UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

U. A. C. P. y P. del C. C. H.

"DISEÑO Y DESARROLLO DE UNA EXTENSION A UN SISTEMA OPERATIVO PARA QUE MANEJE OBJETOS PERSISTENTES".

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

JUAN CARLOS PEREZ CASTAÑEDA

DIRECTOR:

Dr. Renato Barrera Rivera

MEXICO, D. F. 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Indice

Introducción

De unos años a la fecha se ha hecho mucho énfasis en el estudio de los sistemas que manejan objetos. El estudio se ha llevado a cabo en diferentes áreas del terreno de la computación, como por ejemplo: lenguajes de programación, bases de datos, arquitectura de computadoras, etcétera. A nivel nacional se ha venido desarrollando trabajo sobre "TM" [GERZSO 83] (un lenguaje orientado a objetos) [CARDENAS 86] y [JIMENEZ 86]. Otro ejemplo es el sistema de programación llamado "Smalltalk-80" [GOLDBERG 83]. Y son muchos los esfuerzos que se han desarrollado desde hace muchos años para implantar el manejo de objetos en el "hardware" de las computadoras [LEVY 84]. Para este trabajo de tesis los artículos del señor Atkinson y sus colaboradores resultaron muy importantes.

Una implantación como la que se presenta en esta tesis (extender un sistema operativo conocido para que maneje objetos persistentes), da la facilidad de hacer desarrollos específicos (ad hoc), o con propositos de investigación, a partir de la extensión. También permite hacer la evaluación del comportamiento de los métodos implantados, en este caso, en una arquitectura de computadora que no está diseñada para manejar objetos. Por ejemplo, manejadores de bases de datos que trabajen sobre objetos y lenguajes de programación en los que se puedan reconocer y hacer operaciones (como las que se listan más abajo) sobre objetos, entre otras cosas.

Que es un Objeto Persistente.

Después de que se ha diseñado un programa y que se pasa a la etapa de codificación, el manejo de los datos se tiene que hacer a través de las estructuras que proporciona el lenguaje de programación seleccionado. Dichas estructuras (y la información que contienen) existen mientras este activo el proceso que las esta usando. De la misma manera, todos los datos que se generan al ejecutar el programa se pierden si no se escriben en papel o en un archivo. En esta tesis un Objeto Persistente es un ente capaz de almacenar cualquier tipo de información, ya sea una

---

A lo largo de toda la tesis se usan las palabras Objeto y Objeto Persistente con el mismo significado. De la misma manera, las palabras Sistema y Sistema Operativo significan lo mismo.

estructura de datos o lo que resulte de la ejecución de un programa. Se le llama Persistente porque puede existir más allá del lapso de duración que tiene el proceso que lo creó, o que lo usó más recientemente.

Las operaciones que se pueden ejecutar sobre los Objetos, a diferencia de los archivos, son: acceso concurrente al Objeto controlado por candados; el Objeto se maneja con respaldo; el Objeto siempre tiene una copia consistente de su contenido; se pueden revocar (roll back) todas las modificaciones que se hayan hecho a un Objeto para dejarlo en su último estado consistente; el Objeto tiene apuntadores a otros Objetos.

Necesidades que satisfacen los Objetos Persistentes.

En la actualidad la computadora debe brindar apoyo para manejar aplicaciones como las que se listan abajo, pero como el "hardware" no proporciona las condiciones necesarias para hacerlo, se pueden implantar a través de Objetos en el "software" de la programación de sistemas.

Las aplicaciones que debe apoyar la computadora en la actualidad son:

(i) Compartir y comunicar información.

(ii) Protección y seguridad.

Como se implantan los Objetos Persistentes.

Las formas de implantar el manejo de los Objetos Persistentes en una computadora son diversas, pueden ser a través de un compilador, de un interprete, como parte del núcleo del Sistema Operativo o a través del "hardware". En el caso de esta tesis, el manejo de los Objetos Persistentes se construyó como una extensión al Sistema Operativo. Todo el "software" (programación en lenguaje "C") necesario para apoyar el manejo de los Objetos se agregó al Sistema Operativo "UNIX", en particular a la versio'n V que opera en las máquinas "TOWER" de la compañía "NCR".

Para que el Objeto sea Persistente se debe guardar en almacenamiento perdurable (disco), sin embargo, cuando un proceso esta usando un Objeto es mejor tenerlo en la memoria principal. En esta tesis el manejo de los Objetos se hará tanto en la memoria principal de la computadora, como en la secundaria. Esta manipulación es transparente para el usuario.

Objetivos de la tesis.

Los objetivos de la tesis se pueden resumir de la siguiente manera:

Que los Objetos sean persistentes y consistentes, es decir, que el usuario siempre pueda contar con una copia válida del contenido de los Objetos.

Que los Objetos Persistentes sean un método sencillo y uniforme para guardar información.

Que el usuario pueda compartir información con otros usuarios o programas a través del Objeto Persistente.

Que el Objeto Persistente cuente con un mecanismo que le permita proteger su contenido del acceso de usuarios no autorizados.

Las operaciones básicas que se van a implantar para cumplir con los objetivos indicados son:

i) Creación de un Objeto Persistente.

ii) Ligar un Objeto Persistente a la "raíz".

iii) Desligar un Objeto Persistente de la "raíz".

iv) Solicitar acceso a un Objeto Persistente.

v) Leer la información de un Objeto Persistente.

vi) Escribir información sobre un Objeto Persistente.

vii) Lectura y escritura de los apuntadores a otros Objetos.

viii) Compromiso (commit) de un Objeto Persistente.

ix) Revocar (Roll Back) a un Objeto Persistente.

x) Modificación a los permisos de un Objeto.

El significado y el funcionamiento de cada una de las operaciones listadas se explican a lo largo del trabajo.

Contenido de la tesis.

En el capítulo I se dan las definiciones de los conceptos básicos que se van a usar en el trabajo y también se definen las funciones que componen la extensión que se hace al Sistema Operativo. Estas funciones son las que aparecen en la lista anterior.

El capítulo II contiene la especificación de los requerimien-

tos que debe cumplir la extensiòn para implantar el manejo de los Objetos. Dicha especificaciòn se hace desde dos puntos de vista: el del usuario y el del Sistema Operativo. Se define lo que es el Objeto Persistente en la implantaciòn, la manera como se va a almacenar, el tipo de respaldo que se le va a proporcionar, las estructuras de datos que se van a usar para manejarlo y la organizaciòn y mantenimiento del dispositivo de almacenamiento perdurable.

En el capitulo III se describe el desarrollo del trabajo. En èl se se incluye la documentaciòn del manual de usuario, la razòn por la que se hace èsto, es que asi el lector se puede dar una idea màs clara de como quedaron implantadas las funciones de la extensiòn, es decir, toda la programaciòn realizada para esta tesis.

El capitulo IV contiene la descripciòn de las pruebas realizadas a la extensiòn y los resultados que las mismas arrojaron.

Finalmente se dan las conclusiones que se pueden derivar de este trabajo.

Capitulo 1.

Definiciones Bàsicas.

1.1 Necesidades que deben satisfacer los Objetos Persistentes.

Aunque los sistemas actuales difieren significativamente de sus predecesores, en tèrminos de costo, velocidad, confiabilidad, organizaciòn interna y tecnologìa de circuitos, la arquitectura de las computadoras de muchos sistemas actuales, no ha avanzado màs alla de los conceptos de 1950 [MYERS 78]. Esta estructura fue creada màs por su facilidad de construcciòn en "hardware" que por sus propiedades computacionales superiores [BISHOP 77]. No obstante que la arquitectura de von Neumann es un diseño brillante para la primera computadora de programa almacenado, es ajena a la ejecuciòn de programas escritos en lenguajes de alto nivel [MYERS 78]. En la actualidad la computadora debe brindar apoyo para manejar aplicaciones como las que se listan abajo, pero como el "hardware" no proporciona las facilidades necesarias para este manejo, se implantan en el "software" de la programaciòn de sistemas.

Aplicaciones que en la actualidad debe apoyar la computadora:

i) Compartir y comunicar informaciòn. Compartir e intercambiar informaciòn en forma dinàmica, tanto en un sistema de tiempo compartido como en una red. Para compartir la informaciòn es fundamental el direccionamiento de objetos. Serìa màs simple esta tarea si las direcciones se pudieran transmitir entre los procesos y luego se pudieran usar para accesar los datos compartidos [LEVY 84].

ii) Protecciòn y seguridad. Conforme se vaya haciendo màs fàcil compartir informaciòn, los usuarios necesitarèn màs control sobre el acceso a sus datos privados. Debe ser posible compartir informaciòn o correr programas escritos por otros usuarios sin comprometer datos confidenciales. La protecciòn se enriquecerìa si el usuario pudiera restringir el acceso a solamente los objetos que un programa requiere para su ejecuciòn [LEVY 84].

1.2 Que es un Objeto Persistente para el usuario.

En este trabajo un objeto persistente es una herramienta de

programación que cualquier usuario puede utilizar. El objeto sirve para guardar de manera permanente los datos que al usuario le interesen. Es decir, cuando el programa que el usuario este diseñando, tenga necesidad de pedir al sistema que le conserve algún dato en particular, puede solicitar este servicio a través de un objeto persistente. Este objeto le garantiza al usuario lo siguiente:

- Que la información almacenada en él va a existir en el sistema mientras el creador lo desee.

- Que lo van a poder leer, modificar y/o ejecutar (siempre que sea posible) todos los usuarios a los cuales el creador les haya dado permiso.

- Que siempre se contará con una copia válida (consistente) del objeto, independientemente de las fallas del sistema o de las actualizaciones fallidas de los usuarios.

En lo que respecta al ambiente en el que se van a dar los objetos, el usuario no se debe preocupar en absoluto del espacio que estos ocupan, ni del mantenimiento requerido por las estructuras de datos que se usan para controlarlos, ni de los posibles conflictos de concurrencia que puedan ocurrir al usarlos, ya que el sistema se va a encargar de todo esto. El sistema va a manejar los objetos tanto en la memoria principal de la máquina como en el almacenamiento secundario (disco). Estarán de manera permanente en el almacenamiento secundario y en la memoria sólo cuando estén siendo usados por algún(os) proceso(s).

Las operaciones que se pueden ejecutar sobre los objetos, a diferencia de los archivos, son: se puede hacer acceso concurrente, controlado por candados, al objeto persistente; el manejo del objeto se hace con respaldo; siempre se tiene una copia consistente del contenido del objeto; se pueden revocar (roll back) todas las modificaciones que se hayan hecho a un objeto para dejarlo en su último estado consistente; el objeto puede guardar apuntadores a otros objetos.

1.3 Como se implantan los Objetos Persistentes en este trabajo.

Hay varias formas de implantar objetos persistentes en una computadora, como son: a través de un compilador o un interprete, como parte del núcleo del sistema operativo o mediante el "hardware". En este trabajo el apoyo para manejar objetos persistentes se construyó como una extensión al sistema operativo. Todas las funciones que son necesarias para apoyar el manejo de los objetos se agregaron al sistema operativo "UNIX", en particular, a la versión "V" del "UNIX" que maneja la computadora "TOWER" de la compañía "NCR".

Antes de implantar el manejo de los objetos hay que tener en cuenta que cada uno de ellos tiene, al menos, dos niveles de

abstracción: por un lado esta el modelo del objeto, el cual es responsabilidad absoluta del programador; por otro lado se tiene la representación del objeto persistente dentro de la computadora [BISHOP 77]. En este segundo nivel de abstracción es donde se puede ubicar el trabajo que se esta describiendo, es decir, en el apoyo que da la programación de sistemas (en este caso el sistema operativo) para el manejo interno de los objetos persistentes.

1.4 Funciones básicas para el manejo de los Objetos Persistentes.

En esta sección se describen las funciones básicas que se desea que el sistema tenga para el manejo de los objetos persistentes.

1.4.1 Creación de un Objeto Persistente.

Cualquier proceso puede solicitar la creación de un objeto. El sistema debe llevar a cabo las siguientes acciones para asegurar la creación del objeto:

- Reservar área de memoria, en el espacio asignado al usuario, para que éste pueda crear ahí su objeto.
- Dar al usuario la dirección de esta área, para que la pueda accesar sin problemas. A través de dicha dirección se puede leer y escribir el objeto tantas veces como se desee.
- El sistema debe tener en cuenta que se esta creando un nuevo objeto y que debe estar listo para conservarlo, en caso de que el usuario lo solicite (a través de la operación "Compromiso" -commit-).
- Cuando el usuario crea un objeto define los permisos que se darán al resto de los usuarios para poder accesarlo.

El método que se usa para guardar los objetos en el almacenamiento perdurable hace necesario que exista un objeto "raíz", éste no necesita ser creado por ningún usuario ya que se genera desde el momento de la inicialización. El objeto raíz apunta a todos los objetos que están dados de alta en el sistema y que por lo tanto se deben conservar. Más adelante se trata el método que se usa para guardar los objetos en el almacenamiento perdurable y se describen un par de funciones que se usan para dar de alta y para remover del objeto raíz a otro objeto cualquiera.

1.4.2 Ligar un Objeto Persistente al Objeto "raíz".

El dispositivo de almacenamiento perdurable en el que se van a guardar los objetos esta dividido de dos maneras. La primera es

una división de todo el dispositivo en fragmentos del mismo tamaño llamados "bloques". La segunda división se construye sobre la primera, esto es, se agrupan varios bloques del dispositivo y a cada grupo se le conoce como "área". Cuando se crea cada área se crea también dentro de ella un objeto conocido como "raíz". Mediante la operación de ligar un objeto persistente a la raíz, se escribe el identificador de dicho objeto persistente en el objeto raíz. El sistema preserva todos los objetos que estén inscritos en el objeto raíz. En el momento que un objeto ya no es usado por su creador, éste lo borra del objeto raíz. Cada cierto lapso de tiempo se ejecuta un proceso "recolector de basura", que revisa cada una de las áreas y elimina los objetos que no estén dados de alta en el objeto raíz de su área respectiva.

1.4.3 Desligar un Objeto Persistente del Objeto "raíz".

En este caso se trata de eliminar un objeto que ya no se usa. Para ello se solicita al sistema, que remueva de la lista que tiene el objeto "raíz", el identificador del objeto a borrar. Esta operación sólo la puede ejecutar el creador del objeto que se quiere eliminar. El sistema lo marca (al objeto) como borrado, sin embargo, antes de eliminarlo del almacenamiento perdurable, se debe asegurar de que entre todos los objetos activos del sistema, no haya alguno que tenga un apuntador al objeto que se pide eliminar. Si esto se cumple, el objeto será borrado por un proceso de "Recolección de Basura". Mismo que recupera el espacio que ocupaba antes el objeto y lo pone como disponible. De lo contrario el objeto se conserva hasta que no haya nadie apuntándolo.

1.4.4 Acceso a un Objeto Persistente.

Cuando algún proceso de usuario desea tener acceso a un objeto que ya existe en el sistema pueden ocurrir dos situaciones: (i) que el objeto no este siendo usado por nadie, entonces se encuentra en el almacenamiento seguro (disco). En este caso el sistema sólo verifica que los permisos de acceso del usuario y del objeto sean compatibles. Si lo son, se pone accesible al usuario una copia del objeto, de lo contrario esta función devuelve un mensaje de error. (ii) La segunda situación que puede ocurrir, es que el objeto este siendo usado en este momento por otro usuario, entonces se deben cumplir dos condiciones para poder poner accesible el objeto al usuario. La primera condición es que los permisos de acceso del objeto y los del usuario sean compatibles; la segunda condición consiste en garantizar que no haya conflicto de acceso entre el usuario actual y el que acaba de solicitar el objeto. Si alguna condición, o ambas, no se cumplen, se niega el acceso al objeto para el nuevo usuario.

1.4.5 Leer información de un Objeto Persistente.

Antes de poder leer de un objeto persistente, es necesario tener acceso a él. La operación descrita en el inciso anterior es la que permite o no el acceso a un objeto cualquiera. Una vez que el proceso tiene garantizado el acceso al objeto se pone a su disposición una copia del mismo. Sobre esta copia puede trabajar el proceso, esto es, leer toda la información que quiera.

1.4.6 Escribir información sobre un Objeto Persistente.

Al igual que en el caso de lectura, primero se debe poner accesible el objeto persistente. Una vez que el acceso seguro al objeto esta garantizado para el proceso, a éste se le permite escribir todas la modificaciones que necesite. En este caso, el sistema toma nota de cuales fragmentos del objeto han sido modificados, para poder actualizar su estado si el usuario así lo solicita. La actualización se lleva a cabo sólo si se ejecuta la operación de compromiso (commit), de otra manera el objeto queda como estaba antes de que este proceso lo usara.

1.4.7 Lectura y escritura de los apuntadores a otros Objetos.

A partir de aquí se establece que la zona de almacenamiento de apuntadores y la de información, en un objeto, son diferentes y por lo tanto, las funciones que operan sobre dichas zonas no son las mismas. Sin embargo, las operaciones de lectura y escritura de apuntadores son análogas a las de lectura y escritura del contenido del objeto. En ambas se debe garantizar primero un acceso seguro al objeto y una vez hecho esto, se puede llevar a cabo la operación requerida. Como se trata de funciones diferentes no se puede usar la función de leer información para leer apuntadores, ni viceversa. Lo mismo sucede con las funciones de escritura, es decir, éstas no se pueden intercambiar.

1.4.8 Compromiso (Commit) de un Objeto Persistente.

Cuando un objeto ha sido creado, la única forma de hacerlo persistente es a través de la operación de compromiso (commit). Esta operación toma el objeto en el estado que se encuentra actualmente y lo guarda en almacenamiento seguro. Además, genera un identificador único con el cual se va a conocer el objeto dentro del sistema. Este identificador se le entrega al usuario para que, a través de él, pueda referirse a su objeto siempre que lo requiera. El identificador puede ser transferido libremente entre los procesos, ya que hay mucha diferencia entre conocer el identificador de un objeto y tener permiso de accesarlo.

La operación de compromiso (commit) también se usa en el caso

de que se haya actualizado un objeto viejo y se desee conservar su nuevo estado. En esta situación, el usuario solicita el objeto y si tiene derecho a usarlo el sistema se lo proporciona; se ejecutan todas las operaciones de modificación que sean necesarias y finalmente se invoca la operación de compromiso (commit), que hace permanente la actualización.

La operación de compromiso (commit) sobre un objeto, sólo garantiza que el contenido de éste se ha depositado en el dispositivo de almacenamiento perdurable. Si el propietario del objeto quiere que éste se quede permanentemente en el sistema, lo debe dar de alta en el objeto raíz que le corresponde en el dispositivo donde quedó almacenado. De lo contrario, es decir, si no lo da de alta, en el próximo proceso de "recolección de basura" que se aplique al dispositivo de almacenamiento se eliminará dicho objeto. Más adelante en la tesis se trata el tema de como esta organizado el dispositivo de almacenamiento y el proceso de "recolección de basura" que se le aplica.

1.4.9 Revocar (roll back) a un Objeto Persistente.

Puede ocurrir el caso de que un proceso solicite un objeto, éste se le ponga accesible y entonces el proceso realize varias modificaciones sobre dicho objeto; si después de haber hecho esto, el propietario del proceso se arrepiente y quiere dejar el objeto como estaba al principio, debe ejecutar la operación de revocación (roll back). Esta devuelve el objeto a su estado previo de consistencia, es decir, lo deja como estaba antes de que el proceso hiciera su primera modificación. Hay otras razones por las que puede surgir la necesidad de ejecutar una operación de revocación (roll back), por ejemplo, que el sistema elimine (aborte) un proceso por tratar de ejecutar acciones incorrectas, o al quitar un proceso para resolver un problema de "deadlock", etcétera. En estos casos, todos los objetos que venía usando el proceso vuelven a quedar como estaban antes de que se pusieran accesibles.

1.4.10 Modificación de los permisos de operación de un Objeto.

Cuando un objeto es creado inicialmente, su creador le asigna permisos de lectura, escritura y ejecución a cada uno de los diferentes conjuntos de usuarios que existen en el sistema. Estos conjuntos son en nuestro caso, el creador, el grupo de usuarios al que pertenece el creador y todos los otros usuarios que no se incluyen en los dos primeros conjuntos. Si después de cierto tiempo, el creador desea restringir más los permisos iniciales o relajarlos un poco, puede ejecutar la operación de modificar los permisos de operación sobre el objeto. Esta operación sólo se permite ejecutarla al creador del objeto o al sistema y para hacerla, primero es necesario que se tenga acceso al objeto.

Capitulo 2.

Especificaciòn de requerimientos para la implantaciòn de los Objetos Persistentes.

