<M; j++)
            printf("%4d   ", result[i][j]);
        putchar('\n');
    }
}
```

C.2 Multiplicación de matrices utilizando múltiples hilos de ejecución

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <mach.h>
#include <mach/mach_traps.h>

#define N 3
#define P 4
#define M 5
#define THREAD_NOT_DONE 0
#define THREAD_DONE 1
#define STACKSIZE 8192
#define NTHREADS 8

int mat_1[N][P]={{1,2,3,4},{5,6,7,8},{9,0,1,2}};
int mat_2[P][M]={{3,4,5,6,7},{8,9,0,1,2},{3,4,5,6,7},{8,9,0,1,2}};
int result[N][M];
int threads[NTHREADS];
char stacks[NTHREADS][STACKSIZE];

main()
{

#ifdef linux
  mach_init();
#endif


  matrix_multiply(); /* prepara para la multiplicación de las matrices */
  matrix_print();
  exit(0);
}

matrix_multiply()
/* crea e inicia NTHREADS hilos para multiplicar las matrices mat_1 y mat_2 */
{
int thread_num;

  for (thread_num=0; thread_num<NTHREADS; thread_num++) {
    threads[thread_num]=THREAD_NOT_DONE;
    thread_fork(thread_num);
  }
  wait_for_threads();
}

thread_fork(int thread_num)
/* crea, actualiza la información de estado e inicializa el hilo */
{
kern_return_t status;
thread_port_t th;

  status=thread_create(mach_task_self(), &th);
  if (status!=KERN_SUCCESS) {
```

```
    mach_error("thread_create: ", status);
    exit(1);
  }
  setup_thread(th, thread_num);
  status=thread_resume(th);
  if (status!=KERN_SUCCESS) {
    mach_error("thread_resume: ", status);
    exit(2);
  }
}

compute_element(int i,int j)
/* calcula un elemento de la matriz resultante */
{
int l, element;

  element=0;
  for (l=0; l<P; l++)
    element+=mat_1[i][j]*mat_2[l][j];
  return(element);
}

thread_multiply(int thread_num)
{
kern_return_t status;
int i,j;

  for (i=0; i<N; i++)
    for (j=thread_num; j<M; j+=NTHREADS)
      result[i][j]=compute_element(i,j);
  threads[thread_num]=THREAD_DONE;
  status=thread_terminate(mach_thread_self());
  if (status!=KERN_SUCCESS)
    mach_error("thread_terminate: ", status);
}

setup_thread(mach_port_t th, int arg)
{
char *stack;
register int *top;
struct i386_thread_state state, *ts;
kern_return_t status;

  stack=(char *) &stacks[arg]; /* inicializa los punteros de inicio y fin para la pila */
  top=(int *) (stack+STACKSIZE-sizeof(void));
  bzero( (char *) &state, sizeof(state));
  ts=(struct i386_thread_state *) &state;
  ts->eip=(int)(int(*)())thread_multiply; /* actualiza la dirección de inicio de la rutina que ejecutará
                                             el  hilo */
  *--top=(int) arg;
  *--top=0;
  ts->uesp=(int) top;
  status=thread_set_state(th, i386_THREAD_STATE, (thread_state_t) &state,
                    i386_THREAD_STATE_COUNT);
/* notifica al núcleo el estado del nuevo hilo creado */
  if (status!=KERN_SUCCESS)
```

```
    mach_error("setup_thread: ", status);
}

wait_for_threads()
{
int thread_num;

  for (thread_num=0; thread_num<NTHREADS; thread_num++)
    while (threads[thread_num]==THREAD_NOT_DONE)
      sleep(1);
}


matrix_print() /* visualiza la matriz resultante */
{
int i, j;

  printf("mat_1:\n");
  for (i=0; i<N; i++) {
    for (j=0; j<P; j++)
      printf("%4d  ", mat_1[i][j]);
    putchar('\n');
  }
  printf("mat_2:\n");
  for (i=0; i<P; i++) {
    for (j=0; j<M; j++)
      printf("%4d  ", mat_2[i][j]);
    putchar('\n');
  }
  printf("result:\n");
  for (i=0; i<N; i++) {
    for (j=0; j<M; j++)
      printf("%4d  ", result[i][j]);
    putchar('\n');
  }
}
```

