UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"ESPECIFICACIÓN Y DESARROLLO DEL MÓDULO DE ADMINISTRACIÓN DE DATOS DE MEMORIA COMPARTIDA PARA UN SIMULADOR DE PROCESOS NUCLEARES"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A

DIONISIO TELÉSFORO REYES

DIRECTOR:
DR. CARLOS CHÁVEZ MERCADO

CODIRECTOR:
M.C. EDGAR SALAZAR SALAZAR

MÉXICO, D.F. ABRIL 2005

Dedicatorias

A Dios, por haberme invitado a la vida. Sin Él nada soy.

Con inmenso amor a mi madre Eloisa Reyes Galicia por su amor y paciencia conmigo.

A mis amadas hermanitas Araceli, Elvia y Florentina, porque siempre puedo contar con ellas.

A mi director de tesis Dr. Carlos Chávez Mercado por poner su confianza en mi.

A mi codirector de tesis M.C. Edgar Salazar Salazar por su firme respaldo.

A mis sinodales, por sus valiosos comentarios para el mejoramiento de la presente tesis:
Dr. Juan Luis François Lacouture
Dra. Cecilia Martín del Campo Márquez
Dr. Arturo Reinking Cejudo

Al Grupo de Ingeniería Nuclear de la Facultad de Ingeniería por su amistad y constante apoyo.

Gracias

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRONIMOS

ALPHA	Estación de trabajo desarrollada por Digital con sistema operativo de tiempo compartido.
BWR	Reactor de agua ligera en ebullición.
CFE	Comisión Federal de Electricidad.
CNLV	Central Nucleoeléctrica Laguna Verde.
CPU	Unidad de Procesamiento Central.
DataViews	Software que permite manejar objetos gráficos.
INPUTD	Subrutina principal de la estructura modular de RELAP5/SCDAP que procesa los archivos de entrada.
IPC	Comunicación entre procesos en UNIX System V.
LAIRN	Laboratorio de Análisis en Ingeniería de Reactores Nucleares.
RAM	Memoria de acceso aleatorio.
RELAP/SCDAP	Código desarrollado por el Laboratorio Nacional de Energía y Ambiente de Idaho.
SEPS	Sistema de Exhibición de Parámetros de Seguridad.
SHM	Memoria compartida.
TRNCTL	Subrutina del código RELAP5/SCDAP que produce la simulación en estado estable, en un transitorio o para accidente severo.
UNIX System V	Sistema Operativo desarrollado por Bell Labs de AT&T.

INTRODUCCIÓN

La computadora es la herramienta más sofisticada que ha creado el hombre y la que más ha extendido su poder de hacer, de conocer y de saber. La computadora, como herramienta sofisticada, ha dado al hombre capacidades más allá de lo puramente humano con la cual puede ver sin utilizar sus ojos, escuchar sin sus oídos, fabricar productos sin emplear sus manos. La carrera por la innovación se ha conducido sin precedentes desde la aparición de la computadora a fines de los años cuarenta y han conducido a una revolución: la revolución de la información.

Debemos mirar la aplicación intensiva y extensiva de la computación en todos los campos como un fenómeno integrado al desarrollo humano. Ahora hay nuevas características en la investigación científica teórica y experimental. La informática unida a otras ciencias abre la exploración de nuevas fronteras en Astronomía, Biología, Química, Física, etc. lo que la hace presente en todos los campos. Cada vez que el costo de la computación mejora en un factor de 10, las posibilidades de utilización se multiplican y aplicaciones que eran impensables de repente se hacen realizables.

El entorno educativo no puede mantenerse al margen de estos cambios, los instrumentos asistidos por computadora se han convertido en componentes normales de los programas de capacitación. Académicamente los simuladores constituyen instrumentos de enseñanza y capacitación muy eficaces acerca de las características operacionales del sistema simulado, permitiéndonos absorber la mayor parte de los conocimientos y habilidades sin tener miedo a equivocarnos o al fracaso. Además hacen posible contar con las aportaciones de los mejores expertos para construir simulaciones de situaciones reales y se cuente con las herramientas necesarias para analizar los diversos escenarios, permitiéndonos buscar alternativas bajo la premisa de cometer errores para aprender.

Para el analista, el simulador constituye una herramienta muy potente para diseño, prueba y validación que suple la imposibilidad física de manipular el sistema real en el que no hay margen para experimentos interactivos, por un entorno práctico de simulación. En esta línea, el simulador

de aula que se desarrolla actualmente en la Facultad de Ingeniería de la UNAM, por el Grupo de Ingeniería Nuclear, tiene como objetivo la simulación del reactor durante operación normal, transitorios y en situaciones de accidente. La capacitación con simuladores permite que los usuarios adquieran un nivel adecuado de conocimientos prácticos sobre los sistemas y comportamiento de una central nuclear y tengan la oportunidad de poner a prueba sus conocimientos sobre el control del reactor con importantes ahorros económicos, de energía y seguridad.

El prototipo del simulador de aula es un sistema computacional con una arquitectura de simulación distribuida que utiliza códigos nucleares especializados y software gráfico avanzado para visualizar la información. La información se representa en la forma de despliegues gráficos de parámetros de planta, de variables de proceso y de estado, de instrumentación virtual y de componentes (válvulas, bombas, volúmenes termo-hidrodinámicos, etc.) que forman parte de una planta nuclear.

Actualmente el simulador hace uso de segmentos de memoria compartida, los cuales contienen la información originada por múltiples modelos matemáticos, despliegues gráficos interactivos y procesos relacionados. Aún cuando estos segmentos de memoria compartida cumplen su función de forma independiente, la especificación y administración de los mismos es un requisito indispensable.

La presente tesis tiene como objetivo la especificación, desarrollo e implementación del Módulo de Administración de Datos de Memoria Compartida para el Simulador de Aula de Procesos Nucleares, actualmente en desarrollo en la Facultad de Ingeniería de la UNAM. El propósito específico del módulo es la integración de información variante en el tiempo, y su adecuación para una correcta utilización de la misma, de modo que permita el acceso de lectura y escritura a múltiples procesos de comunicación, monitoreo y control del simulador, así como el ajuste de dichos procesos.

A continuación se describe la organización de la tesis:

El capítulo 1 incluye una clasificación de los simuladores y sus aplicaciones, una breve descripción de la Central Nucleoeléctrica de Laguna Verde (CNLV), del Simulador Réplica de la CNLV, y del Simulador de Aula actualmente en desarrollo.

El capítulo 2 describe brevemente el paquete de comunicación que se encuentra integrado en el sistema operativo UNIX System V y que incluye funciones de acceso y administración de memoria compartida.

En el capítulo 3 se revisan las bases conceptuales para la utilización de memoria compartida y se describe el proceso de diseño, especificación y desarrollo del Módulo de Administración de Datos.

El método de evaluación, descripción de las pruebas realizadas al sistema y una breve discusión de los resultados de la evaluación se encuentra en el capítulo 4.

Finalmente las conclusiones del trabajo desarrollado y recomendaciones a trabajos futuros se presentan en el capítulo 5.

CAPÍTULO 1

ASPECTOS PRELIMINARES

El objetivo del presente capítulo es presentar una breve descripción y clasificación de los simuladores así como sus aplicaciones en la industria nuclear, una breve descripción de la Central Nucleoeléctrica de Laguna Verde (CNLV), las características más importantes del simulador réplica de la CNLV y la descripción del prototipo del Simulador de Aula, en desarrollo.

1.1 Papel de los Simuladores en el Entrenamiento de Operadores.

Los simuladores tienen un importante papel en los programas de entrenamiento de operadores de plantas nucleares en todo el mundo. Estadísticamente, el número de simuladores dedicados a entrenamiento por planta nuclear es mayor a uno. El 100% de los operadores del cuarto de control se capacita en simuladores de alcance completo. Más de 80% de los operadores reciben entrenamiento en simuladores réplica, en algunos casos complementados por simuladores especializados. [1]

Existen sistemas de entrenamiento específico con simuladores especializados para apoyar actividades diversas en una central nuclear que van desde entrenamiento básico hasta licenciamiento en operación normal, anormal y de emergencia. Además se encuentran los simuladores de escenarios de accidente y accidentes severos.

El tiempo de simulación para entrenamiento inicial programado para el personal del cuarto de control, incluye una combinación de operación en condición normal, anormal y operación de emergencia. En muchos casos la tendencia es dedicar 1/3 del tiempo para cada condición, con una tendencia general a simular en alto porcentaje condiciones normales. La situación extrema está representada por una distribución de 75% de capacitación en operación en condiciones normales y 5% en condiciones de emergencia.

El tiempo de entrenamiento varía de un mínimo de 20 horas/año hasta aproximadamente 160 horas/año. El programa dedicado a reentrenamiento o relicenciamiento hace énfasis en operaciones de emergencia, empleando hasta un 80% del tiempo total.

Otras actividades que se apoyan con el uso de simuladores son: desarrollo de nuevos procedimientos de operación o modificación de uno existente, verificación de la respuesta del equipo de la planta, planeación del arranque o el apagado de la planta, revisión y/o evaluación de eventos específicos de la planta, evaluación y verificación de cambios en el diseño de la planta o preparación de acciones como respuesta a un plan de emergencia.

1.2 Clasificación de los Simuladores.

Simuladores de Entrenamiento de Alcance Completo (full-scope). Representan en tiempo real el mayor rango de operaciones que pueden realizarse desde el cuarto de control principal. Este consiste de un cuarto de control que simula el sistema de suministro de vapor principal, y los sistemas de balance de planta, incluyendo los sistemas nucleares, convencionales, de servicio y de seguridad más importantes.

Simuladores de Entrenamiento Réplica (o Simuladores Específicos de Planta). Son aquellos simuladores de alcance completo diseñados sobre una planta de referencia, que incorporan modelos detallados de tal planta y sus sistemas, incluyendo una réplica del diseño del cuarto de control. Estos simuladores responden tal como la planta lo haría en condiciones normales y anormales de operación en tiempo real.

Simuladores de Tarea Parcial (part-task). Diseñados para entrenamiento sobre operaciones específicas de la planta o para entrenamiento en fenómenos específicos, que pueden ser simulados con más precisión que en uno de alcance completo. Pueden ser usados para entrenamiento en actividades particulares del cuarto de control, usualmente incluyen controles e instrumentos réplica. Las actividades a simular usualmente se eligen sobre la necesidad de reentrenamiento frecuente en tareas muy importantes o difíciles.

Simuladores de Principios Básicos (o Simuladores Compactos). Son empleados para proveer un entrenamiento general sobre el sistema de la planta, con menos detalle que en un simulador de alcance completo. El modelo del sistema de la planta es usualmente genérico y algunas veces simplificado con énfasis en la comprensión de la dinámica de planta. El ámbito de simulación puede limitarse a sistemas principales, componentes y funciones, o puede ser tratada en esencia la planta completa. El cuarto de control y paneles frecuentemente presentan diseños diferentes en comparación con un cuarto de control convencional. Los teclados son empleados para introducir acciones de control y se usan monitores para mostrar las salidas de la simulación de la planta.

Simuladores de Concepto. Son usados para entrenamiento en conceptos particularmente difíciles, aislando éste de otras consideraciones. Especialmente útil en el refuerzo teórico de conceptos. Para este propósito de entrenamiento no son necesarios los controles réplica, ni los instrumentos en tiempo real.

Simuladores de Propósito Especial. Son usados varios tipos, los más importantes son los simuladores termohidráulicos. Este simulador es usado principalmente para demostrar principios termohidráulicos básicos y fenómenos de planta. Son una combinación de simulación y sistema experto con interfaces basadas en teclados y pantallas.

Simulador Gráfico Interactivo. Es un sistema computacional que permite seguir el comportamiento de un proceso industrial, proporcionando la capacidad de despliegues en tiempo real y la interacción con el sistema y sus componentes por medio de interfaces gráficas.

El simulador en desarrollo en el Laboratorio de Análisis en Ingeniería de Reactores Nucleares debido a que es modular y configurable engloba las clases anteriores.

1.3 Objetivo de los Simuladores para Entrenamiento.

El objetivo principal de proporcionar entrenamiento a los operadores es desarrollar en ellos habilidades y conocimientos específicos sobre el manejo y operación de la planta. A continuación se listan algunos de los aspectos más importantes [1]:

Principales habilidades a considerar en el entrenamiento con simuladores:

- Priorizar, interpretar y verificar alarmas/anunciadores.
- Diagnosticar eventos.
- Interpretar respuestas del sistema y predecir efectos sobre la planta.
- Seleccionar y utilizar procedimientos de la planta.
- Localizar y manipular controles, verificar respuestas y tomar el control de sistemas automáticos cuando se requiera.
- Intercambio de información.
- Supervisar y dirigir operadores licenciados.
- Reconocer y asegurar el cumplimiento de especificaciones técnicas

Conocimientos y condiciones limitantes de operación considerados en el entrenamiento con simuladores:

- Comprender la respuesta integrada de la planta.
- Comprender los procedimientos generales de la planta.
- Comprender los procedimientos de operación de emergencia.
- Comprender el plan de emergencia.
- Comprender las especificaciones técnicas.
- Actitud como miembro del equipo operacional.

1.4 Breve Descripción de la Central Nucleoeléctrica de Laguna Verde (CNLV).

Una nucleoeléctrica es una central térmica de producción de electricidad. Su principio de funcionamiento es básicamente el mismo que el de las plantas que funcionan con carbón, combustóleo o gas: la conversión de calor en energía eléctrica. Esta conversión se realiza en tres etapas: en la primera, la energía del combustible se utiliza para producir vapor a presión y temperatura elevadas; en la segunda etapa la energía del vapor se transforma en movimiento de una

turbina; en la tercera etapa, el giro del eje de la turbina se transmite a un generador, que produce la energía eléctrica.

Las centrales nucleoeléctricas se distinguen de las demás centrales térmicas solamente en la primera etapa de conversión, es decir, en la forma de producir vapor. En las centrales convencionales el vapor se produce en una caldera donde se quema carbón, combustóleo o gas natural; las centrales nucleoeléctricas tienen un reactor nuclear, que equivale a la caldera de las centrales convencionales.

La más importante de las instalaciones nucleares en México es la Central Nucleoeléctrica de Laguna Verde (CNLV), se encuentra localizada sobre la costa del Golfo de México, en el municipio de Alto Lucero, estado de Veracruz. Está integrada por dos unidades, cada una con una capacidad de 682.44 MWe (Mega Watts eléctricos); los reactores son tipo Agua Hirviente (BWR-5) y la contención tipo Mark II de ciclo directo. La primera unidad inició su operación comercial el 14 de agosto de 1990 y la segunda unidad el 12 de abril de 1995. Esta instalación es operada por la Comisión Federal de Electricidad (CFE).

Tabla 1.1 Información General CNLV

Localización	Laguna Verde; 70 km al NNO de la ciudad de Veracruz.
Número de unidades	Dos
Proveedor de los sistemas nucleares de suministro de vapor	General Electric
Tipo de reactor	BWR/5 (reactor de agua ligera en ebullición).
Potencia térmica por reactor	2,027 MWt
Carga inicial de combustible por reactor	444 ensambles; 92 toneladas de combustible (UO2) al 1.87% U235 en promedio
Proveedor de los turbogeneradores	Mitsubishi Corporation
Potencia eléctrica bruta por unidad	682.44 Mwe
Potencia eléctrica neta por unidad	655.14 Mwe
Energía anual generada por unidad	4,782 GWh, al 80% de factor de capacidad
Líneas de transmisión	Tres de 400 KV a Tecali, Puebla y Poza Rica; Dos de 230 KV a la ciudad de Veracruz

La Unidad 1 ha generado más de 65.7 millones de MWh, con una disponibilidad de 84.41% y un factor de capacidad de 80.72%; mientras que la Unidad 2 ha generado más de 46.1 millones de MWh, siendo su factor de disponibilidad de 84.78% y el de capacidad de 81.65% al 31 de diciembre de 2004. Ambas unidades representan el 2.99% de la capacidad efectiva instalada de CFE, con una contribución a la generación anual del 4.48% en el año 2004. [2,3]

La central consta de seis edificios principales y otros secundarios. Los edificios principales son:

- *Edificio del reactor*: En su interior se encuentra el reactor nuclear, sus sistemas auxiliares, dispositivos de seguridad, la plataforma de recambio de combustible y la alberca de combustible gastado.
- *Edificio del turbogenerador*: En este se encuentran instaladas las turbinas de alta y baja presión, generador eléctrico y su excitador, condensador, los precalentadores de agua de alimentación y recalentadores de vapor.
- *Edificio de control*: Aquí se encuentra instalado el cuarto de control principal, la computadora de proceso, bancos de baterías, laboratorios radioquímicos y el acceso de personal a la unidad.
- *Edificio de generadores diesel*: En su interior se encuentran los tres generadores diesel que se utilizan para el suministro de energía eléctrica, en casos de emergencia, para los sistemas de refrigeración del reactor.
- *Edificio de tratamiento de residuos radiactivos*: Aloja los sistemas de tratamiento de residuos sólidos, líquidos y gaseosos de mediano y bajo nivel de radiactividad.
- *Edificio de planta de tratamiento de agua*: Contiene la planta de producción de agua desmineralizada de alta pureza usada en el ciclo de vapor.

Los edificios secundarios son: Obra de toma de agua de enfriamiento para el condensador y los componentes nucleares, subestación eléctrica, administrativo, almacenamiento de partes de repuesto, acceso, almacenamiento temporal de residuos de mediano y bajo nivel de radiactividad, entrenamiento y el centro de información pública. [4]

1.5 El simulador de Entrenamiento de la CNLV.

El simulador de entrenamiento de la CNLV fue encargado por la Comisión Federal de Electricidad al Instituto de Investigaciones Eléctricas en 1984 debido a que la CFE estuvo capacitando a sus operadores en el extranjero a un alto costo y en simuladores de plantas que mantenían diferencias con la CNLV. El simulador, de alcance total, se entregó en 1991 y se encuentra instalado en el Centro de Entrenamiento de Laguna Verde.