En este capitulo se define lo que va a ser un objeto persistente a lo largo de todo el trabajo, asi como el ambiente en el que va a existir. Las definiciones se dan desde dos puntos de vista, el del usuario y el del sistema. Ademàs, ellas se van a tomar como la especificaciòn de los requerimientos que se deben cumplir para manejar los objetos persistentes. Las definiciones abarcan los siguientes puntos (tanto para el usuario como para el sistema):

i) Definiciòn del Objeto Persistente.

ii) Almacenamiento del Objeto Persistente.

iii) Respaldo para el Objeto Persistente.

iv) Estructuras de Datos que ayudan a mantener el Objeto Persistente dentro del sistema.

v) Mantenimiento de la zona de almacenamiento de Objetos Persistentes.

2.1 Definiciòn del Objeto Persistente.

2.1.1 El Objeto Persistente visto por el usuario. (Representaciòn lògica de un Objeto Persistente.)

El usuario ve el objeto persistente como una zona de memoria en donde va a poder almacenar de manera permanente los datos y los apuntadores a los objetos que le interesen. (Vèase figura 1.) Cuando el usuario crea el objeto, determina la longitud de contenido que va a tener (medida en "bytes"), y almacena sus datos con la organizaciòn que a èl le convenga. Cuando lo crea tambièn especifica el número de apuntadores que su objeto va a contener. Una vez creado el objeto, el usuario va a poder ejecutar diversas operaciones sobre èl, como se explica en el capitulo siguiente.

2.1.2 El Objeto Persistente visto por el sistema.

El espacio de memoria dedicado a almacenar los objetos esta dividido en fragmentos del mismo tamaño llamados "páginas". Cuando un objeto se crea se le asigna como lugar de residencia una página lógica. A partir de ese momento, el objeto y su página lógica son inseparables. Sin embargo, la página física en la que reside la página lógica es variable. Para seguir la pista de la página física en la que esta cada página lógica se usa una estructura de datos que ayuda a realizar el mapèo entre ambas. En este trabajo una "página" es un bloque de 512 bytes. Como el tamaño de un objeto es variable, puede suceder que quepa muy bien en una página, o puede suceder lo contrario, es decir, que no quepa en una sola y que tenga que ser almacenado ocupando varias páginas. (Vèase figura 2.) En este último caso, es necesario saber cual es el fragmento inicial del objeto y cuales son sus continuaciones, además de saber en que orden van estas, ya que aunque desde el punto de vista lògico el objeto se almacena consecutivamente, puede suceder que las páginas físicas no sean adyacentes.

La información de la longitud del objeto, asi como la que indica si es el fragmento inicial o una continuación además de otros datos, la debe conservar el sistema para poder manipular correctamente los objetos. Es por esto que toda la información que ayuda a identificar un objeto se almacena en una estructura llamada "encabezado". Entonces, un objeto persistente, desde el punto de vista del sistema, va a estar formado por un "encabezado" y un "contenido". Donde el "contenido" son los datos que el usuario ha almacenado. (Vèase figura 3.) En este trabajo no es necesario que la información del encabezado este físicamente adyacente al contenido del objeto. Lo que si es requisito es que estèn en la misma página. Para facilitar la busqueda de los objetos que estàn en una página, los encabezados y los contenidos se guardan de manera opuesta, es decir, los encabezados de arriba hacia abajo y los contenidos de abajo hacia arriba. Como se puede apreciar en la figura 3.

2.1.3 Consideraciones de diseño.

Las consideraciones de diseèo que se tomaron antes de pasar al desarrollo de la extensión fueron las siguientes:

i) El manejo de la concurrencia se harà a nivel de objeto persistente.

ii) El control de la concurrencia sobre los objetos se harà a través de candados. Las acciones de serializar las solicitudes de acceso concurrente, de detectar el caso de un "deadlock", etcètera, se dejaràn abiertas para que se construyan de acuerdo al manejador que se desarrolle sobre la extensión.

iii) La extensión aquí descrita contará con un mecanismo multiple de recolección de basura. Es decir, el espacio de almacenamiento secundario se dividirá en áreas y el proceso de recolección se podrá aplicar de manera independiente en cada una de ellas. El recolector de basura deja en las áreas solo los objetos que son accesibles. Un objeto es accesible si está apuntado por el objeto raíz de su área o por un objeto accesible, ya sea de esta área, o de cualquier otra.

iv) El respaldo de los objetos se hará a través del procedimiento del archivo sombra, por considerarse a éste más sencillo que algunos otros, por ejemplo, el de conservar el estado anterior y guardar al final de la transacción el nuevo estado.

v) El manejo de las páginas en este sistema será parecido al que se usa en el sistema "R" de IBM o al de "DMS-1100" de UNIVAC.

vi) La extensión debe ser capaz de sustituir en cualquier actividad al sistema común de archivos de UNIX.

vii) Se proporcionará un conjunto mínimo de llamadas al sistema, necesarias para implantar el manejo de objetos, que se puedan integrar a las que ya contiene el sistema "UNIX".

viii) El procedimiento de asignación y manejo de permisos para cada uno de los objetos, y el estilo general de todas las funciones de la extensión, se apegará en lo posible a la filosofía del sistema "UNIX".

2.2 Información básica sobre el Objeto Persistente.

En resumen, para el sistema un objeto va a estar constituido, básicamente, por: (a) la información que el usuario ha almacenado en él y (b) por un conjunto de datos que se necesitan conocer para manipularlo; denominados a partir de ahora como (a): contenido y (b): encabezado del objeto. (Véase figura 3.) La información que el sistema va a almacenar en el encabezado de un objeto es la siguiente:

Datos que contiene el encabezado del fragmento inicial de un objeto:

i) Identificador único del Objeto Persistente.

ii) Longitud total del Objeto Persistente, en "bytes".

iii) Permisos de acceso al Objeto Persistente.

iv) Creador del Objeto Persistente.

v) Dirección inicial del contenido, dentro de la página.

vi) Identificador de la continuación.

vii) Número del último "byte" en este fragmento.

viii) Número de apuntadores a otros Objetos Persistentes.

En seguida se explica que es cada uno de estos datos.

2.2.1 Identificador único del Objeto Persistente.

Con este identificador se conoce al objeto dentro del sistema. La única forma de tener acceso a un objeto es a través de su identificador. El identificador esta compuesto de dos elementos concatenados, el primero es el número de la página lógica donde se encuentra dicho objeto, y el segundo es un número interno que el sistema le asigna. (Véase figura 4.) Para poder saber si se trata del inicio de un objeto o de una continuación se debe revisar su identificador único. Si el identificador es par se trata del inicio, si es non se trata de una continuación. Además, la parte menos significativa del identificador indica de que número de continuación se trata.

El identificador único de un objeto puede ser solicitado, copiado, transferido o borrado (borrar equivale a dejar de hacerle referencia) libremente por cualquier proceso, ya que se consideran como operaciones diferentes la de tener el permiso de acceso a un objeto y la de tener el identificador del mismo.

El identificador de un objeto nunca cambia, es decir, mientras exista el objeto va a tener siempre el mismo identificador que le fue asignado cuando se creó.

2.2.2 Longitud total del Objeto Persistente, en "bytes".

Esto es, la longitud, medida en "bytes", de la información que hay almacenada en el objeto al momento de su creación. La longitud de un objeto es invariable y se determina al momento de su creación.

2.2.3 Permisos de acceso al Objeto Persistente.

Son los permisos que el creador de un objeto da al resto de los usuarios del sistema para que puedan tener acceso al mencionado objeto. La forma de estos permisos es la que se usa en el sistema "UNIX". Los usuarios del sistema se dividen en tres

conjuntos: O (Owner) el propietario; G (Group) el grupo al que pertenece el propietario; y W (World) el mundo, o sea, todos los otros usuarios que no están en los dos primeros conjuntos. Los permisos abarcan las siguientes actividades: R (Read) lectura, W (Write) escritura y X (eXecute) ejecución. El propietario del objeto es el que da o quita derechos de acceso a cada conjunto de usuarios. Los permisos se asignan de manera independiente, esto es, tanto al propietario (O:REX), como al grupo (G:REX) y al mundo (W:REX) se les pueden asignar los permisos de lectura, escritura y ejecución de manera independiente.

Los permisos con que va a operar un objeto se determinan al momento de su creación, pero el usuario creador los puede cambiar cuantas veces quiera durante la existencia de su objeto.

2.2.4 Creador del Objeto Persistente.

En este campo se guarda la identificación del usuario que creó el objeto. Esto se usa principalmente para cuestiones de control. Por ejemplo, el único que puede modificar los permisos de operación con que fue creado un objeto es su creador.

El identificador del creador nunca cambia, es decir, desde que el objeto se crea hasta que se borra va a conservar siempre el mismo identificador del usuario que lo creó.

2.2.5 Dirección inicial del contenido del Objeto, dentro de la página.

Un objeto, o un fragmento de él, pueden ocupar una página lógica completa. Sin embargo, cuando el objeto, o su fragmento, ocupan sólo una parte de la página, es necesario saber a partir de que "byte" de la misma esta almacenado el contenido. (Véase figura 3). En este campo se guarda la dirección inicial, dentro de la página, donde inicia el contenido del objeto.

Como el contenido de un objeto no varía, esta dirección no va a cambiar. Sin embargo, cuando se elimina algún objeto de la misma página, puede suceder que deje un hueco entre los objetos que aun residen ahí. En este caso se lleva a cabo una compactación sobre la página, lo que puede ocasionar que esta dirección cambie.

2.2.6 Identificador de la continuación de un Objeto Persistente.

Cuando un objeto no cabe en una sola página, es necesario saber en donde esta cada una de sus continuaciones. En el encabezado del objeto se guarda el identificador de su continuación más

inmediata. De la misma manera, cada uno de los fragmentos del objeto tiene el identificador de su continuación más próxima. (Véase figura 2.)

Como las continuaciones de un objeto no cambian, es decir, el objeto se conserva siempre del mismo tamaño que fue creado, estos identificadores no van a cambiar nunca mientras el objeto exista.

2.2.7 Número del último "byte" en este fragmento del Objeto.

Si un objeto no cabe en una sola página es necesario saber hasta que "byte", de su longitud total, esta contenido en cada continuación. Esto es necesario, sobre todo cuando se hace la busqueda de un "byte" especifico dentro de todo el objeto (fragmento inicial y continuaciones).

Como un objeto no modifica su longitud, los tamaños de sus continuaciones tampoco cambian. Por lo tanto, el último "byte" contenido en cada fragmento siempre es el mismo.

2.2.8 Número de apuntadores a otros Objetos Persistentes.

Si cuando se crea un objeto se sabe que va a apuntar a otros objetos, se debe solicitar espacio para guardar esos apuntadores. (Véase figura 1.) En el campo de apuntadores del encabezado se guarda únicamente el número de apuntadores que se van a tener, los apuntadores en si se conservan junto al inicio del contenido. (Véase figura 5.) Aunque los apuntadores y el contenido esten juntos, el sistema sólo permite el acceso a cada uno ellos a través de operaciones bien definidas, como son: leer/escribir apuntador y leer/escribir contenido.

Cuando se dice que un objeto tiene apuntadores hacia otros objetos, esos apuntadores son de hecho los identificadores de los objetos apuntados. Por lo tanto, a lo largo de la tesis siempre que se trate de un apuntador se puede pensar en el identificador de dicho objeto.

El número de apuntadores que tiene un objeto se determina en el momento de su creación y nunca cambia.

A través de los apuntadores los objetos pueden formar estructuras como la que muestra la figura 6.

2.2.9 Encabezado de una continuación del Objeto.

Como se sabe que puede haber objetos que estén distribuidos en varias páginas, a cada continuación se le asigna un encabezado en

el que se encuentran sus datos particulares. Dado que no es necesario repetir toda la información, como si se tratara del inicio del objeto, cuando se trate de una continuación se almacenarán menos datos. Estos son los que se listan en seguida, y su significado es el mismo que se dio cuando se describió el encabezado del inicio del objeto. En la figura 7 se puede ver el fragmento inicial del objeto con su encabezado y en la figura 8 una continuación con su encabezado respectivo.

Datos que contiene el encabezado de una continuación de un objeto:

i) Identificador único del objeto.

ii) Dirección inicial del contenido del objeto dentro de la página.

iii) Identificador de la continuación del objeto.

iv) Número del último "byte" en este fragmento del objeto.

v) Número de apuntadores a otros objetos.

2.3 Almacenamiento del Objeto Persistente.

2.3.1 Almacenamiento desde el punto de vista del usuario.

El usuario no necesita saber de que manera esta almacenado su objeto, a él lo único que le interesa es que cuando pida que éste se guarde, el objeto se conserve en el sistema. Y cuando lo solicite para accesarlo, el sistema le proporcione una copia para llevar a cabo operaciones sobre él.

2.3.2 Almacenamiento desde el punto de vista del sistema.

El almacenamiento desde el punto de vista del sistema abarca dos niveles. El primero se refiere a las páginas y el segundo al arreglo que forman estas.

2.3.2.1 Páginas en el sistema.

El elemento básico de almacenamiento para los objetos es la "página". Esta es un arreglo consecutivo de 512 "bytes". Cuando se deposita un objeto en memoria, siempre se pone sobre una o varias páginas, dependiendo de su longitud. Cada página tiene reservados los cuatro primeros "bytes" para funciones de control. En estos "bytes" se guarda información sobre el contenido de la

pàgina. Los datos considerados son: en el primer "byte" se guarda el nûmero de fragmentos iniciales de objetos que hay en la pàgina. El segundo "byte" tiene el nûmero de continuaciones. Y el tercero y cuarto son la direcciòn del primer "byte" desocupado dentro de la pàgina. (Vêase figura 9.)

2.3.2.2 Areas de almacenamiento.

El sistema maneja los objetos tanto en la memoria principal de la màquina como en el almacenamiento permanente (disco). En la memoria principal forman una àrea comûn en la que varios procesos pueden estar consultando al "mismo tiempo" sus objetos. En el disco las pàginas lògicas van a formar un dispositivo aparte, reservado ûnicamente para conservar los objetos que ahí se depositen. En este dispositivo siempre habrà una copia segura y vàlida del objeto, debido a que se maneja como "archivo sombra", esto se explica ñas a detalle en el siguiente inciso.

Los objetos estàn en el almacenamiento perdurable de manera permanente y pasan a memoria principal sôlo cuando algûn proceso los solicita para ejecutar operaciones sobre ellos. Si se desêa conservar un objeto modificado, se pasa de memoria a disco siguiendo el procedimiento del "archivo sombra", como se explica enseguida.

2.4 Respaldo para el Objeto Persistente.

2.4.1 Respaldo desde el punto de vista del usuario.

Para el usuario una de las caracteristicas importantes con que estàn dotados los objetos, ademàs de la persistencia, es su consistencia. Esto quiere decir, que siempre se conservarà una copia de la ûltima versiòn de un objeto cuando un proceso le este haciendo modificaciones, es decir, del contenido que tenia antes de que se empezara a trabajar sobre êl. De esta manera se asegura que, si por cualquier razòn el objeto modificado se pierde, siempre se cuente con la versiòn anterior.

2.4.2 Respaldo desde el punto de vista del sistema.

Para asegurar el estado consistente del objeto el sistema utiliza el esquema del "archivo sombra" (shadow file) de R. Lorie [Lorie 77]. En la figura 10 se pueden ver las estructuras de datos que se van a usar para implantar el manejo del archivo sombra, tanto en la memoria principal como en el almacenamiento secundario (para abreviar denominado "disco" a partir de ahora). En seguida se explica brevemente que significan cada uno de los elementos ahí representados.

El disco esta dividido fisicamente en bloques del mismo tamaño. (Un bloque es totalmente equivalente a una página. Se les denomina bloques cuando pertenecen al disco y páginas cuando se trata de la memoria principal.) Cada bloque tiene un número único y el conjunto de todos ellos representa la capacidad total de almacenamiento en el disco. Al almacenar un objeto en disco se guarda dividido en bloques. Para saber que bloques del disco estan libres y cuales estan ocupados se usa un arreglo de "bits" (bit map) denominado "DMap" (vèase figura 10), donde cada "bit" representa un bloque del disco. Si el "bit" es "0" el bloque esta libre; si el "bit" es "1" el bloque esta ocupado. Se tienen dos mapas ("DMap0" y "DMap1") por cuestiones de seguridad, esto es, mientras se está actualizando un mapa, por ejemplo "DMap0", se tiene una copia de su contenido en "DMap1", si llegara a ocurrir una falla de cualquier tipo, el contenido de "DMap0" se volvería inconsistente, lo que no sucede con el contenido de "DMap1", del que se puede echar mano para recuperar el sistema. Para saber cual de los dos mapas es el que refleja el estado actual de ocupaciòn en el disco, se usa una bandera denominada "DMapswitch" (vèase figura 10). El valor de la bandera ("0" o "1") indica cual es el mapa actual (DMap0 o DMap1). La estructura llamada "DVO" es un arreglo que contiene un elemento por cada bloque que existe en el disco. Cada índice del arreglo equivale a una página lògica y el número guardado en el elemento del arreglo equivale a la página física (bloque). Por razones de consistencia, cuando se actualiza una página lògica, no se reescribe sobre la misma página física en la que estaba originalmente, sino que se busca una página física vacia y ahí se deposita su nueva versiòn. Posteriormente se actualiza el arreglo "DVO", para que la página lògica apunte a la nueva página física y también se actualiza el arreglo "DMap", para que muestre la página anterior como libre y la página actual como ocupada. Por las mismas razones de seguridad que se mencionaron antes, se tienen dos arreglos de este tipo: "DVO" y "DV1".

Continuando con la figura 10, en la memoria de la computadora se tiene una copia de la estructura "DVO" y una replica del "bit map" de disco, es decir, el "DMap" que indique "DMapswitch". La copia de "DVO" es en particular sobre el segmento que se esta usando y al igual que en el disco, se tiene repetida, esto es, "MVO[k]" y "MV1[k]". El "bit map" de memoria se llama "MMap" y ejecuta exactamente las mismas funciones que el "DMap" del párrafo anterior. Estas estructuras sirven para manejar los objetos que pasan a memoria para ser usados por los procesos.

2.4.2.1 Archivo Sombra.

La explicaciòn del funcionamiento de las estructuras de datos para manejar los objetos se hará tomando como ejemplo lo que muestra la figura 10. Se desèa utilizar un objeto, el que en este caso ocupa las páginas lògicas: "c", "n" y "r" y las páginas físicas: "c", "1" y "p". Inicialmente el objeto sòlo se encuentra en disco. La primera vez que se vaya a pasar cualquier

página lógica a memoria, se debe tomar su dirección física del arreglo "MV[0]". Con esta dirección se accesa el disco, se saca un bloque de ahí y se pasa a memoria. En la memoria hay una área reservada para poner únicamente los bloques que vienen del disco. El bloque se translada una sola vez, cualquier otra referencia a él se dirige a su página en memoria. Si el bloque que esta en memoria se modifica (algún proceso escribe sobre él), se enciende una bandera para indicar ese hecho (en la figura 10 se usa el símbolo "~" para representar la bandera). Cuando se guarda el nuevo contenido del objeto (operación de compromiso -commit-), sólo los fragmentos que han sido modificados se reescriben a disco, pero no a sus bloques de origen, sino a un bloque libre (archivo sombra) que se localiza con la ayuda de "MMap". La nueva dirección que ocupa cada fragmento en el disco se anota en los arreglos "MV" y "DV".

2.4.2.2 Procedimiento de respaldo con el archivo sombra y un Objeto como ejemplo.

Supongamos que el objeto del ejemplo tiene su fragmento inicial en la página lógica "c" y que ocupa completamente esta página y la "n", donde tiene su primera continuación; y ocupa parcialmente la página "r" donde esta su último fragmento. Para llegar al estado que muestran las estructuras de datos de la figura 10, el proceso que esta usando el objeto del ejemplo pudo proceder de la siguiente manera: primero solicitó al sistema que se le pusiera accesible el objeto, al ponerlo a su disposición el sistema paso una copia del contenido del bloque "c" de disco, a la página "c" de la zona de memoria común donde residen los objetos. En el momento que el proceso escribió sobre la pagina "c", ésta se marcó como modificada en "MV[0]". Después, el proceso pidió datos que están en la primera continuación del objeto, por lo que el sistema trajo el bloque "l" de disco a la página "l" de la memoria común. Finalmente el proceso solicitó datos que estan en el último fragmento del objeto. Entonces el sistema le trajo el bloque "p" de disco a la página "p" de la memoria común. En el momento que el proceso escribió sobre esta página, ella se marcó como modificada en "MV[0]". Supongamos ahora que el proceso que esta trabajando sobre el objeto de la figura 10 desèa hacerle el compromiso (commit), es decir, almacenarlo tal como se encuentra en ese momento. Las estructuras de datos se modifican como lo indica la figura 11. Para ejecutar ese cambio se llevan a cabo los siguientes pasos:

2.4.2.2.1 El mapa de ocupación de páginas se pasa de disco a memoria.

En este paso se copia el mapa "DMap" que indique la bandera "DMapswitch", a un arreglo temporal de memoria llamado "TMap". Véanse en la figura 10 la bandera "DMapswitch" y el arreglo "DMap[0]"; y en la figura 11 el arreglo "TMap". La razón por la que se hace esta copia, es que el mapa de ocupación de disco no

es igual al que esta en memoria. Esto se debe a que el mapa de disco refleja el estado de ocupaciòn de los bloques, segùn el último estado consistente de los objetos guardados. Mientras que el mapa de memoria contiene el estado de ocupaciòn de las pàginas de todos los objetos que estàn siendo usados actualmente por los procesos.

