Universidad Nacional Autónoma de México

Facultad de Ingeniería

"SUBSISTEMA DE VALIDACIÓN Y PROCESO DE FORMAS ÚNICAS DEL SISTEMA DE NÓMINA"

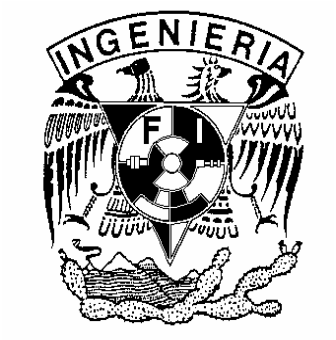
TESIS PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:

MARIBEL CALDERÓN GAMINO

DIRECTOR DE TESIS:
ING. ALBERTO ALCÁNTARA MELÉNDEZ



CIUDAD UNIVERSITARIA, MÉXICO D. F. JUNIO DEL 2004

TABLA DE CONTENIDO

TABLA DE FIGURAS

AGRADECIMIENTOS

*A **mis padres**, que siempre me apoyaron incondicionalmente para que pueda cumplir con las metas que me he propuesto y ser un ser humano feliz y completo.*

*A **mi esposo**, que me alentó para concluir una de las etapas más importantes de mi vida y que me apoya para continuar adelante, siempre adelante.*

*A **mis jefes** que me dieron el apoyo necesario para terminar mis estudios y la oportunidad de demostrar mis aptitudes.*

*A **mis profesores**, quienes fueron formando a la profesionista y compartieron sus conocimientos conmigo para que alcanzara la meta.*

*A la **Universidad Nacional Autónoma de México**, que me proporcionó la oportunidad de trabajar y estudiar, dandome todo lo necesario para cumplir con un objetivo primordial en mi vida.*

1. INTRODUCCIÓN.

Este subsistema valida y procesa los movimientos liberados por el módulo de Captura de Forma Única. Las Formas Únicas sirven para formalizar los movimientos de alta, licencia, reanudación de labores, promoción y baja del empleado.

Los movimientos de Forma Única se validan para confirmar su procedencia, con base al presupuesto y en combinación con otros nombramientos del empleado. Durante el proceso de las Formas Únicas se actualizan los nombramientos, la historia de movimientos y datos para el cálculo de la quincena.

Los movimientos de Forma Única llegan con retraso para su proceso en nómina; por lo tanto, es necesario calcular sueldos e impuestos retroactivos, así como los pagos y descuentos acumulados durante el periodo retroactivo. También se actualiza la antigüedad del empleado.

El subsistema está constituido por varios módulos, mismos que a continuación se explican en forma general.

Módulo de validación de Forma Única. En esta parte se verifica que la información asentada en la Forma Única sea válida para realizar el movimiento. Son varias condiciones que deben cumplir; por ejemplo, si el movimiento es una baja de nombramiento, la categoría, número de plaza, código programático y horas, deben coincidir con los datos del nombramiento que tiene el empleado; en el caso de las altas, verificamos que la plaza exista en el catálogo y que no esté ocupada por otro empleado, que exista el código programático en el catálogo, que el sueldo sea correcto, etc.

Módulo de verificación. Aquí se verifica la consistencia de los nombramientos finales del empleado, sobre todo los sueldos de zonas geográficas y ajustes de asignación de los funcionarios.

Módulo de movimientos continuos. El objetivo principal es obtener un solo folio de movimientos de horas que sean continuos unos con otros; es decir, que la fecha de inicio de una alta sea del día siguiente en que se da de baja. Esto sucede porque a los profesores se les elabora Forma Única cada semestre, y generalmente la continuación del nombramiento llega en la misma quincena en que se da de baja.

Módulo de incompatibles. Se encarga de identificar los movimientos incompatibles, busca las diferentes causas que hacen que un movimiento sea incompatible (Formas Únicas que son o afectan a personal funcionario, únicos pagos de personal funcionario, movimientos de

personal por honorarios, ayudantes de profesor con total de horas mayor a doce, profesores con total de horas mayor a cuarenta, nombramientos con total de horas mayor a cuarenta y ocho, etc.); así como, la emisión del reporte de incompatibles y el de movimientos de altas y bajas, estos reportes se mandan al área encargada de revisarlos.

Módulo de banco de horas. Este módulo realiza la conversión del archivo de texto que proporciona quincenalmente la Dirección General de Programación y Presupuestación, con el total de horas autorizadas por dependencia, partida y categoría, suma y actualiza las horas ejercidas en la quincena anterior, verifica la suficiencia presupuestal de los movimientos de alta del personal de asignatura, actualiza las horas ejercidas y disponibles en la tabla de banco de horas de acuerdo al tipo de movimiento.

Módulo de altas. En esta parte se crea lo que vamos a llamar tarjetas retroactivas, lo cual significa hacer un registro por cada sueldo diferente que exista desde que el movimiento inicie hasta que termine, debido a los aumentos que se dan anualmente. Además, se calculan algunos datos generales que nos van a servir para el cálculo de pagos y descuentos retroactivos.

Módulo de pagos y descuentos retroactivos. Aquí se toman los registros creados en el módulo anterior, para calcular todos los conceptos retroactivos que se le deben pagar o descontar a los empleados que tienen movimientos.

Módulo de afectación a nombramientos. Se van a actualizar los nombramientos de los empleados; así como, los datos del empleado que se tengan que modificar, debido a los movimientos.

2. PROBLEMÁTICA DEL PROCESO DE FORMA ÚNICA.

La problemática que se presenta es el manejo de la Forma Única, es decir, el proceso inicia cuando ya están listas las Formas Únicas que se van a procesar en la quincena, se deben tomar y validar para saber si la información que contienen es correcta. En esta parte se realiza mucha revisión manual, la cual queremos eliminar o por lo menos reducir; además, en algunos casos es necesario conocer cuál será la situación del empleado después de aplicar ciertos movimientos.

Posteriormente se emiten reportes, que se mandan a diversas áreas, las cuales también validan la información, para obtener estos reportes se requiere conocer como va a quedar el empleado después de aplicar los movimientos, pero sin afectar realmente sus nombramientos.

Existen movimientos que son especiales, para los cuales se hacen revisiones adicionales, e incluso cambios manuales para que puedan entrar al proceso, los cuales ocasionalmente se hacen directamente sobre las tablas entre un paso y otro.

Cuando se realizan los cálculos retroactivos se debe consultar el tabulador varias veces para una sola Forma Única, dependiendo de la retroactividad de los movimientos, el proceso se vuelve muy tedioso y por lo tanto muy lento, ya que no sólo necesitamos el tabulador para las diferencias de sueldos, sino también para todos los conceptos retroactivos; además, se hace la validación de sueldos, cálculos de categorías especiales, simulación de la afectación de los nombramientos, etc.

El tabulador se consulta durante todo el proceso y es uno de los principales problemas, debido a que son varios accesos a la tabla para consultar un mismo sueldo durante diferentes etapas del proceso, incluso dentro de un mismo programa.

Asimismo, otro problema se presenta al calcular los conceptos retroactivos, ya que se requieren ciertos datos que se calculan empleado por empleado, estos datos se utilizan en diferentes partes del proceso; por lo tanto, cada vez que requerimos el dato, se calcula, lo cual retrasa el proceso considerablemente.

Es conveniente aclarar, que estamos partiendo de un subsistema, que ya estaba programado, e incluso se encontraba funcionando para la Nómina de

la UNAM; sin embargo, debido a que presentaba algunos problemas, se tomó la decisión de rediseñarlo y reprogramarlo.

El código de programación no es muy claro y se repite en varios programas, lo cual ocasiona que el sistema sea muy difícil de mantener porque al momento de hacer un cambio se debe investigar en todos los programas si existe un código semejante al que acabamos de corregir. Esta programación se complicó debido a que no se tenían contempladas todas las condiciones que se podrían presentar, y se fue agregando código para corregir los problemas.

No se hizo un análisis adecuado del proceso a seguir, al ir agregando correcciones, el código se hizo muy denso, difícil de entender y por lo tanto de mantener.

Al desarrollar el nuevo análisis y diseño, la experiencia obtenida de los problemas que se presentaron durante la producción normal de las quincenas, nos ayudó mucho, porque ya se tenía el contexto real de trabajo y los problemas a los que se enfrenta el usuario diariamente; además, se tenía conocimiento de la mayoría de las condiciones que se presentan.

3. METODOLOGÍA.

La ingeniería del software es todo lo referente a producir lo que el cliente desea, dentro del tiempo y costo requeridos. Un producto de calidad necesita una combinación de habilidades técnicas y organizacionales. La ingeniería del software ha crecido tanto que hoy en día es una verdadera disciplina, derivada de una investigación seria, un estudio minucioso y un debate multitudinario.

Hay numerosos modelos del proceso de desarrollo de software disponibles en la actualidad. Éstos varían en naturaleza desde descriptiva (aquellos que tan sólo describen lo que existe) hasta prescriptivos (los que establecen qué pasos deben tomarse).

Para el desarrollo de este módulo se siguió un "ciclo de vida del software", que se refiere a los pasos a seguir para el desarrollo de un sistema. A continuación se va a tratar de describir de manera sencilla, cuáles son estos pasos:

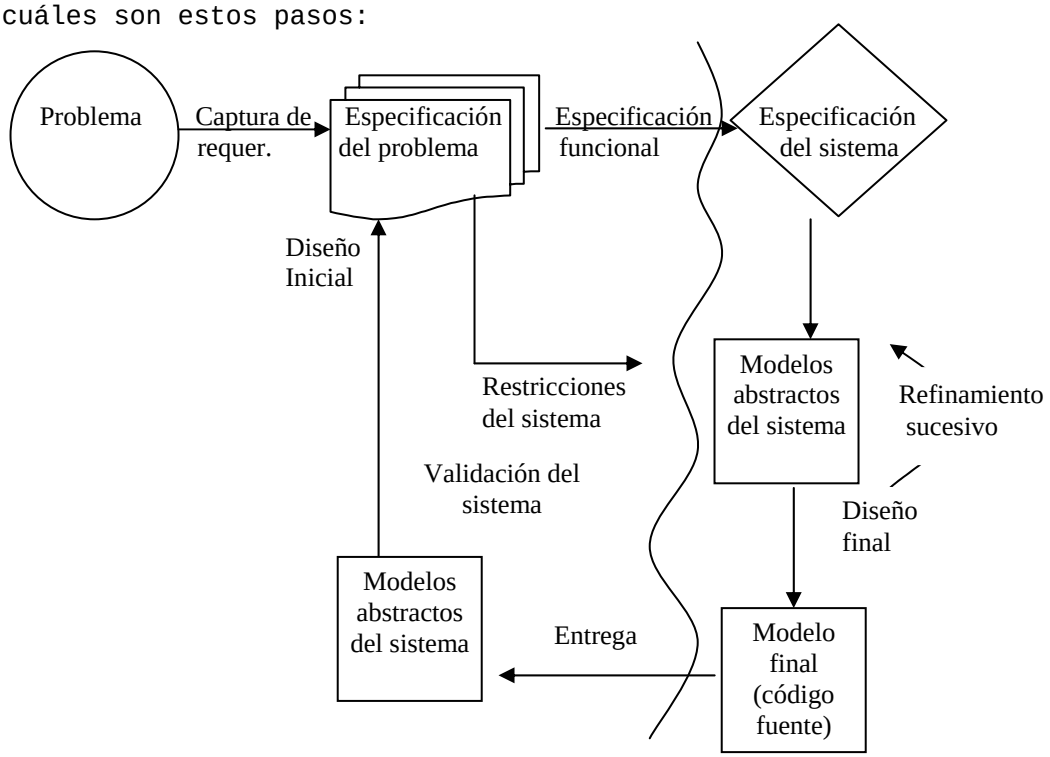


Figura 3.1 Panorámica de las actividades en el desarrollo de sistemas.

En la figura 3.1 los aspectos orientados al problema del diseño (en los que el mundo real está involucrado, en el lado izquierdo) se distinguen de los aspectos orientados al diseño (en los que modelar abstracciones es lo más relevante, en el lado derecho).

Es así como se refleja de una manera sencilla, lo que podría ser el "ciclo de vida de un sistema", para nuestro caso los pasos a seguir dentro del ciclo serían los siguientes:

Análisis de requerimientos del sistema.

Diseño del sistema.

Codificación.

Pruebas.

Mantenimiento.

Éstos representan un ciclo de vida clásico. En el presente trabajo se explicarán y desarrollarán para el módulo de Forma Única cada uno de estos pasos.

Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software. La ingeniería y el análisis del sistema abarcan los requisitos globales a nivel del sistema con una pequeña cantidad de análisis y de diseño a un nivel superior.

El proceso de recopilación de los requisitos se centra e intensifica especialmente para el software, el ingeniero de software ("analista") debe comprender el ámbito de la información del software, así como la función, el rendimiento y las interfaces que se requieren. Los requisitos tanto del sistema como del software, se documentan y se revisan con el cliente.

Ahora bien, con el crecimiento acelerado de la computación, también se han desarrollado diferentes métodos de análisis, siendo quizás el más reciente el análisis y modelado orientado a objetos; sin embargo, para este sistema se trabajó con el análisis estructurado, ya que las herramientas proporcionadas no contenían software orientado a objetos.

3. 1 ANÁLISIS ESTRUCTURADO.

El análisis estructurado es una actividad de construcción de modelos mediante una notación que es única del método de análisis estructurado, se van a crear modelos que reflejen el flujo y el contenido de la información (datos y control). Se divide el sistema funcionalmente y, según los distintos comportamientos, se establece la esencia de lo que se debe construir. No es un método sencillo que se aplique siempre de la misma forma.

El análisis estructurado no fue introducido en un solo artículo o libro clave que incluyera un tratamiento completo del tema. Los primeros trabajos sobre modelos de análisis aparecieron a finales de los 60 y principios de los 70, pero la primera aparición del enfoque de análisis estructurado apareció como complemento de otro tema muy importante como lo fue el "diseño estructurado". Los investigadores necesitaban una notación gráfica para representar los datos y los procesos que los transforman. Esos procesos quedarían finalmente establecidos en una arquitectura de diseño.

Tom DeMarco presentó y denominó los símbolos gráficos clave que permitirían a un analista crear modelos de flujo de información, sugirió heurísticas para utilizar esos símbolos; el uso de un diccionario de datos y narrativas de procesamiento, así como complementos a los modelos de flujo de información y presentó numerosos ejemplos que mostraban el uso del nuevo método.

Posteriormente otros investigadores propusieron variaciones del análisis estructurado, pero en todos los casos el método se centraba en aplicaciones de sistemas de información y no proporcionaba una notación adecuada para los aspectos de control y comportamiento de los problemas de ingeniería de tiempo real.

Más adelante las "ampliaciones" para tiempo real fueron introducidas por Ward y Mellor y, más tarde, por Hatley y Pirghai, con lo que se consiguió un método de análisis más robusto que podía ser aplicado de forma efectiva a problemas de ingeniería. Actualmente se intenta desarrollar una notación consistente y se han publicado tratamientos modernos que nos permitan el uso de herramientas CASE (Ingeniería del software asistida por computadora).

3.1.1 NOTACIÓN BÁSICA.

La información se transforma conforme va fluyendo en un sistema basado en computadora. Un sistema acepta una gran variedad de formas, aplica los elementos de hardware, software y humanos para transformar la entrada en salida y produce salida en una gran variedad de formas. La entrada puede ser una señal de control transmitida por un controlador, una serie de números escritos en el teclado por un operador humano, un paquete de información transmitido por un enlace de una red o un archivo voluminoso de datos que se encuentra en un dispositivo de almacenamiento secundario. La transformación puede ser, desde una sencilla comparación lógica, hasta un complejo algoritmo numérico o un mecanismo de reglas de un sistema experto. La salida puede ser el encendido de un diodo de emisión de luz (LED) o un informe de 200 páginas. Esto quiere decir que se puede crear un modelo de flujo para cualquier sistema de computadora, sin importar el tamaño o la complejidad.

El análisis estructurado es una técnica de modelado del flujo y del contenido de la información, vamos a transformar la información. Y este proceso se representa con una burbuja. Las entidades externas se representan por medio de cuadros, ahí se originan una o más entradas, las cuales se dibujan como flechas etiquetadas. A continuación se presenta la figura 3.2 donde se ilustra cómo se modela la información:

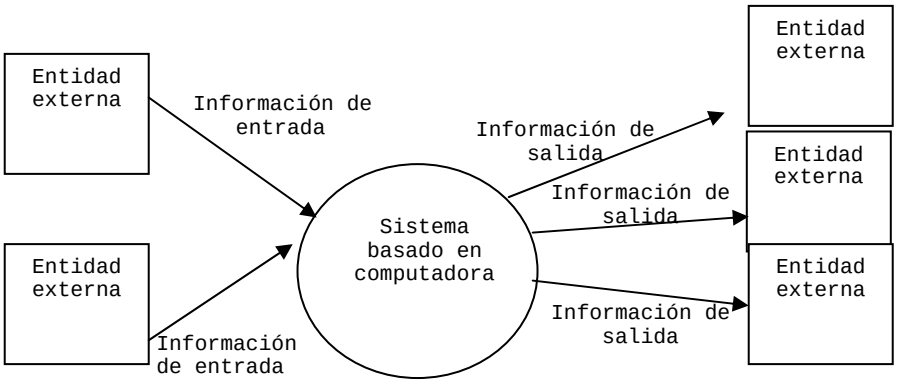


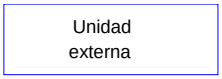
Figura 3.2 Modelo de flujo de información.

El diagrama de flujo de datos (DFD) es una técnica gráfica que representa el flujo de la información y las transformaciones que se aplican a los datos al moverse desde la entrada hasta la salida.

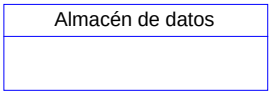
El DFD se usa para modelar un sistema a cualquier nivel de abstracción. Los diagramas de flujo de datos pueden ser refinados en niveles que representen un mayor flujo de información y un mayor detalle funcional. Un DFD de nivel 0 también es denominado modelo fundamental del sistema o modelo del contexto y representa un elemento muy general del sistema.

A partir el DFD de nivel 0, para mostrar más detalles, se va refinando hasta llegar al proceso final y más elemental; es decir, el DFD 0, es una burbuja que contiene el proceso general con sus respectivas entradas y salidas, después el proceso principal se subdivide en otros procesos, y conforme vamos aumentando el nivel del DFD, también aumentan los procesos dentro de estos, cada uno de estos procesos es una subfunción del sistema general. Por ejemplo, un DFD de nivel 1 puede contener cinco o seis burbujas con flechas interconectándolas, y después cada uno de estos subprocesos tienen más procesos internos.

A continuación presentaré la notación básica que se va a usar para crear un DFD en este sistema:

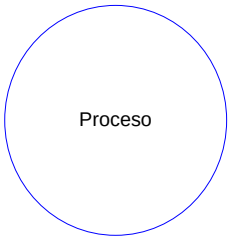


Este rectángulo representa una entidad externa, es un elemento del sistema, como hardware, una persona, etc.



Aquí se representa un depósito de datos que se guardan para ser usados por uno o más procesos; puede ser tan sencillo como una cola, o tan sofisticado como una base de datos relacional.

La flecha es un elemento de datos o una colección de elementos de datos; la cabeza de la flecha indica la dirección del flujo de datos.



El círculo representa un proceso. Es un transformador de información que reside dentro de los límites del sistema a ser modelado.

La sencillez de la notación DFD es una de las razones por las que las técnicas de análisis estructurado son ampliamente utilizadas.

Es importante señalar que el diagrama no proporciona ninguna indicación explícita de la secuencia de procesamiento. El procedimiento o la secuencia pueden estar implícitamente en el diagrama, pero la representación procedimental explícita generalmente queda pospuesta hasta el diseño del sistema.

3.1.2 LA MECÁNICA DEL ANÁLISIS ESTRUCTURADO.

El DFD permite desarrollar modelos del ámbito de información y del funcional al mismo tiempo.

Unas pocas directrices nos pueden ayudar durante la derivación de un diagrama de flujo de datos:

- El diagrama de flujo de datos de nivel 0 debe reflejar el sistema como un solo proceso.
- Se deben anotar cuidadosamente la entrada y la salida principales.
- El refinamiento debe comenzar aislando los procesos, los elementos de datos y los almacenes de datos que sean candidatos a ser representados en el siguiente nivel.
- Todas las flechas y las burbujas deben ser rotuladas con nombres significativos.
- Entre sucesivos niveles se debe mantener la continuidad del flujo de información.
- Se deben refinar las burbujas de una en una.

A veces se llega a complicar en exceso el diagrama de flujo de datos. Esto sucede porque el analista intenta reflejar demasiados detalles muy pronto o representan aspectos procedimentales que complican el flujo de información.

Existe una clase de numerosas aplicaciones que están “conducidas” por sucesos en lugar datos, que producen información de control más que informes o visualizaciones, que procesan información con fuertes limitaciones de tiempo y rendimientos. Estas aplicaciones requieren una modelización del flujo de control además de la del flujo de información. Esta situación no se da en nuestro sistema, por lo que no entraremos en detalle sobre este tema.

Se usa la especificación de procesamiento para describir todos los procesos del modelo de flujo que aparecen en el nivel final de refinamiento. El contenido de la especificación de procesamiento puede incluir una descripción en lenguaje de programa del algoritmo en proceso, ecuaciones matemáticas, tablas, diagramas o gráficos.

La notación original para el análisis estructurado fue desarrollada para aplicaciones de procesamiento de datos convencionales, pero ahora hay ampliaciones que permiten aplicar el método a los sistemas de tiempo real. El análisis estructurado está soportado por una larga lista de herramientas CASE que ayudan en la creación de cada elemento del modelo y también en el mantenimiento de la consistencia y de la corrección.

4. ANÁLISIS DEL PROCESO DE FORMA ÚNICA.

Para la realización de trámites administrativos de su personal, las dependencias elaboran un documento denominado Forma Única (FU), donde se registran los movimientos del empleado referentes a su contratación, licencias, modificaciones a su situación actual o término del contrato.

Una vez elaborada, es enviada a la Dirección General de Personal (DGP), para la verificación y aplicación de los movimientos que contiene.

El módulo de Forma Única es en el que se procesará la entrada de información referente a los movimientos de Formas Únicas, los cuales se validan tomando como base las normas establecidas por la institución.

Las Formas Únicas son ingresadas a la base de datos de nómina, mismas que contienen la siguiente información: número de folio, número de empleado, tipo de movimiento, clave de categoría, sueldo, número de horas, fecha de inicio del movimiento, fecha de fin del movimiento, fecha de elaboración, fecha de liberación, etc.

El tipo de movimiento se refiere a lo que vamos a hacer con el nombramiento del empleado. Los tipos de movimiento que tenemos son:

- Alta.
- Baja.
- Licencia con sueldo.
- Licencia sin sueldo.

Dependiendo de los movimientos que tenga el empleado se pueden clasificar en distintos tipos de proceso:

- Nuevo Ingreso. El empleado no existía en nómina y sólo tenemos movimientos de alta.
- Reingreso. El empleado ya estaba registrado en nómina y se va a dar de alta nuevamente, que fue dado de baja con anterioridad.
- Promoción. Se le va a aumentar el sueldo al empleado. Tiene movimientos de alta y baja de nombramientos.

- Baja parcial. Se le va a dar de baja sólo alguno de sus nombramientos, y seguirá activo.
- Baja total. Se le van a dar de baja todos sus nombramientos y solamente tiene movimientos de baja.

4.1.1 DEFINICIÓN DEL PROBLEMA.

Los movimientos de Forma Única se validan para confirmar su procedencia, de acuerdo al presupuesto y en combinación con otros nombramientos del empleado. Durante el proceso de Formas Únicas se actualizan los nombramientos, la historia de movimientos y datos para el cálculo de la quincena.

Los movimientos de Forma Única llegan con retraso para su proceso en nómina; por lo tanto, es necesario calcular sueldos e impuestos retroactivos, así como, los pagos y descuentos acumulados durante el periodo retroactivo. También se actualiza la antigüedad del empleado.

Uno de los objetivos de este módulo es calcular estos pagos y descuentos para su aplicación durante el proceso de cálculo de la quincena, en los siguientes grupos de conceptos:

- Sueldos y asueldos retroactivos.
- Impuestos retroactivos.
- Pagos y descuentos retroactivos.

Los movimientos de Forma Única producen una serie de cambios, en datos asociados con el empleado, dependiendo del tipo de movimiento y del tipo de empleado. El objetivo de este módulo, es realizar las siguientes actualizaciones:

- Antigüedad del empleado.
- Nombramientos del empleado.
- Datos generales del empleado.
- Ajuste de asignación.
- Historia de movimientos.
- Pagos y descuentos retroactivos de la quincena.

Los diferentes tipos de empleados que tenemos son los siguientes: base, confianza, funcionario, asignatura, carrera y honorarios.

Existen varios grupos de categorías, que en combinación con las partidas, determinan algunos procesos especiales,

- Categorías de funcionarios. Estas categorías se consideran especiales por el trato que implica el ajuste de asignación.
- Categorías de complementarios, zonas geográficas y categorías fuera de tabulador. Éstas se consideran especiales, porque el importe del sueldo se determina de acuerdo a un porcentaje del sueldo de la categoría de un nombramiento o conforme a reglas aceptadas por usos y costumbres en la UNAM.
- Categorías normales. Todas las categorías que no requieren un tratamiento especial y que se encuentran en el tabulador, se consideran categorías normales.

El objetivo principal del módulo es modificar los nombramientos del empleado, los pagos y descuentos retroactivos generados por el movimiento y la actualización de sus datos, antes de tomar la información proporcionada por Forma Única debemos corroborar que los datos sean correctos, para lo cual las validaciones que se hacen son las siguientes:

- De categorías contra sueldos. Es decir, que el sueldo corresponda con la categoría, cuando se trata de un sueldo tabular.
- Número de plaza. Que exista en la plantilla de plazas.
- Código programático. El cual es un conjunto de dígitos que, ordenados en forma sistemática, se constituye en el elemento central para procesar el total de información que demanda el Sistema de Presupuesto por Programas. Debe ser un código vigente.
- Las horas que nos marca el movimiento, las cuales deben corresponder a la categoría propuesta, a excepción de los movimientos de asignatura.
- Las partidas de las categorías especiales.
- Existen empleados que se les da la mitad de una plaza, sobre todo a los de base y confianza, para que con la plaza y media plaza tengan las 48 horas a la semana. Las medias plazas deben corresponder con el código programático y categoría de la plaza.

- En el caso de las bajas, la categoría debe coincidir con la que el empleado tiene registrada en la tabla de nombramientos.
- Las horas de asignatura que se van a dar de baja, no deben sobrepasar las que tiene el empleado registradas en la tabla de nombramientos.
- Se verifica que no se incluyan en el proceso las licencias con sueldo, debido a que el empleado está cobrando y aunque esté de licencia se le debe de pagar.
- No deben existir movimientos duplicados, es decir que exista una alta o baja en la misma quincena de la misma categoría dos veces.
- Para sueldos no tabulares tenemos varios casos, a continuación se presenta una tabla de cómo se deben calcular las zonas geográficas y complementarios. Aquí se verifica que se calcule el sueldo correctamente, si falta algún dato, entonces se debe notificar al usuario que no se hizo el cálculo.

Clave	Descripción	Porcentaje	Forma de cálculo
M01 M03 M21 M22 M23 M24	 San Pedro Mártir. Prof. de asignatura. Prof. de carrera. Invest. de carrera. Técnico académico.	% Aumento	Esto quiere decir que se toma el sueldo tal y como está en la Forma Única y si pasa por algún cambio de tabulador se multiplica por el porcentaje de aumento. SdoFU = SdoFU * (1+ %aum)
M02		Fijo	Para este complementario, se toma el sueldo como esta en la FU, sin aumento.
M04	Montaña administrativo.	75 %	Tomamos los sueldos de los nombramientos que le corresponden de acuerdo a la descripción. Sdo = (Sdos Totales) * 0 .75
M05 M13 M14 M25 M26 M27	Funcionarios. Admvs Juriquilla. Confianza. Invest. de carrera. Técnicos acad. Investigadores de carrera.	40 %	Se toma la suma de los sueldos del tipo de empleado al que le corresponde la descripción y se le aplica el porcentaje. Sdo = Sdos Totales * 0.40
M06 M11 Z08	TV Unam, base San Pedro Mártir.	50%	Se toman los sueldos totales correspondientes a las categorías de la descripción y se les aplica el porcentaje. Sdo = Sdos Totales * 0.50
M07	TV UNAM.	25%	Se toman los sueldos totales que corresponden a las categorías de la descripción y se les aplica el porcentaje. Sdo = Sdos Totales * 0.50
M08	Bomberos.	30%	Se toman los sueldos totales que corresponden a las categorías de la descripción y se les aplica el porcentaje. Sdo = Sdos Totales * 0.30
M09	Bibliotecarios.	33%	Se toman los sueldos totales que corresponden a las categorías de la descripción y se les aplica el porcentaje. Sdo = Sdos Totales * 0.33

Clave	Descripción	Porcentaje	Forma de cálculo
M10	Bibliotecarios.	18%	Vamos a tomar los sueldos de nombramientos de base del empleado, correspondientes a la descripción, después se les aplica el porcentaje. Sdo = Sdos Totales *0.18
M12	Radio Unam.	10%	Sólo se da este complementario al personal de la descripción. Se toman los sueldos de base totales y se les aplica el porcentaje. Sdo = Sdos Totales * 0.10
Z01	Base.	% Dependencia	Para la Z01 se toman los nombramientos de base más el complementario M04, se suman y a esto se le aplica el porcentaje que le corresponda a esa dependencia. Sdo = (Sdos Totales Base + Sdo Complementario M04) * Porc
Z02 Z03 Z04 Z05 Z06 Z07	Asignatura. Carrera. Carrera. Carrera. Funcionarios. Confianza.	% Dependencia	Se toma la suma de sueldos totales de nombramientos del tipo de empleado de la zona geográfica que se trate y se le aplica el porcentaje que la dependencia tenga asignado. Sdo = Sdos Totales * Porc
Z09		50%	La Z09 es un caso especial, para esta zona geográfica se toman los sueldos de la M03 y a esto se le aplica el porcentaje. Sdos = Sdo M03 * 0.50

También se da el caso de empleados que tienen nombramientos de funcionario y nombramientos de asignatura y carrera, o alguno de ellos, a éstos se les conoce como nombramientos de funcionario con base académica, por lo tanto debemos hacer un ajuste, para lo cual sabemos que un sueldo de funcionario se divide en una cuota de asignación (CA) y una cuota fija (CF); al sueldo de la CA se le va a restar el sueldo o los sueldos de la base académica, de tal manera, que el empleado quede con un sueldo de funcionario; sin embargo, es conveniente aclarar que no siempre sucede así y en algunos casos gana un poco más, porque la base académica es mayor que la CA. Para entender lo anterior veremos algunos ejemplos.

> Antes que nada investigamos el sueldo de la CA y CF de la categoría respectiva, debemos saber que el valor de una CF no se puede modificar bajo ninguna circunstancia; después obtenemos la suma de sueldos de carrera (tipo de empleado 5) Sdo Carr, suma de sueldos de asignatura (tipo de empleado 4) Sdo Asig, calculamos el complemento de horas[1] Sdo Comp y el ajuste se hace como sigue:
>
> Sdo Carr + Sdo Asig + Sdo Comp = SdoBaseAcad.

[1] Si el empleado tiene 15 horas de asignatura o más, entonces si le corresponden complemento, su valor en este momento es de 2.5.

Si el empleado tiene 15 horas o más, sólo entonces: Sdo Comp = No. de horas * 2.5.

Sdo Ajustado = CA – (SdoBaseAcad + Sdo Comp).

Si el sueldo ajustado es negativo, quiere decir que la suma de sueldos es mayor que la CA, en este caso el empleado queda con su CF, de lo contrario el valor de la CA = Sdo Ajustado. A continuación se presentan algunos ejemplos:

CASO 1. El empleado tiene lo siguiente:

CA8316	Sdo total	9,552.24
	Sdo CA	7,768.24
	Sdo CF	1,784.00
D4100	Sdo 4 hrs.	703.76

	7,768.24
menos	703.76
Sdo Aju	7,064.48

El empleado queda como sigue:

Sdo CA	7,064.48
Sdo CF	1,784.00
Sdo Horas	703.76
TOTAL	9,552.24

CASO 2. El empleado tiene lo siguiente:

CA5017	Sdo total	8,541.48
	Sdo CA	6,927.48
	Sdo CF	1,614.00
D4100	Sdo 28 hrs.	4,926.35
	Compl.	70.00

	6,927.48
menos	4,926.35
menos	70.00
Sdo Aju	1,931.13

El empleado queda como sigue:

	Sdo CA	1,931.13
	Sdo CF	1,614.00
	Sdo 28 Horas	4,926.35
	Sdo compl.	70.00
	TOTAL	8,541.48

CASO 3. El empleado tiene lo siguiente:

CA5017	Sdo total	10,246.24
	Sdo CA	7,768.24
	Sdo CF	1,478.00
D8682	Sdo	7,133.08

		7,768.24
	Menos	7,133.08
	Sdo Aju	635.16

El empleado queda como sigue:

	Sdo CA	635.16
	Sdo CF	2,478.00
	Sdo Carr	7,133.08
	TOTAL	10,246.24

CASO 4. El empleado tiene lo siguiente:

CA7608	Sdo total	13,999.64
	Sdo CA	10,061.64
	Sdo CF	3,138.00
D5356	Sdo	4,351.74
D4100	Sdo 4 hrs	703.76

		10,061.64
	Menos	4,351.74
	Menos	703.76
	Sdo Aju	5,006.14

El empleado queda como sigue:

Sdo CA	5,006.14
Sdo CF	3,138.00
Sdo Carr	4,351.74
Sdo Asig	703.76
TOTAL	10,246.24

Estos sólo fueron algunos de los casos de ajustes de funcionarios que pueden llegar a presentarse.

- Además de los casos anteriores, existen otros movimientos especiales; los empleados de carrera, por algunas circunstancias se les da un porcentaje de su plaza como sueldo, es decir, en lugar de cobrar el sueldo completo, cobran un porcentaje del sueldo tabular.

- Existe una categoría para compensación académica, llamada D4801 o D4802, la cual se ajusta con las horas de asignatura para que al sumar ambos sueldos, se obtenga el sueldo de un nombramiento de carrera.

- Existen algunas categorías de funcionario, de treinta y dos horas a la semana, que no siempre se ajustan como en los casos presentados anteriormente, sino, que pueden no ir ajustadas o ajustarse sólo con unas horas de asignatura del total que tiene el empleado.

- Como uno de los principales problemas es la retroactividad de los movimientos, hay ocasiones cuando la Forma Única llega a la nómina, el nombramiento ya terminó, aunque el empleado no queda registrado con un nombramiento, se le debe pagar el tiempo que trabajo; es decir, tenemos un movimiento con fecha de inicio 1º de enero y fecha fin 25 de junio del mismo año, pero la Forma Única se procesa en la primera quincena de julio, para esta fecha el empleado ya no está trabajando pero se le deben seis meses de sueldo, por lo que se genera lo que se conoce como un “único pago”.

- Ahora bien, como se contratan por semestre, los únicos pagos generalmente vienen juntos y no es correcto pagarlos como

únicos pagos, ya que hay algunos cálculos que se hacen especiales para éstos, por lo tanto hay que detectarlos y verificar si son continuos y el empleado queda activo o bien hacer un solo movimiento.

Después de revisar todas esas condiciones y hacer los cálculos correspondientes para la validación, debemos saber que el sueldo calculado es el que le corresponde por la fecha de inicio de Forma Única.

Es el primer paso, pero para algunos de los reportes que se generan requerimos conocer la situación final del empleado, aún sin actualizar la tabla donde se guarden los nombramientos del empleado, por lo tanto, debemos aplicar los movimientos de los empleados con el sueldo actual, para poder determinar de manera temporal, como va a quedar su situación final.

Entonces podemos darnos cuenta de que en primera instancia se requirió el sueldo que se proporciona en la Forma Única, el cual puede ser de este año o años anteriores y el sueldo actual para el nuevo nombramiento. Pero también debemos obtener el sueldo por cada uno de los tabuladores, que existen para esa categoría, desde su fecha de inicio hasta la quincena actual; es decir, si la Forma Única tiene una fecha de inicio del 26 de marzo del 1998 y es un empleado de carrera, los cuales han tenido aumento salarial el 1º de febrero de cada año, entonces necesitamos los aumentos del 1º de febrero de 1999, 2000, 2001, 2002 y 2003.

Ahora bien, se debe aclarar que para las categorías especiales como zonas geográficas, complementarios, funcionarios con base académica, etc., se deben realizar los cálculos correspondientes para cada cambio de sueldo, esto quiere decir, que obtenemos los diferentes sueldos para cada aumento.

Como ya se había mencionado con anterioridad uno de los principales problemas que se presentan es tener los sueldos en todo momento para cualquier consulta. Y otro problema, es hacer la simulación de la aplicación de los movimientos de Forma Única, para saber como va a quedar el empleado.

Además debemos buscar la manera de realizar los cálculos de pagos y descuentos retroactivos más rápido, verificar que se tiene que hacer o que datos debemos obtener para calcularlos.

Uno de los reportes que se generan, es el reporte de incompatibles, en el cual se identifican cuáles movimientos tienen alguna condición de incompatibilidad, y reportarlos para que en el área correspondiente los revisen y nos indiquen en cada caso que se debe hacer, principalmente son dos cosas que se hacen, retenerlos o cancelarlos.

Los movimientos incompatibles son los siguientes:

- Aquellos empleados que tienen movimientos retenidos de la quincena anterior.
- Cualquier movimiento de funcionario.
- Todos los movimientos de la dependencia 442 (ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ACATLAN").
- Todos los movimientos de la dependencia 413 (FACULTAD DE CONTADURIA Y ADMINISTRACIÓN).
- Cualquier movimiento de empleados de honorarios.
- De acuerdo con la situación final del empleado, debemos reportar como incompatibles, aquellos empleados de base que tengan más de 48 y menos de 30 horas.
- Los empleados de confianza que tengan más de 48 y menos de 32 horas.
- Los empleados de asignatura que tengan más de 40 horas.
- Los empleados que son ayudantes de profesor no deben tener más de 12 horas, de ser así también son reportados.
- Todos los demás empleados que rebasen las 48 horas deben ser reportados como incompatibles
- Si tenemos un movimiento de asignatura o carrera que afecta a un funcionario también es incompatible.

Aquí reflejamos la situación final del empleado para que verifiquen como va a quedar el empleado después de aplicar los movimientos de la quincena.

En el reporte de altas y bajas se reportan todos los movimientos que serán procesados en la quincena.

Otro de los reportes importantes del módulo, es el de banco de horas; los movimientos de profesor de asignatura disminuyen o aumentan la disponibilidad de horas presupuestadas por dependencia y subdependencia. Estos movimientos se verifican para determinar si el banco de horas puede cubrir las necesidades de los nombramientos actuales, considerando los movimientos de baja y de alta. Los movimientos que no tienen suficiencia presupuestal se identifican en un

reporte que se revisa en la dependencia responsable del control presupuestal, la cual tomará las medidas correspondientes, aunque tenga que retener movimientos.

Finalmente cuando ya se hicieron los cálculos correspondientes, se revisaron las Formas Únicas, se realizaron los reportes y se modificaron los datos pertinentes, tenemos todo listo para cambiar los nombramientos del empleado con la Forma Única. Además de esto, se actualizan los datos del empleado que se generen por dicho movimiento, por ejemplo: lugar de pago, marca de sindicato, estado en NÓMINA, tipo de empleado, fecha de ingreso y reingreso a la UNAM, el pride, etc.

Los nombramientos se actualizan para que el proceso de cálculo pueda hacer los pagos correspondientes a la quincena normal.

4.1.2 ANÁLISIS DE REQUERIMIENTOS.

Después de analizar el proceso, podemos hacer los DFD's correspondientes y comenzaremos con el DFD 0, el cual nos ilustra el proceso principal.

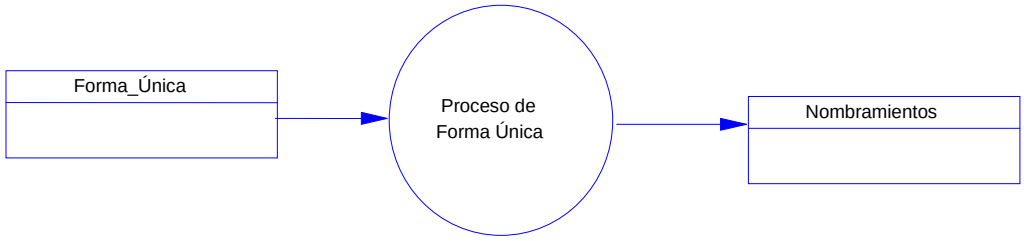


Figura 4.1 DFD general del proceso de Forma Única.

Tenemos como información de entrada la Forma Única, ingresa al proceso de Forma Única y la información de salida son los nombramientos actualizados.

Ahora veamos el proceso desglosado:

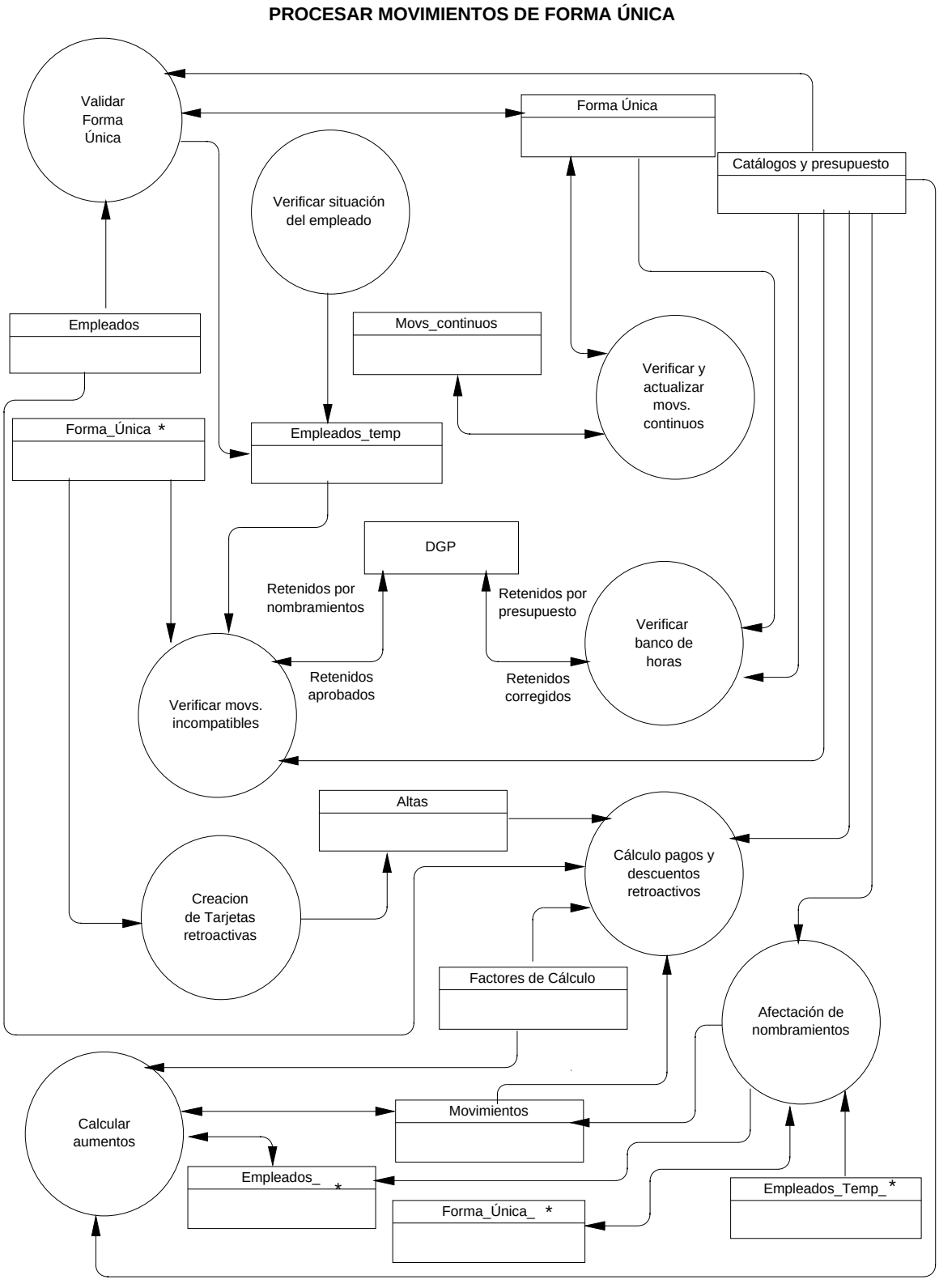


Figura 4.2 Proceso de validación desglosado.

