NNOS: UN MICROKERNEL DISTRIBUIDO PARA DOCENCIA E INVESTIGACIÓN.

Enric Gibert[1], Pere Creus[1], Miquel Nicolau[1]

[1]*Enginyeria i Arquitectura La Salle - Universitat Ramon Lull:*
{enric,perecb,miquelN}@salleURL.edu

RESUMEN: el Grupo de Sistemas Operativos de *Enginyeria i Arquitectura La Salle* está diseñando e implementando un *microkernel* distribuido llamado NNOS como herramienta de docencia e investigación. Este proyecto se encuentra en una fase inicial, si bien ya hay partes completamente operativas. El marco actual nos hace ser optimistas y pensar que a finales de este año dispondremos de una versión local totalmente estable y a finales del curso académico 2001-2002, un prototipo distribuido totalmente funcional.

1.- INTRODUCCIÓN

El Grupo de Sistemas Operativos pertenece a la sección de Arquitectura de Computadores del Departamento de Informática de *Enginyeria i Arquitectura La Salle*. Este grupo está formado por profesores, proyectistas de primer y segundo ciclo y estudiantes colaboradores. Los integrantes son responsables de las áreas de docencia e investigación relacionadas con los sistemas operativos.

Dentro del área docente, el grupo es responsable actualmente de dos asignaturas: *Sistemas Operativos* y *Sistemas y Entornos Distribuidos*. La primera es una asignatura que se imparte en tercer curso de Ingeniería Técnica en Informática de Sistemas. Las clases teóricas se complementan con cuatro prácticas: una en la que los alumnos se familiarizan con las llamadas a sistema de *Unix* y las otras tres se enfocan en la implementación de un pequeño micronúcleo llamado *Quark*. La asignatura de *Sistemas y Entornos Distribuidos* es una asignatura similar a la anterior que se imparte en tercer curso de Ingeniería Técnica Multimedia. Los fundamentos teóricos son similares a los de *Sistemas Operativos*, perfilando la asignatura de manera distinta para adecuarla a esta titulación. Ambas asignaturas tienen 3 horas semanales de clase y 9 créditos.

El proyecto *Quark* también se enmarca dentro de la área docente del Grupo de Sistemas Operativos dado que se usa como práctica para los alumnos de *Sistemas Operativos*. *Quark* es el diseño de los componentes básicos de un micronúcleo que los alumnos deben implementar. Los componentes que lo forman son: un pequeño gestor de interrupciones; un gestor de procesos,

encargado de ofrecer las abstracciones de proceso y *thread* e implementar un planificador preemtivo basado en prioridades; un gestor de entrada/salida; y finalmente, el mecanismo de comunicación entre procesos (IPC), que será un mecanismo de paso de mensajes basado en puertos. La práctica se divide en tres entregas debido a su carga, en las que se van implementando cada uno de los módulos.

La segunda área de la cual es responsable el Grupo de Sistemas Operativos es la de investigación. En estos momentos se trabaja en dos proyectos distintos: el proyecto NNOS y el proyecto OSEK. Del primero hablaremos en más detalle puesto que es el objetivo de esta ponencia. El proyecto OSEK, por otro lado, se basa en el diseño e implementación de un sistema operativo basado en el estándar OSEK/VDX. OSEK (*Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug*) [1] es una especificación de varias empresas (entre ellas BMW, Renault, Volkswagen y Volvo) de un sistema operativo de tiempo real para la automoción. Evidentemente, este diseño se puede extrapolar a cualquier entorno de control.

2.- NECESIDADES ACTUALES Y OBJETIVOS

Antes de iniciar el proyecto NNOS, se detectaron una serie de necesidades en el grupo las cuales dieron paso a este proyecto. En la área de la docencia falta un entorno de desarrollo completo y potente para que el alumno pueda tratar otras partes del sistema operativo. Con *Quark*, los alumnos no pueden trabajar con partes importantes de un sistema operativo como la gestión de memoria, el cargador, etc. debido a que empiezan el sistema desde cero y lo desarrollan bajo 80x86 en modo real. También interesa un entorno como este para las prácticas de futuras asignaturas avanzadas de sistemas operativos que se pretenden ofrecer en segundo ciclo. Además, este entorno se podría utilizar para investigación, puesto que si se dispone de un sistema operativo flexible y distribuido, puede ser fácilmente adaptado para temas de investigación más avanzados.