2.4.2.2.2 Las pàginas modificadas se pasan de memoria a disco.

Todas las pàginas de un objeto que han sido modificadas se deben pasar de memoria a disco. Se revisa el arreglo "MV[0]" para cada una de las pàginas del objeto, si en alguna de ellas està encendida la bandera que indica que ha sido modificada (vèase "MV[0]" en la figura 11, antes del compromiso), se le busca un nuevo bloque de disco con la ayuda del arreglo "MMap". Una vez que se encuentra un bloque libre, se pasa ahí la pàgina modificada. Si la pàgina no ha sido modificada (sòlo se leyò) no es necesario reescribirla. En la figura 11, antes del compromiso, los bloques "h" y "t" estàn libres, como se ve en el arreglo "MMap". Por lo que en ellos se pueden guardar las pàginas modificadas "c" y "p" que muestra el arreglo "MV[0]".

2.4.2.2.3 Liberar espacio de memoria y disco.

Cada vez que se pasa una pàgina modificada de memoria a disco, se actualizan los arreglos "MV[0]" y "MMap", poniendo en el primero el nùmero del nuevo bloque fìsico que ocupa la pàgina lògica, y en el segundo cambiando de "1" a "0" el "bit" que corresponde a el bloque viejo, y haciendo la operaciòn inversa para el bloque nuevo. Estas últimas actualizaciones tambièn se ejecutan para el arreglo temporal "TMap". Para observar el resultado de esta operaciòn hay que comparar los arreglos "MV[0]", "MMap" y "TMap" antes y despúes del compromiso en la figura 11.

2.4.2.2.4 Respaldo de "MV[0]" en disco.

El nuevo estado del arreglo "MV[0]", que contiene las direcciones de los nuevos bloques donde reside el objeto se debe guardar en disco, en particular sobre "DV[0]". Por razones de seguirdad no se escribe directamente "MV[0]" sobre "DV[0]", ya que si ocurriera alguna falla antes de que se terminara la escritura, "DV[0]" quedaría en un estado inconsistente. Para evitar este problema, primero se copia "DV[0]" en "DV[1]", con lo que se tiene una copia segura que se puede usar para recuperar el sistema en caso de falla. Y despuès se copia "MV[0]" en "DV[0]". En la figura 11, despuès del compromiso, se indica mediante una flecha como se copia el contenido de "MV[0]" a "DV[0]". Obsérvese que previamente se habìa copiado el contenido de "DV[0]" en "DV[1]".

2.4.2.2.5 Respaldo de "TDmap" en disco.

Una vez que se ha actualizado "TMap" se debe guardar en disco. La actualización consiste en marcar como ocupados los nuevos bloques donde residen las páginas modificadas y como libres los bloques anteriores. Sin embargo, la escritura de "TMap" no se hace directamente (por las mismas razones de seguridad que se mencionan en el párrafo anterior) sobre el "DMap" del que se copió, sino en el otro "DMap", es decir, el que indica el valor complementado de la bandera "DMapswitch". Una vez que se haya ejecutado la copia, se complementa el valor de la bandera "DMapswitch" para indicar cual es el mapa válido. En la figura 11, después del compromiso, se indica mediante una flecha como se ha copiado el contenido de "TMap" a "DMap[1]", ya que antes del compromiso "DMapswitch" es igual a cero. "DMap[0]" contiene el estado consistente anterior, de ocupación de los bloques.

Finalmente, en la figura 12 se muestra como quedan las estructuras de datos después de haberse ejecutado el compromiso para el objeto del ejemplo.

2.5 Estructuras de datos que ayudan a mantener el Objeto Persistente dentro del sistema.

2.5.1 Tabla de Objetos Persistentes del proceso.

En esta tabla el sistema va a guardar datos específicos de cada uno de los objetos que este manejando cada proceso. Casi todos los datos de la tabla varían en función de los accesos que cada proceso haga a sus objetos. La tabla se crea al mismo tiempo que el proceso y ambos se eliminan cuando el sistema detecta que el proceso ha terminado. (Véase figura 13.) En esta tabla hay una entrada diferente para cada proceso que este usando objetos. En cada entrada se mantiene una lista de los objetos que esta usando el proceso. En seguida se presenta la lista de la información que se maneja para cada objeto que se incluya en la tabla.

i) Identificador único del Objeto Persistente.

ii) Apuntador al contenido del Objeto Persistente.

iii) Fragmentos del Objeto Persistente usados por el proceso.

iv) Objeto Persistente nuevo o viejo.

v) Datos de un Objeto Persistente recién creado.

En seguida se explica que es cada uno de estos datos.

2.5.1.1 Identificador único del Objeto Persistente.

Sirve para identificar a que objeto pertenece esta entrada de la tabla.

2.5.1.2 Apuntador al contenido del Objeto Persistente.

Un apuntador al último "byte" del contenido total (incluyendo fragmento inicial y continuaciones) del objeto que acceso el proceso.

2.5.1.3 Fragmentos del Objeto Persistente usados por el proceso.

Si el proceso accesa un objeto que no tiene una sòla pàgina, sino varias de ellas, se tiene una lista de los identificadores de todos los fragmentos del objeto sobre los que ha trabajado el proceso, indicando además si es que los ha modificado.

2.5.1.4 Objeto Persistente nuevo o viejo.

Un atributo que indica si el objeto es nuevo o viejo. Este atributo indica si el objeto ya existía o no en el sistema. Cuando un proceso crea un objeto, se le califica como nuevo y se le genera un identificador temporal, mismo que se introduce en esta tabla. Si el proceso solicita acceso a un objeto que ya existe en el sistema, se introduce su identificador único en esta tabla y se le califica como viejo.

2.5.1.5 Datos de un Objeto Persistente reciên creado.

Cuando se trata de un objeto nuevo, este campo de informaciòn se llena, de otra manera permanece vacio. Los componentes de este campo son:

i) Una direcciòn que apunta al àrea de memoria donde se encuentra actualmente el contenido del objeto. A través de ésta direcciòn el usuario coloca la informaciòn que va a almacenar en su nuevo objeto.

ii) El tamaño del objeto reciên creado, esto es, el tamaño del àrea que se reservò para guardar los datos iniciales del objeto.

2.5.2 Tabla de Objetos Persistentes del sistema.

Cada vez que se solicita un objeto de los que estàn almacenados en disco, el sistema necesita tener cierta información para poder controlarlo. Dicha información se pone en la tabla del sistema. (Vèase figura 14.) Para poner datos en ella, primero se introduce el identificador único del objeto en una función de "hash" y èsta indica una entrada de la tabla. Puede suceder que màs de un identificador genere la misma entrada de la tabla al introducirlo en la función de "hash". Por lo que en cada entrada se puede llegar a tener una lista de objetos. A continuación se da la lista de los datos que se van a guardar en èsta tabla para cada objeto que se ponga accesible.

i) Identificador único del Objeto Persistente.

ii) Candado de acceso al Objeto Persistente.

iii) Tipo de respaldo para el Objeto Persistente.

iv) Lista de procesos bloqueados que esperan un Objeto Persistente.

v) Fragmentos del Objeto Persistente en la memoria.

vi) Procesos en acceso concurrente sobre un Objeto Persistente.

Cada vez que un proceso deséa hacer acceso a un objeto que ya tiene datos en la tabla, se verifica que no vaya a generar conflictos con los demàs procesos que ya estan trabajando de manera concurrente sobre el objeto. Solamente si esto se puede garantizar se le permite el acceso al proceso màs reciente.

En seguida se describen cada uno de los datos de la lista anterior.

2.5.2.1 Identificador único del Objeto Persistente.

Sirve para identificar a que objeto pertenece toda la información que se esta listando.

2.5.2.2 Candado de acceso al Objeto Persistente.

Los candados implantados actualmente son: lectura compartida, lectura exclusiva y escritura exclusiva. En este campo se guarda el tipo de candado de concurrencia que solicitò el primer proceso que puso accesible el objeto. Todos los procesos que despuès quieran trabajar con el objeto deben ser compatibles con este candado.

Candado solicitado.

Candado asignado	LSH	LEX	WEX
LSH	si	no	no
LEX	no	no	no
WEX	no	no	no

LSH = Lectura compartida.
LEX = Lectura exclusiva.
WEX = Escritura exclusiva.

Tabla 1. Compatibilidad de candados.

2.5.2.3 Tipo de respaldo para el Objeto Persistente.

El primer proceso que pone accesible el objeto indica con ésta bandera si el objeto se maneja con respaldo ("shadow file") o sin él. Si se solicita el objeto sin archivo sombra, se le hacen modificaciones y se hace compromiso (commit) a su nuevo estado, el contenido actual se reescribirá sobre el contenido original, lo que podría provocar inconsistencias en el caso de que ocurriera alguna falla durante la escritura.

2.5.2.4 Lista de procesos bloqueados que esperan un Objeto Persistente.

Cuando se le solicita un objeto al sistema y éste niega el acceso por incompatibilidad de candados, se puede tomar una de dos acciones: la primera es "abortar" el proceso para que más tarde vuelva a intentar su acceso y la segunda es "bloquearlo", para que cuando el objeto este disponible, el proceso compita por lograr su acceso con otros procesos que estén en las mismas condiciones. Este dato es una lista de procesos que están bloqueados porque el candado que solicitan es incompatible con el que tiene actualmente el objeto. Y que están esperando a que éste se desocupe para poder accesarlo. Esta es una lista del tipo "FIFO", es decir, el primero que llega es el primero que se atiende.

2.5.2.5 Fragmentos del Objeto Persistente en la memoria.

Puede suceder que el objeto al que pertenece esta información este repartido en varias pàginas, en este campo se tiene una lista de los fragmentos del objeto que han pasado de disco a memoria. El identificador del fragmento se anota una sola vez, es decir, no se repite ningún identificador. Para cada fragmento se tiene apuntado el número de procesos que lo están usando. Además, si algún fragmento es escrito por algún proceso se marca como modificado.

2.5.2.6 Procesos en acceso concurrente sobre un Objeto Persistente.

El número de procesos que estan haciendo acceso concurrente sobre el objeto.

2.6 Mantenimiento de la zona de almacenamiento de Objetos Persistentes.

2.6.1 Mantenimiento desde el punto de vista del usuario.

Durante la operación normal del sistema (tomando en cuenta sólo los objetos), el usuario puede crear, modificar y también eliminar toda referencia a algunos de sus objetos. Cuando un usuario ya no desèa usar un objeto, hace una llamada al sistema para indicarle que dicho objeto se puede borrar. Antes de calificar al objeto como borrado el sistema verifica que el proceso solicitante pertenezca al creador del objeto. Si se cumple esta condición, el sistema califica como borrado el apuntador que tiene desde su raíz hacia el objeto; de otra manera no hace nada. El usuario sabe que borrar el identificador del objeto no equivale a eliminar su contenido, pero también sabe que cierto tiempo después un proceso de recolección de basura se encargarà de hacerlo.

2.6.2 Mantenimiento desde el punto de vista del sistema.

Cada vez que un usuario borra un objeto, el sistema toma nota de que ese objeto ya no esta siendo usado, pero no lo elimina del almacenamiento, por lo que puede llegar el momento en que se tengan uno o varios objetos en el sistema (en la memoria permanente) y que nadie este haciendo referencia a ellos. El espacio que ocupan estos objetos se esta desperdiciando, ya que no se van a volver a usar. Lo màs conveniente es removerlos. Para ejecutar esta tarea se cuenta con el proceso "Recolector de Basura" (RB), cuya función es detectar y borrar todos los obje-

tos que no estén siendo apuntados por ningún usuario del sistema. Además de la acción de borrado, el RB puede llevar a cabo funciones de compactación y de recolocación. La parte restante de este capítulo esta dedicada a explicar el esquema de RB que se va a usar en este trabajo. Este se basa, en parte, en el procedimiento que describe Bishop en su tesis [Bishop 77].

La RB es un procedimiento que se ejecuta cada vez que se agota el espacio de almacenamiento secundario asignado a una "área", o cuando el procesador tenga tiempo libre. El acceso a los objetos guardados en una área y la recolección de basura sobre la misma son actividades exclusivas, es decir, no se pueden ejecutar en paralelo.

2.6.2.1 Organización lógica en el disco para la Recolección de Basura.

El almacenamiento secundario esta dividido físicamente en fragmentos que contienen una cantidad fija de "bytes", en este caso 512. A estos fragmentos se les llama, indistintamente, bloques o páginas. Se define una "área" del almacenamiento secundario como una agrupación de un número fijo de los bloques mencionados. El tamaño del área esta determinado por el número de bloques que tiene. Cada uno de estos bloques es capaz de contener una página lógica. Hay que recordar que un objeto existe en una página lógica o en varias de ellas. (Véase figura 2.)

La razón por la que el almacenamiento secundario se divide en áreas es que de esa manera es más fácil ejecutar el proceso de recolección de basura. Es decir, en vez de llevar a cabo la recolección de basura en todo el dispositivo de almacenamiento secundario, que tomaría una cantidad de tiempo "x", se ejecuta sólo sobre una área, lo que tarda sólo un fragmento del tiempo "x". Logrando con ésto que cada vez que el procesador tenga que hacer recolección de basura, se suspenda el acceso a los objetos durante un lapso de tiempo menor (comparado con el que tomaría recolectar todo el dispositivo). El precio que hay que pagar por un procedimiento de recolección de basura más corto, es el mantenimiento de las estructuras de datos que hay en cada área, a diferencia de las que existirían si se manejara como un solo dispositivo.

El área y sus estructuras de datos se crean desde el momento en que se inicializa la parte del sistema que maneja los objetos. Así cuando algún proceso solicite servicio, trabaja sobre el área de manera transparente. Para poder llevar la cuenta y localizar los objetos que hay en una área, ésta contiene desde el momento de su creación, una estructura de datos llamada "area object". A través de ella se pueden encontrar todos los objetos que hay en una área y uniendo los que hay en todas las áreas, los objetos que hay en un dispositivo.

2.6.2.2 Estructura de datos "Area Object".

Hay una estructura de este tipo por cada área en que esta dividido el dispositivo de almacenamiento secundario. En dicha estructura se lleva nota de los siguientes datos:

i) Identificador único del área.

ii) Lista de bloques libres. Una lista de los bloques pertenecientes a esta área que se encuentran libres. Se pueden solicitar más bloques para el área y los que el sistema proporcione se agregan a esta lista.

iii) Lista de bloques llenos. Una lista de los bloques de esta área que están completamente ocupados por objetos que residen en ellos.

iv) Lista de bloques parcialmente ocupados. Una lista de los bloques que pertenecen a esta área y que están parcialmente ocupados con objetos. Es decir, los objetos ocupan sólo un fragmento del bloque, por lo que éste tiene espacio libre.

v) Cuota de almacenamiento. Número máximo de bloques de almacenamiento físico asignados al área.

vi) Almacenamiento usado. Número de bloques del almacenamiento físico usados actualmente por el área, ya sea parcial o totalmente llenos.

vii) Lista de referencias hacia esta área. Lista de las "inter-area links" de otras áreas que referencian objetos en esta área. Esta lista la mantiene el sistema. (Véase "inter-area links" más abajo.)

viii) Lista de referencias a otras áreas. Lista de las "inter-area links" en el área actual que referencian objetos en otras áreas. Esta lista también la mantiene el sistema. (Véase "inter-area links" más abajo.)

ix) Candado. Este candado se usa para prevenir el acceso concurrente a las listas de bloques libres, de referencias a esta área y de referencias a otras áreas, cuando se este haciendo la recolección de basura y el sistema intente hacer acceso a los objetos.

x) Objeto raíz que contiene la lista de identificadores de todos los objetos que están en esta área.

2.6.2.3 Apoyo que da el "Inter Area Link" (IAL).

La función que desempeñan los IALs es la de servir como enlace entre objetos que habitan en áreas diferentes del espacio de

almacenamiento. Un IAL se crea cuando un objeto que habita en una área hace referencia a otro objeto que está en una área diferente. El IAL se coloca en el área donde se encuentra el objeto que hace la referencia, y contiene (el IAL) el identificador del objeto referido (el que habita en la otra área). El IAL lleva la cuenta del número de objetos que lo están utilizando como enlace, esto es, el número de objetos que están en la misma área que el IAL y que lo están utilizando para referirse a un objeto que habita en otra área. Conforme los objetos se van desligado del IAL, es decir, los objetos lo van dejando de usar para hacer su referencia, se va decrementando la cuenta, en el momento que ésta llega a cero, se puede borrar el IAL. En la figura 15 se puede ver un ejemplo sencillo del IAL, en el area "A" el objeto "x" hace referencia al objeto "y" del área "B", dicha referencia pasa a través de un IAL en el area "A".

El RB considera accesibles a los objetos que son referidos desde fuera del área, es decir, aquellos que son apuntados a través de un IAL. Todos los objetos que son accesibles no se deben borrar. Si el IAL que estaba apuntando a un objeto "x" desaparece, y además dentro del área no hay ningún otro objeto que esté apuntando al objeto "x", a este se le puede considerar como candidato a ser borrado. Acción que se ejecutará en la próxima pasada del recolector de basura por esta área.

2.6.2.4 Estructura de Datos "Inter-Area Link".

Un "Inter Area Link" (IAL) es una estructura de datos que el sistema crea automáticamente para cada referencia que un objeto haga hacia otro objeto que no esté contenido en la misma área. Esta estructura contiene:

i) El identificador del objeto al que se hace referencia a través de éste IAL.

ii) El área donde se creó éste IAL. Que es la misma en que está el objeto que hizo la referencia.

iii) Un apuntador a otro IAL para formar la lista de referencias externas. Es decir, se forma una lista con todos los IALs que contiene esta área y que hacen referencias a objetos que quedan fuera de dicha área.

iv) Un apuntador a otro IAL para formar la lista de referencias hacia dentro del área. Es decir, se forma una lista con todos los IALs que hay en las otras áreas del dispositivo (sin incluir ésta), que hagan referencia a un objeto que está contenido en ésta área.

v) El número de objetos que residen en la misma área que el IAL y que lo usan para apuntar a un objeto que está en otra área.

2.6.2.5 Manipulaciòn de los "Inter Area Links".

La actualizaciòn de las estructuras que ayudan al recolector de basura se lleva a cabo principalmente cuando se ejecuta la operaciòn de compromiso. Esto se debe, fundamentalmente, a que dichas estructuras operan sobre el contenido del dispositivo de almacenamiento secundario. Cuando se hace compromiso a un objeto modificado o a uno de reciente creaciòn, se buscan bloques de disco para guardar su nuevo contenido, en este caso se pueden necesitar los IALs. Se presentan las siguientes situaciones:

i) Cuando se hace compromiso a un objeto recièn creado, el sistema revisa la zona de apuntadores del objeto. Para cada apuntador guardado ahí verifica que el objeto a que hace referencia estè en la misma àrea donde va a residir el objeto recièn creado, si èsto no se cumple crea un IAL, mismo que liga a sus otras estructuras de datos.

ii) Si se està guardando un objeto, nuevo o viejo, y en la lista de bloques vacios del àrea ya no hay los suficientes para guardar todo el objeto, es necesario que parte del mismo quede en otra àrea. Si es una continuaciòn de objeto la ùnica que cambia de àrea, el fragmento anterior necesita ahora de un IAL para referirse a ella (a la continuaciòn). Por otra parte, si la continuaciòn que cambiò de àrea no es la ùltima del objeto, sino que a su vez apunta a otro fragmento que aùn està en el àrea anterior, se debe crear un IAL a través del cual la continuaciòn pueda apuntar al fragmento que le sigue.

