Introducción y objetivos del trabajo

Introducción general

“La dificultad no es sino una palabra para designar la cantidad de fuerza que es necesaria para vencer un obstáculo”

Warren

Introducción general

I.1. Introducción

El propósito del sistema operativo en una máquina es proporcionar al usuario un interfaz, abstracción del hardware y el *firmware*, protegiendo unos programas de otros en el mismo sistema, dando soporte al almacenamiento de ficheros, y proporcionando una respuesta a las condiciones de excepción que los programas provoquen en el sistema. Un sistema operativo convencional es responsable de la gestión de recursos y de la planificación de trabajos.

En un sistema multiprogramado, las ejecuciones de múltiples programas alternan el uso del procesador, que se mantiene así el mayor tiempo posible ocupado: se puede decir que la clave de la multiprogramación reside en la manera de planificar los trabajos en el procesador.

Los avances tecnológicos actuales han incidido de manera importante en el número de unidades de procesamiento por un lado y en la organización de los módulos de memoria por otro: hoy día, la mayoría de arquitecturas cuentan ya con más de un procesador y la gestión de la memoria, en los sistemas multiprocesadores actuales (tanto de memoria compartida, como distribuida y privada) ofrece una organización jerárquica substancialmente más compleja que en los monoprocesadores convencionales.

Consecuencia de la multiplicidad de procesadores de cálculo en una misma máquina, unida a los avances en el software -nuevos paradigmas, librerías y lenguajes que facilitan la expresión de tareas de forma concurrente-, es que la programación paralela está en un momento de gran desarrollo e interés.

Hasta hace poco el énfasis en la utilización de los sistemas multiprocesadores se ponía, casi exclusivamente, en proporcionar un sistema que fuera lógicamente equivalente a un monoprocesador multiprogramado con un mejor rendimiento y fiabilidad en la multiprogramación, a base de ejecutar diferentes programas simultáneamente en diferentes procesadores y/o dedicar procesadores a determinadas aplicaciones o a un grupo de ellas. Las configuraciones más extendidas han sido las *master/slave*, dedicando algún procesador a servicios del sistema.

Recientemente, ha ido incrementándose el interés en desarrollar aplicaciones paralelas, con múltiples flujos implicados. Se consigue así una programación más sencilla, en comparación a la construcción de la aplicación como un programa secuencial. Además, en un multiprocesador es posible ejecutar varias porciones -flujos- de la misma aplicación simultáneamente mejorando el rendimiento.

En cuanto a la gestión de la memoria en los multiprocesadores actuales, puede experimentarse un amplio margen de latencia en el acceso a los módulos de memoria, dependiendo de la localización de los datos requeridos respecto al procesador que los solicita. Debido a la diferencia en los órdenes de magnitud que existen actualmente entre la velocidad de los componentes de cálculo y la de los componentes de comunicación y almacenamiento, ésto puede producirse tanto en sistemas de memoria distribuida, como en los de memoria compartida -considerados hasta ayer mismo como UMA (Uniform Memory Access)- a partir de un determinado número de procesadores.

Por otra parte, la contención entre múltiples procesadores para acceder a memoria compartida y a canales de comunicación también degrada los tiempos de acceso a la información. La localización de los datos pasa a ser importante, tanto para evitar la multiplicidad en las copias de datos, con el coste de transporte que ello supone (coherencia y mantenimiento de las memorias cache), como para acortar las distancias entre las instrucciones y los datos a los que acceden éstas.

La aplicación es la que conoce el trabajo y semántica de sus procesos para poder obtener el máximo rendimiento de sus componentes. En esta situación, si no existe la posibilidad de distribuir información o trabajo en tiempo de ejecución, es difícil conseguir escalar los programas paralelos sin perder rendimiento. Y esta información ha de proporcionársela el sistema.

En cuanto a los interfaces que se ofrecen a las aplicaciones para poder trabajar, es importante que se adapten a diferentes arquitecturas y máquinas. A la vez, que ofrezcan un medio familiar al programador y usuario. Por su filosofía de trabajo y su expansión en medios tanto comerciales, como educativos y de investigación, el entorno operativo UNIX es adecuado a estos requerimientos si, además, ofrece herramientas que permitan a cada aplicación adaptarse de la manera más adecuada a su perfil de ejecución.

Ésto es hoy posible a partir de la tecnología microkernel. Los entornos de microkernels permiten que el usuario pueda disponer de variedad de servicios y también de variedad en la manera de ofrecer cada servicio. Son máquinas virtuales que dan al usuario los entornos operativos conocidos (UNIX, DOS, OSF/1) a partir de subsistemas o servidores, mejorados y ampliados con toda la potencia del hardware actual subyaciente.