El simulador reproduce fielmente el comportamiento de la unidad 1 de la CNLV, cuenta con 21 de los tableros principales, con instrumentación idéntica a los que se encuentran en el cuarto de control de la central, de modo que los operadores adiestrados en el simulador se familiarizan con la disposición en que se encuentran y la operación de los instrumentos. Los modelos de simulación empleados reproducen el comportamiento de la central en todos sus intervalos de operación, desde arranque "en frío" hasta la operación a plena carga, lo que incluye las maniobras de operación como son los cambios en la energía generada, las pruebas de arranque normal y automático de los sistemas de emergencia de la unidad, así como la puesta en "listo para operar en forma automática" de los mismos.

Los modelos reproducen el comportamiento de la central bajo condiciones anormales de operación como son fallas y contingencias de funcionamiento de diversos equipos, que producen perturbaciones y estados transitorios en los sistemas que la forman. Las fallas simuladas son aquellas que con frecuencia ocurren o que por su gravedad de consecuencias son muy importantes para la operación segura y la integridad de la central. Las fallas genéricas que se pueden simular son 175 y más de 400 si se consideran diversos grados de severidad de varias de las fallas y la multiplicidad del equipo de la planta. [5]

El simulador comprende un cuarto de control, un sistema de computadoras con el software de simulación, una consola del instructor y una consola de monitoreo de procesos que simula a la consola de procesos de la CNLV. El cuarto de control consta de 21 tableros de operación. Las señales enviadas por la computadora a los instrumentos (salidas) y las transmitidas de los instrumentos a la computadora (entrada), son manejadas por el sistema de entrefaces, que se forma

por 87 unidades independientes de adquisición o controladores; que a su vez están formados por tarjetas electrónicas que manejan las señales, ya sean digitales o analógicas, según el tipo de instrumento a los que están conectadas. La configuración general del simulador y las interconexiones de sus módulos más importantes se muestran en la figura 1:

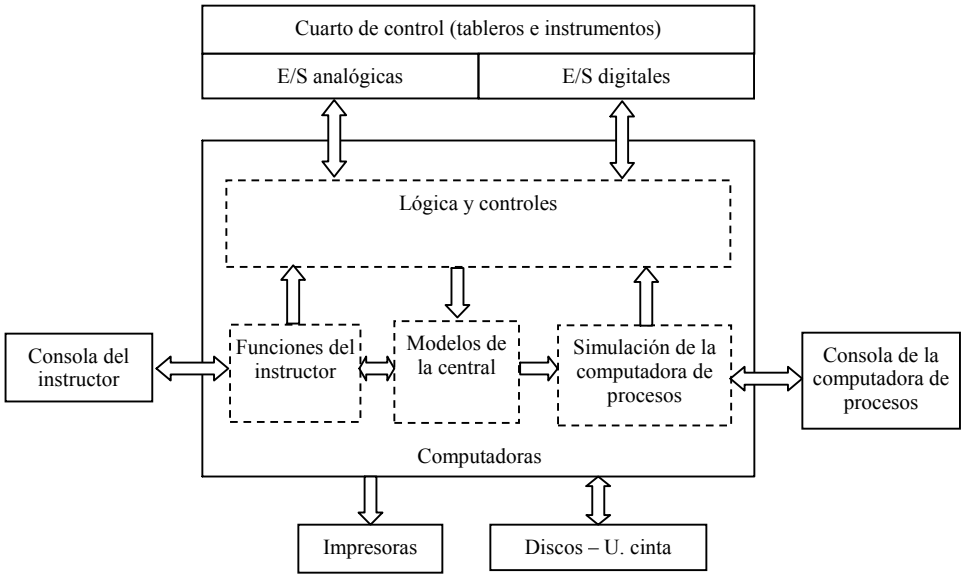


Figura 1.1 Configuración general del simulador.

1.6 El Simulador de Aula.

En el Laboratorio de Análisis en Ingeniería de Reactores Nucleares de la Facultad de Ingeniería actualmente se desarrolla un prototipo funcional de un simulador de procesos nucleares para análisis y capacitación en aula. Dicho prototipo, denominado Simulador de Aula, se está desarrollando bajo un esquema de prototipo rápido. El concepto ***Simulador de Aula*** se refiere a la representación y emulación virtual de paneles de control e instrumentación en diversos despliegues gráficos generados por computadora. Debido a que no se utilizan paneles físicos reales, el simulador es lo suficientemente compacto para ser confinado a una superficie pequeña. A discreción del operador, los despliegues gráficos podrían ser proyectados a pantallas para su adecuada visualización y manipulación. [6-8]

El Simulador de Aula Prototipo constituye el punto de partida para el desarrollo e implementación de un Simulador de Aula Avanzado. Una vez instalado, el Simulador de Aula Avanzado permitirá al personal de operación analizar diversos escenarios operacionales, que ahora son prácticamente imposibles en un modo interactivo. Se espera que el simulador sea usado no sólo como alternativa para reducir la carga actual del simulador para entrenamiento básico e intermedio, sino también como una potente herramienta analítica de diseño, investigación, y docencia.

Actualmente se cuenta con mobiliario (consola del operador, consola del instructor), hardware (computadoras, estaciones de trabajo, monitores) y software de desarrollo (códigos nucleares, paquetería gráfica, compiladores, etc.). Se han desarrollado tres módulos y aplicaciones que permiten realizar varias de las funciones deseadas para el simulador.

1.6.1 Arquitectura general del simulador de aula.

La plataforma computacional empleada en el desarrollo del simulador de aula está constituida por estaciones de trabajo bajo sistema operativo UNIX. La integración de los diversos constituyentes y/o procesos del sistema, se efectúa a través de segmentos dedicados de memoria RAM. Dichos segmentos son accesibles a los diversos componentes mediante programas de comunicación escritos en lenguaje C para tal fin. El uso de memoria compartida (descrito en detalle en el Capítulo 2) y de programas de comunicación especializados proporciona un sistema abierto con una alta modularidad y configurabilidad.

La dinámica para la simulación de los diversos paneles de control es controlada por el módulo de paneles de control. En su modo de operación normal, el proceso del panel de control provee parámetros de salida de controladores manuales o automáticos al módulo de simulación de la planta, vía la memoria compartida de la consola de control, mientras que recibe los parámetros de salida, del módulo de simulación de la planta a través del segmento de memoria compartida de simulación de la planta. [7]

Por medio de sus correspondientes segmentos de memoria compartida, ambos módulos, el de paneles de control y el de simulación de la planta, se comunican con el módulo de monitoreo de parámetros de la planta y/o el módulo de visualización de la planta. Los módulos de monitoreo de parámetros de la planta y el de visualización reciben datos de entrada a través de sus respectivos segmentos de memoria compartida. Estos dos módulos también tienen la capacidad de enviar parámetros de control vía sus segmentos de salida de memoria compartida. El módulo de monitoreo de parámetros es usado para monitorear y controlar los parámetros de seguridad de la planta, mientras que el módulo de visualización se usa para la visualización y control de componentes y sistemas durante secuencias de accidentes severos.

1.6.2 Interfaz del operador.

Uno de los principales componentes del sistema es la interfaz hombre-máquina, con la cual el usuario interactúa con el sistema. La interfaz permite la representación virtual y manipulación de sistemas, paneles de control e instrumentación, así como diversos despliegues de información y diagramas interactivos, vía la manipulación de objetos gráficos que representan la instrumentación de monitoreo y control. La interfaz también permite navegar entre los diversos despliegues involucrados en forma interactiva. [8]

Los despliegues gráficos utilizados en los módulos de interfaz gráfica están diseñados para ser usados en monitores sensibles al tacto y fueron desarrollados usando el paquete DataViews [9] bajo ambiente UNIX. Este software permite manipular objetos gráficos con diversas dinámicas establecidas por las librerías y funciones de DataViews. Crea y edita los objetos, manipula datos y prueba las vistas con un editor propio del software.

La interfaz del operador está compuesta de tres interfaces gráficas, la interfaz de los paneles de control, la interfaz de los parámetros de seguridad de la planta y la interfaz de visualización.

Las ***interfaz de los paneles de control*** es del tipo de manipulación directa que simulan los paneles de control. Proporcionan una representación virtual de los instrumentos y controles cuya

funcionalidad es similar a los del simulador réplica actual. Por medio de la manipulación directa, el sistema permite al operador visualizar y manipular la instrumentación vía interacción con el ratón o mediante pantallas sensibles al tacto (touch-screen).

Actualmente el sistema contiene una representación completa de los paneles BB-09, BB-10 y BB-11 de la CNLV. Estos paneles contienen la instrumentación asociada a los sistemas de refrigeración de emergencia, agua de servicio nuclear, recirculación e instrumentación y control nuclear. Los despliegues gráficos que representan los paneles de control se han diseñado para que tengan una alta similitud con los paneles del simulador de entrenamiento actual, pero tratando de evitar un realismo excesivo e innecesario que podría crear un ambiente complejo para el operador. Las figuras 1.2, 1.3 y 1.4 muestran las interfaces que simulan dichos paneles de control.

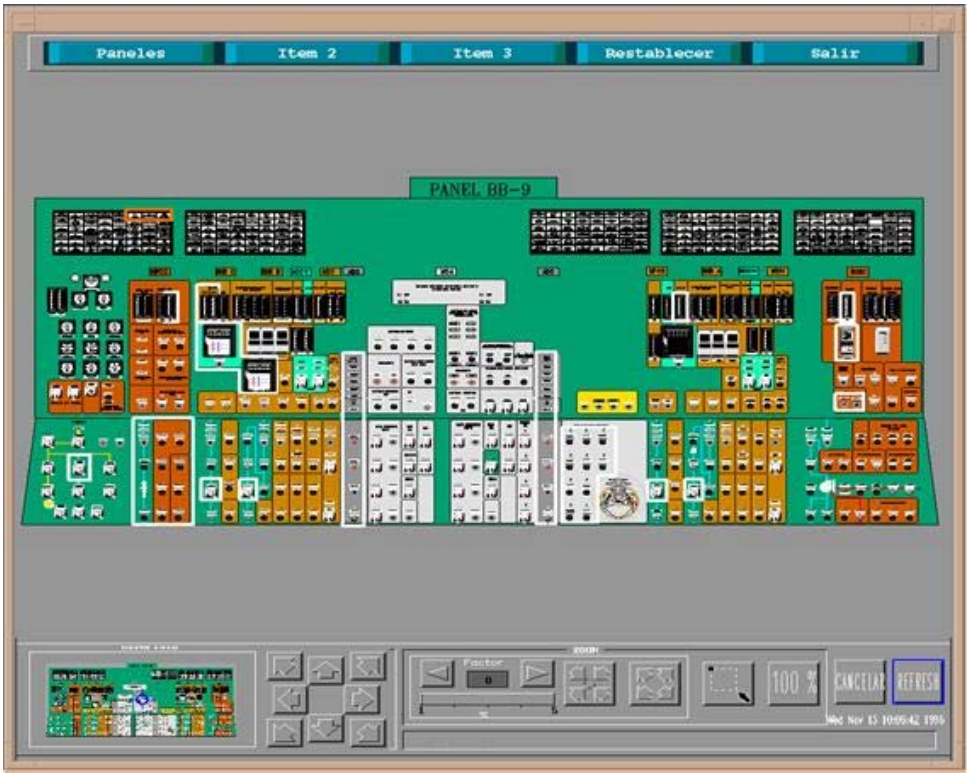


Figura 1.2 Panel BB-09

Figura 1.3. Panel BB-10

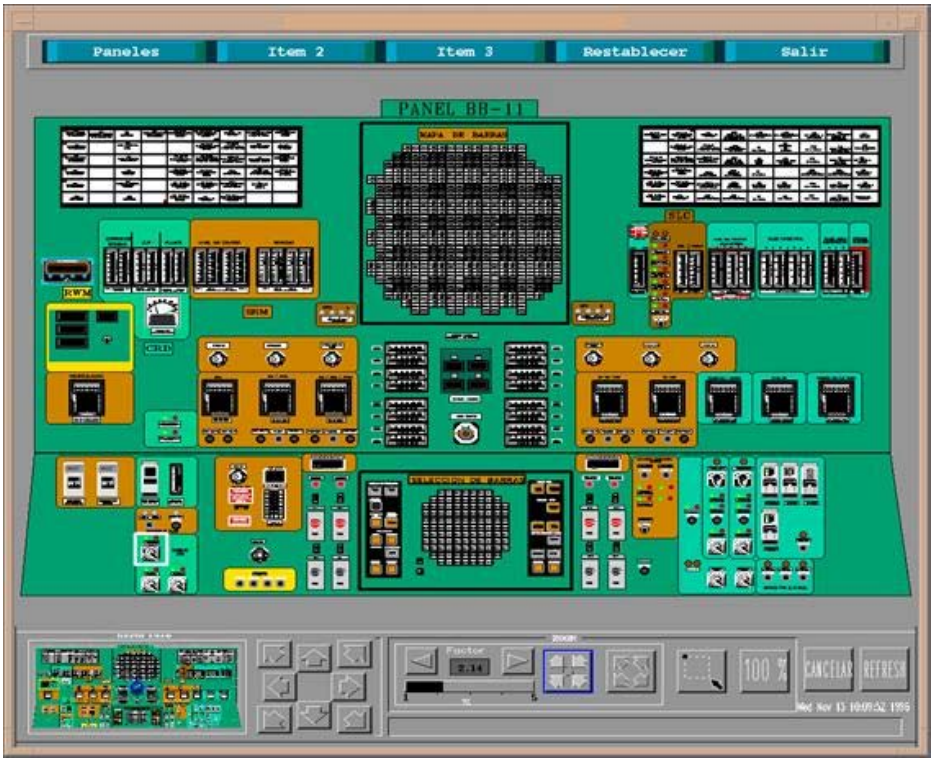


Figura 1.4. Panel BB-11

Los despliegues del módulo de los paneles de control permiten manipular parámetros del módulo de simulación de la planta para la reproducción de diferentes escenarios operacionales. También permiten el monitoreo de parámetros críticos mediante instrumentos de medición e informativos. La interacción con los despliegues recrea la manipulación en un tablero real de la planta.

La interfaz de los paneles de control está dividida en tres áreas: en la parte superior de la pantalla se encuentran el área de menús con diferentes opciones para el manejo de archivos, acceso a la información suplementaria y control de subprocesos. El área central es donde se despliegan los paneles de control y finalmente, en la parte inferior de la interfaz se localiza el área dedicada al control de la visualización de los despliegues (menú de navegación).

La ***interfaz de los parámetros de seguridad*** de la planta es una copia simplificada de algunos despliegues del SEPS (Sistema de Exhibición de Parámetros de Seguridad) de la CNLV. Esta copia contiene solamente los despliegues relacionados con los actuales procesos simulados, entre ellos están los diagramas mímicos de los sistemas de refrigeración de emergencia, despliegues de gráficas de tendencia y gráficas parámetro vs. parámetro.

La ***Interfaz de visualización*** es muy similar en concepto y equivalente en funcionalidad a la interfaz del control de proceso. La diferencia es que esta interfaz permite el acceso a un conjunto de despliegues gráficos que representan la vasija del reactor, los sistemas de refrigeración de emergencia y la contención primaria. Estos despliegues han sido construidos para la animación y visualización de eventos transitorios y para visualización de accidentes severos.

La figura 1.5 muestra el despliegue principal de visualización de eventos transitorios, el cual es una representación gráfica de la vasija del reactor. Incluye el sistema de agua de alimentación, sistema de recirculación, flujo de vapor a la turbina, bombas de sistemas de refrigeración de emergencia y válvulas principales. El despliegue además contiene diversos componentes gráficos para el monitoreo de parámetros principales (posición de válvulas, flujos, nivel, potencia, etc.) así como componentes de control (controladores, disparo de bombas y puntos de ajuste).

La interfaz hombre-máquina para la visualización de eventos transitorios es del tipo manipulación directa mediante la cual es posible controlar el proceso de simulación (código RELAP5) interactuando directamente sobre los principales componentes gráficos mediante el ratón o en pantallas sensibles al tacto.

Adicionalmente se tiene un despliegue para gráficas de tendencia, el cual contiene 19 de las principales variables en forma parametrizada, necesarias para un análisis adecuado de los principales eventos.

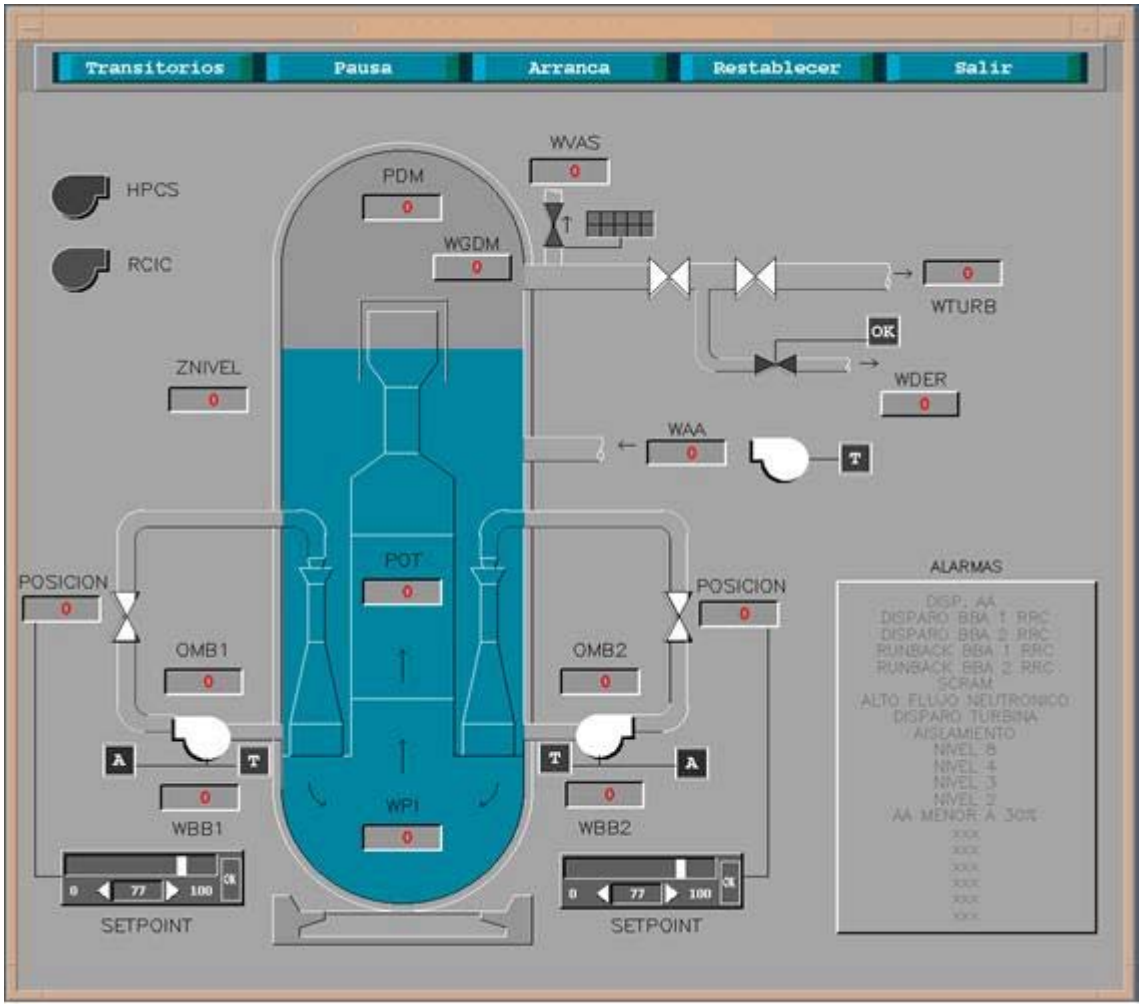


Figura 1.5. Despliegue para visualización de eventos transitorios.

