

SISTEMAS OPERATIVOS

TRABAJO MONOGRÁFICO REALIZADO POR

GRISELDA E. BRESSAN

COMO ADSCRIPTA A LA ASIGNATURA

“SISTEMAS OPERATIVOS”

MAYO – 2003

“EL CASO DE ESTUDIO MACH”

MACH

1. INTRODUCCIÓN

El proyecto Mach estaba asentado en la Universidad Carnegie-Mellon en USA hasta 1994, donde se desarrolló hacia un núcleo en tiempo real. El proyecto fue continuado por grupos de la Universidad de Utah y la Fundación para el Software Abierto (OSF). El proyecto Mach fue el sucesor de otros dos: RIG, desarrollado en la Universidad de Rochester en la década de los años setenta, y Accent, desarrollado en Carnegie-Mellon durante la primera mitad de la década de los años ochenta. A diferencia de sus predecesores, el proyecto Mach nunca tuvo como objetivo el desarrollo de un sistema operativo distribuido completo. En vez de esto, el núcleo Mach se desarrolló para proporcionar una compatibilidad directa con UNIX BSD, y recursos avanzados que complementarían los de UNIX y que permitiera a una implementación UNIX proliferar sobre una red de multiprocesadores y computadoras monoprocesador. Desde el principio, la intención de los diseñadores fue la de implementar la mayor parte de UNIX mediante procesos de nivel de usuario.

A pesar de estas intenciones, la versión 2.5 de Mach, la primera de las dos principales versiones, incluía todo el código de compatibilidad con UNIX dentro del propio núcleo. Se ejecutaba, entre otras computadoras, en SUN-3, IBM RT PC, sistemas uniprocesadores y multiprocesadores VAX y en los multiprocesadores Encore Multimax y Sequent. Desde 1989, Mach 2.5 fue incorporado como la tecnología base para OSF/1, el rival propuesto por la OSF para competir contra System V Release 4 como versión estándar de UNIX para la industria.

A partir de la versión 3.0 del núcleo Mach se eliminó el código UNIX, esta versión es la base de la implementación de MkLinux, una variante del sistema operativo Linux para computadoras Power Macintosh; se utiliza como soporte para la construcción de emulaciones de sistemas operativos, sistemas de base de datos, sistemas de soporte de lenguajes en tiempo de ejecución y otros tipos de software de sistema que llamaremos subsistemas.

Procesos / objetos de aplicación			
UNIX BSD4.3	Base de datos Camelot	OS/2 MkLinux	Soporte de lenguajes orientados a objetos.
Núcleo Mach			
Multiprocesador o monoprocesador			

Figura 1. Mach proporciona soporte a sistemas operativos, bases de datos y otros subsistemas.

La emulación de sistemas operativos convencionales permite ejecutar binarios ya existentes desarrollados para ellos. Además se pueden desarrollar nuevas aplicaciones para dichos sistemas operativos convencionales. Al mismo tiempo, se puede desarrollar middleware y aplicaciones que aprovechen las ventajas de la distribución; y se pueden crear versiones distribuidas de las implementaciones de los sistemas operativos convencionales. Surgen dos cuestiones importantes en la emulación de los sistemas operativos. La primera es que las emulaciones distribuidas no pueden ser completamente exactas debido a los nuevos modos de fallo que aparecen con la distribución. La segunda, es la cuestión todavía sin resolver de si se pueden conseguir niveles de prestaciones aceptables para que su utilización sea generalizada.

1.1. OBJETIVOS Y PRINCIPALES CARACTERÍSTICAS DEL DISEÑO

Los principales objetivos y características de diseño de Mach son los siguientes:

Operación multiprocesador: Mach fue diseñado para ejecutarse en un multiprocesador de memoria compartida de forma que tanto los hilos del núcleo como los hilos en modo usuario pueden ejecutarse en cualquier procesador. Mach proporciona un modelo multi-hilo para procesos de usuario, con entornos de ejecución llamados tareas. Para permitir la ejecución paralela en un multiprocesador de memoria compartida, la planificación de los hilos es prioritaria, o apropiativa, tanto si pertenecen a las mismas como a diferentes tareas.

Extensión transparente para operar en red: Para permitir que los programas distribuidos se extiendan de forma transparente sobre mono y multiprocesadores en una red, Mach ha adoptado un modelo de comunicación independiente de la ubicación usando puertos como destino de la

comunicación. Sin embargo, el núcleo de Mach ha sido diseñado para ser 100% independiente de redes concretas. El diseño de Mach confía en procesos servidores de red de nivel de usuario para el envío de mensajes de forma transparente sobre la red (ver Figura 2). Se trata de una decisión de diseño polémica dados los costos del cambio de contexto. Sin embargo, permite una absoluta flexibilidad en el control de la política de comunicación de la red.

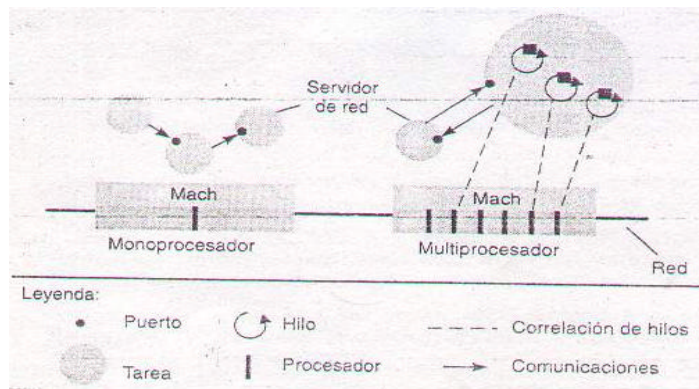


Figura 2. Tareas, hilos y comunicación en Mach.

Servidores de nivel de usuario: Mach implementa un modelo basado en objetos en el que los recursos se gestionan o bien por el núcleo, o bien mediante servidores cargados dinámicamente. Inicialmente solo se permitían servidores de nivel de usuario, pero posteriormente se adaptó Mach para acomodar servidores en el espacio de direcciones del núcleo. Con la excepción de algunos recursos gestionados por el núcleo, los recursos son accedidos de forma uniforme mediante paso de mensajes, independientemente de cómo sean gestionados. A cada recurso le corresponde un puerto gestionado por un servidor. El Generador de Interfaz de Mach (MIG) se desarrolló para generar enlaces RPC utilizados para ocultar los accesos basados en mensajes del nivel de lenguaje.

Emulación de sistema operativo: Para dar soporte a la emulación de nivel de usuario de UNIX y otros sistemas operativos, Mach permite la redirección transparente de las llamadas al sistema operativo sobre llamadas a una biblioteca de emulación y de ahí hacia los servidores del sistema operativo del nivel de usuario; a esta técnica se la conoce como *trampolining*. También se incluye la posibilidad de permitir que las excepciones del tipo de violaciones del espacio de direcciones que ocurren en las tareas de aplicación sean gestionadas por servidores. En www.cdk3.net/oss se pueden encontrar los casos de estudio de emulaciones UNIX bajo Mach y Chorus.