I.2. Entornos paralelos de desarrollo

Cuando el paralelismo se utiliza para ganar rendimiento, la granularidad más fina de las aplicaciones convierte la sobrecarga de las llamadas al sistema para la planificación de los flujos o el paso de mensajes, en un mecanismo de coste prohibitivo. En su lugar, puede recurrirse al código de nivel de usuario -básicamente, librerías- para estas funcionalidades: los costes de tiempo se reducen significativamente por utilizar llamadas a funciones en vez del mecanismo de entrada al kernel (traps), y la aplicación es el lugar adecuado para conocer y poder aplicar funciones más ligeras y más especializadas.

Desde el punto de vista de la aplicación, hay que caracterizar primero qué es realmente una aplicación paralela hoy día y qué necesidades tiene, qué soporte se le da en cuanto a librerías y qué tipo de granularidad le ofrecen el sistema y la librería.

Nos parece buena la definición de aplicación como el conjunto de acciones realizadas para obtener un fin -en una aplicación paralela, varias de estas acciones pueden coincidir en el tiempo-. Cada acción tiene su propio objetivo para acercar al total de la aplicación a su resultado, como fin corporativo. Por eso, entre las diferentes acciones de una misma aplicación, existen unos vínculos mucho más fuertes que entre acciones de diferentes aplicaciones: una aplicación acaba cuando haya conseguido su objetivo. Ésto involucra a todas sus acciones desde el comienzo de su ejecución hasta el final: de nada sirve que una acción haya finalizado hace tiempo, si existen acciones que no pueden continuar.

Vemos, por tanto, que es necesaria una armonía y compenetración entre los distintos trabajos de una aplicación que afecta al rendimiento final de toda ella y no puede pasarse por alto en la planificación global que lleva el sistema.

En este contexto, para dar soporte a la planificación de aplicaciones paralelas encontramos actualmente que:

- los mecanismos del kernel son muy generales, independientes de los objetivos y perfil de una aplicación determinada,
- la aplicación es la que puede especificar qué gestión concreta quiere para sus flujos y las características de sus flujos (tipo de concurrencia, paralelismo, mecanismos de sincronización...).

Aparecen librerías para facilitar este tipo de decisiones y, con ellas, diferentes niveles de planificación con poca o nula comunicación entre ellos, con decisiones y responsabilidades repetidas, y resultados a veces contradictorios: el kernel por un lado, servidores especializados por otro y la aplicación, a partir de sus librerías, por un tercero. Hay que definir quién y cómo debe hacer el qué.

Está comprobado que la organización tradicional de los sistemas operativos no conduce al

mejor rendimiento: en su diseño, el kernel proporciona mucha funcionalidad, mientras que las librerías (situadas entre el kernel y la aplicación) son un nivel relativamente “fino”. En particular, funciones como la planificación de flujos y la comunicación entre nodos dentro de una misma aplicación se realizan en el kernel, además de funciones que trascienden el ámbito de la aplicación, como puede ser el mantenimiento del espacio de direcciones y la asignación de procesadores. En el espacio de direcciones de la aplicación (librerías), pueden realizarse estas funciones con mayor eficiencia y es donde existe el conocimiento necesario para tomar las decisiones más apropiadas.

Esta nueva organización en niveles permite ganancias en rendimiento por el hecho de que las operaciones del “caso general” se llevan a cabo sin intervención del kernel. Ésto proporciona varios beneficios, ya que al sistema se accede mediante un trap y cambio de modo (no pocas veces, sobre todo en sistemas multiprocesadores, supone incluso un cambio de contexto), mientras que la librería es accesible a través de una llamada a procedimiento, mucho más eficiente. Además, las funciones construidas en el kernel han de ser totalmente generales, dando soporte a cada uno de los servicios requeridos por cada una de las aplicaciones, mientras que los servicios que proporciona la librería pueden adaptarse a los requisitos de un lenguaje concreto o de un particular tipo de aplicaciones.

Los procesadores, desde el punto de vista de la aplicación, aparecen ahora como un recurso más del sistema por el que competir, a la vez que nos hará ver la globalidad de la máquina como un subconjunto de máquinas menores dedicadas a cada aplicación. Y surgen dentro del kernel dos niveles de planificación: la que ya existía de asignar procesos a procesadores y la de asignar procesadores a aplicaciones.