La interfaz hombre-máquina para la visualización de eventos transitorios contiene un menú de opciones de control, mediante el cual es posible seleccionar un evento entre una lista preestablecida, así como efectuar el paro/arranque de la simulación, pausa, restablecer y salida. Finalmente, el despliegue cuenta con un anunciador de las principales alarmas para alertar al usuario sobre el desarrollo de cualquier evento transitorio.

En cuanto al despliegue de visualización de código accidentes severos (ver Figura 1.6), es una representación gráfica de la vasija del reactor, núcleo y principales componentes, sistemas de refrigeración de emergencia y contención primaria. El despliegue está orientado a proporcionar una animación completa de sistemas, componentes y parámetros para la visualización, control y principalmente el análisis de secuencia de accidentes severos vía interacción en línea y fuera de línea (mediante archivos de datos de salida) con el código RELAP/SCDAP.

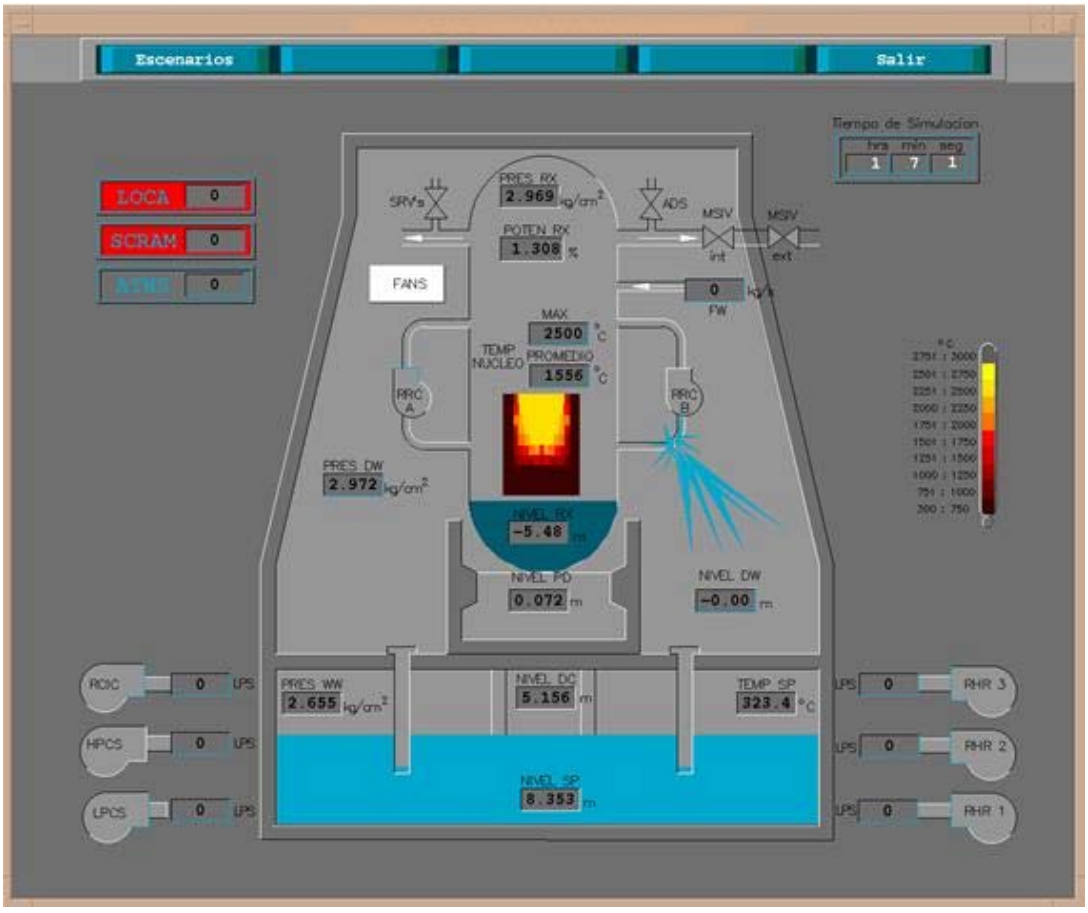


Figura 1.6. Despliegue para visualización del código MAAP

1.6.3 Códigos nucleares en el simulador de aula.

El código empleado en el Simulador de Aula RELAP5/SCDAP se desarrolló en el Laboratorio Nacional de Energía y Ambiente de Idaho y es la unión de los códigos RELAP5/MOD3.2 y SCDAP.

RELAP5 calcula la respuesta termohidráulica de los principales sistemas de la planta, interacciones del sistema de control, cinética del reactor y sistema de gases no condensables. Los modelos del código SCDAP calculan el sobrecalentamiento y progresión del daño de las estructuras del núcleo y base de la vasija del reactor.

RELAP5/SCDAP es un código de mejor estimación en la simulación de los sistemas de refrigeración de reactores de agua ligera, durante eventos transitorios y/o secuencias de accidentes severos. El código puede modelar el comportamiento del sistema refrigerante del reactor, núcleo del reactor, liberación de productos de fisión, accidentes de pérdida de refrigerante por ruptura, transitorios operacionales, pérdida de potencia del sistema, pérdida de agua de alimentación y pérdida de flujo de recirculación.

El código incluye diversos modelos de componentes genéricos que incluyen barras de combustible, barras de control, bombas, válvulas, tubos, separadores, turbinas, acumuladores, bombas de chorro, calentadores eléctricos, estructuras de calor, cinética puntual del reactor y componentes del sistema de control. Es capaz de modelar un amplio rango de configuraciones de sistemas, desde un sistema hidráulico compuesto por simples tubos hasta sistemas experimentales del reactor.

SCDAP/RELAP5 tiene una estructura modular descendente en el que los procedimientos y modelos se encuentran en subrutinas separadas. En la figura 1.7 se muestra la arquitectura del código. La estructura del código consiste en una subrutina principal (INPUTD) para procesar los archivos de entrada, en el que se encuentran los modelos que serán simulados. En la subrutina TRNCTL se procesa la simulación en estado estable o en un transitorio, accidente severo, etc., y en

STRIPF se procesan los bloques de salida. La simulación generada en la simulación se deposita en archivos de salida que será utilizada por otros módulos del simulador.

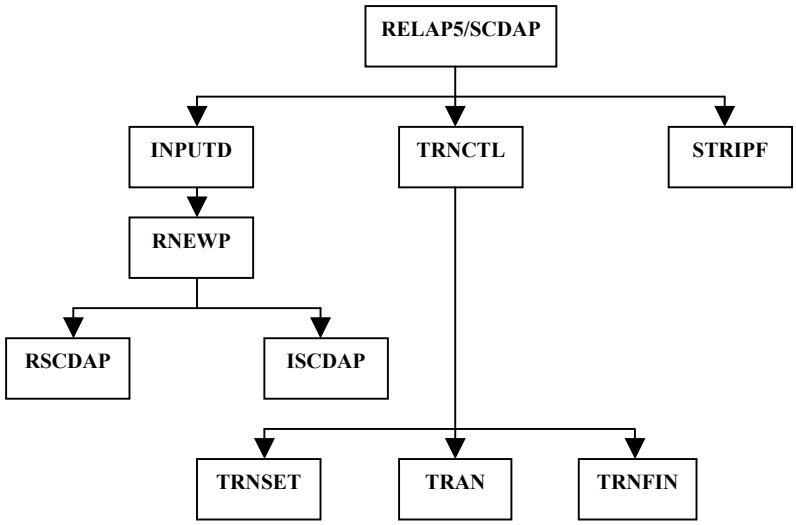


Figura 1.7. Arquitectura General RELAP5/SCDAP

El código procesa la información del archivo de entrada en un formato basado en el concepto de tarjetas, en donde cada línea se identifica por un número de tarjeta. Para un número de tarjeta el código procesa los datos según los campos especificados en el manual del código. Frecuentemente se construye la representación gráfica contenida por bloques rectangulares que representan volúmenes hidrodinámicos unidos por líneas, que constituyen las condiciones de frontera entre éstos. A esta representación se le llama diagrama de nodalización. [10]. Un ejemplo sencillo de un diagrama de nodalización se muestra en la figura 1.8.

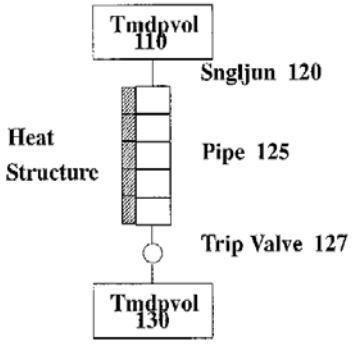


Figura 1.8. Ejemplo de un diagrama de nodalización.

Utilizando el sistema de control y los componentes secundarios, se pueden modelar los controles de la central, turbinas, condensadores y los sistemas de acondicionamiento del agua de alimentación. Con RELAP/SCDAP se puede modelar sistemas de distintas configuraciones, para lo cual los modelos usan un número arbitrario de volúmenes de control de fluidos y uniones de conexión, estructuras de calor, componentes del núcleo y componentes del sistema.

CAPÍTULO 2

MEMORIA COMPARTIDA

El presente capítulo describe brevemente el sistema de comunicaciones integrado al sistema operativo UNIX System V así como los elementos conceptuales para la creación y utilización de memoria compartida.

2.1 Memoria compartida en UNIX SYSTEM V.

UNIX es un sistema interactivo diseñado para manejar múltiples usuarios y múltiples procesos al mismo tiempo. Los dos conceptos centrales sobre los que se basa la arquitectura de UNIX son los archivos y los procesos, las únicas entidades activas en el sistema son los procesos. La figura 2.1 muestra los niveles hardware, núcleo y usuario de la arquitectura en la que notamos que el núcleo está dividido en dos subsistemas principales: subsistema de archivos y subsistema de control de procesos.

El subsistema de archivos controla los recursos del sistema de archivos y tiene como función reservar espacio para los archivos, administrar el espacio libre, controlar el espacio a los archivos, permitir el intercambio de datos entre los archivos y el usuario, etc. El subsistema de archivos se comunica con los dispositivos de almacenamiento secundario (discos duros, unidades de cinta, etc.) mediante los manejadores de dispositivo.

Los procesos interaccionan con el subsistema de archivos con llamadas específicas: open, read, write, etc. El subsistema de control de procesos es el encargado de la planificación de los procesos, de su sincronización, de la comunicación entre los mismos (IPC, *Interprocess Comunication*) así como del control de la memoria principal.

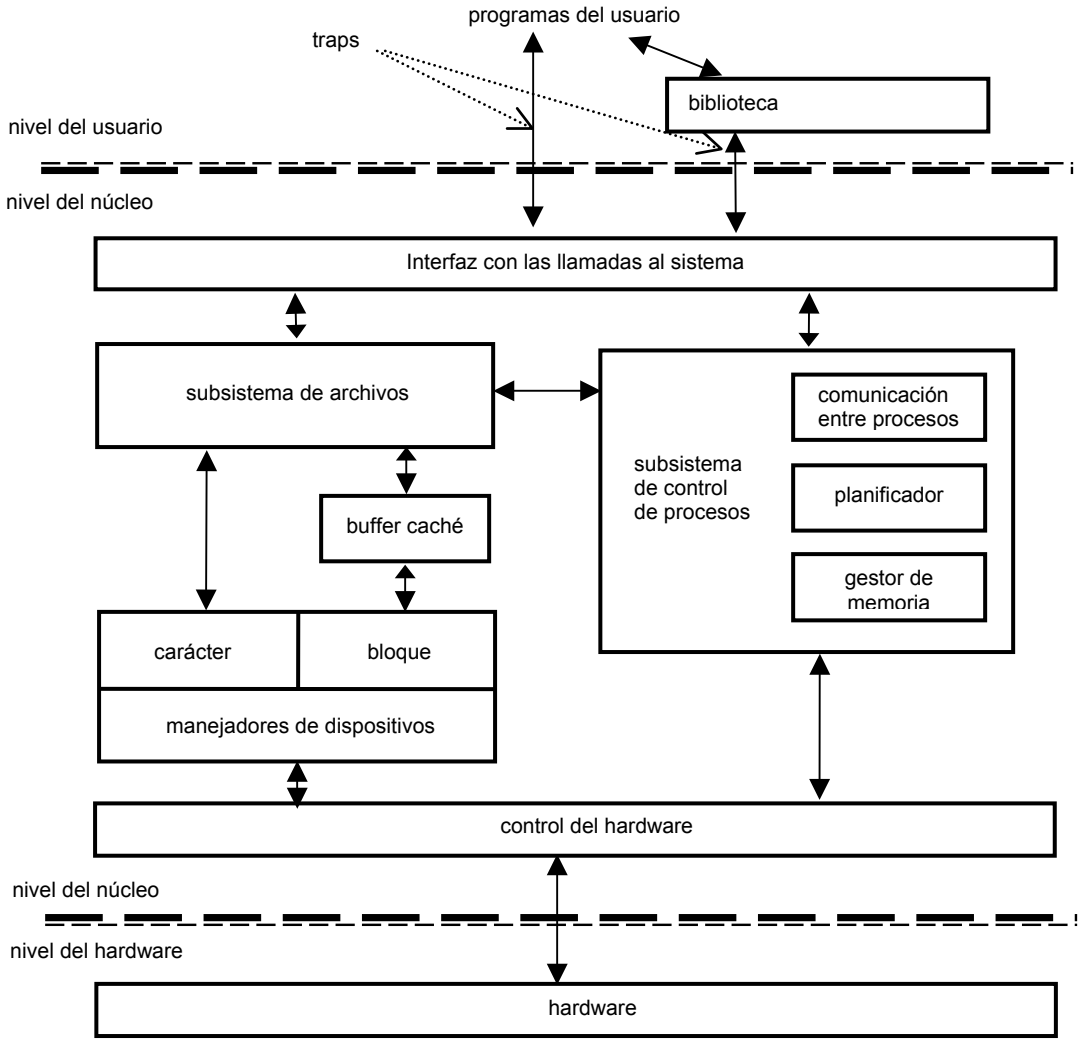


Figura 2.1 Diagrama de bloques del núcleo del sistema operativo UNIX.

El módulo de gestión de memoria es el encargado de controlar los procesos cargados en memoria principal en cada momento. Si no hay suficiente memoria el gestor recurrirá a los mecanismos de intercambio para que todos los procesos tengan un tiempo mínimo de ocupación de la memoria y se puedan ejecutar. El intercambio significa llevar los procesos cuyo tiempo de ejecución expira a memoria secundaria y traer de ésta los procesos a los que se les asigna tiempo de ocupación de memoria principal. El planificador gestiona el tiempo de CPU asignado a cada proceso y decide si el proceso actual tiene derecho a ejecutarse o asigna el tiempo de CPU a otro proceso. [11,12]

Las estructuras IPC se componen de tres mecanismos: semáforos, memoria compartida y cola de mensajes. Los semáforos sirven para sincronizar procesos, la memoria compartida permite que los procesos compartan su espacio de direcciones virtuales y las colas de mensajes posibilitan el intercambio de datos. Las estructuras IPC se crean mediante la función get: semget para semáforos, msgget para colas de mensajes y shmget para memoria compartida. Se encuentran en el sistema como una unidad y comparten las siguientes características comunes:

- Tienen una tabla cuyas entradas describen el uso que se hace de cada mecanismo.
- Cada entrada de la tabla tiene una llave numérica.
- Dispone de una llamada para crear una entrada nueva o recuperar alguna ya existente.
- Cada entrada de la tabla tiene un registro de permisos que incluye: identificador de usuario y de grupo, permisos de lectura, escritura y ejecución.
- Información de estado, identificador del último proceso que utilizó la entrada.
- Llamada de control que permite leer y modificar el estado de una entrada reservada y que también permite liberarla.

Las estructuras IPC se encuentran en el núcleo del sistema operativo. Se le llama objetos IPC a cualquiera de las estructuras anteriores. La memoria compartida es un segmento de memoria configurada por el núcleo con el propósito de intercambiar información entre diferentes procesos, permitiendo que procesos no relacionados (procesos que no tienen un padre común) se comuniquen utilizando memoria compartida. De este modo los datos fluyen libremente entre procesos utilizando mecanismos IPC.

La forma más rápida de comunicar dos procesos es hacer que compartan una zona de memoria (Figura 2.2). Accedemos a cada objeto a través de su identificador, un entero positivo que identifica el objeto y su tipo. Cada identificador es único para cada tipo, pero se puede utilizar el mismo valor de identificador para un segmento de memoria compartida. De este modo el identificador de convierte en el manipulador de todas las operaciones de la estructura.