En la figura 4.2 podemos observar de una manera más clara lo que es el proceso de Forma Única, se identifican los procesos principales y el orden que seguiríamos sería: validación, verificación, movimientos continuos, incompatibles, banco de horas, creación de tarjetas retroactivas, cálculo de conceptos retroactivos y por último la afectación a nombramientos. Ahora veremos el desglose de cada uno de ellos.

VALIDAR FORMA ÚNICA

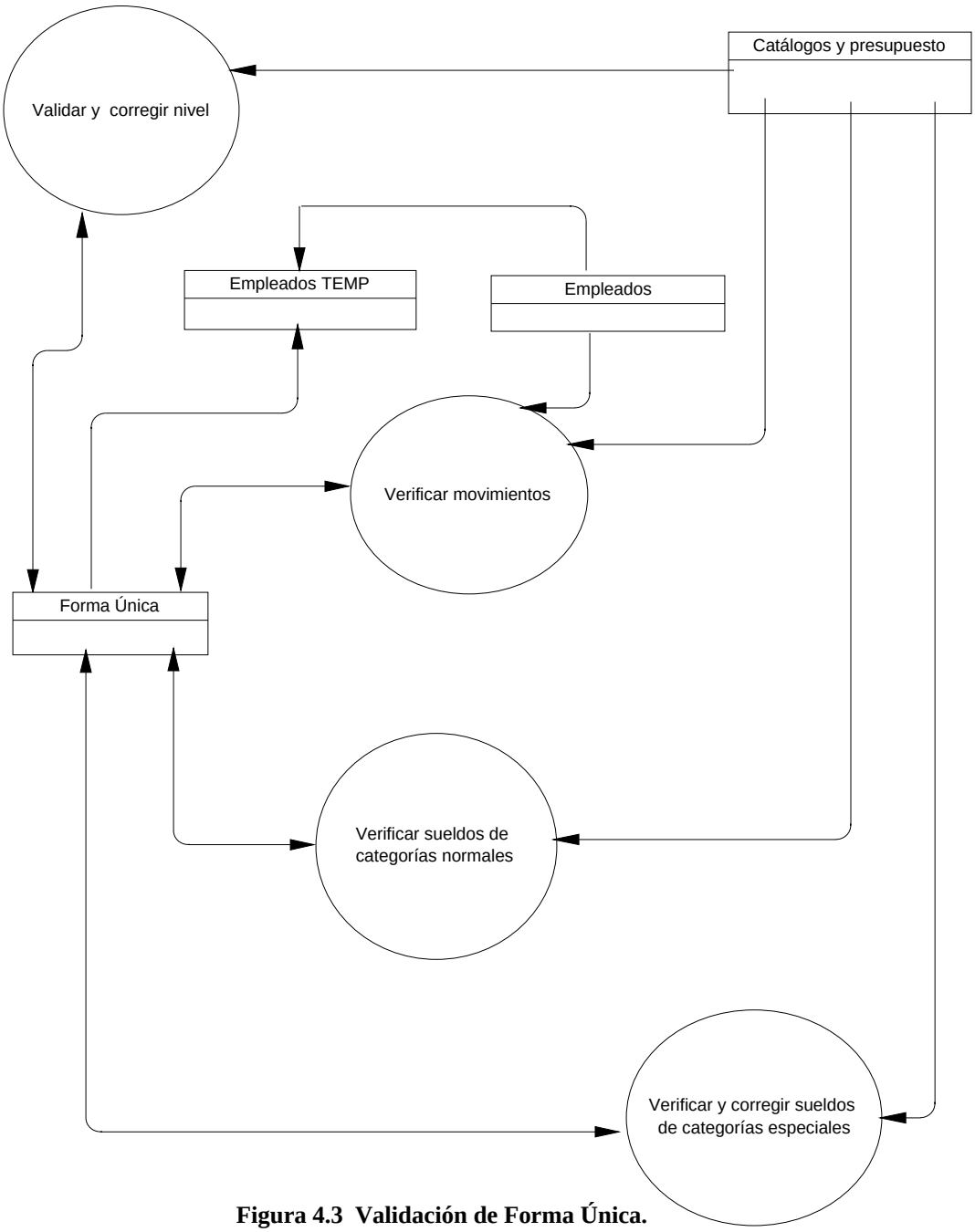


Figura 4.3 Validación de Forma Única.

La validación se encarga de verificar que la información sea correcta de acuerdo a los puntos tratados en la definición del problema y que se refieren a los movimientos duplicados, sueldos, categorías, plazas, códigos programáticos, etc.

Si observamos existe un depósito de datos adicional que se llama Empleados Temp, ya que de acuerdo al problema debemos saber la situación del empleado antes de cambiar los nombramientos originales, por lo tanto, vamos a usar un depósito de datos temporal, en el cual vamos a aplicar los movimientos del empleado, previamente copiamos la situación actual del empleado.

VERIFICAR LA SITUACION DEL EMPLEADO

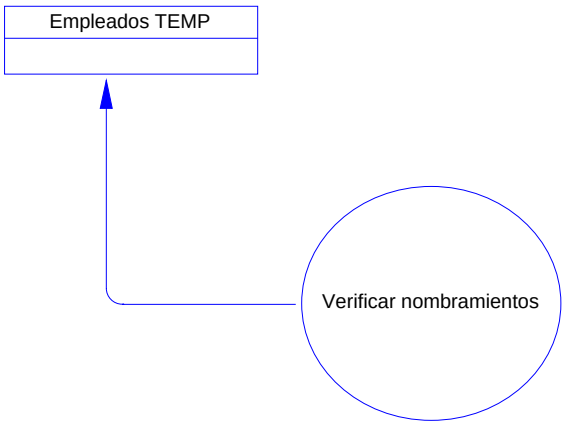


Figura 4.4. Verificación de la situación del empleado.

En esta parte del proceso, verificaremos la información contenida en el depósito de datos de los nombramientos temporales, que mencionábamos en la parte de validación, además de actualizar algunos datos referentes al ajuste de asignación de los funcionarios con base académica y el complemento de horas.

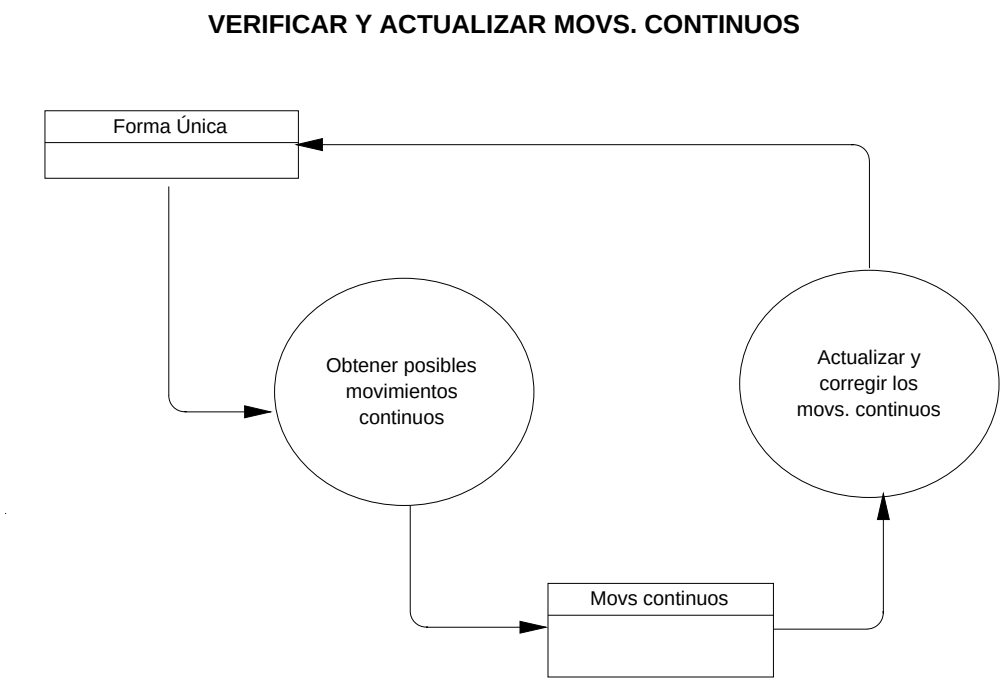


Figura 4.5. Movimientos continuos.

Se realizará un proceso independiente para verificar si hay movimientos continuos, que como se explicaba anteriormente son aquéllos que se generan por únicos pagos que vienen juntos y que nos indican que el empleado ha sido contratado por semestres continuos. Aquí se buscan ese tipo de movimientos y se hacen los cambios pertinentes a la Forma Única.

VERIFICAR MOVIMIENTOS INCOMPATIBLES

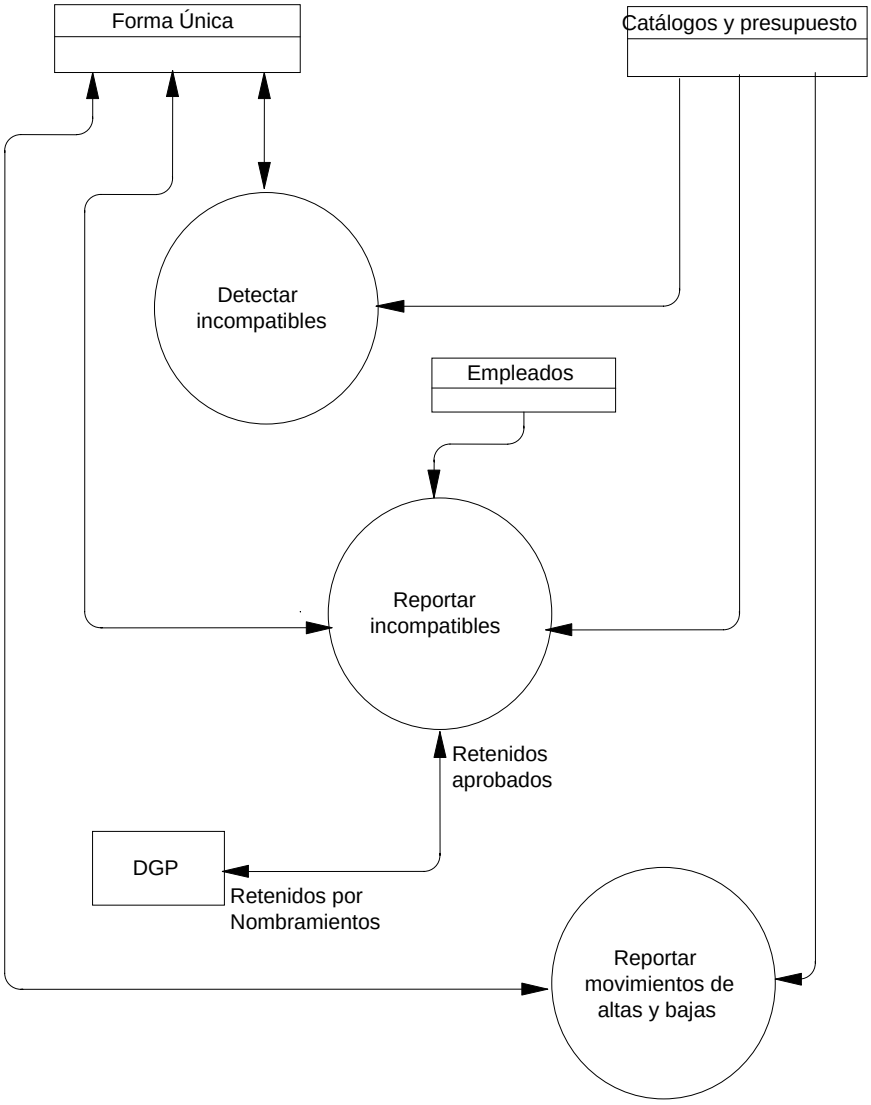


Figura 4.6 Proceso de incompatibles.

Este proceso se encarga de revisar los nombramientos temporales del empleado, y reportar los que tienen alguna condición de incompatibilidad.

VERIFICAR BANCO DE HORAS

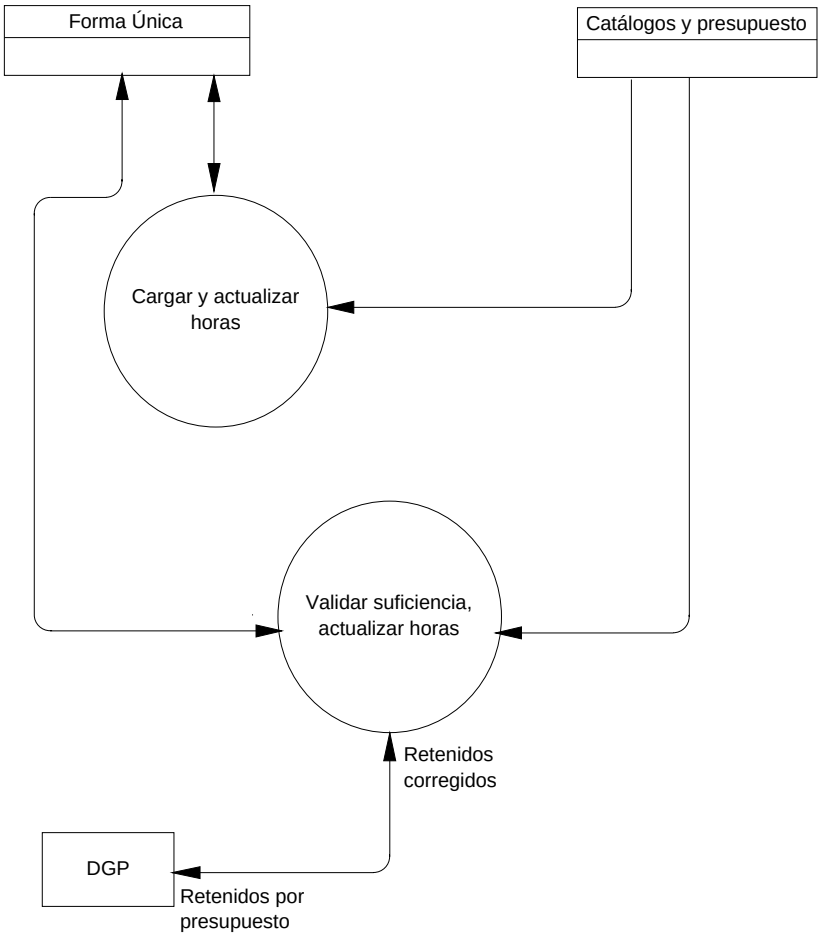


Figura 4.7. Banco de horas.

Este proceso también se encarga de generar un reporte, el de banco de horas. Aquí tenemos depósitos de información especiales para tomar la información del presupuesto que nos proporcionan por dependencia y determinamos si los movimientos tienen suficiencia presupuestal, de no ser así, se realiza el reporte para la revisión que hace el área correspondiente.

CREACION DE TARJETAS RETROACTIVAS

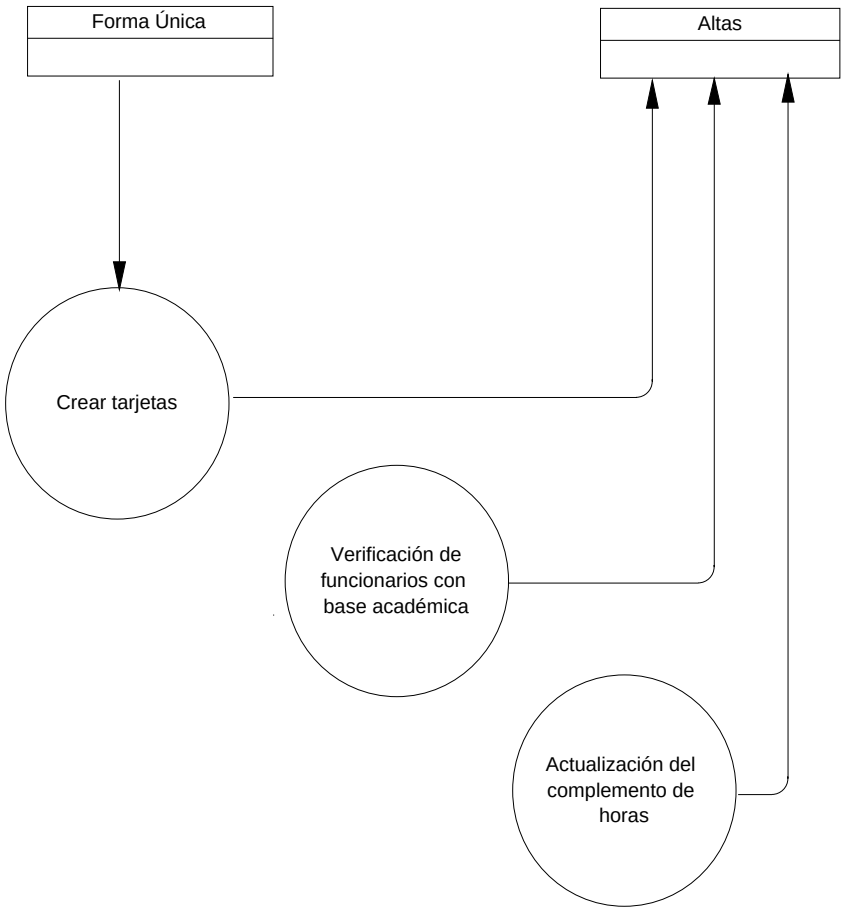


Figura 4.8 Creación de tarjetas retroactivas.

En este proceso se prepara la información para el cálculo de pagos y descuentos retroactivos, dicha preparación consiste en dividir el movimiento de Forma Única en un registro por cada tabulador diferente que exista, es decir, dividir los movimientos de acuerdo a los periodos de aumento de sueldo.

Si tenemos un movimiento que viene a partir del 15 de marzo del 2000, va a quedar activo y es de asignatura, se crean dos registros que vamos a llamar tarjetas, uno a partir del 15 de marzo del 2000 al 31 de enero del 2001 (cuando cambio el tabulador) y otro a partir del 1 de febrero del 2001 a la fecha de inicio de periodo (de la quincena en proceso), y por cada tarjeta se calculan los días de diferencia, para que en el siguiente proceso el cálculo sea más rápido. Cada Forma Única se va a

dividir en varios registros, según la retroactividad que tenga, creando una serie de tarjetas derivadas de un mismo folio.

Además, se calculan los datos generales de cada empleado para los cálculos de retroactividad, tales como antigüedad, sueldo diario para gratificación, los días que le corresponden por gratificación, porcentaje de pride, etc.

Estos datos se guardan en un depósito de datos (tabla), para usarlos posteriormente.

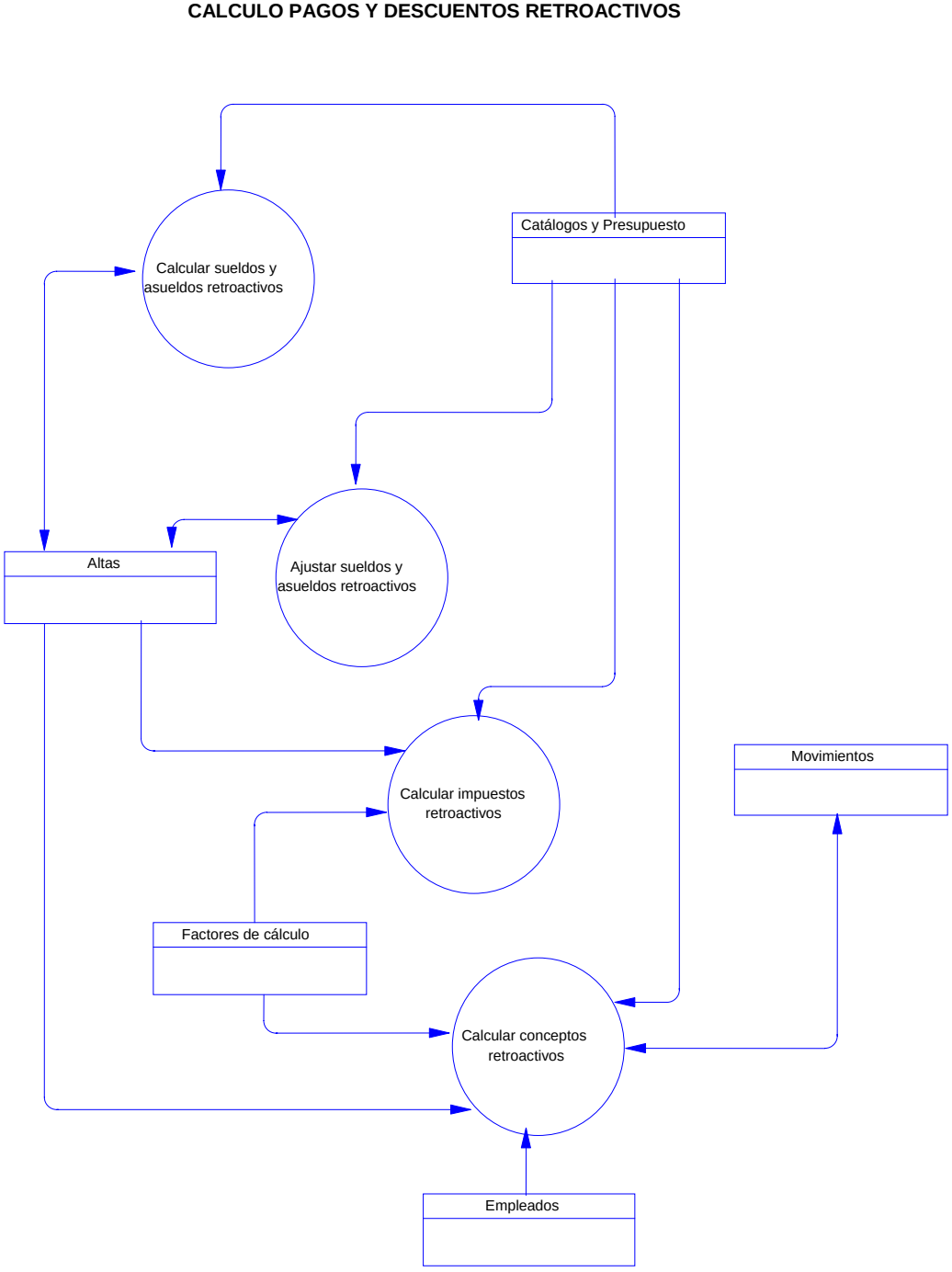


Figura 4.9. Cálculo de pagos y descuentos retroactivos.

Aquí se toman los datos que se generaron en el proceso anterior, primero se realiza el cálculo de sueldos (es el dinero que la universidad le debe a un empleado por una alta retroactiva) y asueldos (un asueldo es el dinero que el empleado debe por una baja retroactiva). Después se hace un ajuste entre los sueldos y asueldos para obtener el neto. Este dato es la base para el cálculo de los

impuestos y conceptos retroactivos, como serían gratificación, prima vacacional, aportación del ISSSTE, SAR, fondo de pensión, etc.

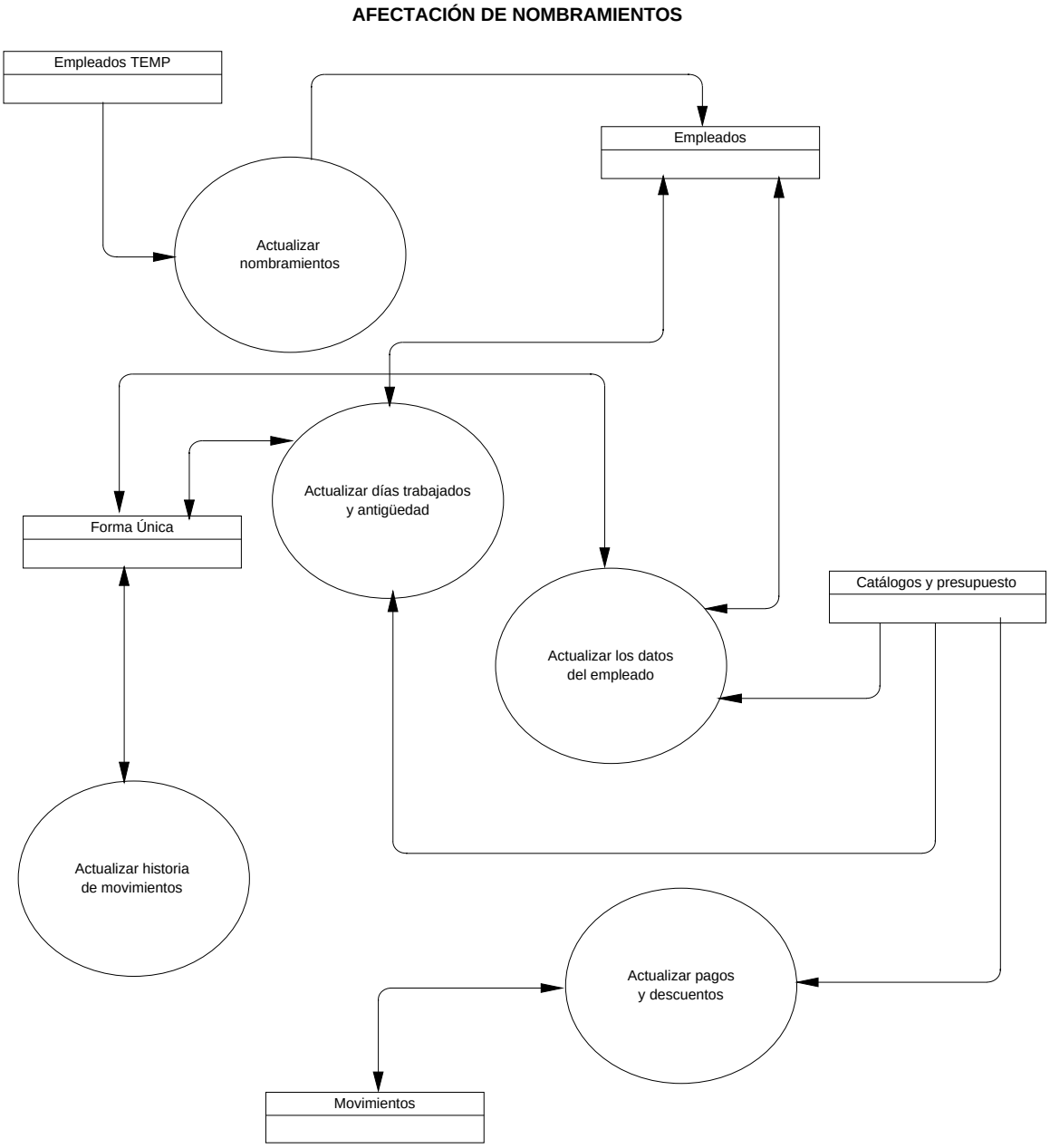


Figura 4.10. Afectación de nombramientos.

Ahora sí se actualizarán los nombramientos reales del empleado y los datos que correspondan. Los importes de los pagos y descuentos de cada empleado se copian a un depósito de información, donde el módulo de cálculo los podrá consultar e incluir en el cheque de la quincena.

5. EL DISEÑO.

El diseño es el primer paso, de la fase de desarrollo de cualquier producto o sistema de ingeniería. Se define como: "el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física".

El objetivo de un diseñador es producir un modelo o representación de una entidad que se construirá más adelante. Este proceso combina la intuición y los criterios basados en la experiencia de construir entidades similares, un conjunto de principios que guían la forma en la que se desarrolla el modelo.

En la etapa de diseño se van a traducir los requisitos en una representación del software. Este diseño se realiza en dos pasos, el diseño preliminar se centra en la transformación de los requisitos y la arquitectura del software. El diseño detallado se ocupa del refinamiento de la representación arquitectónica que lleva a una estructura de datos detallada y a las representaciones algorítmicas.

Además del diseño de datos, diseño arquitectónico y diseño procedimental, en muchas aplicaciones se requiere de un diseño de la interfaz, la cual establece la disposición y los mecanismos para la interacción hombre – máquina.

Los puntos esenciales con los que debe contar un diseño para ser de calidad son los siguientes:

- Debe exhibir una organización jerárquica que haga un uso inteligente del control entre los componentes del software.
- Debe ser modular, estar dividido de forma lógica en elementos que realicen funciones y subfunciones específicas.
- Debe contener representaciones distintas y separadas de los datos y de los procedimientos.
- Un diseño debe llevar a módulos (subrutinas o procedimientos) que exhiban características funcionales independientes.
- Debe llevar a interfaces que reduzcan la complejidad de las conexiones entre los módulos y el entorno exterior.

- Un diseño debe obtenerse mediante un método que sea reproducible y que sea conducido por la información obtenida durante el análisis de los requisitos del software.

En una solución modular se pueden formular muchos niveles de abstracción. En el nivel superior se establece una solución en términos amplios, usando el lenguaje del entorno del problema. En los niveles inferiores de abstracción se toma una orientación más procedimental. En el nivel más bajo de abstracción, se establece la solución, de forma que pueda implementarse directamente.

A medida que nos movemos por diferentes niveles de abstracción, trabajamos para crear abstracciones de datos y de procedimientos. Una abstracción procedimental es una secuencia de instrucciones que tienen una función limitada y específica. Una abstracción de datos es una determinada colección de datos que describen un objeto, como podría ser el cheque de nómina. Este objeto es en realidad un conjunto de muchos trozos de información.

Los conceptos de refinamientos sucesivos y modularidad están muy cerca del concepto de abstracción. Conforme va evolucionando un diseño, cada nivel de módulos de la estructura del programa representa un refinamiento en el nivel de abstracción del software.

La abstracción nos permite representar un objeto de datos a diferentes niveles de detalle y especificarlo dentro del contexto de la operación que se le puede aplicar.

El refinamiento sucesivo es una primera estrategia de diseño descendente propuesta por Nikaus Wirth. La arquitectura de un programa se desarrolla en niveles sucesivos de refinamientos de los detalles procedimientales. Se desarrolla descomponiendo el diseño hasta que se llega a sentencias del lenguaje de programación.

En cada paso del refinamiento, una o varias instrucciones del programa dado, se descomponen en instrucciones más detalladas. Este refinamiento termina cuando las instrucciones están expresadas en términos de la computadora usada o del lenguaje de programación. Conforme se refinan las tareas, también los datos pueden ser refinados, descompuestos o estructurados.

En el diseño también se maneja la modularidad, esto es, el software se divide en componentes con nombres y ubicaciones determinados, que se denominan módulos y que se integran para satisfacer los requisitos del sistema.

La modularidad es un atributo individual del software que nos permite que un programa sea manejable. El número de caminos de control, la expansión de las referencias, el número de variables y la complejidad global podrían hacer imposible su correcta comprensión.

La arquitectura del software se refiere a dos características: la estructura jerárquica de los módulos y la estructura de los datos. Los módulos se obtienen mediante un proceso de partición, que relaciona los elementos de una solución de software con partes de un problema del mundo real definido implícitamente durante el análisis de los requisitos.

La estructura de datos es una representación de la relación lógica existente entre los elementos individuales y datos. Debido a que la estructura de la información afectará al diseño procedimental final, la estructura de datos es tan importante como la estructura del programa en la representación de la arquitectura del software.

La estructura de datos dicta la organización, los métodos de accesos, el grado de asociatividad y las alternativas de procesamiento para la información.

La organización y la complejidad de una estructura de datos tan sólo esta limitada por el ingenio del diseñador. Aunque hay un reducido número de estructuras de datos clásicas, que constituyen los bloques con los que se construyen estructuras más sofisticadas. Las estructuras elementales son: el vector secuencial, la lista enlazada, los arreglos de n x n y el árbol jerárquico.

La estructura del programa define la jerarquía de control, independientemente de las decisiones y secuencias de proceso. El procedimiento del software se centra sobre los detalles de procesamiento de cada módulo individual y debe proporcionar una especificación precisa de éste, incluyendo la secuencia de sucesos, los puntos concretos de decisiones, la repetición de operación e incluso la organización y estructura de los datos.

5. 1 Diseño y flujo de información de Forma Única

Para el diseño del módulo comenzaremos con el diseño de las tablas necesarias para el proceso. Estas tablas no forman parte del diseño de la base de datos para nómina, fueron creadas especialmente para trabajar en este proceso de validación. A continuación se listarán las tablas que se crearon y lo que guarda cada una de ellas.

TAJUSTE. Aquí se guardan los registros de las diferencias de sueldos de manera temporal para realizar los ajustes correspondientes.

TALTAS. Se guardan los registros de las tarjetas retroactivas de acuerdo al tabulador.

TBANCOHORA. Se almacenan los registros del banco de horas que proporciona Presupuesto para aplicarle los movimientos que se generan en la quincena y proporcionar las nuevas horas disponibles y faltantes para cada código programático.

TDIFERENCIA. Almacena todos los registros de diferencias de sueldos y asueldos, así como, todos los conceptos que se calculan de los empleados.

TFUNERR. Aquí se guardan los diferentes errores que se generan durante la validación de la Forma Única, con el fin de obtener posteriormente un reporte.

TMOVDUP. Almacena una copia de los movimientos de Forma Única, pero sólo los de asignatura con el objeto de analizar cuáles son continuos y cuáles no.

TNOMBRAPREVIO. Se almacenan los movimientos de altas y bajas antes de afectar los nombramientos originales. Esta tabla la usamos durante todo el proceso, y al final se hace una copia de los datos de ésta a la tabla de nombramientos, únicamente de aquellos empleados que tengan movimiento.

TREFINCOM. Se guardan las causas de incompatibilidad durante la detección de movimientos incompatibles, para obtener el reporte correspondiente.

Se usan tablas que son catálogos, las cuales nos van a proporcionar información necesaria para el proceso, estas tablas son:

CATRIBUTOSCONCEPTO. Como su nombre lo indica, aquí tenemos los atributos de los diferentes conceptos que se van a pagar.

CCONCEPTO. Conceptos existentes.

CCATEGORIAS. Son las categorías que se tienen en la plantilla de la universidad.

CSUBPROGRAMAS. Códigos programáticos completos para cada dependencia universitaria.

CLUGARPAGO. Lugares de pago.

CMOVCON. Los conceptos que se deben calcular por empleado de acuerdo a los tipos de movimientos.

CMOVFUN. Las actualizaciones de los datos personales del empleado que son originadas por los movimientos de Forma Única.

CPERIODO. Fechas de inicio y de final de cada quincena, y también cual periodo está activo.

CPLAZAS. Se guarda la plantilla de plazas.

Y por último, las tablas que forman parte de la base de datos de nómina:

ADMINISTRADOR. Tabla que lleva el control de diferentes procesos de la nómina.

TEMPESP. Aquí se mantiene un registro de los empleados de carrera que cobran por porcentaje.

TEMPNOMINA. Guarda los datos generales del empleado relacionados con los nombramientos.

TEMPNOMINABAK. Igual que la anterior, pero respalda la información de los empleados que tienen movimientos, antes de que se hagan los cambios que se generan por Forma Única.

TMOVFUND. Aquí se almacenan los datos de la Forma Única, es la tabla fundamental del subsistema del proceso de Forma Única. Es conveniente aclarar que estos datos son el detalle de la Forma Única, es decir, los registros están divididos en altas y bajas.

TMOVFUNDBAK. Guarda lo mismo que la tabla anterior, pero es un respaldo de la información de los empleados que tienen movimiento.

TMOVFUNH. También son datos de la Forma Única, se conoce como el encabezado, es decir, datos generales de la Forma Única; aquí no sabemos si son altas o bajas, solamente que existen movimientos para un empleado en una quincena específica.

TMOVFUNHBAK. Es lo mismo que la anterior, información de respaldo.

TMOVIMIENTOS. Aquí se guardan todos los conceptos tanto de percepciones como de deducciones, que se generan en todos los subsistemas de nómina.

TMOVIMIENTOSBAK. Lo mismo que la anterior.

TNOMBRAMIENTOS. Aquí se guardan los nombramientos de los empleados. Es la tabla de donde se toman los datos para pagarle a un empleado.

TNOMBRAMIENTOSBAK. Tabla de respaldo de la anterior.

TTABLAS. Contiene la tabla del impuesto.

TTABULADOR. Contiene el tabulador, los sueldos tanto actuales como retroactivos de las diferentes categorías.

Para facilitar la identificación de los movimientos y controlar su procesamiento, se mantiene la actualización de una columna que contiene el estado de la Forma Única. Cada programa lee los movimientos de Forma

Única, identificados con un estado determinado y al terminar el proceso, actualiza el estado para indicar que fue procesado.

El módulo de validación de Formas Únicas maneja dos estados, uno es correcto y el otro es incorrecto. Los movimientos que se encuentran con estado correcto continúan al siguiente proceso, mientras que los incorrectos son revisados y tomados nuevamente para su proceso. A continuación un diagrama que muestra los estados que se utilizan en la parte de validación

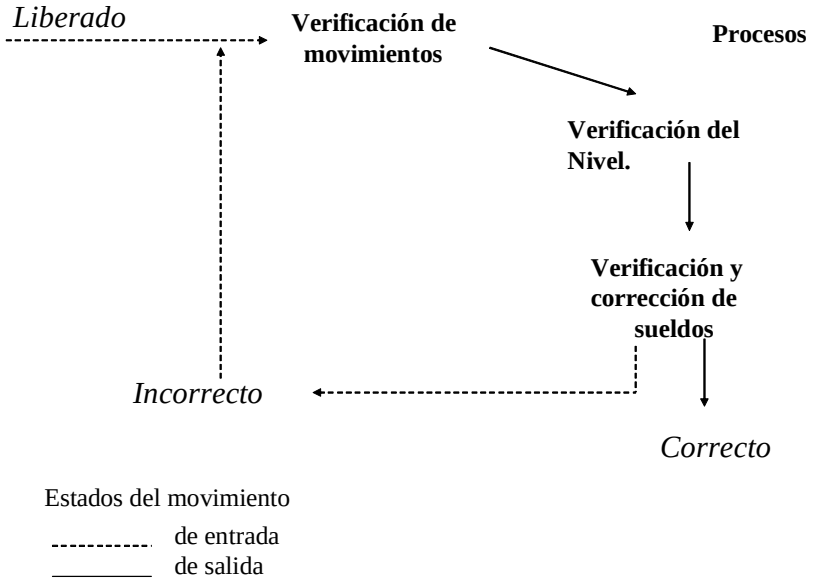


Figura 5.1 Estados en el subproceso de validación.

Para el proceso de validación se creará un programa llamado cnval.

Se preparan las Formas Únicas para el proceso.

- Se actualiza el estado de las Formas Únicas a 45 (SIN_NIVEL), solamente las que están en 35 (LIBERADO_SIFU) y además la fecha de inicio y la fecha de nómina sea igual a la fecha de inicio del periodo.
- Las Formas Únicas que no afectan a la nómina se les cambia su estado a 200 (HISTÓRICO).
- Se marca el movimiento como único pago cuando son altas, están en estado 45, y la fecha límite es menor o igual a la fecha de fin del periodo.
- Se seleccionan los empleados que se encuentran en proceso para actualizar la tabla de TNOMBRAPREVIO, que es donde simularemos

las altas y bajas. En este primer paso, lo que se hace es una copia de los nombramientos de los empleados a esta tabla.

- Se carga el tabulador.
- Seleccionamos el número de empleado de los movimientos que se encuentran en estado 45.

  - Por cada empleado seleccionamos sus nombramientos de la tabla TNOMBRAMIENTOS y mientras tenga nombramientos de ese empleado, se hace lo siguiente:
    - Si es de asignatura y no es zona geográfica, entonces se busca el sueldo actual en el tabulador.
    - Para todos los demás casos, incluyendo zona geográfica, se toma el sueldo que tiene en nombramientos.
    - Se inserta el registro en TNOMBRAPREVIO

  A partir de este momento se comienzan a validar las Formas Únicas. Seleccionamos el número de empleado, RFC (Registro federal de contribuyentes) y tipo de movimiento, ordenados éstos datos por tipo de movimiento en orden descendente, para procesar primero las bajas.

  - En el caso de las ALTAS, BAJAS y LICENCIAS SIN SUELDO, vamos a seleccionar todos los datos de la Forma Única, además algunos de la tabla de CCATEGORIAS, dependiendo del tipo de movimiento, pero si es baja o licencia con sueldo, los movimientos se trabajan por orden de proceso ascendente (de menor a mayor)[*] y en el caso de las altas es por orden descendente.
    - Si la categoría que tenemos es una CF, se revisa el sueldo, para saber si es correcto o debemos cambiarlo por una categoría CA.
    - Si el tipo de empleado es funcionario, no es Z (zona geográfica) ni M (Complementario), entonces se verifica la partida.
    - Si el tipo de empleado es carrera, confianza, base o funcionario con categoría CF.
      - ✓ Vamos a actualizar el valor de la suma de horas prácticas y teóricas de la Forma Única.
      - ✓ Se asigna el valor anterior a horas teóricas y horas prácticas, se actualiza con cero en la Forma Única.
    - Se verifican los datos principales con los que debe contar la Forma Única, tales como, número de plaza, código programático, categoría, horas, etc.

[*] Este orden se refiere a cuáles deben ser procesadas primero de acuerdo a la categoría, y es el siguiente: 1. zonas geográficas, 2. complementarios, 3. funcionarios y 4. categorías normales que se encuentran en el tabulador.

- Si la Forma Única viene marcada como proceso especial, y además es único pago, entonces:
  - ✓ Actualizar la categoría de referencia con la categoría que tiene la Forma Única y el campo cCveCateg con la categoría de proceso CP01, para el control de los cálculos especiales.
- Si la Forma Única viene marcada como movimiento especial y no es único pago:
  - ✓ Actualizar la categoría de referencia con la categoría que tiene la Forma Única y el campo cCveCateg con la categoría de proceso CP02, para el control de los cálculos especiales.
- Si es zona geográfica o funcionario, entonces:
  - ✓ Si es ALTA, buscamos en TNOMBRAPREVIO si tiene nombramientos de carrera o asignatura, y marcamos que el empleado tiene base académica.
  - ✓ Si es BAJA o LICENCIA SIN SUELDO, buscamos en TNOMBRAMIENTOS, si tiene nombramientos de carrera o asignatura, marcamos que el empleado tiene base académica
- Si el movimiento es una baja o licencia sin sueldo, se cambia la fecha límite del movimiento por la fecha de inicio del periodo.
- Si la fecha límite del movimiento es CERO, se va a manejar la fecha de inicio del periodo.
- Si la fecha límite del movimiento es diferente de CERO, mayor que la fecha de inicio del periodo y mayor que la fecha límite del periodo, entonces, se usará la fecha de inicio del periodo.
- Si la fecha límite del movimiento es diferente de CERO y menor o igual que la fecha de fin del periodo, se usará la fecha de la Forma Única.
- Si la categoría es alguna de las siguientes, A04, A05, A08, AA02, ET16, ET17, ET19, ET29 (son categorías de base), entonces:
  - ✓ Si el nivel es 40, 41, 42 ó 43, el sueldo se deja como está.
  - ✓ Si no, buscamos el sueldo en el tabulador y verificamos si es correcto lo que tiene la Forma Única.
- Si la categoría para profesionales de base (P01, P14, P64, P80, P82 ó P90), entonces:
  - ✓ Si es una P64 y el nivel es 40, 41, 42 ó 43 y además es una baja, el sueldo se queda como está.