Implementación de memoria virtual flexible: Para dotar a Mach de la posibilidad de emular sistemas UNIX y dar soporte a otros subsistemas se realizó una aproximación flexible a la distribución del espacio de direcciones de un proceso. Mach soporta un espacio de direcciones grande y disperso, con capacidad potencial para contener múltiples regiones. Por ejemplo, tanto los mensajes como los archivos abiertos pueden aparecer como regiones de memoria virtual. Las regiones pueden ser privadas a una tarea, compartidas entre tareas o copiadas desde regiones sobre otras tareas. El diseño incluye el uso de técnicas de correlación de memoria, principalmente la *copia en escritura*, para evitar la copia de datos cuando, por ejemplo, se pasan mensajes entre tareas. Finalmente, Mach se diseñó para permitir que fueran los servidores, en lugar del propio núcleo, los que implementaran el almacenamiento secundario para las páginas de memoria virtual. Las regiones pueden correlacionarse con datos gestionados por los servidores llamados *paginadores externos*. Estos datos correlacionados (*mapped*) pueden residir en cualquier abstracción de un recurso de memoria, desde memoria compartida distribuida hasta archivos.

Portabilidad: Mach fue diseñado para ser portable sobre varias plataformas hardware. Por esta razón, se trató de aislar, tanto como fue posible, el código dependiente de la máquina.

1.2. REPASO DE LAS PRINCIPALES ABSTRACCIONES DEL NÚCLEO DE MACH

Tareas: Es un entorno de ejecución; básicamente está compuesto por un espacio de direcciones protegido y una colección de habilitaciones *gestionadas por el núcleo* que se utilizan para el acceso a los puertos.

Hilos: Las tareas pueden contener múltiples hilos. Los hilos que pertenecen a una misma tarea se pueden ejecutar en paralelo sobre diferentes procesadores en un multiprocesador de memoria compartida.

Puertos: Es un canal de comunicaciones *uno a uno* y unidireccional con una cola de mensajes asociada. Los puertos no son accesibles directamente por el programador de Mach y tampoco forman parte de una tarea. En su lugar, el programador dispone de apuntadores a *derechos de puerto*. Se trata de habilitaciones, para el envío de mensajes a un puerto o para la recepción de mensajes desde un puerto.

Conjuntos de puertos: Es una colección de derechos de recepción de puertos, locales a una cierta tarea. Se utilizan para recibir mensajes desde cualquier puerto de entre una colección de ellos. Los conjuntos de puertos no deben confundirse con los *grupos* de puertos; estos últimos son destinos de multidifusión y en Mach no están implementados.

Mensajes: Puede contener derechos de puerto además de datos. El núcleo utiliza técnicas de gestión de memoria para transferir de forma eficiente los datos de los mensajes entre las tareas.

Dispositivos: Los servidores de archivos que se ejecutan a nivel de usuario deben poder acceder a los dispositivos. Para ello el núcleo exporta una interfaz de bajo nivel sobre los dispositivos.

Objeto de memoria: Cada región del espacio de direcciones virtuales de una tarea Mach se corresponde con un objeto de memoria. Se trata de un objeto que normalmente está implementado fuera del propio núcleo pero que es accedido por éste cuando realiza operaciones de paginación en la gestión de la memoria virtual. Un objeto de memoria es una instancia de un tipo abstracto de datos que incluye operaciones de búsqueda y almacenamiento de datos que se emplean cuando los hilos generan faltas de página en el intento de hacer referencia a direcciones, en la región correspondiente.

Objeto de memoria caché: Para cada objeto correlacionado en memoria existe un objeto gestionado por el núcleo que contiene una caché de páginas residentes en la memoria principal para la correspondiente región. A esto se le llama objeto de memoria caché. Da soporte a las operaciones necesarias para el paginador externo encargado de implementar el objeto de memoria.

2. PUERTOS, NOMBRES Y PROTECCIÓN

Mach asocia recursos individuales con puertos. Para acceder a un recurso se envía un mensaje sobre el puerto correspondiente. En Mach, se presupone que los servidores manejarán, por lo general, múltiples puertos: uno por cada recurso. Un único servidor UNIX utiliza alrededor de 2.000 puertos; por lo tanto la creación y gestión de estos puertos debe ser barata.

El problema de la protección de un recurso frente a accesos ilegales se equipara al de la protección del correspondiente puerto frente a envíos ilegales. En Mach esto se consigue mediante el control que realiza el núcleo sobre la adquisición de habilitaciones para el puerto y también por el control de los mensajes realizado por el servidor de red.

La habilitación de un puerto tiene un campo que especifica los derechos de acceso sobre el puerto pertenecientes a la tarea que lo posee. Existen tres tipos diferentes de derechos de puerto.

Los *derechos de envío* permiten a los hilos dentro de la tarea que los posee el envío de mensajes al puerto correspondiente.

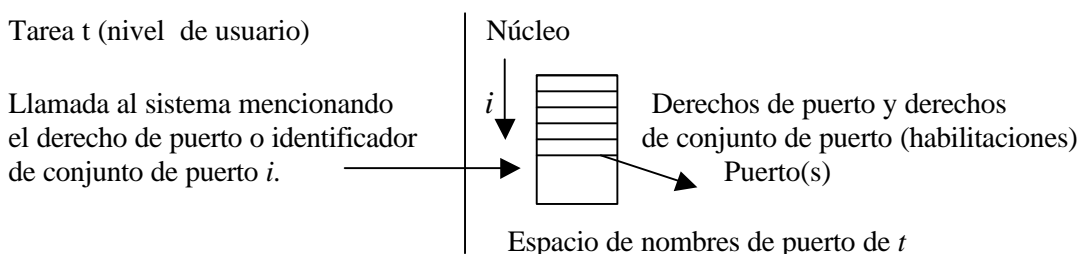


Figura 3. Un espacio de nombres de puerto de una tarea.

Una forma más restringida son los *derechos de un solo envío*, que permiten el envío de tan solo un mensaje, después de lo cual los derechos son destruidos automáticamente por el núcleo. Esta forma restringida permite, por ejemplo, que un cliente obtenga una respuesta desde un servidor sabiendo que el servidor no le enviará mensajes (y por lo tanto protegiéndole de servidores erróneos); además, se libera al servidor de la tarea de eliminar los derechos de envío recibidos desde los clientes.

Finalmente los *derechos de recepción* permiten a los hilos de una tarea recibir mensajes desde la cola de mensajes del puerto. En cada momento sólo una tarea puede poseer los derechos de recepción, mientras que un número variable de tareas pueden poseer los derechos de envío o de *un solo envío*. Mach soporta sólo la comunicación *muchos a uno*: la multidifusión no aparece implementada directamente en el núcleo.

En el momento de su creación a cada tarea se le proporciona un *derecho de puerto de inicio*, que es un derecho de envío que sirve para obtener servicios de otras tareas. Una vez creada, los hilos pertenecientes a la tarea adquieren más derechos de puerto ya sea mediante su creación o mediante su recepción a través de mensajes.