C.3 Información sobre hilos

Código fuente del programa pthinfo.c

```
#include <mach.h>
#include <stdio.h>

main () {
 kern_return_t        status;
 thread_basic_info_t   the_info;
 unsigned int         info_count;
 thread_t             th;


  the_info=(thread_basic_info_t) malloc(THREAD_BASIC_INFO_COUNT);
/* reserva espacio de memoria para  almacenar la información sobre el hilo */
  if (the_info == NULL) {
    perror("th_info");
    exit(1);
    }
  info_count=THREAD_BASIC_INFO_COUNT;
  status=thread_info(thread_self(), THREAD_BASIC_INFO, (thread_info_t) the_info, &info_count);
/* solicitud de información al núcleo */
  if (status != KERN_SUCCESS) {
    mach_error("thread_info: ", status);
    exit(2);
    }
  printf("For thread_self():\n");
/* visualización de los resultados */
  printf("\t    user time: %3d second  %3d microseconds\n", the_info->user_time.seconds,
                                          the_info->user_time.microseconds);
  printf("\t suspend count: %3d\n", the_info->suspend_count);
  status=thread_create(task_self(), &th);
  if (status != KERN_SUCCESS) {
    mach_error("thread_create: ", status);
    exit(3);
    }
  status=thread_info(th, THREAD_BASIC_INFO, (thread_info_t) the_info, &info_count);
  if (status != KERN_SUCCESS) {
    mach_error("thread_info: ", status);
    exit(4);
    }
  printf("For a new thread: \n");
  printf("\t    user time: %3d second  %3d microseconds\n", the_info->user_time.seconds,
                                          the_info->user_time.microseconds);
  printf("\t suspend count: %3d\n", the_info->suspend_count);
  exit(0);
}
```

C.4 Comunicación Interproceso
Servidor

```
#include <mach.h>
#include <mach/msg_type.h>
#include <mach_error.h>
#include <mach/message.h>
#include <servers/netname.h>
#include <stdio.h>
#include <strings.h>
#include <sys/time.h>
#include <sys/resource.h>

main ()
{
char *servicename="Nada";
mach_port_t port, pset;
kern_return_t kr;
struct {
  mach_msg_header_t h;
  mach_msg_format_0_trailer_t t;
  } msg;

  mach_init();
/* cración de un puerto */
  kr=mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_RECEIVE, &port);
  if (kr!=KERN_SUCCESS) {
    fprintf(stderr, "servidor: machipc -> mach_port_allocate: %s \n",
        mach_error_string(kr));
    exit(1);
    }
/* registro del servicio */
  kr=netname_check_in(name_server_port, servicename, mach_task_self(), port);
  if (kr!=KERN_SUCCESS) {
    fprintf(stderr, "servidor: netname -> netname_check_in: %s \n" ,
        mach_error_string(kr));
    exit(1);
    }
/* creación de un conjunto de puerto */
  kr=mach_port_allocate(mach_task_self(), MACH_PORT_RIGHT_PORT_SET, &pset);
  if (kr!=KERN_SUCCESS) {
    fprintf(stderr, "servidor: machipc -> mach_port allocate(pset): %s \n",
        mach_error_string(kr));
    exit(1);
    }
  kr=mach_port_move_member(mach_task_self(), port, pset);
  if (kr!=KERN_SUCCESS) {
    fprintf(stderr, "servidor: machipc -> mach_port_move_member: %s \n",
        mach_error_string(kr));
    exit(1);
    }
  kr=mach_msg_overwrite(&msg.h, MACH_RCV_MSG, 0, sizeof(msg), pset,
                MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL,
                (mach_msg_header_t *) NULL, (mach_msg_size_t) 0);
  if (kr!=MACH_MSG_SUCCESS) {
```

```
        fprintf(stderr, "servidor: machipc -> mach_msg_overwrite: %s \n",
                mach_error_string(kr));
        exit(1);
        }
    for (;;) {
/* espera solicitud y envia mensaje de respuesta */
        msg.h.msgh_bits=MACH_MSGH_BITS(MACH_MSG_TYPE_MOVE_SEND_ONCE, 0);
        msg.h.msgh_local_port=MACH_PORT_NULL;
        (void) mach_msg_overwrite_trap(&msg.h, MACH_SEND_MSG|MACH_RCV_MSG,
                sizeof(msg.h), sizeof(msg), pset, MACH_MSG_TIMEOUT_NONE,
                MACH_PORT_NULL, (mach_msg_header_t *) NULL,
                (mach_msg_size_t) 0);
        }
}
```