- ✓ Si no, buscamos el sueldo en el tabulador y verificamos si es correcto.

❖ Si son complementarios de asignatura (M01, M03, M21, M22, M23 ó M24).

- ✓ Si es una baja o licencia sin sueldo, entonces verificamos si tuvo aumento y el sueldo que tiene es correcto, si no lo es, lo cambiamos.
- ✓ Si el movimiento es una alta, entonces el nivel es igual al sueldo/1000 (sólo se toma el valor entero), se le cambia el nivel a la categoría (cCveCateg) y el sueldo lo dejamos como esta.

❖ Si la categoría es de honorarios (AH), entonces:

- ✓ Si es la partida 186 y es baja o licencia sin sueldo, verificamos si tuvo aumento y el sueldo es correcto.
- ✓ Si es alta, el nivel va a ser igual al sueldo entre 1000 (sólo se toma el valor entero). El sueldo es correcto.

❖ Si la categoría es complementario o de honorarios (M02, H01, H02, DH01, PH56, PH57 ó PH64), entonces:

- ✓ Si es alta, el nivel es igual al sueldo/1000, cambiamos el nivel de la categoría y el sueldo se deja como correcto.

❖ Si la categoría es una CF93 (categoría de funcionario) o CP01, el sueldo es correcto.

❖ Si es una categoría especial para nombramientos de carrera con porcentaje (CP02), entonces:

- ✓ Se calcula el sueldo de acuerdo al porcentaje que se le asigne.
- ✓ Se copia la categoría de referencia (cCatRef) en la categoría (cCveCateg)

❖ Si son complementarios o zonas geográficas (M04, M05, M06, M07, M08, M09, M10, M11, M12, M13, M14, M15, M25, M26, M27, M28, Z02, Z03, Z04, Z05 ó Z07), entonces:

- ✓ Se calcula el sueldo que les corresponde de acuerdo a las reglas establecidas en el análisis.
- ✓ Si es alta, el nivel se obtiene dividiendo el sueldo entre 1000 y actualizamos la categoría.

❖ En el caso de la D48:

- ✓ Actualiza el complemento de horas.
- ✓ Se hace el cálculo de la categoría de compensación académica (D48).

❖ Para las zonas geográficas de base y funcionario (Z01, Z06 ó Z09):

- ✓ Se realizan los cálculos correspondientes de acuerdo a las reglas establecidas.
- ✓ Si es alta, se calcula el nivel con el sueldo entre 1000 y se actualiza.

❖ Si son categorías de honorarios de asignatura o profesores eméritos (DH41, DH42, EH41 ó EH42), entonces:

- ✓ Se busca el sueldo en el tabulador y se verifica si es correcto. Pero esto se hace con el tipo de empleado de referencia.

❖ Si la categoría es una D21 (de carrera), entonces:

- ✓ Si el nivel es 42:
  - ◆ Cambiamos la categoría por una D4200 (asignatura).
  - ◆ Se busca el sueldo en el tabulador y el tipo de empleado de asignatura.
  - ◆ Actualizamos de nuevo la categoría con la D2142.
- ✓ Si no, entonces se busca el sueldo en el tabulador con los datos originales.

❖ Si la categoría es de honorarios de carrera (DH o EH),

- ✓ Se busca el sueldo en el tabulador para verificar si es correcto, usando el empleado de referencia.

❖ Si es categoría de funcionario (CA, CH, RC o CV), entonces:

- ✓ Obtenemos los cuatro últimos dígitos de la categoría.
- ✓ Si es RC, en una variable auxiliar ponemos RF, sino entonces ponemos CF.
- ✓ Si es empleado con base académica, entonces:
  - ◆ Si el movimiento es de Alta, se actualiza el complemento de horas.
  - ◆ Se hace el ajuste de asignación.
- ✓ Si no tiene base académica, se busca el sueldo en el tabulador y verificamos si es correcto.

❖ Si la categoría es funcionario de 32 horas (CM o CD),

- ✓ Obtenemos los cuatro últimos dígitos de la categoría.
- ✓ Obtenemos la categoría CF, concatenando lo obtenido en el punto anterior con una CF.
- ✓ Si es empleado con base académica y se debe ajustar, entonces:
  - ◆ Si es alta, se actualiza el complemento de horas
  - ◆ Se hace el ajuste de asignación.
- ✓ Si no tiene base académica se busca el sueldo en el tabulador.

- ❖ Para cualquier categoría que sea diferente a las mencionadas anteriormente, se busca en el tabulador el sueldo para verificar si es correcto de acuerdo a los datos de la Forma Única.
- ❖ Si al revisar el sueldo en el tabulador, es correcto y las condiciones de la Forma Única también, entonces:
  - ✓ El movimiento se aplica a la tabla de TNOMBRAPREVIO, ya sea alta o baja.
  - ✓ Si todo se hizo correctamente, la Forma Única es correcta y se actualiza con estado 70.

➢ En el caso de la LICENCIA CON SUELDO:

- ❖ Se busca el folio y serial de este movimiento, para mandar el error de Licencia con Sueldo Afecta.

  Se imprime un mensaje en la pantalla: "Terminó el proceso de validación satisfactoriamente...", para indicarle al usuario que no hubo ningún problema al ejecutar el programa.

Es así como funciona el programa cnval, para la validación de Formas Únicas.

Ahora se analizará el programa de verificación de la situación del empleado. El programa se llamará cnver.

Como cada movimiento de Forma Única es independiente y sólo representa un nombramiento, este puede influir en él calculo de los demás nombramientos; por lo tanto, en este módulo se verifican todos los nombramientos de los empleados que tuvieron movimiento, y quedaron con alguna inconsistencia. Este módulo no cambia el estado de las Formas Únicas.

Se actualizará el complemento de horas de la siguiente manera:

- Se selecciona de TNOMBRAPREVIO el número de empleado de todos los que son de asignatura o de honorarios de asignatura (D41, D42, DH41, DH42, EH41, EH42) y que no sean únicos pagos.

➢ Ahora por cada empleado vamos a obtener la suma total de horas que tienen de D41, D42, DH41, DH42, EH41, EH42 y que no sean únicos pagos.

➢ Si la suma de horas es mayor o igual a 15, entonces vamos a actualizar en TNOMBRAPREVIO el campo destinado para el complemento de horas, el cual es igual a la suma de horas totales por 2.5, en todos los nombramientos del empleado que sean de las categorías mencionadas en el punto anterior y que no sean únicos pagos.

➢ En caso de que la suma de horas sea menor a 15, entonces el dato de complemento de horas va a ser igual a cero con las mismas

condiciones que en el punto anterior.

A continuación se hace la verificación de inconsistencias.

- Se carga el tabulador.
- Se seleccionan los datos de los empleados que tengan zona geográfica, complementario, funcionario, categoría D48, los bonos para músicos y las medias plazas, en la tabla de TNOMBRAPREVIO.

  - Se busca el RFC del empleado en TEMPNOMINA.
  - Si la categoría que tenemos es una CF y el nivel es menor de 20 y mayor de 87, se queda como está.
  - Si es una CF93, entonces verificamos el nombramiento.
  - Si la categoría es una Z06, CA, CD, CM, RC o CH, entonces buscamos en TNOMBRAPREVIO, si el empleado tiene nombramientos de Carrera o Asignatura, marcamos que el empleado tiene base académica.
  - La fecha límite va a ser igual a la fecha de inicio de periodo y nos va a servir como límite para el cálculo.
  - Si tenemos que verificar el nombramiento, entonces:
    - Si es M04, M05, M06, M07, M08, M09, M10, M11, M12, M13, M14, M15, M25, M26, M27, M28, Z02, Z03, Z04, Z05, Z07, Z01, Z06, Z09 ó D48 hace el cálculo del sueldo que le corresponde de acuerdo a las reglas.
    - Si la categoría es CA, CM, CD, CH o RC, entonces:
      - ✓ Si el empleado tiene base académica, entonces hacemos el ajuste correspondiente.
      - ✓ Si el empleado no tiene baja académica, buscamos el sueldo en el tabulador.
    - Si la categoría es M01, M03, M21, M22, M23 ó M24 buscamos en TNOMBRAPREVIO si tiene algún nombramiento diferente del complementario y que no sea único pago, si no lo tiene se imprime el número de empleado y el mensaje de que falta una Forma Única de baja.
    - Si la categoría es M04, M05, M06, M07, M08, M09, M10, M11, M12, M13, M14, M15, M25, M26, M27, M28, Z02, Z03, Z04, Z05, Z07, Z01, Z09 ó D48, entonces:
      - ✓ El sueldo calculado y el sueldo que tiene el nombramiento se dividen entre dos.
      - ✓ Si el sueldo calculado es cero, se imprime un mensaje de que falta la baja, con el número de empleado y la categoría.

- ✓ Si el sueldo calculado es diferente del sueldo que tiene la Forma Única y además es diferente de cero, entonces:

  Obtenemos la diferencia entre el suelo calculado y el sueldo de la Forma Única, si la diferencia es mayor de 0.50:

  - ◆ Buscamos la fecha de alta y de baja del movimiento.
  - ◆ Si la fecha de alta es cero y la fecha de baja es diferente de cero, entonces la fecha de alta es igual a la fecha de baja.
  - ◆ Si la fecha de alta es diferente de cero y la fecha de baja es cero, la fecha de baja es igual a la fecha de alta.
  - ◆ La fecha de alta y de baja son iguales a la fecha de inicio de periodo.

  Se imprime el número de empleado, categoría y el mensaje de que falta una baja y una alta para el empleado.

- ❖ Si la categoría es una CA, CM, CD, CH o RC, entonces:
  - ✓ Obtenemos los cuatro últimos dígitos de la categoría y lo concatenamos con una CF o RF, según sea el caso, en la variable cCategCF.
  - ✓ Se borran de TNOMBRAPREVIO las categorías CA y CF del empleado.
  - ✓ Si el sueldo CA, es menor o igual a cero, entonces actualizamos este nombramiento como inactivo.
  - ✓ Se inserta la categoría CA, y el sueldo ajustado que hemos calculado en este proceso.
  - ✓ Se inserta la CF con el sueldo correspondiente y este nombramiento siempre será activo.
  - ✓ Se actualiza la partida que le corresponde, de acuerdo al nombramiento de carrera o asignatura.
- ❖ En el caso de la Z06 (zona geográfica de funcionario):
  - ✓ Se borra el nombramiento de Z06 de la tabla, siempre y cuando no sea único pago.
  - ✓ Se inserta la categoría nuevamente, pero con el sueldo calculado en este proceso.
- ❖ Para todos los demás casos:
  - ✓ Si es una CF se reasigna la partida.
  - ✓ Si no, hacemos lo siguiente:
    - ◆ Investigamos si el empleado tiene nombramientos en TNOMBRAPREVIO, si es media plaza de base o de confianza debe tener un nombramiento de plaza completa, si es un0

bono de músico, debe tener una plaza de músico para que puedan tener este nombramiento.

- Si no tiene nombramientos y no son medias plazas, entonces se imprime el mensaje correspondiente. En este caso se esperan que ingresen los bonos de músicos que se quedaron sin nombramiento.

- Investigamos si ya no tenemos ninguna Forma Única en estado 45, si es así, entonces

  - Seleccionamos folio, periodo, nodo, serial, número de empleado, tipo de movimiento y categoría de los movimientos que están en TMOVFUND con estado correcto y que hayan sido marcados como bajas parciales o C7

  - Investigamos por empleado, si tienen todavía algún nombramiento en TNOMBRAPREVIO que no sea único pago.

  - Si ya no tienen nombramientos, entonces estos folios se marcan con Baja Total o C4.

Se manda el siguiente mensaje a la pantalla: "Terminó verificación de los empleados en TNOMBRAPREVIO..."

Ahora desglosaremos el proceso de verificación de movimientos continuos, cuyo programa será llamado ccont. Como se ha mencionado anteriormente, en esta parte vamos a identificar movimientos de asignatura que son continuos para no generar tantos únicos pagos.

Se verifica que todos los movimientos hayan pasado el proceso de validación satisfactoriamente. Se obtienen todos los movimientos agrupados por código programático, categoría, número de plaza, número de empleado y tipo de movimiento, y se cambian todos los movimientos a estado SIN_NIVEL.

Se les asigna este estado con el fin de poder obtener un reporte de los movimientos que posiblemente sean continuos. Además se copian los registros en una tabla temporal, que se llama TMOVDUP.

Después obtenemos los empleados que tengan más de un movimiento idéntico de la tabla TMOVDUP. Para realizar el análisis de movimientos continuos se requiere un arreglo bidimensional para guardar los datos que nos permitan determinar la continuidad de un movimiento con otro. Esto se hace de la siguiente manera:

- Por cada empleado se inicializa el arreglo, poniendo todas sus variables con cero.

- Se obtienen los movimientos del empleado ordenado por código programático, categoría, horas, fecha de inicio y fecha de fin.

  - Si es el primer registro se asignan todos los valores a las

variables del arreglo.

- Si no es el primer registro hace lo siguiente:
  - Recorre los renglones con i hasta 10 y recorre las columnas con j hasta 10.
  - En el ciclo verifica la posición vacía del primer renglón en i, sí la primera posición esta vacía se asigna el valor de j a la variable vacj e inicializa j igual a 10.
  - En caso contrario hace lo siguiente:
    - ✓ Verifica si en las coordenadas donde se encuentra, la fecha de inicio es idéntica a la fecha de inicio del movimiento.
      - Si es así, verifica que la siguiente casilla este vacía. Si esta vacía asigna a r el valor de i, la columna es igual al valor de j más 1, i y j cambia a 10.
      - Si no esta vacía, al renglón le asigna cero y a la columna el índice o valor del índice j e incrementa en uno el valor del índice j.
    - ✓ Si en la casilla no se encuentra la misma fecha de inicio del movimiento.
      - Resta en un día la fecha de inicio del movimiento.
      - Pregunta si la fecha fin del movimiento es idéntica a la posición de la casilla y además las horas son idénticas.
    - ✓ Y si la siguiente posición en la casilla es igual a cero, entonces el renglón se incrementa en uno con el valor de i, la columna es igual a valor de j, cambia i y j con 10.
    - ✓ Si el renglón y la columna son iguales a cero, i y j ya tienen el valor de 10 entonces asigna a columna el valor del índice, almacena en la posición vacía los datos del movimiento.
- Recorre nuevamente la estructura pero ahora para cambiar los movimientos, vuelve hacer una lectura de cada posición con i y j hasta 10.
  - Si en la primera casilla él número de folio es mayor de cero y la casilla siguiente del renglón también es mayor.
    - ✓ Si es la primera parte del movimiento continuo se guardan el folio y el serial.
    - ✓ Si no es la primera parte, entonces se cambia el estado del movimiento, se cambia la afectación a nomina como ‘N’ y se imprime el dato.

- Si en la primera casilla el número de folio es cero o el de la siguiente casilla.
  - ✓ Si es el último continuo cambia la afectación a 'N', cambia de estado, asigna la fecha fin a la variable, imprime la información.
  - ✓ Se actualiza la fecha fin del último continuo al folio y serial del primer movimiento continuo.
  - ✓ Se asigna el valor de 10 a la columna.
- Si esta en la casilla y el renglón es cero y la posición de la columna ya no tiene número de folio se vuelve asignar j a 10

Al final del proceso tendremos los movimientos que se cambiaron en estado 45, para volver a realizar la validación, el campo que nos indica sí es único pago, porque probablemente ya dejó de serlo. Y los movimientos cancelados ya no afectarán a la nómina y quedarán fuera del proceso.

Continuando con el desglose de los procesos, ahora diseñaremos el proceso de incompatibles, el programa se llamará cninc.

El proceso está compuesto por una función y dos programas que son invocados por un programa integrador, que además de ejecutar cada uno, prepara las tablas temporales que se requieren, produce información de salida y evalúa el estado de salida de los programas.

Los procesos del subsistema se ejecutan con base a conjuntos de movimientos de Forma Única que se encuentran identificados con un estado. Cada programa transforma el estado de la Forma Única para indicar que el movimiento ha sido procesado. A continuación un diagrama que muestra los estados que se utilizan en el proceso.

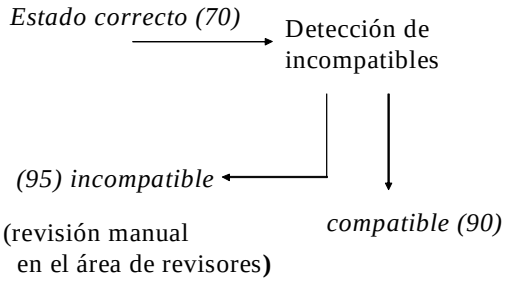


Figura 5.2 Estados en el subproceso de incompatibles.

El programa cuenta el número de registros, vacía la tabla TREFINCOM que conserva la referencia de incompatible, imprime el mensaje de los registros que ingresan al proceso.

Verifica y detecta los diferentes casos de incompatibilidad de movimientos y nombramientos, grabando en la tabla temporal TREFINCOM (contiene las referencias de los movimientos incompatibles), el número de empleado y tipo de incompatibilidad.

Se realizan las siguientes tareas:

Lee los movimientos de Forma Única con estado SDO_CORRECTO, en el caso de encontrar algún movimiento incompatible, inserta en la tabla TREFINCOM el empleado y la referencia, lo marca como movimiento incompatible para cambiar el estado de la Forma Única. Realiza lo siguiente para detectar los tipos de incompatibilidad:

- Lee y verifica, por cada empleado, si tiene movimiento retenido en alguna quincena anterior con estado INCOMPATIBLE.
- Verifica si el movimiento es de personal funcionario y además único pago.
- Graba en la tabla de referencias el número de empleado y tipo de incompatibilidad.
- Verifica que el empleado sea funcionario o de honorarios y tenga un único pago.
- Verifica que el movimiento pertenezca a la dependencia 442.
- Verifica que el empleado sea funcionario o de con baja total.
- Se obtiene la situación final del empleado excluyendo únicos pagos.

  - El total de horas es mayor a 40 para el personal de Asignatura.
  - El total de horas es mayor a 12 para los ayudantes de profesor.
  - El total de horas es mayor a 48 para los demás tipos de empleado.
  - El movimiento es de Funcionario o afecta un nombramiento de Funcionario.
  - El movimiento es de Honorarios.

Posteriormente se genera el reporte, de acuerdo a un formato establecido.

El siguiente proceso es el banco de horas, para lo cual haremos un programa llamado cnban.

El módulo está organizado en una secuencia de varios pasos en un sólo programa, prepara las tablas temporales que se requieren, transforma el archivo plano a delimitado, produce información de salida y evalúa el estado de salida de los programas.

Los procesos del banco de horas se ejecutan cuando se termino la validación y no van a ingresar más formas únicas.

Los estados que maneja esta parte, se muestran a continuación:

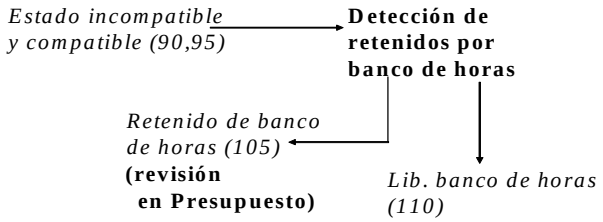


Figura 5.3 Estados en el subproceso de banco de horas.

Primero cuenta el número de registros que entran al proceso de banco de horas con estado compatible e incompatible.

La Dirección General de Presupuesto entrega a Nómina el archivo correspondiente al presupuesto disponible para horas de asignatura, es un archivo plano. Se le da formato y se guarda en un archivo plano llamado TBANCOHORA.dat que posteriormente es insertado en la tabla.

Todos los registros que no corresponden al proceso de banco de horas son cambiados al estado liberado banco de horas para continuar el proceso.

Carga la información de Banco de Horas autorizado a través del archivo TBANCOHORA.dat en la tabla TBANCOHORA.

Actualiza el total de horas ejercidas y disponibles en la tabla de banco de horas con base a los nombramientos de asignatura activos de la tabla TNOMBRAMIENTOS de la siguiente manera:

- Selecciona, suma y agrupa por: función, programa, subprograma, dependencia, subdependencia, partida, y categoría el total de horas de los nombramientos de asignatura, y por cada uno de los registros

agrupados actualiza el total de horas ejercidas y disponibles en la tabla de TBANCOHORA, siempre y cuando la función, programa, subprograma, dependencia, subdependencia, partida y categoría sean idénticas.

- Cuando el código programático no exista se manda un mensaje, con los datos correspondientes y se inserta el código faltante a la tabla de TBANCOHORA.

- Actualiza las columnas horas disponibles y las horas disponibles anteriores con la columna de horas asignadas en el caso de que la columna de horas ejercidas sea igual a cero, quiere decir que ningún nombramiento de nómina tiene ese código programático.

- Se verifica que los movimientos de la Forma Única tengan código programático válido, en caso de no cumplir se imprime el mensaje en un archivo de salida para que se verifique o se cambie el código.

Valida la suficiencia presupuestal de horas únicamente de los movimientos de personal de asignatura, actualiza de acuerdo al tipo de movimiento de la Forma Única el banco de horas de la siguiente manera:

Lee los movimientos de Formas Únicas de personal de Asignatura y con estado igual a CORTE NOMINA.

- Por cada Forma Única, se compara y selecciona todo el código programático y categoría en la tabla TBANCOHORA, el registro correspondiente con el total de horas autorizadas, ejercidas y disponibles para verificar la suficiencia de horas y actualizar el total de horas de acuerdo al tipo de movimiento en la tabla TBANCOHORA.

- Se investiga si el movimiento de alta es una promoción de nombramiento del empleado, para lo cual se buscan los movimientos de baja del empleado con el mismo código programático pero diferente categoría y partida. Son promociones aquellas que cambian de partida de 117 a 111, los que cambian de categoría D4100 a D4200 y D1200 a D1300.

- Si cumple con las condiciones de la promoción se actualiza la tabla para aumentar las columnas de horas ejercidas y horas asignadas y se actualiza el registro como promoción.

- Para las altas, se restan las horas a las horas disponibles.

- Si las horas disponibles son menores de cero se cambia el estado a retenido banco de horas.
- Se actualiza el registro.
- Para la baja se verifica que tenga el registro de alta. Si cumple con las condiciones de la promoción se actualiza la tabla y se disminuyen las horas ejercidas y asignadas, se marca como promoción.

Para las Formas Únicas que son únicos pagos se realiza lo siguiente:

- Se obtienen los empleados con movimientos de único pago.
- Calcula el sueldo retroactivo y se obtiene el total de la diferencia de sueldo.
- Actualiza el estado de los movimientos de único pago.
- Obtiene el último sueldo de la categoría.
- Con la diferencia de sueldo entre el último sueldo de la categoría se obtienen las horas, después las horas son divididas entre los 12 meses de año para obtener el sueldo equivalente.
- Se redondea el resultado.

Si el dígito es mayor a cero se actualizan las horas de únicos pagos con la opción de único pago y alta.

Ahora continuando con el proceso, veremos el relacionado con las altas, para lo cual se va a crear un programa llamado caltas. Este programa actualiza el estado de la Forma Única en TMOVFUND a 120, además de generar en la tabla de TALTAS las tarjetas y ponerles también el estado 120, ya que el proceso de diferencias retroactivas ya no actualiza estados en TMOVFUND sino en TALTAS.

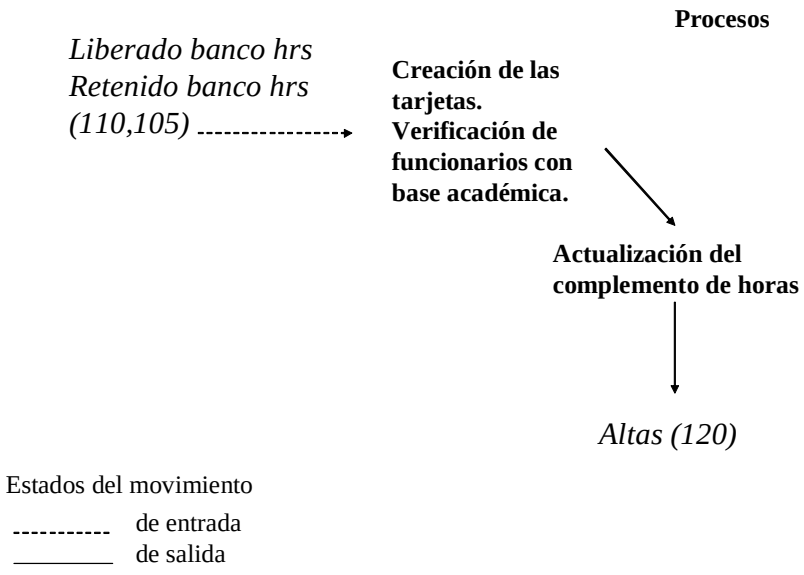


Figura 5.4 Estados en subproceso de altas.

- Se carga el tabulador.
- Antes que nada borramos de TALTAS las tarjetas que ya se hayan creado, haciendo un cruce con TMOVFUND y tomando los movimientos que están en estado 110 y 105.
- Seleccionar RFC, número de empleado y tipo de movimiento de TMOVFUND y TEMPNOMINA , de los movimientos que se encuentran en estado 90 y 95, ordenados por RFC y tipo de movimiento.

En el caso de las altas, baja y licencias sin sueldo:

- Se seleccionan todos los campos de TMOVFUND, se elige únicamente bajas o licencias sin sueldo y el orden de proceso es ascendente. Si es alta, seleccionamos todas las altas y el orden de proceso es descendente.[*]
  - Si el empleado es de carrera, base, confianza o funcionario con categoría CF, se actualizan las horas.
  - Se marca el movimiento para identificarlo, si es suspendido tendrá un valor de 2 y si es movimiento horizontal será 3.
  - Si es Z06, CA, CM, CD, RC o CH.

[*] Ya se explicó en el módulo de validación lo que es el orden de proceso.

- ✓ Si es A2, A3 ó C6, entonces buscamos en TNOMBRAPREVIO si va a tener nombramientos de carrera o asignatura.
- ✓ Si es C7 ó C4, entonces buscamos en TNOMBRAMIENTOS si tiene nombramientos de carrera o asignatura.

❖ Si tiene movimientos de carrera o asignatura, entonces lo marcamos como un funcionario con base académica.

❖ Si el movimiento es C7 ó C4, se cambia la fecha de fin por la fecha de inicio del periodo.

❖ Si la fecha de fin es cero asignamos la variable iFecLim con la fecha de inicio del periodo.

❖ Si la fecha fin de la Forma Única es diferente de cero, mayor que la fecha de inicio del periodo y mayor que la fecha de fin del periodo, la fecha límite va a ser igual que la fecha de inicio del periodo

❖ Si la fecha límite de la Forma Única diferente de cero, es menor que la fecha de fin del periodo, entonces la fecha límite es igual a la fecha fin de la Forma Única.

❖ Si la fecha fin de la Forma Única es diferente de cero y el tipo de movimiento es una licencia sin sueldo entonces la fecha límite se actualiza con la fecha de inicio de periodo.

❖ De acuerdo con la categoría, se van a crear las tarjetas diferentes de cero:

- ✓ Si la categoría es A04, A05, A08, A12, AA02, ET16, ET17, ET19, ET29:
  - ◆ Si es el nivel 40, 41, 42 o 43, se obtiene la diferencia de días entre la fecha de inicio de la Forma Única y la fecha límite asignada anteriormente. Se agrega un movimiento en TALTAS con el sueldo que tiene la Forma Única.
  - ◆ Si no, se buscan los sueldos retroactivos y se generan los movimientos correspondientes.
- ✓ Si tenemos la categoría P01, P14, P64, P80, P82 ó P90
  - ◆ Si es una P64, baja y el nivel es 40, 41, 42 ó 43, se obtiene la diferencia de días entre la fecha de inicio de la Forma Única y la fecha límite. Se inserta la tarjeta con el sueldo de la Forma Única.
  - ◆ Si no, se obtienen los sueldos retroactivos correspondientes y se insertan los movimientos en TALTAS.
- ✓ Si la categoría es M01, M03, M21, M22, M23 ó M24, se buscan los aumentos que le corresponden y se generan los movimientos en TALTAS.
- ✓ En el caso de la AH:
  - ◆ Si es de la partida 186 se buscan los sueldos retroactivos de acuerdo a los porcentajes de aumento.

- Si no, obtenemos la diferencia de días entre la fecha de inicio de la Forma Única y la fecha límite, si es un Único Pago, se le suma un día más y se inserta el registro correspondiente.

✓ Para las categorías M02, H01, H02, DH01, PH56, PH57 ó PH64

- Se obtiene la diferencia de días entre la fecha de inicio de Forma Única y la fecha límite, si es único pago se le agrega un día más. y se inserta el movimiento.

✓ Si es una CP01, se generan los movimientos retroactivos con el aumento que le correspondan al tipo de empleado.

✓ Si la categoría es una CF93, entonces obtenemos los días de diferencia entre la fecha de inicio de Forma Única y la fecha límite y se inserta el registro correspondiente.

✓ Para una CP02, se generan los movimientos retroactivos pero con el porcentaje establecido.

✓ Si la categoría es M04, M05, M06, M07, M08, M09, M10, M11, M12, M13, M14, M15, M25, M26, M27, M28, Z02, Z03, Z04, Z05, Z01, Z06, Z09, D48, Z07, DH41, DH42, EH41 ó EH42 se generan los movimientos retroactivos de acuerdo a las características de cálculo de cada una de ellas.

✓ Para la categoría D21

- Si el nivel es 42, cambiamos temporalmente la categoría por una D4200, se obtienen los sueldos retroactivos con el tipo de empleado de asignatura, al terminar de insertar los movimientos actualiza nuevamente la categoría original.
- Si no es ese nivel, entonces se obtienen los movimientos retroactivos de manera normal.

✓ Si la categoría es CA, CH, RC o CV

- Si es funcionario con base académica se realiza el ajuste de asignación y se obtienen los movimientos retroactivos.
- Si no, se obtienen los movimientos retroactivos consultando el tabulador de la categoría.

✓ Si es una CM o CD

- Si es funcionario con base académica y además se debe ajustar, entonces se realiza el ajuste de asignación y se obtienen los registros retroactivos.
- De otra manera obtenemos los movimientos retroactivos del tabulador.

✓ Para cualquier otra categoría se obtienen los movimientos retroactivos del tabulador.

❖ Ahora obtenemos la diferencia de días entre la fecha de inicio y la fecha fin de la Forma Única.

- ❖ Si es único pago se le agrega un día más.
- ❖ Se agrega el movimiento retroactivo cero, el cual tiene las fechas originales de la Forma Única.
- ❖ Se actualiza la Forma Única al estado 120 (ALTAS).

Se actualizan los funcionarios con base académica.

- Para actualizar el complemento de horas, se obtiene el número de empleado de las tarjetas que se encuentran en TALTAS en estado 120, diferentes de cero y que sean categorías D41, D42, EH41, EH42, DH41 ó DH42.

➢ Se investiga en TNOMBRAMIENTOS si el empleado ya tenía derecho al complemento de horas y se guardan las horas que tiene.

➢ Se seleccionan todos los datos de TALTAS donde las categorías sean de asignatura, no sean único pago y sean tarjetas cero, del empleado en proceso.

- ❖ Si el tipo de movimiento es una alta
  - ✓ Se suman a las horas que teníamos de TNOMBRAMIENTOS
  - ✓ Si el número de horas es mayor o igual a 15 horas y además ya tenía complemento, el nuevo complemento de horas es igual a la multiplicación del número de horas que se van a dar de alta por 2.5
  - ✓ Si las horas son mayores o iguales a 15 pero no tenía complemento de horas, el complemento va a ser igual a la multiplicación de las horas por 2.5 y vamos a considerar un nuevo empleado con complemento de horas.
  - ✓ Si las horas son menores de 15 y no tenía complemento, entonces el empleado es marcado sin complemento de horas.
- ❖ Si el tipo de movimiento es una baja o licencia sin sueldo.
  - ✓ A las horas que tiene en TNOMBRAMIENTOS le restamos las horas que se le van a dar de baja.
  - ✓ Si el número de horas es mayor o igual a 15 horas y ya tenía complemento, el nuevo complemento es igual al número de horas que se van a dar de baja por 2.5.
  - ✓ Si el número de horas es mayor o igual a 15 horas y no tenía complemento, entonces el nuevo complemento es igual a las horas iniciales por 2.5, y se marca que el empleado va a tener complemento.
- ❖ Si el complemento es mayor a cero, se actualiza el campo correspondiente en la tabla de TALTAS y en las tarjetas diferentes de cero, se verifica que cumpla con el folio, periodo, nodo y serial.

Una vez que se tienen los sueldos retroactivos divididos por periodos en la tabla TALTAS, llevamos a cabo el proceso de cálculo de conceptos retroactivos con el programa cncon.

Como en este proceso solamente manejamos las tarjetas de la tabla de TALTAS, los estados que se presentan a continuación son sobre dicha tabla.

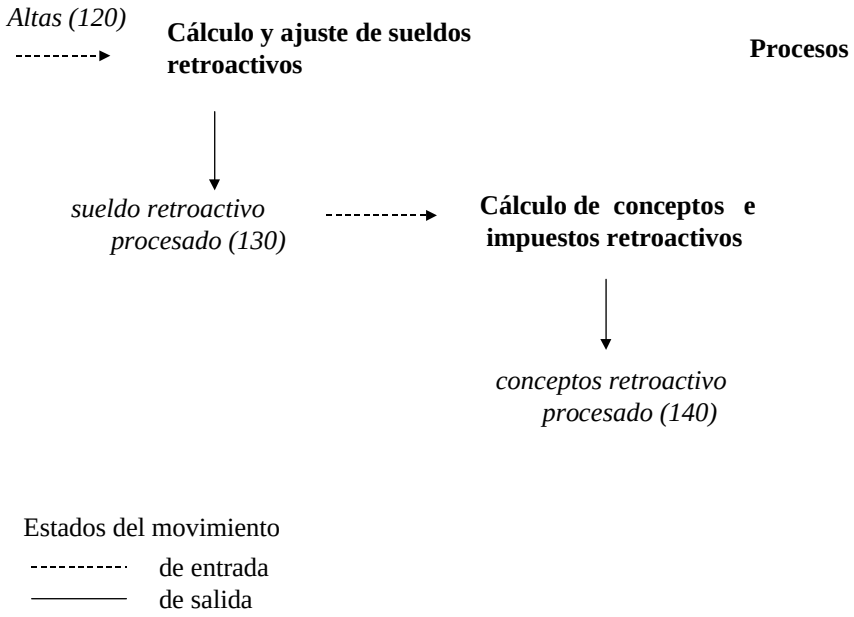


Figura 5.5 Estados en subproceso de cálculo de conceptos.

Este proceso se dividió por pasos, los cuales se van a presentar en un menú de opciones que es el que sigue:

1 = Calcular Sueldos Retroactivos

2 = Calcular Conceptos Retroactivos

3 = Revisar los funcionarios con base académica.

4 = Todo el proceso

5 = Salir

- Obtenemos la fecha de inicio y fin de periodo, así como el periodo en proceso; la fecha del día para ponerla como fecha de modificación, obtenemos el usuario de la base de datos, la ultima fecha del año anterior, la primera fecha del año anterior, el año anterior, año actual, fecha de inicio del año actual.

- Obtenemos la fecha de la gratificación, última fecha de gratificación y el periodo.
- Los campos de la fecha de la prima de vacación anterior y actual se asignan directamente con un valor, y cada que se pague una nueva prima vacacional se tienen que actualizar estos valores en el programa. Lo mismo sucede con la fecha fin de los días de julio.
- Determinamos si es una quincena par o non. También se investiga si el año es bisiesto.
- Seleccionamos el mes de la fecha de inicio de periodo. Si es menor que el mes 7, entonces es sueldo de julio, sino es sueldo de gratificación.
- Se llena una tabla para guardar los valores para el cálculo de los conceptos de fondo de pensión, ISSSTE y sueldo tope. Se obtiene el porcentaje.
- Se cargan los parámetros para el cálculo del impuesto.
- Si la opción elegida es la uno o la cuatro.

  - Se abre el archivo rep.suel.
  - Borra todos los datos de la tabla de TDIFERENCIA de los empleados que se vayan a procesar en ese momento.
  - Se realiza el cálculo de los sueldos retroactivos.
  - Se hace el ajuste entre sueldos y asueldos.

  Si la opción elegida es la dos o la cuatro.

  - Se borra de TDIFERENCIA todos los conceptos calculados de los empleados que se van a procesar, esto más que nada como medida de seguridad en un reproceso[2] para no generar registros duplicados. Los conceptos que se calculan en esta parte del proceso y que debemos borrar son los siguientes: diferencias de fondo de pensión, servicio médico, seguro de grupo, SAR, SAR negativo, asueldo anterior, asueldo pride, antigüedad administrativa, antigüedad docente, días de julio, prima vacacional, gratificación, pride director, pride (93, 94, 96, 97, 98, 99 y 00), paipa, vale de despensa, efectivale, antigüedad académica administrativa, recati, impuesto, crédito al salario, ispt de los días de julio y diferencia de ispt de la

[2] El reproceso se da cuando ya se procesaron todos los empleados y hay que corregir alguno de ellos, durante el análisis se determinó que es necesario tener esta posibilidad.

prima vacacional.

- Se abre el archivo rep.conc.
- Se seleccionan los registros únicos de TALTAS de todas las tarjetas que se encuentran en estado 130.
  - Se actualizan los datos del empleado que acabamos de obtener como son: antigüedad única, antigüedad docente, antigüedad administrativa, antigüedad de montaña, el dato de sí el empleado fue liquidado, el importe del SAR y el valor total de las percepciones anteriores.
  - Si tiene algún dato en el campo de último periodo de pago, obtenemos la fecha de inicio de ese periodo, sino, se pone en cero esta variable.
  - Se realizan los cálculos retroactivos.
  - Si las diferencias de sueldo son mayores de cero, entonces se calcula el impuesto retroactivo.
  - Se actualizan todas las tarjetas al siguiente estado (140)

- Si la opción es tres o cuatro.
  - Se carga el tabulador.
  - Seleccionamos a todos los funcionarios con base académica.
  - Investigamos la fecha de inicio del movimiento.
  - Obtenemos la suma total de sus sueldos sin incluir las zonas geográficas y complementarios.
  - Investigamos si tiene una diferencia de único pago.
  - Obtenemos el sueldo tabular de la categoría de funcionario.
  - Si la suma total de sueldos es menor o igual que el sueldo tabular, la fecha de inicio de funcionario es menor o igual que la fecha de inicio del movimiento o tiene diferencias de único pago, entonces se borran todas las diferencias calculadas, con excepción de las diferencias de pride y complemento de horas.
  - Si la suma de sueldos totales es mayor que la del sueldo tabular se imprimen los datos del empleado y el mensaje de que rebasa el tabulador.
  - Si la fecha de inicio de funcionario es mayor que la del movimiento, se imprime un mensaje notificándolo, así como los datos del empleado.

Posteriormente, y ya que tenemos todos los cálculos retroactivos, procedemos a la afectación de nombramientos para lo cual haremos el programa cnpro.c.

Los movimientos de Forma Única producen una serie de cambios en datos asociados con el empleado, dependiendo del tipo de movimiento y del tipo de empleado. Estas actualizaciones son las que se harán en este programa:

- Nombramientos del empleado.
- Datos generales del empleado (lugar de pago, días trabajados en el año, estado actual en TEMPNOMINA, banco del SAR, etc.)
- Historia de movimientos.
- Pagos y descuentos de la quincena.

El módulo está organizado en una secuencia de seis funciones principales, las cuales son invocadas por un programa integrador, produce información de salida y evalúa el estado de salida de las funciones.

Para facilitar la identificación de los movimientos y controlar su procesamiento, seguimos manteniendo el estado de la Forma Única. Cada programa lee los movimientos de Forma Única identificados con un estado determinado y al terminar el proceso, actualiza la Forma Única con el estado siguiente para indicar que el movimiento ha sido procesado.

A continuación se presenta un diagrama que muestra los estados de la Forma Única que se utilizan en el módulo. Hay que recordar que el módulo de cálculo de conceptos no actualizó estados en TMOVFUND, así que los vamos a retomar a partir del estado 120 (ALTAS).

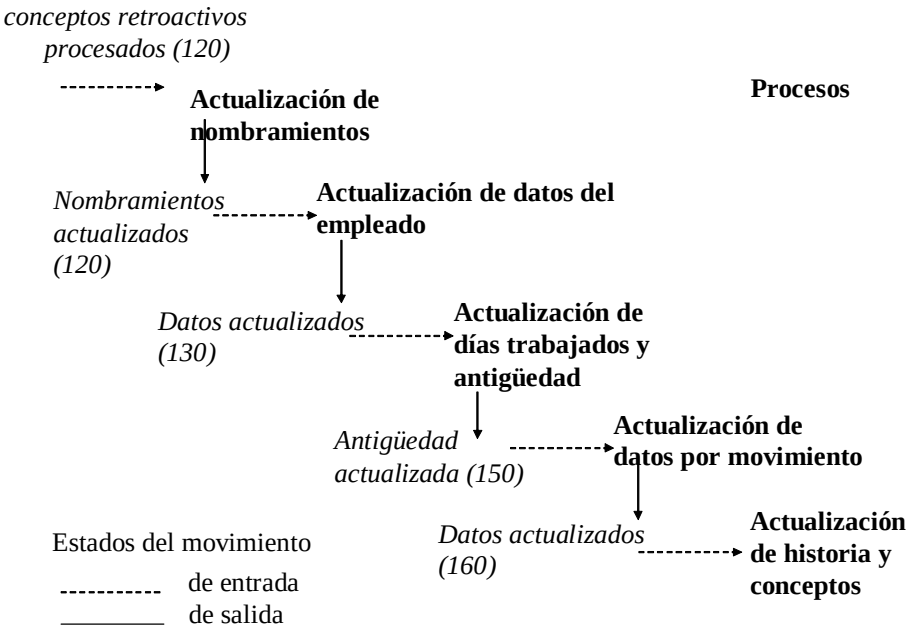


Figura 5.6 Estados en subproceso de afectación de nombramientos.

El programa hará lo siguiente:

- Se carga el tabulador.
- Se abre el archivo cnpro.out, se obtienen las fechas del periodo, la fecha actual para ponerla como referencia de las actualizaciones y el usuario.
- El primer proceso es la actualización de nombramientos para lo cual tenemos que borrar la tabla de TNOMBRAMIENTOS haciendo un cruce con los empleados que tienen Forma Única en estado 120 en TMOVFUND.
- Seleccionamos todos los datos que se encuentran en TNOMBRAPREVIO
  - Si la categoría es D12, D41, D42, DH41, DH42, EH41 ó EH42 entonces:
    - Asignamos las horas del nombramiento a una variable y las multiplicamos por 10.
    - Después convertimos a número entero las horas y sacamos el residuo de la división entre 10, y el dato obtenido nos va a decir si tenemos medias horas.
    - Si el valor obtenido es igual a 2, 5 ó 7 entonces:
      - ✓ Se busca el sueldo tabular multiplicado por el número de horas.
      - ✓ Se le aplica una función que trunca los centavos y éste es el sueldo que se grabara en la tabla de nombramientos.
  - Si la categoría es CA, CM, CH, RC o CD, entonces:
    - Se redondea el sueldo a centavos
  - Si la partida es 155, 171 o 177 y las horas totales son 16, al sueldo se le truncan los centavos.
  - Se inserta en TNOMBRAMIENTOS los datos que tiene la tabla TNOMBRAPREVIO.
- Se actualiza los datos por empleado, es decir, de acuerdo a los movimientos hay datos personales del empleado que deben cambiar.
- Actualizamos los días trabajados en el año, además de las variables de las primas vacacionales y días de ajuste.
- Se actualiza la antigüedad.
- Actualizamos la historia de movimientos.