Los derechos de puerto de Mach se almacenan en el núcleo y son protegidos por él (ver Figura 3). Las tareas nombran los derechos de puerto mediante identificadores locales válidos únicamente en el *espacio de nombres de puerto* local a la tarea. Esto permite al equipo de desarrollo del núcleo elegir representaciones eficientes para estas habilitaciones (tales como punteros a colas de mensajes) y elegir nombres locales de tipo entero a conveniencia del núcleo para permitir búsquedas y fácilmente la habilitación dado el nombre. De hecho, análogamente a los descriptores de archivo de UNIX, los identificadores locales son enteros utilizados para indexar una tabla del núcleo que contiene las habilitaciones de la tarea.

El esquema de nombres y protección de Mach permite un acceso rápido a las colas de mensajes locales dado un identificador de usuario. Como contrapartida a esta ventaja está el gasto realizado por el núcleo al procesar todos los derechos transmitidos en los mensajes entre tareas. Como mínimo, los derechos de envío deben tener asociado un nombre local en el espacio de nombres de la tarea receptora, junto con espacio en sus tablas del núcleo.

3. TAREAS E HILOS

Una tarea es un entorno de ejecución: las propias tareas no pueden realizar ninguna acción, únicamente lo pueden realizar sus hilos. Los principales recursos asociados directamente con una tarea son su espacio de direcciones, sus hilos, sus derechos de puerto, sus conjuntos de puertos y el espacio de nombres local en el que se buscan los derechos de puerto y los conjuntos de puertos.

Creación de una nueva tarea. La operación *fork* de UNIX crea un nuevo proceso copiando uno ya existente. El modelo de creación de procesos de Mach es una generalización del de UNIX. Las tareas son creadas con referencia a lo que llamaremos *tarea tipo* (no necesariamente la creadora). La nueva tarea reside en el mismo computador que la tarea tipo. Al no proporcionar Mach la posibilidad de migración de tareas, la única forma de establecer una tarea sobre un computador remoto es a través de otra tarea que ya resida allí. El nuevo derecho de puerto de inicio de la tarea se hereda de la tarea tipo y su espacio de direcciones puede estar vacío o bien ser heredado de la misma forma. Una tarea recién creada no tiene hilos. En su lugar, el creador de la tarea solicita la creación de un hilo dentro de la tarea hijo. Posteriormente podrán crearse otros hilos a partir de los hilos existentes dentro de la tarea. En la Figura 4 aparecen algunas de las llamadas de Mach relacionadas con la creación de tareas e hilos.

Invocación de operaciones del núcleo. Cuando se crea una tarea o hilo en Mach se le asigna un denominado *puerto del núcleo*. Las llamadas al sistema de Mach se dividen entre las implementadas directamente como trampas (*traps*) y las implementadas mediante paso de mensajes a los puertos del núcleo. Este último método tiene la ventaja de permitir operaciones con transparencia de red sobre tareas e hilos que pueden ser remotos o locales. Un servicio del núcleo gestiona recursos del núcleo de la misma forma en que un servidor de nivel de usuario maneja otros recursos. Cada tarea tiene derechos de envío sobre su puerto del núcleo, lo que le permite invocar operaciones por sí misma (como la creación de un nuevo hilo). Cada uno de los servicios del núcleo accedido mediante paso de mensajes tiene una definición de interfaz. Las tareas acceden a esos servicios mediante procedimientos de resguardo, los cuales se generan de sus definiciones de interfaz mediante el Generador de Interfaz de Mach.

task_create (*tarea_padre*, *hereda_memoria*, *hijo_tarea*)
tarea_padre es la tarea utilizada como tarea tipo en la creación, *hereda_memoria* especifica si el hijo debe heredar el espacio de direcciones de su padre o bien se le debe asignar un espacio de direcciones vacío, *tarea_hijo* es el identificador de la nueva tarea.

thread_create (*padre_tarea*, *hijo_hilo*)
tarea_padre es la tarea en la que se quiere crear el nuevo hilo, *hilo_hijo* es el identificador del nuevo hilo. El hilo creado no tiene estado de ejecución y está suspendido.

thread_set_state (*hilo*, *aroma*, *nuevo_estado*, *contador*)
hilo es el identificador del hilo al que se quiere proporcionar un estado de ejecución, *aroma* especifica la arquitectura de la máquina, *nuevo_estado* especifica el estado (el contador de programa y el puntero de pila, por ejemplo), *contador* es la talla del estado.

thread_resume (*hilo*) Se utiliza para reanudar el hilo suspendido identificado por *hilo*.

Figura 4. Creación de tareas e hilos.

Gestión de excepciones. Las tareas y (a veces) los hilos poseen, además del puerto del núcleo, un *puerto de excepciones*. Cuando ocurren ciertos tipos de excepciones el núcleo reacciona intentando enviar un mensaje que describe la excepción a un puerto de excepciones asociado. Si el hilo carece de dicho puerto, busca el de la tarea. El hilo que recibe el mensaje puede intentar reparar el problema y devuelve, a continuación, un valor de estado en un mensaje de respuesta. Si el núcleo encuentra un puerto de excepciones y recibe una respuesta indicando éxito, entonces reinicia el hilo que generó la excepción. En otro caso el núcleo elimina dicho hilo.

Por ejemplo, cuando una tarea realiza una violación de acceso sobre el espacio de direcciones o bien una división por cero, el núcleo envía un mensaje a un puerto de excepciones. El propietario de este puerto podría ser una tarea de depuración de errores, que podría estar ejecutándose en cualquier punto de la red gracias al sistema de comunicación independiente de ubicación de Mach. Las faltas de página son manejados por paginadores externos.

Gestión de tareas e hilos. Alrededor de cuarenta procedimientos en la interfaz del núcleo están relacionados con la creación y gestión de tareas e hilos. El primer argumento de cada procedimiento es un derecho de envío al puerto del núcleo asociado, y se usan llamadas al sistema por paso de mensajes para solicitar la operación al núcleo destino. Algunas de esas llamadas de gestión de tareas e hilos se muestran en la Figura 4. En resumen, las prioridades de planificación de los hilos se pueden configurar de forma individual; los hilos y las tareas pueden ser suspendidos, reanudados y terminados; y el estado de ejecución de los hilos puede ser creado de forma externa, leído y modificado. Otras invocaciones a la interfaz del núcleo sirven para asignar hilos sobre ciertos *conjuntos de procesadores*. Mediante estas asignaciones, los recursos computacionales disponibles pueden asignarse toscamente a tipos de actividades diferentes.

4. MODELO DE COMUNICACIÓN

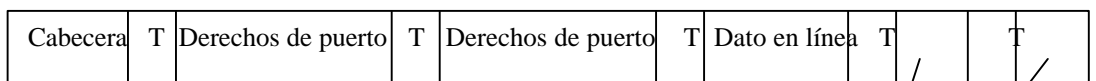
Mach proporciona una única llamada al sistema para el paso de mensajes: *mach_msg*.

4.1. MENSAJES

Un mensaje está formado por una cabecera de tamaño fijo seguida por una lista de tamaño variable de campos de datos (ver Figura 5). La cabecera de tamaño fijo contiene:

El puerto de destino: Por simplicidad forma parte del mensaje en lugar de especificarse como un parámetro separado de la llamada al sistema *mach_msg*. Se indica mediante el identificador local de los correspondientes derechos de envío.

