Un modelo de memoria virtual compartida distribuida para Mapaná

Alba Eugenia Urrea Cardozo*
Rafael Gómez**

* M.Sc. - Profesora Escuela de Ingeniería de Sistemas y Computación - Facultad de Ingeniería - Universidad del Valle - Santiago de Cali,Colombia. e-mail:albaeu@univalle.edu.co

** D.E.A. (Francia) - Ingeniero de Sistemas y Computación - Profesor Departamento de Ingeniería de Sistemas y Computación - Facultad de Ingeniería - Universidad de los Andes - Bogotá, Colombia. e-mail:ragomez@uniandes.edu.co

Fecha de recepción: Febrero 4 de 2005
Fecha de aprobación: Junio 10 de 2005

RESUMEN

Este artículo describe el diseño e implementación de un modelo de memoria virtual compartida distribuida para Mapaná. Mapaná es una plataforma de máquina virtual paralela que se ejecuta en computadores homogéneos conectados en red. El modelo provee la creación de un espacio de direcciones lógico, único para cada máquina virtual; así, los procesos que pertenecen a una determinada máquina virtual tienen el mismo espacio y pueden compartirlo, a pesar de estar en máquinas físicas diferentes. Se tiene un proceso origen, o raíz, que, al ser replicado en otros nodos de la red, da lugar a los procesos homólogos o parientes con idénticos

espacios de direcciones.

El modelo se implementó en Linux, para lo cual se modificó el sistema de memoria virtual, extendiéndolo para incluir el concepto de páginas remotas y la administración de las mismas. Se describe la interacción entre módulos y programas de usuario implementados para llevar a cabo la replicación de un proceso y la ocurrencia de un fallo de página remoto.

Palabras clave: memoria virtual compartida distribuida, proceso raíz, proceso homólogo, página remota, fallo de página.

ABSTRACT

This paper describes the design and implementation of a distributed and shared virtual memory model for Mapaná. Mapaná is a parallel virtual machine platform that executes in a homogeneous computer network. The model provides creation of a logic address space that is unique for each virtual machine, so, processes belonging to a certain virtual machine have the same space and they can share it, in spite of being in different physical machines. There is an origin or root process that when it is replicated in other network nodes, it gives rise to homologous or relative processes with the same address space.

The model was implemented in Linux, modifying its virtual memory system, extending it to include the concept of remote pages and of their administration. The interaction between modules and user programs implemented to carry out the replication of a process and the occurrence of a remote page fault is described.

Key words: distributed shared virtual memory, root process, homologous process, remote page, page fault.

1. INTRODUCCIÓN

Ballesteros [Ballesteros, 1998], en su tesis doctoral en informática, afirma que la administración moderna de memoria desea tener simultáneamente el mayor número posible de procesos, por ello el espacio físico disponible se divide entre más procesos. Esta frase hace considerar la importancia de la unión, en un modelo de memoria para una plataforma paralela, de las tres características que componen el titulo de este artículo: virtual, compartido y distribuido, para adoptar un esquema eficiente.

Al ser virtual, se dispone de más memoria de la que realmente se tiene, permitiendo ejecutar aplicaciones muy grandes, y asegurando que todos los procesos de una misma máquina virtual hagan referencia al mismo contenido de memoria física a través de sus direcciones virtuales, ya que las direcciones físicas pueden ser diferentes.

Por manejar un rango idéntico de direcciones virtuales, también llamadas lógicas, la memoria se comparte entre procesos de diferentes nodos como si se tratara de un proceso padre clonado en un mismo nodo.

Y, finalmente, la distribución le brinda a los procesos el beneficio de tener sólo las páginas de memoria que necesitan para ejecutarse y obtenerlas a través de cualquier otro proceso que comparte el código y datos.

El modelo de memoria propuesto para Mapaná bajo Linux reúne las ventajas de estas tres características; así, cuando se crea un proceso raíz, sus regiones de memoria son replicadas en los procesos parientes.