- Se buscan todos los empleados en TEMPNOMINA que tengan estado intermedio (20 o 24, son los que están en la lista negra [3]).
  - Buscamos nombramientos de estos empleados en TNOMBRAMIENTOS.
  - Si tienen nombramiento, entonces se imprime el mensaje "ESTADO UNICO PAGO CON NOMBRAMIENTOS" y el número de empleado.
- Se actualizan todas las tarjetas de la tabla TALTAS, al estado 200, de aquellas que están en 140 y además son del periodo de proceso.
- Se copian las diferencias que se calcularon y que se encuentran en la tabla TDIFERENCIA a la tabla TMOVIMIENTOS.

Todos estos fueron los procesos generales que se realizan en el subsistema de Forma Única.

5. 2 CONSIDERACIONES GENERALES.

Al crear un programa nuevo debemos escribir el código correspondiente para abrir las conexiones a la base de datos de Nómina. El administrador se encarga del mantenimiento de las bases de datos que son generales para la Nómina, pero nuestro subsistema se debe encargar de mantener las tablas propias del proceso.

Todos los cambios y actualizaciones se hacen con el usuario del subsistema, para tener un control de quien realiza las modificaciones.

En los movimientos de asignatura se multiplicará el sueldo tabular por las horas para compararlo con el que trae la Forma Única, si no es correcto se modificará.

Solamente se modifican por programa los sueldos especiales o calculados, como podrían ser los sueldos ajustados, zonas geográficas o complementarios, pero para las Formas Únicas que deben tener sueldo tabular y no es correcto, se mandará un mensaje de error.

Se considerará para el cálculo de la diferencia de ISSSTE y fondo de pensión retroactividad de 5 años. Y para los sueldos retroactivos se va a considerar una retroactividad máxima de 10 años; los movimientos que rebasen este límite se mandarán al área correspondiente para su revisión.

[3] En la lista negra se encuentran los empleados que tuvieron algún problema con la Universidad, y que no se desea que vuelvan a ser recontratados, aunque tenemos excepciones.

6. HERRAMIENTAS DE DESARROLLO.

6.1 EL LENGUAJE DE PROGRAMACIÓN C.

El lenguaje C fue diseñado en los años sesenta por Dennis Ritchie, en los Laboratorios Bell, con el propósito de ser el lenguaje del sistema operativo UNIX. Se desarrolló a partir de dos lenguajes de programación de sistemas, BCPL y B.

La primera versión que se tuvo del lenguaje, se describió en el libro "The C programming Language".

Con el paso del tiempo ya había muchas versiones del lenguaje, se modificó para aprovechar mejor sus características. Durante este periodo de tiempo, numerosos fabricantes introducen mejoras en el lenguaje, las cuales son recogidas por un comité de estandarización ANSI y establecen las especificaciones de lo que se conoce hoy en día como 'ANSI C'.

El estándar esta basado en el manual de referencia original. El lenguaje ha cambiado relativamente poco, uno de los propósitos del estándar fue asegurar que la mayoría de los programas existentes pudieran permanecer vigentes, que sean compatibles o, al menos, que los compiladores pudieran producir mensajes de advertencia acerca del nuevo comportamiento.

Con el estándar, el cambio más importante es una nueva sintaxis para declarar y definir funciones. Una declaración de función ahora puede incluir una descripción de los argumentos de la función; la sintaxis de la definición cambia para coincidir. Esta información extra permite que los compiladores detecten más fácilmente los errores causados por argumentos que no coinciden.

Una segunda contribución significativa al estándar, es la definición de una biblioteca, la cual contiene funciones para tener acceso al sistema operativo (por ejemplo, leer de archivos y escribir en ellos), entrada y salida con formato, asignación de memoria, manipulación de cadenas y otras actividades semejantes. Una colección de encabezadores

(headers) estándar proporcionan un acceso uniforme a las declaraciones de funciones y tipos de datos. Los programas que utilizan esta biblioteca están asegurando un comportamiento compatible.

C es un lenguaje de propósito general que ha sido estrechamente asociado con el sistema UNIX en donde fue desarrollado, puesto que tanto el sistema como los programas que corren en él están escritos en lenguaje C. Sin embargo no está ligado a ningún sistema operativo ni a ninguna máquina, aunque se le llama "lenguaje de programación de sistemas" debido a su utilidad para escribir compiladores y sistemas operativos. Por lo tanto, nos podemos dar cuenta que se puede utilizar para cualquier disciplina.

6.1.1 CARACTERÍSTICAS Y ALCANCES DEL LENGUAJE.

En la jerarquía de lenguajes se encuentra entre el Pascal y el Ensamblador, ya que su finalidad es ser un lenguaje de alto nivel como el primero con la versatilidad del ensamblador, el cual es de bajo nivel. Un lenguaje de alto nivel, es aquel que tiene una interfaz amigable para el usuario, y cuyo set de instrucciones es mas entendible para el ser humano, cosa que no sucede con un lenguaje de bajo nivel, el cual está mas orientado a la máquina, sus instrucciones se dirigen más a las direcciones de memoria. Además, como C se diseño junto con Unix, esta orientado para trabajar bajo este sistema.

Para desarrollarlo se siguieron un conjunto de directivas, tales como:

- El compilador debe ser lo más pequeño y eficiente posible.
- Un conjunto reducido de sentencias, es decir, palabras reservadas.
- No existe anidamiento de procedimientos.
- La entrada/salida no se debe considerar parte del lenguaje en sí, ya que para realizar este tipo de operaciones se cuenta con funciones almacenadas en una librería, lo mismo sucede para cualquier otro tipo de instrucciones complejas.
- Para escribir un programa se debe poder usar poco texto. Para lograr esto se reduce el número de palabras claves.

Fue así como se obtuvo un compilador con un poderoso juego de instrucciones, que permite aumentar la productividad de los programadores. Aún así, el C es un lenguaje rápido de aprender, que deriva en compiladores sencillos de diseñar, robustos, y que generan objetos pequeños y eficientes.

Una de las principales ventajas de C, es su gran portabilidad, es decir la facilidad de escribir programas que puedan correr sin cambios en una variedad de máquinas, ya que deja en las librerías el manejo de las funciones dependientes de la máquina, sin que por esto tengamos restringido el acceso a esta. Aunque C coincide con las capacidades de muchas computadoras, es independiente de cualquier arquitectura.

Estas y otras características lo hacen adecuado para la programación en diversas áreas, tales como:

- Programación de sistemas
- Estructuras de datos y sistemas de bases de datos
- Aplicaciones científicas
- Software gráfico
- Análisis numérico

En la figura 6.1 se muestra el esquema básico del proceso de compilación de programas, módulos y creación de bibliotecas en C. Los rectángulos nos indican cuales son los programas involucrados y los cilindros tipo fichero con la extensión correspondiente.

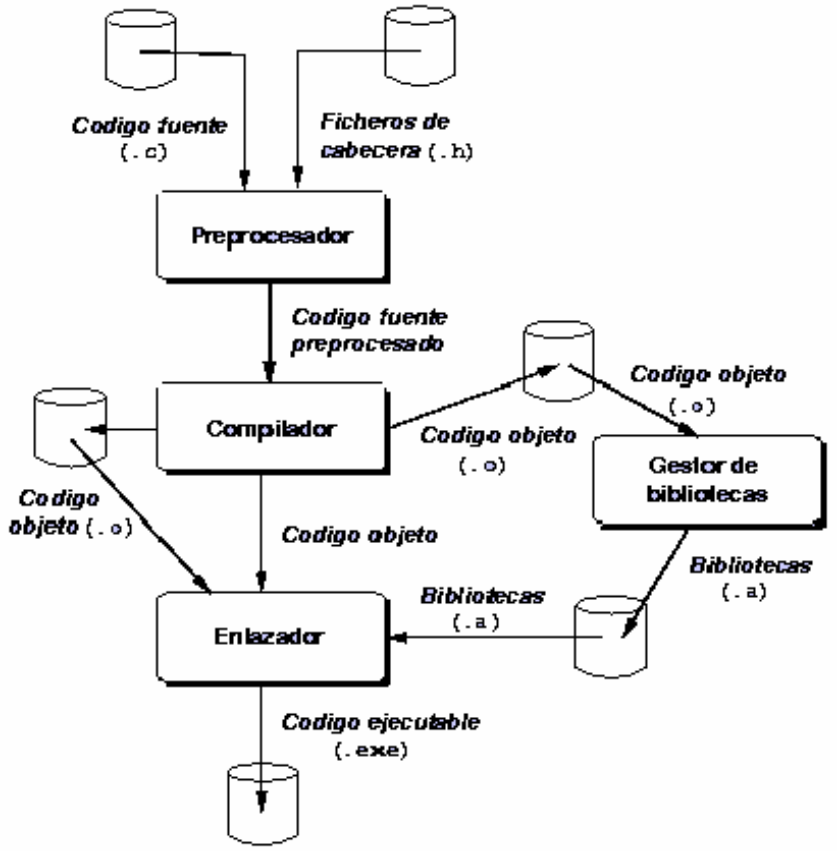


Figura 6.1. Esquema del proceso de compilación (generación de programas ejecutables) en C.

Este esquema puede servirnos para enumerar las tareas de programación habituales. La tarea más común es la generación de un programa ejecutable; como su nombre indica, es un fichero que contiene código directamente ejecutable por el procesador. Éste puede construirse de diversas formas:

1. A partir de un fichero con código fuente.

2. Enlazando ficheros con código objeto.

3. Enlazando el fichero de código objeto con una biblioteca.

Las dos últimas requieren, que previamente se hayan construido los ficheros objeto (opción 2) y los ficheros de biblioteca (opción 3) Como se puede comprobar en el esquema anterior, la creación de éstos también está contemplada en el esquema. Así, es posible generar únicamente ficheros objeto para:

a) Enlazarlos con otros para generar un ejecutable. La primera alternativa exige que uno de los módulos objeto que se van a enlazar contenga la función main() Esta forma de construir ejecutables es muy común y usualmente los módulos objeto se borran, ya que no tiene

interés su permanencia. Si realmente es así, se opta por la otra alternativa.

b) Que se incorporen los módulos objeto a una biblioteca.

Una biblioteca, en la terminología de C, será una *colección de módulos objeto*. Entre ellos existirá alguna relación, que debe entenderse en un sentido amplio: si dos módulos objeto están en la misma biblioteca, contendrán funciones que trabajen sobre un mismo tema (por ejemplo, funciones de procesamiento de cadenas de caracteres)

Si el objetivo final es la creación de un ejecutable, en última instancia uno o varios módulos objeto de una biblioteca se enlazarán con un módulo objeto que contenga la función main().

El preprocesador acepta como entrada código fuente y se encarga de:

- Eliminar los comentarios.
- Interpretar y procesar las directivas de preprocesamiento, precedidas siempre por el símbolo #.

Dos de las directivas más comúnmente empleadas en C, son #include y #define:

- #include: Sustituye la línea por el contenido del fichero especificado. Por ejemplo, #include <stdio.h> incluye el fichero stdio.h, que contiene declaraciones de tipos y funciones de entrada/salida de la biblioteca estándar de C. La inclusión implica que *todo* el contenido del fichero incluido sustituye a la línea #include.
- #define: Define una constante (identificador) simbólico. Sustituye las apariciones del identificador por el valor especificado, salvo si el identificador se encuentra dentro de una constante de cadena de caracteres (entre comillas) Por ejemplo, #define MAX_SIZE 100 establece el valor de la constante simbólica MAX_SIZE a 100. En el programa se utilizará la constante simbólica y el preprocesador sustituye cada aparición de MAX_SIZE por el literal 100 (de tipo entero)

En cualquier caso, retomando el esquema mostrado en la Figura 6.1 destacaremos que el preprocesador no genera un fichero de salida en el sentido de que no se guarda el código fuente preprocesado). El código resultante se pasa directamente al compilador. Así, aunque formalmente pueden distinguirse las fases de preprocesado y compilación, en la práctica el preprocesado se considera como la primera fase de la compilación. Gráficamente, a continuación mostramos el esquema de la fase de compilación.

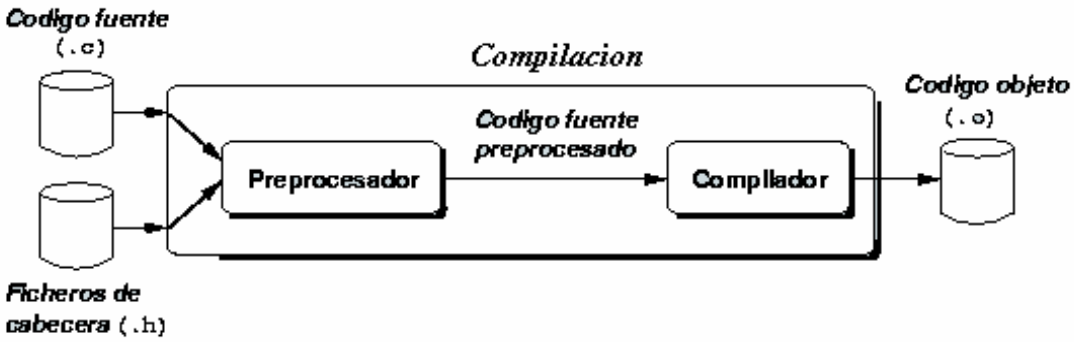


Figura 6.2 Compilación

En la figura 6.2, el enlazador (del inglés, *linker*) resuelve las referencias a objetos externos que se encuentran en un fichero fuente. Estas referencias son a objetos que se encuentran en otros módulos compilados, ya sea en forma de ficheros objeto o incorporados en alguna biblioteca (del inglés, *library*).

No es usual llamar al enlazador explícitamente sino que éste es invocado convenientemente por el compilador cc, cuando se llama a cc y uno de los ficheros involucrados es un módulo objeto o un fichero de biblioteca, el último paso que realiza cc es llamar al enlazador ld. Así, vemos que cc es más que un compilador (formalmente hablando) ya que al llamar a cc se procesa el código fuente, se compila, e incluso se enlaza.

Para generar un fichero ejecutable se requiere que uno de los módulos objeto que se están enlazando contenga la función main(), por ejemplo: supongamos un fichero llamado principal.c que contiene la función main(), en este fichero se hace uso de la función que calcula el seno, cuyo prototipo (declarado en math.h) es el siguiente: double sin(double x). Se quiere generar un fichero ejecutable a partir de este módulo fuente.

```
#include <stdio.h>
#include <math.h>
   ...
int main (void)
{
      ...
   double s, x;
      ...
   scanf ("%f", &x);
      ...
   s = sin (x);
   printf ("El seno de %3.2f es %3.2f\n", x, s);
      ...
}
```

Cuando el compilador procesa este código no puede generar código ejecutable ya que no puede completar la referencia a la función sin(): lo único que encuentra en math.h es el prototipo o declaración de la función sin(), así, el código ejecutable asociado a este cálculo no puede completarse y queda "pendiente". Puede comprobarse empíricamente. La ejecución del comando:

```
cc -o principal principal.c
```

Produce un error de enlace ya que el enlazador no encuentra el código asociado a la función sin(). Para que se genere el código ejecutable se debe indicar *explícitamente* que se enlace con la biblioteca matemática:

```
cc -o principal principal.c -lm
```

En el ejemplo, se puede observar además de sin() se ha utilizado la función printf() y ésta no se ha definido en el programa (sólo se ha introducido su declaración con la directiva include <stdio.h>), ni se ha especificado, a la hora del enlace, en qué biblioteca se encuentra. Esto se debe a que ciertas bibliotecas se enlazan automáticamente a la hora de crear ejecutables, y la función printf() se encuentra en una de ellas. Esta es la razón de que no se indique de forma explícita en la llamada a cc.

C es un lenguaje muy reducido, muchas de las posibilidades incorporadas en forma de funciones en otros lenguajes, no se incluyen en el repertorio de instrucciones de C; por ejemplo, el lenguaje no incluye ninguna facilidad de entrada/salida, manipulación de cadenas de caracteres, funciones matemáticas, gestión dinámica de memoria, etc.

Esto no significa que C sea un lenguaje pobre. Todas estas funciones se incorporan a través de un amplio conjunto de bibliotecas que *no* forman parte, hablando propiamente, del lenguaje de programación. No obstante, como indicamos anteriormente, algunas bibliotecas se enlazan automáticamente al generar un programa ejecutable, lo que induce al error de pensar que, por ejemplo, printf() es una función propia del lenguaje C. Otra cuestión es que se ha definido y estandarizado la llamada **biblioteca estándar de C** (en realidad, bibliotecas) de forma que cualquier compilador que quiera ser *compatible con ANSI C* debe asegurar que las funciones proporcionadas en esas bibliotecas se comportan de forma similar a como especifica el comité ANSI. Para efectos prácticos, las funciones de la biblioteca estándar pueden considerarse parte del lenguaje C.

Aún teniendo en cuenta estas consideraciones, el conjunto de bibliotecas disponible y las funciones incluidas en ellas pueden variar de un compilador a otro y el programador responsable deberá asegurarse que cuando usa una función, ésta forma parte de la biblioteca estándar: *este es el procedimiento más seguro para construir programas transportables entre diferentes plataformas y compiladores*.

Cualquier programador puede desarrollar sus propias bibliotecas de funciones y enriquecer de esta manera el lenguaje de una forma completamente estandarizada. En la figura 6.2 muestra el proceso de creación y uso de bibliotecas propias; se ilustra que una biblioteca es, en realidad, una "objetoteca", si se nos permite el término. De esta forma nos referimos a una biblioteca como a una *colección de módulos objeto*. Estos módulos objeto contendrán el código objeto correspondiente a variables, constantes y funciones que pueden usarse por otros módulos si se enlazan de forma adecuada.

El programa de GNU que se encarga de incorporar, eliminar y sustituir módulos objeto en una biblioteca es AR. En realidad, AR es un programa de aplicación más general que la de gestionar bibliotecas de módulos objeto pero no detallaremos esto.

C maneja tres tipos de datos fundamentales que son: caracteres, enteros y números de punto flotante de varios tamaños. Además tenemos tipos de datos derivados de los principales, los cuales tienen una jerarquía entre sí, estos tipos derivados se crean con apuntadores, arreglos, estructuras y uniones. Las expresiones se forman partiendo de operadores y operandos y cualquiera puede ser una proposición aún las asignaciones o llamadas a función. Los apuntadores nos dan una aritmética de direcciones independiente de la máquina.

También nos proporciona las construcciones principales, para el control de flujo que se requiere en los programas bien estructurados: agrupación de proposiciones, toma de decisiones (if - else), selección de un caso entre un conjunto de opciones (switch), iteración con condición de paro en la parte superior (while, for) o en la parte inferior (do), y terminación prematura de ciclos (break)

Las funciones pueden regresar valores de tipos básicos, estructuras, uniones o apuntadores. Cualquier función puede ser llamada recursivamente. Las funciones pueden estar guardadas en archivos fuentes separados y compilarse cada una de manera independiente del programa que las utilice.

El compilador (del inglés, *compiler*) analiza la sintaxis y la semántica del código fuente preprocesado y lo traduce, generando un fichero que contiene el código objeto. La extensión por defecto de estos ficheros es .o (GNU) ó .OBJ (Borland C)

Antes de continuar, debemos hacer una importante observación, en el proceso de compilación se interpreta el código fuente, salvo las referencias a objetos externos (funciones o variables) en el módulo de código que se está compilando. Estos objetos externos, especialmente funciones, se declaran en este módulo, pero se definen en otros. La declaración servirá al compilador para comprobar que las referencias externas son sintácticamente correctas.

Veamos con un ejemplo muy simple e ilustrativo de que esto que hemos comentado lo encontramos muy a menudo. Supongamos un fichero.c que contiene la función main(), en ella se usa una instrucción printf(), para comprobar si la sintaxis es correcta, el compilador necesita conocer el prototipo de esta función, de ahí que nuestro programa necesite incluir el módulo de cabecera stdio.h; sin embargo, el código "ejecutable" asociado a esta función se encuentra en otro fichero (una biblioteca del sistema), por lo que la traducción de nuestro programa se hará dejando el "hueco" correspondiente a la función printf(). Este hueco lo completará el enlazador a partir de una biblioteca del sistema, generando definitivamente un código ejecutable.

6.1.2 LIMITACIONES DEL LENGUAJE C EN COMPARACIÓN CON OTROS LENGUAJES DE PROGRAMACIÓN.

Dentro de las principales críticas que se le achacan podemos destacar:

- Lo poco estricto que es el lenguaje, con la comprobación de los tipos de datos, dejando esta tarea muchas veces en manos del programador.
- El no verificar automáticamente los límites de los vectores.
- La repetición que hace de símbolos en operadores diferentes (=,*,-) lo cual se conoce como sobrecarga de operadores. También algunos operadores tienen la precedencia equivocada.
- El no poder anidar funciones, con lo que se dificulta la estructuración y la abstracción de datos.
- La incertidumbre existente en el orden de evaluación de las listas de expresiones y parámetros.
- Algunos elementos de la sintaxis podrían ser mejores.
- La facilidad con la que los programas pueden llegar a ser crípticos. Por ejemplo: los siguientes son programas crípticos:

```
for(i=0;i<10;i++)
for (j=0;j<10;)
              *(*(x + i)+j++)=(y[i][j]>*(*(z + i)+j))?1:2;
```

```
for(;*a |*b; *p++ = (*a >*b)?*a++; *b++);
```

6.2 LOS SISTEMAS MANEJADORES DE BASES DE DATOS (DBMS).

La base de datos es el componente estructural más importante en el diseño de un sistema, ya que es el centro de integración de un sistema de información. En grandes organizaciones como la UNAM, muchos usuarios requieren tener acceso simultáneo a la información. Una base de datos consta de elementos de datos organizados en registros y archivos en forma tal que satisfagan los requerimientos de información de los usuarios. Los usuarios van a tener acceso a estos datos mediante un sistema de administración de datos.

Lo anterior se refiere al proceso de almacenar y recuperar los datos, el cual se divide en tres tareas básicas:

- Describir la organización real y la interrelación de los datos en una definición estándar de datos. Para lograr este objetivo debemos saber que los datos representan objetos físicos de la vida real. Estos objetos se denominan entidades, las cuales pueden ser un objeto tangible o intangible. Esta entidad tiene atributos que se desean registrar, los cuales son características propias de la entidad que van a tener un valor determinado.

- Almacenar físicamente los datos en un formato específico en algún medio de almacenamiento. Los atributos se almacenan para ser recuperados por los usuarios, el formato de los atributos incluye tamaño y organización de los datos. La organización se refiere al orden físico que van a tener las entidades en el medio físico, como serían los discos ópticos o magnéticos.

- Recuperar los datos de forma que proporcionen información válida a los usuarios del sistema. Una vez que se definen los atributos y la organización física de una entidad, los usuarios pueden recuperar la información.

Inicialmente las bases de datos eran archivos de datos, pero eran muy difíciles de manejar por las siguientes desventajas:

Redundancia de la información, es decir, se tenía que volver a introducir o solicitar información que ya se había solicitado previamente.

Inconsistencia, existen archivos que manejan la misma información pero en algunos de ellos dicha información está actualizada y en otros no.

No eran concurrentes, no era posible que varios usuarios tuvieran acceso a la misma información al mismo tiempo.

Entonces surge el concepto *base de datos*, la cual se define como un conjunto exhaustivo y no redundante de datos estructurados, organizados de forma independiente a su utilización o implantación en máquina, accesibles en tiempos reales y compatibles con usuarios concurrentes y sus respectivas necesidades (peticiones) de información.

Las consideraciones que se deben hacer para la selección de la base de datos son el modelo y la topología de las bases de datos. El modelo incluye la estructura del almacenamiento de los datos, y describe cómo serán presentados los datos al usuario o programador, así como las relaciones entre los elementos de una base de datos, mientras que la topología incluye la configuración física tanto de datos como del procesamiento.

Las bases de datos se clasifican, con respecto a su modelo, en:

Modelo Jerárquico. Este modelo fue el primero en aparecer y se caracteriza por organizar la información a través de una estructura de árboles, en la que las relaciones entre instancias o registros se expresan mediante una jerarquía. Dicha jerarquía distribuye y ordena los datos mediante un recorrido en preorden.

Esta estructura cuenta con una gran cantidad de inconvenientes conocidos, por lo que su utilización en la actualidad es prácticamente nula. La flexibilidad y falta de estructuración de este modelo permite crear registros cuyos campos sean variables en número y tamaño; además, cuando la información almacenada en la base de datos es muy grande, se convierte en inmanejable. En cuanto a limitaciones lógicas cabe destacar que este modelo sólo soporta relaciones del tipo uno a muchos (ya que las de tipo muchos a muchos requieren un inaceptable nivel de redundancia).

Modelo de Red. Este modelo fue concebido como una ampliación del modelo jerárquico, cuya finalidad era solucionar las deficiencias lógicas de este último. Al igual que el anterior, también se emplea un árbol como estructura base, pero con la diferencia de que un mismo hijo puede tener diferentes padres, con lo que es posible representar relaciones muchos a muchos sin redundancia aparente. En algunas versiones modernas de éste modelo, encontramos la aparición de registros enlaces para establecer relaciones muchos a muchos. Como inconveniente presenta la complejidad que alcanza el entramado de enlaces

entre las instancias cuando se almacenan gran cantidad de datos, así como, la hostilidad de los lenguajes de programación y control de estas bases de datos.

Modelo Relacional. Este modelo distribuye los datos en tablas bidimensionales, llamadas relaciones, dónde las columnas recogen los diferentes atributos o campos y las filas almacenan las diferentes instancias u ocurrencias (registros). También se establecen varios tipos de dependencias entre las tablas (interrelaciones), y según la naturaleza de éstas, se implementan mediante atributos "clave extranjera" o mediante tablas relacionales.

Como reglas básicas de formación se establecen las siguientes: no se permiten ocurrencias duplicadas en una tabla, hay un único valor para un atributo dado de una determinada ocurrencia, todos los atributos que no forman parte de la clave dependen sólo de esta, todas los valores en un atributo que sea clave extranjera deberán aparecer en la tabla donde dicho atributo es clave principal.

De acuerdo a su topología, las bases de datos se clasifican en: centralizadas, de *red de área Local* (*Local Área Network*, *LAN*) de computadoras personales y cliente / servidor.

Un sistema de bases de datos es un conjunto de *datos*, los cuales son valores registrados físicamente en la base de datos, *hardware* que corresponde al almacenamiento físico, *software* que son los programas que manejan la base de datos y los *usuarios* que corresponden a los usuarios finales, el programador o desarrollador y el DBA que es la persona encargada de la creación y administración de la base de datos.

En un sistema de base de datos, una desventaja importante es la seguridad y la integridad, si no se tiene un buen control; sin embargo, las ventajas son mayores ya que se reduce la redundancia, se evita la inconsistencia, es posible aplicar restricciones de seguridad, se tiene un mayor control de la concurrencia.

Un sistema manejador de bases de datos (Database Management System, DBMS) está formado por:

- La base de datos.
- El software para manipular (almacenar, recuperar y modificar) los datos.

El principal objetivo de un DBMS, es crear un ambiente en el que sea posible guardar y recuperar información de la base de datos en forma eficiente.

En cuanto a los manejadores de bases de datos, existen varios en la industria, que podemos usar para el desarrollo de un sistema. Cada uno de ellos tiene características especiales que los diferencian de los demás, aunque su objetivo final es el mismo. Sin embargo para elegir el mejor se toma en cuenta el precio y las características.

Para el sistema de Nómina se uso el DBMS de Sybase Versión System 10, aunque actualmente ya se migró al System 11, debido a su manejo y facilidad de uso que otros manejadores no presentan; como por ejemplo, los respaldos en línea, que para nuestro caso es muy importante poderlos hacer sin afectar a los múltiples usuarios, hay manejadores que no permiten hacer estos respaldos; también a su facilidad para manejar grandes volúmenes de información con rapidez. Otro de los aspectos importantes fue el precio y las posibilidades de financiamiento con las que contaba la Universidad y las facilidades que dió la empresa.

6.2.1 PRINCIPALES DBMS EXISTENTES EN LA INDUSTRIA DE LAS BASES DE DATOS.

Se explicará brevemente algunos DBMS existentes en la industria y posteriormente el DBMS de Sybase.

IBM DB2. Es el sistema de bases de datos relacional de IBM y es uno de los DBMS relacionales más antiguos del mercado. Se utiliza principalmente en sistemas de computadoras mainframe como AS/400 y RS/6000. Este DBMS proporciona muchas características avanzadas y se utiliza, principalmente, para soluciones de bases de datos a gran escala. Los servidores DB2 soportan cualquier comunicación basadas en los protocolos APPC, IPX/SPX, NetBios, TCP/IP, Pipes con nombres. Las plataformas que admite el producto son en principio OS/2, Windows NT y UNIX. Esta última versión del software del servidor, ha sido adaptado para funcionar también con OS/2 WARP, HP-UX, AIX, SCO-UNIX UE7, Windows 98, NT y Solaris.

INFORMIX. La presencia de Informix se deja sentir en una cantidad de plataformas, concretamente Windows NT y UNIX. Una de las características de este sistema, es un completo conjunto de herramientas gráficas que permiten asistir tanto en los procesos de instalación como administración del servidor, sin necesidad de que el administrador tenga grandes conocimientos. Existe la posibilidad de gestionar múltiples bases de datos remotas de una única y centralizada consola donde se muestran gráficamente tanto la base de datos como los objetos que contiene (tablas índices, procedimientos, etc.). También podrá establecerse un calendario de tareas a ejecutar en cualquier objeto o grupo de objetos. Adicionalmente Informix proporciona tablas que forman el SMI (interfaz de monitorización del sistema). Este nuevo tipo de bases de datos, orientadas a

objetos, mejorará en gran medida las soluciones que puedan ofrecer los desarrolladores a sus clientes.

ORACLE. Para muchos, el líder en el mercado de las bases de datos es el DBMS Oracle de Oracle Corporation. Es el DBMS relacional de uso más extendido y ofrece varias características que facilitan su uso.

Oracle Entreprise Manager proporciona la posibilidad de gestionar múltiples grupos de trabajo remoto desde una única y centralizada consola. El administrador verá una representación gráfica de todos los objetos importantes. También podrá establecer un calendario de tareas a ejecutar en un objeto o grupo de objetos, para lo cual se verá ayudado por el Inteligent Agent, que obedece las órdenes ejecutadas desde la consola central.

Otro cometido de este agente es detectar problemas que puedan ir surgiendo e informar de los mismos. En lo referente a Internet, las aplicaciones web pueden acceder a los datos almacenados en la base de datos de Oracle, así como presentar documentos HTML generados dinámicamente a partir de un modelo de una consulta.

Microsoft SQL Server. La versión de SQL Server de Microsoft ha sido mejorada para su uso en aplicaciones reales de bases de datos y se incluye en el paquete BackOffice. Este DBMS es, en cierta forma, menos costoso que el resto de los productos de bases de datos y se utiliza más para el desarrollo de aplicaciones pequeñas. Se podría decir que en cierta medida pretende ser el servidor de bases de datos genérico para Windows. No tanto por que la causa de desarrollo sea la misma, ni siquiera porque el SQL Server, a diferencia de otros servidores sólo trabaja bajo Windows, sino porque Microsoft promete integración con todos los productos suyos (por ejemplo MsOffice 2000, ya que Access 2000 traerá consigo un nuevo MSDE-DATA-Engine, como alternativa al existente y compatible con SQL Server). También será posible llamar a SQL Server desde Microsoft Access.

La escalabilidad es total. No es necesario decir que el producto puede funcionar en un Servidor NT multiprocesador de elevadas prestaciones, pero si lo es decir, que también puede funcionar en un portátil con Windows 95/98. Las características de SQL Server son impresionantes. Tenemos soporte de transacciones OLPT, una maquinaria de búsqueda de textos completos que permite localizar información a lo largo de una tabla (lo que hace las delicias, si el proyecto a considerar es para Internet). Posibilidad de ejecutar consultas en paralelo, así como homogéneas y distribuidas. Bloqueos dinámicos a nivel de filas, optimizador de consultas, estadísticas automáticas, unicode nativo, replicación avanzada, replicación dinámica de datos y un largo etcétera. Pero lo mejor de todo es la sencillez de comprensión y navegación entre procesos, así como lo intuitivo de la herramienta.

6.2.2 EL DBMS DE SYBASE INC.

Inicialmente el DBMS de Sybase se llamaba SQL Server y era soportado por múltiples plataformas como UNIX, Windows y OS/2. Posteriormente, Sybase cambia el nombre de su manejador por el de Adaptive Server Enterprise (ASE), agregándole así nuevas características que lo colocan como uno de los DBMS más utilizados en la industria.

Actualmente se distribuye principalmente en dos versiones: el Adaptive Server Anywhere (ASA) y Adaptive Server Enterprise (ASE), este último es el adecuado para ambientes corporativos ya que presenta una gran robustez, mientras que el ASA es más adecuado a computadoras personales y portátiles en las que se desea un DBMS sólido y robusto como el de Sybase.

Con la evolución del producto se ha ido mejorando la interfaz sustancialmente y en la versión NT cada vez son más las pantallas gráficas y los asistentes.

El rendimiento de Sybase, al igual que otros productos evaluados, conforme crece el hardware del servidor ya que también admite configuración especifica para multiprocesadores simétricos (SMP). La idea es que la consulta, ordenación de datos y otros procesos puedan ser ejecutados en paralelo. Según los datos ofrecidos por Sybase las consultas distribuidas en paralelo, pueden proporcionar respuestas hasta 15 veces más rápida (en situaciones óptimas).

Backup Server se utiliza para las copias de seguridad de datos. Monitor Server o Historial Server se encargan de capturar, mostrar y evaluar datos de rendimiento, así como ajustar el Component Integration Service, se encargan de extender la funcionalidad presentando una vista de los datos de forma uniforme en las aplicaciones clientes.

Server Config es la herramienta de configuración y SQL remote proporciona replicación basada en mensajes entre una base de datos central y un conjunto de bases de datos en ordenadores de sobremesa en la misma ubicación u otra diferente. Por la parte del cliente existen más componentes especialmente útiles, si no tenemos ninguna aplicación propia que manipule los datos.

Pero tal vez la herramienta más útil para el administrador sea SQL Modeler, con la cual es posible diseñar estructuras de bases de datos de forma óptima, incluyendo eventos y procedimientos integrados. Esto puede ser muy interesante porque también es posible realizar procesos de ingeniería inversa interpretando la estructura de las bases de datos de otros sistemas (Access, DB2, SQL Server...). Una vez interpretada se construye la equivalente de Sybase, que se modifica con las herramientas del entorno.

Una de las posibilidades que ofrece el producto es la replicación y se logra gracias al componente SQL remote. Además ofrece un Replication Server, cuando los servidores de bases de datos son un pequeño número y el tiempo característico de actualización es en orden de segundos. Un ejemplo de uso característico de SQL Remote, unido al Adaptative Server, es la creación de una Intranet dónde se puedan añadir puestos móviles.

Las plataformas que admite el producto son WinNT, Digital Unix, DCR System, SCO-UnixWare, Silicon Graphics, HPUX, Aix, Solaris y Linux. Se soportan los siguientes protocolos: APPC, IPX/SPX, NetBios, TCP/IP, Pipes con nombres.

Finalmente, podemos decir que la seguridad de integridad de los datos es uno de los objetivos de Sybase, con una certificación ISO 9001 DBMS. Incluye duplicación / espejo de disco, volcado de seguridad y restauración de alta velocidad, transacciones en línea con *roll-back* automático.

Entre las principales características que presenta, están las siguientes:

- Ambiente SQL estándar.
- Arquitectura multiprocesos y abierta.
- El SQL Server Monitor que ayuda a visualizar y optimizar el performance del sistema.

- Detección y resolución automática de bloqueos del sistema (deadlocks).
- Procedimientos almacenados locales y remotos Servidor a Servidor.
- Soporte al cómputo distribuido a través del Sybase SQL Server Manager.
- Escalabilidad a través de aplicaciones y datos que se pueden portar fácilmente vía el SQL Anywhere.
- Mantiene la integridad y seguridad de los datos a través de respaldos y duplicidad de bases de datos, así como de la codificación de passwords.
- Fácil instalación y mantenimiento.

7. LA ARQUITECTURA CLIENTE – SERVIDOR.

La arquitectura cliente-servidor permite al usuario en una máquina, llamada el cliente, requerir algún tipo de servicio de una máquina a la que está unido, llamada el servidor, mediante una red LAN (Red de Área Local) o una WAN (Red de Área Mundial). Estos servicios pueden ser peticiones de datos de una base de datos, de información contenida en archivos o los archivos en sí mismos o peticiones de imprimir datos en una impresora asociada. Aunque clientes y servidores suelen verse como máquinas separadas, pueden ser dos áreas separadas en la misma máquina. Por tanto, una única máquina Unix puede ser al mismo tiempo cliente y servidor. También es posible tener el *cliente* ejecutándose en un sistema operativo y el *servidor* en otro distinto.

7.1 COMUNICACIONES EN AMBIENTES DE RED E INTRODUCCIÓN A LA ARQUITECTURA CLIENTE – SERVIDOR.

El modelo Cliente-Servidor es el modelo estándar de ejecución de aplicaciones en una red. Este modelo aplicado a un sistema de computadoras conectadas en red se relaciona con tres componentes: el cliente, el servidor y la red. Un servidor es un proceso que se está ejecutando en un nodo de la red (terminal o periférico conectado a la red) y que gestiona el acceso a un determinado recurso. Un cliente es un proceso que se ejecuta en el mismo o en diferentes nodos y que realiza peticiones de servicio al servidor. Las peticiones se originan por la necesidad de tener acceso al recurso que gestiona el servidor, como observaremos en la figura 7.1:

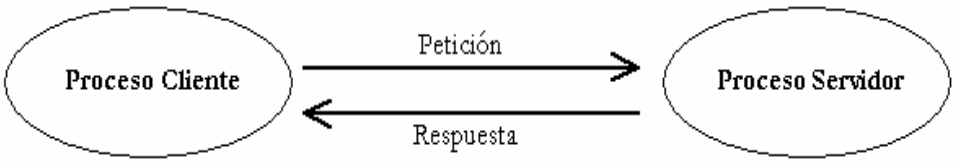


Figura 7.1 Cliente – servidor.

Hay varios tipos comunes de máquinas clientes en entornos cliente-servidor. Uno de los clientes más populares es una computadora personal basada en Intel que ejecuta aplicaciones de DOS en un entorno Windows. Otra cliente popular es una terminal X; de hecho, el sistema X Windows es un modelo cliente-servidor clásico. Hay también clientes Unix que ejecutan sistemas operativos como UnixWare. Un servidor que pide cosas a otro servidor es un cliente de la máquina a la que está pidiendo. Sin considerar el tipo de cliente que se esté usando en una red cliente-servidor, se realizan al menos una de las funciones básicas descritas aquí como funciones del cliente.

Funciones del cliente

Los clientes en una red cliente-servidor son las máquinas o procesos que piden información, recursos y servicios a un servidor. Estas peticiones pueden ser cosas como proporcionar datos de una base de datos, aplicaciones, partes de archivos o archivos completos a la máquina cliente. Los datos, aplicaciones o archivos pueden residir en un servidor y ser simplemente accedidos por el cliente o pueden ser copiados o movidos físicamente a la máquina cliente. Esta disposición permite a la máquina cliente ser relativamente pequeña. Para cada tipo de entorno de cliente, hay habitualmente software específico (y a veces hardware) en el cliente, con algún software y hardware análogo en el servidor.

Los servidores pueden ser sistemas operativos diferentes como Windows NT, Windows 95, OS/2, Unix. Unix es popular porque como sistema operativo de servidores puede ser utilizado en muchos tipos de configuraciones sobre máquinas servidor, además como servidores de archivos y servidores de impresión.

Funciones generales de un servidor

Los servidores en una red cliente-servidor son los procesos que proporcionan información recursos y servicios a los clientes de la red. Cuando un cliente pide un recurso, por ejemplo, un archivo, datos de una base de datos, acceso a aplicaciones remotas o impresión centralizada, el servidor proporciona estos recursos al cliente. Los procesos del servidor pueden residir en una máquina que también actúa como cliente de otro servidor. Además de proporcionar este tipo de recursos, un servidor puede dar acceso a otras redes, actuando como un servidor de comunicaciones que conecta a otros servidores o mainframes o mini computadoras que actúan como hosts de la red.

También puede permitir enviar faxes o correo electrónico desde un cliente en una red a un cliente en otra red. Puede actuar como servidor de seguridad, como servidor de gestión de la red, como servidor multimedia, como servidor de directorios o de acceso

El sistema de entrada/salida de unix sigue los pasos abrir-leer-escribir-cerrar. Antes de que un proceso de usuario pueda realizar operaciones de entrada/salida, debe hacer una llamada a abrir (open), para indicar y obtener permisos para su uso, del archivo o dispositivo que quiere utilizar. Una vez que el objeto está abierto, el proceso de usuario realiza una o varias llamadas a Leer (read) y Escribir (write), para conseguir leer y escribir datos respectivamente. Leer toma datos desde el objeto y los transfiere al proceso de usuario, mientras que escribir transfiere datos desde el proceso de usuario al objeto. Una vez que todos estos intercambios de información estén concluidos, el proceso de usuario llamará a cerrar (close), para informar al sistema operativo que ha finalizado la utilización del objeto que antes había abierto.

Las características principales de Unix son las siguientes:

Multitareas

Esta palabra describe la habilidad de ejecutar, aparentemente al mismo tiempo, numerosos programas sin obstaculizar la ejecución de cada uno por separado. Esto se conoce como multitareas preferentes, porque cada programa tiene garantizada la posibilidad de ejecutarse, esto es, cada programa no se ejecuta hasta que el sistema operativo lo aparta para permitir que otros programas lo hagan. Otros sistemas operativos no soportan multitareas preferentes, únicamente la llamada multitareas cooperativa, bajo la cual los programas se ejecutan hasta que ellos mismos permiten la ejecución de otro programa o no tienen otra cosa que hacer durante este periodo.

Es fácil apreciar los beneficios de tener capacidades multitareas preferentes. Además de reducir los tiempos muertos, la flexibilidad de no tener que cerrar las ventanas de aplicaciones antes de abrir y trabajar en otras es mucho más conveniente.

Linux y otros sistemas de multitareas preferentes realizan este tipo de procesamiento mediante el monitoreo, tanto de los procesos que están en espera de ejecución como de los que se están ejecutando. Entonces, el sistema programa cada proceso para que tenga la misma oportunidad de acceso al microprocesador. El resultado es que las aplicaciones abiertas parecen correr al mismo tiempo. La capacidad de asignar tiempo a las aplicaciones que están en ejecución, nos permite mayor velocidad de procesamiento.

Multiusuario

El concepto de que numerosos usuarios pudieran acceder aplicaciones o el potencial de procesamiento de una sola PC, era un sueño desde hace unos años. La capacidad para asignar tiempo del microprocesador a numerosas aplicaciones simultáneas se prestó como consecuencia a servir a numerosas personas al mismo tiempo, cada una ejecutando una o más

aplicaciones. Una particularidad de esta característica, es que más de una persona puede trabajar en la misma versión de la misma aplicación de manera simultánea, desde las mismas terminales o desde terminales separadas. Esto no debe confundirse con numerosos usuarios que actualizan un archivo a un tiempo, particularidad que es potencialmente desconcertante y peligrosa, a la vez que indeseable.