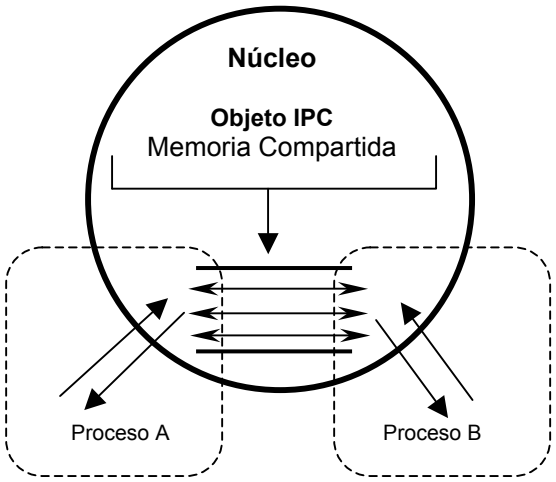


Figura 2.2 Procesos no relacionados intercambian datos.

Si los permisos están establecidos en forma adecuada, cada proceso que desee acceder al segmento puede hacer correspondencia con él en su propio espacio de direcciones privado. Si un proceso actualiza los datos del segmento la actualización es vista de inmediato por los procesos que la acceden. Un segmento creado por un proceso puede ser leído o escrito por otros procesos. Múltiples procesos comparten el acceso al segmento y a la información que contiene como se muestra en la figura 2.3.

El segmento de memoria compartida puede consistir en datos de la RAM física como en páginas de memoria que se han cambiado al disco. Lo mismo se aplica para al espacio de direcciones de los procesos que se adjunta al segmento de la memoria compartida (SHM), cuya una o más páginas pueden estar residiendo transitoriamente en el disco, aunque la transferencia de datos ocurre estrictamente en memoria, en la RAM.

En la Figura 2.3 el segmento de memoria creado en memoria principal, mostrado como un cuadro compartido sombreado (SHM) con los procesos B y C muestran que los dos procesos han hecho corresponder el segmento en sus propios espacios de direcciones, o sea que han mapeado dicho segmento a sus propios espacios de memoria. También se observa que cada uno de los cuatro procesos tiene su propio espacio en memoria que se mapea a alguna región de la RAM física, pero este espacio no está disponible para los demás por ser el espacio en memoria de un proceso privado.

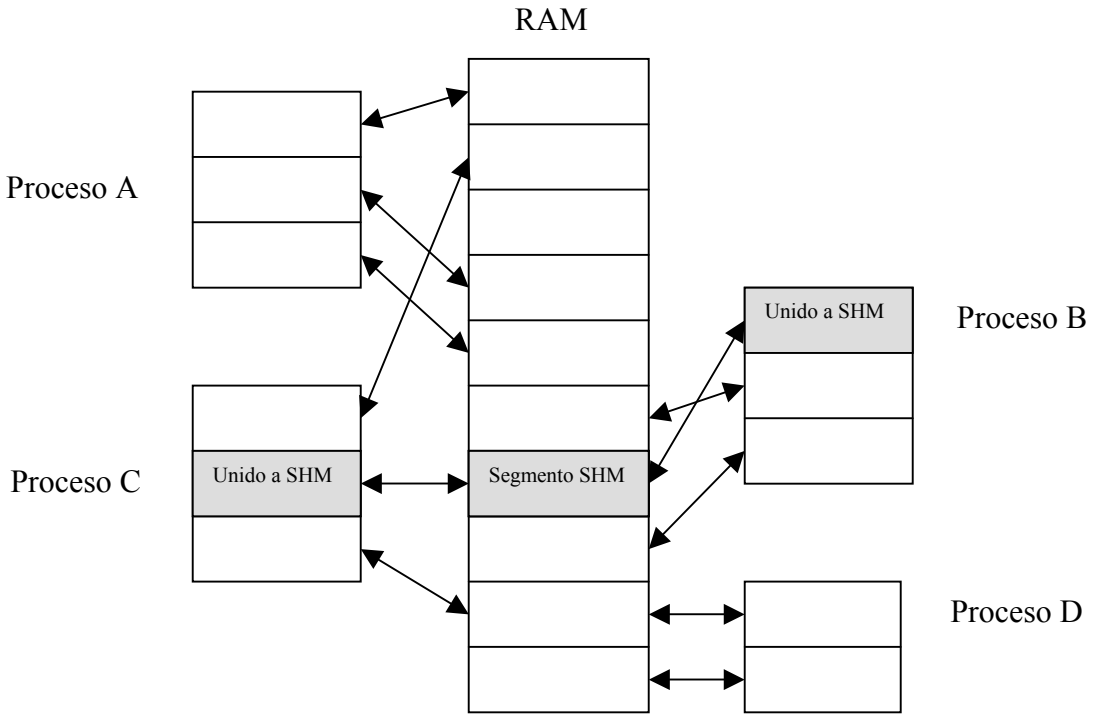


Figura 2.3 Correspondencia de segmentos de memoria compartida.

Cada estructura tiene un modo, o conjunto de permisos, semejante al modo de un archivo solo que en las estructuras IPC no existe el concepto de permisos de ejecución. Al crear una estructura IPC uno debe efectuar una operación en octal en el argumento **shmflg** con los permisos específicos o no se podrá acceder a la estructura creada. Algunas de las limitaciones del IPC System V son [11,13]:

- Interfaz de programación compleja para los beneficios que provee. Debido a que las estructuras IPC están sólo en el núcleo son desconocidas para el sistema de archivos por lo que, para realizar E/S en ellas requiere aprender otra interfaz de programación.

- Número restringido de estructuras IPC que un sistema permite tener en uso al mismo tiempo, comparándolo con el número de archivos que puede tener abiertos o el número de procesos activos que permite.

- Las estructuras IPC no mantienen un conteo de referencia, o recuento del número de proceso que está utilizando una estructura al mismo tiempo. Por lo que IPC System V no recupera las estructuras abandonadas con algún mecanismo de manera automática.

Si se crea una estructura IPC, se le ingresan datos a la misma y se termina sin eliminar ni la estructura ni los datos, la estructura permanecerá allí hasta que:

- Se reinicie el sistema.
- Se emplee el comando **ipcrm** (*remove IPC structure*) sobre la estructura.
- Otro proceso con los permisos de acceso lea los datos, elimine la estructura o ambas.

2.2 Manejo de memoria compartida. [11,13]

2.2.1 Creación de un segmento de memoria compartida.

La función necesaria para la creación de memoria compartida es shmget. Los archivos de encabezado requeridos para utilizar cualquier función relacionada con la memoria compartida son ‹sys/types.h›, ‹sys/ipc.h›, ‹sys/shm.h›. Si shmget tiene éxito regresa el identificador de segmento, un número entero no negativo, si fracasa regresa –1. Su prototipo es:

int **shmget**(key_t *key*, int *size*, int *shmflg*);

donde:

int *key* llave numérica de identificación del segmento de memoria.

int *size* tamaño en bytes de la zona de memoria que queremos crear.

int *shmflg* máscara de bits que indican el modo de adquisición del identificador. Los 9 bits menos significativos de *shmflg* indican los permisos del segmento, 0 si el

segmento ya está creado. Los bits de permisos deben estar escritos en notación octal.

Numérico	Simbólico	Permisos
0400	SHM_R	lectura para el usuario
0200	SHM_W	modificación para el usuario
0060	SHM_RW	lectura y modificación para el grupo
0040	SHM_R	lectura para el grupo
0020	SHM_W	modificación para el grupo
0006	SHM_RW	lectura y modificación para el resto de usuarios
0004	SHM_R	lectura para el resto de usuarios
0002	SHM_W	modificación para el resto de usuarios
	IPC_CREAT	Bandera IPC especial
	IPC_EXCL	Bandera IPC especial

IPC_CREAT indica que si aún no existe ningún segmento asociado con *key* deberá ser creado uno nuevo.

IPC_EXCL garantiza que si el segmento ya existe la llamada fracasará, en vez de regresar el identificador de un segmento ya asignado.

El siguiente código muestra cómo crear una zona de memoria de 4096 bytes, donde todos los usuarios tendrán permiso de lectura y escritura. Muestra el identificador que retorna shmget y la salida del comando ipcs –m el número de procesos adosados al segmento.

```
#include <sys/types.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

#define TAMANO_BUF 4096
int main(void)
```

```
{
        int identificadorSegmento;
        if((identificadorSegmento = shmget(IPC_PRIVATE, TAMANO_BUF, 0666))<0 {
        perror("shmget");
        exit(EXIT_FAILURE);
        }
        printf("Identificador del segmento creado: %d\n", identificadorSegmento);
        system("ipcs -m");
        exit(EXIT_SUCCESS);
}
```

2.2.2 Operaciones con memoria compartida. **shmat** (atar) y **shmdt** (desatar).

Un proceso no puede utilizar un segmento de memoria compartida hasta que lo atemos, o sea que el proceso mapee la dirección del mismo a su propio espacio de memoria. Se adjunta el segmento utilizando la llamada shmat, para cancelar y quitar el correspondiente mapeo se requiere la llamada shmdt.

Las llamadas al sistema para realizar estas operaciones tienen los siguientes prototipos:

char ***shmat** (int *shmid*, char **shmaddr*, int *shmflg*);

donde:

shmid es el identificador de una zona de memoria creada llamando a la función shmget.

shmaddr es la dirección virtual donde queremos que empiece la zona de memoria compartida Una llamada a **shmat** que funcione correctamente devolverá un puntero a la dirección virtual a la que está unido el segmento de memoria compartida. Esta dirección puede o no coincidir con la dirección de *shmaddr*, según la decisión del núcleo. Lo normal es que *shmaddr* valga 0, lo cual deja al núcleo la decisión de la dirección de inicio. Si no vale 0 el núcleo intentará satisfacer la petición del usuario, aunque no siempre se consigue. El último argumento *shmflg* es la máscara de bits de la forma de acceso al segmento.

int **shmdt**(char **shmaddr*);

donde:

shmaddr es la dirección virtual del segmento de memoria compartida que queremos separar del proceso.

Si shmat y shmdt funcionan correctamente, la primera devuelve la dirección a la que está unido el segmento de memoria compartida y la segunda devuelve 0. Si algo falla ambas funciones devuelven –1. El siguiente ejemplo ata un segmento de memoria compartida y luego cancela el vínculo.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        int identificadorSegmento;  /* ID del segmento */
        char buf_sgtomemcomp;  /* puntero a dirección en espacio de mem. del proceso */

        /* se espera que haya un identificador de segmento de la línea de comandos */
        if(argc != 2) {
                puts("MODO DE EMPLEO: atshm <identificador>");
                exit(EXIT_FAILURE);
        }
        identificadorSegmento = atoi(argv[1]);

        /* atar el segmento de memoria compartida */
        if((buf_sgtomemcomp = shmat(identificadorSegmento, 0, 0)) < (char *) 0) {
                perror("shmat");
                exit(EXIT_FAILURE);
        }
```

```
        /* dirección en que ha sido atado el segmento */
        printf("El segmento se ha acoplado en %p\n", buf_sgtomemcomp);

        /* comprobación de la operación anterior */
        system("ipcs -m");

        /* desatar segmento*/
        if((shmdt(buf_sgtomemcomp)) < 0) {
                perror("shmdt");
                exit(EXIT_FAILURE);
        }
        puts("Segmento desacoplado\n");

        /* verifica que realmente esté desacoplado */
        system("ipcs -m");
        exit(EXIT_SUCCESS);
}
```

2.2.3 Operaciones de control sobre memoria compartida. **shmctl.**

Podemos realizar operaciones de control con shmctl sobre las zonas de memoria compartida creadas con shmget. Si se realiza exitosamente retorna un valor de 0, si la operación falla retornará el valor –1 y el error será colocado en errno. Su prototipo es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl (int shmid, int cmd, struct shmid_ds *buf);
```

donde:

shmid es un identificador válido devuelto por la llamada a shmget, especifica el ID de la región de memoria compartida.

cmd especifica el tipo de operación de control a realizar. Los comandos posibles son:

IPC_STAT	Lee el estado de la estructura de control de la memoria y lo devuelve a través de la zona de memoria apuntada por *buf*.
IPC_SET	Inicializa algunos de los campos de la estructura de control de la memoria compartida.
IPC_RMID	Borra del sistema la zona de memoria compartida identificada por shmid. Si el segmento está atado a varios procesos, el borrado ocurrirá hasta que todos los procesos liberen la memoria.
SHM_LOCK	bloquea en memoria el segmento identificado por shmid. Lo que significa que no se realizará intercambio sobre él. Sólo los procesos cuyo identificador sea igual al del superusuario podrán realizar esta operación.
SHM_UNLOCK	Desbloquea el segmento de memoria compartida, de modo que se realizará intercambio sobre él cuando se necesite; trasladarlo de memoria principal a la secundaria y viceversa. Sólo los procesos cuyo identificador sea igual al del superusuario podrán realizar esta operación.

shmid_ds es una estructura que se define como sigue:

```
struct shmid_ds
{
        struct ipc_per shm_per;  /* estructura de permisos */
        int  sh_segsz;           /* tamaño del área de memoria compartida */
        int  pad1;               /* usado por el sistema */
        ushort  shm_lpid;        /* PID del proceso que hizo la última operación en el segmento*/
        ushort  shm_cpid;        /* PID del proceso creador del segmento */
        ushort  shm_nattach;     /* número de procesos atados al segmento */
        short  pad2;             /* usado por el sistema */
        time_t  shm_atime;       /* fecha de la última unión al segmento */
        time_t  shm_dtime;       /* fecha de la última separación del segmento */
```

```
        time_t  shm_ctime;    /* fecha del ultimo cambio en el segmento */
};
```

El último argumento *buf* especifica la dirección de la estructura shmid_ds

Por ejemplo, podemos borrar de la RAM una zona de memoria compartida usando:
shmctl (shmid, IPC_RMID, 0);

2.3 Control de recursos IPC desde la línea de comandos. **ipcs** e **ipcrm.**

Mediante los programas estándar **ipcs** e **ipcrm** podemos controlar los recursos IPC del sistema. El primero, **ipcs**, se ocupa para ver los mecanismos que están asignados y a quién, acompañados de información estadística. Mientras que **ipcrm** se emplea para liberar un recurso IPC asignado y dejar libre la entrada que ocupa.

La forma de empleo es: $ ipcs [opciones]

Si no se muestran opciones se mostrará un resumen de la información administrativa almacenada de los recursos IPC asignados. Sólo se muestran las opciones más importantes.

opciones	
-q	muestra información de las colas de mensajes que están activas
-m	muestra información de los segmentos de memoria compartida activos
-s	muestra información de los semáforos que hay activos
-b	informa sobre los tres recursos IPC activos

$ ipcrm [opciones]

opciones

-q msqid borra la cola de mensajes con identificador msqid

-m shmid borra la zona de memoria compartida con identificador shmid

-s semid borra el semáforo con identificador semid

CAPÍTULO 3

DISEÑO, ESPECIFICACIÓN Y DESARROLLO DEL MÓDULO DE ADMINISTRACIÓN DE DATOS.

En este capítulo se describen las bases conceptuales para la utilización de memoria compartida, así como la metodología e implementación del Módulo de Administración de Datos para su incorporación al simulador de aula.

3.1 Bases de Diseño.

La Figura 3.1 muestra un diagrama simplificado del prototipo del simulador. La comunicación entre los procesos en un mismo nodo, o en diferentes nodos, se da mediante la utilización de segmentos de memoria compartida [6,7]. La presente tesis no cubre el proceso de comunicación entre los nodos. El objetivo del Módulo de Administración de datos es el de proporcionar una mayor funcionalidad para generar los segmentos de memoria compartida en forma automatizada en base a los componentes constituyentes del modelo termo-hidrodinámico de entrada y a los elementos del sistema de control, así como la generación de un reporte que permita al usuario conocer las características de cada segmento generado, y que localidades de memoria han sido asignados para almacenar variables de estado, variables de control, valores lógicos y condiciones de disparo de los componentes y sistemas simulados.

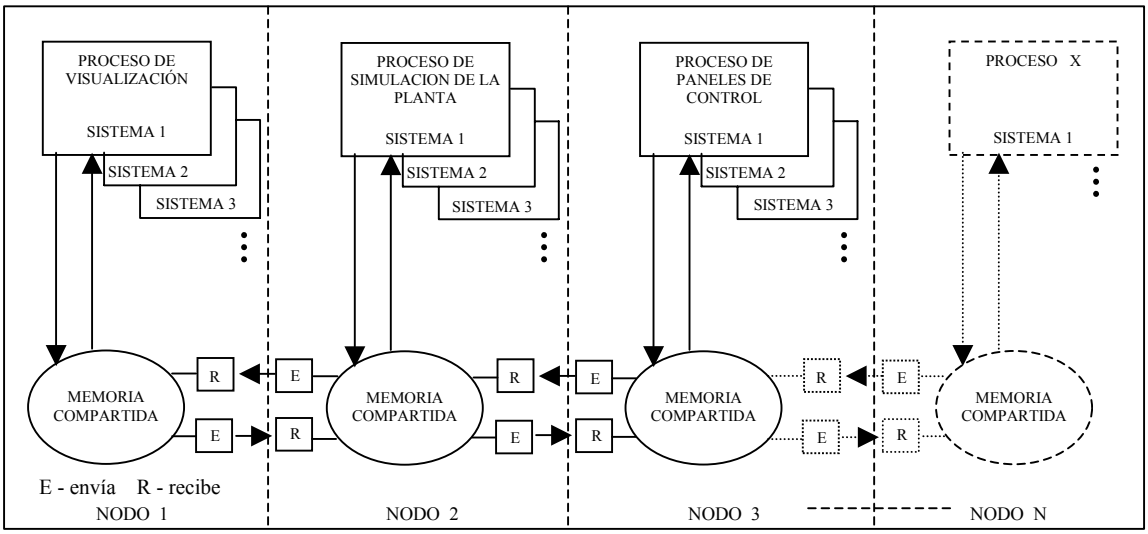


Figura 3.1 Diagrama simplificado del prototipo del simulador.