Cuando ocurre un fallo de página, si es local, se sigue el procedimiento normal de Linux; es decir, verificar si la página nunca ha sido accedida (función do_anonymous_page), o si la operación nopage de la región de memoria a la que pertenece la página faltante está permitida, o, por último, si fue llevada a disco. Si se trata de un

fallo remoto, debe suministrarse la información necesaria que permita identificar de dónde proviene el fallo (proceso + máquina virtual) y qué página es requerida. Con esta información, y por medio de módulos (programación externa a nivel de kernel que aumenta la funcionalidad de éste), se obtienen y agregan páginas al espacio de direcciones de los procesos de una máquina virtual. Existe un administrador central para ejecutar estas dos operaciones con las páginas y para mantener actualizados los propietarios de cada una de ellas.

A continuación se describe el diseño y la implementación de un modelo de memoria virtual compartida distribuida para Mapaná.

2. MODELO DE MEMORIA VIRTUAL COMPARTIDA DISTRIBUIDA MVCD- PARA MAPANÁ

El modelo MVCD pretende ser la base para el administrador de memoria remota para Mapaná. Para su funcionamiento es indispensable la creación de los procesos tanto raíz como parientes.

El proceso raíz es un proceso de Linux, (creado con el Fork implementado por dicho sistema operativo), que ejecuta un programa de usuario. La información sobre este proceso disponible en el kernel (por ejemplo, descriptor de proceso, descriptor de memoria), sirve para generar replicas de él en los nodos de la red sobre los que se ejecuta una máquina virtual determinada. Estas replicas aparecen cuando se ha creado el proceso raíz y son llamadas procesos parientes u homólogos. Un proceso pariente se ejecuta paralelamente al proceso raíz pero en otra máquina. Tanto raíz como parientes acceden al mismo espacio de memoria virtual y por ello comparten código y datos. Este espacio es generado por el sistema operativo cuando está creando al proceso raíz.

Cuando se están ejecutando dichos procesos, ellos acceden continuamente a localizaciones de memoria; si estas localizaciones se refieren a páginas que no están localmente, se produce un fallo de página remota y se eleva una solicitud a una entidad central que conoce la ubicación de la página y puede recuperarla. De esta manera, las páginas migran de acuerdo con su demanda. Se espera que, ante un fallo remoto, la entidad central o servidor de páginas sea capaz de administrar correctamente la memoria virtual compartida distribuida de los procesos parientes.

2.1. Arquitectura global del sistema

Mapaná es una plataforma virtual que involucra computadores homogéneos en red, concebida para ofrecer servicios de manejo de memoria, de procesos y de E/S. La idea fundamental es que, para un usuario de esta plataforma, los recursos sean suministrados transparentemente durante el lanzamiento de los procesos en paralelo, por lo que el usuario puede trabajar como si se tratara de un multiprocesador y no de una red.

En este sistema participan un proceso raíz, uno o varios procesos parientes y un servidor de páginas.

El servidor de páginas se ejecuta en la misma máquina física o nodo de red donde se crea el proceso raíz; su función básica es la administración de las páginas remotas de una máquina virtual. Este servidor es centralizado, sólo se tiene uno por nodo. Si se tienen varios procesos raíz en un mismo nodo, lo cual significa diferentes máquinas virtuales (ya que existe un proceso raíz por máquina virtual), el Servidor de Páginas es único y trabaja para todos ellos. Pero, si existen varios procesos raíz en diferentes nodos, en cada nodo se tiene un Servidor de Páginas diferente.

El Servidor de Páginas implementa el algoritmo de Servidor Central con migración [CNE Tutorial Modules Central]. Por su sencillez de implementación, el riesgo que se tiene de caída de la plataforma es el mismo que si se tuvieran

varios servidores para una sola máquina virtual; la razón es que cualquier nodo que quede fuera entorpece la ejecución de todos los procesos de la máquina virtual. En la figura 1 se representa la interacción entre los procesos raíz y parientes, para petición de páginas.

En la red se pueden ejecutar varias máquinas virtuales. Al levantamiento de una máquina virtual, el proceso raíz es quien tiene páginas en memoria y en disco; los parientes tienen todas las páginas marcadas como remotas y pasan a locales de acuerdo a la demanda de ellas.