Un puerto de respuesta: Si se necesita una respuesta, entonces se insertan los derechos de envío a un puerto local (es decir, un puerto para el que el hilo que envía tenga derechos de recepción) en el mensaje.



T = información tipo

↓ ↓
Punteros a datos no alineados

Figura 5. Un mensaje en Mach conteniendo derechos sobre puerto y datos no alineados.

Un identificador de operación: Identifica una operación (procedimiento) en la interfaz de servicio que sólo puede ser interpretada por las aplicaciones.

Tamaño de los datos extra: A continuación de la cabecera aparece una lista de tamaño variable de datos toados. No existe límite para su tamaño, excepto el número de bits del campo y el tamaño del espacio de direcciones.

Cada ítem en la lista consecutiva de la cabecera del mensaje, puede ser uno de los siguientes:

Datos de mensaje tipado: Individuales, datos en línea etiquetados con su tipo.

Derechos de puerto: Referidos a sus identificadores locales.

Punteros a datos no alineados: Datos que residen en un bloque de memoria separado y no contiguo.

Los mensajes en Mach están formados por una cabecera de tamaño fijo y múltiples bloques de datos de tamaño variable, algunos de los cuales pueden estar *no alineados* (es decir en bloques separados). Sin embargo, cuando se envían mensajes con datos no alineados, el núcleo (no la tarea receptora) escoge la ubicación de los datos recibidos en el espacio de direcciones de la tarea receptora. Se trata de un efecto colateral de la técnica de la *copia en escritura* utilizada para transferir estos datos. Las regiones de memoria virtual extra recibidas en un mensaje que no se vayan a usar más deben ser liberadas de forma explícita por la tarea receptora. Al ser los costos de las operaciones de memoria virtual superiores a los costos de la copia de datos para pequeñas cantidades de datos, se supone que sólo se envían de modo no alineado grandes cantidades de datos.

La ventaja de permitir múltiples componentes de datos en los mensajes consiste en que posibilita al programador la asignación de memoria de forma separada para los datos y para los metadatos. Por ejemplo, un servidor de archivos puede asignar bajo petición un bloque de disco desde su caché. En lugar de copiar el bloque en el búfer del mensaje, de forma contigua a la cabecera, los datos pueden ser tomados directamente desde donde residen mediante la inserción de un puntero en el mensaje de respuesta. Se trata de una variante de la *E/S de acopio disperso*, en la que los datos son leídos o escritos desde múltiples ubicaciones del espacio de direcciones del invocador mediante una llamada al sistema. Las llamadas al sistema *readv* y *writv* de UNIX actúan de esta forma.

El tipo de cada elemento de datos en un mensaje Mach es especificado por el emisor. Esto permite a los servidores de red de nivel de usuario empaquetar los datos con formato estándar, cuando se transmiten sobre la red. Sin embargo, este esquema de empaquetamiento tiene peores prestaciones en comparación con el empaquetamiento y desempaquetamiento realizado por los procedimientos de resguardo generados desde las definiciones de la interfaz. Los procedimientos de resguardo conocen los tipos de datos implicados, no necesitan incluirlos en los mensajes y pueden empaquetarlos directamente en el mensaje. Un servidor debería copiar los datos con tipo del emisor dentro de otro mensaje al empaquetarlos.

4.2. PUERTOS

Un puerto en Mach tiene una cola de mensajes cuyo tamaño puede ser cambiado dinámicamente por la tarea con los derechos de recepción. Esta posibilidad permite a los receptores implementar cierta forma de control de flujo. Cuando se utiliza un derecho de envío normal, un hilo que intente enviar un mensaje a un puerto cuya cola de mensajes esté llena se bloqueará hasta que haya espacio disponible. Cuando un hilo utiliza un derecho de *un solo envío*, el receptor siempre inserta en la cola el mensaje, aunque la cola de mensajes esté llena. Al utilizarse el derecho de *un solo envío*, se sabe que no habrá más mensajes desde dicho emisor. Los hilos del servidor pueden evitar la espera enviando derechos de *un solo envío* en sus respuestas a los clientes.

Envío de derechos de puerto. Cuando los derechos de envío de un puerto se insertan en un mensaje, el receptor los adquiere sobre el mismo puerto. Sin embargo, cuando lo que se transmite son los derechos de recepción son desasignados automáticamente de la tarea que los emite. La razón es que los derechos de recepción no pueden asignarse a más de una tarea al tiempo. Todos los mensajes en la cola del puerto y todos los mensajes transmitidos posteriormente podrán ser recibidos por el nuevo propietario de los derechos de recepción, y esto se consigue de forma transparente a las tareas que envían mensajes al puerto. La transferencia transparente de derechos de recepción es relativamente fácil de conseguir cuando estos derechos se transfieren dentro de un único computador. La habilitación adquirida es simplemente un puntero a una cola local de mensajes.

Monitorización de la conectividad. El núcleo está diseñado para informar a los emisores y receptores de la aparición de ciertas condiciones bajo las que la emisión o recepción de mensajes es inútil. Para conseguirlo gestiona información sobre el número de derechos de envío y recepción sobre un puerto dado. Si no existe ninguna tarea con los derechos de recepción sobre cierto puerto (por ejemplo, al fallar la tarea que disponía de dichos derechos), entonces todos los derechos de envío en los espacios de nombres de puerto de las tareas locales se convierten en *nombres muertos*.

Cuando un emisor intenta utilizar un nombre referido a un puerto para el que los derechos de recepción ya no existen, el núcleo convierte el nombre en un nombre muerto y devuelve una indicación del error. Análogamente las tareas pueden solicitar al núcleo la notificación asíncrona de la condición de que ya no existan derechos de envío para un cierto puerto. El núcleo realiza esta notificación mediante el envío de un mensaje, utilizando los derechos de envío proporcionados para ello por el hilo. La condición de no existencia de derechos de envío puede ser detectada mediante un contador de referencias que se incrementa cuando se crea un derecho de envío y que se decrementa cuando se destruye.

El chequeo de estas condiciones en un sistema distribuido es una operación compleja y cara. Dado que los derechos pueden ser enviados en los mensajes, los derechos de envío o recepción para un cierto puerto podrían residir en cualquier tarea, o estar en un mensaje dentro de la cola de un puerto, o en tránsito entre computadoras.

Conjuntos de puertos. Son colecciones de puertos creadas dentro de una única tarea y gestionadas localmente. Cuando un hilo realiza una operación de recepción sobre un conjunto de puertos, el núcleo devuelve un mensaje que fue entregado a algún miembro del conjunto. También devuelve el identificador de los derechos de recepción del puerto para que el hilo pueda realizar el correspondiente tratamiento del mensaje.

Los conjuntos de puertos son útiles ya que normalmente un servidor debe proporcionar servicio a mensajes de clientes que llegan por todos los puertos y en cualquier instante. El intento de recibir un mensaje desde un puerto cuya cola de mensajes está vacía provoca la detención del hilo y esta espera se mantiene aunque desde otro puerto se reciba posteriormente un mensaje. La asignación de un hilo a cada puerto resuelve este problema pero no es factible para servidores con un número grande de puertos ya que los hilos son recursos más caros que los puertos. Mediante la asociación de puertos en conjuntos de puertos, puede utilizarse un único hilo para dar servicio a los mensajes entrantes sin temor a perder ninguno de ellos. Además este hilo se inmovilizará si no hay mensajes disponibles desde ningún puerto del conjunto, evitando esperas activas en las que el hilo sondea los puertos hasta que llega de un mensaje por alguno de ellos.