   Suponiendo que son varios fragmentos del objeto los que cambian de àrea, solo el último fragmento del objeto que quedò en el àrea anterior necesita un IAL para referirse a su continuaciòn. Los demàs, como estàn en la misma àrea no necesitan IAL. A menos que haya un fragmento cuya continuaciòn aun estè en el àrea anterior, lo que seria equivalente al caso mencionado en el pàrrafo anterior.

iii) Tratàndose de un objeto que ya existia en el sistema y que se desèa volver a guardar, si es necesario cambiarlo totalmente de àrea se deben llevar a cabo cuatro revisiones, èstas son:

   Primera:
   Se revisa la zona de apuntadores del objeto que cambia de àrea. Para cada uno de los objetos apuntados que aùn residen en el àrea anterior se crea un IAL en el àrea nueva. A través de esos IAL el objeto que cambiò de àrea sigue apuntando a sus objetos. Para cada uno de los objetos apuntados que tenian un IAL, es decir, que estaban en otra àrea antes de que cambiara el objeto que apunta, se ejecuta uno de estos dos casos: a) si el objeto apuntado y el que apunta estàn ahora en la misma àrea, se decrementa la cuenta de objetos usuarios del IAL; si la cuenta llega a cero se elimina dicho IAL, de

otra manera sigue existiendo. b)si el objeto apuntado y el que apunta aún están en diferentes áreas, se decrementa la cuenta de objetos usuarios del IAL que está en el área anterior. Si la cuenta es cero, se elimina el IAL de esa área. Se revisan los IALs del área actual del objeto que apunta, si hay alguno que tenga el identificador del objeto apuntado, se incrementa su cuenta de objetos usuarios. Si no existe tal IAL, se crea uno.

Segunda:
Se deben revisar las zonas de apuntadores de todos los objetos que están en el área anterior del objeto que cambió de área. El objetivo es encontrar todos los apuntadores que los objetos del área anterior tienen hacia el objeto que cambió. Si existen dichos apuntadores se crea un IAL y el valor de su contador de objetos usuarios es igual al número de apuntadores encontrados.

Tercera:
La tercera revisón se hace sobre la lista de referencias de entrada, ya que puede suceder que algún objeto de otra área este apuntando al objeto que cambió de área. Si ésto sucede, y el área del objeto apuntado y del apuntador (IAL) son diferentes, se debe reubicar el IAL. Si ahora las áreas coinciden, se debe eliminar el IAL.

Cuarta:
Como el objeto que cambió de área puede ocupar varias páginas, hay que revisar si en alguna de ellas hay otros objetos, o fragmentos de estos, que no pertenezcan al objeto que cambió. Si existen esos objetos, o sus fragmentos, van a ser arrastrados al cambio de área cuando se cambie la página física en que residen (la página lógica no cambia) por lo que es necesario ejecutar para cada uno de ellos las tres revisiones anteriores.

2.6.2.6 Ejemplo del manejo de los IALs.

En la figura 16 se puede apreciar el manejo de los objetos dentro de las áreas y también se puede ver como se usan los "area object" y los IALs. El objeto "x" en el area "A" tiene un apuntador al objeto "y" y al objeto "w" del area "B" y al objeto "z" del area "C". En la figura 16 se puede observar como las referencias se mantienen a través de los IALs. Los IALs del área "A" forman una lista de referencias externas, esa lista esta apuntada desde el "area object" del área "A". (Campo "D" de la estructura "área" en el área "A" de la figura. 16) De la misma manera, el objeto "y" del área "B" hace referencia al objeto "v" del área "A". El IAL que se usa en este caso es el único elemento de la lista de referencias externas de la estructura "area" del área "B". En el área "C" el objeto "z" hace referencia al objeto "x" del área "A". El IAL usado es también el único elemento de la lista de referencias externas de la estructura

"àrea" del àrea "C".

La lista de referencias internas para el àrea "A" esta formada por el IAL que hay en el àrea "B" y el IAL que hay en el àrea "C". Estas son las referencias que los objetos que estàn en otras àreas hacen a los objetos que estàn en el àrea "A". Vèase la lista de IALs que apunta el campo "I" de la estructura "àrea" del àrea "A". De la misma manera la lista de referencias internas del àrea "B" esta formada por los dos primeros IALs que hay en el àrea "A", como se puede ver en la lista apuntada por el campo "I" de la estructura "àrea" del àrea "B". Para el àrea "C" la lista de referencias de entrada esta compuesta por un solo IAL, èste es el último que aparece en el àrea "A"; vèase la lista de IALs apuntada por el campo "I" de la estructura "àrea" del àrea "C".

2.6.2.7 Funcionamiento general del Recolector de Basura (RB).

El proceso de RB esta dividido en tres fases. La primera es la de marcado, la segunda es la de compactaciòn y la tercera es la de recolocaciòn. El marcado de los objetos accesibles se ejecuta en dos etapas. La primera etapa es para revisar la lista de los identificadores de los objetos que estàn activos en el àrea, es decir, los que no estàn borrados. Para cada identificador de esta lista se marca el objeto como accesible, se revisa su zona de apuntadores y todos los objetos que estèn apuntados tambièn se marcan como accesibles. La segunda etapa es para revisar la lista de referencias internas, todos los objetos que estèn siendo apuntados desde algùn IAL de esta lista se marcan como accesibles, asi como tambièn los objetos a los que apunten. Los objetos que no hayan sido marcados son candidatos a ser borrados.

La segunda fase es la de recolecciòn del espacio libre y compactaciòn. Para cada objeto que se borra se busca cada una de las pàginas que ocupa. Si la pàgina sòlo esta ocupada por el objeto se pone de inmediato en la lista de libres. Si la pàgina esta compartida con otros objetos, se elimina el objeto que se debe borrar, si al hacerlo queda un espacio libre entre dos objetos consecutivos, se compacta la pàgina para que presente siempre el mismo tipo de ocupaciòn que muestra la figura 9.

La tercera fase es la de recolocaciòn, èsta consiste en pasar al sistema informaciòn acerca de la localidad que se desèa tener para un grupo de objetos. El sistema se encarga de colocar dichos objetos en la misma pàgina fisica si es posible, o en pàginas fisicas adyacentes, o en el peor de los casos en pàginas fisicas lo màs cercanas que sea posible.

Finalmente se efectùa el almacenamiento seguro del nuevo estado de las pàginas del àrea, asi como de las estructuras de datos que ayudan a ejecutar la recolecciòn de basura. Una vez hecho èsto se puede permitir de nuevo el acceso de los procesos a los objetos del àrea.

F I G U R A S

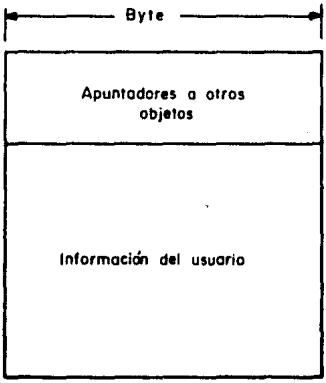


Figura 1. Un Objeto Persistente desde el punto de vista del usuario.

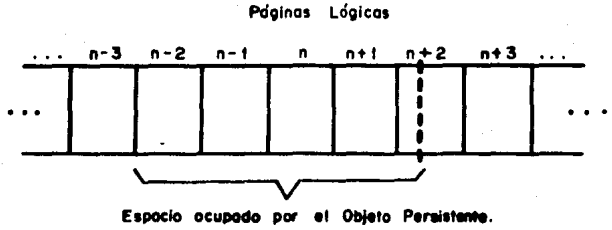


Figura 2. Un Objeto Persistente ocupando varias páginas.

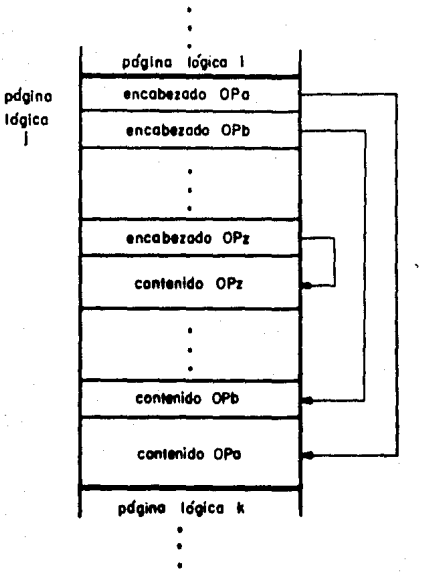


Figura 3. Página lógica con Objetos Persistentes dentro.

Figura 4. Identificador de un Objeto Persistente.

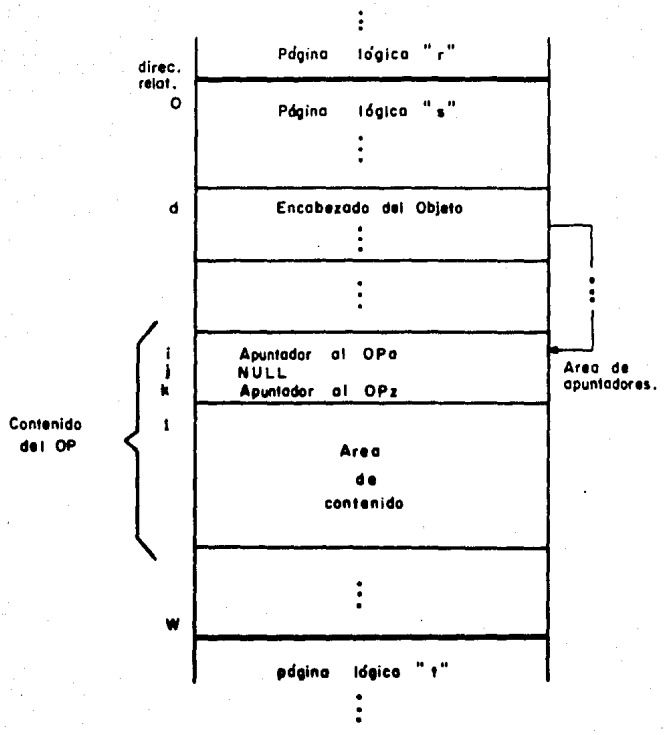


Figura 5. Datos sobre las referencias a otros Objetos Persistentes en el encabezado.

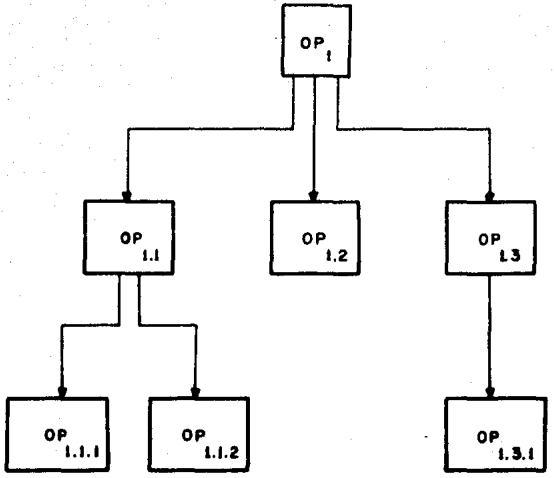


Figura 6. Un Objeto Persistente (OP) con varios descendientes.

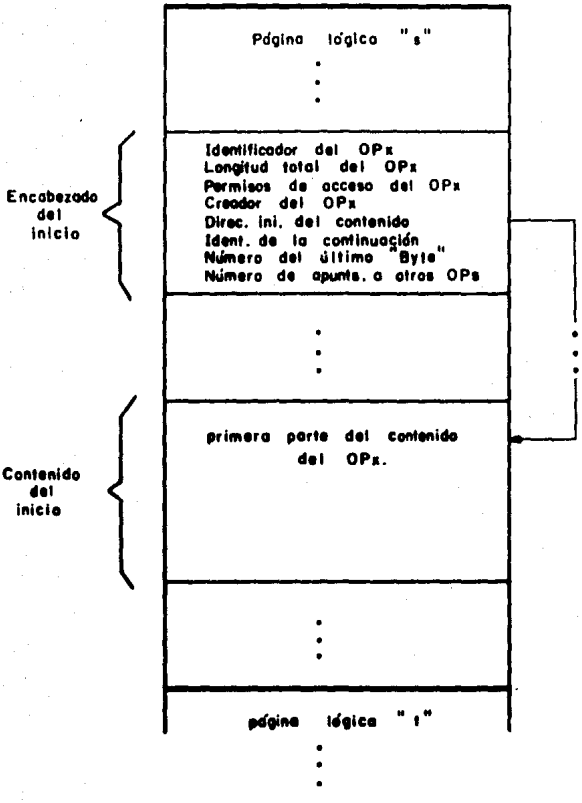


Figura 7. Representación de la primera parte de un Objeto Persistente (OP).

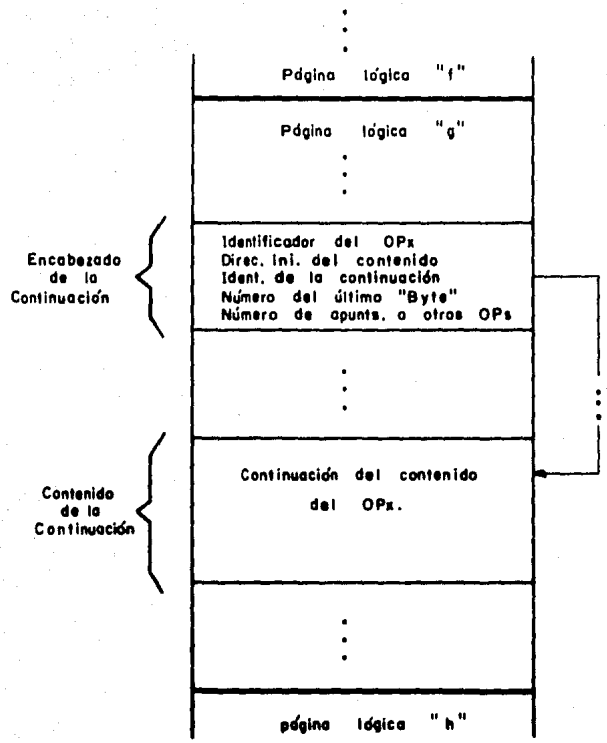


Figura 8. Representación de la continuación de un Objeto Persistente (OP).

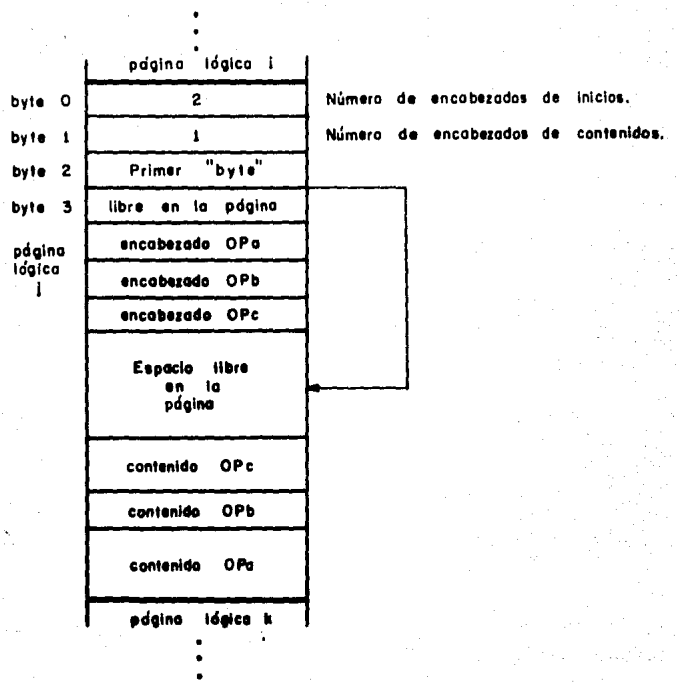


Figura 9. Página lógica mostrando sus "bytes" de control.

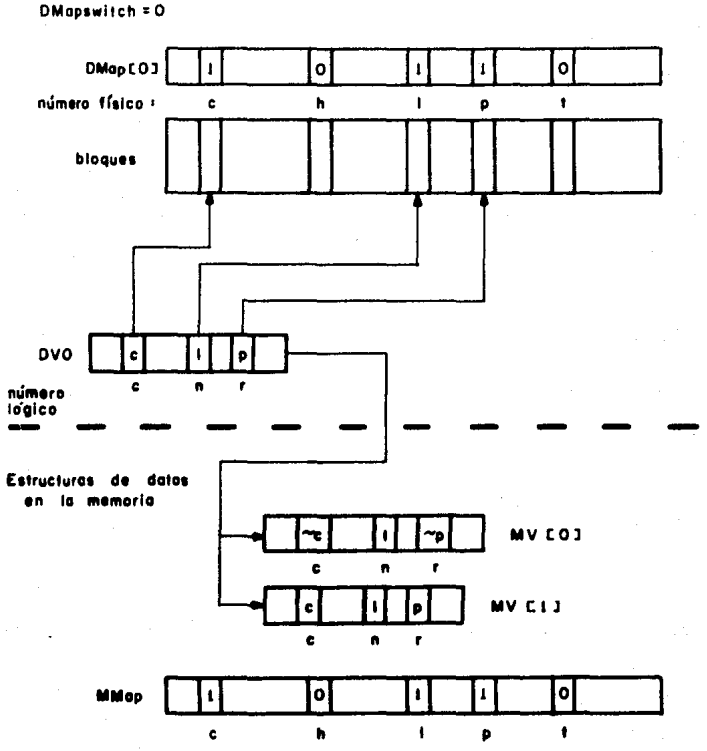
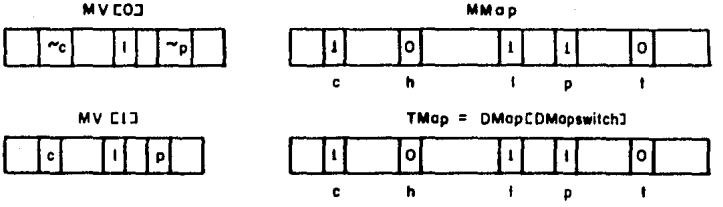


Figura 10. Instantánea del manejo de un Objeto Persistente.

Antes del compromiso (commit).

Memoria.



En disco se tiene: DMapswitch = 0

Después del compromiso (commit).

Memoria.

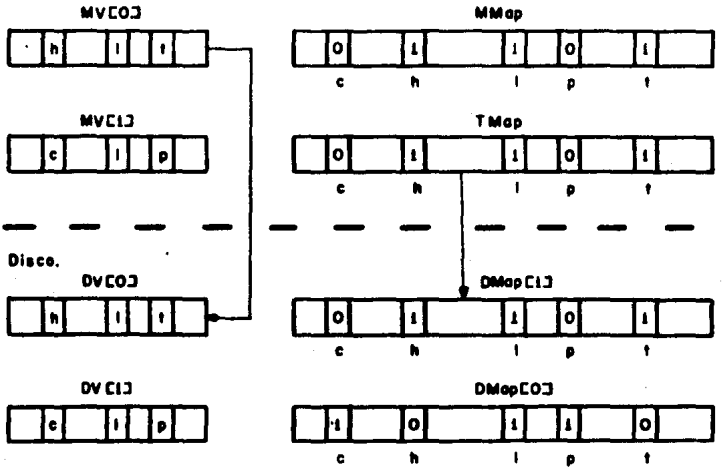


Figura 11. Estructuras de datos, antes y después del compromiso (commit).

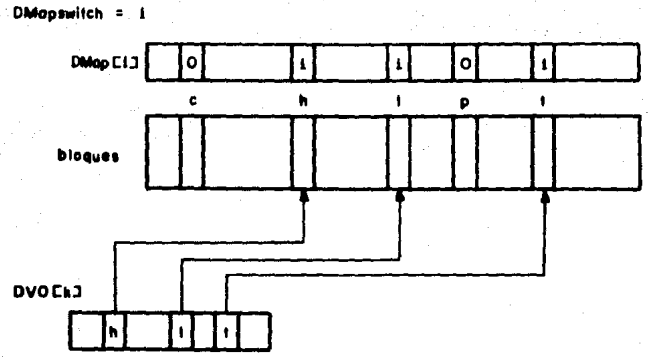


Figura 12. Estado final de las estructuras de datos, en disco, después del compromiso (commit).

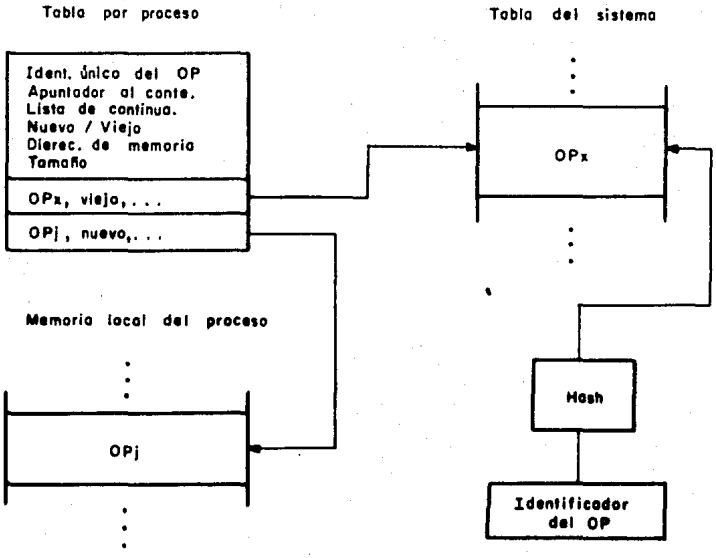


Figura 13. Tabla del Proceso para manejar Objetos Persistentes (OP).