Shells programables

La comunicación con el sistema UNIX se da mediante un programa de control llamado SHELL. Este es un lenguaje de control, un intérprete, y un lenguaje de programación, cuyas características lo hacen sumamente flexible para las tareas de un centro de cómputo. Como lenguaje de programación abarca los siguientes aspectos:

- Ofrece las estructuras de control normales: secuenciación, iteración condicional, selección y otras.
- Paso de parámetros.
- Sustitución textual de variables y Cadenas.
- Comunicación bidireccional entre órdenes de shell.

El shell permite modificar en forma dinámica las características con que se ejecutan los programas en UNIX:

Las entradas y salidas pueden ser redireccionadas o redirigidas hacia archivos, procesos y dispositivos. Es posible interconectar procesos entre sí.

Diferentes usuarios pueden "ver", versiones distintas del sistema operativo, debido a la capacidad del shell para configurar diversos ambientes de ejecución. Por ejemplo, se puede hacer que un usuario inicie directamente en su sección, ejecute un programa en particular y salga automáticamente del sistema al terminar de usarlo.

Existen aún mas características que merecen mencionarse sobre el sistema operativo con el que se trabajó en este sistema, pero para el desarrollo de la misma no es necesario describir tan detalladamente. Estas características son las siguientes:

- Independencia de dispositivos bajo Unix.
- Comunicaciones y capacidades de la red.
- Portabilidad de sistemas abiertos.

En Unix, un proceso tiene un conjunto de descriptores de entrada/salida desde donde Leer y por donde Escribir. Estos descriptores pueden estar referidos a archivos, dispositivos, o canales de comunicaciones

(sockets). El ciclo de vida de un descriptor, aplicado a un canal de comunicación (socket), está determinado por tres fases:

Creación y apertura del socket.

Lectura y escritura, recepción y envío de datos por el socket.

Destrucción, cierre del socket.

Los sockets son puntos finales de enlaces de comunicaciones entre procesos. Los procesos los tratan como descriptores de archivos, de forma que se pueden intercambiar datos con otros procesos que se están transmitiendo y recibiendo a través de sockets. El tipo de sockets describe la forma en la que se transfiere información a través de ese socket.

Sockets Stream (TCP, Transmission Control Protocol)

Son un servicio orientado a conexión, donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.

El protocolo de comunicaciones con streams, es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

Sockets Datagrama (UDP, User Datagram Protocol)

Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la confiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

Sockets Raw

Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.

Dominios de comunicaciones:

El mecanismo de sockets está diseñado para ser todo lo genérico posible. El socket por sí mismo no contiene información suficiente para describir la comunicación entre procesos. Los sockets operan dentro de

dominios de comunicación, dentro de estos dominios se define sí los dos procesos que se comunican, se encuentran en el mismo sistema o en sistemas diferentes y cómo pueden ser direccionados.

Dominio Unix

Bajo Unix, hay dos dominios, uno para comunicaciones internas al sistema y otro para comunicaciones entre sistemas.

Las comunicaciones intrasistema (comunicaciones entre dos procesos en el mismo sistema) ocurren en el dominio unix. Se permiten tanto los sockets stream como los datagrama. Los sockets de dominio unix bajo Solaris 2.x se implementan sobre TLI (Transport Level Interface). En el dominio unix no se permiten sockets de tipo raw.

7.1.1 LA ARQUITECTURA CLIENTE – SERVIDOR.

La base de datos consiste en sólo un archivo de base de datos. En este caso, el único servidor requerido es uno de archivos. En esta arquitectura el motor de bases de datos puede ejecutarse en la máquina cliente o en el servidor, muy estrechamente unida a la aplicación cliente. Para Access, Visual Basic y FoxPro este motor se conoce como JET y se compone de varios DLLs que residen en la máquina cliente.

Por ahora nos concentraremos en los servidores de bases de datos.

Ventajas de usar la arquitectura cliente–servidor:

- Cuando un servidor de bases de datos procesa una consulta, la respuesta a esta petición dependerá de la máquina servidora, no del cliente.
- El proceso del servidor activo, devuelve sólo la información solicitada en la red (contrario a los grandes bloques de entrada y salida), de tal modo, que el tráfico en la red es sustancialmente reducido. Esto permite crear aplicaciones que acceden grandes cantidades de datos en un módem, por ejemplo, el cual tiene un ancho de banda mucho menor.
- Un proceso de servidor activo puede asegurar más eficazmente la integridad de los datos.

La arquitectura de dos capas típica en cliente-servidor.

La mayoría de las aplicaciones cliente-servidor funcionan bajo una arquitectura de dos capas en lenguajes de cuarta generación. Estas aplicaciones son bifurcadas en las siguientes capas: el llamado front-end (la interfaz del usuario, llamadas a SQL, aplicación de escritorio, etcétera) y el llamado back-end (servidor de Bases de datos SQL, sistema operativo multitareas, etc.).

El proceso front-end se desarrolla en algún lenguaje de 4ª generación (4GL) como Visual Basic. Se llama front-end dado que es la capa en donde el usuario interactúa con su PC. El proceso back-end es el servidor de bases de datos como SQL Server ú Oracle. Se llama así, porque típicamente reside en un servidor central en un entorno controlado.

La seguridad puede ser establecida por el back-end del servidor de bases de datos o por la aplicación que sirve de front-end. Cada una tiene sus limitaciones: el primero consiste en dar privilegios a los objetos de la base de datos y a los usuarios. Sin embargo, las corporaciones no requieren sólo asegurar cuales datos pueden ser actualizados o accedidos, sino cómo. En cuanto al segundo punto, que es el más usado, aunque el usuario puede acceder a la base de datos con su identificación, tiene dos problemas:

- Dado que ninguno de los objetos en la base de datos es seguro, cualquier usuario puede tener acceso total a la misma con alguna otra herramienta de front-end (como Excel, Access, etc.).
- La implantación de la seguridad deberá ser desarrollada, probada y mantenida en absolutamente toda la red (no importa dónde se encuentren las estaciones cliente).

Aunque el entorno de dos capas provee grandiosas herramientas para el front-end y el back-end, el desarrollador se regresará a las herramientas de tercera generación (3GL) como C y COBOL para crear procesos en lotes.

La arquitectura de 3 capas

Una arquitectura de 3 capas (o de n capas), se define también como el modelo de servicios. Las bases de datos, las herramientas de desarrollo y los corporativos se están moviendo hacia esta arquitectura dadas las limitaciones de la de dos capas.

La capa adicional provee de una capa explícita para las reglas de los negocios que se sitúa entre lo que se ha llamado front-end y back-end. Esta capa intermedia encapsula el modelo de negocios (o "reglas de negocios") asociado con el sistema y lo separa de la presentación y el código de bases de datos.

En la documentación de Visual Basic, Microsoft llama adecuadamente a las aplicaciones cliente, "Servicios del usuario", dado que son éstas las que interactúan directamente con los usuarios finales.

Una de las capas se comunica con su padre, hijo o similar, lo que significa que puede hacer solicitudes y devolver respuestas a un proceso desde su propia capa, inmediatamente arriba de su capa o inmediatamente abajo de su capa. Normalmente, la única comunicación que nunca ocurre es la de una aplicación con el servicio de datos.

En una arquitectura tradicional, una capa puede comunicarse sólo con otra directamente arriba o abajo de ella. En este otro caso los servicios de usuarios, de negocios y de datos pueden comunicarse con ellos mismos. Este modelo se conoce como el modelo de servicios, dado que, lejos del comportamiento de un modelo de capas, cualquier servicio puede invocar a otro dentro de su capa.

Un particular servicio de usuario, de negocios o de datos se forma de componentes. Cada componente radica en el contexto de una simple capa y servicio, y cada capa contiene varios servicios creados con componentes.

Mientras un servicio es un concepto lógico, un componente describe un paquete físico de funcionalidad. De este modo, cada servicio puede describirse como un grupo lógico de componentes físicos.

Existen algunas reglas al respecto:

La capa de aplicaciones cliente, se compone de aplicaciones cliente (como un pedido o mantenimiento de productos), las cuales se crean a partir de componentes de aplicaciones cliente.

La capa del servidor de negocios se compone de servidores de negocios (como el proceso de órdenes y el manejo del almacén), la cual se crea a partir de componentes de aplicaciones de servidor de negocios.

La capa del servidor de datos se compone de servidores de datos (como órdenes y productos), que se crean a partir de componentes de servidores de datos.

Esto es más sencillo de usar que de explicar. La aplicación cliente puede llamarse de múltiples formas: servicio de usuario, cliente, aplicación, front-end, capa de presentación, GUI, etc. La función de la aplicación cliente es la de permitir al usuario una interfaz para los servicios de negocios. Una aplicación cliente bien diseñada permite que el usuario entienda los servicios de negocios como un todo y navegar eficientemente por estos servicios. Esta es la capa que se crea con lenguajes de 4ª generación como Visual Basic, así, como aplicaciones de escritorio como Excel.

El servicio o servicios de negocios crean la unión entre las aplicaciones del cliente y los servicios de datos. La función de esta capa lógica es primordialmente, la de hacer valer las políticas del

negocio y encapsular un modelo de los negocios así como exponer tal modelo a las aplicaciones cliente.

Finalmente los servicios de datos son aquellos cuyo manejo se lleva a cabo mediante sistemas manejadores de datos basados en SQL, como SQL Server y Oracle. Estos son los que manejan los datos, información y transacciones para los servidores de negocios. Así, estos servidores mantendrán inalterable sólo la integración de datos que manipulan.

Así, la arquitectura de tres capas depende del proceso interno de comunicación. Para que dos servicios se comuniquen, deberán hacerlo en el mismo lenguaje o protocolo. Esta comunicación se puede hacer con OLE, DDE, OpenDoc, CORBA, DB-LIB, WinSock, etc.

Contrario a la arquitectura de dos capas, las ventajas son:

- Mejor manipulación del sistema, servidores de negocios que pueden compartirse, los objetos y procesos serán seguros, los usuarios podrán crear sus propias aplicaciones cliente, entre otros. De esta manera, se pueden lograr sistemas distribuidos, que son físicamente localizados en varias máquinas dentro de varias locaciones. El grado en que pueda distribuirse una aplicación es directamente proporcional al grado en que lógicamente se ha particionado.

- Finalmente, como ya se había dicho, la arquitectura de 3 capas depende en gran parte de los procesos de comunicación. Un común denominador debe existir para permitir cualquier proceso. La comunicación deberá ser local (en las mismas máquinas) o remota (en máquinas distantes). Deberá ser transparente al proceso que envuelve sin importar si es local o remota.

La arquitectura de 3 capas es una guía, no un requerimiento. Este es un modelo lógico, no físico, que describe cómo se diseña la aplicación no cómo se despliega.

7.2 EL CLIENTE Y EL SERVIDOR DE APLICACIONES.

Debido a que se implementará la arquitectura de ambientes distribuidos en este sistema, se requiere considerar a los procesos cliente y servidor de aplicaciones. El proceso cliente de este sistema es el correspondiente a la interfaz gráfica de usuario (GUI) en la que inicialmente, introduce su login y password. Entonces el cliente se comunica a través de sockets con el servidor de aplicaciones para enviarle el login y el password del

usuario. Una vez que el servidor ha recibido estos datos, se comunica con el servidor de bases de datos para enviarle el login y el password del usuario, para que sea verificado en la tabla en la cual se encuentra esa información propia de cada usuario.

Una vez que el servidor ha verificado estos datos, envía al servidor de aplicaciones la respuesta, ya sea de aceptación o rechazo, y una vez que el servidor recibe dicha respuesta, la envía al cliente. Cuando se acepta la entrada del usuario, entonces se despliega el mapa del laboratorio con todas las computadoras. Para cada acción que el usuario desee realizar, el cliente se debe comunicar con el servidor de aplicaciones y éste con el servidor de bases de datos ya que la información referente al estado de todas las computadoras y a los apartados de los usuarios se encuentra en tablas de la base de datos.

8. DESARROLLO.

Para el desarrollo del subsistema de validación y proceso de Forma Única se creó un esquema estándar de desarrollo y una biblioteca de funciones estándar de uso general.

Los propósitos de los estándares fueron proporcionar un ambiente de desarrollo con el uso de herramientas estándar y facilitar el mantenimiento y la evolución de código de manera consistente.

Estos estándares de desarrollo sólo son aplicables en el sistema operativo unix, la base de datos usa el manejador de Sybase 11, la biblioteca de interfase es Sybase DB-Lybrary y el compilador de C es el SPARcompiler C 3.01.

Esquema General

Se creo un esquema general para la preparación de programas nuevos, para lo cual se hacen las siguientes tareas:

- Preparar el esqueleto estándar con un shell especial llamado **np**, más el nombre del programa y crea un archivo con extensión c, como el siguiente:

```
/*************************************************************************
/*  ESTE ES EL ENCABEZADO DE UN PROGRAMA */
   Nombre del programa :
   Nombre del proyecto :            Nomina UNAM.
   Descripción         :            Breve descripción de lo que realiza
                                    el programa.

   Archivos asociados  :            Nombres de los archivos con los que
                                    el programa tiene alguna relación.

   Referencias         :            Breve descripción de los documentos
                                    relacionados con el programa
                                    (miniespecificaciones, diagramas de
                                    flujo de datos, etc.).
   Comentarios         :

   Fecha    : DD MMM HI             Fecha de creación del programa.
*************************************************************************/
#include <stdio.h>                    /* Prototipos de E/S estandar        */
#include <stdlib.h>                   /* Prototipos de biblioteca estandar */
#include <signal.h>                   /* Prototipos de manejo de se#ales   */
#include <string.h>                   /* Prototipos de manejo de cadenas   */

#include <sybfront.h>                 /* Constantes para DB-Lib de Sybase  */
#include <sybdb.h>                    /* Estructuras y prots DB-Lib Sybase */
#include <syberror.h>                 /* Constantes para errores de Sybase */
```

Se utilizan los encabezados más comunes para Unix, los requeridos por DB-Lybrary y los relativos a la aplicación.

```
/*-----------------------------------------------------------------------*/
                          /*  INICIALIZACION   */
```

```
#include "aplstd.h"                    /* Constantes y prototipos estandar  */
#include "apldef.h"                    /* Constantes de la aplicacion       */
#include "aplfun.h"                    /* Constantes y prototipos aplicacion*/
#include "prgvar.h"                    /* Almacenamiento de columnas        */

/*-------------------------------------------------------------------------*/


main(int argc, char *argv[])
{

DBPROCESS       *stMain, *stQuery, *stUpdate;
RETCODE         rcMain;

/*-------------------------------------------------------------------------*/
dbsetversion(DBVERSION_100);
vGetPasswd(argc, argv);
DBSETLENCRYPT(stLog, TRUE);

stMain = dbopen(stLog, NULL);
stQuery = dbopen(stLog, NULL);
stUpdate = dbopen(stLog, NULL);

dbcmd(stMain, "SET TRANSACTION ISOLATION LEVEL 1");
dbcmd(stQuery, "SET TRANSACTION ISOLATION LEVEL 1");
dbcmd(stUpdate, "SET TRANSACTION ISOLATION LEVEL 1");
dbsqlexec(stMain);
dbsqlexec(stQuery);
dbsqlexec(stUpdate);
```

/* El punto de entrada principal recibirá como argumentos la clave del usuario y la contraseña para acceder el servidor de la base de datos.

Variables y conexiones. Enseguida se declaran las conexiones a la base de datos y las variables que manipulara el programa. Después se inicializa la conexión a la base de datos y se definen los niveles de aislamiento de cada conexión. */

```
/*-------------------------------------------------------------------------*/

                            /*  TEXTO PRINCIPAL  */


dbcmd(stMain, "SELECT columnas FROM tablas ");
dbfcmd(stMain, "WHERE condición ", variables);
dbsqlexec(stMain);

while ((rcMain = dbresults(stMain)) != NO_MORE_RESULTS) {
  if ((rcMain == SUCCEED) && (DBROWS(stMain) == SUCCEED)) {

    dbstruct(stMain, stvTab, sNumCol);

    while (dbnextrow(stMain) != NO_MORE_ROWS) {

      dbcmd(stQuery, "SELECT columnas FROM tablas ");
      dbfcmd(stQuery, "WHERE condición ", variables);
      dbsqlrun(stQuery);


      dbhandletran(stUpdate, BEGIN_TRAN);


      dbcmd(stUpdate, "UPDATE tabla SET columna FROM tabla ");
      dbfcmd(stUpdate, "WHERE condición ", variables);
      dbsqlrun(stUpdate);
```

```
      dbhandletran(stUpdate, END_TRAN);

    }
  }
}
```

/* A través de este segmento de código se realiza la lectura principal de una tabla, utilizando además, la función **dbstruct,** de la biblioteca estándar, que minimiza el código necesario para relacionar las columnas con las variables del programa.

```
/*----------------------------------------------------------------------*/

                                /*FINALIZACION  */


dbexit();                                       /* Cerrar conexion y salir */
exit(STDEXIT);

}

/************************************************************************
```

- Se prepara la definición de una vista de las columnas a manipular en el programa y la declaración de las variables que almacenarán las columnas, con el programa **cstruct**, escrito en DB-Library, el cual tiene la siguiente sintaxis; **cstruct <**usuario> <password> <tabla1> <tabla2>.<tablan><nombre del programa>

Esta instrucción da la siguiente salida:

```
$> cstruct mari mcg99  TMOVFUND nuevo.c

Procesando: TMOVFUND
include iNumFol (y/n) : y
include nPer (y/n) : y
include siCveNodo (y/n) : y
include siSerial (y/n) : y
include iNumEmp (y/n) : y
include tiCveTipMov (y/n) : y
include siNumCau (y/n) : y
include iNumPlaza (y/n) : y
include tiCveFunc (y/n) : y
include tiCveProg (y/n) : y
include siCveSubProg (y/n) : y
include siCveDep (y/n) : y
include tiCveSubDep (y/n) : n
include siCvePart (y/n) : y
include cConacyt (y/n) : y
include cCveTipEjer (y/n) : n
include cDigCon (y/n) : n
include cCveCateg (y/n) : y
include mSdo (y/n) : y
include cAfeNom (y/n) : y
include nHorTeo (y/n) : y
include nHorPra (y/n) : y
include sdFecIniFun (y/n) : y
include sdFecFinFun (y/n) : y
include cDefFun (y/n) : y
include cCoaCoc (y/n) : y
include siPagUni (y/n) : y
include siTipoEmp (y/n) : y
include vcDescMat (y/n) : y
include siStatus (y/n) : y
```

```
include siCveNodPag (y/n) : y
include iCveNodosTra (y/n) : y
include siTipProc (y/n) : y
include siStatusSip (y/n) : n
include cCatRef (y/n) : n
include sdFecIniOrig (y/n) : n
include sdFecFinOrig (y/n) : n
include mSdoPla (y/n) : n
        Total de columnas 30

LOS ARCHIVOS GENERADOS SON:  nuevo.c_infTMOVFUND.h

struct dbview stvTMOVFUND[] = {
        {CS_ILLEGAL_TYPE  },
        {SYBINT4,       "iNumFol",       &stcTMOVFUND.iNumFol},
        {SYBNUMERIC,    "nPer",          &stcTMOVFUND.nPer},
        {SYBINT2,       "siCveNodo",     &stcTMOVFUND.siCveNodo},
        {SYBINT2,       "siSerial",      &stcTMOVFUND.siSerial},
        {SYBINT4,       "iNumEmp",       &stcTMOVFUND.iNumEmp},
        {SYBINT1,       "tiCveTipMov",   &stcTMOVFUND.tiCveTipMov},
        {SYBINT2,       "siNumCau",      &stcTMOVFUND.siNumCau},
        {SYBINT4,       "iNumPlaza",     &stcTMOVFUND.iNumPlaza},
        {SYBINT1,       "tiCveFunc",     &stcTMOVFUND.tiCveFunc},
        {SYBINT1,       "tiCveProg",     &stcTMOVFUND.tiCveProg},
        {SYBINT2,       "siCveSubProg",  &stcTMOVFUND.siCveSubProg},
        {SYBINT2,       "siCveDep",      &stcTMOVFUND.siCveDep},
        {SYBINT2,       "siCvePart",     &stcTMOVFUND.siCvePart},
        {SYBCHAR,       "cConacyt",      stcTMOVFUND.cConacyt},
        {SYBCHAR,       "cCveCateg",     stcTMOVFUND.cCveCateg},
        {SYBFLT8,       "mSdo",          &stcTMOVFUND.mSdo},
        {SYBCHAR,       "cAfeNom",       stcTMOVFUND.cAfeNom},
        {SYBNUMERIC,    "nHorTeo",       &stcTMOVFUND.nHorTeo},
        {SYBNUMERIC,    "nHorPra",       &stcTMOVFUND.nHorPra},
        {SYBDATETIME,   "sdFecIniFun",   &stcTMOVFUND.sdFecIniFun},
        {SYBDATETIME,   "sdFecFinFun",   &stcTMOVFUND.sdFecFinFun},
        {SYBCHAR,       "cDefFun",       stcTMOVFUND.cDefFun},
        {SYBCHAR,       "cCoaCoc",       stcTMOVFUND.cCoaCoc},
        {SYBINT2,       "siPagUni",      &stcTMOVFUND.siPagUni},
        {SYBINT2,       "siTipoEmp",     &stcTMOVFUND.siTipoEmp},
        {SYBCHAR,       "vcDescMat",     stcTMOVFUND.vcDescMat},
        {SYBINT2,       "siStatus",      &stcTMOVFUND.siStatus},
        {SYBINT2,       "siCveNodPag",   &stcTMOVFUND.siCveNodPag},
        {SYBINT4,       "iCveNodosTra",  &stcTMOVFUND.iCveNodosTra},
        {SYBINT2,       "siTipProc",     &stcTMOVFUND.siTipProc},
};

EL SIGUIENTE ARCHIVO ES:  nuevo.c_preTMOVFUND.h

stTMOVFUND[2].stPrec.precision=6;       stTMOVFUND[2].stPrec.scale=0;
stTMOVFUND[18].stPrec.precision=4;      stTMOVFUND[18].stPrec.scale=2;
stTMOVFUND[19].stPrec.precision=4;      stTMOVFUND[19].stPrec.scale=2;
~

Y POR ULTIMO:    nuevo.c_varTMOVFUND.h

struct stpTMOVFUND {
DBINT iNumFol;              DBFLT8 nPer;                DBSMALLINT siCveNodo;
DBSMALLINT siSerial;        DBINT iNumEmp;              DBTINYINT tiCveTipMov;
DBSMALLINT siNumCau;        DBINT iNumPlaza;            DBTINYINT tiCveFunc;
DBTINYINT tiCveProg;        DBSMALLINT siCveSubProg;    DBSMALLINT siCveDep;
DBSMALLINT siCvePart;       DBCHAR cConacyt[1];         DBCHAR cCveCateg[7];
DBFLT8 mSdo;                DBCHAR cAfeNom[1];          DBFLT8 nHorTeo;
DBFLT8 nHorPra;             DBDATETIME sdFecIniFun;     DBDATETIME sdFecFinFun;
DBCHAR cDefFun[1];          DBCHAR cCoaCoc[1];          DBSMALLINT siPagUni;
DBSMALLINT siTipoEmp;       DBCHAR vcDescMat[180];      DBSMALLINT siStatus;
DBSMALLINT siCveNodPag;     DBINT iCveNodosTra;         DBSMALLINT siTipProc;

} stcTMOVFUND;
```

El primero y el último, se usan ambos en un archivo con extensión h para el programa en c, y de esta manera poder ligar los datos obtenidos de un “select”, a una estructura y poderlos manipular y el segundo se usa en el programa para determinar la precisión que se requiere para ese tipo de variables.

Se prepara el archivo include, el archivo de encabezado para el programa, es la concatenación de las vistas y columnas que se utilizarán y se prepara con el shell **makeh**, el cual tiene la siguiente sintaxis: **makeh <**nombre del programa>.

Aquí se juntan el primer archivo y el tercero, dejándolos en un archivo con extensión h, como ya se había explicado anteriormente.

La compilación se realiza a través de la utilería estándar de unix: **make**, la cual provee un método para mantener versiones actualizadas de los programas y de los archivos relacionados. El archivo **makefile** contiene las descripciones necesarias para producir el código ejecutable y proporciona además una interfase automática con la herramienta SCCS.

Para la depuración de los programas, tanto de errores de sintaxis como para la revisión de operaciones ejecutadas a través de las conexiones al servidor, es posible revisar si la compilación se realiza con la bandera **DEBUG,** en el makefile de la biblioteca estándar. De la misma manera, utilizando el shell **makedebug**, se produce una versión de depuración del módulo.

Aplicación del lenguaje C

Formato del código fuente.

La identación debe ser de dos espacios en cada nivel de anidación del programa. Las funciones también deben iniciar con este nivel. No se prevén excepciones. Para dar claridad al código, deben incluirse espacios entre las palabras clave del lenguaje C, variables y operadores. En el caso de las llamadas a funciones, los argumentos también deben separarse con un espacio.

Comentarios.

Los comentarios deben incluirse principalmente en los segmentos de código que realizan las tareas más importantes. Con los comentarios debe procurarse darle seguimiento a la manipulación de las variables más importantes en los programas. Los comentarios deben aportar mayor información acerca del código. Deben evitarse comentarios que únicamente describen el código. Debe considerarse la actualización de los comentarios durante el mantenimiento o evolución del código.

El formato de los comentarios es que deben ser independientes del texto del programa, es decir, ser colocados en renglones independientes, para soportar cambios en el código, sin afectar la documentación interna. Los

comentarios deben alinearse a la derecha del código utilizando una regla de 80 caracteres.

Uso de construcciones.

Instrucciones SQL. Para facilitar la lectura del código, todas las palabras clave del lenguaje SQL se escriben con mayúsculas. También las llamadas a las funciones y los argumentos para construir las operaciones SQL se alinean como en el siguiente ejemplo:

```
dbcmd(stQuery, ”SELECT iNumEmp, vcRfcEmp´, sdFecNac “);
dbcmd(stQuery, ”  FROM TEMPNOMINA “);
dbfcmd(stQuery,”  WHERE iNumEmp between %d AND %d”,
                  1500, 2000);
```

if-else-if – while. Las líneas con este tipo de construcciones deben mantener la identación en dos caracteres. Las llaves de apertura de bloques condicionales deben colocarse en la misma línea del if, else if o while, según sea el caso.

Funciones. Para facilitar la visualización de las llamadas a funciones, los argumentos de las funciones se alinean de la siguiente forma:

```
RcUnomb(stQuery,    stcMov.iNumFol,    stcMov.nPer,    stcMov.iNumEmp,
        stcMov.sdFecIniFun, stcMov.tiCveTipMov);
```

Switch . case . default. Como una validación adicional de las rutas, decisiones y condiciones de proceso, las construcciones switch deben contener un default para detectar condiciones imprevistas.

Nombres de variables comunes. Para aportar claridad en el código, los nombres de las variables deben determinarse de acuerdo a su tipo y propósito. Todas las variables deben nombrarse utilizando un prefijo que indica el tipo de dato de acuerdo con la siguiente tabla.

Tipo de datos en SQL-SERVER	**Tipo de datos en DB-LIBRARY**	**Tipo de Datos en Lenguaje C**	**Prefijo al nombrar variables**
Enteros			
TINYINT	DBTINYINT	UNSIGNED CHAR	ti
SMALLINT	DBSMALLINT	SHORT	si
INT	DBINT	LONG	i
Numéricos exactos			

Tipo de datos en SQL-SERVER	**Tipo de datos en DB-LIBRARY**	**Tipo de Datos en Lenguaje C**	**Prefijo al nombrar variables**
NUMERIC	DBNUMERIC		n
DECIMAL	DBDECIMAL		d
Numéricos con aproximación			
FLOAT	DBFLT8	DOUBLE	f
REAL	DBREAL	FLOAT	r
DUBLE	DBFLT8	DOUBLE	f
Carácter			
CHAR	DBCHAR	CHAR	c
VARCHAR	DBCHAR	CHAR	c
Binario y Bit			
BINARY	DBBINARY	UNSIGNED CHAR	bin
BIT	DBBIT	UNSIGNED CHAR	bit
Datetime			
SMALLDATETIME	DBDATETIME4	STRUCT	dt4
DATETIME	DBDATETIME	STRUCT	dt
Money			
SMALLMONEY	DBMONEY4	STRUCT	m4
MONEY	DBMONEY	STRUCT	m

Las variables que se utilizan para almacenar códigos de retorno estándar de funciones, deben ser de tipo RETCODE, definido por Sybase, los valores para este tipo pueden ser: SUCCEED para ejecución correcta o FAIL en caso de error.

Los nombres de las variables para almacenar columnas, deben tener el mismo nombre que la columna en la base de datos. Estos nombres se recuperan del diccionario de datos de Sybase durante la preparación de las vistas y variables para columnas, a través del programa **cstruct**.

Las funciones que retornan datos, deben declararse con tipo RETCODE y con el prefijo rc, como en el siguiente ejemplo:

```
RETCODE rcLeerMovs(DBPROCESS *stQuery, DBINT iNumEmp).
```

Portabilidad de tipos. Con el propósito de preservar la portabilidad de tipos de datos, se utilizan los tipos definidos por Sybase para DB-Library. De esta manera, si la siguiente declaración:

```
#define DBSMALLINT          CS_SMALLINT (short)

          cambia a:

#define DBSMALLINT          CS_INT (long)
```

Las variables declaradas con tipo DBSMALLINT, no requerirán de mantenimiento y solamente será necesario recompilar el código.

Formato de mensajes y errores. El formato de los mensajes del servidor de la base de datos, así como de los mensajes de la aplicación, es el estándar definido por Sybase y contiene:

Número de mensaje.

Nivel de severidad.

Estado.

Nombre del servidor.

Nombre del store-procedure.

Número de línea.

Texto del mensaje.

Última operación SQL.

El formato de los errores del servidor de base de datos, así como del sistema operativo es el estándar definido por Sybase y contiene:

Número de error del servidor.

Texto del error del servidor.

Última operación SQL.

Texto del error del sistema operativo.

Control de Código.

El control de código se lleva a cabo, a través de una herramienta estándar de soporte a la programación en UNIX: **Sistema de Control de**

Código Fuente (SCCS por sus siglas en inglés), que es un conjunto de programas que se usan para registrar la evolución de archivos.

A continuación se mencionan brevemente las opciones más importantes que deben utilizarse para el control de código fuente:

Para iniciar un nuevo programa, se hace con **sccs create**.

Para obtener una versión editable, se utiliza **sccs edit**.

Se puede cancelar la edición de un programa con **sccs unedit.**

El registro de cambios se realiza con **sccs delta**.

Biblioteca estándar.

La biblioteca estándar es un grupo de funciones, cuyo propósito es simplificar la escritura de operaciones comunes, como minimizar la extensión del código. Las funciones de esta biblioteca permiten:

Extraer una columna específica.

Probar si un renglón existe.

Extraer un renglón específico.

Manejar transacciones.

Relacionar columnas de resultado con variables del programa.

Manejar errores y mensajes.

Todas las funciones utilizan en código de retorno estándar de Sybase: RETCODE, que puede contener los siguientes valores: SUCCEED, en caso correcto y FAIL, en caso de error. Esta biblioteca se compila por separado y se liga durante la aplicación.

Estas funciones son para obtener un cierto tipo de dato, su sintaxis es el nombre de la función y los parámetros correspondientes, los cuales son los siguientes:

- *dbproc. Un apuntador a la estructura DBPROCESS que provee la conexión para un proceso.
- *varaddr. La dirección de la variable del programa en la cual será copiado el dato.
- *view. Una estructura que contiene información de los datos a procesar: tipo, nombre, dirección, precisión y escala.
- *numcols. Número de columnas a procesar.
- *action. Un número que indica el tipo de operación a realizar: BEGIN_TRAN, END_TRAN, ABORT_TRAN.

- *login. La clave de usuario en el servidor de la base de datos
- *password. La contraseña del usuario en el servidor de la base de datos.
- *appname. El nombre del programa, para facilitar el monitoreo de la base de datos.

Estas funciones regresan generalmente SUCCEED, en caso de que haya sido posible realizar la operación y FAIL en caso contrario. La mayoría de las funciones simplifican las siguientes tareas:

- Envían el comando SQL al servidor.
- Inicializan los resultados del siguiente SELECT.
- Relacionan la columna de la base de datos con la variable del programa.
- Lee el siguiente renglón que resulta del SELECT.
- Copia el dato y cancela el último comando SQL.

Las funciones principales son las siguientes:

dbgetinyint. Permite leer una columna de tipo TINYINT.

dbgetsmallint. Permite leer una columna de tipo SMALLINT.

dbgetint. Permite leer una columna de tipo INT, entero.

dbgetnumeric. Lee una columna de tipo NUMERIC.

dbgetflt8. Permite leer una columna de tipo DOUBLE.

dbgetmoney. Permite leer una columna de tipo MONEY.

dbgetstring. Permite leer una columna de tipo VAR

o VARCHAR.

dbgetarow. Permite leer un renglón específico de una tabla. Relaciona las columnas de la base de datos con las variables del programa, utilizando la función dbstruct, misma que se describe posteriormente. Lee el número de renglones afectados por la operación y esto nos permite evaluar si fue posible obtener el renglón.

dbfind. Evalúa la existencia de un dato.

dbhandletran. Maneja las transacciones que explícitamente se definen en la aplicación. La llamada a la función se realiza antes de construir el comando SQL y también al terminar la operación. El nivel de aislamiento definido para la conexión será utilizado al iniciar y terminar la transacción.

dbsqlrun. Ejecuta comandos SQL y evalúa sus resultados. Es posible enviar uno o más comandos a ejecución.

dbstruct. Relaciona los datos a procesar con variables del programa. Para poder disponer de un dato en la variable de un programa, se requiere: construir el comando SQL, enviar el comando SQL al servidor, inicializar los resultados del siguiente SELECT, relacionar la columna de la base de datos con la variable del programa, llamando a la función dbbind (la complejidad de la sintaxis y parámetros de la función dbbind, favorecen la producción de código extenso y errores en la codificación), leer el siguiente renglón que resulta del SELECT y copiar el dato. Debido a los pasos anteriores, la función dbstruct proporciona un mecanismo que utiliza una estructura de la información, que describe los datos que van a ser utilizados en un programa, para evitar la escritura de llamadas a la función dbbind, para cada uno de ellos. Esta estructura puede generarse con la información del diccionario de datos de Sybase, soportando así la mayor parte de los cambios en el esquema de la base de datos.

rcSybIni. Inicializar la conexión con el servidor de base de datos.

siMsgHandler. Cuando DB-Library recibe un mensaje informativo, llama automáticamente al manejador de mensajes. La forma de reportar los mensajes es la misma que se observa en isql. El manejador presenta los mensajes tal como se encuentran en la tabla sysmessages, que contiene aproximadamente 3000 mensajes sin traducirlos. Los mensajes definidos específicamente para la aplicación, son recuperados de la tabla sysusermessages. El manejador recupera la última operación SQL para mostrarla como parte del mensaje.

8. 1 PROGRAMACIÓN DE LAS FUNCIONES GENERALES.

Todas las funciones generales se encuentran en un directorio especial, las cuales son compiladas en conjunto para obtener un archivo con extensión "a", el cual como ya sabemos es una biblioteca, y la vamos a compilar con los programas principales. Para compilar esta biblioteca usamos un archivo makefile, en el cual vienen integradas todas las funciones del subsistema.

```
CON_H   = rcObacad.o rcItarba.o rcInstar.o vAtab.o rcItaraf.o\
          rcVIesp.o rcVIespimp.o rcVIporc.o rcDnomb.o rcUnomb.o\
          rcIfu.o rcVsdocf.o rcApartasig.o \
          rcVesaum.o rcVnor.o rcVplaza.o rcVId48.o rcAfpiste.o \
          rcAvarimp.o rcUdias.o rcSindicato.o rcVerLugPag.o \
          rcLugPagBanSar.o  rcLeerAntig.o rcAdif.o\
          rcDif.o rcCheqFun.o rcFecFun.o rcAntig.o rcAvaraum.o \
          rcVIm01.o rcVIm30.o


SIN_H   =  fungen.o

con_h:
        make "CFLAGS =  -xs -g -O -DDEBUG $(INCLUDE) " \
                "OBJS = $(CON_H)" "RANLIB = ls -l" lib

sin_h:
        make "CFLAGS =  -xs -g -O -DDEBUG $(INCLUDE)" \
                "OBJS = $(SIN_H)" "RANLIB = ls -l" lib

clean:
        rm libdbfun.a
comp:
        make sin_h con_h
```

En el fragmento anterior del makefile observamos que todos los archivos objetos son unidos para formar un archivo que se llama libdbfun.a

Estas son las funciones generales, cada función tiene una tarea muy específica, y pueden ser llamadas desde cualquier programa en donde se incluya esta librería.