Sin embargo hay páginas que siempre son locales y no pueden migrar, son las que representan la parte del sistema operativo e información de la plataforma Mapaná relacionada con el proceso o la máquina local. En este artículo no se trata este aspecto puesto que es completamente dependiente de Mapaná.

Una página puede estar presente en memoria física o en memoria secundaria, pero, para adoptar un modelo de memoria virtual compartida distribuida MVCD-, se necesita de un tercer estado llamado Remoto. Una página en estado Remoto no está localmente, ni en memoria ni en disco, la petición se debe dirigir a quien conozca el paradero de cada página (el servidor de páginas).

Tras un primer fallo de página, el proceso raíz posiblemente habrá otorgado la página y su entrada respectiva en la tabla de páginas cambiará a estado remoto.
MVCD adopta unas políticas sencillas para la migración de páginas:

- ☑ Si se trata de una página con permiso de escritura, la política es liberar el marco de página que la contiene; el proceso propietario que cede dicha página debe actualizar la respectiva entrada en su tabla de páginas a estado remoto. El servidor de páginas debe actualizar la información sobre el nuevo proceso propietario.
- ☑ Si se trata de una página con permiso de lectura, se permite simplemente una copia del contenido de ella. No hay implicaciones para el proceso propietario actual; sólo se indica el estado Presente de la página en ambos procesos: el propietario y el solicitante.

El procedimiento general para solicitar una página al presentarse un fallo remoto implica que:

- ☑ El proceso solicitante debe realizar la petición al servidor de páginas.
- ☑ El servidor de páginas debe conocer el actual proceso propietario de la página y su nodo en la red.
- ☑ El actual proceso propietario de la página debe retornar la página al servidor de páginas.
- ☑ El servidor de páginas puede entregar la página al proceso solicitante.

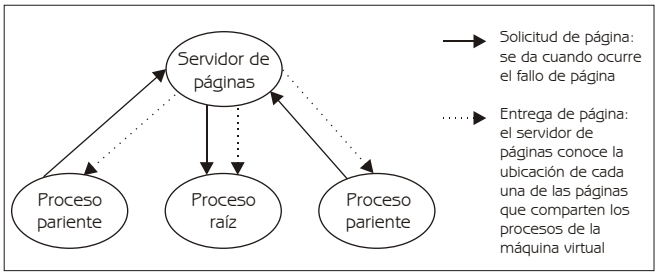

Figura 1. Esquema de petición de páginas

2.2. Implementación del sistema

El esquema de trabajo planteado para implementar un modelo de memoria virtual compartida distribuida se hizo por medio de módulos Linux. La idea principal fue manejar este modelo de memoria como una parte extensible del kernel de dicho sistema operativo y evitar así su continua recompilación. Además se pretendía que dicho kernel tuviera el mínimo posible de cambios para no afectar su estabilidad.

Por ello, y de acuerdo con estos antecedentes, se tienen actualmente varios módulos que pueden cargar este modelo de memoria en cualquier momento sin impedir la ejecución normal del modelo de memoria implementado por Linux. Las aplicaciones que deseen hacer uso del modelo de memoria virtual compartida distribuida deben acogerse a una serie de funciones,

implementadas en los módulos mencionados; en caso contrario, se estará utilizando el modelo de memoria Linux.

Para la implementación del modelo MVCD para Mapaná se crearon nuevos syscalls por medio de la tabla sys_call_table del kernel ubicada en el archivo entry.S del directorio /usr/src/linux-2.4/arch/i386/kernel. Los syscalls se encuentran implementados en cuatro módulos Linux: dos del lado servidor y dos del lado cliente.

Así mismo, es importante adicionar los nombres de algunas funciones y objetos del kernel para que los módulos puedan utilizarlos. Estos datos se ponen en el archivo ksyms.cr. Luego de realizadas estas modificaciones, el kernel debe recompilarse por primera y única vez.

Linux cuenta con un sistema de archivos denominado /proc encargado de llevar hacia el exterior los datos del kernel, que, de otra manera, no se podrían dar a conocer al usuario. Esta estrategia es útil para acceder a las estructuras de datos del sistema operativo desde un programa de usuario.