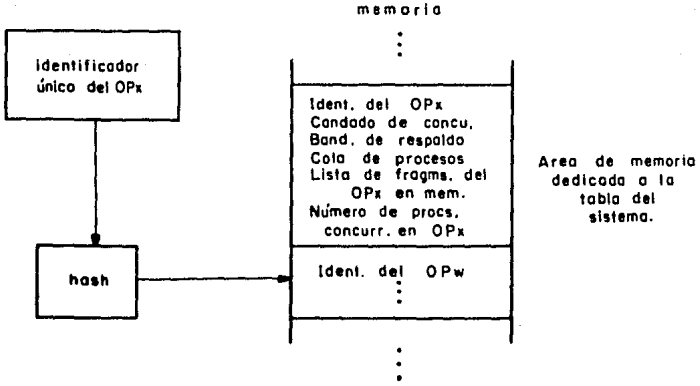


Figura 14. Tabla del sistema que contiene la información sobre los Objetos Persistentes (OP).

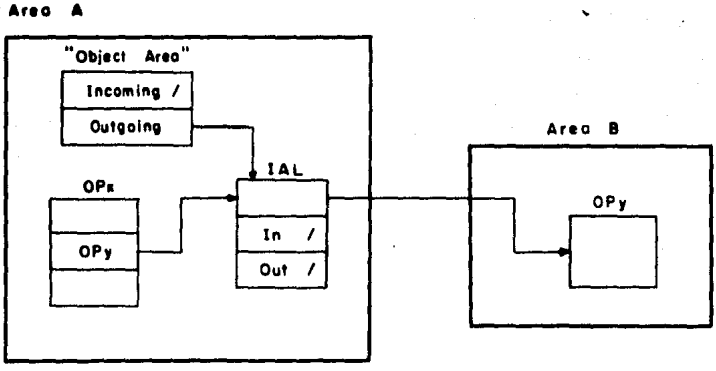


Figura 15. "Inter Area Links", para referencias entre áreas.

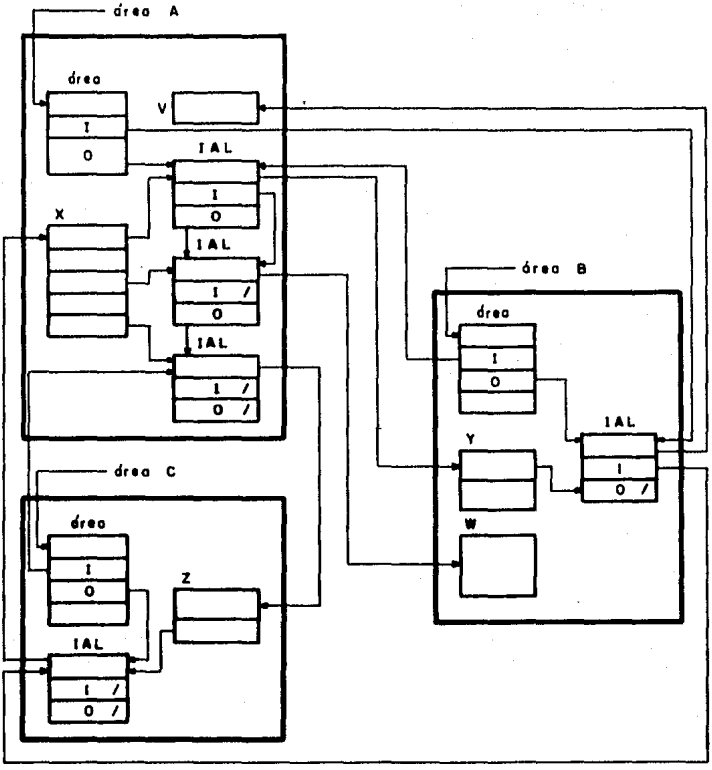


Figura 16. Estructuras de datos que ayudan al recolector de basura a manejar los Objetos Persistentes. (Tomada de la referencia [BISHOP 77].)

Capitulo 3

Desarrollo de la extensión para manejar los Objetos Persistentes.

Toda la programación que se desarrolló para apoyar el manejo de los objetos queda resumida en las funciones que se pueden llamar desde los programas de usuario. A continuación se describen dichas funciones. Estas se muestran como quedaron documentadas para su uso en general. El formato de la documentación es el que se usa en los manuales de "UNIX", esto se debe a que las funciones se crearon como una extensión a dicho sistema operativo, por lo que se consideró conveniente apegarse a ese formato.

Toda la programación que se documenta en esta sección fue desarrollada en el lenguaje de progrmación "C" y se implantó en la computadora "TOWER" de la compañia "NCR", que es controlada por el sistema operativo "UNIX", versión "V".

NOMBRE

creaop - crea un Objeto Persistente para el usuario que lo invoca.

SINOPSIS

```
struct      topp                    /* elemento de la tabla de OPs
                                       por proceso */
{  int      opid;                   /* identificador del OP */
   int      ulbyac;                 /* ultimo byte del OP accesado*/
   struct   conts      *cntp;       /* ptr a las conts de este OP */
   char     edad;                   /* OP viejo o nuevo */
   struct   memnvo     *opnv;       /* ptr a un OP nvo en memoria*/
   struvt   topp       *prox;       /* sig entrada en la lsta*/
}  ;

struct      conts                    /* conts de un OP, segun la
                                       topp */
{  int      opid;                    /* ident de la continuacion */
   char     writ;                    /* ha sido escrito? si o no */
   struct   conts      *prox;        /* sig continuacion del OP */
}  ;

struct      memnvo                   /* solo para un OP nuevo */
{  char     *mem;                    /* ptr a su area de memoria */
   int      siz;                     /* taman~o del OP nuevo */
   int      numptrs;                 /* # de ptrs con que se creo*/
   int      *ptrs;                   /* dir de el vector de ptrs */
   int      prms;                    /* permisos con que se crea
                                       el OP */
}  ;

struct  topp  *sistp[MXPR];          /* ptr a cada lista de OPs X
                                       proceso */

struct topp *creaop(siz,apuns,numapuns,perms)
int   siz;                  /* taman~o con que se crea el OP */
int   *apuns;               /* lista de ptrs con q' se crea el OP*/
int   numapuns;             /* # de ptrs en la lista anterior */
int   perms;                /* permisos con que se crea el OP */
```

DESCRIPCION

Esta función permite crear un objeto, dicho objeto se crea en el área de "heap" del usuario. Para invocar esta función se le deben pasar como parámetros: el tamaño del objeto; un lista de los apuntadores con los que se va a crear, si el objeto no va a tener apuntadores la lista puede estar vacia; el número de apuntadores que tiene la lista mencionada, que debe valer cero en el caso que la lista este vacia; y los permisos de acceso para otros usuarios con que se va a crear el objeto, la codificación de permisos se hace al estilo "UNIX".

Al ejecutar la función se genera un elemento del mismo tipo que la estructura llamada "topp", vèase la sinopsis, los datos de este elemento se llenan de acuerdo a los parámetros que se pasaron en la llamada. También se genera un nuevo elemento

del tipo de la estructura "memnvo", vèase la sinopsis. En èste se guardan los siguientes datos: el apuntador al àrea de memoria donde va a residir el objeto, el tamaño del objeto, el número de apuntadores con que se creò, la lista de esos apuntadores y los permisos de acceso sobre el objeto que se darán a los usuarios.

Ya que se crearon las estructuras de datos donde va a estar la información del objeto, se averigua que entrada en la lista "sistp" (vèase la sinopsis) corresponde al proceso que hizo la llamada. Si el proceso ya habìa hecho uso de objetos, ya tiene una entrada definida en la lista, y entonces la nueva estructura "topp" sòlo se agrega a la entrada que le corresponde. Si es la primera vez que el proceso crea un objeto, se le asigna una entrada en la lista y se coloca ahì la estructura "topp" recièn creada.

La función devuelve el apuntador al elemento "topp" que se acaba de crear.

EJEMPLO

A continuaciòn se muestra un fragmento de un programa que crea un objeto con varios apuntadores.

```
char   *pmem;          /* apuntador al contenido del OP */
int    bufp[];         /* apuntadores a otros OPs */
struct topp  *ptrt;    /* ptr a la estruc. que representa al OP
                          creado */

/* crea un OP de 256 bytes, con 5 apuntadores y permisos de
   lecutra, escritura y ejecucion para el propietario (7),
   lectura y escritura para el grupo (6) y lecutra para el
   sistema (4)
*/
if ((ptrt=creaop(256,bufp,5,0764)) == NULL)
   printf("\n problemas al crear el OP");
else
{  pmem=ptrt->opnv->mem;
   /* se guarda el contenido del OP */
}
```

CONDICIONES DE FALLA

"Creaop" falla si hace falta memoria para crear sus estructuras de datos dinàmicas.

VALOR QUE REGRESA

Esta función regresa el valor "NULL" en caso de que ocurra cualquier error durante su ejecución, de otra manera regresa un apuntador a la estructura de datos (de tipo "topp") que se creò para manejar el objeto.

DIAGNOSTICOS

La función va detectando los errores que se pueden producir a lo largo de la ejecución, si alguno de ellos es fatal la labor se suspende, sin embargo, no se indica que tipo de error fue,

simplemente regresa el valor "NULL".

RESTRICCIONES

El tamaño màximo de un objeto recièn creado està limitado por la cantidad de memoria de que puede disponer el usuario en el momento de la creaciòn. El nûmero màximo de apuntadores que se pueden poner en un objeto recièn creado es de 120. Los permisos con que se crea un objeto son del tipo "UNIX".

PRECAUCIONES

Toda la informaciòn que se haya guardado en el objeto se pierde irremediablemente si no se ejecuta una operaciòn de compromiso antes de que termine el proceso.

COMENTARIOS

Uno de los datos importantes que quedan guardados junto con el objeto es el identificador del usuario que lo creò (identificador del usuario y grupo al que pertenece). El sistema encuentra quien es el creador del objeto identificando al usuario que ejecuta el proceso desde el cual se hace la llamada a està funciòn.

Esta es siempre la primera funciòn que se debe llamar cuando se trate de crear nuevos objetos.

VEASE TAMBIEN

Compromiso.

NOMBRE

compromiso - ejecuta el compormiso (commit) sobre todos los objetos persistentes que pertenecen al proceso que ejecuta la llamada.

SINOPSIS

```
int compromiso(opsc,opsv);
int *opsc;          /* el numero de OPs recien creados */
int opsv[];         /* los identificadores de c/u de los OPs
                       creados */
```

DESCRIPCION

Esta funciòn permite conservar el estado actual de todos los objetos sobre los que esta trabajando el proceso que ejecuta la llamada. La operaciòn se encarga de revisar la lista de objetos que esta usando el proceso; y de guardar el nuevo estado de cada uno de los fragmentos que este proceso haya modificado. Esto se lleva a cabo a travès de un procedimiento conocido como "archivo sombra" [LORIE 77]. El espacio de memoria que estaban usando los objetos que sòlo eran usados por este proceso se libera. Si algún objeto esta siendo usado por otro proceso se deja activo en la memoria.

Si el proceso creò objetos, para cada uno de ellos se obtiene un identificador ùnico en el momento de hacer el compromiso. Esta funciòn cuenta el nùmero de objetos recièn creados y junta en un arreglo los identificadores que les corresponden. Si el proceso no crea objetos la cuenta vale cero.

EJEMPLO

En seguida se muestra un fragmento de un programa que ejecuta el compromiso para sus objetos.

```
/* en este ejemplo el proceso puede crear cuando mas diez OPs
   los identificadores en el arreglo "opsidents" ocupan desde
   el elemento cero hasta el elemento nueve
*/
int opscreados;             /* numero de OPs creados */
int opsidents[10];          /* identificadores de los OPs
                               creados */

if (compromiso(&opscreados,opsidents) < 0)
   printf("\n ***** mal compromiso");
```

CONDICIONES DE FALLA

El compromiso falla si hace falta memoria para crear sus estructuras dinàmicas.

El compromiso falla si falta espacio para colocar los objetos en el dispositivo de almacenamiento secundario.

VALOR QUE REGRESA

En caso de que la funciòn detecte cualquier tipo de error regresa un valor negativo. Si la funciòn se ejecuta exitosa-

mente regresa un valor positivo.

Los valores que devuelve en los argumentos son: en "opsc" el número de objetos recién creados durante la operación y en "opsv" la lista de los identificadores de los objetos recién creados. En esta lista hay tantos identificadores como indique la cuenta "opsc". Los identificadores ocupan desde el elemento cero del arreglo hasta el "n-1", según la convención de índices de arreglos de "C". Si no se crea ningún objeto o el compromiso termina debido a un error, el argumento "opsc" contiene el número de objetos que se alcanzaron a crear bien y en el arreglo "opsv" están los identificadores de dichos objetos.

DIAGNOSTICOS

Pueden ocurrir diferentes tipos de error a lo largo de la ejecución de la función de compromiso, sin embargo, cuando ésta termina, el valor que devuelve sólo indica si acabó bien o mal.

RESTRICCIONES

Esta función ejecuta el compromiso para todos los objetos que este manejando el proceso.

PRECAUCIONES

Esta función guarda el contenido de todos los objetos que pertenezacan al proceso que ejecuta la llamada, por lo tanto, si hay algún objeto que contenga datos que no se deséan almacenar el usuario debe eliminarlos antes de hacer el compromiso.

Si el proceso solicitó algún objeto sin respaldo (sin archivo sombra), y ocurre alguna falla antes de que el compromiso se termine de ejecutar, el objeto puede quedar en un estado inconsistente.

COMENTARIOS

El sistema encuentra quien es el usuario propietario del proceso que emitió la llamada (identificador del usuario y grupo al que pertenece). Con esta información determina en que estructuras de datos debe buscar los objetos que el proceso está usando actualmente.

VEASE TAMBIEN

Creaop, Ponac.

NOMBRE

ponac - pone accesible un objeto para un usuario.

SINOPSIS

```
ponac(opid,rspl,cand)
int  opid;          /* ident del OP que se quiere accesar */
char rspl;          /* respaldo con el que se va a accesar */
char cand;          /* candado que se pone para accesarlo */
```

DESCRIPCION

Mediante el uso de esta función un usuario pide que se ponga a su disposición un objeto en particular. Con el primer parámetro de la llamada el usuario indica que objeto, con el segundo solicita el tipo de respaldo que desèa manejar para el objeto y finalmente, con el tercer parámetro establece el candado con el que quiere abrir el objeto.

EJEMPLO

Abajo se muestra un fragmento de un programa que se puede usar para que un usuario solicite el acceso a un objeto.

```
#define  NSHF 0   /* no hay respaldo para el OP */
#define  SHF  1   /* respaldo de archivo sombra para el OP */
#define  LSH  0   /* candado de acceso de lectura compartida */
#define  LEX  1   /* candado de acceso de lectura exclusiva  */
#define  WEX  2   /* candado de acceso de escritura exclusiva*/
#define  NOAC 15  /* error: un OP no se pudo poner accesible */

int   opid;   /* identificador del OP que se quiere accesar */
int   numerr; /* variable global que indica el error ocurrido*/

/*solicita un OP y si lo obtiene hace algunas actualizaciones*/
if (ponac(opid,SHF,WEX) < 0)
{  printf("\n  no lo pudo poner accesible: opid %x",opid);
   numerr = NOAC;
   exit();
}
printf("\n ya lo puso accesible");
/* procesa el OP */
```

CONDICIONES DE FALLA

"Ponac" falla si el identificador del objeto que se le pasa como argumento no corresponde a ninguno de los que existen en el sistema.

"Ponac" falla si el tipo de respaldo solicitado no es alguno de estos dos: "NSHF" o "SHF".

"Ponac" falla si el candado solicitado no es alguno de estos tres: "LSH", "LEX" o "WEX".

"Ponac" falla si los permisos de acceso que tiene guardados el objeto, no permiten el acceso al usuario propietario del proceso que hizo la solcitud.

"Ponac" falla si no hay memoria disponible para crear sus estructuras de datos dinàmicas.

VALOR QUE REGRESA

Si la funciòn puede poner accesible el objeto regresa un valor mayor que cero. En caso de que ocurra cualquier tipo de error regresa un valor negativo.

DIAGNOSTICOS

Esta funciòn puede detectar un error o alguna condiciòn que no le permita poner accesible el objeto, en cualquiera de estos casos, el ùnico còdigo que regresa para indicar la falla es un valor menor que cero.

RESTRICCIONES

La circulaciòn de los identificadores de los objetos entre los usuarios es libre, es decir, culaquier usuario puede tener los identificadores de los objetos que desee. Sin embargo, desde que el objeto es creado tiene asignados permisos de acceso para cada conjunto de usuarios en el sistema. Por lo que el hecho de tener el identificador del objeto no garantiza tener su acceso. Esto es, el objeto conserva los permisos de lectura y escritura que se van a dar tanto al usuario creador del objeto, como al grupo al que pertence èste y al resto de los usuarios. La funciòn "ponac" averigua en que grupo de usuarios queda el proceso que la llamò y compara los permisos que tiene asignados el objeto para ese grupo con el tipo de operaciòn que se solicita (lectrua o escritura), si los permisos que tiene el objeto permiten ejecutar la operaciòn el objeto se pone accesible, de lo contrario se niega el acceso.

PRECAUCIONES

Como el acceso que hacen los procesos sobre los objetos es impredecible, puede suceder que dos o màs procesos quieran usar un solo objeto al mismo tiempo, o en intervalos muy cortos. El primer proceso que pone accesible el objeto, le asigna un tipo de respaldo y un candado de operaciòn. Si despuès hay otro proceso que solicita el mismo objeto, el sistema verifica primero que el candado de operaciòn que quiere el nuevo proceso sea compatible con el que tiene asignado el objeto, si èsto no se cumple, se le niega el acceso al proceso màs reciente. Si los candados son compatibles el sistema verifica que los tipos de respaldo de ambos procesos sean iguales, en cuyo caso se permite el acceso al proceso màs reciente, de lo contrario se le niega.

COMENTARIOS

El sistema se encarga de averiguar el identificador del usuario dentro del sistema y el del grupo al que pertenece. Con estos datos consulta los permisos que tiene establecidos el objeto para cada conjunto de usuarios y determina si se le puede asignar el objeto al proceso.

VEASE TAMBIEN

Creaop, Modper.

NOMBRE
letxt - lee el contenido de un Objeto Persistente.

SINOPSIS
```
letxt(opid,bybs,nbys,buf)
int     opid;       /* ident del OP que se quiere leer */
int     bybs;       /* byte inicial de lectura */
int     nbys;       /* numero de bytes que se van a leer */
char    *buf;       /* deposito para poner los bytes leidos*/
```

DESCRIPCION
Con esta función se puede leer el contenido de cualquier objeto. El usuario se debe asegurar, antes de usar esta función, de que se le haya concedido el acceso al objeto. Los argumentos de la función se usan para lo siguiente: el primero identifica al objeto del que se quiere leer, el segundo indica a partir de que "byte" del contenido del objeto se va a leer, el tercero es el número de "bytes" que se van a leer y el cuarto tiene la dirección de un arreglo donde se van a depositar los "bytes" leidos.

EJEMPLO
```
#define  SHF  1    /* respaldo de archivo sombra para el OP*/
#define  WEX  2    /* candado d acceso d escrit exclusiva */
#define  NOAC 15   /* error: un OP no se pudo poner accesible*/
#define  ERLE 17   /* error: no se pudo leer de un OP */

int   numerr;      /* var global que indica el error ocurrido*/
int   opid;        /* ident del OP que se quiere accesar */
int   longi;       /* numero de bytes que se van a leer */
int   i;           /* indice */
char  bufr[20001]; /* vector para guardar lo leido */

if (ponac(opid,SHF,WEX) < 0)
{  printf("\n  no lo pudo poner accesible: opid %x",opid);
   numerr = NOAC;
   exit();
}
printf("\n ya lo puso accesible");
/* en este caso se lee a partir del primer "byte" de contenido
   del OP */
if (letxt(opid,0,longi,bufr) < 0)
{  printf("\n no lo pudo leer: opid %x",opid);
   numerr = ERLE;
   exit();
}
printf("\n lo que leyo del OP: \n");
for (i=0 ; i<longi ; i++) printf("\n byte %d : %x",i,bufr[i]);
```

CONDICIONES DE FALLA
"Letxt" falla si el identificador del objeto que se le pasa como argumento no corresponde a ninguno de los objetos que exsiten en el sistema.