Dadas estas necesidades, se emprendió el proyecto NNOS con los siguientes objetivos:

1. Disponer de una herramienta de docencia adecuada para asignaturas introductorias y avanzadas de sistemas operativos. La herramienta también debería ser útil para la investigación.

2. La herramienta debe ser flexible facilitando la inclusión de nuevos módulos o políticas por parte de los alumnos e investigadores.

3. Todos los componentes del sistema deben ser fácilmente mantenibles, sacrificando, si fuera necesario, el rendimiento. Esta mantenibilidad implica que el código sea lo más claro posible y que todas las funciones y estructuras utilizadas se documenten exhaustivamente.

4. El sistema ha de ser portable a otras plataformas.

3.- VISIÓN GLOBAL DE NNOS

NNOS es un sistema operativo multiproceso, *multithread* y distribuido, basado en una arquitectura micronúcleo. Está formado por un núcleo o *kernel*, que es el encargado de proporcionar los mecanismos básicos necesarios, y un conjunto de procesos servidores, que se ejecutan en modalidad usuario y que proporcionan parte de la funcionalidad del sistema

operativo [2]. Dado que el núcleo de NNOS no implementa políticas, será necesario que sean los servidores los que las definan e implementen, lo que implica una gran flexibilidad.

Dentro del núcleo podemos distinguir varios módulos diferenciados como son el gestor de comunicación entre procesos (IPC), el gestor de procesos (PH), el gestor de interrupciones (IH), el gestor de memoria (MEM), entre otros. Cada uno de estos módulos, sin embargo, está dividido en dos capas:

- **Parte dependiente del Hardware o HDL**: es la parte del núcleo del sistema operativo que interacciona con la máquina y que, por lo tanto, hay que modificar dependiendo de la plataforma sobre la cual se ejecute NNOS.

- **Parte independiente del Hardware o HIL**: es la parte del núcleo que utiliza las abstracciones que ofrece el HDL para realizar operaciones de más alto nivel.

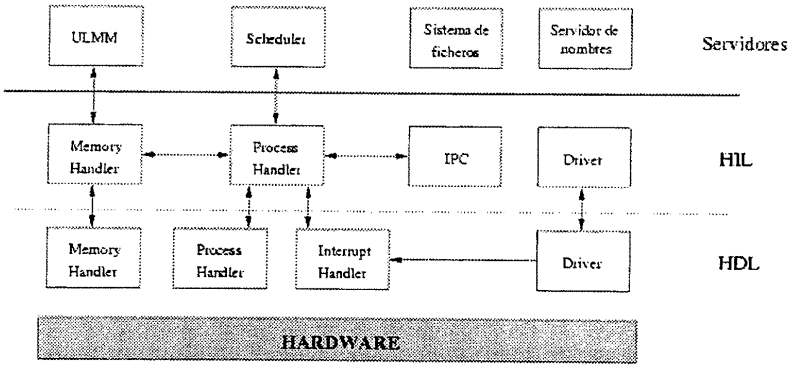

Figura 1. Modelo del Sistema Operativo NNOS.

NNOS responde a las siglas de *No Name Operating System*. Aunque en esta primera versión el sistema operativo no es aún distribuido, todas las estructuras subyacentes están pensadas para que, en un futuro, se puedan utilizar de forma distribuida. Lo que da a NNOS la apariencia de sistema operativo distribuido es la transparencia en las llamadas a sistema. Existe un único conjunto global de llamadas a sistema común para todas las máquinas que responden todas las peticiones del sistema, tanto locales como las remotas.

4.- MÓDULOS DEL MICROKERNEL

A continuación presentamos, con un poco más de detalle, los componentes que forman el *microkernel*.

a) Gestor de interrupciones

El gestor de interrupciones es el encargado de ejecutar la rutina de servicio de interrupción (RSI) adecuada según el tipo de interrupción que se haya producido. Este módulo ofrece servicios a nivel de *kernel* para asociar y desasociar rutinas RSI a interrupciones.

b) Gestor de procesos

NNOS ofrece las abstracciones de tarea y *thread* a nivel de *kernel*. Una tarea es un contenedor de recursos (espacio de memoria y puertos) donde se ejecutan uno o más *threads*. Sin embargo, NNOS no ofrece ninguna política de planificación a nivel de *kernel*, sinó que éstas se implementarán mediante uno o más procesos servidores planificadores (llamados planificadores de alto nivel o simplemente planificadores).