El código RELAP5 contiene modelos relacionados con componentes del sistema hidrodinámicos, de conducción de calor, cinética del reactor, sistema de control, lógica de disparos y modelos especiales de componentes como son bombas, válvulas, turbinas, etc. La entrada de datos del código es descrita en términos de registros de entrada, o tarjetas, donde cada registro o tarjeta es una línea de 80 caracteres válidos como máximo. Se les llama "tarjetas" debido al modo antiguo de programar mediante tarjetas perforadas. Al leer el archivo de entrada (input deck) el código RELAP5 interpreta los datos línea por línea, manteniendo la filosofía del uso de palabra de tarjeta (word card). Esta interpretación de parámetros enteros, alfabéticos y reales se encuentra descrita en el manual del código NUREG/CR-5535-V2, que cubre la organización de los datos y los requerimientos de las tarjetas de datos [14].

La siguiente lista muestra la clasificación de las tarjetas de entrada con las que se pueden especificar cualquier tipo de problema permitido en el código nuclear RELAP5. El módulo de administración da datos basa su diseño actual en la generación de segmentos de localidades de memoria compartida asociadas a las tarjetas de los componentes hidrodinámicos y a las tarjetas de funciones de control y disparos, debido a que los modelos empleados en la prueba del diseño, emplean básicamente estos elementos [15,16].

- Funciones de Control (100-199)
- Funciones de paso de tiempo (200-299)
- Minor edits (301-399)
- Disparos (400-799 y 20600000-20620000)
- Componentes Hidrodinámicos (CCCXXNN)
- Estructuras de Calor (1CCCGXNN)
- Propiedades Térmicas de las Estructuras de Calor (201MMMnn)
- Tablas generales (202TTTNN)
- Cinética (30000000-3099999)
- Sistema de Control (20500000)

El diseño base del módulo de memoria compartida está formada por hasta 18 segmentos independientes de memoria compartida, cada uno asociado a uno de los elementos involucrados (ver Figura 3.2). Cada segmento es creado a partir de la detección de la presencia de un elemento. En caso de no detectarse un elemento predefinido, no se creará su segmento asociado. Esto permite ahorro en RAM, permitiendo mayores recursos de memoria para los procesos de ejecución. El intercambio de información sobre memoria compartida sólo se realiza en RAM, de allí la importancia de mantener estructuras pequeñas e independientes que puedan ser localizadas o paginadas según su demanda. Estas operaciones de "subir" o "bajar" de RAM estructuras IPC son transparentes al usuario y las gestiona directamente el núcleo del sistema operativo. Sin embargo existe la opción de bloquear los segmentos para que permanezcan en RAM mediante la orden SHM_LOCK, pero esto imposibilita el traslado de memoria principal a secundaria y viceversa [11].

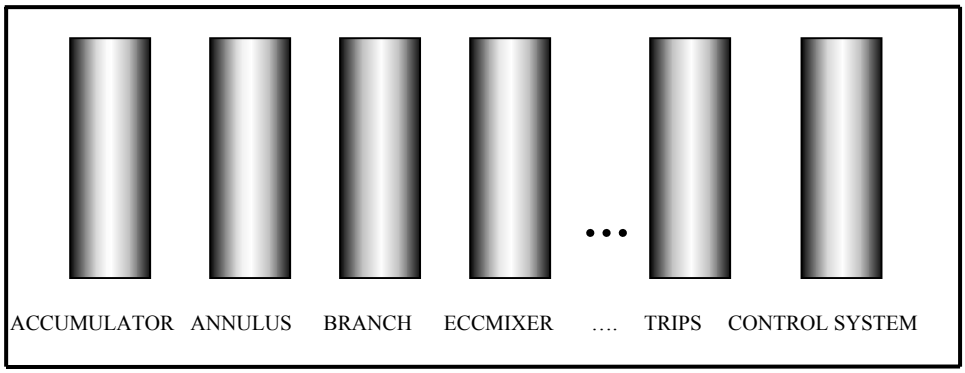
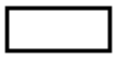
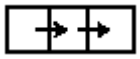
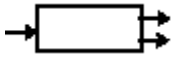


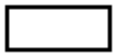
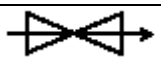
Figura 3.2 Segmentos relocalizables de memoria compartida.

3.2 Componentes hidrodinámicos.

En el caso de los componentes hidrodinámicos el número de tarjeta se encuentra dividido en campos, donde CCC identifica el número de componente, XX es el tipo de tarjeta y NN es un número de tarjeta sin tipo. La Tabla 3.1 muestra un resumen de estos componentes hidrodinámicos utilizando los nombres en inglés y su mnemónico [17].

Tabla 3.1 Componentes hidrodinámicos de RELAP5.

Componente	Etiqueta	Esquema	Uso principal
Single-volume	SNGLVOL		Representa un volumen específico que puede representar parte o todo de un componente o sistema.
Pipe o Annulus	PIPE		Representa un tubo en el sistema. PIPE puede tener de 1 a 100 sub-volúmenes. PIPE con más de 1 sub-volumen tiene uniones internas conectando los sub-volúmenes.
	ANNULUS		Una forma especial de PIPE. Tiene las mismas características de PIPE pero es usado para simular el paso de flujo anular.
Branch	BRANCH		Representa una ramificación de flujo. Puede tener hasta 10 uniones definidas.

Componente	Etiqueta	Esquema	Uso principal
	SEPARATR		Forma especial de BRANCH que simula un separador de agua en un generador de vapor.
	JETMIXER		Forma especial de BRANCH que simula una bomba jet.
	TURBINE		Forma especial de BRANCH que simula una turbina de vapor.
	ECCMIX		Forma especial de BRANCH que simula un mezclador (liquido subenfriado y condensado)
Single-junction	SNGLJUN		Unión simple diseñada para conectar un componente con otro.
Multiple-junction	MTPLJUN		Conecta los componentes hacia otros componentes. Permite hasta 100 conexiones.
Time-dependent volume	TMDPVOL		Especifica las condiciones de estado de un volumen en función del tiempo.
Time-dependent junction	TMDPJUN		Conecta un componente con otro y especifica las condiciones de flujo en función del tiempo.
Valve	VALVE		Simula seis diferentes tipos de válvula: de paro, de disparo, de apertura inercial, motorizada, de servomecanismo y de alivio.
Pump	PUMP		Simula diferentes tipos de bombas.
Accumulator	ACCUM		Simula un acumulador cilíndrico o esférico.

3.3 Descripción de las tarjetas empleadas.

Para la asignación de los segmentos de memoria compartida, el programa desarrollado requiere de información particular para definir las características de cada segmento. Para tal fin se seleccionaron datos específicos de las tarjetas de los componentes hidrodinámicos de RELAP y algunos datos de las tarjetas del sistema de control, esto con el objeto de poder detectar el tipo de componentes hidrodinámicos empleados en un archivo de entrada "input deck", el modo en que

estos componentes se encuentran conectados y algunas variables características de dichos componentes [14]. A continuación se describe por componente la información de referencia.

COMPONENTES HIDRODINAMICOS CCCXXNN

ccc0000 Nombre del componente y tipo.
W1(A) Nombre del componente.
W2(A) Tipo de componente.

Componente Single-Volume

ccc0101 → ccc0109 Datos de volumen Single-Volume coordenada-X
W1(R) Área de flujo de volumen (m^2, ft^2).
W2(R) Longitud de volumen (m, ft).
W3(R) Volumen del volumen (m^3, ft^3).

Componente Time-Dependent Volume

ccc0101 → ccc0109 Tarjetas de geometría Time-Dependent Volume
W1(R) Área de flujo de volumen (m^2, ft^2).
W2(R) Longitud de volumen (m, ft).
W3(R) Volumen of volumen (m^3, ft^3).

Componente Single-Junction

ccc0101 → ccc0109 Tarjetas de geometría Single-Junction
W1(I) Código del volumen anterior.
W2(I) Código del volumen posterior.
W3(R) Área de la unión (m^2, ft^2).

Componente Time-Dependent Junction

ccc0101 Tarjetas de geometría Time-Dependent Junction
W1(I) Código del volumen anterior.
W2(I) Código del volumen posterior.
W3(R) Área de la unión (m^2, ft^2).

Componente Pipe o Annulus

ccc0001 Tarjeta de información para Pipe, Annulus
W1(I) Número de volúmenes.

Componente Branch, Separator, Jetmixer, Turbine o ECC Mixer

ccc0001 Branch, Separator, Jetmixer, Turbine o ECC Mixer
W1(I) Número de uniones.

ccc0101 → ccc0109 Datos de volumen Coordenada-X para Branch, Separator, Jetmixer, Turbine o ECC Mixer
W1(R) Área de flujo de volumen (m^2, ft^2).

W2(R) Longitud de volumen (m, ft).
W8(R) Diámetro hidráulico (m, ft).

cccn101 → cccn109 Tarjeta de geometría de unión Branch, Separator, Jetmixer, Turbine o ECC Mixer

W1(I) Código del volumen anterior.
W2(I) Código del volumen posterior.
W3(R) Área de la unión (m^2, ft^2).

ccc0400 Tarjeta de datos de rendimiento para Turbine

W1(I) Tipo de turbina.
W2(R) Eficiencia actual ho en el punto de diseño de máxima eficiencia.
W3(R) Fracción de reacción de diseño, r.
W4(R) Radio medio de etapa, r (m, ft).

Componente Valve

ccc0101 → ccc0109 Tarjetas de geometría de la Válvula

W1(I) Código del volumen anterior.
W2(I) Código del volumen posterior.
W3(R) Área de la unión (m^2, ft^2).

ccc0300 Tarjeta de tipo de Valvula

W1(A) Tipo de válvula.

Componente Pump

ccc0101 → ccc0107 Tarjetas de geometría de volumen para Pump

W1(R) Área de flujo de volumen (m^2, ft^2).
W3(R) Volumen del volumen (m^3, ft^3).

ccc0108 Tarjeta de unión interna para Pump (Succión)

W1(I) Código de volumen del volumen de conexión interno.
W2(R) Área de la unión (m^2, ft^2).

ccc0109 Tarjeta de unión externa para Pump (Descarga)

W1(I) Código de volumen del volumen de conexión externo.
W2(R) Área de la unión (m^2, ft^2).

Componente Multiple-Junction

ccc0001 Tarjeta de información para Multiple Junction

W1(I) Número de uniones.

ccc0nnm Tarjeta de geometría para Multiple Junction

W1(I) Código del volumen anterior.
W2(I) Código del volumen posterior.
W3(R) Área de la unión (m^2, ft^2).

Componente Accumulator

ccc0101 → ccc0109 Tarjetas de geometría de volumen para Accumulator

W1(R) Área de flujo de volumen (m^2, ft^2).
W2(R) Longitud de volumen (m, ft).
W3(R) Volumen del volumen (m^3, ft^3).
W8(R) Diámetro hidráulico (m, ft).

ccc1101 Tarjeta de geometría de unión para Accumulator
W1(I) Código del volumen posterior.
W2(R) Área de la unión (m^2, ft).

La tarjeta asociada al sistema de control es más sencilla, inicia con el número que la caracteriza 205, seguida de tres dígitos CCC, que permite rangos de 001 a 999 variables de control. Si el formato de la tarjeta incluye CCCC el rango se amplía de 1 a 9999 variables de control.

DATOS DE ENTRADA DEL SISTEMA DE CONTROL 205CCCNN o 205CCCCN

205ccc00 o 205cccc0 Tarjeta para componente de control
W1(A) Nombre alfanumérico.
W2(A) Tipo de componente de control.
W4(R) Valor inicial.

Esta tarjeta permite los siguientes tipos de componentes de control:

Suma-Diferencia, Multiplicación, División, Diferenciación, Integración, Función, Función Estándar, Retardo, Disparo Unitario, Disparo Retardado, Exponente Entera, Exponente Real, Exponente Variable, Integral-Proporcional, Retardo, Constante, Shaft, Control de bomba, Control de Flujo, Control de Alimentación.

DATOS DE ENTRADA PARA DISPAROS "TRIPS" 400 → 799 o 20600000 → 20620000

Los dos rangos de tarjetas representan series de tarjetas diferentes disponibles para la introducción de los datos de disparo, aunque sólo una puede ser usada en cada problema. La primera serie permite 199 disparos de variable y 199 disparos de lógica. La segunda serie permite hasta 1000 disparos de variable y 1000 disparos de lógica.

401 → 599 o 20600010 → 20610000

W1(A) Código de variable.
W2(I) Parámetro.
W3(A) Relación. Permite las siguientes relaciones FORTRAN estándar:
EQ, NE, GT, GE, LT o LE
W4(A) Código de variable.
W5(I) Parámetro.
W6(R) Constante aditiva.
W7(A) Indicador de cerrojo.

601 → 799 o 20610010 → 20620000

W1(I) Número de disparo.
W2(A) Operador. Puede ser AND, OR, XOR.
W3(I) Número de disparo.
W4(A) Indicador de cerrojo.

3.4 Desarrollo.

El módulo se desarrolló en lenguaje ANSI C que es el lenguaje natural para programar el paquete de comunicaciones UNIX System V. El diagrama en la Figura 3.3 muestra la estructura de bloque del módulo.

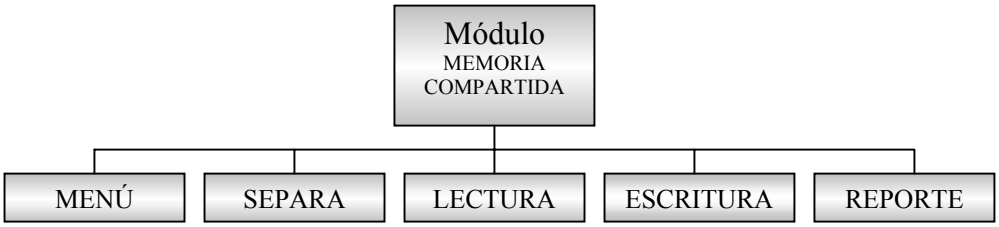


Figura 3.3 Funciones relacionadas con el Módulo Memoria Compartida.

3.4.1 Descripción de los Bloques del Módulo de Administración de Datos.

MENÚ.

Este bloque permite seleccionar funciones de control del programa, generar el reporte y terminación del programa.

SEPARA.

Identifica los números de tarjeta capturando información contenida en éstas. Utiliza el bloque de escritura para almacenar la información en los segmentos de memoria compartida que correspondan.

LECTURA.

Obtiene la información contenida en las localidades de memoria compartida de los segmentos generados.

ESCRITURA.

Almacena la información en las localidades de memoria compartida de los segmentos correspondientes.

REPORTE.

Despliega en pantalla el contenido de los segmentos de memoria existentes, generando al mismo tiempo un archivo de salida.