4.3. MACH_MSG

La llamada al sistema *mach_msg* es extremadamente complicada ya que proporciona tanto servicios de paso de mensajes asíncrono como interacciones de tipo petición-respuesta. La llamada completa es como sigue:

mach_msg (*cabecera_msg*, *opción*, *tam_envío*, *tam_recep*, *nombre_recep*, *timeout*, *notifica*)
cabecera_msg es un puntero a una cabecera de mensajes común para los mensajes de envío y recepción, *opción* indica si es un envío, una recepción o ambos, *tam_envío* y *tam_recep* son los tamaños de los búferes de mensajes de envío y recepción, *nombre_recep* especifica los derechos de recepción del puerto o conjunto de puertos (si se recibe un mensaje), *timeout* pone un límite al tiempo total para el envío y/o recepción de un mensaje, *notifica* proporciona derechos de puerto que utiliza el núcleo para enviar mensajes de notificación bajo circunstancias excepcionales.

Mach_msg tanto envía un mensaje, como lo recibe o ambas cosas. Es una única llamada al sistema, que utilizan los clientes para enviar un mensaje de petición y recibir una respuesta, y los servidores para responder al último cliente y recibir el siguiente mensaje de petición. Otro beneficio de la utilización de llamadas, de envío/recepción combinadas, es que para el caso de un cliente servidor ejecutándose en la misma computadora la implementación puede utilizar una optimización llamada *planificación con traspaso*. Consiste en que cuando una tarea va a inmovilizarse una vez que ha enviado un mensaje a otra tarea, *dona* el resto de su intervalo de tiempo de ejecución (*timeslice*) al hilo de la otra tarea. Esto es más barato que ir a la cola de hilos preparados para seleccionar el siguiente a ejecutar.

Los mensajes enviados por el mismo hilo se entregan en el orden de envío, siendo esta entrega fiable. Por lo menos, esto se garantiza cuando los mensajes se envían entre tareas bajo un mismo núcleo, incluso cuando no hay espacio en el búfer. Cuando los mensajes se transmiten a

través de la red a una computadora independiente, se utiliza la semántica de entrega *como máximo una vez*.

El *timeout* es útil en situaciones en las que no es deseable que un hilo se inmovilice indefinidamente, por ejemplo en espera de un mensaje que puede no llegar nunca, o esperando por espacio en una cola sobre un puerto de servidor erróneo.

5 IMPLEMENTACIÓN DE LA COMUNICACIÓN

Uno de los aspectos más interesantes de la implementación del sistema de comunicación en Mach es la utilización de servidores de red de nivel de usuario. Los servidores de red (*netmsgservers*), uno por computadora, son colectivamente responsables de extender la semántica de comunicación local sobre toda la red. Esto incluye preservar, en la medida de lo posible, las garantías de reparto y hacer que la comunicación sobre la red sea transparente. También incluye la realización y monitorización de las transferencias de derechos de puertos. En concreto, los servidores de red son los responsables, de la protección de los puertos frente a accesos ilegales y del mantenimiento de la privacidad de los mensajes enviados por la red.

5.1. GESTIÓN TRANSPARENTE DE MENSAJES

Debido a que los puertos son siempre locales al núcleo de Mach, es necesario añadir una abstracción impuesta de forma externa, el *puerto de red*, utilizada para enviar los mensajes de red.

Un puerto de red es un identificador de canal globalmente único gestionado exclusivamente por los servidores de red y que éstos asocian a cada único puerto Mach en cada momento. Los servidores de red poseen los derechos de envío y recepción sobre los puertos de red, de la misma forma que las tareas poseen los derechos de envío y recepción sobre los puertos Mach.

En la Figura 6 aparece una descripción de la transmisión de un mensaje entre tareas localizadas en diferentes computadoras. Los derechos de la tarea emisora se refieren a los puertos locales, sobre los cuales el servidor de red local dispone de los derechos de recepción. En la Figura 6, el identificador local del servidor de red para los derechos de recepción es 8. El servidor de red en la computadora emisora utiliza este identificador para buscar una entrada en una tabla de puertos de red para los que tiene derechos de envío. Esta tabla proporciona, un puerto y sugiere una dirección de red. A continuación *envía* el mensaje al que se añade el puerto de red, al servidor de red de la dirección indicada en la tabla. Allí este servidor de red local extrae el puerto de red y lo utiliza para buscar en una tabla de puerto de red para los que tiene derechos de recepción.

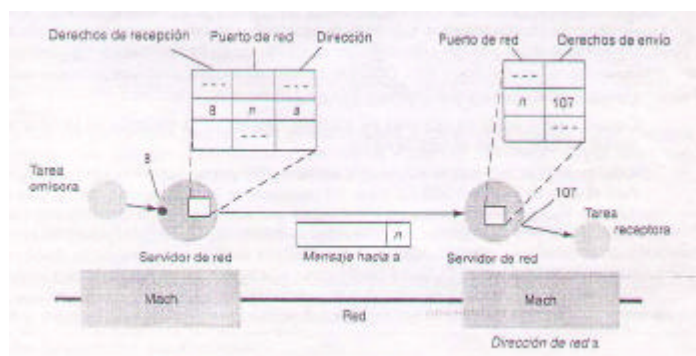


Figura 6. Comunicación de red en Mach.

Si encuentra una entrada válida (el puerto de red podría haber sido reasignado a otro núcleo), entonces esa entrada contiene el identificador de los derechos de envío a un puerto local en Mach. El servidor de red reenvía el mensaje utilizando estos derechos, consiguiendo así enviar el mensaje al puerto apropiado. El proceso completo realizado por los servidores de red es transparente tanto para el emisor como para el receptor.

El servidor de red de un computador recién inicializado se enrola en un protocolo de inicialización, por el cual se obtienen derechos de envío sobre los servicios de toda red.

Considérese qué ocurre después de esto, cuando se transfiere entre servidores de red un mensaje que contiene derechos de puerto. Estos derechos están tipados de modo que los servidores de red pueden seguirles la pista. Si una tarea emite los derechos de envío de un puerto local, el servidor de red local crea un identificador de puerto de red y una entrada en la tabla para el puerto local (si no existía ninguna) y enlaza el identificador al mensaje que envía. El servidor de red

receptor también crea una entrada en la tabla si no existe ninguna. La situación se vuelve más complicada cuando se transmiten los derechos de recepción. La principal cuestión relativa a esta transferencia de derechos es cómo conseguir que los mensajes enviados al puerto ahora lleguen a la computadora a la que se han transferido los derechos de recepción. Una posibilidad sería que Mach siguiera la pista de todos los servidores de red que dispusieran de los derechos de envío sobre un cierto puerto de red, y notificar directamente a dichos servidores en el momento en que los derechos de recepción hubieran sido transferidos. Este esquema se rechazó por su costo de gestión. Una alternativa más barata sería utilizar un medio de difusión (*broadcast*) hardware que difundiera el cambio de ubicación a todos los servidores de red. Sin embargo este servicio de broadcast no es fiable y no está disponible en todas las sedes. En su lugar, la responsabilidad de localizar un puerto de red se asignó a los servidores de red que disponen de los derechos de envío sobre dicho puerto.