2.2.1. Creación de proceso pariente:

Cuando es necesario replicar el proceso raíz, su identificador se guarda en una estructura de tipo tarea, que a su vez será llevada al /proc. El módulo transmisor, a partir de este identificador, construye el descriptor que deberá recibir el módulo receptor para dar origen a un proceso pariente. La estructura tarea está definida por:

```
struct tarea{
        pid_t pid; }
```

El descriptor de proceso a construir es una estructura del tipo process_data que a su vez almacena otras estructuras:

```
Struct process_data{
        struct vmas vm[MAX_MAP_COUNT];
        struct paginas_reservadas
        pag_reservadas[MAX_PAG_RESERVADAS];
        struct task_struct tsk;
        struct mm_struct mm;
        int tamano_vmas;
        int tamano_pag_reservadas;
        void *old_ldt;
        int mv; };
```

La estructura vmas contiene la información de las regiones de memoria, y paginas_reservadas es otra estructura que guarda las páginas reservadas del proceso, sus definiciones están dadas en términos de los siguientes campos:

```
struct vmas{
        unsigned long flags;
        unsigned long start;
        unsigned long end;
        pgprot_t prot; };

    struct paginas_reservadas{
        unsigned long address;
        struct page * pagina; };
```

Los procesos de una máquina virtual tienen un espacio privado y uno compartido. El espacio privado corresponde a las páginas reservadas; éstas nunca tienen estado remoto, solamente migran cuando el proceso pariente es creado. El espacio compartido está representado por las páginas remotas.

Una vez se crea el proceso raíz, el identificador asignado por el sistema operativo se coloca en el /proc por medio de la estructura "tarea", ya descrita. Antes de enviar a un nodo el descriptor de proceso para la replicación, se recorre su espacio de direcciones virtual y la descripción de cada región componente se almacena en vmas; se toman sus permisos de acceso y su intervalo de direcciones, igualmente se tienen en cuenta las páginas reservadas que se agregan en paginas_reservadas. El descriptor de envío requiere ahora de la Tabla de Descriptores Local LDT y de la máquina virtual a la que va a pertenecer el proceso replicado. Una vez obtenido el descriptor del proceso raíz, el módulo Fork[1] crea la lista de páginas remotas, dicha lista

[1] Este módulo contiene la funcionalidad para crear un proceso pariente con respecto a la información recibida del descriptor del proceso raíz, y para agregar y eliminar páginas de memoria del proceso pariente.

será necesaria para el servidor de páginas en el seguimiento de la ubicación de cada página en la red y en la respuesta a las peticiones de memoria requerida.

La estructura para páginas remotas contiene el identificador del proceso propietario y el identificador del proceso bajo el cual se creó el espacio de direcciones lógico, la dirección virtual a la que hace referencia, el índice de la página dentro del espacio del proceso y la máquina virtual:

```
static struct {
        Unsigned long address;
        pid_t pid;
        pid_t pid_owner;
        int vm;
        int num;
    }Paginas_remotas[NUM_PAGINAS_REMOTAS];
```

Luego es necesario incluir el proceso en la máquina virtual. Para esto se tiene una estructura llamada lista_maquinas que mantiene la información útil de reconocimiento, no sólo de los procesos que pertenecen a una máquina virtual, sino también de los nodos en los cuales se ejecutan. Su definición es la siguiente:

```
static struct {
        int vm;
        char * host;
        pid_t pid;
    }Lista_maquinas[NUM_MAQUINAS_VIRTUALES];
```

Es inevitable hacer un paréntesis para describir brevemente el modelo de comunicación en la red. Básicamente se hace por medio de programas de usuario que implementan sockets, pero ellos están acompañados de dos módulos que, de igual forma, hacen de socket cliente y socket servidor; ésta es la manera en que se pueden recibir estructuras del kernel de nodo a nodo sin ningún inconveniente.

El módulo cliente implementa un syscall para el manejo de un socket cliente. Este syscall se activa en el momento en que el programa de usuario, que representa la parte visible del sistema, lo invoca para generar un proceso pariente.

