Visualizador de Estructuras de un Sistema Operativo Real con Fines Educativos

Graciela De Luca[1], Martín Cortina[1], Nicanor Casas[1], Esteban Carnuccio[1], Sebastián Barillaro[1], Sergio Martín[1], Gerardo Puyo[1]

[1] Universidad Nacionald de La Matanza,
San Justo, Buenos Aires Argentina
{gdeluca, mcortina , ncasas, ecarnuccio, sbarillaro, smartin, gpuyo}@ing.unlam.edu.ar

Resumen. El presente trabajo se centra en los avances relativos al desarrollo de una interfaz de depuración remota. Si bien el desarrollo de la interfaz y el graficador es genérico, inicialmente se basará en el sistema operativo S.O.D.I.U.M[1], del cual tenemos completo conocimiento y control.
Para asegurar la interoperabilidad de nuestro desarrollo con el depurador GDB[2], se está analizando e incorporando un módulo remoto denominado gdbstub, que resuelve la comunicación a nivel lógico, implementando el protocolo RSP. Se analizan también las técnicas utilizadas por los depuradores modernos en cuanto a la implementación del mecanismo de *breakpoints* y el soporte que la arquitectura IA32 provee para facilitar dicha tarea. En función de esto estudiaremos también las responsabilidades que debe tener un manejador de excepciones de depuración por hardware.

Palabras Clave: Visualizador - Sistema Operativo Educativo – S.O.D.I.U.M. - Breakpoints - Gdbstub - Comunicación Serial

1 Introducción

S.O.D.I.U.M. es un sistema operativo realizado con propósitos didácticos, que permite durante su ejecución la reconfiguración, cambiando los algoritmos utilizados por los administradores del sistema. Esto tiene como propósito didáctico permitir la comprensión y el análisis del funcionamiento interno del sistema operativo, pudiendo de esta manera adquirir competencias en la evaluación de los algoritmos y la elección de los sistemas operativos comerciales estableciendo rendimientos y comportamientos de acuerdo al entorno en el que se realiza la ejecución. Para esto la investigación se centra en la construcción de una aplicación con fines didácticos que permita la visualización gráfica del funcionamiento interno, estructuras de datos del sistema, valores de las variables, estados de los procesos en S.O.D.I.U.M., con el fin de modificar mínimamente la ejecución del sistema, se decidió realizar esta visualización en otra máquina, estudiando para este propósito los diferentes protocolos utilizados a tal fin. La enseñanza teórica tradicional de un sistema

[1] Sistema Operativo del Departamento de Ingeniería de La Universidad de La Matanza
[2] GNU Project Debugger

operativo se basa principalmente en el estudio de una gran cantidad de bibliografía relacionada con el tema. La forma tradicional de la enseñanza universitaria está basada en proporcionar conocimientos teóricos y luego pasar a la práctica para aplicar la teoría aprendida, con esta propuesta se podrá estudiar distintos casos propuestos por el docente, analizando el comportamiento interno del sistema operativo, sin interferir prácticamente en su ejecución.
En consecuencia, el desarrollo de una herramienta que permita visualizar el funcionamiento de un sistema operativo facilitará la enseñanza de los profesores a sus alumnos, dado que se podrá analizar el comportamiento de los distintos módulos del mismo en menor cantidad de tiempo. Otro de los beneficios, es que podrán visualizarse gráficamente los mecanismos utilizados por un sistema operativo real al momento de resolver problemas específicos. De esta forma el alumno podrá aprender más rápidamente su funcionamiento, pudiendo ver el estado del sistema en un determinado momento. La enseñanza tradicional de un sistema operativo se basa en tres pilares, los estudiantes modifican o amplían parte de un sistema operativo; ellos escriben código para demostrar aspectos de la tecnología en un sistema operativo comercial; además ejecutan código que simulan partes de la tecnología de un sistema operativo.
El sistema operativo S.O.D.I.U.M. se construyó siguiendo la primera premisa. Como consecuencia a cada grupo se les asignó una investigación y el desarrollo de una sección determinada de este sistema operativo, con lo que su complejidad fue aumentando a medida que fueron pasando los años. A su vez, se fue incrementando la necesidad de poder visualizar gráficamente el funcionamiento, para apreciar el esfuerzo de tanto tiempo de trabajo y que luego éste pueda ser utilizado por otras instituciones educativas. Por lo tanto la investigación en curso se dispone a desarrollar un visualizador que permita la observación de su comportamiento

2 Estado del arte

Las actuales aplicaciones que permiten visualizar el comportamiento de sistemas complejos, se diferencian en la manera en que estos han sido implementados, debido a que se desarrollan de acuerdo al entorno en el que van a ser utilizados.