Se debe recordar que un servidor de red utiliza una indicación de ubicación cuando reenvía un mensaje a otro servidor de red. Las posibles respuestas son:

- *el puerto está aquí*: el destino dispone de los derechos de recepción;
- *el puerto está muerto*: el puerto ha sido eliminado;
- *el puerto no está aquí, está transferido*: los derechos de recepción fueron transferidos a una cierta dirección;
- *el puerto no está aquí, lo desconozco*: no existe información sobre el puerto de red;
- *no hay respuesta*: el computador destino no responde.

Si se devuelve una dirección de reenvío, el servidor de red emisor envía el mensaje a dicha dirección; sin embargo se trata sólo de una sugerencia que puede ser inexacta. Si en algún punto el emisor se queda sin direcciones de reenvío, entonces recurre a la difusión. En este tipo de algoritmos de localización, y en concreto cuando se utilizan sobre una WAN, las dos principales cuestiones de diseño son cómo manejar las cadenas de direcciones de reenvío y qué hacer cuando una computadora que contiene una dirección de reenvío falla.

Una segunda cuestión para conseguir la transparencia en la migración es cómo sincronizar la entrega de los mensajes. Mach garantiza que dos mensajes enviados por el mismo hilo son entregados en el mismo orden. Si esto no se resuelve correctamente un mensaje podría ser entregado por el nuevo servidor de red antes de que se haya redirigido un mensaje anterior almacenado en la cola del computador original. El servidor de red puede conseguir esto reteniendo la entrega de todos los mensajes en la computadora origen hasta que todos los mensajes almacenados en la cola hayan sido transferidos a la computadora destino. Así, el reparto de mensajes puede redirigirse de forma segura, y devolver a los emisores la dirección de reenvío.

5.2. EXTENSIBILIDAD: PROTOCOLOS Y GESTORES DE DISPOSITIVOS

Protocolos de transporte. Los servidores de red de Mach más usados emplean únicamente TCP/IP como protocolo de transporte. Esto fue impulsado, en parte por la compatibilidad con UNIX, y en parte al ser seleccionado por la Universidad Carnegie-Mellon dada la complejidad de su red, que contiene más de 1.700 computadoras, alrededor de 500 de ellas ejecutando Mach. Se ajustó TCP/IP para conseguir robustez de forma bastante eficiente sobre dicha red. Sin embargo, por motivos de prestaciones, esto puede no ser apropiado en una LAN donde predominan las interacciones petición-respuesta.

Gestores de dispositivos de red del nivel de usuario. Algunos servidores de red proporcionan sus propios gestores de dispositivos (drivers) de red a nivel de usuario. Su objetivo es acelerar los accesos a la red. El situar los drivers en el nivel de usuario es, además de un método para mejorar la flexibilidad, una forma de compensar la degradación de prestaciones debida a la utilización de servidores de red al nivel de usuario. El núcleo exporta una abstracción para cada dispositivo, que incluye una operación para proyectar los registros de control del dispositivo sobre el espacio de usuario. Para el caso de una Ethernet, tanto los registros como los búferes de paquetes utilizados por el controlador pueden correlacionarse dentro del espacio de direcciones del servidor de red. Además, el núcleo ejecuta un código especial para despertar al hilo de nivel de usuario (que pertenece, en este caso, al servidor de red) cuando ocurre una interrupción. Este hilo es así capaz de gestionar la interrupción transfiriendo datos hacia o desde los búferes utilizados por el controlador y reiniciando el controlador para la siguiente operación.

6. GESTIÓN DE MEMORIA

Mach no sólo se destaca por emplear espacios de direcciones grandes y dispersos, sino también por sus técnicas de memoria virtual entre tareas que permiten la compartición de memoria entre las tareas. No sólo puede compartirse la memoria física entre tareas en ejecución sobre el mismo núcleo Mach, sino que además el soporte de paginadores externos de Mach (llamados *gestores de memoria*) permite compartir los contenidos de la memoria virtual entre tareas, aunque residan en computadores diferentes. Finalmente, es relevante la implementación de memoria virtual de Mach que se estratifica en niveles dependientes e independientes de máquina, lo que facilita la portabilidad del núcleo.

6.1 ESTRUCTURA DEL ESPACIO DE DIRECCIONES

La visión del espacio de direcciones en Mach es la de una colección de grupos de páginas contiguas que se nombran mediante sus direcciones, en lugar de regiones identificables separadamente. Consecuentemente, la protección en Mach se aplica a las páginas en lugar de a las regiones. Las llamadas al sistema de Mach se refieren a direcciones y tamaños en lugar de usar identificadores de región. Por ejemplo, Mach no *hace crecer la región de pila*. En su lugar se asignan algunas páginas justo por debajo de las que están siendo utilizadas para la pila.

Llamaremos región a una colección contigua de páginas con propiedades comunes a la región.

Como hemos visto Mach soporta un gran número de regiones, que pueden usarse para múltiples fines como almacenar datos o proyectar archivos en memoria. Las regiones pueden crearse mediante cualquiera de las siguientes formas:

- Pueden ser asignadas de forma explícita mediante una llamada a *vm_allocate*. Las regiones creadas con *vm_allocate* están puestas a cero por defecto.
- Puede crearse una región en asociación con un *objeto de memoria*, utilizando *vm_map*.
- Pueden asignarse regiones a tareas recién creadas a través de la declaración como herencia desde una tarea tipo, utilizando *vm_inherit* aplicado a la región de tarea tipo.
- Pueden asignarse automáticamente regiones en el espacio de direcciones de una tarea como efecto colateral del paso de mensajes.

Todas las regiones pueden establecer permisos de lectura/escritura/ejecución, mediante *vm_protect*. Pueden copiarse regiones dentro de las tareas mediante *vm_copy*, y otras tareas podrán leer o modificar los contenidos mediante *vm_read* y *vm_write*. Cualquier región previamente asignada puede ser liberada usando *vm_deallocate*.

6.2. COMPARTICIÓN DE MEMORIA: HERENCIA Y PASO DE MENSAJES

Mach permite una generalización de la semántica de UNIX *fork* mediante el mecanismo de *herencia de memoria*. Hemos visto que cada tarea se crea desde otra tarea, que actúa como plantilla. Una región heredada desde una tarea tipo contiene el mismo rango de direcciones y su memoria puede ser:

- *Compartida*: secundada por la misma memoria.
- *Copiada*: almacenada en una zona de memoria copiada de la memoria de la tarea tipo en el mismo instante en que se creó la región

También es posible, rechazar la herencia de cierta región que no sea necesaria en el hijo.