Definido el diseño e identificados los componentes y sus variables, el primer paso de la implementación fue traducir esta información a lenguaje C para generar las estructuras correspondientes, integrando las variables de acceso y de identificación de segmento. La implementación de los segmentos de memoria compartida la forman arreglos de estructuras de los tipos predefinidos que se muestran a continuación:

```
/* sistema de control */
typedef struct {
     int channel;         /* canal */
     long idvc;           /* identificador de variable de control (#tarjeta) */
     char nomvc[LVAR];    /* nombre de variable de control  */
     double valvc;        /* valor de la variable de control  */
     char tipvc[LVAR];    /* tipo de variable de control  */
     int numshmid;        /* identificador de segmento  */
} CONSYS;                 /* estructura para control system */

/* componentes hidrodinámicos */
typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
```

```
     double valhid;       /* valor de componente hidrodinámico */
     double volflowa;     /* área de flujo del volumen */
     double lenvol;       /* longitud del volumen */
     double volvol;       /* volumen del volumen */
     int numshmid;        /* identificador de segmento  */
} SNGLV;                  /* estructura para single_volume */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     double volflowa;     /* área de flujo del volumen */
     double lenvol;       /* longitud del volumen */
     double volvol;       /* volumen del volumen */
     int numshmid;        /* identificador de segmento  */
} TMDPV;                  /* estructura para time-dependent volume */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     long fromconn;       /* dirección de origen de la unión */
     long toconn;         /* dirección de destino de la unión */
     double juna;         /* área de la unión */
     int numshmid;        /* identificador de segmento  */
} SNGLJ;                  /* estructura para single-junction */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     long fromconn;       /* dirección de origen de la unión */
     long toconn;         /* dirección de destino de la unión */
     double juna;         /* área de la unión */
     int numshmid;        /* identificador de segmento  */
} TMDPJ;                  /* estructura para time-dependent junction */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     int numvols;         /* número de volúmenes */
     int numshmid;        /* identificador de segmento  */
} PIP;                    /* estructura para pipe */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     int numvols;         /* número de volúmenes */
     int numshmid;        /* identificador de segmento  */
```

```
} ANNL;                    /* estructura para annulus */


typedef struct {
     int channel;          /* canal */
     long idcomp;          /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];   /* nombre del componente */
     char cmptype[LVAR];   /* tipo de componente */
     double valhid;        /* valor de componente hidrodinámico */
     int numjun;           /* número de uniones */
     double vfa;           /* área de flujo del volumen */
     double longvol;       /* longitud del volumen */
     double diamhi;        /* diámetro hidráulico */
     long fromconn;        /* dirección de origen */
     long toconn;          /* dirección de destino */
     double juna;          /* área de la unión */
     int numshmid;         /* identificador de segmento  */
} BRAN;                    /* estructura para branch */

typedef struct {
     int channel;          /* canal */
     long idcomp;          /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];   /* nombre del componente */
     char cmptype[LVAR];   /* tipo de componente*/
     double valhid;        /* valor de componente hidrodinámico */
     int numjun;           /* número de uniones */
     double vfa;           /* área de flujo del volumen */
     double longvol;       /* longitud del volumen */
     double diamhi;        /* diámetro hidráulico */
     long fromconn;        /* dirección de origen*/
     long toconn;          /* dirección de destino */
     double juna;          /* área de la unión */
     int numshmid;         /* identificador de segmento  */
} SEPR;                    /* estructura para separator */

typedef struct {
     int channel;          /* canal */
     long idcomp;          /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];   /* nombre del componente */
     char cmptype[LVAR];   /* tipo de componente*/
     double valhid;        /* valor de componente hidrodinámico */
     int numjun;           /* número de uniones */
     double vfa;           /* área de flujo del volumen */
     double longvol;       /* longitud del volumen */
     double diamhi;        /* diámetro hidráulico */
     long fromconn;        /* dirección de origen*/
     long toconn;          /* dirección de destino */
     double juna;          /* área de la unión */
     int numshmid;         /* identificador de segmento  */
} JETMX;                   /* estructura para jetmixer */

typedef struct {
     int channel;          /* canal */
     long idcomp;          /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];   /* nombre del componente */
     char cmptype[LVAR];   /* tipo de componente*/
     double valhid;        /* valor de componente hidrodinámico */
     double vfa;           /* área de flujo del volumen */
     double longvol;       /* longitud del volumen */
     int tipotur;          /* tipo de turbina */
     double efic;          /* eficiencia */
```

```
     double fracrr;       /* fracción de reacción de diseño r */
     double radmede;      /* radio medio de etapa */
     int numshmid;        /* identificador de segmento  */
} TURB;                   /* estructura para turbina */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     int numjun;          /* número de uniones */
     double vfa;          /* área de flujo del volumen */
     double longvol;      /* longitud del volumen */
     double diamhi;       /* diámetro hidráulico */
     int numshmid;        /* identificador de segmento  */
} ECCMX;                  /* estructura para ecc mixer */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     long fromconn;       /* dirección de origen  */
     long toconn;         /* dirección de destino */
     double ajun;         /* área de la unión */
     char valtype[LVAR];  /* tipo de válvula */
     int numshmid;        /* identificador de segmento  */
} VALJUN;                 /* estructura para valve junction */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     double vfa;          /* área de flujo del volumen */
     double volvol;       /* volumen del volumen */
     int inlets;     /* código de volumen del volumen de conexión de entrada */
     double jasuc;        /* área de la unión de succión */
     int outlets;    /* código del volumen del volumen de conexión de salida */
     double jades;        /* área de la unión de descarga */
     int numshmid;        /* identificador de segmento  */
} PUM;                    /* estructura para pump */

typedef struct {
     int channel;         /* canal */
     long idcomp;         /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];  /* nombre del componente */
     char cmptype[LVAR];  /* tipo de componente */
     double valhid;       /* valor de componente hidrodinámico */
     int numjunts;        /* número de uniones */
     long fromconn;       /* dirección de origen */
     long toconn;         /* dirección de destino */
     double juna;         /* área de la unión */
     int numshmid;        /*  identificador de segmento  */
} MULJUN;                 /* estructura para multiple junction */

typedef struct {
     int channel;         /* canal */
```

```
     long idcomp;          /* identificador de componente (#tarjeta) */
     char cmpname[LVAR];   /* nombre del componente */
     char cmptype[LVAR];   /* tipo de componente */
     double valhid;        /* valor de componente hidrodinámico */
     double vfa;           /* área de flujo del volumen */
     double lonvol;        /* longitud del volumen */
     double volvol;        /* volumen del volumen */
     double hyddia;        /* diámetro hidráulico */
     long toconn;          /* dirección de destino del componente */
     double juna;          /* área de la unión */
     int numshmid;         /* identificador de segmento  */
} ACCUM;                   /* estructura para accumulator */

/* disparos */

typedef struct {
     int channel;          /* canal */
     long idcomp;          /* identificador de componente (#tarjeta) */
     char triptype[LVAR]; /* tipo de trip */
     char codvar[LVAR];    /* código de variable 1 */
     int param;            /* parámetro 1 */
     char relac[LVAR];     /* relación lógica/
     char codvar2[LVAR];   /* código de variable 2 */
     int param2;           /* parámetro 2 */
     double consad;        /* constante aditiva */
     char indcerr[LVAR];   /* indicador de cerrojo */
     int numshmid;         /* identificador de segmento  */
} VARTRIP;                 /* estructura para disparos de variable */

typedef struct {
     int channel;          /* canal */
     long idcomp;          /* identificador de componente (#tarjeta) */
     char triptype[LVAR]; /* tipo de trip */
     int numdisp;          /* número de disparo 1 */
     char oper[LVAR];      /* operador lógico */
     int numdisp2;         /* número de disparo 2 */
     char indcerr[LVAR];   /* indicador de cerrojo */
     int numshmid;         /* identificador de segmento  */
} LOGTRIP;                 /* estructura para disparos de lógica */
```

El bloque “SEPARA” cubre diversos objetivos: controla el acceso al archivo de disco “input deck”, realiza funciones de pre-procesador (lee el archivo línea por línea, elimina comentarios, obtiene la información de las tarjetas en forma individual e integra los grupos de tarjetas para formar un elemento), crea las llamadas para formación de segmentos de memoria compartida y crea los registros.

Debido a que la codificación de los números de tarjeta existente en el manual de RELAP presenta una gran similitud entre sí en sus formatos, pero la información entre éstas es diferente debe realizarse la validación o pertenencia a un conjunto predefinido en el manual, para no dar de alta segmentos de memoria de elementos inexistentes o realizar asignaciones de datos erróneas. Aún

cuando no es responsabilidad del módulo validar el archivo de entrada, se hace para garantizar que no se darán de alta segmentos de memoria con nombres de elementos inexistentes, y debido también al modo de programación de UNIX System V que es estricto en cuanto a la petición de zonas de memoria compartida. En cuanto a las variables del sistema de control la validación de las operaciones permitidas se hace para darle mayor confiabilidad a la información depositada en su segmento. Sirve también para crear los nombres de las variables de retención w (de word) en base al número asignado en la búsqueda binaria de un elemento hidrodinámico en el arreglo tipohidro[] junto con las características propias de la variable: su número de posición en la estructura, si es entera, real, alfanumérica y su posición en el arreglo tipohidro. Por ejemplo: w3r_14, representa la tercer palabra retenida del segmento de memoria compartida definido por el número 14. Las variables de retención son exclusivamente para los datos de las variables obtenidas de las tarjetas.

A continuación se muestra un fragmento de código de la función “separa” en la que se muestra la característica de preprocesador descrita anteriormente, junto a las variables de retención w y la declaración de algunas variables de utilidad:

```
/*************************************************************************/
int separa(char *nombre)
/*************************************************************************/
{
int rango=-1, indcom=-1, indant=15;
int indice=15, indicedos=15, t, clave;

int aster='*';
long valor, consec, anterior=0;
char *loc, *sa1, *sa2, *sa3, a1, a2, a3, b1,b2,b3;        /* digitos de inicio */
char *pa1, *pa2, *pa3, *pa4, *pa5, c1, c2, c3, c4,c5;/*digitos de terminaciones*/

double valin;  /*  valores iniciales de entrada a funciones de escritura*/

char *tipohidro[15]={"accum","annulus","branch","eccmix","jetmixer","mtpljun",
"pipe","pump","separatr","sngljun","snglvol","tmdpjun","tmdpvol","turbine","valve"};

char bufa[20], *ptr, *key0, **key1;

char
*tipocontrol[22]={"constant","delay","diffrend","diffreni","div","feedctl","function","in
tegral","lag","lead-lag",
"mult","poweri","powerr","powerx","prop-
int","pumpctl","shaft","stdfnctn","steamctl","sum","tripdlay","tripunit"};

char
buflinea[MAXLEN], bufcop[MAXLEN], varcode[EXTENLIN], valpar[EXTENLIN],
alph1[EXTENLIN], alph2[EXTENLIN], alph3[EXTENLIN], alph4[EXTENLIN],
alph5[EXTENLIN], alph6[EXTENLIN], alph7[EXTENLIN], alph8[EXTENLIN];
```

```
char w1a_15[17]="variable";
char w1a_16[17]="logical";

double valh_0=0.0, valh_1=0.0, valh_2=0.0, valh_3=0.0, valh_4=0.0, valh_5=0.0,
valh_6=0.0, valh_7=0.0, valh_8=0.0, valh_9=0.0, valh_10=0.0, valh_11=0.0, valh_12=0.0,
valh_13=0.0, valh_14=0.0;

char w1a_0[LVAR], w2a_0[LVAR];
double w1r_0=0.0, w2r_0=0.0, w3r_0=0.0, w8r_0=0.0, w2r_bis0=0.0;
long w0i_0=0, w1i_0=0;

char w1a_1[LVAR], w2a_1[LVAR];
int w1i_1=0;
long w0i_1=0;

char w1a_2[LVAR], w2a_2[LVAR];
int w1i_2=0;
long  w0i_2=0, w1i_bis2=0, w2i_2=0;
double w1r_2=0.0, w2r_2=0.0, w8r_2=0.0, w3r_2=0.0;

char w1a_3[LVAR], w2a_3[LVAR];
int w1i_3=0;
long w0i_3=0;
double w1r_3=0.0, w2r_3=0.0, w8r_3=0.0;

char w1a_4[LVAR], w2a_4[LVAR];
int w1i_4=0;
long w0i_4=0, w1i_bis4=0, w2i_4=0;
double w1r_4=0.0, w2r_4=0.0, w8r_4=0.0, w3r_4=0;

char w1a_5[LVAR], w2a_5[LVAR];
int w1i_5=0;
long  w0i_5=0, w1i_bis5=0, w2i_5=0;
double  w3r_5=0.0;

char w1a_6[LVAR], w2a_6[LVAR];
int w1i_6=0;
long w0i_6=0;

char w1a_7[LVAR], w2a_7[LVAR];
int w1i_7=0, w1i_bis7=0;
long w0i_7=0;
double w1r_7=0.0, w3r_7=0.0, w2r_7=0.0, w2r_bis7=0.0;

char w1a_8[LVAR], w2a_8[LVAR];
int w1i_8=0;
long w0i_8=0, w1i_bis8=0, w2i_8=0;
double w1r_8=0.0, w3r_8=0, w2r_8=0.0, w8r_8=0.0;

char w1a_9[LVAR], w2a_9[LVAR];
long  w0i_9=0, w1i_9=0, w2i_9=0;
double w3r_9=0.0;

char w1a_10[LVAR], w2a_10[LVAR];
long w0i_10=0;
double w1r_10=0.0, w2r_10=0.0, w3r_10=0.0;

char w1a_11[LVAR], w2a_11[LVAR];
long w0i_11=0, w1i_11=0, w2i_11=0;
double w3r_11=0.0;
```

```
char w1a_12[LVAR], w2a_12[LVAR];
long  w0i_12=0;
double w1r_12=0.0, w2r_12=0.0, w3r_12=0.0;

char w1a_13[LVAR], w2a_13[LVAR];
int w1i_13=0;
long w0i_13=0;
double w1r_13=0.0, w2r_13=0.0, w2r_bis13=0.0, w3r_13=0.0, w4r_13=0.0;

char w1a_14[LVAR], w2a_14[LVAR], w1a_bis14[LVAR];
long  w0i_14=0, w1i_14=0, w2i_14=0;
double w3r_14=0.0;

char w2a_15[LVAR], w3a_15[LVAR], w4a_15[LVAR], w5a_15[LVAR];
int w1i_15=0, w2i_15=0;
double w1r_15=0.0;

char w2a_16[LVAR], w3a_16[LVAR];
int w1i_16=0, w2i_16=0;

FILE *cfptr;

     w1a_0[0]='\0'; w2a_0[0]='\0';
     w1a_1[0]='\0'; w2a_1[0]='\0';
     w1a_2[0]='\0'; w2a_2[0]='\0';
     w1a_3[0]='\0'; w2a_3[0]='\0';
     w1a_4[0]='\0'; w2a_4[0]='\0';
     w1a_5[0]='\0'; w2a_5[0]='\0';
     w1a_6[0]='\0'; w2a_6[0]='\0';
     w1a_7[0]='\0'; w2a_7[0]='\0';
     w1a_8[0]='\0'; w2a_8[0]='\0';
     w1a_9[0]='\0'; w2a_9[0]='\0';
     w1a_10[0]='\0'; w2a_10[0]='\0';
     w1a_11[0]='\0'; w2a_11[0]='\0';
     w1a_12[0]='\0'; w2a_12[0]='\0';
     w1a_13[0]='\0'; w2a_13[0]='\0';
     w1a_14[0]='\0'; w2a_14[0]='\0'; w1a_bis14[0]='\0';
     w2a_15[0]='\0'; w3a_15[0]='\0'; w4a_15[0]='\0'; w5a_15[0]='\0';
     w2a_16[0]='\0'; w3a_16[0]='\0';

     if ((cfptr=fopen(nombre,"r")) == NULL) {
          printf("Problema. No puedo abrir archivo %s.\n", nombre);
          exit(0);
          }
     else
          do{

               buflinea[0]='\0'; bufcop[0]='\0'; varcode[0]='\0'; valpar[0]='\0';
               alph1[0]='\0'; alph2[0]='\0'; alph3[0]='\0'; alph4[0]='\0';
               alph5[0]='\0'; alph6[0]='\0'; alph7[0]='\0'; alph8[0]='\0';

               t=fgetc(cfptr);
               if(isdigit(t)){
                    ungetc(t,cfptr);
                    fgets(buflinea,MAXLEN-1,cfptr);
                    loc=strchr(buflinea,aster);
                    if(loc==NULL)
                         strcpy(bufcop,buflinea);
                    else
                    {
                         strncpy(bufcop,buflinea,loc-buflinea-1);
```

```
                    /*elimina comentario despues del asterisco*/
                    bufcop[loc-buflinea-1]='\0';
               }

               sscanf(bufcop,"%d%s%s%s%s%s%s%s%s",
               &valor,alph1,alph2,alph3,alph4,alph5,alph6,alph7,alph8);
               sa1=&bufcop[0];
               a1=*sa1;
               sa2=&bufcop[1];
               a2=*sa2;
               sa3=&bufcop[2];
               a3=*sa3;
               pa1=&bufcop[3];
               c1=*pa1;
               pa2=&bufcop[4];
               c2=*pa2;
               pa3=&bufcop[5];
               c3=*pa3;
               pa4=&bufcop[6];
               c4=*pa4;

               sscanf(buflinea,"%d",&valor);
               rango=numclave(valor);
```

En este punto del programa se llama a la función de selección “numclave( )” que discrimina los rangos de numeración correspondientes a componentes y elementos de control, según se describe en el manual del código RELAP. Se realiza la detección de todos los rangos para poder incluir otro tipo de componentes y para poder identificar las tarjetas que no pertenecen a esta codificación.

```
/*************************************************************************/
int numclave(long numero)
/*************************************************************************/
               /* seleccion  generica elemental por intervalos*/
{
     if(( numero >=100) && (numero<= 199)) return 0;
     /*  control options*/
     else if(( numero >=200) &&(numero<= 299)) return 1;
     /*  time step options*/
     else if(( numero >=301) && (numero<= 399)) return 2;
     /* minor edit */
     else if((( numero >=401) && (numero<=599)) ||
     (( numero >=20600010) && (numero<=20610000 ))) return 3;
     /*  variable trips */
     else if((( numero >=601) && (numero<=799)) ||
     (( numero >=20610010) && (numero<=20620000)) ) return 4;
     /*  logical trips*/
     else if(( numero >=1000000) && (numero<= 9999999)) return 5;
     /*  hidrodinamics*/
     else if(( numero >=10000000) && (numero<= 19999999)) return 6;
     /*  heat structures*/
     else if(( numero >=20100000) && (numero<= 20199999)) return 7;
     /*  heat structures thermal properties*/
     else if(( numero >=20200000) && (numero<= 20299999)) return 8;
     /*  general tables data*/
     else if(( numero >=30000000) && (numero<= 39999999)) return 9;
```

```
     /*  kinetics*/
     else if(( numero >=20500000) && (numero<= 20599999)) return 10;
     /*  control systems*/
     else if(( numero >=20800000) && (numero<= 20809999)) return 11;
     /*  optional restart record request*/
     else if(( numero >=20300000) && (numero<= 20499999)) return 12;
     /*  plot request*/
     else if(( numero >=60000000) && (numero<= 69999999)) return 13;
     /*  radiation*/
     else if(( numero >=1001) && (numero <= 1999)) return 14;
     /* strip request data  o:  compare dump files control data*/
     else{
          printf("\n       Numero de tarjeta %ld fuera de rango.\n", numero);
          return 0;
      }

} /* FIN numclave*/
```

En base a la obtención del rango obtenido con la función anterior se ingresa a las categorías y en caso de pertenecer a los componentes hidrodinámicos o al sistema de control se validará la información de ingreso:

```
          switch(rango)
          {


               case 5:  /* hidrodinamics */
                    {
                    strcpy(bufa,alph2);
                    key0=bufa;
                    key1=&key0;
                    ptr= (char *)bsearch(key1,
                        tipohidro,15,sizeof(tipohidro[0]), comp);
                    if(ptr != NULL){     /*¿es componente?*/

                         for (indice=0; indice<15; indice++)
                              if (!strcmp(bufa,tipohidro[indice]))
                                                  ;
                    }/*if_ptr*/
                    else{
                         printf("\n%s No se encuentra ", bufa);
                    }
```

Esencialmente, la creación de los segmentos de memoria compartida se encuentra en el siguiente segmento de código, en el que se solicita en el primer caso la creación del segmento para el tipo de componente identificado. Utiliza números arbitrarios como llaves únicas para evitar crear otro segmento con la llamada a “shmget” mientras no se libere la llave actual. Existe la opción de tomar como llave el valor IPC_PRIVATE, que crea llaves en forma automática, pero impide saber si el segmento solicitado realmente fue creado. La creación del segmento puede confirmarse mediante

comandos en teclado, empleando la instrucción UNIX ipcs, mediante el cual se reporta en pantalla qué mecanismos están asignados y a quién. En nuestro código la función "shmget" se encuentra integrada en la función "create_shm" que se explica más adelante.