Existe una función que se encuentra aquí, que es la que carga el tabulador, tarea primordial en este proceso. Por lo tanto ejemplificaremos esta parte con esa función:

```
/*************************************************************************

Nombre del programa :       vAtab.c

Nombre del proyecto :       Nomina UNAM

Descripcion         :       Carga el tabulador en memoria.
                            Funciones que buscan el tabulador.
```

```
Comentarios         :
Autor               :       Maribel Calderon
Fecha               :       24 Abril de 2000

*************************************************************************/
#include <stdio.h>                      /* Prototipos de E/S estandar        */
#include <stdlib.h>                     /* Prototipos de biblioteca estandar */
#include <signal.h>                     /* Prototipos de manejo de se#ales   */
#include <string.h>                     /* Prototipos de manejo de cadenas   */

#include <sybfront.h>                   /* Constantes para DB-Lib de Sybase  */
#include <sybdb.h>                      /* Estructuras y prots DB-Lib Sybase */
#include <syberror.h>                   /* Constantes para errores de Sybase */

/*-------------------------------------------------------------------------*/

#include "aplstd.h"                     /* Constantes y prototipos estandar  */
#include "apldef.h"                     /* Constantes de la aplicacion       */
#include "aplfun.h"                     /* Constantes y prototipos aplicacion*/
#include "vAtab.h"
                                  /* Definición de los arreglos necesarios */

TAB_BASE base[NIVELES_BASE][TABS];

TAB_CONF conf[40][TABS];

TAB_FUNC func[105][TABS];

TAB_ASIG asig[10][TABS];

TAB_CARR carr[40][TABS];

CATEG_BASE categorias[CATEGO];

CATEG_BASE *cats;
                              /* Definición de las funciones         */

void vBnivel(DBCHAR cCveCateg[7], DBSMALLINT siTipoEmp, DBINT *iNiv);

void vBfecha(DBSMALLINT siTipoEmp, DBINT *iCont, DBINT iFec);

void vBsdo(DBINT iFecFu, DBCHAR cCveCateg[7], DBFLT8 nHorTot,
          DBSMALLINT siTipoEmp, RET sueldos[TABS], DBINT *iResult);

void vBsdoret(DBINT iFecIniFu, DBINT iFecFinFu, DBCHAR cCveCateg[7],
             DBFLT8 nHorTot, DBSMALLINT siTipoEmp, RET sueldos[TABS]);

/*-------------------------------------------------------------------------*/
RETCODE vAtab(DBPROCESS *stFec);

RETCODE vAtab(DBPROCESS *stFec)  {

 RETCODE         rcMov;

 DBINT           iFecIni, iFecFin,iFecAct,*iFec,i=0,j=0,iNiv;
 DBCHAR          cNivel[3], cCveCateg[7], cTipoTab[2];
 DBFLT8          fSdo, fSdoCF, fTotSdo, fSdoMD, fSdoCH;
 DBSMALLINT      siNivel=0;
/*-------------------------------------------------------------------------*/
                              /*  Se inicializan los arreglos para
                                  que no lleven datos innecesarios*/
 cats=&categorias[0];
 while (i<CATEGO) {
   strcpy(cats->cCveCateg,"       ");
   cats->iNivel = 0;
   i++;
 }
 i = 0;


/*-------------------------------------------------------------------------*/
```

```
                                          /*Se van a copiar las fechas */
                                           /*Fechas de funcionario  */
dbcmd(stFec, "SELECT DISTINCT CONVERT(CHAR(10), sdFechaInicio,112),");
dbcmd(stFec, " CONVERT(CHAR(10), sdFechaFin,112)");
dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
dbcmd(stFec, " AND c.siTipoEmp IN (3)");
dbcmd(stFec, " AND t.cTipoTabulador NOT IN ('AS')");
dbfcmd(stFec," AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);
dbsqlexec(stFec);

while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

  dbbind(stFec, 1, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
  dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecFin);

  while (dbnextrow(stFec ) != NO_MORE_ROWS) {
   iFec = &iFec_Func[i][0];
   *(iFec) = iFecIni;
   *(iFec+1) = iFecFin;
   i++;
  }
 }
}
i = 0;
                             /*  Fechas del tabulador de carrera */

dbcmd(stFec, "SELECT DISTINCT CONVERT(CHAR(10), sdFechaInicio,112),");
dbcmd(stFec, " CONVERT(CHAR(10), sdFechaFin,112)");
dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
dbcmd(stFec, " AND c.siTipoEmp IN (5)");
dbcmd(stFec, " AND t.cTipoTabulador NOT IN ('AS')");
dbfcmd(stFec, " AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);

dbsqlexec(stFec);

while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
 if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

  dbbind(stFec, 1, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
  dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecFin);

  while (dbnextrow(stFec ) != NO_MORE_ROWS) {
   iFec = &iFec_Carr[i][0];
   *(iFec) = iFecIni;
   *(iFec+1) = iFecFin;
   i++;
  }
 }
}
i = 0;
                                   /* Fechas del tabulador de base*/

dbcmd(stFec, "SELECT DISTINCT CONVERT(CHAR(10), sdFechaInicio,112),");
dbcmd(stFec, " CONVERT(CHAR(10), sdFechaFin,112)");
dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
dbcmd(stFec, " AND c.cRangoBase = t.cNivelTabulador");
dbcmd(stFec, " AND c.siTipoEmp IN (1)");
dbfcmd(stFec, " AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);

dbsqlexec(stFec);

while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
 if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {
```

```
  dbbind(stFec, 1, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
  dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecFin);

  while (dbnextrow(stFec ) != NO_MORE_ROWS) {
   iFec = &iFec_Base[i][0];
   *(iFec) = iFecIni;
   *(iFec+1) = iFecFin;
   i++;
  }
 }
}
i = 0;


                                  /* Fechas del tabulador de confianza*/

dbcmd(stFec, "SELECT DISTINCT CONVERT(CHAR(10), sdFechaInicio,112),");
dbcmd(stFec, " CONVERT(CHAR(10), sdFechaFin,112)");
dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
/*dbcmd(stFec, " AND c.cRangoBase = t.cNivelTabulador");*/
dbcmd(stFec, " AND c.siTipoEmp IN (2)");
dbfcmd(stFec, " AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);

dbsqlexec(stFec);

while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
 if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

  dbbind(stFec, 1, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
  dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecFin);

  while (dbnextrow(stFec ) != NO_MORE_ROWS) {
   iFec = &iFec_Conf[i][0];
   *(iFec) = iFecIni;
   *(iFec+1) = iFecFin;
   i++;
  }
 }
}
i = 0;
                                     /* Fechas de asignatura */

dbcmd(stFec, "SELECT DISTINCT CONVERT(CHAR(10), sdFechaInicio,112),");
dbcmd(stFec, " CONVERT(CHAR(10), sdFechaFin,112)");
dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
dbcmd(stFec, " AND c.cRangoBase = t.cNivelTabulador");
dbcmd(stFec, " AND c.siTipoEmp IN (4,6)");
dbcmd(stFec, " AND t.cTipoTabulador IN ('AS')");
dbfcmd(stFec, " AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);
dbsqlexec(stFec);
while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
 if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

  dbbind(stFec, 1, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
  dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecFin);

  while (dbnextrow(stFec ) != NO_MORE_ROWS) {
   iFec = &iFec_Asig[i][0];
   *(iFec) = iFecIni;
   *(iFec+1) = iFecFin;
   i++;
  }
 }
}
```

```
/*************************************************************************/
                         /* Carga categorías con sus respectivos niveles, */
                        /* tanto de base como de asignatura */

 i = 0;
 strcpy(cCveCateg, "       ");
 strcpy(cNivel,"   ");
 dbcmd(stFec, "SELECT DISTINCT c.cCveCateg, siNumBase");
 dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
 dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
 dbcmd(stFec, " AND c.cRangoBase = t.cNivelTabulador");
 dbcmd(stFec, " AND c.siTipoEmp IN (1,12,4,6)");
 dbcmd(stFec, " AND siNumBase IS NOT NULL ");
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);
 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

   dbbind(stFec, 1, CHARBIND, (DBINT) 0, (BYTE *) cCveCateg);
   dbbind(stFec, 2, SMALLBIND, (DBINT) 0, (BYTE *) &siNivel);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {
    strcpy(cats->cCveCateg, cCveCateg);
    cats->iNivel = siNivel;
    cats++;
    strcpy(cCveCateg, "       ");
    strcpy(cNivel,"   ");
   }
  }
 }

                                          /* Carga del tabulador de base */
 i = 0;
 dbcmd(stFec, "SELECT DISTINCT t.siNumBase, ");
 dbcmd(stFec, " CONVERT(CHAR(10), t.sdFechaInicio, 112), ");
 dbcmd(stFec, " t.mSueldo, t.mSueldoCF ");
 dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t ");
 dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador ");
 dbcmd(stFec, " AND t.cNivelTabulador=c.cRangoBase ");
 dbcmd(stFec, " AND c.siTipoEmp in (12,1)");
 dbfcmd(stFec, " AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);

 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

   dbbind(stFec, 1, SMALLBIND, (DBINT) 0, (BYTE *) &siNivel);
   dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
   dbbind(stFec, 3, FLT8BIND, (DBINT) 0, (BYTE *) &fSdo);
   dbbind(stFec, 4, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoMD);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {
    while (iFec_Base[i][0] != iFecIni)
     i++;
    base[siNivel][i].fSdo = fSdo;
    base[siNivel][i].fSdoMed = fSdoMD;
    i = 0;
   }
  }
 }
/*************************************************************************/
                                    /* Carga del tabulador de confianza  */
 i = 0;
 strcpy(cNivel,"   ");
 dbcmd(stFec, "SELECT distinct t.cNivelTabulador,");
 dbcmd(stFec, " CONVERT(CHAR(10),t.sdFechaInicio,112),");
 dbcmd(stFec, " t.mSueldo, t.mSueldoCF, c.cTipoTabulador");
 dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
```

```
 dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
 dbcmd(stFec, " AND c.siTipoEmp in (2)");
 dbfcmd(stFec," AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);
 dbcmd(stFec, " ORDER BY c.cTipoTabulador");
 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec) == SUCCEED)) {

   dbbind(stFec, 1, CHARBIND, (DBINT) 0, (BYTE *) cNivel);
   dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
   dbbind(stFec, 3, FLT8BIND, (DBINT) 0, (BYTE *) &fSdo);
   dbbind(stFec, 4, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoMD);
   dbbind(stFec, 5, CHARBIND, (DBINT) 0, (BYTE *) cTipoTab);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {
    while (iFec_Conf[i][0] != iFecIni)
     i++;
                                              /* Se tiene que hacer una separación de los
                                              distintos tipos de tabulador para confianza
                                              */
    if (strncmp(cTipoTab,"CQ",2) == EQUAL) {
     conf[atoi(cNivel)][i].fSdoCQ = fSdo;
     conf[atoi(cNivel)][i].fSdoCQMed = fSdoMD;
    }
    else if (strncmp(cTipoTab, "CO",2) == EQUAL) {
     conf[atoi(cNivel)][i].fSdoCO = fSdo;
     conf[atoi(cNivel)][i].fSdoCOMed = fSdoMD;
    }
    else {
     conf[atoi(cNivel)][i].fSdoCS = fSdo;
     conf[atoi(cNivel)][i].fSdoCSMed = fSdoMD;
    }
    i = 0;
    strcpy(cNivel, "   ");
   }
  }
 }

/*************************************************************************/
                                    /* Carga del tabulador de funcionario */
 i = 0;
 strcpy(cNivel,"   ");
 dbcmd(stFec, "SELECT distinct t.cNivelTabulador,");
 dbcmd(stFec, " CONVERT(CHAR(10),t.sdFechaInicio,112),");
 dbcmd(stFec, " t.mSueldo,t.mSueldoCF, t.mSueldo+t.mSueldoCF,");
 dbcmd(stFec, " t.cTipoTabulador");
 dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t ");
 dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
 dbcmd(stFec, " AND c.siTipoEmp in (3) AND t.cTipoTabulador");
 dbcmd(stFec, " IN ('CA', 'CF', 'CM', 'CV', 'CD')");
 dbfcmd(stFec," AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);
 dbcmd(stFec, " ORDER BY c.cTipoTabulador");
 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec) == SUCCEED)) {

   dbbind(stFec, 1, CHARBIND, (DBINT) 0, (BYTE *) cNivel);
   dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
   dbbind(stFec, 3, FLT8BIND, (DBINT) 0, (BYTE *) &fSdo);
   dbbind(stFec, 4, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoCF);
   dbbind(stFec, 5, FLT8BIND, (DBINT) 0, (BYTE *) &fTotSdo);
   dbbind(stFec, 6, CHARBIND, (DBINT) 0, (BYTE *) cTipoTab);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {
    while (iFec_Func[i][0] != iFecIni)
     i++;
```

```
    if (strncmp(cTipoTab,"CA",2) == EQUAL){
     func[atoi(cNivel)][i].fSdoCA = fSdo;
     func[atoi(cNivel)][i].fSdoCAF = fSdoCF;
     func[atoi(cNivel)][i].fSdototCAF = fTotSdo;
    }
    else if (strncmp(cTipoTab,"CV",2) == EQUAL){
     func[atoi(cNivel)][i].fSdoCV = fSdo;
     func[atoi(cNivel)][i].fSdoCVF = fSdoCF;
     func[atoi(cNivel)][i].fSdototCVF = fTotSdo;
    }
    else if (strncmp(cTipoTab,"CD",2) == EQUAL){
     func[atoi(cNivel)][i].fSdoCD = fSdo;
     func[atoi(cNivel)][i].fSdoCDF = fSdoCF;
     func[atoi(cNivel)][i].fSdototCDF = fTotSdo;
    }
    else if (strncmp(cTipoTab,"CM",2) == EQUAL){
     func[atoi(cNivel)][i].fSdoCM = fSdo;
     func[atoi(cNivel)][i].fSdoCMF = fSdoCF;
     func[atoi(cNivel)][i].fSdototCMF = fTotSdo;
    }
    else {
     func[atoi(cNivel)][i].fSdoCF = fSdo;
    }
    i = 0;
    strcpy(cNivel, "   ");
   }
  }
 }
/*************************************************************************/
                                  /* Carga de tabulador de asignatura */

i=0;
 dbcmd(stFec, "SELECT DISTINCT t.siNumBase, ");
 dbcmd(stFec, "  CONVERT(CHAR(10),t.sdFechaInicio,112), ");
 dbcmd(stFec, "  t.mSueldo, t.mMatDidact, t.mCompHoras ");
 dbcmd(stFec, "  FROM CCATEGORIAS c, TTABULADOR t ");
 dbcmd(stFec, "  WHERE c.cTipoTabulador = t.cTipoTabulador ");
 dbcmd(stFec, "  AND t.cNivelTabulador=c.cRangoBase ");
 dbcmd(stFec, "  AND c.siTipoEmp in (4,6)");
 dbfcmd(stFec," AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);

 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

   dbbind(stFec, 1, SMALLBIND, (DBINT) 0, (BYTE *) &siNivel);
   dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
   dbbind(stFec, 3, FLT8BIND, (DBINT) 0, (BYTE *) &fSdo);
   dbbind(stFec, 4, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoMD);
   dbbind(stFec, 5, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoCH);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {
    while (iFec_Asig[i][0] != iFecIni)
     i++;
    asig[siNivel][i].fSdo = fSdo;
    asig[siNivel][i].fMatDid = fSdoMD;
    asig[siNivel][i].fCompHr = fSdoCH;
    i = 0;
   }
  }
 }
```

```
/*************************************************************************/
                                    /* Carga del tabulador de carrera  */

                           /* Primero se carga un arreglo con los niveles */

 strcpy(cNivel, "   ");
 i = 0;
 dbcmd(stFec, "SELECT DISTINCT t.cNivelTabulador");
 dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
 dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
 dbcmd(stFec, " AND SUBSTRING(cCveCateg,6,2) = t.cNivelTabulador");
 dbcmd(stFec, " AND c.cTipoTabulador in ('AP', 'DI', 'TA')");
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);

 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec ) == SUCCEED)) {

   dbbind(stFec, 1, CHARBIND, (DBINT) 0, (BYTE *) cNivel);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {
    iNivCarr[i] = atoi(cNivel);
    i++;
    strcpy(cNivel,"   ");
   }
  }
 }
 /*
 for(i=0; i<=40; i++)
   printf("iNivCarr[i]  %d \n", iNivCarr[i]);
*/
                                /*Se carga el tabulador de carrera*/
 i = 0;
 iNiv = 0;
 strcpy(cNivel,"   ");
 dbcmd(stFec, "SELECT distinct t.cNivelTabulador,");
 dbcmd(stFec, " CONVERT(CHAR(10),t.sdFechaInicio,112),");
 dbcmd(stFec, " t.mSueldo, c.cTipoTabulador,");
 dbcmd(stFec, " t.mMatDidact, t.mCompHoras ");
 dbcmd(stFec, " FROM CCATEGORIAS c, TTABULADOR t");
 dbcmd(stFec, " WHERE c.cTipoTabulador = t.cTipoTabulador");
 dbcmd(stFec, " AND SUBSTRING(cCveCateg,6,2) = t.cNivelTabulador");
 dbcmd(stFec, " AND c.cTipoTabulador in ('AP', 'DI', 'TA')");
 dbfcmd(stFec," AND t.sdFechaInicio >= '%d'", FECHATABULADOR);
 dbfcmd(stFec," AND c.siStatus = %d", ACTIVO);
 dbcmd(stFec, " ORDER BY c.cTipoTabulador");
 dbsqlexec(stFec);

 while ((rcMov = dbresults(stFec )) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stFec) == SUCCEED)) {

   dbbind(stFec, 1, CHARBIND, (DBINT) 0, (BYTE *) cNivel);
   dbbind(stFec, 2, INTBIND, (DBINT) 0, (BYTE *) &iFecIni);
   dbbind(stFec, 3, FLT8BIND, (DBINT) 0, (BYTE *) &fSdo);
   dbbind(stFec, 4, CHARBIND, (DBINT) 0, (BYTE *) cTipoTab);
   dbbind(stFec, 5, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoMD);
   dbbind(stFec, 6, FLT8BIND, (DBINT) 0, (BYTE *) &fSdoCH);

   while (dbnextrow(stFec ) != NO_MORE_ROWS) {

    while (iFec_Carr[i][0] != iFecIni)
     i++;
    while (iNivCarr[iNiv] != atoi(cNivel))
     iNiv++;

/*    printf("cNivel %s, iFecIni %d, fSdo %f, cTipoTab %s\n", cNivel, iFecIni,
          fSdo, cTipoTab);*/
    if ((strncmp(cTipoTab,"DI",2) == EQUAL)
     || (strncmp(cTipoTab, "AP",2) == EQUAL)
     || ((strncmp(cTipoTab, "TA",2) == EQUAL) && ((iNivCarr[iNiv] != 20)
```

```
     && (iNivCarr[iNiv] != 27) && (iNivCarr[iNiv] != 33)
     && (iNivCarr[iNiv] != 44) && (iNivCarr[iNiv] != 46)
     && (iNivCarr[iNiv] != 58) && (iNivCarr[iNiv] != 67)
     && (iNivCarr[iNiv] != 89))){
     carr[iNiv][i].fSdoDI = fSdo;
     carr[iNiv][i].fMatDidDI = fSdoMD;
     carr[iNiv][i].fCompHrDI = fSdoCH;
    }
    else {
     carr[iNiv][i].fSdoTA = fSdo;
     carr[iNiv][i].fMatDidTA = fSdoMD;
     carr[iNiv][i].fCompHrTA = fSdoCH;
    }
    i = 0;
    iNiv = 0;
    strcpy(cNivel, "   ");
    strcpy(cTipoTab, "  ");
   }
  }
 }

/*************************************************************************/
}

                                    /* BUSCA UN SUELDO EN BASE A UNA FECHA*/
/* Los parametros que recibe son: la fecha del sueldo que se va a buscar, que */
/* puede ser la de inicio de forma unica o la fecha del periodo actual, la    */
/* categoría, las horas totales, tipo de empleado, la estructura de sueldos,  */
/* donde se van a guardar los datos obtenidos, y un parametro entero el cual  */
/* tiene un 1 si encontro datos y un 0 si no encontro nada.                   */
/* En este caso, la estructura de sueldos únicamente se va a tener datos en el*/
/* renglón cero del arreglo                                                   */

void vBsdo(DBINT iFecFu, DBCHAR cCveCateg[7], DBFLT8 nHorTot,
          DBSMALLINT siTipoEmp, RET sueldos[TABS], DBINT *iResult)  {

 DBINT iConA, iConB, *iFec, iNivel, *iFechas;
 DBCHAR cCateg[3];
 DBFLT8 fSdoRed;
 strcpy(cCateg, "   ");

                       /* Buscamos los dos caracteres claves de la categoría para poder
                       obtener el sueldo correcto */

 if(siTipoEmp == CONFIANZA || siTipoEmp == FUNCIONARIO
  || siTipoEmp == ASIGNATURA)  {
  cCateg[0] = cCveCateg[1];
  cCateg[1] = cCveCateg[2];
 }
 if(siTipoEmp == CARRERA) {
  cCateg[0] = cCveCateg[3];
  cCateg[1] = cCveCateg[4];
 }
                       /*Aquí no entran los empleados de honorarios, debido a que no tienen
                       categorías que se encuentren en el tabulador*/

 if (siTipoEmp != HONORARIOS) {
  iConA = iConB = CERO;

  vBnivel(cCveCateg, siTipoEmp, &iNivel);          /*Obtenemos el nivel  */

  if( iNivel != NIVELES_BASE)  {                          /* Tenemos un tope para los
                                                          niveles*/

   vBfecha(siTipoEmp,&iConA, iFecFu);                     /* Buscamos la fecha
                                                          correspondiente  en los
                                                          distintos arreglos de acuerdo al
                                                          tipo de empleado*/

   if(siTipoEmp == BASE )
    iFechas = &iFec_Base[iConA][0];
```

```
else if(siTipoEmp == CONFIANZA)
 iFechas = &iFec_Conf[iConA][0];

else if(siTipoEmp == FUNCIONARIO )
 iFechas = &iFec_Func[iConA][0];

else if(siTipoEmp == CARRERA)
 iFechas = &iFec_Carr[iConA][0];
else
 iFechas = &iFec_Asig[iConA][0];

sueldos[0].iFecIni = *(iFechas);
sueldos[0].iFecFin = *(iFechas+1);

                    /* Con la fecha correcta, procedemos a buscar con este dato el
                    sueldo en el arreglo correspondiente, de acuerdo al tipo de
                    empleado,  ya que cada uno tiene su metodología para buscarlo*/

if (iConA < TABS) {

 if (siTipoEmp == BASE) {
  if (nHorTot > 8 && nHorTot <= 16)
   sueldos[0].fSdoTot = base[iNivel][iConA].fSdoMed;
  else
   sueldos[0].fSdoTot = base[iNivel][iConA].fSdo;
 }
 else if (siTipoEmp == CONFIANZA)  {

  if (strncmp(cCateg,"CF",2) == EQUAL) {
   if (nHorTot > 0 && nHorTot <= 16)
    sueldos[0].fSdoTot = conf[iNivel][iConA].fSdoCOMed;
   else
    sueldos[0].fSdoTot = conf[iNivel][iConA].fSdoCO;
  }
  else if(strncmp(cCateg,"FQ",2) == EQUAL) {

    sueldos[0].fSdoTot = conf[iNivel][iConA].fSdoCQ;
  }
  else {

    sueldos[0].fSdoTot = conf[iNivel][iConA].fSdoCS;
  }
 }
 else if (siTipoEmp == FUNCIONARIO)  {

  if((strncmp(cCateg,"CA", 2) == EQUAL)
   || (strncmp(cCateg,"CH", 2) == EQUAL)
   || (strncmp(cCateg,"RC", 2) == EQUAL)) {
   sueldos[0].fSdo = func[iNivel][iConA].fSdoCA;
   sueldos[0].fSdoCF = func[iNivel][iConA].fSdoCAF;
   sueldos[0].fSdoTot = func[iNivel][iConA].fSdototCAF;
  }
  else if (strncmp(cCateg,"CV",2) == EQUAL)  {
   sueldos[0].fSdo = func[iNivel][iConA].fSdoCV;
   sueldos[0].fSdoCF = func[iNivel][iConA].fSdoCVF;
   sueldos[0].fSdoTot = func[iNivel][iConA].fSdototCVF;
  }
  else if (strncmp(cCateg,"CM",2) == EQUAL)  {
   sueldos[0].fSdo = func[iNivel][iConA].fSdoCM;
   sueldos[0].fSdoCF = func[iNivel][iConA].fSdoCMF;
   sueldos[0].fSdoTot = func[iNivel][iConA].fSdototCMF;
  }
  else if (strncmp(cCateg,"CD",2) == EQUAL)  {
   sueldos[0].fSdo = func[iNivel][iConA].fSdoCD;
   sueldos[0].fSdoCF = func[iNivel][iConA].fSdoCDF;
   sueldos[0].fSdoTot = func[iNivel][iConA].fSdototCDF;
  }
  else if((strncmp(cCateg,"CI",2) == EQUAL)  ||
    (strncmp(cCateg,"CC",2) == EQUAL) ||
    (strncmp(cCateg,"CN",2) == EQUAL)) {
```

```
      sueldos[0].fSdo = func[iNivel][iConA].fSdoCF;
      sueldos[0].fSdoTot = func[iNivel][iConA].fSdoCF;
     }
    }
    else if (siTipoEmp == ASIGNATURA)  {
     if (strncmp(cCateg," M",2) != EQUAL) {
      sueldos[0].fSdoTot = asig[iNivel][iConA].fSdo*nHorTot;
     }
     else
      sueldos[0].fSdoTot = asig[iNivel][iConA].fSdo;

     sueldos[0].fMatDid = asig[iNivel][iConA].fMatDid * nHorTot;
     sueldos[0].fCompHrs = asig[iNivel][iConA].fCompHr;
    }
                        /* Una vez loalizadas las coordenadas donde se encuentran los
                        sueldos se graban en los campos de la estructura que les corresponde
                        */
    else {
     if (((strncmp(cCateg,"72",2) == EQUAL) && (iNivCarr[iNivel] == 20))
      || ((strncmp(cCateg,"73",2) == EQUAL) && (iNivCarr[iNivel] == 27))
      || ((strncmp(cCateg,"81",2) == EQUAL) && (iNivCarr[iNivel] == 33))
      || ((strncmp(cCateg,"75",2) == EQUAL) && (iNivCarr[iNivel] == 44))
      || ((strncmp(cCateg,"83",2) == EQUAL) && (iNivCarr[iNivel] == 46))
      || ((strncmp(cCateg,"76",2) == EQUAL) && (iNivCarr[iNivel] == 58))
      || ((strncmp(cCateg,"96",2) == EQUAL) && (iNivCarr[iNivel] == 89))
      || ((strncmp(cCateg,"84",2) == EQUAL) && (iNivCarr[iNivel] == 67))) {
      sueldos[0].fSdoTot = carr[iNivel][iConA].fSdoTA;
      sueldos[0].fMatDid = carr[iNivel][iConA].fMatDidTA;
      sueldos[0].fCompHrs = carr[iNivel][iConA].fCompHrTA;
     }
     else {
      sueldos[0].fSdoTot = carr[iNivel][iConA].fSdoDI;
      sueldos[0].fMatDid = carr[iNivel][iConA].fMatDidDI;
      sueldos[0].fCompHrs = carr[iNivel][iConA].fCompHrDI;
     }
    }

    if(sueldos[0].fSdoTot == ZERO)
     *iResult = 0;
    else
     *iResult = 1;
   }
   else {
               /* Si no encuentra una fecha en el tabulador, manda un mensaje */

    printf("\n*LA FECHA %d NO ESTA EN EL TABULADOR,PARA LA CATEGORIA %.7s\n",
          iFecFu, cCveCateg);
    *iResult = 0;
   }
  }
  else  {
                        /* De igual manera nos alerta cuando buscamos una categoría
                        inexistente */

    printf("\n***LA CATEGORIA %.7s NO SE ENCUENTRA EN EL TABULADOR***\n",
            cCveCateg);
    *iResult = 0;
  }
 }
 else {
  printf("\n***LA CATEGORIA %.7s NO SE ENCUENTRA EN EL TABULADOR HON***\n",
  cCveCateg);
    *iResult = 0;
 }

 return;
}

/*************************************************************************/

               */  BUSCA SUELDOS RETROACTIVOS */
```

```
/* Los parámetros son:  fecha de inicio y de fin de Forma Única, categoría,  */
/* horas totales, tipo de empleado, la  estructura donde se van a guardar los*/
/* sueldos y un parámetro entero donde se va a regresar el número de renglón, */
/* hasta donde se llenó la estructura de sueldos y en caso de que no         */
/* encuentre sueldos este parámetro lo regresa con -1                        */


void vBsdoret(DBINT iFecIniFu, DBINT iFecFinFu, DBCHAR cCveCateg[7], DBFLT8 nHorTot,
DBSMALLINT siTipoEmp, RET sueldos[TABS])
{
 DBCHAR cCateg[2];
 DBINT iConA, iConB, iCon, iCont, iNivel, *iFechas, iDifDias;
 DBFLT8 fSdoRed;


 iConA = iConB = iCon = iCont = iDifDias = iNivel = CERO;
 strcpy(cCateg, "   ");
                       /* Preparamos la categoría para la búsqueda, de acuerdo al tipo de
                       empleado */

 if(siTipoEmp == CONFIANZA || siTipoEmp == FUNCIONARIO
  || siTipoEmp == ASIGNATURA)  {
  cCateg[0] = cCveCateg[1];
  cCateg[1] = cCveCateg[2];
 }
 if(siTipoEmp == CARRERA) {
  cCateg[0] = cCveCateg[3];
  cCateg[1] = cCveCateg[4];
 }

 /*printf("\niFecIni %d,  iFecFin %d, cCveCateg %s\n", iFecIniFu, iFecFinFu,
                                  cCveCateg);*/

              /*  Los empleados de honorarios no tienen tabulador  */

 if(siTipoEmp != HONORARIOS) {
  vBnivel(cCveCateg, siTipoEmp, &iNivel);

  if (iNivel != NIVELES_BASE)  {

   vBfecha(siTipoEmp, &iConA, iFecIniFu);
   vBfecha(siTipoEmp, &iConB, iFecFinFu);

   if(iConA <= TABS && iConB <= TABS)  {

                       /* Buscamos la fecha correspondiente y con este apuntador, buscamos
                       en el tabulador correspondiente, de acuerdo al tipo de empleado*/

    for(iCon = iConA; iCon <= (iConB); iCon++)  {

     if(siTipoEmp == BASE)
      iFechas = &iFec_Base[iCon][0];
     else if(siTipoEmp == CONFIANZA)
      iFechas = &iFec_Conf[iCon][0];
     else if(siTipoEmp == FUNCIONARIO)
      iFechas = &iFec_Func[iCon][0];
     else if(siTipoEmp == CARRERA)
      iFechas = &iFec_Carr[iCon][0];
     else
      iFechas = &iFec_Asig[iCon][0];

     if (iCon == iConA)
      sueldos[iCont].iFecIni = iFecIniFu;
     else
      sueldos[iCont].iFecIni = *(iFechas);
     if (iCon == iConB)
      sueldos[iCont].iFecFin = iFecFinFu;
     else
      sueldos[iCont].iFecFin = *(iFechas+1);
```

```
                    /* A diferencia de la búsqueda de un sólo sueldo, aquí buscamos
                    sueldos retroactivos y obtenemos la diferencia de días entre las
                    fechas proporcionadas y divididas por los distintos cambios de
                    tabulador */

iOdifdia(sueldos[iCont].iFecIni, sueldos[iCont].iFecFin, &iDifDias);
sueldos[iCont].iDias = iDifDias;
if(siTipoEmp == BASE) {
 if(nHorTot > 8 && nHorTot <= 16)
  sueldos[iCont].fSdoTot = base[iNivel][iCon].fSdoMed;
 else
  sueldos[iCont].fSdoTot = base[iNivel][iCon].fSdo;
}

else if (siTipoEmp == CONFIANZA)  {

 if (strncmp(cCateg,"CF",2) == EQUAL) {
  if (nHorTot > 0 && nHorTot <= 16)
   sueldos[iCont].fSdoTot = conf[iNivel][iCon].fSdoCOMed;
  else
   sueldos[iCont].fSdoTot = conf[iNivel][iCon].fSdoCO;
 }
 else if (strncmp(cCateg,"FQ",2) == EQUAL) {

   sueldos[iCont].fSdoTot = conf[iNivel][iCon].fSdoCQ;
 }
 else {

   sueldos[iCont].fSdoTot = conf[iNivel][iCon].fSdoCS;
 }
}
else if (siTipoEmp == FUNCIONARIO)  {
 if ((strncmp(cCateg,"CA",2) == EQUAL)
  || (strncmp(cCateg,"CH",2) == EQUAL)
  || (strncmp(cCateg,"RC",2) == EQUAL))  {
  sueldos[iCont].fSdo = func[iNivel][iCon].fSdoCA;
  sueldos[iCont].fSdoCF = func[iNivel][iCon].fSdoCAF;
  sueldos[iCont].fSdoTot = func[iNivel][iCon].fSdototCAF;
 }
 else if (strncmp(cCateg,"CV",2) == EQUAL)  {
  sueldos[iCont].fSdo = func[iNivel][iCon].fSdoCV;
  sueldos[iCont].fSdoCF = func[iNivel][iCon].fSdoCVF;
  sueldos[iCont].fSdoTot = func[iNivel][iCon].fSdototCVF;
 }
 else if (strncmp(cCateg,"CM",2) == EQUAL)  {
  sueldos[iCont].fSdo = func[iNivel][iCon].fSdoCM;
  sueldos[iCont].fSdoCF = func[iNivel][iCon].fSdoCMF;
  sueldos[iCont].fSdoTot = func[iNivel][iCon].fSdototCMF;
 }
 else if (strncmp(cCateg,"CD",2) == EQUAL)  {
  sueldos[iCont].fSdo = func[iNivel][iCon].fSdoCD;
  sueldos[iCont].fSdoCF = func[iNivel][iCon].fSdoCDF;
  sueldos[iCont].fSdoTot = func[iNivel][iCon].fSdototCDF;
 }
 else if((strncmp(cCateg,"CI",2) == EQUAL)  ||
  (strncmp(cCateg,"CC",2) == EQUAL) ||
  (strncmp(cCateg,"CN",2) == EQUAL)) {
  sueldos[iCont].fSdo = func[iNivel][iCon].fSdoCF;
  sueldos[iCont].fSdoTot = func[iNivel][iCon].fSdoCF;
 }
}
else if (siTipoEmp == ASIGNATURA)  {
 if (strncmp(cCateg," M",2) != EQUAL) {
  fSdoRed = asig[iNivel][iCon].fSdo*nHorTot;
  rcRedCal(&fSdoRed);
  sueldos[iCont].fSdoTot = fSdoRed;
 }
 else
  sueldos[iCont].fSdoTot = asig[iNivel][iCon].fSdo;
```

```
                        /*  Una vez encontrados los sueldos se guardan en el arreglo que se
                        definió como parámetro */

      sueldos[iCont].fMatDid = asig[iNivel][iCon].fMatDid * nHorTot;
      sueldos[iCont].fCompHrs = asig[iNivel][iCon].fCompHr;

     }
     else {
      if (((strncmp(cCateg,"72",2) == EQUAL) && (iNivCarr[iNivel] == 20))
       || ((strncmp(cCateg,"73",2) == EQUAL) && (iNivCarr[iNivel] == 27))
       || ((strncmp(cCateg,"81",2) == EQUAL) && (iNivCarr[iNivel] == 33))
       || ((strncmp(cCateg,"75",2) == EQUAL) && (iNivCarr[iNivel] == 44))
       || ((strncmp(cCateg,"83",2) == EQUAL) && (iNivCarr[iNivel] == 46))
       || ((strncmp(cCateg,"76",2) == EQUAL) && (iNivCarr[iNivel] == 58))
       || ((strncmp(cCateg,"96",2) == EQUAL) && (iNivCarr[iNivel] == 89))
       || ((strncmp(cCateg,"84",2) == EQUAL) && (iNivCarr[iNivel] == 67))
       ) {
       sueldos[iCont].fSdoTot = carr[iNivel][iCon].fSdoTA;
       sueldos[iCont].fMatDid = carr[iNivel][iCon].fMatDidTA;
       sueldos[iCont].fCompHrs = carr[iNivel][iCon].fCompHrTA;
      }
      else {
       sueldos[iCont].fSdoTot = carr[iNivel][iCon].fSdoDI;
       sueldos[iCont].fMatDid = carr[iNivel][iCon].fMatDidDI;
       sueldos[iCont].fCompHrs = carr[iNivel][iCon].fCompHrDI;
      }
     }
     iCont++;
    }
     iLim = iCont-1;
   }
   else {             /*  Si no encontramos la fecha manda un mensaje de alerta */

    printf("\n*LA FECHA %d %d NO ESTA EN EL TABULADOR,PARA LA CATEGORIA %.7s\n",
           iFecIniFu, iFecFinFu, cCveCateg);
   iLim = -1;
   }
  }
  else {              /* También si no se encuentra la categoría, manda un mensaje */
   printf("\n****LA CATEGORIA %.7s NO SE ENCUENTRA EN TABULADOR***\n",
         cCveCateg);
   iLim = -1;
  }
 }
 else {
  printf("\n****LA CATEGORIA %.7s NO SE ENCUENTRA EN TABULADOR,HON***\n",
         cCveCateg);
  iLim =- 1;
 }
/*
 printf("Sueldos \n");
 for (iCont = 0; iCont <= 2; iCont++)  {
   printf("fSdoTot  %f\n", sueldos[iCont].fSdoTot);
 }
*/
 return;
}

/*************************************************************************/
                                              /* Busca el nivel */
                        /*  Esta funcion busca el nivel de acuerdo al tipo de empleado, para
                        lo cual necesitamos la categoría, el tipo de empleado y una variable
                        para regresar el valor del nivel encontrado */

void vBnivel(DBCHAR cCveCateg[7], DBSMALLINT siTipoEmp, DBINT *iNiv)  {
 DBINT iCon = CERO, iContCat=CERO, i=CERO;

 DBCHAR cNivel[2];
 strcpy(cNivel,"   ");
```

```
                        /*   Para el nivel de base y asignatura, buscamos en el arreglo de
                        categorías, en donde asignamos un nivel de manera arbitraria. */

 if (siTipoEmp == BASE || siTipoEmp == ASIGNATURA)  {

  cats=&categorias[0];
  iCon = 0;
  while(iCon == 0 && iContCat != CATEGO) {
   if ((strncmp(cats->cCveCateg,cCveCateg,7)) ==  EQUAL) {
    iCon = UNO; *iNiv = cats->iNivel;
   }
   cats++; iContCat++;
  }
 }
 else  {
                        /* Para todos los demás tipos de empleados tenemos el nivel
                        implícito dentro de la categoría*/

  cNivel[0] = cCveCateg[5];
  cNivel[1] = cCveCateg[6];

                        /*  Para el nivel de carrera tenemos un arreglo adicional para
                        asignarlo de manera consecutiva, y encontrar el correcto en el
                        tabulador */

  if (siTipoEmp == CARRERA)  {
   while ((iNivCarr[iCon] != atoi(cNivel)) && iCon<35)
    iCon++;
   if (iCon == 35) {
    *iNiv = NIVELES_BASE;
    printf("No se encontro el nivel %d\n", atoi(cNivel));
   }
   else
    *iNiv = iCon;
  }
  else
    *iNiv = atoi(cNivel);
 }

 if (iContCat == CATEGO)
  *iNiv = NIVELES_BASE;
 return;

                        /* Si no se encuentra el nivel, entonces se guarda un número máximo,
                        el cual nos indica que hay un problema. */
}

/*************************************************************************/
                                          /* Busca el indice de la fecha */

                        /*  Se busca la fecha dentro de los diferentes arreglos de fechas
                        que cargamos con anterioridad, de tal manera que obtenemos un
                        apuntador, con el cual nos vamos a posicionar en un determinado
                        lugar del arreglo de sueldos */


void vBfecha(DBSMALLINT siTipoEmp, DBINT *iCont, DBINT iFec)  {

 DBINT iCuenta = CERO, *iFechas, iConA = CERO;


                        /* Dependiendo del tipo de empleado, buscamos en arreglos
                        diferentes, puesto que sus sueldos también cambian en fechas
                        diferentes. */


 while ((iCuenta == CERO) && (iConA <= TABS))  {
  if (siTipoEmp == BASE )
   iFechas = &iFec_Base[iConA][0];

  else if (siTipoEmp == CONFIANZA)
```

```
   iFechas = &iFec_Conf[iConA][0];

  else if (siTipoEmp == FUNCIONARIO)
   iFechas = &iFec_Func[iConA][0];

  else if (siTipoEmp == CARRERA)
   iFechas = &iFec_Carr[iConA][0];

  else
   iFechas = &iFec_Asig[iConA][0];

  if(*(iFechas) > iFec) {
   iCuenta = UNO; iConA = 22;
  }
  if((*(iFechas) <= iFec)  && (iFec <= *(iFechas+1)))
   iCuenta = UNO;
  iConA++;
 }

 *iCont = iConA-1;
}
```

Esta es una función que se encarga especialmente del tabulador, pero aún asi necesitamos funciones especializadas, las cuales fueron incluidas en la misma función, ya que solamente aquí son utilizadas.

8. 2 PROGRAMACIÓN DE LOS PROGRAMAS PRINCIPALES.

Para estos se genera un directorio donde se guardan y se compilan todos los programas, los cuales pueden usar una o varias funciones de la librería creada. Tenemos un archivo makefile como el que sigue:

```
LIBDIR    = $(SYBASE)/lib
DBLIBS    = $(PRODIR)/func/libdbfun.a $(GRLDIR)/std/libdbnom.a $(LIBDIR)/libsybd
b.a

INCLUDE   = -I. -I$(INCDIR) -I$(GRLDIR)/std -I$(PRODIR)/func

EXEDIR    = /home2/procfu2/ejec

CFLAGS    = -g -DDEBUG
#CFLAGS    = -xs -O -C
#
#
#
all: caltas cnval cnver crepfu funica cnban cninc cncon cheqdif cresp cnpro cdif
b csusp csusfal crrep crinc crban crca ccont cpaguni cprom cmovpla consulta cnac
t crepsar
aum: cauadm caudoc caufun cretab
clean:
        rm *.c *.h
#
#
#

caltas: $(HEADERS) caltas.c  caltas.h
        cc $(CFLAGS) $(INCLUDE) caltas.c $(DBLIBS) -lnsl -lm -o $(EXEDIR)/caltas

creporte: $(HEADERS) creporte.c  creporte.h
```

```
        cc $(CFLAGS) $(INCLUDE) creporte.c $(DBLIBS) -lnsl -lm -o $(EXEDIR)/crep
orte
crisste: $(HEADERS) crisste.c  crisste.h
```