Para el caso de *fork* en UNIX, el texto del programa de la tarea tipo se hereda y se comparte con la tarea hija. Lo mismo ocurre para una región que contiene código de biblioteca compartida. Sin embargo, la pila y el montón del se heredan mediante copias de las regiones de la tarea tipo. Además, si se solicita a la tarea tipo que comparta con su hijo cierta región de datos, podrá configurarse esta región como heredada para compartir.

Los datos de los mensajes no alineados se transfieren entre las tareas con un método similar a la herencia con copia. Mach crea una región en el espacio de direcciones del receptor, y sus contenidos iniciales son una copia de la región que contiene los datos no alineados del emisor.

Al contrario de la herencia, la región del receptor no suele ocupar el mismo rango de direcciones que la región emisora. El rango de direcciones de la región de envío puede haber sido previamente utilizada por una región existente en el receptor.

Mach utiliza *copia en escritura* tanto para la herencia con copia como para el paso de mensajes. Para ello Mach realiza una suposición optimista, parte o toda la memoria que es

heredada con copia o pasada como datos no alineados de mensajes, no será escrita por tarea alguna, incluso cuando existan permisos de escritura.

Para justificar esta suposición, considérese de nuevo la llamada al sistema de *UNIX fork*.

Normalmente la utilización de *fork* está seguida de cerca por una llamada a *exec*, que sobrescribe los contenidos del espacio de direcciones, incluyendo el montón y la pila. Si la memoria ha sido copiada físicamente al efectuar *fork*, entonces la mayor parte del copiado se desperdiciara: se modifican pocas páginas entre esas dos llamadas.

La optimización de *copia en escritura* ayuda a compensar los costos del cambio de contexto que se generan en la transmisión a través de un servidor de red.

6.3. EVALUACIÓN DE LA COPIA-EN-ESCRITURA

A pesar de que la *copia en escritura* ayuda a pasar datos de mensajes entre las tareas y el servidor de red, no puede usarse para facilitar la transmisión sobre la red. Esto es porque las computadoras involucradas no comparten la memoria física.

La *copia en escritura* es eficiente siempre que haya una cantidad suficiente de datos para enviar. La ventaja de evitar la copia física debe superar los costos de las manipulaciones de la tabla de páginas. En Abrossimov y otros [1989] aparecen valores de prestaciones de una implementación Mach en una Sun 3/60; reproducimos algunos de ellos en la Figura 7.

Tamaño de la región	Copia simple	Crear región	Cantidad de datos copiados (en escritura)		
			0 kilobytes (0 páginas)	8 kilobytes (1 página)	256 kilobytes (32 páginas)
8 kilobytes	1,4	1,57	2,7	4,82	-----
256 kilobytes	44,8	1,81	2,9	5,12	66,4

Nota: todos los tiempos están en milisegundos.

Figura 7. Sobrecargas de la *copia en escritura*.

Las primeras dos columnas sirven sólo de referencia. La columna *Copia simple* muestra el tiempo necesario para copiar todos los datos entre dos regiones ya existentes (es decir, sin utilizar la *copia en escritura*). La columna *Crear región* indica el tiempo necesario para crear una región inicializada a cero (pero a la que no se accede). El resto de columnas proporcionan los tiempos medidos en los experimentos en los que una región existente fue copiada en otra utilizando la *copia en escritura*. Estos valores incluyen el tiempo necesario para la creación de la región copia, la copia de datos en modificación y la destrucción de la región copia. Para cada tamaño de región, se proporcionan valores asociados a diferentes cantidades de datos modificados en la región fuente.

Para el envío de un mensaje de tamaño 8 kilobytes entre dos tareas locales, el envío de datos no alineados (es decir, la utilización de la *copia en escritura*) no parece tener una clara ventaja sobre el envío alineado, en el que simplemente se copia. Por otra parte, la transmisión de un mensaje no alineado de tamaño 256 kilobytes es mucho menos cara que la transmisión alineada.

Un usuario puede especificar regiones mediante rangos de direcciones que no comiencen ni terminen en fronteras de páginas. Sin embargo, en Mach está obligado a compartir memoria a nivel de página. Todos los datos dentro de las páginas involucradas pero no especificados por los usuarios serán en cualquier caso copiados entre las tareas. Este es un ejemplo de lo que se conoce como *compartición falsa*.

6.4 PAGINADORES EXTERNOS

El núcleo de Mach no soporta directamente archivos o cualquier otra abstracción de almacenamiento externo. En su lugar asume que dichos recursos son implementados por paginadores (*paggers*) externos. Al igual que Multics, Mach ha elegido el modelo de acceso correlacionado para objetos de memoria. En lugar de acceder a los datos utilizando explícitamente operaciones *lee* y *escribe*, el programador sólo necesita acceder directamente a las correspondientes posiciones de memoria virtual. Una ventaja del acceso correlacionado es su uniformidad: se presenta al programador un único modelo de acceso a los datos, en lugar de dos.

La memoria virtual de Mach consta primordialmente del protocolo entre el núcleo y un paginador externo necesario para gestionar la correspondencia de los datos almacenados por este último.

Mach permite, mediante la llamada a *vm_map*, asociar una región con datos contiguos que arrancan de cierta posición de un objeto de almacenamiento. Esta asociación implica que los accesos de lectura a direcciones de la región se satisfacen con datos almacenados en el objeto de memoria, y los datos de la región modificados por accesos de escritura se propagan al objeto de almacenamiento. En general, el objeto de memoria es manejado desde un paginador externo, a pesar de que existe un paginador por defecto, implementado por el propio núcleo.

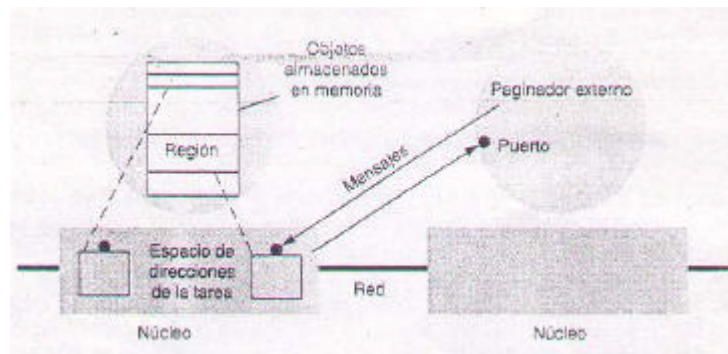


Figura 8. Paginador externo.

El objeto de memoria está representado por los derechos de envío a un puerto utilizado por el paginador externo, el cual satisface las peticiones concernientes al objeto de memoria provenientes del núcleo. Para cada objeto de memoria correlacionado, el núcleo gestiona un recurso local llamado *objeto cacheado en memoria* (ver Figura 8). En esencia se trata de una lista de páginas que contienen datos respaldados por el correspondiente objeto de memoria.

Las funciones de un paginador externo son:

- almacenar los datos desalojados por el núcleo desde su caché de páginas,
- proporcionar los datos de las páginas según los vaya solicitando el núcleo,
- imponer las limitaciones de consistencia asociadas a la abstracción del recurso de memoria subyacente para el caso en que el recurso de memoria sea compartido y varios núcleos puedan mantener de forma simultánea varios objetos almacenados en memoria para el mismo objeto de memoria.