Una vez que la llamada ocurre exitosamente obtenemos un espacio de memoria que ya está unida al espacio de direcciones virtuales del proceso y accederemos a ella a través de apuntadores. A continuación, a manera de ejemplo, se muestra la creación del segmento de memoria compartida para los componentes tipo ACCUM, mediante la llamada a la función create_shm.

```
switch(indice){      /*  crea_SHM*/
     case 0:/*accum*/
          {
          if(bam)
               {
               bam=0;
               clave=110;
               SHM_SIZEAM=sizeof(ACCUM);
               if(SHM_C(regptram=(ACCUM *) create_shm(&regidam, clave,
                         SHM_SIZEAM, &bufregam))<0)
                    {
                    exit(-1);
                    printf("Error al crear shm_accum\n");
                    printf("Identificador de segmento:  %d\n", regptram);
                    }
               }
          indcom=indice;
          break;
          }
```

La creación de localidades de memoria (registros) se realiza con la llamada a las funciones del bloque LECTURA. El respaldo al archivo en disco para el reporte y la reinicialización de las variables de retención w, ocurre como se muestra en el siguiente fragmento (ejemplo para TMDPVOL):

```
     case 12:  /* tmdpvol */
     {
          if ((a1!=b1)&&(a2!=b2)&&(a3!=b3))
          {
          shm_wtmdpv( numerocanal, valor,
          w1a_12, w2a_12, valh_12,
          w1r_12, w2r_12, w3r_12,
          regidtv);

          fprintf(salptr,"\nNumero de canal: %d", numerocanal);
          fprintf(salptr,"\nIdentificador de  componente (#tarjeta): %d",valor);
          fprintf(salptr,"\nNombre del componente: %s",w1a_12);
          fprintf(salptr,"\nTipo de componente: %s",w2a_12);
          fprintf(salptr,"\nValor de componente hidrodinámico: %f",valh_12);
```

```
            fprintf(salptr,"\nArea de flujo del volumen: %f",w1r_12);
            fprintf(salptr,"\nLongitud del volumen: %f",w2r_12);
            fprintf(salptr,"\nVolumen del volumen: %f",w3r_12);
            fprintf(salptr,"\nIdentificador de segmento: %d", regidtv);
            fprintf(salptr,"\n\n");

            numerocanal++;
            regptrtv++;
            conttv++;

            w1a_12[0]='\0';
            w2a_12[0]='\0';

            w1r_12=0.0;
            w2r_12=0.0;
            w3r_12=0.0;
            }
      break;
      }
```

La retención de información del conjunto de tarjetas que corresponden a un componente, es necesaria, ya que es forzoso distinguir a qué componente corresponde antes de escribir la información en memoria compartida. Se muestran algunos ejemplos consecutivos.

```
switch(indcom)  /* captura datos_SHM */
{
      case 0:  /* accum */
      {
            if((c1=='0')&&(c2=='0')&&(c3=='0')&&(c4=='0'))
            {
                  strcpy(w1a_0,alph1);
                  strcpy(w2a_0,alph2);
            }
            if((c1=='0')&&(c2=='1')&&(c3=='0')&&((c4>='1')&&(c4<='9'))&&(consec!=1))
            {
                  w1r_0=atof(alph1);
                  w2r_0=atof(alph2);
                  w3r_0=atof(alph3);
            }
            if((c1=='0')&&(c2=='1')&&(c3=='0')&&((c4>='1')&&(c4<='9'))&&(consec==1))
            {
                  w8r_0=atof(alph4);
            }
            if((c1=='1')&&(c2=='1')&&(c3=='0')&&(c4=='1'))
            {
                  w1i_0=atol(alph1);
                  w2r_bis0=atof(alph2);
            }
            break;
      }

      case 1:  /* annulus */
      {
            if((c1=='0')&&(c2=='0')&&(c3=='0')&&(c4=='0'))
            {
                  strcpy(w1a_1,alph1);
```

```
            strcpy(w2a_1,alph2);
        }
        if((c1=='0')&&(c2=='0')&&(c3=='0')&&(c4=='1'))
        {
            w1i_1=atol(alph1);

        break;
    }
```

La implementación del bloque de lectura está formada por 16 funciones de lectura para acceder a los 18 segmentos potenciales de memoria compartida que podrían existir. Éstas se muestran en la Tabla 3.1.

Tabla 3.2 Funciones que integran el bloque LECTURA.

shm_rcs( )	shm_rsnglv( )	shm_rtmdpv( )	shm_rsnglj( )
shm_rtmdpj( )	shm_rpip( )	shm_rannl( )	shm_rbran( )
shm_rsepr( )	shm_rjetmx( )	shm_rturb( )	shm_reccmx( )
shm_rvaljun( )	shm_rpum( )	shm_rmuljun( )	shm_raccum( )
shm_rlogtrip()	shm_rvartrip()		

A continuación se muestra como ejemplo el caso de la función de acceso a la sección de memoria compartida PIPE:

```
/*************************************************************************/
void shm_rpip(void)
/*************************************************************************/
{

      printf("\nCanal: %d",regptrpp->channel);
      printf("\nIdentificador de componente (#tarjeta): %d",regptrpp->idcomp);
      printf("\nNombre del componente: %s",regptrpp->cmpname);
      printf("\nTipo de componente: %s",regptrpp->cmptype);
      printf("\nValor de componente hidrodinámico: %f",regptrpp->valhid);
      printf("\nNumero de volumenes: %d",regptrpp->numvols);
      printf("\nIdentificador de segmento: %d",regptrpp->numshmid);

}
```

La implementación del bloque de escritura también está integrada por 16 funciones, mostradas en la Tabla 3.2.

Tabla 3.3 Funciones que forman el bloque ESCRITURA.

shm_wcs( )	shm_wsnglv( )	shm_wtmdpv( )	shm_wsnglj( )
shm_wtmdpj( )	shm_wpip( )	shm_wannl( )	shm_wbran( )
shm_wsepr( )	shm_wjetmx( )	shm_wturb( )	shm_weccmx( )
shm_wvaljun( )	shm_wpum( )	shm_wmuljun( )	shm_waccum( )
shm_wlogtrip()	shm_wvartrip()		

Se muestra shm_wpip( ) como ejemplo del contenido de las otras funciones shm_w( ).

```
/*********************************************************************/
void shm_wpip( int canal, long identcomp,
char *nombcomp, char *tipocomp, double valorh,
int nvolums, int identif)
/*********************************************************************/
{
      regptrpp->channel=canal;
      regptrpp->idcomp=identcomp;
      strcpy(regptrpp->cmpname,nombcomp);
      strcpy(regptrpp->cmptype,tipocomp);
      regptrpp->valhid=valorh;
      regptrpp->numvols=nvolums;
      regptrpp->numshmid=identif;

}
```

La petición de un segmento de memoria compartida al núcleo del sistema operativo se realiza llamando a la función crea_shm( ), la cual integra las llamadas “get” que controlan la utilización de los mecanismos IPC por las cuales se obtienen: el segmento en RAM físicamente, las funciones de control propias del segmento, su adosamiento a las estructuras y su identificador de segmento, asignado por el núcleo del sistema operativo.

```
/*********************************************************************/
char *create_shm(int *shmid, key_t key, int size, struct shmid_ds *buf)
/*********************************************************************/

{
   char *shm_seg;

   if ((*shmid = shmget(key, size, IPC_CREAT | 0666)) < 0) {
      perror("create_shm(): shmget()");
      return ((char *) -1);
   }
   if (shmctl(*shmid, IPC_STAT, buf) < 0) {
      perror("create_shm(): shmctl()");
```

```
      return ((char *) -1);
   }

  if ((shm_seg = shmat(*shmid, (char *) NULL, IPC_CREAT | 0666)) == (char *) -1) {
      perror("create_shm(): shmat()");
      return ((char *) -1);
   }
   return (shm_seg);
}
```

La explicación del bloque REPORTE se da en el siguiente capítulo, con el fin de evitar redundancias y por que es el procedimiento para obtener resultados de la ejecución del programa *modulo*.

El código completo del programa *modulo.c* se encuentra disponible en CD a solicitud del interesado.

CAPITULO 4

EVALUACIÓN Y DISCUSIÓN DE RESULTADOS

Este capítulo describe el proceso de evaluación, la descripción de las pruebas y los resultados asociados a estas pruebas. También se señalan las limitaciones y potenciales del proyecto así como los problemas y dificultades asociadas al empleo de memoria compartida de UNIX System V.

4.1 Evaluación del desarrollo.

Para evaluar la implementación se realizaron corridas empleando dos archivos de entrada; Turbina y Browns Ferry, que describen algunos modelos del simulador. En cuanto a hardware se utilizó una estación de trabajo ALPHA con sistema operativo UNIX y los recursos software consistieron en el compilador cc nativo.

4.2 Limitaciones del programa.

Para realizar las corridas con el archivo Browns Ferry se modificaron algunas de sus variables para evitar lecturas equivocadas de 'palabras' (agrupamiento de caracteres). Por ejemplo, la variable "old time" está compuesta por dos palabras: ["old] y [time"]. La variable "bp w dem" está compuesta por tres agrupamientos ["bp] [w] [dem"]. Mientras que en el mismo código existen variables con palabras unidas por un guión bajo como en "dem_vlv". Se cambió el espacio en blanco por guión bajo "_" para obtener un solo agrupamiento de caracteres ya que "bp_w_dem" no implica ninguna operación o notación para el programador, como otros separadores que sí la implican y que aparecen en las siguientes variables: "tbl/fctn", "auto# sv", "dm-citor", "blsp+tse".

Otra limitante es la detección de campos de variables cuando se utilizan múltiples tarjetas y el campo de interés está muy separado de la primera tarjeta que describe un componente, ya que RELAP permite el uso de una única tarjeta o múltiples tarjetas para un mismo número de campos.

Por ejemplo, el código turbina ingresa los datos de la tarjeta 101 del componente snglvol stmline1 en forma contínua:

1550000 stmline1 snglvol
1560101 8.2 100. 0. 0. 0. 0. 0. 0. 11010

Mientras que en el archivo Browns Ferry para ingresar los datos de la tarjeta 101 del componente snglvol "jpdischr" se emplean dos tarjetas:

2900000 "jpdischr" snglvol
2900101 108,634 3.337 0.0 0.0 –90 –3.337
2900102 0.00015 1.93 00

Pero el dato de interés puede estar en la tercera, cuarta o hasta la novena tarjeta. Por simplicidad el dato de interés sólo se busca en las dos primeras tarjetas. Otra limitación se encuentra en la capacidad de registros, preestablecido en 1024 localidades por segmento, aunque se ajusta según se requiera, cambiando el valor en la línea de código #define MAXCONTROL.

4.3 Descripción de los programas desarrollados para la evaluación de la implementación.

El nombre del programa ejecutable del módulo de administración de datos es *modulo*.exe. Esta aplicación es la que procesa los archivos de entrada a RELAP. Se seleccionaron dos archivos para prueba de la funcionalidad del programa desarrollado, los mismos que se utilizaron para las pruebas de desarrollo de las funciones internas del programa. El primer archivo es el modelo de Turbina. Este archivo contiene un modelo simplificado de la turbina de la CNLV. Es un archivo corto y de baja complejidad para su procesamiento. El segundo archivo de prueba está relacionado con el modelo del reactor Browns Ferry, y de una alta complejidad para su procesamiento. Al hacer la corrida, por separado, del modelo Turbina y del modelo Browns Ferry se obtuvieron los siguientes resultados, que se abrevian por simplicidad.

La ejecución del programa requiere como único argumento el nombre del archivo de entrada a procesar. En el caso del modelo de turbina se deberá teclear:

```
./modulo turbina
```

Al ejecutarse el programa inicia la detección de los componentes hidrodinámicos y variables de interés para la activación de los segmentos de memoria compartida correspondientes así como la captura de información asociada a estos componentes o variables. Esta información, obtenida del archivo de entrada, se deposita en las estructuras que fueron creadas en memoria compartida. La detección y activación de componentes y variables se realiza sobre todo el archivo de entrada, aun cuando salgamos de inmediato del programa. Si ejecutamos el programa como se mostró anteriormente e inspeccionamos las estructuras IPC desde el teclado veremos los segmentos de memoria compartida activados para los componentes y variables de interés descritos en el input deck leído por el programa. Y si leemos el archivo *rep_sal* veremos la información contenida en los segmentos activos de memoria compartida.

Los menús que aparecen nos muestran las acciones permitidas:

```
          MENU

---------------------
1    Reporte
2    Salir

          Opcion -->
```

La opción 2 simplemente termina la ejecución de la aplicación. Si se selecciona la opción 1, se despliega en pantalla el menú de Reporte, que se muestra a continuación.

```
         REPORTE

-----------------------
0    Mostrar estado IPC
1    Por segmentos
2    Por canal
3    Salir

          Opcion --> 0
```

La opción 0 nos muestra la información completa que entrega el sistema de los mecanismos activos IPC. La llave bajo la que fueron creados (en hexadecimal), su identificador de segmento (identificador interno de proceso), el nombre del propietario, los permisos de acceso, el tamaño en bytes de cada segmento así como los procesos adjuntos al segmento. La salida que se muestra a continuación se obtuvo de correr el programa en la estación de trabajo ALPHA.

```
---- Segmentos memoria compartida ----
key        shmid     propietarioperms     bytes     nattch    estado
0x00000064 3073       dioni     666       45056     2
0x000000e6 3074       dioni     666       68        2
0x000000fa 3075       dioni     666       72        2
0x000000d2 3076       dioni     666       69632     2
0x000000f0 3077       dioni     666       90112     2
0x00000082 3078       dioni     666       90112     2

------ Matrices semáforo -------
key        semid     propietarioperms     nsems     estado

------ Colas de mensajes -----
key        msqid     propietarioperms     bytes utilizadosmensajes


          Salir:   0   enter   ->
```

La opción 1 nos permite conocer qué segmentos de memoria compartida fueron dados de alta, por lo que se les muestra como “Segmentos activos”. Como se mencionó anteriormente, cada segmento de memoria compartida incluye todos los componentes encontrados en el archivo de entrada asociados a dicho segmento.

```
            REPORTE

      ----------------------
      0    Mostrar estado IPC
      1    Por segmentos
      2    Por canal
      3    Salir

           Opcion --> 1
```

Al seleccionar la opción 1 se muestra en pantalla el siguiente diálogo:

```
Por segmentos.

Segmentos activos:

        control_system(1)       branch(4)

        snglvol(12)
        tmdpvol(14)     turbine(15)     valve(16)

Ver segmento SHM numero:
```

En este ejemplo podemos notar que existen 6 segmentos activos, cada uno con un número entre paréntesis que corresponde al número de clave solicitado en la parte baja de la pantalla. A continuación se muestra el despliegue individual de cada elemento encontrado en el segmento de memoria compartida. En la parte superior encontramos el número de dirección en RAM del elemento, el número de registro en que se encuentra y el total de registros almacenados en el segmento reportado. En la parte del registro se muestra la información almacenada en cada campo de la estructura asociada al componente. Podemos desplazarnos en forma secuencial hacia adelante y hacia atrás de los registros del segmento usando las teclas ‘a’ y ‘s’, con lo que es posible recorrer completamente cada segmento activo. Salimos de la lectura del segmento al avanzar sobre el último registro. Para el sistema de control obtenemos la siguiente salida:

```
Dir_SHM: 1075658752   Reg. Actual: 0   Reg. Ultimo: 27

Numero de canal: 1
Identificador de variable de control (#tarjeta): 20589900
Nombre de variable de control: flecha
Valor de la variable de control: 1800.000000
Tipo de variable de control: shaft
Identificador de segmento: 3073

          (a)delante   atra(s)      Enter


Dir_SHM: 1075658796   Reg. Actual: 1   Reg. Ultimo: 27

Numero de canal: 17
Identificador de variable de control (#tarjeta): 20500100
Nombre de variable de control: pfv
Valor de la variable de control: 0.000000
Tipo de variable de control: sum
Identificador de segmento: 3073
```

```
          (a)delante   atra(s)      Enter

..............

Dir_SHM: 1075659940   Reg. Actual: 27   Reg. Ultimo: 27

Numero de canal: 43
Identificador de variable de control (#tarjeta): 20595100
Nombre de variable de control: valvcf
Valor de la variable de control: 1.000000
Tipo de variable de control: function
Identificador de segmento: 3073

          (a)delante   atra(s)      Enter
```

Otro ejemplo de la salida del reporte por segmento, son los componentes tipo turbina. Se muestra el primer y último elemento:

```
Dir_SHM: 1075703808   Reg. Actual: 0   Reg. Ultimo: 8

Numero de canal: 7
Identificador de componente (#tarjeta): 9000000
Nombre del componente: turbF
Tipo de componente: turbine
Valor de componente hidrodinamico: 0.000000
Area de flujo del volumen: 8.200000
Longitud del volumen: 100.000000
Tipo de turbina: 2
Eficiencia: 0.000000
Fraccion de reaccion de diseño r: 0.500000
Radio medio de etapa: 1.000000
Identificador de segmento: 3077

          (a)delante   atra(s)      Enter

..............


Dir_SHM: 1075884736   Reg. Actual: 8   Reg. Ultimo: 8

Numero de canal: 15
Identificador de componente (#tarjeta): 9080000
Nombre del componente: turb8
Tipo de componente: turbine
Valor de componente hidrodinamico: 0.000000
Area de flujo del volumen: 200.000000
Longitud del volumen: 100.000000
Tipo de turbina: 2
Eficiencia: 0.900000
```

```
Fraccion de reaccion de diseño r: 0.500000
Radio medio de etapa: 1.000000
Identificador de segmento: 3077

          (a)delante   atra(s)      Enter
```

La opción 2 nos permite obtener un archivo en disco llamado *rep_sal*, o reporte de salida, el cual contiene los datos que se obtuvieron de la memoria compartida en todos sus segmentos activos en el tiempo en que fue llamada la función REPORTE. El archivo *rep_sal* muestra todos los canales en forma secuencial y la información de cada elemento de dicho canal. Este archivo es generado y actualizado en cada llamada a la opción REPORTE. También podemos observar la información por pantalla empleando el editor vi, con el que cuentan todos los sistema UNIX y al que invocamos en forma automática en modo de sólo lectura. Por ello, para salir del editor vi usamos el comando shift :q

```
                REPORTE

          ------------------------
          0    Mostrar estado IPC
          1    Por segmentos
          2    Por canal
          3    Salir

               Opcion --> 2
```

```
Numero de canal: 1
Identificador de  variable de control (#tarjeta): 20589900
Nombre de variable de control: flecha
Valor de la variable de control: 1800.000000
Tipo de variable de control: shaft
Identificador de segmento: 3073

Numero de canal: 2
Identificador de componente (#tarjeta): 1500000
Nombre del componente: linvapor
Tipo de componente: tmdpvol
Valor de componente hidrodinamico: 0.000000
Area de flujo del volumen: 8.200000
Longitud del volumen: 100.000000
Volumen del volumen: 820.000000
Identificador de segmento: 3074

............

Numero de canal: 42
```

```
Identificador de  variable de control (#tarjeta): 20595000
Nombre de variable de control: posvc
Valor de la variable de control: 0.000000
Tipo de variable de control: constant
Identificador de segmento: 3073

Numero de canal: 43
Identificador de  variable de control (#tarjeta): 20595100
Nombre de variable de control: valvcf
Valor de la variable de control: 1.000000
Tipo de variable de control: function
Identificador de segmento: 3073
```