Continuando con la creación de un proceso, una vez el nodo destino obtiene el descriptor de tipo process_data, invoca un fork normal de Linux y luego modifica el espacio de direcciones virtual. El procedimiento que se sigue es tomar cada región de memoria y asignarle los datos del proceso raíz. A las páginas reservadas se les asignan marcos de página para que las contengan, hecho que no ocurre con una página identificada como remota.

En el módulo llamado Fork2, donde se tiene la funcionalidad para generar las replicas del proceso raíz (los procesos parientes), existe una función para hacer el recorrido de las regiones de memoria que deben componer el espacio virtual del proceso pariente. Cada una de las entradas de la tabla de páginas que apunta a este espacio, debe indicar el estado remoto; esto es necesario para conocer cuándo debe ocurrir un fallo de página remoto. Cuando se termina este procedimiento, el proceso queda listo para incluirse a la máquina virtual del proceso raíz que le dio origen, por ello su identificador es almacenado en el /proc. El programa de comunicación a nivel de usuario lee la entrada del /proc y obtiene el identificador, éste es enviado al nodo servidor para su inclusión en la lista de procesos de la máquina virtual determinada.

2.2.2. Ocurrencia de un fallo de página remoto:
Cuando una página no está presente en memoria, su bit de presencia está inactivo y se produce un fallo para obtenerla. Esta situación puede presentarse porque nunca antes se había accedido a la página, y esto se conoce porque todos los bits de la entrada de la Tabla de Páginas están inactivos. En tal caso, Linux invoca la función do_no_page para cargarla. Pero, si la entrada indica activo el bit Read/Write, se llama a

una nueva función llamada do_no_page_remoto que trata el fallo remoto. En la figura 2 se muestra la traducción de una dirección virtual remota.

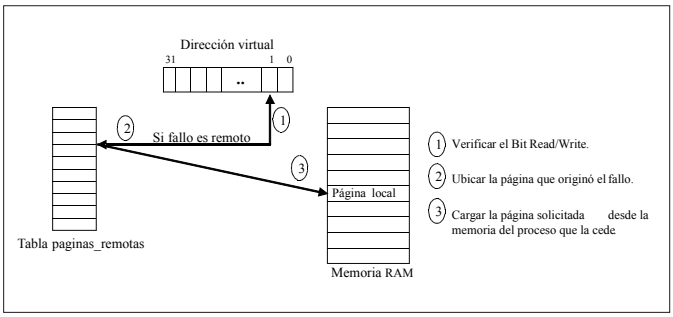

Figura 2. Traducción de una dirección remota

Otro caso de una página no presente puede deberse a que la entrada de la Tabla de Páginas contiene un índice que permite ubicarla en el área swap. La función para recuperarla es do_swap_page.

Un fallo de página remota en el modelo MVCD para Mapaná se detecta mediante el valor de los dos bits menos significativos de la dirección virtual. Estos bits corresponden al bit Present y al bit Read/Write. Este bit 2, el segundo menos significativo de los 32 bits de una dirección virtual, no tiene significado alguno cuando la página está en el swap y por tanto se puede aprovechar. Una entrada en área swap indica que los primeros 30 bits más significativos no son iguales a cero pero los 2 últimos bits sí, lo cual confirma la presencia de una página en el swap.

La función do_no_page_remoto permite poner, por medio del sistema de archivos /proc de Linux, la información del kernel referente al fallo de página remoto en un área accesible por una aplicación de usuario. Es necesario implementarlo de esta manera porque la parte de comunicación entre nodos se hace a través de programas a nivel de usuario que manejan sockets de comunicación.

Desde do_no_page_remoto se crea una entrada en el /proc que almacena el contenido de la estructura "pagina" y que representa un fallo remoto. Dado que el /proc se actualiza con esta estructura cada vez que ocurre un fallo, se debe leer continuamente para generar una petición de página. La estructura está definida así:

```
struct pagina{
        pid_t pid;
        int num; }
```

Las páginas reservadas de un proceso pariente nunca tendrán un estado remoto, como ya se afirmó, por lo tanto nunca harán uso de esta función.