Los principales componentes del protocolo de paso de mensajes entre el núcleo (N) y el paginador externo (PE) se resumen en la Figura 9. Cuando se invoca a *vm_map*, el núcleo local contacta con el paginador externo utilizando el derecho de envío del puerto asociado al objeto de memoria proporcionado en la llamada *vm_map*, enviándole un mensaje de tipo *memory_object_init*. El núcleo inserta en este mensaje los derechos de envío, los cuales son utilizados por el paginador externo para controlar el objeto almacenado en memoria. También declara el tamaño y la posición de los datos solicitados al objeto de memoria, y el tipo de acceso (lectura/escritura). El paginador externo responde con un mensaje de tipo *memory_object_set_attributes*, que indica al núcleo que el paginador está preparado para gestionar solicitudes de datos y proporciona información adicional sobre los requisitos del paginador en relación con el objeto de memoria. Cuando un paginador externo recibe un mensaje *memory_object_init*, es capaz de determinar si necesita o no implementar un protocolo de consistencia, ya que todos los núcleos que quieren acceder al correspondiente objeto de memoria deben enviar este mensaje.

6.5. GESTION DE ACCESOS A UN OBJETO DE MEMORIA

En el caso de que un objeto de memoria no sea compartido (sólo esté correlacionado en una computadora), utilizando un archivo como objeto de memoria, suponemos que no se realiza prebúsqueda de los datos del archivo desde el paginador externo, todas las páginas en la región correlacionada con este archivo se protegen inicialmente por el hardware frente a todos los accesos, al no existir ningún dato del archivo. Cuando un hilo intenta leer una de las páginas de la región, se genera un fallo de página.

Evento	Emisor	mensaje
<i>vm_map</i> invocado por la tarea	N → PE	<i>memory_object_init</i>
	PE → N	<i>memory_object_set_attributes</i>
	PE → N	<i>memory_object_data_error</i>
Faltas de página de la tarea al no existir un marco de pagina	N → PE	<i>memory_object_data_request</i>
	PE → N	<i>memory_object_data_provided</i>
	PE → N	<i>memory_object_data_unavailable</i>
El núcleo escribe una página modificada sobre almacenamiento persistente	N → PE	<i>memory_object_data_write</i>
El paginador externo ordena al núcleo escribir una pagina/ poner permisos de acceso	PE → N	<i>memory_object_lock_request</i>
	N → PE	<i>memory_object_lock-completed</i>
Faltas de página de la tarea al no existir suficientes datos	N → PE	<i>memory_object_data_unlock .</i>
	N →	<i>memory_object_lock_request</i>
El objeto de memoria deja de estar mapeado	N → PE	<i>memory_object_terminate</i>

Figura 9. Mensajes del paginador externo.

El núcleo busca el derecho de envío del puerto asociado al objeto de memoria correspondiente a la región correlacionada y envía un mensaje *memory_object_data_request* al PE que sería un servidor de archivos. Si todo se realiza correctamente, el PE responde con los datos de la página, utilizando un mensaje *memory_object_data_provided*.

Cuando el archivo de datos es modificado por la computadora que lo ha correlacionado, a veces el núcleo necesitará escribir la página desde su objeto almacenado en memoria. Para ello envía un mensaje de tipo *memory_object_data_write* al PE en el que se inserta la página de datos. Las páginas modificadas se transmiten al PE como en efecto colateral del reemplazo de páginas (cuando el núcleo necesita espacio para otra página). Además, el núcleo puede decidir la escritura de la página en el almacenamiento de respaldo para garantizar su persistencia. Por ejemplo, las implementaciones de UNIX escriben en disco los datos modificados normalmente cada 30 segundos, en caso de un fallo del sistema. Algunos sistemas operativos permiten a los programas controlar la seguridad de sus datos mediante una operación *flush* sobre un archivo abierto, el cual provoca que todas las páginas modificadas sean escritas en el archivo para el momento en que haya vuelto la llamada.

El propio PE puede solicitar al núcleo, mediante un mensaje de tipo *memory_object_lock_request*, que los datos modificados en cierto rango sean enviados al paginador para su consumación sobre el almacén permanente de acuerdo con estas garantías. Cuando el núcleo haya completado las acciones solicitadas, envía al PE un mensaje *memory_object_lock_completed*. Todos los mensajes descriptos son enviados asincrónicamente, incluso si se generan en una combinación petición-respuesta. Esto ocurre, en primer lugar, para que los hilos no sean suspendidos y puedan realizar otras tareas después del envío de las solicitudes. Además, un hilo no queda colgado cuando ha enviado una solicitud a un PE o núcleo que resulta que ha fallado. Finalmente, un PE puede utilizar el protocolo de mensajes asíncrono basado en mensajes para implementar una política de prebúsqueda de paginas. Puede enviar datos de páginas en mensajes *memory_object_data_provided* a los objetos cacheados en memoria anticipándose a su utilización, en lugar de esperar que se genere un fallo de páginas y se soliciten los datos.

Gestión de accesos compartidos a un objeto de memoria. En el caso de que varias tareas residentes en diferentes computadoras se correlacionan a un mismo archivo únicamente para lectura, no existe problema de consistencia y las solicitudes a las páginas de los archivos pueden satisfacerse inmediatamente. Sin embargo, si al menos una tarea correlaciona el archivo para su escritura, entonces el paginador externo o el servidor de archivos debe implementar un protocolo para asegurar que las tareas no lean versiones inconsistentes de la misma página.

7. CONCLUSIONES

Mach puede ejecutarse tanto en multiprocesadores como en computadores monoprocesador conectados por redes. Debe permitir la evolución de los nuevos sistemas distribuidos manteniendo la compatibilidad con UNIX.

Mach incluye mecanismos complejos de comunicación entre procesos y de gestión de la memoria virtual. Sus principales abstracciones tienen que ver con los hilos y las tareas, los puertos y la memoria virtual, destacándose la comunicación entre tareas y el soporte para la compartición de objetos entre computadores mediante la paginación.

El micronúcleo Mach 3.0 fue utilizado como base de la implementación MkLinux.

El núcleo de Mach es complejo y sofisticado en cuanto a sus posibilidades.

El modelo de tareas e hilos de Mach y la integración de la gestión de la memoria virtual con la comunicación representan una mejora importante sobre las posibilidades básicas de UNIX.

El modelo de comunicación entre tareas de Mach es muy rico y complejo.

La filosofía de Mach de proporcionar toda la funcionalidad extendida en el nivel de usuario ha sido abandonada, pero Mach sigue utilizándose y es una referencia inestimable para los desarrollos de arquitecturas de núcleos.

8. BIBLIOGRAFÍA

1. G. Coulouris, J. Dollimore, T. Kindberg. **Sistemas Distribuidos – Conceptos y Diseño – 3/E**. Addison Wesley, España, 2001. ISBN 84-7829-049-4.
2. <http://www.cdk3.net/oss/> - **Distributed Systems: Concepts and Design Edition 3**. By George Coulouris, Jean Dollimore and Tim Kindberg Addison-Wesley, ©Pearson Education 2001.
3. <http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html> - Project Mach was an operating systems research project of the Carnegie Mellon University [School of Computer Science](http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html) from 1985 to 1994.