- **Simulación de máquina**. Software que simula todos los componentes Hardware que constituyen una computadora con la finalidad de poder ejecutar un sistema. Gracias a esto se puede acceder totalmente al Hardware, en forma no intrusiva, y al estado del software utilizando simulaciones.
- **Instrumentación en Tiempo Real**: Otros sistemas usan el detalle del Hardware físico, Firmware e instrucciones de Software para observar el rendimiento de los sistemas en Tiempo Real.

Los Sistemas de visualización descriptos en [2], [3] y [4] simulan una parte del funcionamiento de un sistema operativo para su enseñanza y aprendizaje. Esto trae como desventaja que el alumno puede no llegar adquirir los conocimientos necesarios sobre el funcionamiento completo de un sistema operativo. Dado que solo se estudia determinadas funcionalidades de dicha plataforma. Además, no se estaría trabajando con Hardware real, ya se emplean en entornos simulados.

Por otra parte se encuentra el Sistema Rivet [1], que si bien trabaja con hardware físico en tiempo real y permite la creación rápida de prototipos para mostrar datos puntuales, no está orientado al ámbito educativo sino más bien al análisis de rendimiento de Sistemas.

En consecuencia a lo anteriormente mencionado, se pretende que el Visualizador del sistema operativo S.O.D.I.U.M. sea una aplicación destinada para el estudio educativo sobre el funcionamiento completo de un sistema operativo tradicional. Por lo que se procurará representar gráficamente las características más importantes que el sistema ya posee. El cual además, incorporará las funcionalidades más significativas de los visualizadores previamente descriptos. Para ello, se procura que pueda ser utilizado tanto en entornos simulados como reales, brindando de esta forma mayor libertad a los usuarios, ya que se estará otorgando la posibilidad de que dicho visualizador pueda ser ejecutado tanto en la misma terminal en donde se estará ejecutando S.O.D.I.U.M. como en otra distinta. De esta forma se obtiene una gran diferencia con respecto de los visualizadores existentes, ya que se estaría trabajando tanto en un entorno real como simulado, pudiendo visualizar el completo funcionamiento de un sistema operativo real. Cabe mencionar, que para poder efectuar la comunicación entre la maquina servidor, donde se estará ejecutando S.O.D.I.U.M. y la maquina cliente, donde se estará ejecutando el visualizador del sistema, se desarrollaron distintos mecanismos de comunicación serial. Los cuales permiten el intercambio de datos entre programa visualizador y el sistema que se desea graficar.

3 Visualizador del Sistema Operativo S.O.D.I.U.M.

En el escrito [5], se describe de qué forma se planea construir la aplicación del Visualizador del S.O.D.I.U.M. . En él se plantea que dicho software sea desarrollado de manera tal que pueda ser utilizado por los usuarios de dos formas distintas:

a.- Ejecutando S.O.D.I.U.M. en una máquina virtual

b.- Ejecutado S.O.D.I.U.M. en una máquina física.

Utilizando la primera alternativa mencionada, el alumno podrá ejecutar una imagen del sistema operativo en una máquina virtual dentro de una misma plataforma, y al mismo tiempo interactuar con este utilizando el programa visualizador. Seguidamente, en la figura 1 se muestra como se implementaría la aplicación en la misma terminal. Si bien un entorno simulado no presenta, en su totalidad, las mismas características que una maquina real, se determinó que era conveniente desarrollar esta opción para el caso en particular en que los educadores y estudiantes no posean dos terminales en donde realizar las pruebas pertinentes con el Visualizador de S.O.D.I.U.M. En consecuencia, los usuarios podrán ejecutar el paquete completo del Visualizador en la misma terminal, utilizando la máquina virtual Bochs para poder ejecutar una imagen compilada de S.O.D.I.U.M. dentro de un entorno de trabajo bajo Linux. Esto es importante debido a que la utilización de dicha VM permite trabajar emulando casi en su totalidad las mismas prestaciones que ofrece el hardware de una máquina real en una computadora totalmente distinta.