Para probar el acceso a los segmentos creados mediante el programa principal se crearon dos programas que permiten respectivamente la escritura y lectura de cualquier canal deseado en forma externa al programa *modulo*. El primero llamado *escribir.c*, genera números aleatorios con formato C (double) y los escribe ininterrumpidamente en el canal que se le indica mediante el menú, en el campo correspondiente a "valor de la variable de control". El segundo programa, llamado *leer.c*, realiza la lectura ininterrumpida de la información del canal seleccionado. A manera de ejemplo se muestra el fragmento para acceso del segmento de memoria compartida del sistema de control mediante ambos programas.

```
/*************************************************************
                    escribir.c
*************************************************************/
#include "stdlib.h"
#include "stdio.h"
#include "ctype.h"
#include "string.h"
#include "sys/ipc.h"
#include "sys/shm.h"
#include "sys/types.h"
#define   LVAR   12

typedef struct {
     int channel;    /* canal */
     long idvc;      /* identificador de variable de control (#tarjeta) */
     char nomvc[LVAR];/* nombre de variable de control  */
     double valvc;   /*  valor de la variable de control  */
     char tipvc[LVAR];/* tipo de variable de control  */
     int numshmid;   /*  identificador de segmento  */
} CONSYS;            /* estructura para control system */

CONSYS *regptr;
FILE *ref;

int main()
```

```
{
     int espera, cont, iregcs=0;
     int regid, numcanal, canalref, segmentref, regiref, ipcref;

     system("clear");
     if((ref=fopen("refid.lst","r")) == NULL)
          printf("\nProblema. No puedo abrir %s", "refid.lst");
     printf("\n< Escribir >");
     printf("\n\ncanal: ");
     scanf("%d",&numcanal);
      do{
          fscanf(ref,"%d %d %d %d",&canalref,&segmentref,&regiref,&ipcref);
          if(numcanal==canalref)
          {
               iregcs=regiref;
               regid=ipcref;
          }
     }while(!feof(ref));
     fclose(ref);

     regptr=shmat(regid,0,0);
     for(cont=0; cont<iregcs; cont++)
          regptr++;

     while(1)
     {
          printf("\n\nCanal: %d   Variable de control: %s",regptr->channel, regptr-
>nomvc);
          regptr->valvc= (double) rand();
          printf("\nescribo: %f", regptr->valvc);
          for(espera=0;espera<100000000;espera++)
               ;
          system("clear");
     }
}



/*************************************************************************
                    leer.c
*************************************************************************/

#include "stdlib.h"
#include "stdio.h"
#include "ctype.h"
#include "string.h"
#include "sys/ipc.h"
#include "sys/shm.h"
#include "sys/types.h"
#define   LVAR   12

typedef struct {
     int channel;    /* canal */
     long idvc;      /* identificador de variable de control (#tarjeta) */
     char nomvc[LVAR];/* nombre de variable de control  */
     double valvc;   /*  valor de la variable de control  */
     char tipvc[LVAR];/* tipo de variable de control  */
     int numshmid;   /*  identificador de segmento  */
} CONSYS;            /* estructura para control system */

CONSYS *regptr;
```

```
FILE *ref;

int main()
{
      int espera, cont, iregcs=0;
      int regid, numcanal, canalref, segmentref, regiref, ipcref;

      system("clear");
      if((ref=fopen("refid.lst","r")) == NULL)
            printf("\nProblema. No puedo abrir %s", "refid.lst");
      printf("\n< Leer >");
      printf("\n\ncanal: ");
      scanf("%d",&numcanal);
       do{
            fscanf(ref,"%d %d %d %d", &canalref,&segmentref,&regiref,&ipcref);
            if(numcanal==canalref)
            {
                  iregcs=regiref;
                  regid=ipcref;
            }
      }while(!feof(ref));
      fclose(ref);

      regptr=shmat(regid,0,0);
      for(cont=0; cont<iregcs; cont++)
            regptr++;

      while(1)
      {
            printf("\n\nCanal: %d  Variable de control: %s",regptr->channel, regptr-
            >nomvc);
            printf("\nLeo:   %f",regptr->valvc);
            for(espera=0;espera<100000000;espera++)
                  ;
            system("clear");
      }
}
```

La ejecución del programa *escribir.c* adosa el registro y el segmento correspondiente al número de canal que se le indica, como se muestra a continuación:

```
./escribir
```

Aparece en pantalla la indicación que entramos en < Escribir > y la solicitud de número de canal:

```
< Escribir >

canal:
```

La ejecución de *leer* nos muestra la información que se encuentra en el campo del registro del número de canal adosado y el nombre de la variable a la que corresponde:

```
./leer                                          (ejecución)

< Leer >

canal: 1                                        (solicitud de canal para leer)


Leo:      1987231011.000000                     (lectura del segmento SHM)

Canal: 1   Variable de control: flecha
```

De modo similar se corrió el archivo de entrada Browns Ferry, obteniéndose los siguientes resultados:

```
   ---- Segmentos memoria compartida ----
key       shmid     propietarioperms     bytes     nattch    estado
0x00000064 9217      dioni     666       45056     2
0x000000e6 9218      dioni     666       68        2
0x000000dc 9219      dioni     666       61440     2
0x000000fa 9220      dioni     666       72        2
0x000000aa 9221      dioni     666       49152     2
0x000000c8 9222      dioni     666       61440     2
0x00000082 9223      dioni     666       90112     2
0x000000b4 9224      dioni     666       86016     2
0x00000096 9225      dioni     666       90112     2
0x000000d2 9226      dioni     666       69632     2
0x000000be 9227      dioni     666       90112     2
0x00000078 9228      dioni     666       49152     2

------ Matrices semáforo -------
key       semid     propietarioperms     nsems     estado

------ Colas de mensajes -----
key       msqid     propietarioperms     bytes utilizadosmensajes
```

Con el listado anterior comprobamos que el número de segmentos activos corresponden al número de segmentos esperados y que el tamaño en bytes de los segmentos, proporcionalmente corresponde a la cantidad de elementos que los integran. Es importante resaltar que el usuario no se debe preocupar por hacer las conversiones de número de llave en hexadecimal, para adosar el

segmento, o por indagar el número de identificador IPC ya que esto se hizo transparente al usuario en los programas *escribir* y *leer*. Al ejecutarse el programa *modulo* se crea el archivo refid.lst en el que se guarda la información de cada canal para el correcto acceso de los programas *escribir* y *leer*. El método se puede generalizar para las otras estructuras.

A continuación se muestra la corrida del archivo Browns Ferry en forma simplificada:

```
                REPORTE

            ----------------------
            0    Mostrar estado IPC
            1    Por segmentos
            2    Por canal
            3    Salir

                Opcion --> 1


Por segmentos.

Segmentos activos:

        control_system(1)       annulus(3)      branch(4)
        jetmixer(6)     pipe(8)
        pump(9)         separatr(10)    sngljun(11)     snglvol(12)
        tmdpjun(13)     tmdpvol(14)     valve(16)


Ver segmento SHM numero: 1

      Dir_SHM: 1076453376   Reg. Actual: 0   Reg. Ultimo: 257


Numero de canal: 1
Identificador de variable de control (#tarjeta): 20500100
Nombre de variable de control: "del_time"
Valor de la variable de control: 0.020000
Tipo de variable de control: sum
Identificador de segmento: 9217

          (a)delante   atra(s)      Enter

       Dir_SHM: 1076453420   Reg. Actual: 1   Reg. Ultimo: 257


Numero de canal: 2
Identificador de variable de control (#tarjeta): 20500200
Nombre de variable de control: "old_time"
Valor de la variable de control: 900.000000
Tipo de variable de control: sum
```

```
Identificador de segmento: 9217

          (a)delante   atra(s)      Enter
.....................

Dir_SHM: 1076464684   Reg. Actual: 257   Reg. Ultimo: 257


Numero de canal: 348
Identificador de variable de control (#tarjeta): 20540100
Nombre de variable de control: "dem_vlv"
Valor de la variable de control: 1.000000
Tipo de variable de control: sum
Identificador de segmento: 9217

          (a)delante   atra(s)      Enter



                      REPORTE

               ------------------------
               0    Mostrar estado IPC
               1    Por segmentos
               2    Por canal
               3    Salir

                        Opcion --> 2


Numero de canal: 1
Identificador de  variable de control (#tarjeta): 20500100
Nombre de variable de control: "del_time"
Valor de la variable de control: 0.020000
Tipo de variable de control: sum
Identificador de segmento: 9217

Numero de canal: 2
Identificador de  variable de control (#tarjeta): 20500200
Nombre de variable de control: "old_time"
Valor de la variable de control: 900.000000
Tipo de variable de control: sum
Identificador de segmento: 9217

.............................

Numero de canal: 347
Identificador de  variable de control (#tarjeta): 20540000
Nombre de variable de control: "dta_apt"
Valor de la variable de control: 0.000000
Tipo de variable de control: sum
Identificador de segmento: 9217

Numero de canal: 348
Identificador de  variable de control (#tarjeta): 20540100
```

```
Nombre de variable de control: "dem_vlv"
Valor de la variable de control: 1.000000
Tipo de variable de control: sum
Identificador de segmento: 9217
~
```

La ejecución del programa *modulo* sobre los archivos Turbina y Browns Ferry reportaron la activación correcta de todos los segmentos que se esperaban, y que son congruentes con la clasificación propuesta en el diseño. Se cotejó que la información almacenada en cada registro fuera exacta siguiendo el modelo de tarjetas de RELAP, esto es, que la información capturada en memoria compartida correspondiera con la información del archivo de entrada codificada según el modelo de tarjetas de RELAP. En cuanto al acceso a los segmentos de memoria compartida se pudo verificar el correcto adosamiento de lectura y escritura a los segmentos creados para los archivos de entrada utilizados como prueba. Se verificaron los componentes y variables constituyentes de cada segmento activo. Así mismo se probó el buen funcionamiento del control del programa en la emisión del reporte en sus dos opciones; por segmento y por canal, y la creación de los archivos *rep_sal* y *refid.lst.*

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se mencionan las principales conclusiones del presente trabajo de tesis y se comenta algunas propuestas para trabajo a futuro relacionadas con el desarrollo del simulador de aula.

5.1 Conclusiones.

El Módulo de Administración de Datos de Memoria Compartida desarrollado, aumentará sensiblemente la funcionalidad del simulador de aula para la administración de la información, permitiendo la generación de segmentos de memoria compartida con un formato único preestablecido y de manera automática y organizada. Adicionalmente, la generación de un reporte de la memoria compartida gestionada, le brinda al usuario la información completa y necesaria para su correcta utilización posterior de los datos contenidos en memoria, lo que facilitará la integración, interconectividad y mantenimiento de los múltiples componentes y procesos con que cuenta el simulador.

El Módulo de Administración de Datos de Memoria Compartida constituye una herramienta para monitoreo y administración de variables de proceso y control así como de las variables internas del código en tiempo de ejecución o fuera de línea. También hace posible el almacenamiento de datos a manera de registro histórico permitiendo la documentación de cualquier cambio realizado en el sistema.

La implementación del Módulo de Memoria Compartida facilita una mayor comprensión o visualización de los componentes empleados por los modelos especificados en los archivos de entrada sin tener que editar, analizar e interpretar los mismos, evitando un laborioso procedimiento. Mediante el Módulo de Administración se identifican fácilmente los componentes hidrodinámicos, sus variables, las variables de control y los disparos de una forma organizada con posibilidades de

interactividad para una búsqueda de los elementos que serán de utilidad en un escenario o propósito especifico. Para lograr esta identificación el programa *modulo* permite dos modos de lectura de los segmentos activos: por segmento y por canal. En la identificación por segmentos se obtiene en forma secuencial el contenido de la memoria compartida y tipo de componente hidrodinámico, variable de control o disparos asociados al segmento. Se pueden recorrer los registros en modo ascendente o descendente para la identificación de la totalidad de los elementos que la integran. En la identificación por canal obtenemos un listado ordenado descendente por número de canal asignado a los componentes hidrodinámicos, variables de control o disparos. Ambos reportes son actualizados cada vez que se solicitan.

Los programas que hacen uso de segmentos de memoria compartida anteriores al desarrollo de esta aplicación no brindan ninguna facilidad sobre el contenido de las localidades de memoria compartida o de alguna información que ayude a reconocer qué tipo de componente y/o qué tipo de variable están activos, de modo que cada escenario ejecutado está ligada a un archivo de entrada específico y cualquier modificación del mismo requiere de un laborioso proceso de adecuación, limitando severamente la funcionalidad del simulador de aula. Tampoco presentan ninguna clasificación respecto al tipo de variables contenidas en sus segmentos o que permita obtener interactivamente información de interés. Debido a ello no se aprovecha debidamente la información que podría obtenerse de las localidades de memoria, de ahí la importancia de la presente aplicación como herramienta de gestión, de administración y de monitoreo de segmentos de memoria.

5.2 Trabajos futuros.

Se puede incrementar la funcionalidad del programa *modulo* incorporando otros modelos y componentes como son las estructuras de calor, el modelo de cinética, el modelo de radiación, o bien, incorporar las tarjetas que modelan los tipos de problemas que resuelve SCDAP, para accidentes severos.

Otro tipo de actividad a futuro se encuentra en el desarrollo de una interfaz gráfica para el programa *modulo* para mejorar el control y la visualización de los segmentos. Otro tipo de procesos se le podrían añadir como son: lectura, grabación y reproducción de datos de segmentos activos con

el fin de crear archivos de datos para su análisis posterior. Básicamente el programa *modulo* será utilizado por las diversas aplicaciones que integran el simulador de aula y que procesan información de los segmentos de memoria compartida. El programa *modulo* podría ser la plataforma de otros programas que generen salidas automáticas, como un paso previo antes de ejecutarse ellos mismos, tomando como base la lectura del reporte del Módulo de Administración de Datos de Memoria Compartida. Por ejemplo, se podría tener un programa que genere despliegues visuales en forma automática en base al reconocimiento de componentes hidrodinámicos previamente generados en forma automática y que tome los nombres de las variables correspondientes de las estructuras en memoria compartida para etiquetar los despliegues.

Se podría crear un módulo de comunicaciones del simulador de aula que enlace los nodos de interés en forma automatizada, en el que cada nodo representa una o más computadoras corriendo uno o más modelos o despliegues, y que requiere indispensablemente la existencia de segmentos de memoria compartida bien identificados.

REFERENCIAS

1. Committee on the safetey of nuclear Installations. *"TASK 5. Role of Simulators in Operator Training"*. Nuclear Energy Agency. Organisation for Economic Co-operation and Development, NEA/CSNI/R(97)13, June 1998.
2. www.cfe.gob.mx marzo, 2005.
3. www.cnsns.gob.mx/inucl.htm febrero, 2005.
4. CFE-CLV. *"Del fuego a la energía Nuclear"*. México, agosto 2002.
5. Barrero L. P. *"Descripción general del simulador de la Central Laguna Verde"*. Boletín IIE, marzo-abril de 1992.
6. C. Chávez-Mercado, *"A Classroom Analysis Simulator for the Laguna Verde Nuclear Power Plant."* Proceedings of the ANS International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies.Vol. I, pp 445-450. The Pennsylvania State University, PA. May 6-9, 1996.
7. C. Chávez Mercado, *"An Advanced Graphical Human-Machine Interface for a Classroom Analysis Simulator of Nuclear Processes."* Proceedings of the IERE Workshop on Human Factors in Nuclear Power Plants.Operation Session (2) Interface. TEPCO R & D Centers, Tokio, Japan. May 27-30, 1996.
8. B. E. Zayas Pérez, C. Chávez Mercado, *"Interfase Gráfica Hombre-Máquina Avanzada para un Simulador de Procesos Nucleares para Entrenamiento en Aula"*. México Nuclear, Revista de la Sociedad Nuclear Mexicana, año 2 No. 5. págs. 8-16, Abril-Junio 1996.
9. DataViews. DV-Tools Reference Manual, DV-Tools User's Guide. DataViews Corporation. Northampton, Ma. USA. 1995.
10. Coryell, E. W. et al. *"SCDAP/RELAP5 Programmers Manual"*. Idaho, Idaho National Engineering Laboratory, May 1994.
11. Márquez García, Francisco Manuel, *"UNIX Programación Avanzada"*, Alfaomega, México 2001.
12. Rosen H. Kenneth et al. *"UNIX Sistema V Versión 4"*. McGraw Hill. España 1997.
13. Kart Wall, *"Programación en LINUX 2ª edición Al descubierto"*, Prentice Hall, España 2001.

14. The SCADAP/RELAP5 Development Team. *"RELAP5/MOD 3.2 code manual"*. Appendix A. Idaho National Engineering and Environmental Laboratory, USA(1995).
15. Salazar Cravioto, José Humberto. *"Incorporación de los códigos RELAP/SCADAP al simulador de procesos nucleares para análisis y entrenamiento en aula"*. Tesis profesional para obtener el titulo de Ingeniero Mecánico Electricista. Facultad de Ingeniería, UNAM. Marzo 2002.
16. Cortés Martell, Fco. Samuel. *"Desarrollo de interfaces graficas avanzadas prototipo basadas en los códigos nucleares RELAP/SCADAP y MELCOR"*. Tesis Profesional para obtener el título de Ingeniero Eléctrico Electrónico. Facultad de Ingeniería, UNAM. Marzo 2003.
17. The SCADAP/RELAP5 Development Team. *"RELAP5/MOD 3.2 code manual"*. Volume V. Idaho National Engineering and Environmental Laboratory, USA(1995).