Si la entrada en el /proc para el fallo de página contiene valores no nulos, se construye la petición y se envía al servidor de páginas.

El servidor de páginas busca en su lista de páginas remotas el identificador del proceso propietario, utilizando el identificador de máquina virtual y el número de página. Teniendo el identificador del proceso propietario, se conoce el nodo en el cual se ejecuta este proceso. Una vez identificados el proceso y el nodo propietarios, se entrega la dirección virtual equivalente a la función que retornará la página requerida. A partir de este momento se invocan las funciones propias del kernel de Linux para ubicarse en el espacio de memoria del proceso propietario, luego en la región apropiada y, siguiendo el orden, en el directorio global de páginas, en el directorio medio de páginas, en la entrada en la Tabla de Páginas y, por último, en la página. [Bovet & Cesati, 2001, 220]. Se revisa el tipo de permiso de la página para aplicar en ella las políticas de reemplazo que se describieron en párrafos anteriores. Ya en esta fase el servidor de páginas tiene la página y sólo le resta enviarla al proceso y nodo solicitantes. Alternadamente al envío de la página se hace una verificación de la política aplicada para actualizar los datos del propietario en la lista de páginas remotas, dado el caso de necesitarlo.

Si la página pedida no está presente en memoria en el proceso propietario, se invocan las funciones del fallo de página local para recuperarla.

Cuando el proceso y nodo solicitantes reciben la página, se ejecutan la misma serie de pasos que para obtenerla; es decir, ubicarse en el proceso, luego en su espacio de memoria, después en la región afectada y así hasta llegar a la Tabla de Páginas. Se solicita un marco de página para almacenar la página allí. La copia local de la página para el proceso solicitante se traduce en modificar la entrada correspondiente en la Tabla de Páginas a estado Presente y con los permisos de acuerdo con la región, con lo cual termina el proceso de migración de una página.

3. PRUEBAS REALIZADAS

Las pruebas realizadas están dirigidas principalmente a verificar la actualización del espacio de memoria de cada proceso perteneciente a una determinada máquina virtual por medio de las peticiones de páginas de un proceso pariente. Estas pruebas se refieren a la actualización de:

- ☑ La entrada de la tabla de páginas, cada entrada indica si una página es local o remota.
- ☑ El contenido de cada página y
- ☑ El propietario de cada página (proceso).

Para verificar lo anterior, es necesario acceder al espacio de memoria del proceso raíz y de cada proceso pariente, esto es a través de las funciones del kernel que son invocadas en los módulos propuestos para MVCD.

El proceso de configuración para cargar el modelo de memoria virtual compartida distribuida es sencillo; como paso previo se deben agregar unas entradas en el archivo de funciones exportables del kernel llamado ksyms ubicado en el directorio /usr/src/linux-2.4/kernel/. De igual modo, se deben agregar nuevas entradas a la tabla del kernel sys_call_table (se encuentra en /usr/src/linux-2.4/arch/i386/kernel/entry.S). Estos archivos se obtienen ya modificados y se copian en los respectivos directorios. Después de terminado este paso, es necesario recompilar el kernel para que tome los cambios. Con la nueva versión del kernel, se pueden cargar los módulos y demás programas.

Ahora es importante determinar un nodo en la red que sea servidor y otro que sea cliente para ejecutar los respectivos módulos así:
Del lado cliente, se carga el módulo que contiene la funcionalidad para crear un proceso pariente con respecto a la información recibida del descriptor del proceso raíz y para agregar y eliminar páginas de memoria del proceso pariente. Luego, se carga el módulo que establece la comunicación para recibir dicho descriptor.
Del lado servidor, se carga el módulo que contiene la funcionalidad para la administración de las páginas remotas a partir de la información brindada por el descriptor de memoria del proceso raíz; también hay funcionalidad para agregar y eliminar páginas del espacio de memoria del raíz. Posteriormente, se carga el módulo que realiza la comunicación para enviar el descriptor de proceso raíz. También de este lado, se carga el programa de prueba.