Por otra parte se está desarrollando la posibilidad de ejecutar S.O.D.I.U.M. en una máquina real conectada a otra terminal en donde se estará ejecutando el Visualizador. Por dicho motivo, en la figura 2 se podrá observar esta situación en particular.

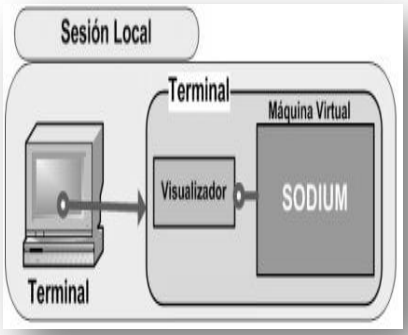

Fig. 1. Representación gráfica de la ejecución del visualizador y S.O.D.I.U.M. en la misma terminal

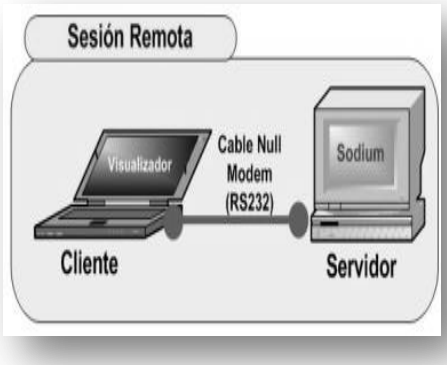

Fig. 2. Representación gráfica de la ejecución del visualizador y SODIUM en dos terminales.

4 Conexión entre terminales

Como se mencionó en [5], con el objeto de conseguir la comunicación entre visualizador y sistema operativo, se desarrolló un driver que permite el intercambio de datos a través de los puertos serie. Por consiguiente, se consideró necesario establecer un contrato en la configuración de dicha conexión entre la aplicación cliente y servidor, con la finalidad de conseguir correctamente la transferencia de información. Utilizándose inicialmente una transmisión con el formato estándar 8N1, a una velocidad de transferencia de 9600 bps y utilizando un cable Null-Modem con conectores RS-232. Este acuerdo fue necesario dado que la rapidez de la transacción de datos depende fundamentalmente de factores físicos, tales como el modelo del chip UART utilizado y la longitud del cable serial.

En consecuencia, se están realizando distintas pruebas utilizando un hardware determinado, con el objetivo de poder ir aumentando gradualmente la velocidad de transferencia de información. Intentando ver, la factibilidad de alcanzar la máxima velocidad posible de transferencia de 115200bps a través de medios seriales.

5 Visualización de Estructuras del Sistema Operativo

Según [1], cualquier sistema visualizador debe ser capaz de gestionar grandes cantidades de datos, manipulándolos y realizando cálculos que permitan generar rápidamente prototipos para que el usuario pueda observar y comprender la información obtenida. Por consiguiente, se pretende que el visualizador reciba información sobre los componentes y estructuras del sistema operativo durante su ejecución, de forma que al recibir estos datos la aplicación generé automáticamente gráficos comprensibles por el estudiante.

La interacción entre el usuario y el visualizador es la característica fundamental en el diseño de todo visualizador de un sistema complejo, dado que de esta manera el alumnado puede llegar a entender más rápido los conceptos. Por consiguiente, como el visualizador es un aplicación externa, se determinó que era conveniente investigar la forma de implementar un mecanismo que permita al usuario controlar la ejecución del sistema operativo remotamente. Así, el visualizador ofrecerá la posibilidad de detener las operaciones que realiza S.O.D.I.U.M. y ver el estado de sus componentes en el momento en que se desee.

La primera parte del análisis, consistió en comprender el funcionamiento básico de los mecanismos utilizados por los debuggers para controlar la ejecución de cualquier programa.

6 Análisis de técnicas utilizadas por los Debuggers

La tarea principal es detener la ejecución de los programas en determinados momentos establecidos por los usuarios, donde se podrá analizar su estado y el del procesador en ese instante, lo que se pretende realizar en S.O.D.I.U.M. imitando la esencia de dicho comportamiento.