C.5 Comunicación interproceso
Cliente

```
#include <mach.h>
#include <mach/msg_type.h>
#include <mach_error.h>
#include <mach/message.h>
#include <servers/netname.h>
#include <stdio.h>
#include <strings.h>
#include <sys/time.h>
#include <sys/resource.h>

double periter (struct timeval *before, struct timeval *after, int valor)
{
  return (((after->tv_sec - before->tv_sec) * 1000000 +
        (after->tv_usec - before->tv_usec)) / (double) valor);
}

main ()
{
char *servicename="Nada";
mach_port_t port, reply;
kern_return_t kr;
int i;
struct {
  mach_msg_header_t h;
  mach_msg_format_0_trailer_t t;
  } msg;
struct rusage rbefore, rafter;
struct timeval tbefore, tafter;

  mach_init();
  kr=netname_look_up(name_server_port, "*", servicename, &port);
/* solicitud del puerto por el cual el servidor atenderá las solicitudes */
  if (kr!=KERN_SUCCESS) {
    fprintf(stderr, "cliente: machipc -> netname_look_up: %s \n",
         mach_error_string(kr));
    exit(1);
    }
  reply=mach_reply_port();
/* preparación del encabezado del mensaje */
  msg.h.msgh_id=0;
  msg.h.msgh_local_port=reply;
  msg.h.msgh_remote_port=port;
  msg.h.msgh_bits=MACH_MSGH_BITS(MACH_MSG_TYPE_COPY_SEND,
                                 MACH_MSG_TYPE_MAKE_SEND_ONCE);
  kr=mach_msg_overwrite(&msg.h, MACH_SEND_MSG|MACH_RCV_MSG,
                    sizeof(mach_msg_header_t),
                    sizeof(msg), reply, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL, 0,
                    (mach_msg_size_t) 0);
  if (kr!=KERN_SUCCESS) {
    fprintf(stderr, "cliente: machipc -> mach_msg_overwrite: %s /n",
          mach_error_string(kr));
    exit(1);
```

```
    }
  (void) gettimeofday (&tbefore, (struct timezone *) NULL);
  (void) getrusage(RUSAGE_SELF, &rbefore);
  for (i=0; i<100; i++) {
    msg.h.msgh_bits=MACH_MSGH_BITS(MACH_MSG_TYPE_COPY_SEND,
                MACH_MSG_TYPE_MAKE_SEND_ONCE);
/* solicitud del servicio */
    (void) mach_msg_overwrite_trap(&msg.h, MACH_SEND_MSG|MACH_RCV_MSG,
                sizeof(mach_msg_header_t), sizeof (msg),
                reply, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL, 0,
                (mach_msg_size_t) 0);
    }
  (void) getrusage(RUSAGE_SELF, &rafter);
  (void) gettimeofday(&tafter, (struct timezone *) NULL);
/* visualización de los resultados */
  printf("Tiempo transcurrido en rpc: %7.3f \n",
      periter(&tbefore, &tafter, 100));
  printf(" usuario: %7.3f \n", periter(&rbefore.ru_utime, &rafter.ru_utime, 100));
  printf(" systema: %7.3f \n", periter(&rbefore.ru_stime, &rafter.ru_stime, 100));
}
```

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará **el Ing. Héctor Diez Rodríguez,** declaramos que hemos revisado la tesis titulada:

"Implantación de un servidor Linux sobre Mach", consideramos que cumple con los requisitos para obtener el grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Héctor Ruíz Barradas

Dr. Adriano de Luca Pennacchia

Dra. Elizabeth Pérez Cortés