Para generar un fallo de página, se ejecuta desde el servidor y/o cliente, un programa de usuario que solicita determinada página; ella es representada por un número único dentro de la máquina virtual. Al solicitar la página, se está invocando explícitamente la función del manejo de fallo de página implementada por Linux, pero para efectos de probar el modelo, se ha llevado su código a los dos módulos encargados de la funcionalidad de la administración de páginas y de creación de procesos parientes, ya descritos. Mediante este procedimiento de solicitud de páginas se están generando fallos de página remotos simulados.

Los programas usuario que están invocando funciones de los módulos implementados,

Muestran resultados en pantalla como: qué procesos se crean, qué páginas migran y sus contenidos.

3.1. Ejecución de pruebas:

1. Verificación del esquema de comunicaciones entre equipos para la replicación del proceso raíz.

En esta parte se prueba la comunicación por medio de sockets C que son llevados a programas de usuario. El programa de usuario para manejo de socket invoca a la función respectiva dentro del módulo, quien es el que realmente se comunica con el kernel para obtener información y hacer la copia de descriptores.

Del lado cliente (nodo donde se crea un proceso pariente) se está a la espera de la información necesaria para replicar el proceso raíz. Para ello se define una nueva estructura cuyo contenido permitirá generar un proceso pariente con las características del proceso raíz: regiones de memoria, tabla local de descriptores, páginas reservadas y máquina virtual.

2. Construcción de la máquina virtual

Una vez creado el proceso raíz, se toma su identificador para ubicar su espacio de memoria y así recorrer sus regiones y construir la lista de páginas remotas.

También se construye la lista de páginas reservadas que se mantendrán locales sin posibilidad de tener estado remoto.

Luego se ingresa el proceso en la lista de máquinas, donde se tiene en cuenta su identificador, máquina virtual y dirección IP.

3. Construcción del espacio de direcciones de un proceso pariente.

A través de la comunicación realizada en el numeral 1 y tomando los valores de la estructura que contiene el descriptor de proceso raíz, se asignan las regiones de memoria al proceso pariente, se construye la lista de páginas remotas, se crea un proceso como tal y se retorna al nodo servidor el identificador que le fue asignado por el sistema operativo, este identificador permite ingresar el proceso pariente a la lista de procesos de la máquina virtual a la que debe estar asociado.

4. Verificación de peticiones de páginas entre procesos de una máquina virtual.

La función do_page_fault, definida en el kernel de Linux, es colocada en los módulos encargados de administrar las páginas de los procesos; así dicha función será invocada por los programas de usuario descritos.

La trama de transmisión entre estos dos programas de usuario, viene dada por una estructura que contiene el identificador de proceso solicitante de una página, el identificador del proceso propietario de la página, máquina virtual, dirección IP, número de página requerida y contenido de la página, una vez ella es ubicada.

Los módulos contienen funciones que se encargan de retornar el identificador del propietario de una página, el equipo propietario y hasta una dirección virtual de acuerdo con el número de ésta.

Como resultado de esta prueba, se ejecutan las funciones de un fallo de página como si fuera atrapado por la propia función del kernel. Además se creó y actualizó la entrada en el /proc con la información para ubicar la página requerida en la máquina virtual respectiva.

5. Actualización del contenido de las páginas de un proceso.

Todas las peticiones de página se dirigen al servidor de páginas. Una vez recibida una petición de página, el programa usuario para el manejo del socket servidor ejecuta una serie de pasos, ya mencionados, implementados como funciones dentro de los módulos.

Por medio del programa de usuario que solicita determinadas páginas, se pidió:

1. Una página de escritura en estado Presente en el espacio de memoria del proceso propietario.
2. Una página de lectura en estado Presente en el espacio de memoria del proceso propietario.
3. Una página en estado No Presente y mapeada anónimamente por el proceso propietario.
4. Una página de lectura en estado No Presente y obtenida por la operación nopage de la región de memoria a la que está asociada.