"Letxt" falla si la posición inicial donde debe empezar a

leer queda fuera del rango de contenido del objeto. Es decir, se le pasa una posiciòn negativa o un número de "byte" que es mayor que el tamaño del contenido del objeto.

"Letxt" falla si al hacer la suma de la posiciòn inicial de lectura, màs el número de "bytes" que se van a leer, el resultado es mayor que el tamaño del contenido del objeto.

"Letxt" falla si no encuentra suficiente memoria para crear sus variables dinàmicas.

VALOR QUE REGRESA

En caso de que ocurra algún error al tiempo de ejecuciòn, regresa un valor negativo. Si la funciòn se ejecuta correctamente regresa un valor positivo.

DIAGNOSTICOS

Esta funciòn puede identificar varios tipos de errores, sin embargo, cuando su ejecuciòn termina por causa de uno de ellos, no especifica cual error ocurriò, simplemente envia de regreso un valor negativo.

RESTRICCIONES

Para ejecutar esta operaciòn se debe tener garantizado el acceso al objeto, de lo contrario regresarà un còdigo de error.

PRECAUCIONES

Cuando un objeto se pone accesible para lectura concurrente, los procesos que lo estàn usando compiten para adueñarse de èl y poder ejecutar sus lecturas. Si despuès de cierto tiempo ocurre que algún proceso es indefinidamente relegado cuando quiere tener acceso al objeto, el sistema lo detecta y lo elimina . Para èsto hay un proceso especial que se encarga de revisar periòdicamente la informaciòn del sistema. Con base en sus observaciones, este proceso especial determina si alguno de los procesos de lectura a caido en una situaciòn como la descrita para eliminarlo.

COMENTARIOS

El sistema obtiene el identificador del usuario que ejecutò la llamada a esta funciòn, y antes de efectuar la lectura verifica que el objeto permita esta operaciòn para ese usuario. Si no se cumple esta condiciòn, la funciòn lo interpreta como un error y termina sin leer nada. De esta manera se evita, por ejemplo, que alguien que solicita un objeto para ejecuciòn trate de leerlo.

VEASE TAMBIEN

Ponac.

NOMBRE

esctxt - escribe en el contenido de un Objeto Persistente.

SINOPSIS

```
esctxt(opid,bybs,nbys,buf)
int   opid;   /* ident del OP que se quiere escribir */
int   bybs;   /* byte a partir del que se empieza a escribir*/
int   nbys;   /* numero de bytes a escribir */
char  *buf;   /* vector que contiene los bytes a escribir */
```

DESCRIPCION

Esta funciòn permite escribir sobre el contenido de cualquier objeto. Los argumentos le indican a la funciòn los siguientes datos: el objeto sobre el que se quiere escribir; la posiciòn del primer "byte" a partir del cual se va a escribir; el nùmero total de "bytes" que se van a escribir y la direcciòn del arreglo donde se encuentran almacenados los "bytes" que se van a escribir sobre el objeto.

EJEMPLO

En seguida se muestra un fragmento de còdigo que se puede usar para escribir datos a un objeto.

```
#define  ERES 14     /* error: no se pudo escribir a un OP */

int   numerr;       /* var global q' indica el error ocurrido*/
int   opid;         /* ident del OP sobre el que se escribe */
int   longi;        /* num de "bytes" a escribir en el OP */
char  bufr[2000];   /* contiene los "bytes" a escribir */

/* en este caso se escribe a partir del primer "byte" de
   contenido del OP */
if (esctxt(opid,0,longi,bufr) < 0)
{  printf("\n no lo pudo escribir opid %x",opid);
   numerr = ERES;
   exit();
}
printf("\n ya lo escribio");
```

CONDICIONES DE FALLA

"Esctxt" falla si el identificador del objeto que se le pasa como argumento no corresponde a ninguno de los objetos que exsiten en el sistema.

"Esctxt" falla si la posiciòn inicial donde debe empezar a escribir queda fuera del rango de contenido del objeto. Es decir, se le pasa una posiciòn negativa o un nùmero de "byte" que es mayor que el tamaño del contenido del objeto.

"Esctxt" falla si al hacer la suma de la posiciòn inicial de escritura, màs el nùmero de "bytes" que se van a escribir, el resultado es mayor que el tamaño del contenido del objeto.

"Esctxt" falla si no encuentra suficiente memoria para crear sus variables dinàmicas.

VALOR QUE REGRESA

En caso de que ocurra algún error durante la ejecución de la función, èsta regresa un valor negativo. Si la función termina bien regresa un valor mayor que cero.

DIAGNOSTICOS

Cuando esta función detecta algún error simplemente termina devolviendo un valor negativo, sin precisar el tipo de error que detectò.

RESTRICCIONES

El proceso que trate de ejecutar esta función debe estar bien seguro de que el sistema ya le puso accesible el objeto deseado.

PRECAUCIONES

Si se escriben datos sobre el contenido de un objeto y despuès se toma la decisiòn de no almacenarlos, se debe reintegrar dicho objeto a su estado anterior de consistencia antes de ejecutar la operaciòn de compromiso, ya que de lo contrario esas modificaciones se harèn permanentes.

Si el proceso que ha escrito sobre los objetos no manda ejecutar la operaciòn de compromiso antes de terminar, todas las actualizaciones hechas se pierden irremediablemente.

COMENTARIOS

Si antes de escribir sobre un objeto se sabe que esas modificaciones no se van a guardar (por ejemplo, que se esten haciendo pruebas), se puede pedir que se ponga accesible el objeto sin respaldo, es decir, sin archivo sombra.

El sistema obtiene el identificador del usuario que ejecutò la llamada, y antes de efectuar la escritura verifica que el objeto permita esta operaciòn para ese usuario. Si no se cumple esta condiciòn, la función lo interpreta como un error y termina sin escribir nada. De esta manera se evita, por ejemplo, que alguien que solicita un objeto para lectura trate de escribirlo.

VEASE TAMBIEN

Compromiso, Ponac.

NOMBRE

leptr - lee un apuntador (a un objeto) del contenido de otro Objeto Persistente.

SINOPSIS

```
leptr(opid,ptrno)
int   opid;      /* ident del OP que contiene el apuntador */
int   ptrno;     /* numero de apuntador que se quiere leer */
```

DESCRIPCION

Cuando se dice que un objeto tiene el apuntador a otro objeto, tal apuntador es en realidad el identificador del objeto apuntado, por lo tanto, en la descripciòn de esta funciòn se usan con el mismo significado apuntador e identificador.

Los apuntadores estàn guardados en una zona del contenido del objeto que sòlo es manejada directamente por el sistema. Cada vez que un usuario quiere ver o modificar el contenido de la zona de apuntadores, lo tiene que hacer a través de esta funciòn o de otra que se llama "escptr". La razòn por la que se tienen los apuntadores en una zona especial, es porque de esta manera el sistema da apoyo a las ligas que se establecen entre los objetos apuntados y los apuntadores.

Esta funciòn lee un apuntador del contenido de un objeto. Los argumentos de la funciòn le indican los siguientes datos: cual es el objeto del que se va a leer el apuntador, y que nùmero de posiciòn consecutiva, a partir de cero, ocupa el mencionado identificador en la zona de apuntadores.

EJEMPLO

En seguida se muestra un fragmento de programa que lee de manera secuencial todos los apuntadores que contiene un objeto.

```
#define  ERLEP 9     /* error: no se pudo leer de un OP */

int   numerr;     /* var global q' indica el error ocurrido */
int   ii;         /* indice */
int   apun;       /* num de identificadores que tiene el OP */
int   ptr;        /* apuntador leido */
int   opid;       /* identificador del OP que se va a leer */

for (ii=0 ; ii<apun ; ii++)
   if ((ptr=leptr(opid,ii)) < 0)
   {  printf("\n no lo pudo leer: opid %x (ii %d)",opid,ii);
      numerr = ERLEP;
      exit();
   }
   else printf("\n apuntador leido: %d",ptr);
```

CONDICIONES DE FALLA

"Leptr" falla si el identificador del objeto que se le pasa como argumento no corresponde a ninguno de los objetos que existen en el sistema.

"Leptr" falla si el número de apuntador que se le pide leer es negativo o es mayor que el número total de apuntadores que tiene almacenados el objeto.

"Leptr" falla si no cuenta con la memoria suficiente para crear sus estructuras de datos dinàmicas.

VALOR QUE REGRESA
Cuando esta función termina correctamente devuelve el valor del identificador leido. En caso de que ocurra cualquier falla regresa un valor negativo (ningún identificador vàlido es negativo).

DIAGNOSTICOS
Cuando esta función detecta cualquier tipo de error termina devolviendo un valor negativo, sin precisar cual fue el error que ocasionó la terminación.

RESTRICCIONES
Antes de que cualquier proceso pueda efectuar la lectura de un apuntador debe solicitar y obtener el acceso al objeto del que desea leer. De lo contrario esta función regresarà un código de error.

PRECAUCIONES
Cuando un objeto se pone accesible para lectura concurrente, los procesos que lo estàn usando compiten para adueñarse de él y poder ejecutar sus lecturas. Si después de cierto tiempo ocurre que algún proceso es indefinidamente relegado cuando quiere tener acceso al objeto, el sistema lo detecta y lo elimina . Para esto hay un proceso especial que se encarga de revisar periódicamente la información del sistema. Con base en sus observaciones, este proceso especial determina si alguno de los procesos de lectura a caido en una situación como la descrita para eliminarlo.

COMENTARIOS
El sistema obtiene el identificador del usuario que ejecutó la llamada y antes de efectuar la lectura, verifica que el objeto permita esta operación para ese usuario. Si no se cumple esta condición, la función lo interpreta como un error y termina sin leer nada. De esta manera se evita, por ejemplo, que alguien que solicita un objeto para ejecución trate de leerlo.

VEASE TAMBIEN
Escptr, Ponac.

NOMBRE

escptr - escribe un apuntador (de un objeto) en el contenido de un Objeto Persistente.

SINOPSIS

```
escptr(opid,ptrno,ptr)
int   opide;   /* ident del OP sobre el q' se va a escribir */
int   ptrno;   /* posicion del ident que se va a escribir */
int   ptr;     /* nuevo identificador que se va a escribir */
```

DESCRIPCION

Cuando se dice que un objeto tiene el apuntador a otro objeto, tal apuntador es en realidad el identificador del objeto apuntado, por lo tanto, en la descripciòn de esta funciòn se usan con el mismo significado apuntador e identificador.

Los apuntadores estàn guardados en una zona del contenido del objeto que sòlo es manejada directamente por el sistema. Cada vez que un usuario quiere ver o modificar el contenido de la zona de apuntadores, lo tiene que hacer a travès de esta funciòn o de otra que se llama "leptr". La razòn por la que se tienen los apuntadores en una zona especial, es porque de esta manera el sistema da apoyo a las ligas que se establecen entre los objetos apuntados y los apuntadores.

Esta funciòn sirve para escribir el apuntador de un objeto, en el contenido de otro objeto. Los argumentos de esta funciòn se usan para indicarle tres datos, ellos son: primero: el objeto sobre el que se va a escribir; segundo: la posiciòn que ocupa el identificador en la zona de auntadores; y tercero: el valor del apuntador que se va a escribir.

EJEMPLO

En este ejemplo se presenta el fragmento de un programa que sirve para escribir, de manera secuencial, todos los identificadores que caben en la zona de apuntadores de un objeto.

```
#define  ERESP 10   /* error: no se pudo escribir a un OP */

int   numerr;      /* var global q' indica el error ocurrido*/
int   ii;          /* indice */
int   apun;        /* num de identificadores que tiene el OP */
int   opid;        /* ident del OP que se va a escribir */
int   idens[];     /* lista de idents que se van a escribir */

for (ii=0 ; ii<apun ; ii++)
   if (escptr(opid,ii,idens[ii]) < 0)
   {  printf("\n no lo pudo escribir opid %x (ii %d)",opid,ii);
      numerr = ERESP;
      exit();
   }
   else printf("\n apuntador escrito: %d)",idens[ii]);
```

CONDICIONES DE FALLA

"Escptr" falla si el identificador del objeto que se le pasa

como argumento no corresponde a ninguno de los objetos que existen en el sistema.

"Escptr" falla si el nûmero de apuntador que se le pide escribir es negativo o es mayor que el nûmero total de apuntadores que tiene almacenados el objeto.

"Escptr" falla si no cuenta con la memoria suficiente para crear sus estructuras de datos dinàmicas.

VALOR QUE REGRESA

Si la funciòn termina correctamente regresa un valor positivo. En caso de que ocurra cualquier tipo de error regresa un valor negativo.

DIAGNOSTICOS

Cuando esta funciòn detecta algûn error, termina sin ejecutar la escritura y devuelve un valor negativo, sin precisar el tipo de error que detectò.

RESTRICCIONES

Antes de que cualquier proceso pueda efectuar la escritura de un apuntador, debe solicitar y obtener el acceso al objeto en el que desèa escribir. De lo contrario esta funciòn regresarà un côdigo de error.

PRECAUCIONES

Si se escriben datos sobre el contenido de un objeto y despuès se toma la decisiòn de no almacenarlos, se debe reintegrar dicho objeto a su estado anterior de consistencia, antes de ejecutar la operaciòn de compromiso, ya que de lo contrario esas modificaciones se harân permanentes.

Si el proceso que ha escrito sobre los objetos no manda ejecutar la operaciòn de compromiso antes de terminar, todas las actualizaciones hechas se pierden irremediablemente.

Hay que tener presente que el nûmero de identificadores que puede almacenar un objeto se define cuando èste se crea, por lo tanto, tratar de escribir màs apuntadores de los que estân definidos no hace que se amplie la zona donde estân almacenados, sino que se marca como un error.

COMENTARIOS

Si antes de escribir sobre un objeto se sabe que esas modificaciones no se van a guardar (por ejemplo, que se estân haciendo pruebas), se puede pedir que se ponga accesible el objeto sin respaldo, es decir, sin archivo sombra.

El sistema obtiene el identificador del usuario que està ejecutando la llamada, y antes de efectuar la escritura, verifica que el objeto permita esta operaciòn para ese usuario. Si no se cumple esta condiciòn, la funciòn lo interpreta como un error y termina sin escribir nada. De esta manera se evita, por ejemplo, que alguien que solicita un objeto para

lectura trate de escribirlo.

VEASE TAMBIEN
Compromiso, Leptr y Ponac.

- modifica los permi eración que tiene un Ob Persistente.

SINOPSIS
```
modper(opid,nvoprm)
int     opid;     /* ident del OP que se va a modificar */
short   nvoprm;   /* codificacion tipo UNIX del nuevo permiso*/
```

DESCRIPCION
Cuando se crea un objeto, se le asignan una serie de permisos que definen de que manera lo van a poder accesar los usuarios del sistema. Si tiempo después el creador del objeto desèa relajar, o restringir màs el acceso a su objeto, ejecuta la llamada a esta función, a través de la cual se asignan una nueva serie de permisos de acceso para los usuarios.

EJEMPLO
El fragmento de código que se muestra abajo de este pàrrafo, se usa para asignar una nueva serie de permisos a un objeto cualquiera.

```
#define  NSHF 0   /* no hay respaldo para el OP */
#define  SHF  1   /* respaldo de archivo sombra para el OP */
#define  LSH  0   /* candado de acceso de lectura compartida*/
#define  LEX  1   /* candado de acceso de lectura exclusiva */
#define  WEX  2   /* candado de acceso de escritura exclusiva*/
#define  NOAC 15  /* error: un OP no se pudo poner accesible */
#define  ERMP 20  /* error: no se pudieron modificar los
                            permisos a un OP */

int      opid;          /* ident del OP que se quiere accesar */
int      numerr;        /* var global q' indica el err ocurrido*/
int      opscreados;           /* numero de OPs creados */
int      opsidents[10];        /* idents de los OPs creados */

/*solicita un OP y si lo obtiene hace algunas actualizaciones*/
if (ponac(opid,SHF,WEX) < 0)
{  printf("\n  no lo pudo poner accesible: opid %x",opid);
   numerr = NOAC;
   exit();
}
printf("\n ya lo puso accesible %x",opid);
/*los nuevos permisos son: O:rwx (7), G:rwx (7) y W:rwx (7)*/
if (modper(opid,0777) < 0)
{  printf("\n no pudo modificar permisos del opid: %x",opid);
   numerr = ERMP;
   exit();
}
printf("\n modifico permisos del op %x",opid);
if (compromiso(&opscreados,opsidents) < 0)
   printf("\n ***** mal commit");
```

CONDICIONES DE FALLA
"Modper" falla si el identificador del objeto que se le pasa

como argumento no corresponde a ningún objeto de los que están activos en el sistema.

"Modper" falla si el valor absoluto de los permisos que se le pasan como argumento, son menores que cero o mayores que el valor octal: 777.

"Modper" falla si no encuentra la memoria suficiente para crear sus estructuras de datos dinámicas.

VALOR QUE REGRESA
Esta función regresa un valor positivo en caso de que su ejecución haya sido correcta y regresa un valor negativo en caso de que ocurra cualquier tipo de falla.

DIAGNOSTICOS
Aunque esta función vaya revisando posibles causas de error a lo largo de su ejecución, cuando ésta termina por culpa de uno de ellos no informa cual fue, simplemente regresa un valor negativo.

RESTRICCIONES
Antes de que cualquier proceso trate de modificar los permisos de operación que tiene asignados un objeto, se debe asegurar de que tiene garantizado, por el sistema, el acceso a dicho objeto.

Esta función la puede ejecutar solamente el creador del objeto. En otras palabras, el único que puede modificar los permisos de operación que tiene un objeto es su creador.

PRECAUCIONES
Esta función únicamente escribe los nuevos permisos con los que va a operar el objeto de ese momento en adelante, sin embargo, para que estos permisos queden definitivamente grabados en el objeto, se debe ejecutar una operación de compromiso, de lo contrario la nueva asignación se pierde y el objeto queda con los permisos que tenía antes.

COMENTARIOS
El sistema averigua quien es el propietario del proceso que ejecuta la llamada a la función, es decir, obtiene el identificador del usuario y el del grupo al que pertenece. Con estos datos y con los que tiene el objeto para identificar a su creador, el sistema decide si se le permite o no, al proceso, ejecutar la operación de modificar los permisos. Ya que los permisos anteriores pueden permitir el acceso a diferentes tipos de usuarios, pero el único al que se le permite modificar dichos permisos es al creador del objeto.

VEASE TAMBIEN
Compromiso y Ponac.

NOMBRE

rmoprz - remueve un Objeto Persistente que ya no se va a usar, del Objeto raiz donde se encontraba.

SINOPSIS

```
rmoprz(opid)
int opid;        /* identificador del OP que se va a borrar */
```

DESCRIPCION

Cuando un usuario decide que ya no va a usar un objeto de su propiedad, ejecuta esta función para notificar al sistema que ese objeto se puede borrar del almacenamiento permanente. El sistema verifica dos cosas antes de borrar un objeto, la primera es que el solicitante sea en verdad el creador del objeto; y la segunda es que no haya ningún otro objeto dentro del sistema que esté apuntando al objeto que se va a borrar. Si ambas condiciones se cumplen, el sistema marca el objeto como borrado (aunque no lo borre directamente en ese momento).

EJEMPLO

El fragmento de código que se muestra abajo se usa para borrar un objeto que es propiedad del usuario que ejecuta el proceso.

```
#define  ERRM 21     /* error: no se pudo borrar un OP */

int     opid;       /* ident del OP que se quiere accesar */
int     numerr;     /* var global que indica el error ocurrido*/

if (rmoprz(opid) < 0)
{  numerr = ERRM;
   exit();
}
printf("\n se borro el OP. \n");
```

CONDICIONES DE FALLA

"Rmoprz" falla si el identificador del objeto que se le pasa como argumento no corresponde a ningún objeto de los que están activos en el sistema.

"Rmoprz" falla si no encuentra la memoria suficiente para crear sus estructuras de datos dinámicas.

VALOR QUE REGRESA

Esta función regresa un valor positivo en caso de que se termine de ejecutar correctamente. Si termina debido a algún error regresa un valor negativo.

DIAGNOSTICOS

Aunque esta función vaya detectando posibles causas de error a lo largo de su ejecución, cuando termina debido a alguna de ellas no indica de cual se trató, simplemente regresa un valor negativo.

RESTRICCIONES

El único quue puede borrar un objeto es su creador.