El corazón de todo depurador es el breakpoint, también conocido como punto de parada. Los cuales pueden ser clasificados según el mecanismo utilizado durante su desarrollo [6]. Los Breakpoints desarrollados por software son los más utilizados por los debuggers existentes. Esto es debido a su simplicidad y alcance durante su implementación. Si bien estos no presentan las mismas utilidades que los que ofrecen los desarrollados por Hardware, presentan el beneficio de poder ser utilizados en gran cantidad, mientras que los otros se limitan a utilizar únicamente cuatro puntos de paradas, como consecuencia de que dicha cantidad es determinada por los registros de la CPU DR 0, 1, 2 y 3. Otra diferencia destacable, es que para su implementación solo es necesario reemplazar un solo byte en la dirección de inicio de la instrucción donde se desea detener la ejecución, por el Opcode de la instrucción assembler “Int3”. Mientras que para la utilización de los breakpoint por hardware, es indispensable indicar en los registros DR0-DR3 la dirección en donde se desea detener la ejecución, e indicar además las condiciones que deberán cumplir esas direcciones en el registro DR7. Gracias al empleo del último registro nombrado, los HW Breakpoints pueden

ser utilizados para detener la ejecución de un programa al momento de ejecutar una instrucción. Así como también, cuando se lean o escriban datos en una dirección particular de memoria. Esta característica puede llegar a ser provechosa, para la situación especial en que se desea detener la ejecución de S.O.D.I.U.M. cuando se escriban o lean datos en las direcciones asignadas a los puertos COM durante la comunicación con el Visualizador.

En consecuencia, algunos debuggers solamente implementan breakpoints por Software. Sin embargo, también existen depuradores que utilizan ambas clases de puntos de paradas, como por ejemplo GDB.

6.1 Ejecución de Breakpoints implementados por Software

Esto se puede realizar en el código del programa de dos formas distintas. La primera de ellas es insertando en el código fuente la instrucción assembler INT 3, pero este mecanismo únicamente puede ser utilizado antes de la compilación de la aplicación. La segunda posibilidad es la que llevan a cabo los debuggers, comentada en el punto anterior, que es reemplazar el Opcode de la instrucción a detener por el de Int3 (0xCC) durante la ejecución [7][8]. Por lo que después que se produce dicha sustitución, el sistema operativo deberá retroceder el registro EIP en un byte con la finalidad de ejecutar dicha instrucción. Una vez que se produce la ejecución de dicho comando, de cualquiera de las dos formas antes descriptas, se producirá una excepción 3, siendo capturado por el handler del sistema operativo luego de este evento.

6.2 Ejecución de Breakpoints implementados por Hardware

Una vez configurados, los registros de parada, el procesador compara la dirección de la instrucción en ejecución con el valor contenido en los registros DR0-DR3, y si existe coincidencia posteriormente evaluará las condiciones de debug declaradas en el registro DR7. En el caso de las comparaciones sean satisfactorias, la CPU emitirá una excepción 1, la que deberá ser capturada por el handler en el Kernel del sistema operativo.

Hasta el momento en las pruebas iniciales en la investigación en curso, se consiguió capturar dichos sucesos desarrollando los handlers correspondientes, de forma tal, que pudieron ser invocados utilizando la inserción de la instrucción INT en el código fuente de los programas utilizados en S.O.D.I.U.M. Actualmente, se está analizando la factibilidad de implementar la segunda posibilidad para la ejecución de los SW Breakpoints, que es la inserción del Opcode de la instrucción INT 3 durante la ejecución del sistema operativo, además se está determinando si es viable el desarrollo de los HW Breakpoints configurando los registros de la CPU.

7 Análisis de Gdbstub

GDB ofrece un módulo denominado Gdbstub [9] que permite analizar el funcionamiento de programas en entornos remotos. Esto es particularmente deseable donde estos entornos, por sus características de implementación (escases de recursos, ausencia de consola local, problemas de accesibilidad, etc.), no son capaces de darle al desarrollador la posibilidad de trabajar localmente.

Este módulo se provee en forma de código fuente en lenguaje C como parte del kit de desarrollo del depurador GDB, del cual además se disponen varias implementaciones, cada una especializada para interactuar con una arquitectura de procesador en particular.

El equipo desde donde se efectúa el control del programa principal es llamado *host* (huésped), mientras que la computadora donde está funcionando la aplicación a analizar es conocida como *target* (objetivo). Cabe mencionar, que entre ambas terminales se establece un vínculo por medio de una conexión serie.