Las pruebas ejecutadas permiten observar los casos posibles de recuperación de una página cuando se produce un fallo de página remota. Esto hace referencia a que la página ya no está presente en memoria en el proceso que es dueño actual de ella; bien porque ha sido llevada a disco por Linux a través de su liberador de memoria, o incluso también porque tal página nunca antes había sido accedida. Al ser solicitada ésta por el servidor de páginas al proceso propietario, se genera un fallo de página local en este proceso (si se cumplen los casos mencionados) y se trata como tal. Una vez ejecutadas las funciones adecuadas al caso local que se presenta, la página es llevada a memoria y de acuerdo con las políticas de manejo para ella, puede ser eliminada o no de la memoria virtual a la que estaba vinculada hasta entonces. La página siempre es almacenada en una dirección física diferente a la que ocupaba en cada nodo, esto se confirma al leer sus atributos.

4. CONCLUSIONES

El modelo MVCD para Mapaná es una versión inicial para arquitecturas Intel x86. En esta versión no se tiene en cuenta aspectos de seguridad, pues no se incluyeron dentro de los objetivos. La meta principal alcanzada es un modelo funcionando, aunque el desempeño seguramente se verá afectado por el trabajo extra que tiene ahora el kernel para ubicar información remota. Sin embargo, como parte de una funcionalidad completa, se pueden implementar esquemas de seguridad y eficiencia deseados.

Como resultado del diseño e implementación de un modelo MVCD se tiene un administrador de memoria remota para la plataforma paralela Mapaná; siendo otro acercamiento en la construcción de sistemas operativos distribuidos.

La implementación del modelo de memoria virtual compartida distribuida se trabajó sobre la versión 2.4.19 del kernel, la más reciente al momento de inicio de esta investigación. La programación es a nivel kernel y a nivel de usuario. El código fuente fue escrito en lenguaje C. La parte kernel está representada en módulos de Linux que ofrecen entre otras ventajas las siguientes:

- ☑ La portabilidad a versiones posteriores del kernel en las que sus estructuras no sufran cambios drásticos, de esta manera el administrador de memoria puede tener un mayor ciclo de vida software.
- ☑ Menor dificultad para que el programador de la plataforma Mapaná ejecute la parte de memoria, pues basta con cargar los módulos.
- ☑ No se afecta la ejecución normal del sistema operativo al cargar o descargar el módulo porque se hace en caliente.
- ☑ La depuración es un proceso menos complicado, si se compara con hacerlo dentro del kernel.

Una característica fundamental del modelo MVCD está en que no es tarea del sistema operativo elegir dónde se distribuye un proceso, sino que el usuario elige dónde distribuirlo; esto no ofrece transparencia completa de la red, ya que deben conocerse los nodos que la componen.

Este trabajo de investigación ayuda a comprender el funcionamiento específico de un administrador de memoria y la complejidad de la localización en el espacio de memoria.

5. REFERENCIAS

1. Aivazian, Tigran. Agosto 2001. Dentro del núcleo Linux 2.4. http:// www.linuxdoc.org/ guides. html: 79p

2. Ballesteros, Francisco. Febrero de 1998. Off- Un nuevo enfoque en la construcción de sistemas operativos distribuidos. Universidad Politécnica de Madrid, España: 138p.

3. Ballesteros, Francisco. 1996. Advice: An Adaptable and Extensible Distributed Virtual Memory Architecture. In Parallel and Distributed Computing System.

4. Bovet, Daniel & Cesati, Marco. Enero 2001. Understanding the Linux Kernel. O'Reilly: 683p.

5. Burgos, Elkin. 2002. Modelo de Network RAM. Tesis de Maestría en Ingeniería de Sistemas. Universidad de los Andes. Bogotá

6. CNE Tutorial Modules Central. http://cne.gmu.edu/modules/dsm. Enero 2002.

7. http://www.cos.ufrj.br/~thobias/module/sockets. Enero 2002.

8. Kernel de Linux 2.4.19. 2002. Http://www.kernel.org

9. Riflet, Jean-Marie. 1993. Comunicaciones en unix. McGrawHill: 382p.

10. Rusling, David. Enero de 1998. El núcleo Linux.

11. Tanembaum, Andrew . Distributed Operating Systems. Prentice Hall.

12. Walton, Sean. 2001. Programación de socket Linux. Prentice Hall: 588p.