Uno de los programas principales es el de validación de la Forma Única, el cual se presenta como ejemplo:

```
/*****************************************************************************

   Nombre del programa : cnval
   Nombre del proyecto : Nomina UNAM

   Descripción         : Validación de formas únicas

   Archivos asociados  : rcplaus, rcnomb
   Referencias         :
   Comentarios         :

   Fecha               : 7 Julio 2000

*****************************************************************************/
#include <stdio.h>                    /* Prototipos de E/S estandar        */
#include <stdlib.h>                   /* Prototipos de biblioteca estandar */
#include <signal.h>                   /* Prototipos de manejo de se#ales   */
#include <string.h>                   /* Prototipos de manejo de cadenas   */

#include <sybfront.h>                 /* Constantes para DB-Lib de Sybase  */
#include <sybdb.h>                    /* Estructuras y prots DB-Lib Sybase */
#include <syberror.h>                 /* Constantes para errores de Sybase */

/*---------------------------------------------------------------------------*/

#include "aplstd.h"                   /* Constantes y prototipos estandar  */
#include "apldef.h"                   /* Constantes de la aplicacion       */
#include "aplfun.h"                   /* Constantes y prototipos aplicacion*/
#include "cnval.h"                    /* Constantes y prototipos aplicacion*/

#define NIVELAR_TABULADOR
DBINT iLim, iFechaModifica;
DBCHAR vcUsuario[13];
DBFLT8  nPerPag;
RET sdoret[TABS];

/*---------------------------------------------------------------------------*/
main(int argc, char *argv[])
{

DBPROCESS       *stMov, *stQuery, *stUpdate, *stAux, *stDep, *stAlt,*stAux2;

DBCHAR          cRunCmd[CHARBUF],  cDigNvo[1],
               *cLimp,vcRfcEmp[13],cCategAux[5],cCategCU[7],
               cArch[12];
RETCODE         rcMov, rcQuery, rcConCateg, rcConBasAcad, rcAlt,rcActualizo;
DBDATETIME      sdIniTab;
DBSMALLINT      siNvaPart, siCvePart, siPlaus, siTipProc, siRevSdo,
               siStatusNomb;
DBINT           iNivel, iFecIniPer, iFecFinPer, iEmp, iEmpNomb, iEmpEsp,
               iFecBaja, iFecIniFun, iNoRet, iFecLib, iFecFin,
               iIniTab,iCuenta,iMovBaj,iCveNomb, iBajaFunc,
               iReg,iCorrecto,iConBasAcad,iFecLim,iConOtroNomb, iQuincena;
DBFLT8          nHorFU,fTotHor,mSdoTab, fSueldo;
DBTYPEINFO      stPrec;

FILE            *Bajas;

/*---------------------------------------------------------------------------*/
dbsetversion(DBVERSION_100);
vGetPasswd(argc, argv);
DBSETLENCRYPT(stLog, TRUE);
```

```
stAlt = dbopen(stLog, NULL);
stMov = dbopen(stLog, NULL);
stQuery = dbopen(stLog, NULL);
stUpdate = dbopen(stLog, NULL);
stAux = dbopen(stLog, NULL);
stAux2 = dbopen(stLog, NULL);
stDep = dbopen(stLog, NULL);

dbcmd(stAlt, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbcmd(stMov, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbcmd(stQuery, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbcmd(stUpdate, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbcmd(stAux, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbcmd(stAux2, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbcmd(stDep, "SET TRANSACTION ISOLATION LEVEL 1 ");
dbsqlexec(stAlt);
dbsqlexec(stMov);
dbsqlexec(stQuery);
dbsqlexec(stUpdate);
dbsqlexec(stAux);
dbsqlexec(stAux2);
dbsqlexec(stDep);
/*-------------------------------------------------------------------------*/
                                                  /* Obtener periodo vigente */
rcPer(stQuery, &iFecIniPer, &iFecFinPer,&nPerPag);
iQuincena = (int) nPerPag % 100;

dbcmd(stQuery, "SELECT CONVERT(CHAR(10), getdate(), 112) ");
dbgetint(stQuery, &iFechaModifica);

dbcmd(stQuery, "SELECT user_name() ");
dbgetstring(stQuery, vcUsuario);

/*-------------------------------------------------------------------------*/
              /* Anular formas únicas que contienen movimientos incorrectos */
fprintf(stderr, "\nValidacion y proceso de movimientos de forma unica...\n");

rcAnuFol(stUpdate);

/*-------------------------------------------------------------------------*/
                                          /* Movs que sólo afectan historia */

dbhandletran(stUpdate, BEGIN_TRAN);

dbfcmd(stUpdate,"UPDATE TMOVFUND SET siStatus = %d ", HIST_RETOK);
dbfcmd(stUpdate,"  WHERE siStatus = %d ", SIN_NIVEL);
dbfcmd(stUpdate,"  AND cAfeNom = '%c' ", NO_AFECTA_NOMINA);
dbsqlrun(stUpdate);

dbhandletran(stUpdate, END_TRAN);

dbhandletran(stUpdate, BEGIN_TRAN);

dbfcmd(stUpdate,"UPDATE TMOVFUNH SET siStatus = %d, ", HIST_RETOK);
dbfcmd(stUpdate,"  nNumPerPag = %8.2f ", nPerPag);
dbcmd(stUpdate, "  FROM TMOVFUND, TMOVFUNH ");
dbcmd(stUpdate, "  WHERE TMOVFUND.iNumFol = TMOVFUNH.iNumFol ");
dbcmd(stUpdate, "  AND TMOVFUND.iNumEmp = TMOVFUNH.iNumEmp ");
dbfcmd(stUpdate,"  AND TMOVFUNH.siStatus = %d ", ASIG_RETOK);
dbfcmd(stUpdate,"  AND cAfeNom = '%c' ", NO_AFECTA_NOMINA);
dbsqlrun(stUpdate);
dbhandletran(stUpdate, END_TRAN);
/*-------------------------------------------------------------------------*/
                                                    /* Marca de únicos pagos */
dbhandletran(stUpdate, BEGIN_TRAN);
dbfcmd(stUpdate,"UPDATE TMOVFUND SET siPagUni =  %d ", UNICO_PAGO);
dbfcmd(stUpdate,"  WHERE siStatus = %d ",SIN_NIVEL);
dbfcmd(stUpdate,"  AND tiCveTipMov = %d ",ALTA);
dbfcmd(stUpdate,"  AND (sdFecFinFun IS NOT NULL AND sdFecFinFun <= '%d') ",
                   iFecFinPer);
dbsqlrun(stUpdate);
```

```
dbhandletran(stUpdate, END_TRAN);

dbhandletran(stUpdate, BEGIN_TRAN);

                        /* Marcamos el campo siPagUni con un uno para saber que son pagos
                        únicos */

dbfcmd(stUpdate,"UPDATE TMOVFUND SET siPagUni =  %d ", CERO);
dbfcmd(stUpdate,"  WHERE siStatus = %d ",SIN_NIVEL);
dbfcmd(stUpdate,"  AND tiCveTipMov = %d ",ALTA);
dbfcmd(stUpdate,"  AND (sdFecFinFun IS NULL OR sdFecFinFun > '%d') ",
                  iFecFinPer);
dbsqlrun(stUpdate);

dbhandletran(stUpdate, END_TRAN);

/*-------------------------------------------------------------------------*/
                    /* Seleccionar empleados para insertar en TNOMBRAPREVIO */

dbcmd(stQuery, "SELECT COUNT(*) FROM TMOVFUND ");
dbfcmd(stQuery,"  WHERE siStatus IN (%d,%d,%d,%d,%d,%d,%d) ",
                 SDO_CORRECTO,VERIF_OK,INCOMPATIBLE,COMPATIBLE,RETENIDO_BH,
                 LIBERADO_BCO_HORAS,ALTAS);
dbgetint(stQuery, &iReg);

if (iReg == CERO) {
  dbcmd(stUpdate, "DELETE TNOMBRAPREVIO ");
  dbsqlrun(stUpdate);
}
else {
  dbcmd(stUpdate, "DELETE TNOMBRAPREVIO ");
  dbcmd(stUpdate, "  FROM TNOMBRAPREVIO TN, TMOVFUND TM");
  dbcmd(stUpdate, "  WHERE TN.iNumEmp = TM.iNumEmp ");
  dbfcmd(stUpdate, "  AND TM.siStatus = %d ", SIN_NIVEL);
  dbsqlrun(stUpdate);
}
/*-------------------------------------------------------------------------*/
                                                    /* Copiar nombramientos */
fprintf(stderr, "\nCarga del tabulador ...");

sprintf(cArch,"%s", "cnval.out");
freopen(cArch, "w", stdout);

vAtab(stQuery);                    /* Se carga el tabulador para tenerlo en memoria,
                                   siempre disponible para cualquier consulta */

fprintf(stderr, "\nInsertar nombramientos a la tabla ...");

                        /* Se determina de cuales empleados vamos a copiar sus nombramientos
                        */
dbcmd(stMov, "SELECT DISTINCT iNumEmp FROM TMOVFUND ");
dbfcmd(stMov,"  WHERE siStatus IN (%d) ", SIN_NIVEL);

dbsqlexec(stMov);
while ((rcMov = dbresults(stMov)) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stMov) == SUCCEED)) {

    dbbind(stMov, 1, INTBIND, (DBINT) 0, (BYTE *) &iEmp);

    while (dbnextrow(stMov) != NO_MORE_ROWS) {

                                        /* Copiar nombramientos con plaza */

      dbcmd(stQuery, "SELECT DISTINCT tiCveFunc, tiCveProg, siCveSubProg, ");
      dbcmd(stQuery, "  siCveDep,tiCveSubDep,siCvePart, ");
      dbcmd(stQuery, "  TNOMBRAMIENTOS.cCveCateg, ");
      dbcmd(stQuery, "  iNumPlaza, vcRfcEmp, iNumEmp, cCveTipEjer,");
      dbcmd(stQuery, "  cDigCon, nTotHor, mCompHoras, mSdoPla, ");
      dbcmd(stQuery, "  mMatDidact, cDefPla, cAjuste,");
      dbcmd(stQuery, "  CONVERT(CHAR(10),sdFecIniPla,112), ");
```

```
dbcmd(stQuery, "  CONVERT(CHAR(10),sdFecFinPla,112), ");
dbcmd(stQuery, "  TNOMBRAMIENTOS.siTipoEmp,TNOMBRAMIENTOS.siStatus,");
dbcmd(stQuery, "  siCveNodPag, iCveNodosTra, ");
dbcmd(stQuery, "  CONVERT(CHAR(10),TNOMBRAMIENTOS.sdFechaModifica,112),");
dbcmd(stQuery, "  siProcEsp  FROM TNOMBRAMIENTOS, CCATEGORIAS ");
dbfcmd(stQuery,"  WHERE iNumEmp = %d ", iEmp);
dbcmd(stQuery, "  AND SUBSTRING(TNOMBRAMIENTOS.cCveCateg, 1,5)");
dbcmd(stQuery, "  = SUBSTRING(CCATEGORIAS.cCveCateg,1, 5) ");
dbfcmd(stQuery,"  AND CCATEGORIAS.siStatus IN (%d) ", ACTIVO);
dbcmd(stQuery, "  AND SUBSTRING(TNOMBRAMIENTOS.cCveCateg,1,5) ");
dbcmd(stQuery, "  NOT IN (' PR01')");
dbsqlexec(stQuery);

while ((rcQuery = dbresults(stQuery)) != NO_MORE_RESULTS) {
  if ((rcQuery == SUCCEED) && (DBROWS(stQuery) == SUCCEED)) {

    stvNom[12].stPrec.precision = 4;        stvNom[12].stPrec.scale = 2;
    stvNom[20].stPrec.precision = 6;        stvNom[20].stPrec.scale = 0;
    dbstruct(stQuery, stvNom, 26);

    while (dbnextrow(stQuery) != NO_MORE_ROWS) {

     if (stcNom.siTipoEmp == ASIGNATURA
       && (stcNom.siProcEsp != M21 && stcNom.siProcEsp != Z02
       && stcNom.siProcEsp != D48)) {
        vRsdos(sdoret);                  /*  Se limpia la estructura donde se guardan
                                         los sueldos*/

                /* Si son empleados de asignatura se vuelve a recalcular el sueldo
                para insertarlo en la tabla TNOMBRAPREVIO */

       if (stcNom.siProcEsp == D21) {
         vBsdo(iFecIniPer,"  D4200" ,stcNom.nTotHor*2,
              stcNom.siTipoEmp,sdoret, &iCorrecto);
        }
       else {
         vBsdo(iFecIniPer,stcNom.cCveCateg,stcNom.nTotHor*2,
              stcNom.siTipoEmp,sdoret, &iCorrecto);
       }

                /* De lo contrario se toma el sueldo tal y como viene y se
                multiplica por 2 */

        stcNom.nTotHor *= 2;
        stcNom.mSdoPla = sdoret[0].fSdoTot;
        stcNom.mMatDid = sdoret[0].fMatDid;
     }
     else {
       stcNom.nTotHor *= 2;
       stcNom.mSdoPla *= 2;
       stcNom.mMatDid *= 2;
     }
        /* Se inserta el nombramiento, con la function */

      rcInomb(stUpdate, stcNom.tiCveFunc, stcNom.tiCveProg,
              stcNom.siCveSubProg, stcNom.siCveDep,
              stcNom.tiCveSubDep, stcNom.siCvePart,
              stcNom.cCveCateg, stcNom.iNumPlaza,
              stcNom.vcRfcEmp, CERO, stcNom.iNumEmp,
              stcNom.cCveTipEjer, stcNom.cDigCon,
              stcNom.nTotHor, stcNom.mSdoPla, stcNom.mMatDid,
              stcNom.cDefPla, stcNom.cAjuste,
              stcNom.iFecIniPla, stcNom.iFecFinPla,
              stcNom.siTipoEmp, stcNom.siStatus,
              stcNom.siCveNodPag, stcNom.iCveNodosTra,
              "dbo", stcNom.iFechaModifica,CERO,
              CERO,CERO,CERO);


    }
  }
```

```
      }
      cLimp = stcNom.vcRfcEmp; while (*cLimp) *cLimp++ = ' '; *cLimp = '\0';

    }
  }
}

/*-------------------------------------------------------------------------*/
                                        /* Ejecución de programas */
                                  /*Mensajes para saber que se está haciendo*/

fprintf(stderr, "\nClasificacion y verificacion de movimientos...");
fprintf(stderr, "\nVerificacion de categorias de funcionario...");
fprintf(stderr, "\nRevision y correccion de sueldos especiales...");
fprintf(stderr, "\nRevision y correccion de sueldos...");
fprintf(stderr, "\nRevision y correccion del nivel en tabulador...");

iLim = CERO;
fSueldo = ZERO;

                               /* Proceso por empleado y tipo de movimiento */
                                            /* Orden licencia, baja, alta */

              /* Primero se obtienen los empleados que se van a procesar */


dbcmd(stAlt, "SELECT DISTINCT vcRfcEmp, TMOVFUND.iNumEmp, tiCveTipMov ");
dbcmd(stAlt, "  FROM TMOVFUND, TEMPNOMINA ");
dbcmd(stAlt, "  WHERE TMOVFUND.iNumEmp = TEMPNOMINA.iNumEmp ");
dbfcmd(stAlt,"  AND TMOVFUND.siStatus =  %d ", SIN_NIVEL);
dbcmd(stAlt, "  ORDER BY vcRfcEmp, tiCveTipMov DESC ");
dbsqlexec(stAlt);

while ((rcAlt = dbresults(stAlt)) != NO_MORE_RESULTS) {
  if ((rcAlt == SUCCEED) && (DBROWS(stAlt) == SUCCEED)) {

    dbstruct(stAlt, stvAlt, 3);

    while (dbnextrow(stAlt) != NO_MORE_ROWS) {

      switch(stcAlt.tiCveTipMov) {

/*-------------------------------------------------------------------------*/
        case ALTA:
        case BAJA:
        case LICENCIA_SIN_SUELDO:

          dbcmd(stMov, "SELECT DISTINCT iNumFol, nPer, siCveNodo, siSerial, ");
          dbcmd(stMov, "  iNumEmp, tiCveTipMov, iNumPlaza, tiCveFunc, ");
          dbcmd(stMov, "  tiCveProg, siCveSubProg, siCveDep, ");
          dbcmd(stMov, "  tiCveSubDep, siCvePart, cConacyt, cCveTipEjer, ");
          dbcmd(stMov, "  cDigCon, TMOVFUND.cCveCateg, mSdo, ");
          dbcmd(stMov, "  cAfeNom, nHorTeo, nHorPra,  ");
          dbcmd(stMov, "  CONVERT(CHAR(10), sdFecIniFun, 112), ");
          dbcmd(stMov, "  CONVERT(CHAR(10), sdFecFinFun, 112), ");
          dbcmd(stMov, "  cDefFun, cCoaCoc, siPagUni, ");
          dbcmd(stMov, "  TMOVFUND.siTipoEmp, TMOVFUND.siStatus, ");
          dbcmd(stMov, "  siStatusSip, siCveNodPag, iCveNodosTra, ");
          dbcmd(stMov, "  siTipProc, cCatRef, siProcEsp, siEmpRef, ");
          dbcmd(stMov, "  SUBSTRING(TMOVFUND.cCveCateg,6,2), cConPza ");
          dbcmd(stMov, "  FROM TMOVFUND, CCATEGORIAS ");
          dbcmd(stMov, "  WHERE SUBSTRING(TMOVFUND.cCveCateg, 1,5)");
          dbcmd(stMov, "  = SUBSTRING(CCATEGORIAS.cCveCateg,1, 5) ");
         dbfcmd(stMov,"  AND CCATEGORIAS.siStatus IN (%d) ", ACTIVO);
          dbfcmd(stMov,"  AND TMOVFUND.siStatus =  %d ", SIN_NIVEL);
          dbfcmd(stMov,"  AND TMOVFUND.iNumEmp =  %d ", stcAlt.iNumEmp);

                               /*   Orden zonas geograficas, complementarios, */
                               /*        funcionarios con base academica, tal */
                               /*       y como viene, categorias en tabulador */
          if (stcAlt.tiCveTipMov == BAJA) {
```

```
  dbfcmd(stMov,"  AND TMOVFUND.tiCveTipMov IN (%d) ", BAJA);
  dbcmd(stMov, "  ORDER BY siOrdProc ASC ");
}
if (stcAlt.tiCveTipMov == LICENCIA_SIN_SUELDO) {
  dbfcmd(stMov,"  AND TMOVFUND.tiCveTipMov IN (%d) ",
                 LICENCIA_SIN_SUELDO);
  dbcmd(stMov, "  ORDER BY siOrdProc ASC ");
}

              /* Orden categorías en tabulador, tal y como viene, */
              /* funcionarios con base académica, complementarios,*/
              /* zonas geográficas                                */
if (stcAlt.tiCveTipMov == ALTA) {
  dbfcmd(stMov,"  AND TMOVFUND.tiCveTipMov IN(%d) ", ALTA);
  dbcmd(stMov, "  ORDER BY siOrdProc DESC ");
}
dbsqlexec(stMov);

while ((rcMov = dbresults(stMov)) != NO_MORE_RESULTS) {
  if ((rcMov == SUCCEED) && (DBROWS(stMov) == SUCCEED)) {

    stvMov[2].stPrec.precision = 6;   stvMov[2].stPrec.scale = 0;
    stvMov[20].stPrec.precision = 4;  stvMov[20].stPrec.scale = 2;
    stvMov[21].stPrec.precision = 4;  stvMov[21].stPrec.scale = 2;

    dbstruct(stMov, stvMov, 37);

    while (dbnextrow(stMov) != NO_MORE_ROWS) {

     iConBasAcad = siRevSdo = CERO;
     rcConBasAcad = rcActualizo = FAIL;

                                  /* Verificar categoria CA, CF */

      if (strncmp(stcMov.cCveCateg, " CU",3) == EQUAL)
       rcVsdocf(stUpdate, iFecIniPer, iFecFinPer);


             /* Si son funcionarios se llama a la función rcApartasig, para
             determinar que partida le corresponde, lo cual se hace en relación a
             los nombramientos académicos que tenga */


      if (stcMov.siTipoEmp == FUNCIONARIO
        && ((strncmp(stcMov.cCveCateg, "  Z",3) != EQUAL)
       && strncmp(stcMov.cCveCateg, "  M",3) != EQUAL
       && strncmp(stcMov.cCveCateg, " CP",3) != EQUAL)
       && (stcMov.siCvePart != 181 && stcMov.siCvePart != 182))
        rcApartasig(stQuery,stUpdate,TMOVFUND);

             /* Verificación de horas  para poder dar de alta el nombramiento con
             las horas correctas*/

     if ((stcMov.siTipoEmp == CARRERA
       || stcMov.siTipoEmp == CONFIANZA
       || stcMov.siTipoEmp == BASE
       || (stcMov.siTipoEmp == FUNCIONARIO
       && (strstr(stcMov.cCveCateg," CI")
       || strstr(stcMov.cCveCateg," CN")
       || strstr(stcMov.cCveCateg," CC"))))
       && (strncmp(stcMov.cCveCateg, "  D2142", 7) != EQUAL)) {

        nHorFU = stcMov.nHorPra + stcMov.nHorTeo;
        rcHoras(stQuery,stcMov.cCveCateg,stcMov.siCvePart,
              stcMov.siTipoEmp, &nHorFU);

       stcMov.nHorTeo = nHorFU;
       stcMov.nHorPra = ZERO;
      }
```

```
                        /* Se llama a esta función con la finalidad de saber si la Forma
                        Única tiene algún dato inconsistente, que nos pudiera causar algún
                        problema al momento de dar de alta o baja el nombramiento */

                rcplaus(stQuery, stUpdate, &siPlaus, stcAlt.vcRfcEmp);

/*.....................................................................*/
                           /* Excluir categorías que no se deben de corregir */



                        /* Una vez que se ha verificado lo anterior, procedemos a verificar
                        que el sueldo sea correcto, pero para esto algunas categorías se
                        discriminan, ya que son movimientos especiales, a los cuales debemos
                        dejar el sueldo que trae independientemente del tabulador*/

                        /* Existen únicos pagos que se dejan tal y como vienen */

                if (strncmp(stcMov.cDefFun,"E",1) == EQUAL
                 && stcMov.siPagUni == UNICO_PAGO) {
                  dbhandletran(stUpdate, BEGIN_TRAN);

                  dbcmd(stUpdate, "UPDATE TMOVFUND ");
                  dbfcmd(stUpdate,"  SET cCatRef = '%.7s', cCveCateg = '%.7s'",
                                  stcMov.cCveCateg, CP01STRING);
                  dbfcmd(stUpdate,"  WHERE iNumEmp = %d", stcMov.iNumEmp);
                  dbfcmd(stUpdate,"  AND iNumFol = %d", stcMov.iNumFol);
                  dbfcmd(stUpdate,"  AND nPer = %f", stcMov.nPer);
                  dbfcmd(stUpdate,"  AND siSerial = %d", stcMov.siSerial);
                  dbcmd(stUpdate, "  AND cCatRef IS NULL ");
                  dbsqlrun(stUpdate);

                  dbhandletran(stUpdate, END_TRAN);

                  stcMov.siProcEsp = CP01;
                }
                if (strncmp(stcMov.cDefFun,"E",1) == EQUAL
                 && (stcMov.tiCveTipMov == BAJA
                 || stcMov.tiCveTipMov == LICENCIA_SIN_SUELDO)
                 && stcMov.siTipoEmp != CARRERA && stcMov.siProcEsp != CP01
                 && stcMov.siProcEsp != CP02) {
                  dbhandletran(stUpdate, BEGIN_TRAN);

                  dbcmd(stUpdate, "UPDATE TMOVFUND ");
                  dbfcmd(stUpdate,"  SET cCatRef = '%.7s', cCveCateg = '%.7s'",
                                  stcMov.cCveCateg, CP03STRING);
                  dbfcmd(stUpdate,"  WHERE iNumEmp = %d", stcMov.iNumEmp);
                  dbfcmd(stUpdate,"  AND iNumFol = %d", stcMov.iNumFol);
                  dbfcmd(stUpdate,"  AND nPer = %f", stcMov.nPer);
                  dbfcmd(stUpdate,"  AND siSerial = %d", stcMov.siSerial);
                  dbcmd(stUpdate, "  AND cCatRef IS NULL ");
                  dbsqlrun(stUpdate);

                  dbhandletran(stUpdate, END_TRAN);

                  stcMov.siProcEsp = CP03;
                }

                        /* Movimientos de carrera que vienen marcados como especiales, se
                        llama a una función especial para el cálculo, ya que se requieren
                        otras condiciones, para lo cual le ponemos una categoría diferente,
                        que nos va a ayudar a distinguir el proceso a seguir */


                if (strncmp(stcMov.cDefFun,"E",1) == EQUAL
                 && stcMov.siTipoEmp == CARRERA && stcMov.siProcEsp != CP01) {

                  dbhandletran(stUpdate, BEGIN_TRAN);

                  dbcmd(stUpdate, "UPDATE TMOVFUND ");
```

```
                dbfcmd(stUpdate,"  SET cCatRef = '%.7s', cCveCateg = '%.7s'",
                                stcMov.cCveCateg, CP02STRING);
                dbfcmd(stUpdate,"  WHERE iNumEmp = %d", stcMov.iNumEmp);
                dbfcmd(stUpdate,"  AND iNumFol = %d", stcMov.iNumFol);
                dbfcmd(stUpdate,"  AND nPer = %f", stcMov.nPer);
                dbfcmd(stUpdate,"  AND siSerial = %d", stcMov.siSerial);
                dbcmd(stUpdate, "  AND cCatRef IS NULL ");
                dbsqlrun(stUpdate);

                dbhandletran(stUpdate, END_TRAN);

                stcMov.siProcEsp = CP02;
               if (strncmp(stcMov.cCveCateg, " CP0200", 7) != EQUAL)
                 strncpy(stcMov.cCatRef, stcMov.cCveCateg, 7);
              }
/*.....................................................................*/
                                          /* Condiciones especiales */

                    /*  Si son funcionarios, pero no son zonas geográficas de
                    funcionarios, se debe investigar si tienen base académica, es decir,
                    tienen nombramientos de asignatura o carrera, para lo cual
                    dependiendo del tipo de movimiento, usamos TNOMBRAPREVIO O
                    TNOMBRAMIENTOS */


              if (stcMov.siProcEsp == Z06 || stcMov.siProcEsp == CA
                || stcMov.siProcEsp == CM || stcMov.siProcEsp == CD
                || stcMov.siProcEsp == RC || stcMov.siProcEsp == CH
               || stcMov.siProcEsp == M28) {

                switch (stcMov.tiCveTipMov) {
                  case ALTA:
                    dbcmd(stQuery, "SELECT DISTINCT iNumEmp ");
                    dbcmd(stQuery, "  FROM TNOMBRAPREVIO ");
                    dbfcmd(stQuery,"  WHERE iNumEmp = %d ", stcMov.iNumEmp);
                    dbcmd(stQuery, "  AND SUBSTRING(cCveCateg,1,3) ");
                    dbcmd(stQuery, "  IN (' DH',' EH','  D','  I')");
                   dbfcmd(stQuery,"  AND siPagUni != %d ", UNICO_PAGO),
                    dbgetint(stQuery, &iConBasAcad);
                  break;

                  case BAJA:
                  case LICENCIA_SIN_SUELDO:
                    dbcmd(stQuery, "SELECT DISTINCT iNumEmp ");
                    dbcmd(stQuery, "  FROM TNOMBRAMIENTOS ");
                    dbfcmd(stQuery,"  WHERE iNumEmp = %d ", stcMov.iNumEmp);
                    dbcmd(stQuery, "  AND SUBSTRING(cCveCateg,1,3) ");
                    dbcmd(stQuery, "  IN (' DH',' EH','  D','  I')");
                    dbgetint(stQuery, &iConBasAcad);
                  break;

                  default:
                    vIcond(stQuery, stcMov.tiCveTipMov,
                           stcMov.siTipProc, stcMov.siTipoEmp,
                           stcMov.iNumEmp, stcMov.iNumFol,
                           stcMov.siSerial, INVALID_CONDITION);
                  break;
                 }
                if (stcMov.iNumEmp == iConBasAcad)
                  rcConBasAcad = SUCCEED;
              }
/*.....................................................................*/
                                    /* Excluir categorias sin validacion */
              switch (stcMov.siProcEsp) {
               case A04:          /*  Estas categorías, si son nivel 40,41, */
               case A05:          /*  42 o 43, se deja el sueldo tal y como */
               case A08:          /*  viene, de lo contrario se busca en el */
               case A12:          /*  tabulador  */
               case AA02:
               case ET16:
               case ET17:
```

```
      case ET19:
      case ET29:
case O02:
       if (stcMov.siNivel == 40 || stcMov.siNivel == 41
         || stcMov.siNivel == 42 || stcMov.siNivel == 43) {

        if (stcMov.tiCveTipMov == ALTA)
           siRevSdo = SUCCEED;
         else
          siRevSdo = rcVesaum(stQuery, stUpdate, iFecLim,
                              VALIDACION, stcMov.siTipoEmp,
                              stcAlt.vcRfcEmp, iFecIniPer,
                              ACTUALIZA, sdoret);
       }
       else

        siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                          VALIDACION, stcMov.siTipoEmp,
                          stcAlt.vcRfcEmp, CERO);
      break;

      case P01:          /*Aquí sí es una categoria en   */
      case P14:          /* particular y determinados niveles    */
      case P64:          /* se deja como está, si no se busca el */
      case P80:          /* tabulador */
      case P82:
      case P90:
       if (stcMov.siProcEsp == P64 && (stcMov.siNivel == 40
         || stcMov.siNivel == 41 || stcMov.siNivel == 42
         || stcMov.siNivel == 43) && stcMov.tiCveTipMov == BAJA)
          siRevSdo = SUCCEED;
        else
         siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                          VALIDACION, stcMov.siTipoEmp,
                          stcAlt.vcRfcEmp, CERO);
      break;
                                    /* Categorías sin tabulador */
      case M01:          /* tiene una función especial para buscar
                            el sueldo */
        rcVIm01(stQuery, stUpdate, stDep, stcMov.iFecIniFun,
                iFecLim, VALIDACION, stcMov.tiCveTipMov,
                stcMov.iNumEmp, stcMov.siProcEsp,
                NO_ACTUALIZA, stcMov.siTipoEmp,
                stcMov.mSdo, stcAlt.vcRfcEmp, CERO,
               &fSueldo, sdoret);

        rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
                stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
                stcMov.siSerial, stcMov.tiCveTipMov,
                stcMov.siPagUni, &siRevSdo);

        if (stcMov.tiCveTipMov == ALTA)  {
          iNivel = sdoret[0].fSdoTot / FACTOR_1000;
          sprintf(stcMov.cCveCateg,"%.5s%.2ld",
                  stcMov.cCveCateg, iNivel);
        }
     break;

      case M03:          /* Únicamente para las bajas, se verifica */
      case M21:          /* el sueldo, por si tuvo aumento */
      case M22:          /* para las altas se deja como está */
      case M23:
      case M24:
       if (stcMov.tiCveTipMov == BAJA
         || stcMov.tiCveTipMov == LICENCIA_SIN_SUELDO) {

        siRevSdo = rcVesaum(stQuery, stUpdate, iFecLim,
                            VALIDACION, stcMov.siTipoEmp,
                           stcAlt.vcRfcEmp, iFecIniPer,
                           ACTUALIZA, sdoret);
        }
```

```
      /* funcion de categorias con aumento y sin tabulador */
    if (stcMov.tiCveTipMov == ALTA)  {
      iNivel = stcMov.mSdo / FACTOR_1000;
      sprintf(stcMov.cCveCateg,"%.5s%.2ld",
            stcMov.cCveCateg, iNivel);
      siRevSdo = SUCCEED;
    }
  break;



  case AH:

      /* categorías especiales sin aumento y sin tabulador */
       /* se deja tal y como viene en la Forma Única y sólo en el caso de
       las bajas, verificamos si tuvo aumento. */

    if (stcMov.siCvePart == 182 && (stcMov.tiCveTipMov == BAJA
     || stcMov.tiCveTipMov == LICENCIA_SIN_SUELDO)) {
     siRevSdo = rcVesaum(stQuery, stUpdate, iFecLim,
                         VALIDACION, CARRERA, stcAlt.vcRfcEmp, iFecIniPer,
           ACTUALIZA, sdoret);
    }
    if (stcMov.siCvePart == 181)
     siRevSdo = SUCCEED;

    if (stcMov.tiCveTipMov == ALTA)  {
      iNivel = stcMov.mSdo / FACTOR_1000;
      sprintf(stcMov.cCveCateg,"%.5s%.2ld",
            stcMov.cCveCateg, iNivel);
      siRevSdo = SUCCEED;
    }
  break;


  case M02:
  case H01:
  case H02:
  case DH01:
  case PH56:
  case PH57:
  case PH64:

      /* categorías especiales sin aumento y sin tabulador */
      /* Se les deja el sueldo que trae la Forma Única */

    if (stcMov.tiCveTipMov == ALTA)  {
      iNivel = stcMov.mSdo / FACTOR_1000;
      sprintf(stcMov.cCveCateg,"%.5s%.2ld",
            stcMov.cCveCateg, iNivel);
    }
   siRevSdo = SUCCEED;
  break;

  case CE93:         /* se deja el sueldo que trae */
   siRevSdo = SUCCEED;
  break;

 case CP01:
   siRevSdo = SUCCEED;
  break;

 case CP02:    /* Categorías que fueron marcadas anteriormente. */
               /* Se usa una función especial */

    rcVIporc(stQuery, stUpdate, stDep, stcMov.iFecIniFun,
           iFecLim, VALIDACION, stcMov.tiCveTipMov,
             stcMov.iNumEmp, NO_ACTUALIZA, stcAlt.vcRfcEmp,
             CERO, &fSueldo, sdoret);
```

```
      strncpy(stcMov.cCveCateg,stcMov.cCatRef,7);

      rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
            stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
            stcMov.siSerial, stcMov.tiCveTipMov,
            stcMov.siPagUni, &siRevSdo);
    break;

    case CP03:
      siRevSdo = SUCCEED;
     break;

     case M04:
     case M05:
     case M06:
     case M07:
     case M08:
     case M09:
     case M10:
     case M11:
     case M12:
     case M13:
     case M14:
     case M15:
     case M16:
     case M25:
     case M26:
     case M27:
     case Z02:
     case Z03:
     case Z04:
     case Z05:
     case Z07:

  /* Todas estas categorías se calculan con una función especial */

      rcVIesp(stQuery, stUpdate, stDep,
            stcMov.iFecIniFun, iFecLim,
              VALIDACION, stcMov.tiCveTipMov,
              stcMov.iNumEmp, stcMov.siProcEsp,
            NO_ACTUALIZA, stcAlt.vcRfcEmp,
              CERO, &fSueldo, sdoret);


      rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
            stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
            stcMov.siSerial, stcMov.tiCveTipMov,
            stcMov.siPagUni, &siRevSdo);

      if (stcMov.tiCveTipMov == ALTA)  {
        iNivel = fSueldo / FACTOR_1000;
        sprintf(stcMov.cCveCateg,"%.5s%.2ld",
       stcMov.cCveCateg, iNivel);
      }
     break;

     case M29:          /* Se deja el sueldo tal y como viene */
      siRevSdo = SUCCEED;
     break;

     case D48:          /* se sigue un proceso especial para el cálculo*/

       if (stcMov.tiCveTipMov == ALTA)
         rcAcomp(stQuery,stUpdate,stcMov.iNumEmp);

       rcVId48(stQuery, stUpdate, stDep, stcMov.iFecIniFun,
             iFecLim, VALIDACION, stcMov.tiCveTipMov,
             stcMov.iNumEmp, stcMov.siProcEsp,
             NO_ACTUALIZA, stcAlt.vcRfcEmp,
             stcMov.cCveCateg, CERO, &fSueldo, sdoret);
```

```
      rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
            stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
            stcMov.siSerial, stcMov.tiCveTipMov,
            stcMov.siPagUni, &siRevSdo);


    break;

    case Z01:
    case Z06:
    case Z09:

/*  Estas categorías tienen uan función especial  */

   case M28:
      rcVIespimp(stQuery, stUpdate, stDep, stAux,
               stAux2,stcMov.iFecIniFun, iFecLim,
                 VALIDACION, stcMov.tiCveTipMov,
                 stcMov.iNumEmp, stcMov.siProcEsp, NO_ACTUALIZA,
               stcAlt.vcRfcEmp, rcConBasAcad,
                 CERO, &fSueldo, sdoret);


      rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
            stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
            stcMov.siSerial, stcMov.tiCveTipMov,
            stcMov.siPagUni, &siRevSdo);

      if (stcMov.tiCveTipMov == ALTA)  {
        iNivel = fSueldo / FACTOR_1000;
        sprintf(stcMov.cCveCateg,"%.5s%.2ld",
       stcMov.cCveCateg, iNivel);
      }

    break;

    case DH41:
    case DH42:
    case EH41:
    case EH42
                /* Se maneja como nombramiento de asignatura */
                /* se manda el empleado de referencia para   */
                /* calcularlos como tal.*/

     siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                       VALIDACION, stcMov.siEmpRef,
                       stcAlt.vcRfcEmp, CERO);
    break;

    case D21:

          /* Si es un nivel 42, entonces se calcula como una categoría de
          asignatura, por lo tanto se tiene que manejar aparte */


     if (stcMov.siNivel == 42) {
       strcpy(stcMov.cCveCateg, "  D4200");

        siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                       VALIDACION, ASIGNATURA,
                       stcAlt.vcRfcEmp, CERO);
        strcpy(stcMov.cCveCateg, "  D2142");
       }
     else {
       siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                       VALIDACION, stcMov.siTipoEmp,
                       stcAlt.vcRfcEmp, CERO);
       }
   break;
```

```
case DH:
case EH:
 siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                   VALIDACION, stcMov.siEmpRef,
                   stcAlt.vcRfcEmp, CERO);
break;

case CA:
case CH:
case RC:
case CV:
     /* Para las categorías de funcionarios se tienen funciones
     especiales, además algunas categorías se tienen que cambiar de
     nombre para efectos de búsqueda en el tabulador, y algunas otras no
     se ajustan como normalmente ocurre con los funcionarios */


 vOsubstr(cCategAux, stcMov.cCveCateg, 4, 4);
  if (stcMov.siProcEsp == RC)
    sprintf(cCategCU,"%.3s%.4s", " CB", cCategAux);
  else
    sprintf(cCategCU,"%.3s%.4s", "CU", cCategAux);

  if (rcConBasAcad == SUCCEED) {
    if (stcMov.tiCveTipMov == ALTA)
      rcAcomp(stQuery,stUpdate,stcMov.iNumEmp);

    rcObacad(stQuery, stUpdate, stDep, stAux,
           stcMov.iFecIniFun, stcMov.iFecFinFun,
           VALIDACION, stcMov.tiCveTipMov,
           stcAlt.iNumEmp, NO_ACTUALIZA, NORMAL,
           CERO, &fSueldo, sdoret);

    rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
          stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
          stcMov.siSerial, stcMov.tiCveTipMov,
          stcMov.siPagUni, &siRevSdo);
  }
  if (rcConBasAcad == FAIL)
    siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                      VALIDACION, stcMov.siEmpRef,
                      stcAlt.vcRfcEmp, CERO);
break;

case CM:
case CD:

 vOsubstr(cCategAux, stcMov.cCveCateg, 4, 4);
 sprintf(cCategCU,"%.3s%.4s", " CU", cCategAux);

  if (rcConBasAcad == SUCCEED
    && (strncmp(stcMov.cDefFun,"N",1) != EQUAL)) {

    if (stcMov.tiCveTipMov == ALTA)
      rcAcomp(stQuery,stUpdate,stcMov.iNumEmp);

    rcObacad(stQuery, stUpdate, stDep, stAux,
           stcMov.iFecIniFun, stcMov.iFecFinFun,
           VALIDACION, stcMov.tiCveTipMov,
           stcAlt.iNumEmp, NO_ACTUALIZA, NORMAL,
           CERO, &fSueldo, sdoret);

    rcVsdos(stUpdate, sdoret[0].fSdoTot, stcMov.mSdo,
          stcMov.iNumFol, stcMov.iNumEmp, stcMov.nPer,
          stcMov.siSerial, stcMov.tiCveTipMov,
          stcMov.siPagUni, &siRevSdo);

  }
  else
    siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                      VALIDACION, stcMov.siTipoEmp,
```

```
                                    stcAlt.vcRfcEmp, CERO);
                  break;

                       /* Todos las categorías que no entran en ningún caso anterior, se
                       les calcula el sueldo de acuerdo a como está en el tabulador, para
                       lo cual tenemos una función general, para todos los tipos de
                       empleado. */


                  default:
                   siRevSdo = rcVnor(stQuery, stUpdate, iFecLim,
                                     VALIDACION, stcMov.siTipoEmp,
                                     stcAlt.vcRfcEmp, CERO);
                  break;

                 }
                if ((siRevSdo == SUCCEED) && (siPlaus == SUCCEED)
                  && (stcMov.siTipProc != ZERO) && stcMov.siTipoEmp == CARRERA
                  && stcMov.siTipProc == C7) {

                       /*  Aqui se verifican las bajas de carrera, que podrían ser bajas
                       con porcentaje, de tal manera que nos mande un aviso para marcar la
                       Forma Única como corresponde y se calcule el sueldo de manera
                       correcta */


                  dbcmd(stQuery, "SELECT iNumEmp FROM TEMPESP");
                  dbfcmd(stQuery,"  WHERE iNumEmp = %d", stcMov.iNumEmp);
                  dbfcmd(stQuery,"  AND tiCveTipMov = %d", stcMov.tiCveTipMov);
                  dbfcmd(stQuery,"  AND siStatus = %d", ACTIVO);
                  dbgetint(stQuery, &iEmpEsp);

                  if (stcMov.iNumEmp == iEmpEsp) {
                    rcUpdErr(stUpdate, stcMov.iNumFol, EMP_PORCENTAJE,
                              stcMov.tiCveTipMov, stcMov.siSerial, ZERO);
                   }
                 }
/*.....................................................................*/


                       /*  Si todo es correcto, entonces el movimiento se actualiza al
                       estado correcto  de lo contrario, en su momento se mando el mensaje
                       de error correspondiente, y la Forma Única se queda en estado
                       incorrecto */


                if ((siRevSdo == SUCCEED) && (siPlaus == SUCCEED)
                  && (stcMov.siTipProc != ZERO)) {

                                    /* Actualizacion de nombramientos */
                 rcActualizo = rcnomb(stMov,stQuery,stUpdate,stDep,
                                   stAux,stAux2,iFecIniPer,
                                     iFecFinPer,iFecLim,iFechaModifica,
                                   vcUsuario, rcConBasAcad, sdoret);
                  if (rcActualizo == SUCCEED) {

                    dbhandletran(stUpdate, BEGIN_TRAN);

                    dbfcmd(stUpdate,"UPDATE TMOVFUND SET siStatus = %d,  ",
                                     SDO_CORRECTO);
                    dbfcmd(stUpdate,"  siTipProc = %d  ", stcMov.siTipProc);
                    dbfcmd(stUpdate,"  WHERE iNumFol = %d AND nPer = %f ",
                                     stcMov.iNumFol, stcMov.nPer);
                    dbfcmd(stUpdate,"  AND siCveNodo = %d AND siSerial = %d ",
                                     stcMov.siCveNodo, stcMov.siSerial);

                    dbsqlrun(stUpdate);

                    dbhandletran(stUpdate, END_TRAN);
                  }
                 }
```

```
              cLimp = stcMov.cConacyt;
             while (*cLimp) *cLimp++ = ' ';
             *cLimp = '\0';
            }   /* dbnextrow */
          }     /* DBROWS */
         }      /* dbresults */
        break;

                    /*  Inicialmente se verificaron bajas, altas y licencia sin sueldo,
                    pero las licencias con sueldo son considerados movimientos
                    incorrectos, por lo cual se marcan como erróneas */


        case LICENCIA_CON_SUELDO:

         dbcmd(stQuery, "SELECT DISTINCT iNumFol, siSerial ");
         dbcmd(stQuery, "  FROM TMOVFUND ");
         dbfcmd(stQuery,"  WHERE siStatus = %d ", SIN_NIVEL);
         dbfcmd(stQuery,"  AND iNumEmp = %d ", stcAlt.iNumEmp);
         dbfcmd(stQuery,"  AND tiCveTipMov = %d ", LICENCIA_CON_SUELDO);
        dbgetarow(stQuery, stvLic,2);

         rcUpdErr(stUpdate, stcLic.iNumFol, LIC_CONSDO_AFE,
                LICENCIA_CON_SUELDO, stcLic.siSerial, ZERO);

        break;

                    /* En el default se manda un mensaje de error, por si llegara  a
                    presentarse una condición inesperada y que no se encuentra dentro de
                    lo normal */

        default:
          vIcond(stQuery, stcAlt.tiCveTipMov,
                 stcMov.siTipProc, stcMov.siTipoEmp,
                 stcAlt.iNumEmp, stcMov.iNumFol,
                 stcMov.siSerial, INVALID_CONDITION);
        break;
      }
     cLimp = stcAlt.vcRfcEmp; while (*cLimp) *cLimp++ = ' '; *cLimp = '\0';

    }   /* dbnextrow */
  }     /* DBROWS */
}       /* dbresults */

dbexit();                                       /* Cerrar conexión y salir */
return(STDEXIT);

}
```

8. 3 INTEGRACIÓN DEL SUBSISTEMA

El subsistema quedo integrado de la siguiente manera:

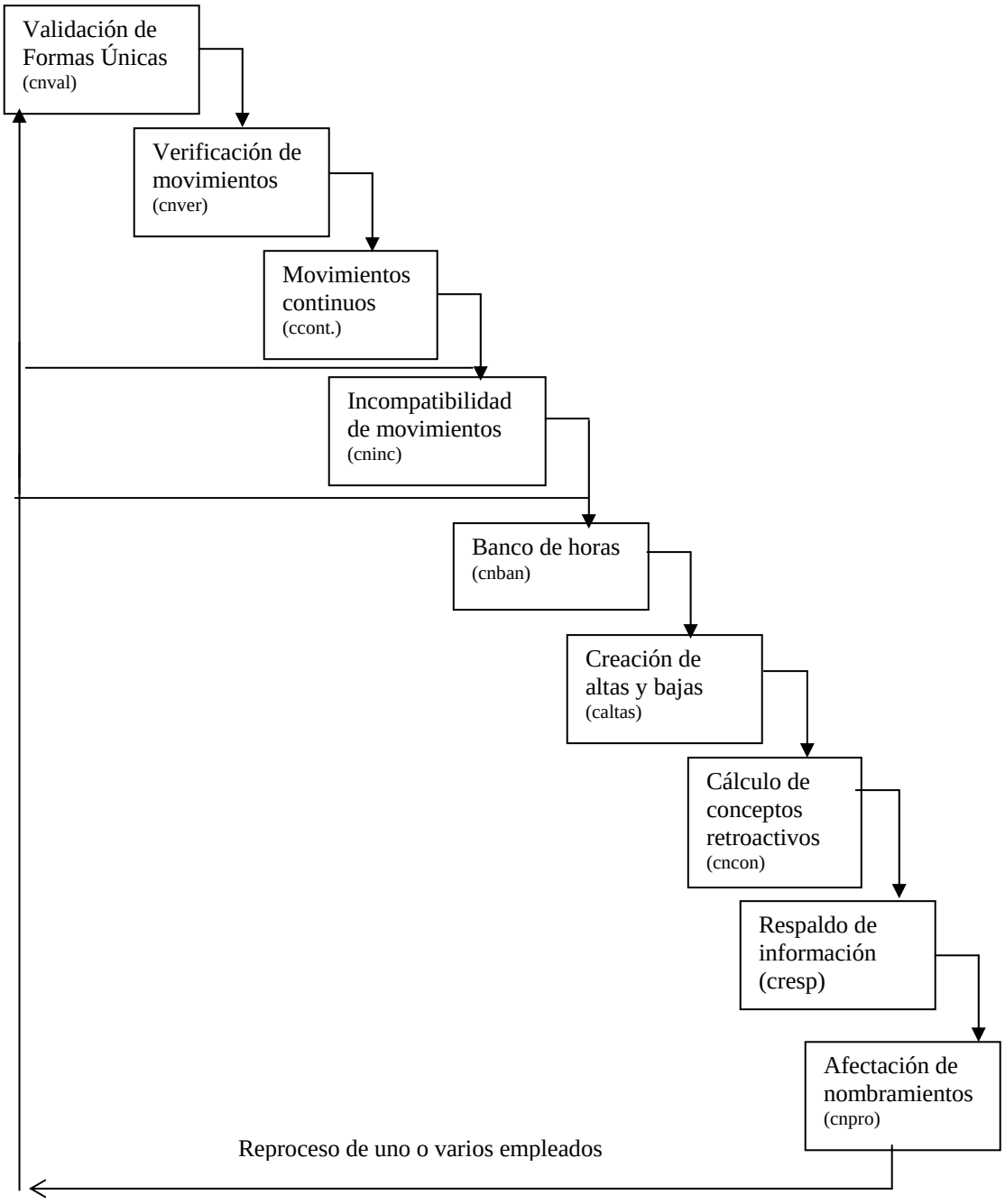


Figura 8.1Esquema general del proceso.

La figura 8.1 presenta el orden en el cual se van ejecutando los programas que conforman este subsistema y como puede observarse, después de ciertos procesos es posible volver a comenzar.

Los estados que se manejan durante el proceso son los siguientes:

35 LIBERADO DEL SIFU (Sistema Integral de Forma Única). Quiere decir que las Formas Únicas que se encuentran en este estado, son las que se van a procesar para la quincena, además del estado, en la tabla TMOVFUNH (encabezado) en el campo sdFecNom, tiene la fecha de inicio de periodo de la quincena que se va a procesar, debido a que cuando estamos procesando, el SIFU sigue liberando Formas Únicas, las cuales serán para la siguiente quincena, de esta manera garantizamos que únicamente se van a procesar las Formas Únicas de la quincena activa.

37 CANCELADO. Son Formas Únicas que han sido canceladas por el personal que se encarga de revisar los reportes de incompatibles y altas y bajas.

39 En este estado se dejan las Formas Únicas que tienen errores, pero que fueron generadas por el SIP.

45 Son Formas Únicas que están en proceso, pero la información que contienen puede tener uno o varios errores, por lo tanto es un estado incorrecto.

70 Están en proceso, ya han sido validadas, por lo tanto es estado correcto.

80 Formas Únicas que no tienen inconsistencias.

90 Formas Únicas que ya fueron revisadas, y además no presentan ninguna condición de incompatibilidad.

95 Forma Única correcta, pero con alguna condición de incompatibilidad.

97 Son Formas Únicas retenidas por el banco de horas, en quincenas anteriores.

107 Formas Únicas retenidas por banco de horas en la quincena en proceso.

110 Son Formas Únicas retenidas por incompatibles.

120 Son Formas Únicas que no son incluidas en el reporte de banco de horas.

130 Son Formas Únicas que ya pasaron el proceso de altas.

150 Nos indica que la Forma Única ya fue considerada para la actualización por empleado.

150 Ya se actualizaron los días trabajados en el año actual y anterior.

160 Ya se realizó la actualización por tipo de movimiento (A2, A3, C6, C7, C4).

192 Son movimientos que fueron “eliminados”, por así decirlo, porque ya no van a afectar a la nómina y el área correspondiente toma esa decisión.

200 Formas Únicas que ya fueron procesadas, se dice que es un estado histórico, donde se quedan todas las Formas Únicas procesadas.