PRECAUCIONES

Si un usuario ejecuta una operaciòn de borrado sobre un objeto que le pertenece, pero el sistema detecta que hay otro objeto que tiene un apuntador al primero, no lo borra del todo. Es decir, para el usuario que solicitò que se borrara el objeto, este aparece como borrado. Pero para el (los) usuario(s) no propietarios que tengan un apuntador a dicho objeto, èste aún existe. Por lo tanto este objeto se eliminarà completamente hasta que no haya ningún otro objeto apuntàndolo.

COMENTARIOS

El sistema averigua quien es el propietario del proceso que ejecuta la llamada a la funciòn, es decir, obtiene el identificador del usuario y el del grupo al que pertenece. Con estos datos y con los que tiene el objeto para identificar a su creador, el sistema decide si se le permite o no, al proceso, ejecutar la operaciòn de borrar el objeto. Ya que al único que se le permite borrar un objeto es al creador del mismo.

VEASE TAMBIEN

Creaop.

NOMBRE

ligoprz - introduce el identificador de un Objeto Persistente en el objeto raíz del área donde se encuentra localizado.

SINOPSIS

```
ligoprz(opid)
int opid;        /* identificador del OP que se va a agregar */
```

DESCRIPCION

Después de que un usuario a obtenido el identificador de un objeto, a través de la operación de compromiso, puede solicitar que ese identificador se guarde de manera permanente en el objeto raíz del área donde se guarda el objeto. Esta función coloca un identificador de objeto en el objeto raíz del área donde se guarda el primero.

EJEMPLO

El fragmento de código que se muestra abajo se usa para poner el identificador de un objeto en el objeto raíz del área donde residen ambos.

```
#define  ERLG 27     /* error: no se pudo ligar un OP */

int   opid;        /* ident del OP que se quiere ligar */
int   numerr;      /* var global que indica el error ocurrido*/

if (ligoprz(opid) < 0)
{  numerr = ERLG;
   exit();
}
printf("\n se ligo el OP. \n");
```

CONDICIONES DE FALLA

"Ligoprz" falla si el identificador del objeto que se le pasa como argumento no corresponde a ningún objeto de los que estan activos en el área.

"Ligoprz" falla si no encuentra la memoria suficiente para crear sus estructuras de datos dinámicas.

VALOR QUE REGRESA

Esta función regresa un valor positivo en caso de que se termine de ejecutar correctamente. Si termina debido a algún error regresa un valor negativo.

DIAGNOSTICOS

Aunque esta función vaya detectando posibles causas de error a lo largo de su ejecución, cuando termina debido a alguna de ellas no indica de cual se trató, simplemente regresa un valor negativo.

RESTRICCIONES

El único que puede ligar un identificador al objeto raíz es su creador.

PRECAUCIONES

Cualquier objeto sobre el que se haya hecho compromiso queda guardado en el dispositivo de almacenamiento secundario. Sin embargo, cuando se ejecuta el proceso de recolecciòn de basura, sòlo los objetos que tengan su identificador dado de alta en el objeto raiz del àrea donde habitan van a permanecer. Esto es, si se hace compromiso sobre un objeto pero no se escribe su identificador en el objeto raiz del àrea donde habita, en la primera pasada del recolector sobre esa àrea se eliminarà dicho objeto.

COMENTARIOS

El sistema averigua quien es el propietario del proceso que ejecuta la llamada a la funciòn, es decir, obtiene el identificador del usuario y el del grupo al que pertenece. Con estos datos y con los que tiene el objeto para identificar a su creador, el sistema decide si se le permite o no, al proceso, ejecutar la operaciòn de dar de alta el identificador en el objeto raiz. Ya que al ùnico que se le permite hacer esta operaciòn es al creador del objeto.

VEASE TAMBIEN

Creaop.

NOMBRE

revocar - esta función regresa varios Objetos Persistentes a su estado anterior de consistencia.

SINOPSIS

revocar()

DESCRIPCION

La operacion de revocar regresa a su estado anterior de consistencia a todos los objetos que venía usando un proceso. La solcitud de reintegrar los objetos a su estado anterior puede surgir por necesidades del usuario, o cuando el sistema suspende algún proceso por errores de ejecución. Esta función saca de las estructuras de datos y quita de la memoria principal a todos los objetos que reintegra a su estado anterior.

EJEMPLO

A continuación se muestra un fragmento de código que sirve para ejecutar la revocación a todos los objetos que venía usando un proceso.

```
if (revoca() < 0) printf("\n error al ejecutar la revocacion.");
```

CONDICIONES DE FALLA

La revocación falla si no encuentra suficiente espacio en memoria para generar sus estructuras de datos dinàmicas.

VALOR QUE REGRESA

Esta función regresa un valor positivo en caso de que se termine de ejecutar correctamente. Si ocurre algun error regresa un valor negativo.

DIAGNOSTICOS

Aunque esta función vaya detectando posibles errores a lo largo de su ejecución, cuando termina por causa de uno de ellos no indica de cual se trató, simplemente regresa un valor negativo.

PRECAUCIONES

Al ejecutar esta operación se pierden irremediablemente todas las modificaciones que el proceso había hecho sobre todos los objetos que tenía accesibles.

COMENTARIOS

El sistema se encarga de averiguar quien es el usuario que mando ejecutar esta operación y con esos datos actualiza sus estructuras de datos y libera el espacio de memoria ocupado por sus objetos.

VEASE TAMBIEN

Creaop y Compromiso.

Capítulo 4.

Pruebas y Resultados de la extensión al Sistema Operativo.

En este capítulo se describen las pruebas que se llevaron a cabo sobre la extensión al sistema operativo y los resultados que ellas arrojaron.

4.1 Programas auxiliares.

Antes de hablar sobre las pruebas realizadas, se describirán tres programas que se hicieron para auxiliar en la tarea de depurar el código. Dichos programas son: uno que se encarga de escribir el contenido de todas las estructuras de datos que está usando la extensión para controlar los objetos; otro que escribe el contenido del dispositivo de almacenamiento perdurable y finalmente, otro que muestra cual es el contenido de las estructuras de datos del recolector de basura.

Como el primer programa permite conocer el estado de las estructuras de datos que maneja la extensión, éste se puede utilizar antes de hacer la llamada a una de las funciones del capítulo anterior, así como después de que dicha función se ejecuta. La llamada a este programa también se puede intercalar en el código de cualquiera de las funciones mencionadas. Usando este programa se puede saber exactamente como modifica cada una de las funciones de la extensión, el contenido de las estructuras de datos. Esto sirve como herramienta para afinar el comportamiento de cada una de las funciones y para detectar datos que nunca se usan, datos redundantes y datos que hacen falta.

El segundo programa se usa para revisar el estado del dispositivo de almacenamiento perdurable. Este programa se manda ejecutar antes y después de llevar a cabo alguna de las operaciones del capítulo anterior. A través de la ejecución del programa se sabe como estaba el almacenamiento antes y como queda después de haber ejecutado la operación. Este procedimiento es particularmente útil para probar las operaciones de creación de un objeto, escritura sobre un objeto, modificación de un apuntador a un objeto, pero sobre todo, para ver el resutlado de la operación de compromiso. Es decir, a través de los resultados que da este programa se puede observar como al hacer permanentes los resultados de una modificación al objeto, se abandona un bloque que contiene el estado anterior de consistencia y se escribe en otro

bloque lo que es el último estado de consistencia. Además, se pueden observar otras cosas, como por ejemplo, la forma en que se va llenando un bloque que está parcialmente ocupado, etcètera.

El tercer programa tiene como función mostrar el contenido de las estructuras de datos del recolector de basura. Esto es necesario para depurar el código del recolector, ya que cada vez que se ejecuta la operación de compromiso para un objeto, èste cambia de bloques físicos, por lo que se debe verificar que las estructuras se hayan actualizado con los números de los nuevos bloques.

Usando estos programas como herramientas de depuración se probaron las funciones que se describen en el capítulo anterior.

4.2 Pruebas a la extensión del Sistema Operativo.

Antes de diseñar los programas que se usaron para probar las funciones que se describen en el capítulo anterior, se revisaron las actividades que cada una de ellas debía llevar a cabo, a partir de esta revisión se determinó en donde se podrían producir los posibles errores (de cada una de ellas), tanto en el comportamiento de la función como en los datos que comparte con otras funciones. Los casos de prueba se diseñaron con el objetivo de cubrir de la manera más amplia los posibles lugares y circunstancias de ocurrencia de error. Al ejecutar dichas pruebas se encontraron errores del tipo previsto, pero no solo eso, también se encontraron algunos otros errores que no se esperaba que ocurrieran. Aunque los casos de prueba ejecutados pretenden asegurar el comportamiento correcto de las funciones, de la manera más amplia posible, no se descarta del todo que aún puedan aparecer algunos detalles de mal funcionamiento en el uso futuro de la extensión, mismos que se deben ir corrigiendo durante la etapa de mantenimiento de la extensión.

4.3 Creación de un Objeto Persistente.

Para probar al programa que crea los objetos se ejecutaron diferentes casos que lo único que hacían eran tres cosas: primero: llamar a la función que crea los objetos; segundo: una vez que ésta devolvía el apuntador a la estructura de datos donde se encuentra la dirección del objeto, se escribían en èl datos aleatorios y tercero: se revisaban las estructuras de datos para saber si se había hecho una buena creación del objeto.

4.4 Compromiso a un Objeto Persistente.

Una vez que se vio que los objetos se creaban correctamente se procedió a guardar su contenido usando la operación de compromi-

so. A lo largo de la ejecución del compromiso se fue intercalando el programa que muestra el contenido de las estructuras de datos de la extensión. También se ejecutó el programa que muestra el contenido del dispositivo de almacenamiento secundario, tanto antes del compromiso como después de él. A través de la ejecución de los casos de prueba y de las correcciones que estas generaban se dejó trabajando correctamente la función. Esto consiste básicamente en guardar el contenido de un objeto recién creado en el dispositivo de almacenamiento secundario; si se trata de un objeto que ya estaba creado, se actualiza su nuevo estado en bloques físicos libres y se abandonan todos los bloques anteriores.

4.5 Poner accesible un Objeto Persistente.

Ya que se tuvieron objetos guardados en el dispositivo de almacenamiento perdurable, se probó la función que los pone accesibles a los procesos usuarios. Se hicieron varios casos en los que se solicitaba alguno de los objetos que estaban almacenados y se revisaban las estructuras de datos para ver si en verdad dicho objeto ya era accesible. En estos casos surgió una pequeña dificultad, ésta es que todos los programas de prueba se están ejecutando desde la misma cuenta de usuario, por lo que la función siempre iba a encontrar un solo identificador de usuario, causando que se probara de manera limitada la verificación de permisos. Para evitar esto se hizo una modificación temporal en el código de la función, ésta consiste en pasar como parámetros el identificador del usuario y el del grupo al que pertenece, para que la función no lo tenga que averiguar (a través de la llamada "GETUID(2)" de la versión V de "UNIX") y de esta manera se puedan ir variando dichos parámetros y por lo tanto, probar más ampliamente el trabajo de la función.

4.6 Leer de un Objeto Persistente.

Una vez que ya se pueden crear, conservar y poner accesibles los objetos, se prueba leer su contenido. Mediante estos casos de prueba se verificó que el contenido que mostraban los objetos cuando se ejecutaba la función de lectura, era el mismo que se veía en el dispositivo de almacenamiento perdurable cuando se ejecutaba el programa que vaciaba su contenido. Los casos que se trataron más para detectar problemas fueron: cuando hay un objeto que ocupa sólo un fragmento de la página, el caso de un objeto que ocupa una página y un fragmento de otra y el caso de un objeto que ocupa varias páginas. En todos los casos se atacaron los problemas de leer los primeros "bytes", los últimos y todo el contenido del objeto. Se puso especial atención a los "bytes" que quedaban en el límite de almacenamiento de cada página, es decir, los "bytes" que ocupan las últimas localidades de una página y los que están en el inicio de otra. De esta manera se detectaron y corrigieron los errores que tenía esta

función.

4.7 Escribir a un Objeto Persistente.

Si el contenido de un objeto se puede leer correctamente, la siguiente prueba es escribir sobre él (modificar su cotenido) y volverlo a leer para verificar el cambio. En estos casos de prueba, igual que en los anteriores (los de lectura), se probaron las situaciones de objetos que ocupaban un fragmento de página, una página y un pedazo de otra y varias páginas. También se puso atención a los "bytes" de los límites para evitar los errores de desbordamiento de página. Se ejecutó la operación de compromiso para conservar todas las modificaciones, y con el programa que sirve para revisar el contenido del dispositivo de almacenamiento secundario, se corrigieron todos los errores hasta que se comprobó que en verdad la función escribía correctamente sobre el contenido del objeto.

4.8 Leer un apuntador a un Objeto Persistente.

Para verificar el desempeño de esta función se hicieron varios casos de prueba. En ellos se ejectuaba la lectura de algún apuntador en diferentes objetos, o de todos los apuntadores de un objeto (uno por uno). Y se comparaba lo que había leído la función con lo que mostraban los objetos cuando se sacaba su contenido del dispositivo de almacenamiento permanente. Esta función y la que sirve para leer del contenido del objeto son análogas y sus pruebas también son similares, la diferecia esta en la zona que pueden accesar y en que una puede leer un número variable de "bytes" mientras que la otra sólo lee un apuntador a la vez.

4.9 Escribir un apuntador a un Objeto Persistente.

Para probar esta función se hicieron varios casos de prueba, en ellos se ejecutaban tres pasos consecutivos: primero se ponía accesible un objeto, segundo: se leían sus apuntadores y finalmente se modificaban tales apuntadores (se escribían sobre el objeto). Para verificar que la función había trabajado bien se volvían a leer dichos apuntadores y finalmente se ejecutaba un compromiso para hacer permanentes los nuevos apuntadores. Posteriormente se sacaba del dispositivo de almacenamiento secundario el contenido del objeto, para verificar que lo último que se había escrito era en realidad lo que el objeto tenía guardado. A través de estas pruebas se dejó trabajando correctamente esta función. Esta es equivalente a la función que escribe sobre el contenido del objeto, y sus casos de prueba y los procedimientos empleados son similares a los ejecutados con la función que escribe sobre el contenido del objeto.

4.10 Modificar los permisos de operación de un Objeto Persistente.

Los permisos de operación con que trabaja un objeto sirven para restringir, o ampliar, el acceso que los diferentes grupos de usuarios del sistema tienen hacia un objeto. Los casos de prueba que se hicieron para esta función se dividían en tres partes: primero ponían accesible el objeto, luego emitían la función que modifica los permisos y finalmente guardaban el contenido del objeto para hacer permanentes los nuevos permisos. Las situaciones sobre las que se hizo énfasis fueron: que no cualquier usuario pudiera poner accesible un objeto; que un usuario pueda tener acceso restringido a un objeto (sólo lectura o sólo escritura); y que sólo el creador del objeto pueda modificar los permisos de operación. Finalmente, se verificó que la modificación de permisos se hicera permanente después de la operación de compromiso para el creador del objeto, y que no hubiera tal modificación si la llamada la ejecutaba cualquier otro usuario.

4.11 Revocar a un Objeto Persistente.

El objetivo de esta función es el de volver a su último estado consistente a un objeto que ha sido puesto a disposición de un usuario y que ha sido modificado. En cada caso de prueba se pedía uno o varios objetos, ya que estos estaban accesibles se ejecutaban algunas o todas las funciones de modificación que se han descrito en párrafos anteriores y finalmente se ejecutaba la operación de revocación. Después se hacía un vaciado del contenido del dispositivo de almacenamiento permanente, para verificar que los objetos que se habían estado modificando no habían cambiado. También se revisaron las estructuras de datos que usa la extensión para controlar los objetos y se verificó que ya no estaban en la memoria ni en dichas estructuras. Las pruebas se hicieron hasta que la revocación funcionó correctamente.

4.12 Ligar un Objeto Persistente.

Los objetos están distribuidos en el dispositivo de almacenamiento perdurable en varias áreas diferentes. Cuando el recolector de basura entra para liberar espacio, puede ser que sólo trabaje sobre una área, por lo que sólo eliminará los objetos que no tengan su identificador dado de alta en el objeto raíz de dicha área. Los casos de prueba para esta función consistieron en crear objetos y dar de alta algunos de los identificadores de esos objetos en el objeto raíz. Después se ejecutó la recolección de basura sobre el área y se verificó que los objetos que fueron dados de alta en la raíz aun existían, mientras que los que no se dieron de alta fueron eliminados por el recolector. Hay que hacer notar que en los casos en que un objeto que no está dado de alta en el objeto raíz de una área, pero que es apuntado

por un objeto que esta en otra área, el primero no se borra sino hasta que ya no es apuntado por nadie.

4.13 Desligar un Objeto Persistente.

Los objetos están distribuidos en el dispositivo de almacenamiento perdurable en varias áreas diferentes. Cuando el recolector de basura entra para liberar espacio, puede ser que sólo trabaje sobre una área, por lo que solo eliminará los objetos que no estén dados de alta en el objeto raíz del área que se está recolectando. Los casos de prueba para esta función fueron tres: desligar sólo algunos objetos del área, desligarlos todos o no desligar ninguno. Después de cada uno de los casos mencionados se hacia funcionar el recolector de basura y se revisaban, tanto el contenido del dispositivo de almacenamiento secundario, como las estructuras de datos, para verificar que realmente los objetos se habian eliminado. Hay que hacer notar que en los casos en que un objeto que está borrado en una área, es apuntado por un objeto que está en otra área, el primero no se borra sino hasta que ya no es apuntado por nadie. A través de estas pruebas se depuró tanto el funcionamiento de la operación de desligado (que fue lo más sencillo), como el funcionamiento del recolector de basura (lo más complicado).

4.14 Conveniencia del "hardware" especializado.

4.14.1 Manejo del archivo sombra.

El manejo del archivo sombra es muy parecido al de memoria virtual, por lo que se estima que su manejo desde el "hardware" es más eficaz que a través de la programación. Las operaciones básicas que se llevan a cabo para el archivo sombra son: i) pasar un objeto de disco a memoria, ii) modificar las estructuras de datos que reflejan el estado del archivo sombra según las operaciones efectuadas y iii) cuando se ejecuta el compromiso (commit) pasar a disco las páginas del objeto que se hayan modificado.

El movimiento de las páginas en las que están los objetos, de la memoria secundaria a la principal y viceversa, se haría más rápido si fuera ejecutado por el "hardware". La actualización de las estrucutras que controlan los objetos también se haría más veloz.

4.14.2 Recolección de basura.

Otro de los procesos que se podrían manejar desde el "hardware"

es la recolección de basura. Esto haría que independientemente del método de recolección de que se trate, éste se ejecutara más rápido. Para esto sería necesario contar con un procesador especializado y adecuarlo al manejo que se hace de los objetos.

4.15 Resultados de las pruebas medidos en tiempo.

En este inciso se muestran los resultados que arrojaron algunos programas de prueba. Estas pruebas se realizaron para determinar el comportamiento, en tiempos de ejecución, de las funciones de la extensión bajo condiciones bien determinadas.

Todos los tiempos se muestran en milisegundos. En cada uno de los cuadros de las tablas aparecen dos tiempos, el que está en la parte superior es el tiempo del procesador (CPU) que utilizó el programa para ejecutar sus instrucciones; el tiempo que esta en la parte inferior de cada cuadro es el que ocupó el proceso ejecutando llamadas al sistema UNIX.

Las pruebas que se ejecutaron son:

i) La creación de varios objetos de diferentes tamaños ejecutando un compromiso (commit) al final.

ii) La creación de varios objetos de diferentes tamaños ejecutando una revocación (roll back) al final.

iii) La consulta de varios objetos de diferentes tamaños, es decir, la lectura de ellos en una o varias ocasiones.

iv) La actualización del contenido de varios objetos de diferentes tamaños, es decir, la lectura y escritura del contenido de varios objetos en una o varias ocasiones, ejecutando un compromiso al final.

v) La actualización del contenido de varios objetos de diferentes tamaños, es decir, la lectura y escritura del contenido de varios objetos en una o varias ocasiones, ejecutando una revocación al final.

vi) El borrado de todos los objetos que se hayan creado y la recuperación del espacio que estos ocupaban a través del recolector de basura.

4.15.1 Creación de objetos con compromiso (commit).

Esta prueba consitió en la creación de un número determinado de objetos y en la ejecución del compromiso para cada uno de ellos. En la tabla 4.1 se muestran los tiempos que se obtuvieron como resultado de la ejecución de las pruebas.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	1150 866.6	1133.3 1066.6	1166.6 866.6	1166.6 1116.6
50	4016.6 900	3950 933.3	4066.6 1083.3	4483.3 966.6
100	7833.3 1000	7833.3 1066.6	8066.6 1016.6	8966.6 1316.6
200	16466.6 1100	16550 1100	17200 1333.3	19866.6 1483.3

Tabla 4.1 Tiempos de ejecución para la creación y compromiso de varios objetos.