El papel del kernel en una organización de este tipo es el de mediar entre diferentes aplicaciones (por ejemplo, para particionar la máquina y/o llevar la gestión de los procesadores que se asignan a las aplicaciones) y de facilitar información a la librería sobre una determinada aplicación, para que pueda tomar sus propias decisiones (por ejemplo, en cuanto a la planificación de los flujos).

Está claro que el sistema ha de traspasar funciones a la aplicación. Es importante resaltar que esta nueva organización no supone un incremento en la complejidad del código de alto nivel de la aplicación, ni aumenta el trabajo del programador de aplicaciones. Se trata sencillamente de dar al nivel usuario (librerías y compiladores) más oportunidades de optimizar el uso de los recursos del sistema, de modo que puedan proporcionar una implementación más eficiente de los programas de usuario. La aplicación y los interfaces de programación no cambian: continúan trabajando con las tradicionales llamadas a librerías.

I.3. Nuestro planteamiento

Nuestro estudio, al intentar ofrecer un entorno adecuado a la ejecución eficiente de aplicaciones paralelas en sistemas multiprocesadores de memoria compartida, se ha basado en el siguiente fundamento: **plantear las funciones operativas del kernel como de subsidiariedad respecto de la aplicación y facilitar los recursos para que ella misma los gestione**.

Es necesario facilitar al usuario la información y los mecanismos necesarios, que hasta ahora residían en el kernel, para que pueda definir y aplicar las políticas que piense más convenientes para cada una de sus aplicaciones. Hay que poder definir a nivel usuario, por ejemplo, a partir de librerías y con todas las ventajas de eficiencia que ello supone, políticas y mecanismos de planificación tan potentes como los que ofrecen los microkernels actuales (*hint*, *handoff*). Y todo ello con la misma funcionalidad que las herramientas del kernel.

No basta con suministrar información al usuario sobre cómo ha ido cada una de sus opera-

ciones, sino que hay que avisarle además del estado del sistema en los momentos en que sería adecuado tomar una decisión por parte de la aplicación en cuanto a qué trabajos realizar. Éso supone añadir nuevas llamadas al sistema para que, interactivamente, el usuario pueda adaptar y conocer los parámetros del sistema que afecten a su rendimiento.

Es esencial definir unas plantillas básicas en las librerías (funciones que inicialicen el entorno con un funcionamiento correcto por defecto) para que el usuario pueda favorecerse de las mejoras introducidas, sin necesidad de conocimientos de programación especiales. Plantillas definidas como bloques ortogonales para que el usuario tenga la oportunidad de ajustar su aplicación en aquel punto concreto que necesite, sin afectar a otras partes que vea más complejas y personalmente intratables.

I.4. Contribuciones

Las principales contribuciones que hemos querido aportar con este trabajo son:

- Hemos identificado la ausencia de una definición clara del concepto de aplicación paralela y, por tanto, de la existencia de un objeto o abstracción a nivel de sistema operativo o de usuario que ayude a su gestión. Vemos la necesidad cada vez mayor, mostrada por otra parte en definiciones intuitivas e implícitas en los trabajos actuales que versan sobre el tema, y proponemos una primera aproximación y estudio. Conseguimos así individuar el comportamiento propio de cada aplicación, aislándola a la vez del resto de aplicaciones del sistema. Es la unidad de planificación de trabajos, compuesta de *tasks* y *threads*, a la que se le asignan recursos de procesamiento y los puede sintonizar conforme se adecúe mejor a su perfil.
- Partiendo de los estudios ya realizados de planificación en multiprocesadores con memoria compartida, que proponen políticas de espacio compartido (subdivisión de la máquina con procesadores dedicados por aplicación), frente a los de tiempo compartido, mostramos que un algoritmo basado en dicha política es también el más apropiado en un entorno multiprocesador y multiusuario de propósito general. Hemos desarrollado utilidades (librerías y servidores) que permitan la comunicación entre el usuario y el sistema para manifestar las necesidades de la aplicación en cuanto al recurso procesador y facilitar, por otro lado, que la aplicación se adapte a las restriciones a las que le pueda someter el sistema. El entorno concreto en que hemos basado nuestras realizaciones ha sido el microkernel Mach 3.0.
- Proponemos un entorno para la planificación de flujos de usuario (*user level threads*), en el cual se encuentren no sólo las políticas habituales que podemos encontrar en una planificación más o menos compleja de sistema (prioridades, *round-robin*), sino también mecanismos actuales de los sistemas más modernos (*smart scheduling*, *handoff*, *hints*,...). Además, conseguimos que esta planificación no se base en ninguna funcionalidad del kernel, como ocurre en muchas librerías. Para ello, hemos modificado la librería de threads de usuario CThreads.
- Mostramos por último la necesidad de una cooperación entre el kernel y la aplicación para la gestión de flujos que sea más apropiada a cada aplicación, para lo cual hemos desarrollado un entorno de trabajo. Por parte del kernel, hemos construido mecanismos de *upcall* para que comunique a la aplicación aquellos acontecimientos que puedan suponer para ella una replanificación en el avance de su ejecución. Por parte de la aplicación, hemos añadido nuevas llamadas al