El propósito concreto de este módulo es el de implementar la capa lógica de comunicación entre el depurador GDB que se ejecuta en el equipo *host* y el programa o sistema a ser depurado en el *target*. Para ello se utiliza el protocolo RSP, que ya fue comentado en [5].

Al aislar al implementador de la necesidad de resolver el desarrollo de la capa lógica de este protocolo de comunicación, se gana estabilidad y robustez en la solución, permitiendo además enfocar el esfuerzo sobre la interacción específica de este módulo con el sistema operativo en sí mismo.

Esta aislación se logra presentando al desarrollador una interfaz clara (contrato) donde se define una serie de métodos cuyas responsabilidades el mismo deberá implementar para facilitar al módulo gdbstub la obtención de información del sistema.
Estas rutinas son las que permiten atender y responder los mensajes recibidos por el vínculo serial,

Gdbstub implementa un *handle-exception* que toma el control cuando se detiene la ejecución del proceso, por ejemplo, en un breakpoint. En ese momento, handle-exception se comunica con GDB en la máquina host. Handle-exception actúa como un representante de GDB en la máquina target. Comienza por enviar un resumen de información del estado del proceso. Luego, continúa la ejecución, recibiendo y transmitiendo cualquier información que GDB necesita. Cuando GDB ordena resumir la ejecución normalmente, Handle-exception devuelve el control al propio código de la aplicación en la máquina target. Cada vez que handle-exception es llamada, ésta tiene la oportunidad de tomar el control. Esto puede suceder todo el tiempo, inclusive cuando se reciben caracteres por la comunicación serial. De todas formas, se puede forzar la interrupción llamando a la función *breakpoint.*

8 CONCLUSIONES

Hasta la fecha en la investigación en curso, se consiguió establecer la base inicial de una de las funcionalidades esenciales que deberá ofrecer el visualizador de S.O.D.I.U.M. en cuanto a la interacción con el estudiante. A partir del handler desarrollado, se buscará que el usuario pueda detener la ejecución del sistema operativo en forma remota desde el visualizador. Posteriormente podrá observar detalladamente el estado del sistema en ese instante. En consecuencia, como se mencionó anteriormente, se está evaluando la factibilidad de desarrollar breakpoints por software y/o hardware así como también complementar las funcionalidades del handler construido con las que ofrece el módulo de Gdbstub. De esta manera, se está construyendo una de las partes fundamentales que tendrá el visualizador de estructuras de un sistema operativo con fines educativos.

Referencias

1. Robert P. Bosch Jr., "*Using Visualization to Understand The Behavior of Computer System",* Agosto 2001.

2. Farzaneh Zareie y Mahsa Najaf-Zadeh " *OSLab: A Hand-on Educational Software for Operating System Concepts Teaching and Learning",* Research WebPub, Septiembre 2013

3. Besim Mustafa, "*Visualizing the Modern Operating System: Simulation Experiments Supporting Enhanced Learning",* Edge Hill University, SIGITE'11, Año 2011

4. Ali Alharbi, Frans Henskens, and Michael Hannaford,*"Integrated Standard Environment for the Teaching and Learning of Operating Systems Algorithms Using Visualizations",* The University of Newcastle, Australia, IEEE, Año 2010

5. Graciela De Luca, Martín Cortina, Nicanor Casas, Esteban Carnuccio, Sergio Martín, "*Mecanismos de visualización de estructuras de un sistema operativo en ejecución a través de la comunicación serial",* Universidad Nacional de La Matanza, Congreso WICC 2014, Ushuaia, Tierra del Fuego.

6. Intel, "*Intel® 64 and IA-32 Architectures Software Developer's Manual",* Mayo 2007

7." Debugging in AMD64 64-bit Mode in Theory", http://x86asm.net/articles/debugging-in-amd64-64-bit-mode-in-theory/index.html

8. ***"****How debuggers work: Part 2 – Breakpoints",* http://eli.thegreenplace.net/2011/01/27/how-debuggers-work-part-2-breakpoints/

9. *"Debuging Remote Program:",* https://sourceware.org/gdb/onlinedocs/gdb/Remote-Debugging.html#Remote-Debugging