Todos los programas se deben ejecutar de manera continua, es decir, uno después de otro, a excepción del banco de horas y el reporte de incompatibles, porque sólo generan reportes; estos programas son los únicos que se podrían omitir en un momento dado, aunque si debemos actualizar el estado de las Formas Únicas para que puedan ingresar en el siguiente proceso.

Para el programa que realiza el cálculo de conceptos también se tienen ciertas condiciones, sin las cuales no puede ser ejecutado, tales como, cambio de algunas fechas, proceso que realiza otro subsistema, que es el de Movimientos Suspendidos.

9. IMPLEMENTACIÓN, INSTALACIÓN Y PRUEBAS.

Para que los programas puedan funcionar, debemos tener un cierto ambiente en unix, en este sistema operativo podemos delimitar cuáles son los directorios donde se encuentran las librerías y los programas que se van a ejecutar

```
# Variables Unix
COMPILER=/opt/SUNWspro
HOME=/home2/procfu2/trab/mari
TERM=vt100

# Opciones para vi
# autoindent, autowrite backup, number lines, show match, show mode, shift width
# EXINIT="set ai aw nu sm smd sw=4"
EXINIT="set ai aw sm smd sw=4 directory=/tmp"
export COMPILER HOME TERM EXINIT

# Variables Sybase
SYBASE=/sybase
DSQUERY=ULTRA2
LOGNAME=procfu
export PASSWORD SYBASE DSQUERY

# Ambiente grafico
OPENWINHOME=/usr/openwin
export OPENWINHOME
#PS1='$PWD% '
PS1='mary-> '
export PS1 OPENWINHOME DISPLAY

# Paths
PATH=/usr/bin:/opt/usr/local/bin:/home2/gral/impresion:/usr/sbin:$COMPILER/bin:$
SYBASE/bin:/home2/gral/shells:/home2/procfu2/utils:/home2/procfu2/ejec:/usr/ccs/
bin:$SYBASE/install:/usr/openwin/bin:/home2/procfu2/dat:/home2/procfu2/trab/mari
:.
CDPATH=/home2/procfu2/trab/mari:/home2/procfu2/trab/mari/run:/home2/procfu2:/hom
e2/gral:/home2/procfu2/trab:/home1:.
MANPATH=$COMPILER/man:/usr/man
LD_LIBRARY_PATH=/usr/lib:$COMPILER/lib:$SYBASE/lib
export PATH MANPATH LD_LIBRARY_PATH CDPATH HOME
# Opciones
stty erase ^H
cd
set -o vi
export TZ=EST
```

Como podemos observar, también se encuentran las librerías de Sybase, sin las cuales no es posible realizar las conexiones necesarias a la base de datos. De igual manera se encuentran definidos los directorios de trabajo.

Al iniciar una sesión en Unix, aparece como sigue:

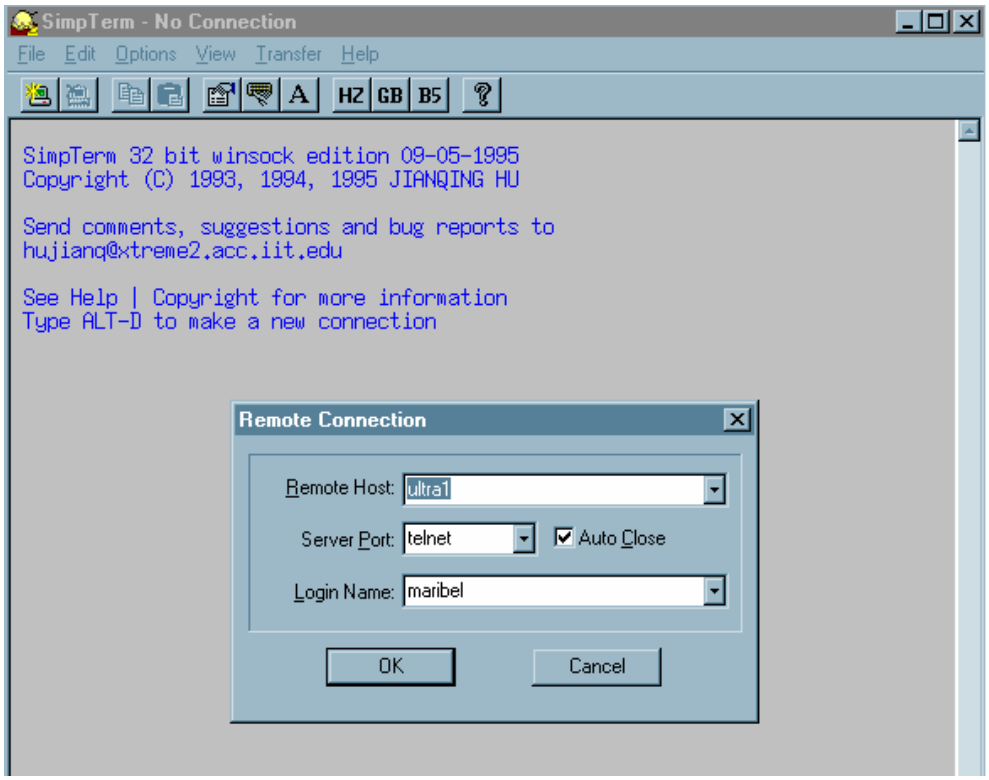


Figura 9.1 Conexión.

Se proporciona el login y el password para iniciar sesión, verificamos los estados de las Formas Únicas, las que están en 35, son las que debemos tomar para el proceso.

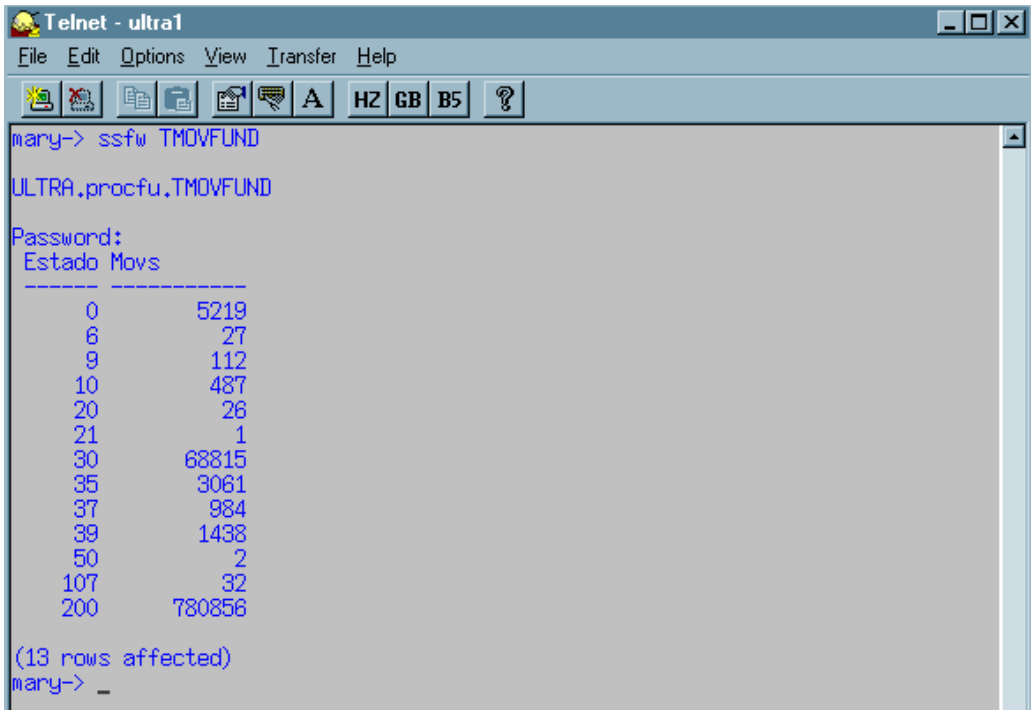


Figura 9.2 Estados iniciales.

Se inicia el proceso con el primer programa que es la validación.

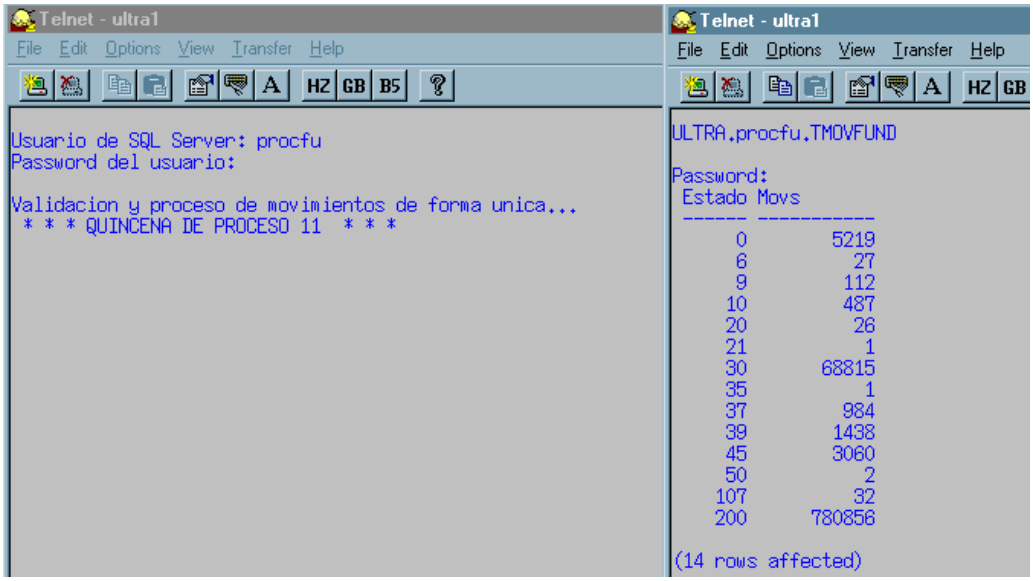


Figura 9.3 Inicio de la validación.

Podemos observar que todas las Formas Únicas que ingresan al proceso se actualizan al estado 45.

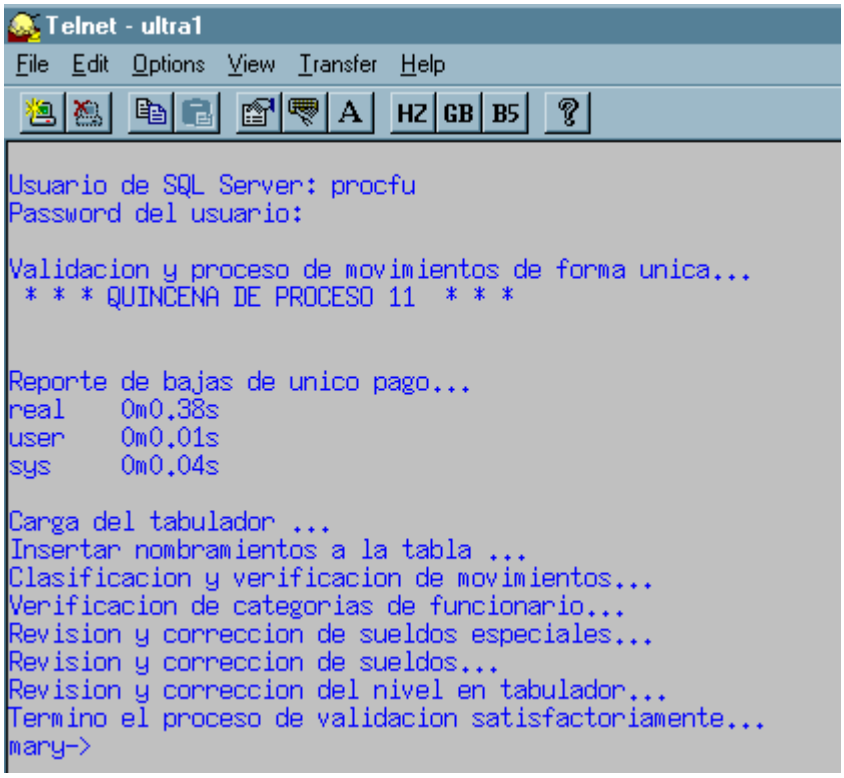


Figura 9.4 Validación.

Cuando la validación de la Forma Única termina, se verifican nuevamente los estados, ya que los que se encuentran en 45, son aquéllas Formas Únicas que tienen algún error.

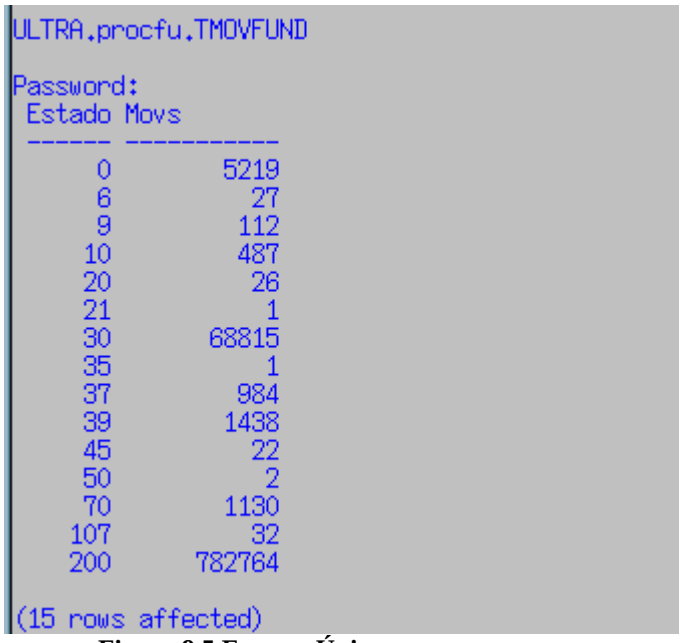


Figura 9.5 Formas Únicas con error.

Se corrigen los errores para poder continuar con el proceso. Una vez hecho esto, se ejecuta el programa de verificación de inconsistencias.

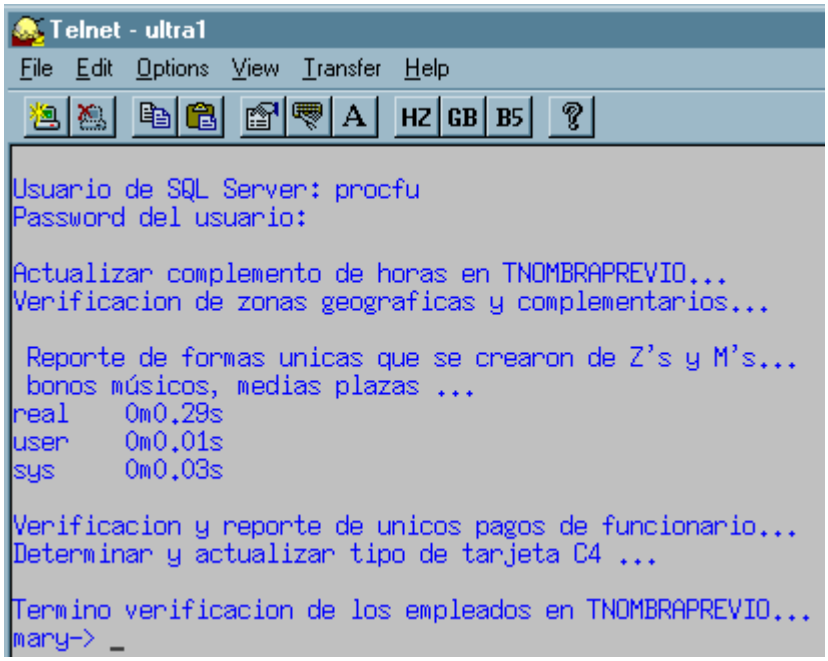


Figura 9.6 Verificación de inconsistencias.

Al final de este programa, verificamos que no haya inconsistencias, para lo cual se hace un listado con el número de empleado que genera el programa, ya sea que se haga algún movimiento de Forma Única que corrija el problema o bien se tome nota para reportarlo en incompatibles.

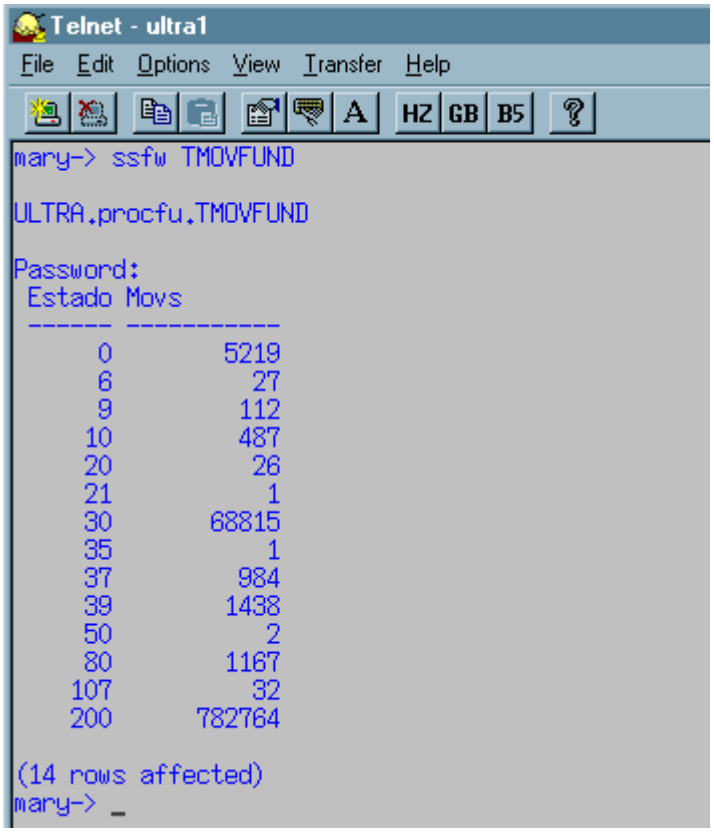


Figura 9.7 Verificación de estados.

Como podemos observar, ahora todos los movimientos se encuentran en estado 80, el siguiente paso es generar el reporte de incompatibles.

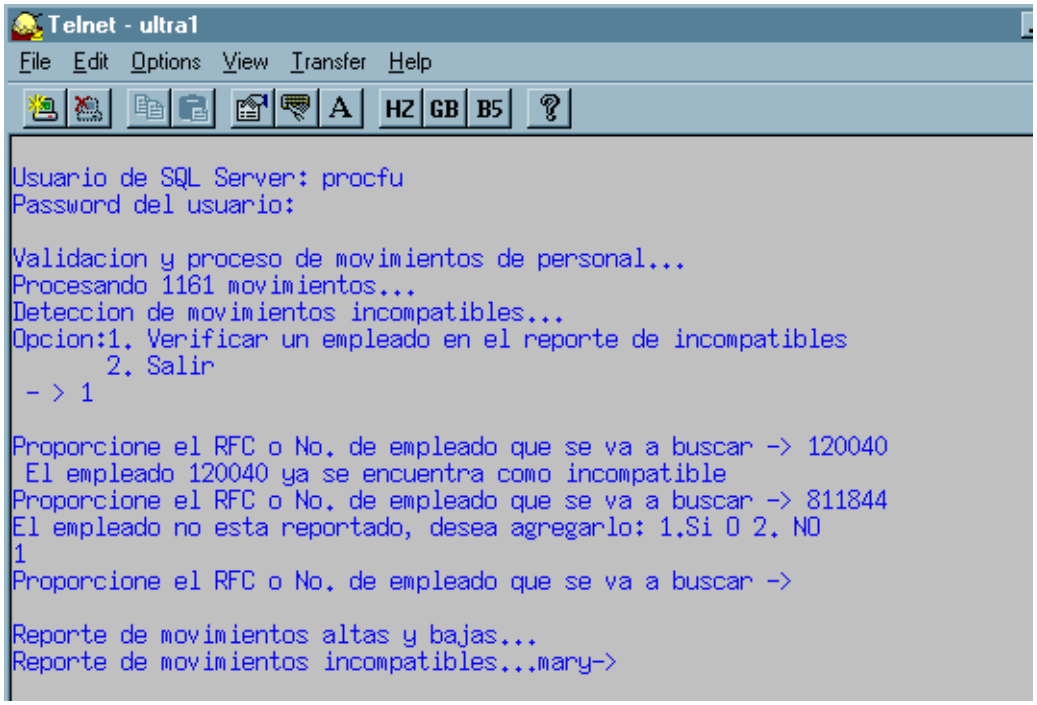


Figura 9.8. Generación de reporte de incompatibles.

Este proceso genera un reporte, del se presenta un ejemplo:

```
                    UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
                       *DIRECCION GENERAL DE PERSONAL*
                  R.=( 32 )          *SUBDIRECCION DE INFORMATICA*
                          QNA NO. 18      HOJA NUM. 1
                  *** REPORTE DE ALTAS Y BAJAS INCOMPATIBLES ***

*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*

          AAAD481003    ALDAMA AVALOS DANIEL     67346

          S I T U A C I O N   A C T U A L
   PROF TIT A T C   D6489 23773-24   40.0  10,061.64  1002-44401-121-08
   FUNC JEFE DIV    CF5411 49196-90   0.0   3,071.00  1002-44401-125-00


                   M O V I M I E N T O S

TIPO  HORAS  CATEGORIA      CVECATEG NO.PLAZA SUELDO  COD.PROGRAMATICO F.ALTA  F.LIMITE FOLIO QNA  OBSERVACIONES

BAJA  48.0     ASG ACAD ADMVA  CA5411  49196-90   3,071.00  1002-44401-125-00 0   1/09/2001          80049             ( )( )( )


*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*

          S I T U A C I O N   F I N A L

   PROF TIT A T C   D6489  23773-24   40.0   10,061.64  1002-44401-121-08
*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*

                          SUELDO ACTUAL     13,132.64    HORAS QUE COBRA   40.0

                          SDO.PROPUESTO     10,061.64    HORAS PROPUESTAS  40.0


                          *** I N C O M P A T I B L E ***
                                 * FUNCIONARIO *
```

Se verifican los estados, y aquéllos movimientos que se encuentran en estado 45, son los que estaban retenidos y al tener el empleado movimientos en la quincena se liberan.

Estado	Movs
0	5219
6	27
9	112
10	487
20	26
21	1
30	68815
35	1
37	984
39	1438
45	4
50	2
90	1061
95	100
107	28
192	6
200	782764

(17 rows affected)

Figura 9.9 Estados después de incompatibles.

Una vez el área revisora indica cuáles son los cambios correspondientes, se hacen y se procede a la generación del reporte de

banco de horas y de algunos otros que se entregarán al finalizar la quincena.

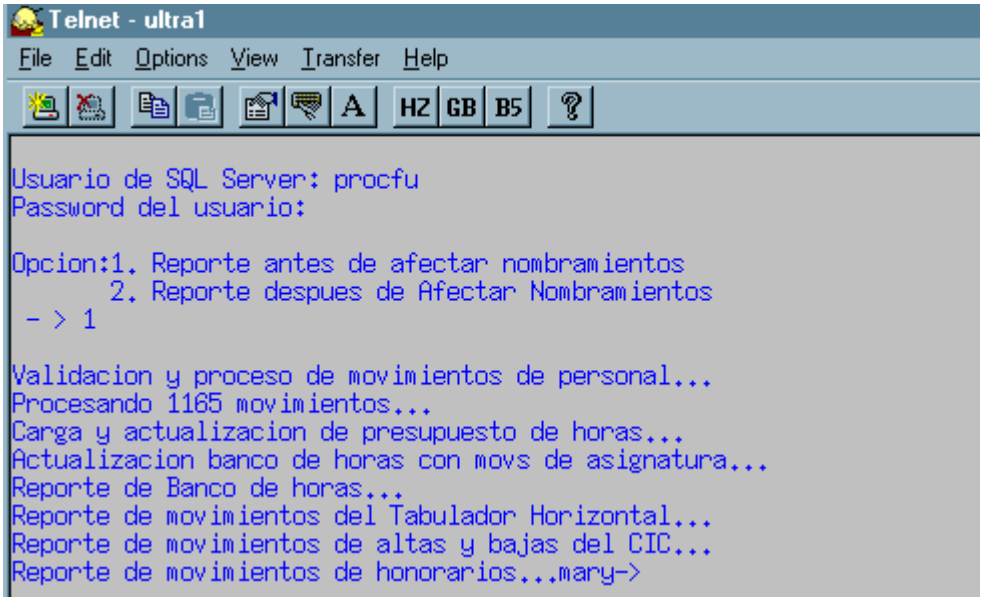


Figura 9.10 Proceso de banco de horas.

Algunos ejemplos de estos reportes son:

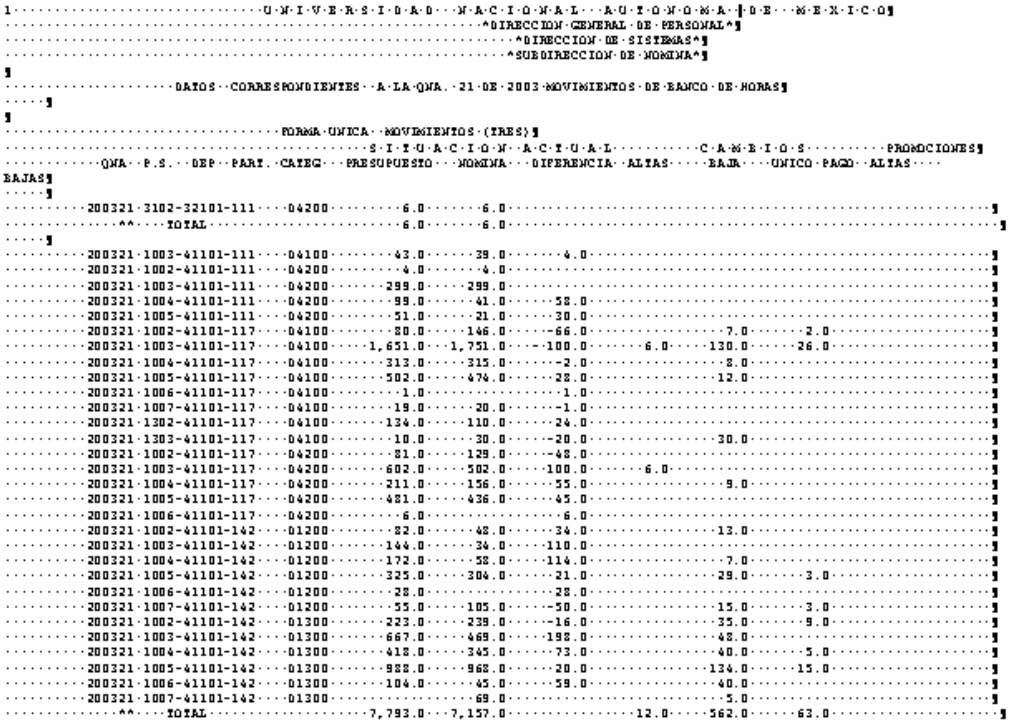


Figura 9.11 Banco de horas con Forma Única.

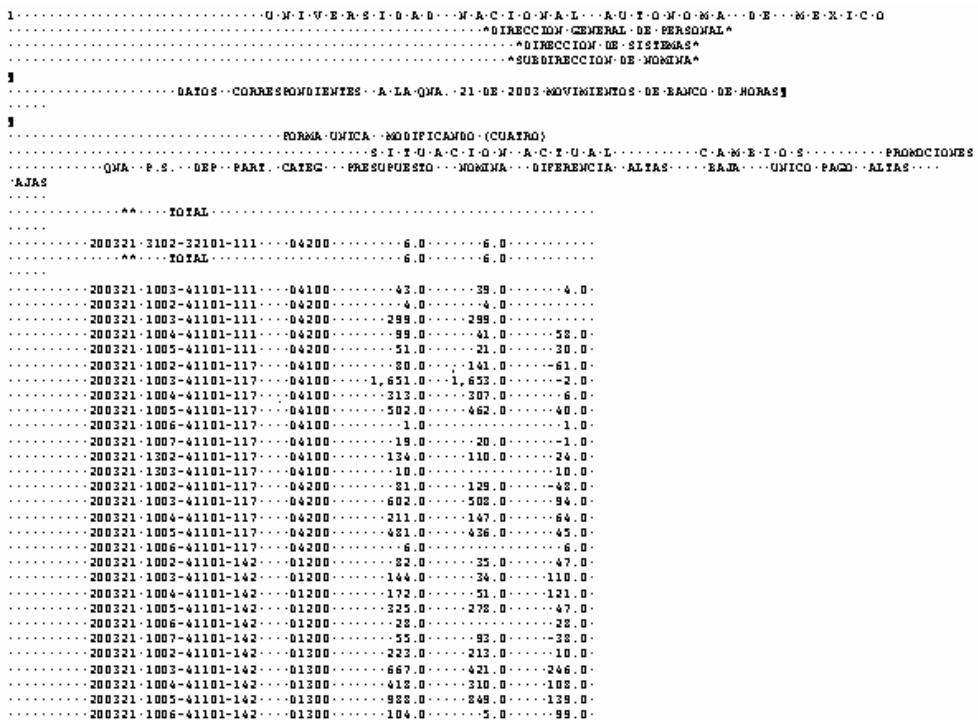


Figura 9.12. Banco de horas antes de Forma Única.

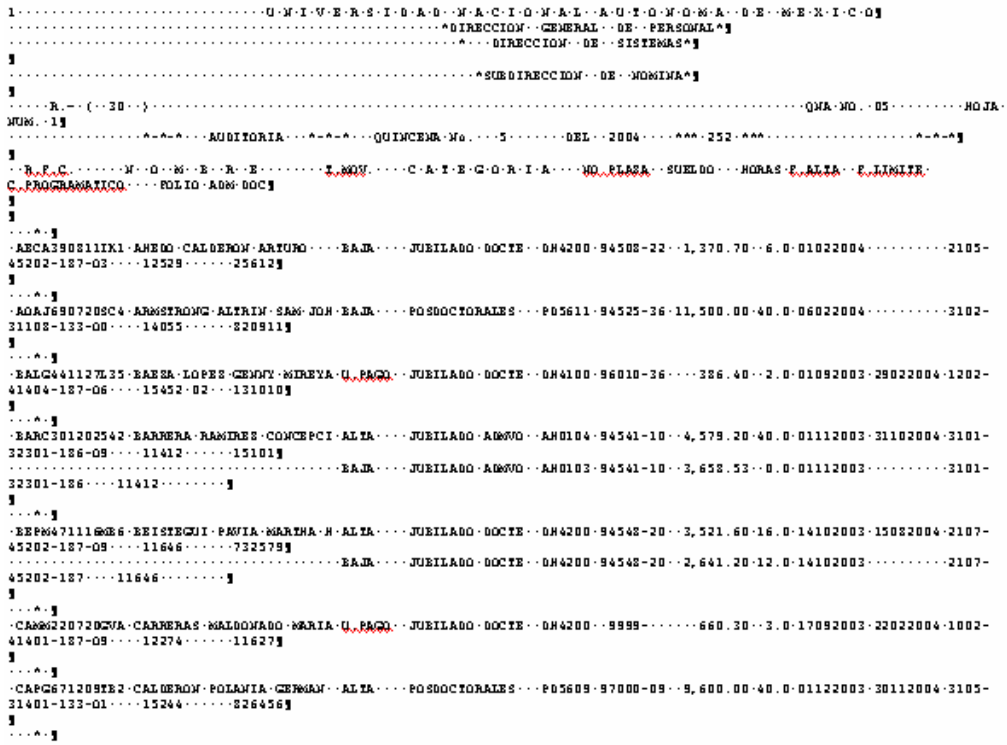


Figura 9.13 Reporte de honorarios.

A continuación se ejecuta el programa de creación de tarjetas retroactivas.

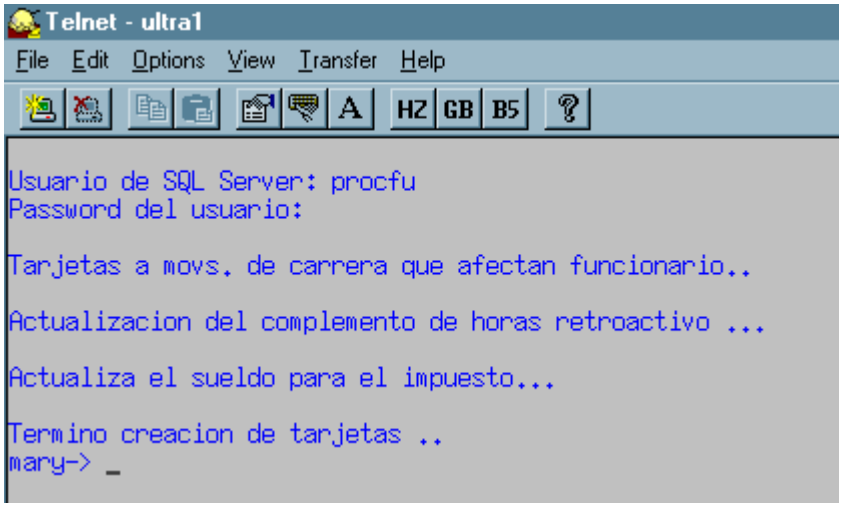


Figura 9.14 Proceso de Altas

El proceso de altas deja todas las Formas Únicas en estado 120, pero también crea las tarjetas correspondientes a los movimientos, en estado 120 pero en la tabla TALTAS

```
mary-> ssfw TMOVFUND

ULTRA.procfu.TMOVFUND

Password:
 Estado Movs
 ------ -----------
      0        5219
      6          27
      9         112
     10         487
     20          26
     21           1
     30       68815
     35           1
     37         984
     39        1438
     50          21
    107          28
    120        1146
    192           6
    200      782764

(15 rows affected)
mary-> ssfw TALTAS

ULTRA.procfu.TALTAS

Password:
 Estado Movs
 ------ -----------
    120        2775
    200      278454
```

Figura 9.15. Estados después de altas.

```
 Proceso  1 = Calcular Sueldos Retroactivos
          2 = Calcular Conceptos Retroactivos
          3 = Revisar los funcionarios con base academica
          4 = Todo el proceso
          5 = Salir 4
iUltFecGratif 20021231
iUltFecDiasJul 20020805
iFecPrimVacAct 20021216
iFecPrimVacAnt 20020805
PORC_SERV_MED 0.020000
Topes 1 543.600000 0.000000 0.000000 19980101 19981231
Topes 1 543.600000 0.000000 9060.000000 19980101 19981231
Topes 1 543.600000 181.200000 9060.000000 19980101 19981231
Topes 2 620.100000 0.000000 0.000000 19990101 19991231
Topes 2 620.100000 0.000000 10335.000000 19990101 19991231
Topes 2 620.100000 206.700000 10335.000000 19990101 19991231
Topes 3 682.200000 0.000000 0.000000 20000101 20001231
Topes 3 682.200000 0.000000 11370.000000 20000101 20001231
Topes 3 682.200000 227.400000 11370.000000 20000101 20001231
Topes 4 726.300000 0.000000 0.000000 20010101 20011231
Topes 4 726.300000 0.000000 12105.000000 20010101 20011231
Topes 4 726.300000 242.100000 12105.000000 20010101 20011231
Topes 5 758.700000 0.000000 0.000000 20020101 20021231
Topes 5 758.700000 0.000000 12645.000000 20020101 20021231
Topes 5 758.700000 252.900000 12645.000000 20020101 20021231
Topes 6 785.700000 0.000000 0.000000 20030101 20771231
Topes 6 785.700000 0.000000 13095.000000 20030101 20771231
Topes 6 785.700000 261.900000 13095.000000 20030101 20771231

Ya cambiaron las fechas de los nombramientos suspendidos?? 1.Si, 2.No -> 1

Validacion y proceso de movimientos de personal...
Procesando 2775 movimientos...
Calculo de sueldos retroactivos...
Ajuste de sueldos y asueldos...
Calculo de conceptos retroactivos...
Calculo de impuestos retroactivos...mary->
```

Figura 9.16 Calculo de conceptos.

Se ejecuta el proceso de cálculo de conceptos, como vemos en la imagen anterior, nos presenta todos los datos necesarios para calcular los conceptos y nos pregunta sobre ciertas condiciones que deben estar listas antes de ejecutar el proceso, en este caso sólo es el cambio de fechas de los movimientos suspendidos.

Una vez finalizado el cálculo, se hace un respaldo, antes de afectar los nombramientos en las tablas reales, de tal manera que si hay algún error o cambio posterior, solamente se tiene que procesar un empleado y no todos.

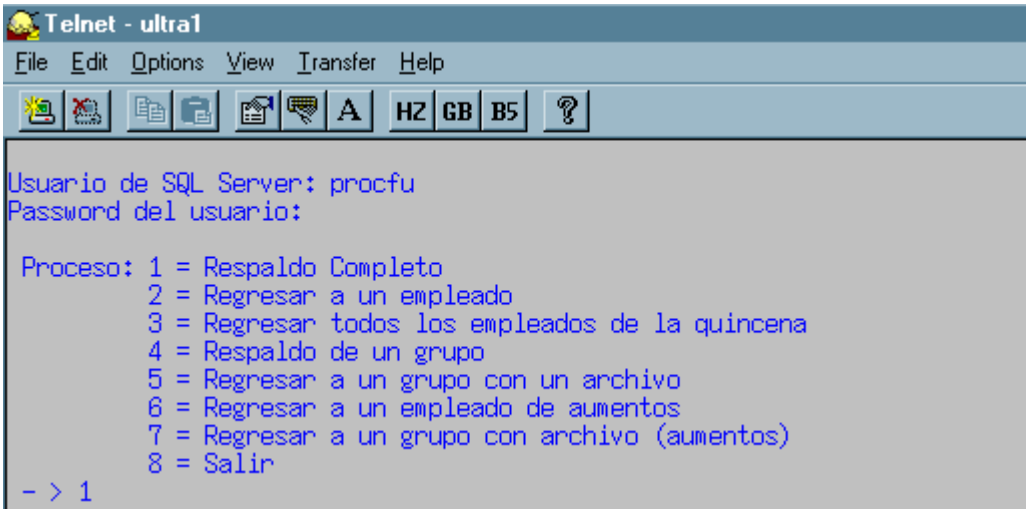


Figura 9.17 Respaldo

La opción 1, sólo se presenta una vez, al ejecutar el programa de respaldo, debido a que tiene otras opciones, esto con el objeto de no incurrir en errores al elegir una de ellas, ya que al ejecutar indebidamente esta opción, borraría todo el respaldo y nos dejaría sin la opción de reproceso.

Se pide un respaldo completo al administrador de la base de datos y sólo entonces se procede a la afectación de nombramientos con el cnpro.

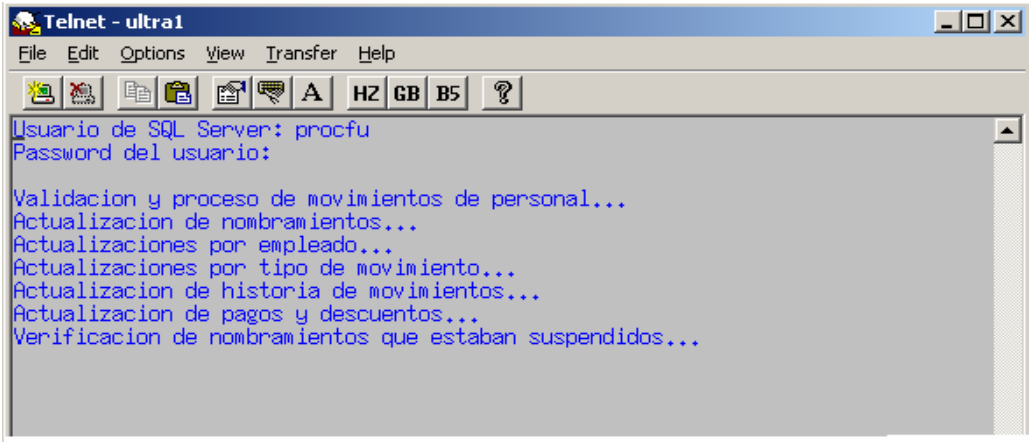


Figura 9.18. Afectación de nombramientos.

Esta es la última fase del proceso, los nombramientos están actualizados y las Formas Únicas se encuentran en el histórico.

10. CONCLUSIONES

El Subsistema se encuentra actualmente trabajando para el sistema de nómina de la UNAM. Se le da mantenimiento cuando se hacen modificaciones o se agregan nuevas condiciones de trabajo.

Del desarrollo de este subsistema se concluye que:

- Aunque existen nuevas herramientas para el desarrollo de sistemas, a veces por la naturaleza de los requerimientos de los mismos sistemas es necesario usar herramientas menos modernas, pero que satisfacen mejor dichos requerimientos.

- La programación estructurada sigue siendo muy útil para el desarrollo de sistemas.

- Aunque C no es un lenguaje de muy alto nivel (es decir con una interfaz amigable al usuario, e instrucciones sencillas), como sería Visual Basic, Power Builder, etc., sigue siendo un gran lenguaje de programación; sin embargo, consideró conveniente analizar si es posible desarrollar este subsistema en un lenguaje orientado a objetos.

- Otra conclusión importante es que muchas veces por la naturaleza del sistema no es posible dar por terminado el proyecto al 100%, debido a los múltiples cambios que surgen cada quincena; ya que han llegado a cambiar por completo las condiciones o la forma de calcular ciertos pagos, otras veces se han agregado condiciones completamente nuevas, categorías nuevas e incluso se han modificado condiciones que por muchos años se han manejado de cierta manera, por ejemplo, cuando se inició el desarrollo del sistema se manejaba una categoría de 6 caracteres, y recientemente se decidió que sería de 7.

- Cuando se desarrollan los sistemas también es muy importante tomar en cuenta el mantenimiento,que durante su vida útil estos pueden requerir, es decir, estructurar los programas de tal manera, que sea fácil darles mantenimiento. Para este sistema, esto se aprendió con la experiencia, la primera versión fue programada de cierta manera y con esta reestructuración se buscó principalmente que los programas quedaran organizados de tal forma, que fuera muy sencillo en un

futuro hacer cambios al código, sin que se corriera el riesgo de tener errores por no haber considerado algún otro programa.

- En cuanto al objetivo principal al iniciar la reestructuración de este sistema, se cumplieron las metas, ya que al cambiar la forma de proceso, se mejoraron en un 80% los tiempos de proceso. Teníamos un cálculo de conceptos que tardaba 3 horas aproximadamente y que ahora tarda 30 minutos.
- El manejo del tabulador ahora es más sencillo, y sobre todo ahorra mucho tiempo de proceso al tenerlo en memoria.
- Los datos claves del empleado que se requieren para el proceso se obtienen una sola vez y se guardan para utilizarlos cuando se requieran, por lo cual también tenemos aquí un ahorro de tiempo y sobre todo es posible mantener fácilmente el código.
- La revisión manual se redujo considerablemente, sin embargo, aún se tiene que hacer, pero es mínima, sobre todo porque cada quincena la información varía y se llegan a dar casos nunca antes considerados. Aunque estos casos nos permiten estar mejorando constantemente el sistema.

BIBLIOGRAFIA

Diseño de sistemas de información
Teoría y Práctica
Autor: Burch Grudnitski
Editorial: Noriega Editores

Ingeniería del software
Un enfoque práctico
Autor: Roger S. Presuman
Editorial: McGraw Hill

El lenguaje de programación C.
Autores: Brian W. Kernighan
Dennis M. Ritchie
Editorial: Prentice Hall

Desarrollo y gestión de proyectos informáticos
Como dominar planificaciones ajustadas de software
Autor: Steve McConnel
Editorial: McGraw Hill

Turbo C / C++
Manual de referencia
Autor: Herbert Schildt
Editorial: McGraw Hill

Diseño de sistemas de información
Teoría y práctica
John G. Burch y Gary Grudnistski
Grupo Noriega Editores

Ingeniería de software explicada
Autores: Mark Norris y Meter Rugby
Editorial: Noriega Editores

Sistemas de información
Análisis, diseño y puesta a punto.
Autor: Henry C. Lucas, JR.
Editorial: Paraninfo, S.A.