Para conseguir esto, NNOS divide el ciclo de planificación en un número determinado de intervalos de tiempo fijo llamados *slots* que se repiten cíclicamente. Estos *slots* se dividen en unidades de ejecución más pequeñas (*thread slots*) y tienen una cola de *threads* asociada con las hebras que deberán ejecutarse durante el tiempo que dura el *slot*. Los planificadores se registrarán en uno o más de estos *slots* teniendo en cuenta que un *slot* sólo podrá pertenecer a un único planificador. En dichos *slots* el planificador decidirá qué política utilizará (y por tanto, qué *threads* deberán ejecutarse a continuación) actualizando su cola correspondiente.

El mecanismo implementado internamente se basa en ceder el procesador al primer *thread* de la lista correspondiente al *slot* actual durante los *k thread slots* asignados por el planificador a dicha hebra. Una vez que se haya agotado este tiempo o que el *thread* en ejecución se bloquee, se cede el procesador al siguiente. Cuando la cola queda vacía o se produce un acontecimiento interesante para el planificador (p.e. se despierta un *thread*), se cede el procesador al *thread* del planificador para que llene de nuevo la cola o la modifique.

c) Gestor de *capabilities*

Todos los recursos (tareas, puertos, regiones de memoria...) de NNOS se identifican con *capabilities* [3]. Una *capability* está compuesta por el nombre o identificador del recurso, los permisos sobre éste y finalmente una firma digital para protegerlas ya que residen en espacio de usuario. Cuando una tarea de usuario crea un recurso, el *kernel* le devuelve una *capability* con todos los permisos activos que lo identifica. A partir de ésta, la tarea puede derivar otras *capabilities* que hagan referencia al mismo recurso con distintos permisos.

El módulo de *capabilities* es la parte del *kernel* encargada de crear, comprobar y destruir las *capabilities* de los recursos. Este módulo es usado por los demás componentes del *kernel*. Por ejemplo, cuando un usuario crea una tarea, realiza una llamada a sistema que es servida por el gestor de procesos y éste acaba llamando al gestor de *capabilities* para crear una *capability*. Cuando un usuario haga una llamada a sistema para suspender una tarea especificando la

capability de ésta, el gestor de procesos pedirá al gestor de *capabilities* que la compruebe y verifique que tiene permisos para dicha acción.

d) Gestor de comunicación entre procesos (IPC)

El mecanismo de comunicación entre procesos que ofrece el *kernel* se basa en el paso de mensajes a través de puertos. Existen dos tipos distintos de puertos: los puertos normales y los puertos de *send/receive*. Los primeros son puertos en los que se pueden dejar mensajes (llamada a sistema *send*) o recojerlos (llamada a sistema *receive*), mientras que los segundos no pueden ser accedidos con un *send* sinó con una operación *send/receive* que combina ambas acciones dentro del *kernel*. Este segundo tipo de puertos es similar al mecanismo de llamada remota a procedimientos (RPC) [4] y son necesarios en una arquitectura *microkernel* puesto que la mayoría de servicios son ofrecidos por procesos de usuario. La comunicación es totalmente síncrona (estilo *rendez-vous*) y los mensajes son de tamaño variable.