sistema que permitan definir los puntos de entrada de estas upcalls, y su intención de que se le comuniquen o no determinados eventos. Entre ellos está un temporizador para permitir políticas con desbanque (*round-robin*, *timeouts*) a nivel de usuario. Se han añadido módulos de planificación a nivel de usuario que pueden ser modificados por programadores más avanzados y permiten una total particularización de problemas. La comunicación entre el kernel y la aplicación se completa mediante "objetos de primera clase" y la resolución de las exclusiones mutuas y abrazos mortales que aparecen en un sistema de planificación en modo usuario, desbancable por las *upcalls*.

I.5. Organización del documento

El resto del documento lo hemos dividido en cuatro partes, cada una de ellas con sus correspondientes capítulos.

La primera parte es totalmente introductoria. En el primer capítulo vemos el estado actual de las arquitecturas y sistemas operativos multiprocesadores. Empezando por una visión amplia, global, de lo que se conoce por multiprocesador, vamos definiendo nuestro marco de trabajo hasta presentar la arquitectura base de nuestro diseño. Presentamos también el estado del arte en los sistemas operativos para multiprocesadores de memoria compartida y su evolución hasta la tecnología microkernel actual, con su filosofía de diseño. Por ser el tema de nuestro estudio, comentamos también la gestión de procesadores en multiprocesadores y las políticas de planificación de procesadores de partición de la máquina.

En el Capítulo 2, subimos al nivel de la aplicación. Veremos qué es la concurrencia y qué ventajas ofrece la programación concurrente; qué herramientas y utilidades permiten expresar la concurrencia de los programas y cómo de ella se puede explotar el paralelismo real. Volvemos al sistema operativo en cuanto a soporte de paralelismo para la aplicación en su planificación de flujos. Vemos mecanismos y políticas se están desarrollando e investigando en la actualidad para mejorar el rendimiento de las aplicaciones paralelas. Nuestro objeto de estudio ha sido siempre la planificación de flujos y los factores que influyen en ella: sincronización, comunicación y políticas de desbanque, principalmente.

La segunda parte es la parte central de desarrollo de nuestras aportaciones. Empezamos con el estudio del concepto de aplicación paralela, partiendo de la definición de aplicación, más o menos implícita que se entrelee en otros trabajos actuales. Hacemos una pequeña disertación para ver a qué nivel corresponde su tratamiento como objeto. Acabamos este capítulo proponiendo el soporte a la planificación que creemos conveniente, sin añadir complejidad al modelo de programación elegido, y que se verá en los siguientes capítulos.

En el Capítulo 4 presentamos la librería CThreads$^+$ como extensión de la librería CThreads, con nuevos mecanismos y políticas de planificación a nivel usuario. A partir de ella, la aplicación adquiere una potencia de planificación adaptable al perfil de sus trabajos, equivalente a la de los sistemas operativos más actuales. Se consigue además independizarla de todo soporte con utilidades del sistema.

En los capítulos 5 y 6 se desarrolla el entorno de trabajo que hemos diseñado para que la aplicación tenga acceso a los procesadores físicos y gestione sus flujos. Son los capítulos centrales del trabajo. El Capítulo 5 va mostrando la filosofía de diseño que nos ha ido guiando en cada una de las decisiones de realización. El Capítulo 6 plantea las concreciones del desarrollo, tanto a nivel del módulo planificador de la aplicación, como de las partes implicadas en el sistema para mantener actualizada la información de la aplicación.

En la tercera parte, evaluamos el entorno de planificación realizado. Detallamos también

las herramientas software y hardware que hemos utilizado y las llamadas al sistema que hemos añadido para dar soporte y poder ajustar los parámetros del sistema que hemos desarrollado.

A partir de los resultados obtenidos, comentamos las conclusiones que hemos extraído y proponemos algunas líneas para continuar el trabajo realizado. Entre ellas, el estudio del concepto de aplicación en más profundidad, para tener en cuenta en el futuro diseño de microkernels.