4.15.2 Creación de objetos con revocación (roll back).

Esta prueba consitió en la creación de un número determinado de objetos y en la ejecución de la revocación para todos ellos. En la tabla 4.2 se muestran los tiempos que se obtuvieron como resultado de la ejecución de las pruebas.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	500 816.6	483.3 850	483.3 900	466.6 733.3
50	516.6 1033.3	533.3 850	500 783.3	516.6 883.3
100	600 950	650 800	633.3 900	650 1100
200	900 1000	933.3 800	950 1016.6	916.6 750

Tabla 4.2 Tiempos de ejecución para la creación y revocación de varios objetos.

4.15.3 Lectura de objetos (consulta).

Esta prueba consistió en solicitar varios objetos, ponerlos accesibles y ejecutar varias lecturas sobre ellos. Las pruebas se realizaron ejecutando una lectura, cinco lecturas y quince lecturas sobre un mismo objeto. En las tablas 4.3, 4.4 y 4.5 se pueden ver los resultados de la ejecución de las pruebas.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	633.3 883.3	616.6 983.3	583.3 916.6	750 750
50	1483.3 950	1400 883.3	1616.6 850	1966.6 1333.3
100	3033.3 1016.6	2900 933.3	3133.3 833.3	3650 1233.3
200	7383.3 983.3	7433.3 1000	7516.6 1050	8550 1233.3

Tabla 4.3 Tiempos de ejecución para poner accesibles los objetos y para ejecutar una lectura sobre ellos.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	850 816.6	866.6 966.6	883.3 933.3	1050 983.3
50	2716.6 800	2616.6 900	2916.6 850	3616.6 1000
100	5550 983.3	5300 983.3	5800 866.6	7266.6 1016.6
200	12466.6 1016.6	12050 916.6	13000 916.6	15800 1350

Tabla 4.4 Tiempos de ejecución para poner accesibles los objetos y para ejecutar cinco lecturas sobre ellos.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	1383.3 850	1483.3 983.3	1583.3 1000	1966.6 866.6
50	5850 866.6	5633.3 800	6266.6 1016.6	8183.3 916.6
100	11983.3 883.3	11233.3 933.3	12766.6 883.3	16383.3 1133.3
200	25300 900	23850 900	27033.3 1116.6	34716.6 2950

Tabla 4.5 Tiempos de ejecución para poner accesibles los objetos y para ejecutar quince lecturas sobre ellos.

4.15.4 Actualización de objetos con compromiso (commit).

Esta prueba consistió en poner accesibles varios objetos, efectuar la lectura y escritura de su contenido varias veces (una, cinco y quince veces) y después ejecutar el compromiso. En las tablas 4.6, 4.7 y 4.8 se pueden ver los tiempos que resultaron de la ejecución de las pruebas.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	1100 983.3	1183.3 900	1250 1033.3	1450 1116.6
50	4116.6 1166.6	4116.6 1150	4716.6 1083.3	5783.3 1250
100	8250 1266.6	8483.3 1316.6	9416.6 1266.6	12183.3 1516.6
200	17900 1633.3	18366.6 1783.3	21216.6 1916.6	27050 2200

Tabla 4.6 Tiempos de ejecución para poner accesibles los objetos, leerlos y escribirlos una vez y ejecutar el compromiso.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	1633.3 1050	1850 1066.6	2116.6 1016.6	2866.6 1033.3
50	6866.6 1016.6	7400 1233.3	9333.3 1466.6	12916.6 1350
100	14250 1666.6	14883.3 1466.6	18300 1383.3	26116.6 1500
200	29133.3 1533.3	31700 1800	39050 2083.3	54550 3000

Tabla 4.7 Tiempos de ejecución para poner accesibles los objetos, leerlos y escribirlos cinco veces y ejecutar el compromiso.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	2883.3 1033.3	3450 950	4400 883.3	6350 1016.6
50	13700 1133.3	15800 1033.3	20316.6 1250	30183.3 1200
100	27966.6 1383.3	31550 1300	41100 1850	61083.3 1516.6
200	57233.3 1600	65116.6 2266.6	86550 3233.3	124450 2526.6

Tabla 4.8 Tiempos de ejecución para poner accesibles los objetos, leerlos y escribirlos quince veces y ejecutar el compromiso.

4.15.5 Actualización de objetos con revocación (commit).

Esta prueba consistió en poner accesibles varios objetos, efectuar la lectura y escritura de su contenido varias veces (una, cinco y quince veces) y después hacer la revocación de las modificaciones. En las tablas 4.9, 4.10 y 4.11 se pueden ver los tiempos que resultaron de la ejecución de las pruebas.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	733.3	783.3	833.3	916.6
	816.6	933.3	833.3	950
50	2216.6	2183.3	2466.6	3166.6
	733.3	833.3	900	900
100	4383.3	4333.3	4966.6	6400
	983.3	1122.2	916.6	1366.6
200	9983.3	9933.3	11100	14150
	950	1000	1383.3	1516.6

Tabla 4.9 Tiempos de ejecución para poner accesibles los objetos, leerlos y escribirlos una vez y ejecutar la revocación.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	1250	1366.6	1750	2350
	833.3	900	916.6	916.6
50	4933.3	5516.6	6850	10066.6
	866.6	783.3	933.3	766.6
100	9966.6	10866.6	13750	20033.3
	966.6	900	950	1183.3
200	21233.3	23016.6	28966.6	41350
	933.3	1116.6	1883.3	1233.3

Tabla 4.10 Tiempos de ejecución para poner accesibles los objetos, leerlos y escribirlos cinco veces y ejecutar la revocación.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	2533.3 833.3	3066.6 916.6	3900 866.6	5716.6 1033.3
50	11950 1083.3	13600 883.3	18116.6 1233.3	27133.3 966.6
100	24083.3 1083.3	27533 1366.6	35866.6 800	54500 1233.3
200	49966.6 1583.3	55683.3 916.6	73100 1650	109666.6 1450

Tabla 4.11 Tiempos de ejecución para poner accesibles los objetos, leerlos y escribirlos quince veces y ejecutar la revocación.

4.15.6 Borrado de objetos y recolección de basura.

Esta prueba consistió en el borrado de todos los objetos que se habían creado y en la recuperación de ese espacio por medio del recolector de basura. En la tabla 4.12 se pueden observar los tiempos que resultaron de la ejecución de esta prueba.

Tamaño en "bytes".

Número de Objetos	10	50	100	200
10	1316.6 1133.3	1283.3 883.3	1266.6 950	1316.6 983.3
50	2983.3 1083.3	2666.6 983.3	2566.6 1000	2316.6 1216.6
100	5016.6 950	4416.6 1000	4216.6 1216.6	3816.6 1516.6
200	9133.3 1616.6	7800 1216.6	7500 1450	7100 1816.6

Tabla 4.12 Tiempos de ejecución para el borrado y la recuperación del espacio que ocupaban los objetos.

Conclusiones

En esta tesis se presenta el diseño y el desarrollo de una extensión a un sistema operativo para manejar Objetos Persistentes. Los objetos se construyeron como una herramienta de programación. Los usuarios pueden conservar su información en estos objetos. A través de un conjunto de funciones se permite realizar operaciones bien definidas sobre dichos objetos.

Los tiempos invertidos en el desarrollo del trabajo fueron los siguientes: el análisis y el diseño del sistema tomaron de 6 a 7 meses trabajando medio tiempo; el desarrollo de la progrmación del sistema tomo 3 meses de tiempo completo; y el diseño y la ejecución de las pruebas tomaron 2 meses de tiempo completo. El sistema, incluyendo los programas de prueba, consta de aproximadamente 3 mil lineas de código (sin contar los comentarios), escritas en el lenguaje de programación "C".

En seguida se lista una serie de problemas que se tuvieron que reslover para hacer la implantación de los objetos y las consecuencias que las soluciones acarrearon.

1. Uno de los objetivos fundamentales de la implantación de los objetos es que estos sean un mecanismo sencillo y uniforme para almacenar información.

2. Sujetarse a una arquitectura de computadora del tipo de von Neumann. En este caso la definición lógica y el manejo de los objetos no tienen nada que ver con la arquitectura de la computadora. Si la arquitectura facilitara el manejo de objetos, estos se implantarían más fácil y eficazmente.

3. Se debe distinguir entre dos tipos de información en el objeto, una es la que el usuario deposita en él y la otra es la que el sistema necesita conocer para manejarlo. A la del usuario se le conoce como contenido y a la del sistema como encabezado, y se manejan separadas. Al dividir los objetos en contenido y encabezado, las solicitudes para revisar un dato se hacen más costosas y más lentas. Esto se debe a que hay que leer primero la información del encabezado para saber si el solicitante tiene derecho de revisar el dato, si está condición se cumple se ejecuta otra lectura, pero ahora sobre el contenido del objeto para obtener el dato.

4. La división básica del almacenamiento perdurable en bloques de 512 "bytes", se hizo para evitar la fragmentación durante el manejo de los contenidos de los objetos, aunque el problema no se evite del todo por lo menos se reduce. Esto ocasiona que cuando un objeto ocupa varios bloques, la busqueda secuencial de un dato se haga más lenta, ya que hay que ir buscándolo en cada bloque, es decir, se va leyendo un bloque tras otro hasta que se encuentra el dato buscado.

5. Para que los objetos sean persistentes se deben almacenar de manera perdurable, esto se consigue guardándolos en disco. Sin embargo, cuando un proceso desèa trabajar sobre el objeto lo mejor es tenerlo en memoria principal. Esto ocasiona que los objetos que están siendo usados por los procesos se manejen en dos niveles de memoria, esto es, en disco y en memoria principal. El costo de tener esos objetos en ambos niveles de almacenamiento es mayor que si sólo se tuvieran en disco. La ventaja que se logra es que el acceso a esos objetos es más rápido.

   Una de las ventajas que proporcionan los objetos, es que esta distinción entre los niveles de memoria es totalmente transparente para el usuario.

6. Otra de las ventajas que proporcionan los objetos es que siempre se conserva una copia válida del contenido de cada uno de ellos. Esto se logra gracias a que se ha implantado el método del archivo sombra para manejarlos. Además, esta operación de respaldo de los objetos se maneja de manera transparente para el usuario.

   Una de las consecuencias de esto es que el almacenamiento de cada uno de los objetos toma más tiempo y necesita más memoria (aunque la memoria excedente después se libere).

7. Cada una de las funciones que están definidas en la extensión al sistema operativo necesita manejar ciertas estructuras de datos que le ayuden a trabajar sobre los objetos. El espacio de memoria y el tiempo de proceso que se gasta en mantenerlas son un gasto extra para los programas que usan los objetos. Este gasto es inevitable y es parte del precio que se tiene que pagar por usar los objetos.

8. Uno de los objetivos de los objetos es que sirvan para compartir información entre varios usuarios. Pero también hay que considerar el hecho de restringir el acceso a esa información a los usuarios no autorizados. Para esto se implantó un sistema de permisos de acceso del mismo tipo que se definen en el sistema operativo huesped (UNIX). Esta es una de las ventajas que se proporciona al trabajar con objetos.

9. Un objeto contiene apuntadores a otros objetos. Mediante el uso de estas referencias se pueden formar diferentes tipos de estructuras, todo bajo el control del usuario.

10. Para eliminar los objetos que ya no son usados por nadie en el sistema y para recuperar el espacio que ocupan se cuenta con un proceso de recolección de basura. Como este proceso hace una revisión completa del área de almacenamiento perdurable consume mucho tiempo de proceso. Además, mientras él se este ejecutando ningún proceso de usuario puede tener acceso a los objetos. Para reducir estos porblemas lo que se hizo fue implantar un método de recolección de basura que actúa sobre áreas reducidas del dispositivo de almacenamiento perdurable. Es decir, el contenido de dicho dispositivo se dividió en áreas y el recolector de basura puede funcionar sobre cada una de ellas por separado. De esta manera se ahorra tiempo de proceso, ya que el recolector trabaja sólo sobre un fragmento del espacio total de almacenamiento, y además los usuarios pueden disponer más pronto de sus objetos.

    Sin embargo, el hecho de dividir el dispositivo de almacenamiento en áreas, obliga a crear estructuras de datos para manejar los objetos que hay en cada área, el espacio disponible en ella, el cambio de área de un objeto, etcétera. Por lo tanto, al tiempo que se ahorra al recolectar sólo una área, se le debe restar el tiempo gastado para mantener actualizadas las estructuras de datos que controlan las áreas. Entre mayor sea el número de áreas en que se divide el almacenamiento perdurable, más estructuras de datos se tendrán que manejar y más lento será el proceso que las mantiene.

11. En la actualidad el manejo de los objetos persistentes con código puro se restringe a evitar que lo accesen usuarios no autorizados y a que no lo usen para leer ni para escribir los usuarios autorizados. No se da la facilidad de convertirlo en proceso y ejecutarlo.

12. El manejo de los apuntadores en una zona especial se debe a que de esa manera el sistema apoya las referencias entre los objetos. Este apoyo consiste, básicamente, en que al momento de hacer la recolección de basura se revisa el área de apuntadores y todos los objetos que estén incluidos ahí no se eliminan aunque su propietario ya los haya desligado. Si los identificadores sólo se tienen en el área de contenido se puede perder el objeto apuntado antes de eliminar el apuntador.

13. Actualmente cuando un proceso trata de hacer acceso a un objeto que ya está accesible y falla por incompatibilidad de candado o de respaldo, se elimina. Esto se debe a que se consideran como problemas separados el manejo de los objetos y el manejo de los candados de concurrencia. Para propósitos de prueba se construyó un procedimiento sencillo de verificación de candados. Si a partir de esta extensión se construye un manejador de objetos, a éste se le debe agregar un mecanismo completo de manejo de candados de concurrencia, el mecanismo debe considerar las diferentes compatibilidades

de candados, la suspensión o el bloqueo de los procesos y su reinicialización posterior, entre otras cosas. En la extensión ya está considerada la existencia de candados, por lo que sólo hay que implantar el mecanismo sobre ella.

14. Aunque la extensión se ha probado de la manera más amplia posible, en este momento no se puede dar una garantía total y absoluta sobre su funcionamiento. La prueba definitiva sería el uso que le den las personas que trabajen sobre ella y que señalen los detalles que funcionen inadecuadamente.

15. A través del funcionamiento del recolector de basura se ha visto que la distribución de los objetos en el dispositivo de almacenamiento perdurable siempre trabaja sobre la primera área, en donde el factor de ocupación es siempre más alto que en las demás áreas. (Factor de ocupación es el procentaje del área que se encuentra ocupado con objetos). Esto encamina a pensar en una modificación al algoritmo de almacenamiento de los objetos cuando se hace el compromiso, para mejorar el funcionamiento del recolector de basura.

16. Finalmente, los usuarios a través de su trabajo con las funciones de la extensión podrán ir dando sugerencias sobre procedimientos no incluidos o modificaciones a las funciones actuales para obtener un mejor comportamiento.

Bibliografía

[ARNBORG 72] Arnborg, S., "Storage Administration in a Virtual Memory SIMULA System", Bit, número 2, volumen 12, abril de 1972, páginas 125-141.

[ATKINSON 82] Atkinson, M., Chisholm, K. y Cockshott, P., "PS-Algol: An Algol with a Persistent Heap", SIGPLAN Notices of the ACM, número 7, volumen 17, julio de 1982, páginas 24-31.

[ATKINSON 83-1] Atkinson, M., Chisholm, K., Cockshott, P. y Marshall, R., "Algorithms for a Persistent Heap", Software - Practice and Experience, número 3, volumen 13, marzo de 1983, páginas 259-271.

[ATKINSON 83-2] Atkinson, M., Chisholm, K. y Cockshott, P., "CMS - A Chunk Management System", Software - Practice and Experience, número 3, volumen 13, marzo de 1983, páginas 273-285.

[BAECKER 72] Baecker, H. D., "Garbage Collection for Virtual Memory Computer Systems", Communications of the ACM, número 11, volumen 15, noviembre de 1972, páginas 981-986.

[BISHOP 77] Bishop, Peter B., "Computer Systems with a Very Large Address Space and Garbage Collection", MIT, Ph. D. Tesis, Computer Science, mayo de 1977.

[BUCHMANN 86] Buchmann, A., Carrera, R. y Vazquez, M. A., "A Generalized Constraint and Exception Handler for an Object - Oriented CAD - DBMS", IIMAS, UNAM, 1986.

[CARDENAS 86] Cardenas, S., "Una Máquina Virtual para TM.", Tesis de Maestría, UNAM, 1986.

[CICHELLI 80] Cichelli, R. J., "Minimal Perfect Hash Functions Made Simple", Communications of the ACM, número 1, volumen 23, enero de 1980, páginas 17-19.

[COHEN 81] Cohen, J., "Garbage Collection of Linked Data Structures", Computing Surveys, número 3, volumen 13, septiembre de 1981, páginas 341-367.

[COHEN 83] Cohen, J. y Nicolau, A., "Comparison of Compacting Algorithms for Garbage Collection", ACM Transactions on Programming Languages and Systems, número 4, volumen 5, octubre de 1983, páginas 532-553.

[DEUTSCH 76] Deutsch, L. P. y Bobrow, D. G., "An Efficient, Incremental, Automatic Garbage Collector", Communications of the ACM, número 9, volumen 19, septiembre de 1976, páginas 522-526.

[DIEL 84] Diel, H., Kreissig, G., Lenz, N., Scheible, M. y Schoener, B., "Data Management Facilities of an Operating System Kernel", Proceedings of annual meeting SIGMOD ACM, SIGMOD Record número 2, volumen 14, junio de 1984, páginas 58-69.

[FAGIN 79] Fagin, R., Nievergelt, J., Pippenger, N. y Strong, H. R., "Extendible Hashing - A Fast Access Method for Dynamic Files", ACM Transactions on Database Systems, número 3, volumen 4, septiembre de 1979, páginas 315-344.

[GERZSO 83] Gerzso, M., "Report on the Language TM: its design and definition.", IIMAS, UNAM, 1983.

[GOLDBERG 83] Goldberg, A. y Robson, D., "Smalltalk-80: The Language and Its Implementation.", Addison-Wesley, EUA, 1983.

[GRAY 77] Gray, J., "Notes on Database Operating Systems", Operating Systems: An Advanced Course, Springer-Verlag, 1977, páginas 394-481.

[JIMENEZ 86] Jiménez, F., "Diseño e Implantación de un Sistema Orientado a Objetos ", Tesis de Maestría, UNAM, 1986.

[LARSON 78] Larson, Per-Ake., "Dynamic Hashing", Bit, número 2, volumen 18, abril de 1978, páginas 184-201.

[LEVY 84] Levy, Harry. M, "Capability-Based Computer Systems", Digital Press, EUA, 1984.

[LITWIN 78] Litwin, W., "Virtual Hashing: A Dynamically Changing Hashing", Proceedings of Fourth International Conference on Very Large Data Bases, West Berlin, Germany, IEEE, septiembre de 1978, páginas 517-523.

[LORIE 77] Lorie, R. A., "Physical Integrity in a Large Segmented Database", ACM Transactions on Database Systems, número 1, volumen 2, marzo de 1977, páginas 91-104.

[MAURER 75] Maurer, W. D. y Lewis, T. G., "Hash Table Methods", Computing Surveys, número 1, volumen 7, marzo de 1975, páginas 5-19.

[MORRIS 78] Morris, F. L., "A Time and Space Efficient Garbage Compactation Algorithm", Communications of the ACM, número 8, volumen 21, agosto de 1978, páginas 662-665.

[MYERS 78] "Advances in Computer Architecture", Myers, G. J., John Wiley and Sons, EUA, 1978.

[QUITTNER 83] Quittner, Pàl., "Efficient Combination of the Index Tables and Hashing", Software - Practice and Experience, número 6, volumen 13, junio de 1983, pàginas 471-478.

[STONEBRAKER 81] Stonebraker, M., "Operating Systems Support for Database Management", Communications of the ACM, número 7, volumen 24, julio de 1981, pàginas 412-418.

[THORELLI 72] Thorelli, Lars-Erik, "Marking Algorithms", Bit, número 4, volumen 12, octubre de 1972, pàginas 555-568.

[VALDURIEZ 84] Valduriez, P. y Viemont, Y., "A Multikey Hashing Scheme Using Predicate Trees", Proceedings of annual meeting SIGMOD ACM, SIGMOD Record número 2, volumen 14, junio de 1984, pàginas 107-113.