El gestor de comunicación entre procesos también ofrece eventos como un segundo mecanismo de comunicación. Un evento es un acontecimiento que altera temporalmente el flujo de ejecución de un *thread* concreto. NNOS ofrece dos tipos de eventos: síncronos y asíncronos. Los primeros suceden en un instante concreto (como por ejemplo la división por cero, un fallo de página...) y la rutina asociada a ellos se ejecuta en el contexto del *thread* que lo ha producido. Los eventos asíncronos, por su parte, no se producen en instantes concretos y su significado lo definen las dos tareas involucradas (vendrían a ser como los *signals SIGUSR1* y *SIGUSR2* de Unix). La ejecución de la rutina asociada se realiza en el contexto de cualquier *thread* de la tarea receptora.

e) Gestor de memoria

El gestor de memoria de NNOS ofrece las funcionalidades básicas de gestión de memoria que se complementan con funcionalidades y políticas más complejas a través de gestores externos llamados ULMM (*User Level Memory Manager*). Las dos abstracciones básicas definidas por el *kernel* son las regiones de memoria (*mem_regions*, o secciones contiguas del espacio lógico de una tarea) y las páginas de memoria (*mem_pages*). NNOS ofrece llamadas para reservar espacio lógico para una tarea (crear una *mem_region* que se identificará con una *capability*), reservar páginas físicas y mapear o desmapear páginas.

Los gestores externos serán los encargados de aplicar las funcionalidades para compartir memoria y ofrecer memoria virtual. Para ello, se ha diseñado un ULMM por defecto que define la abstracción de objeto de memoria (*memory_object*) o conjunto de páginas físicas. Este proceso ofrece el servicio de asociar una *mem_region* a un *mem_object*, lo que implica la asociación de páginas lógicas a físicas. Si dos tareas distintas se mapean el mismo *memory_object* compartirán memoria.

f) Servidores a nivel de usuario

También se ha pensado inicialmente en dos procesos más de usuario (a parte del planificador de alto nivel y del gestor de memoria ULMM) necesarios para el funcionamiento básico de NNOS en su primera versión. Estos procesos son: un servidor de nombres, para poder obtener fácilmente referencias a recursos; y un servidor de ficheros y cargador para poder ejecutar procesos dinámicamente.

5.- CAPACIDADES DISTRIBUIDAS

Uno de los primeros objetivos de NNOS es que sirva como base para un sistema operativo distribuido. De hecho, se añade dicha funcionalidad permitiendo que las *capabilities* no sólo hagan referencia a recursos locales, sinó que pueden identificar recursos remotos de manera transparente al usuario.

Así pues, cuando se hace referencia a un recurso remoto (por ejemplo se hace un *send* a un puerto de una tarea remota), el gestor de *capabilities* ha de poder detectar la ubicación de este recurso y encaminar la petición a su destino de manera totalmente transparente al usuario. Lo mismo sucede con el resto de recursos identificados con *capabilities* como las tareas, regiones de memoria, ...

6.- ESTADO ACTUAL DEL PROYECTO

El proyecto NNOS se encuentra en un estado inicial, si bien empieza a tener partes totalmente operativas. Actualmente tenemos implementado el gestor de interrupciones, el gestor de procesos, el gestor de IPC (si bien se está rediseñando este), parte del gestor de *capabilities* y el gestor de memoria para arquitecturas 80x86 en modo protegido [5]. Estos módulos han sido implementados por proyectistas de primer ciclo. En fase de desarrollo tenemos el servidor de ficheros y cargador, el servidor de nombres, el diseño de la fase de arranque del sistema y algún que otro *device driver*. Finalmente, estamos diseñando el mecanismo DIPC (*Distributed IPC*) para tener un sistema operativo totalmente distribuido y un gestor de memoria ULMM que ofrezca memoria compartida bajo arquitecturas multicomputador.

La planificación que tenemos para este proyecto incluye la finalización total del prototipo local (sin ningún mecanismo distribuido) para finales de año y tener un prototipo distribuido para finales del curso académico siguiente.

REFERENCIAS

[1] *OSEK/VDX. Operating System specification 2.1*. Disponible en http://www.osek-vdx.org/.

[2] William Stallings. Operating Systems: internals and design principles. Prentice Hall, 1998.

[3] A. S. Tanenbaum, S. J. Mullender, R. Van Renesse. *Using Sparse Capabilities in a Distributed Operating System*. Proc. Of the 6th Int. Conference on Distributed Computing Systems, IEEE Press, 1986.

[4] Pradeep K. Sinha. *Distributed Operating Systems. Concepts and Design*. IEEE Computer Society Press, 1996.

[5] Pentium Pro Family Developer's Manual. Volume 3: Operating System Writer's Guide. Intel Corporation, 1995.