Mecanismos de visualización de estructuras de un sistema operativo en ejecución a través de la comunicación serial.

Graciela De Luca, Martín Cortina, Nicanor Casas
Esteban Carnuccio, Sergio Martín.

Departamento de Ingeniería e investigaciones Tecnológicas
Universidad Nacional de La Matanza
Florencio Varela 1903 - San Justo, Buenos Aires, Argentina
Te.(54-11) 4480-8900
gdeluca@ing.unlam.edu.ar; martin.cortina@yahoo.com.ar; ncasas@ing.unlam.edu.ar
estebancarnuccio@gmail.com; smartin@ing.unlam.edu.ar

Resumen

Dentro del contexto de un proyecto en el que nos proponemos lograr el control y la visualización de las estructuras internas de un sistema operativo en tiempo de ejecución, nos encontramos con la necesidad de implementar un protocolo de comunicación viable entre el sistema estudiado y el de control. En este documento compartiremos las líneas de investigación que estamos llevando a cabo para lograrlo, teniendo como objetivos imponer un mínimo costo adicional en la comunicación y compatibilidad con herramientas de depuración remota basadas en GDB (GNU ***DeBugger***). Para ello estudiaremos el protocolo ya implementado por este depurador y su capacidad de expansión. También analizaremos la factibilidad de llevar a cabo esta comunicación tanto por puerto serie como a través de la arquitectura de plug-ins disponible en Bochs

Palabras clave: GDB, Remote Serial Protocol (RSP), gdbstub, Comunicación Serial, Instrumentación, Bochs.

Contexto

Esta Línea de Investigación es parte del proyecto "Visualización de estructuras internas de un Sistema Operativo en ejecución como herramienta didáctica", dependiente de la Unidad Académica del Departamento de Ingeniería e Investigaciones Tecnológicas perteneciente al programa de Investigaciones CyTMA2 de la Universidad Nacional de La Matanza, el cual es continuación de otros proyectos, en especial del sistema operativo de características didácticas: SODIUM.

Introducción

El proyecto en el cual está inserta esta línea de investigación estará enfocado a permitir la visualización de diversas estructuras internas y estadísticas de un sistema operativo, mientras el mismo se encuentra en ejecución, primeramente del sistema operativo SODIUM desarrollado por este equipo de investigación y luego posteriormente se prevé ampliarlo a otros sistemas operativos. Se pretende poder comprobar didácticamente el funcionamiento de los algoritmos de planificación y administración de recursos utilizados, brindando de esa forma una herramienta importante a la

hora de enseñar la teoría relacionada a sistemas operativos y arquitectura de computadoras.

Se decidió la implementación de un driver de puerto serie para SODIUM debido a que es un dispositivo de fácil interacción y el costo de manejarlo en cuanto a cantidad de ciclos de CPU no es significativo. Estimamos que entre las alternativas posibles (puerto serie, placa de red, puerto USB) esta es la que menos *overhead* proporcionará al Sistema Operativo a nivel de ciclos de CPU.

En una primera etapa tenemos la intención de desarrollar un protocolo de comunicación bidireccional que nos permita alterar la ejecución de un sistema operativo y obtener los estados y estructuras internas del mismo. Dentro de este contexto, deberemos definir un set de puntos de instrumentación necesarios para identificar la situación actual de cada módulo y estructura de interés.

Implementación de puerto serie en SODIUM

Debido a que hasta el momento SODIUM carecía de un controlador de puerto serie, fue necesario desarrollarlo, para lo cual recopilamos la información necesaria para su construcción [4][5]. Teniendo en cuenta que para la comunicación serial se utilizarán *interrupciones y polling*, indistintamente, fue necesario profundizar en el funcionamiento de la UART[1] [4], específicamente de los registros para el manejo de señales mediante *interrupciones* en modo asíncrono y llevar un control que se define en el driver del puerto Serie.

El driver presenta un conjunto de funciones que permiten inicializar los puertos COM, establecer la configuración de comunicación (velocidad de transmisión, formato de envío de datos, entre otros), transmisión, recepción y tratamiento de errores, contemplando el uso de *interrupciones* y *polling*, utilizando los ***Registros de direcciones de los puertos de E/S*** (tabla 1), para definir los puertos COM y una línea de interrupciones IRQ para llamar la atención del procesador y los ***Registros utilizados para la comunicación*** (tabla 2).

Puerto	Registro	(IRQ)
COM1	3F8	4
COM2	2F8	3
COM3	3E8	4
COM4	2E8	3

Tabla1 Direcciones estándar en la mayoría de las arquitecturas en base hexadecimal

Registros E/S	Descripción
3F8/2F8	Registro de datos Emisión/Recepción
3F9/2F9	Registro de habilitación de interrupciones
3FA/2FA	Registro de identificación de interrupciones
3FB/2FB	Registro de control de la línea
3FC/2FC	Registro de control del MODEM
3FD/2FD	Registro de estado de la línea
3FE/2FE	Registro de estado del MODEM

Tabla 2 Registros para la comunicación

Se utilizará inicialmente una transmisión de 9600bps con formato 8N1 (8 bits de datos | sin paridad | 1 bit de parada). Posteriormente se evaluará incrementar la velocidad gradualmente hasta 115200bps.

El cable a utilizar es un NULL MODEM con un extremo de cable RS-232 y el otro extremo un USB adaptado, el cual permitirá la conexión con cualquier dispositivo que se use como terminal (laptop, PC sin conexión Serie, etc.).

[1] "**U**niversal **A**synchronous **R**eceiver-**T**ransmitter"

Inicio de la sesión de visualización

Durante una sesión de visualización, se utilizarán dos computadoras personales, de las cuales una de ellas puede llegar a encontrarse virtualizada. A la primera la denominaremos *Cliente* y es en donde se estará ejecutando la aplicación de visualización. Esta máquina posee el programa GDB instalado con la finalidad de depurar SODIUM remotamente. La segunda terminal, *Servidor* es donde se estará ejecutando el sistema operativo a ser estudiado. Para interconectar ambos equipos, la situación varía, por lo que detallaremos cada caso individualmente.

Computadora Física: El sistema operativo puede ejecutarse en una PC de escritorio conectada físicamente con el *cliente* a través del cable serie. Para los casos en los que la PC cliente no posee un puerto serie propio, se puede anexar un puerto serie por USB que, una vez reconocido, funcionará correctamente.

Máquina Virtual: Necesaria para el caso particular en que el usuario no cuente con disponibilidad para utilizar más de una máquina para poder probar el funcionamiento. Como consecuencia, se permitirá ejecutar este sistema operativo en un emulador como Bochs o Vmware.
En la investigación en curso, se utiliza un entorno de desarrollo en el cual se puede ejecutar una imagen de Linux montada en máquina virtual Vmware, y a su vez dentro de ella es posible ejecutar el Sistema Operativo SODIUM en una VM Bochs.

Análisis y extensión del protocolo de comunicaciones utilizado por GDB.

Mediante la utilización de este puerto se prevé la posibilidad de controlar dicho sistema operativo de forma remota, permitiendo detener y reanudar su ejecución cuando el usuario lo desee. Dado que las implementaciones actuales del GDB nos permiten depurar remotamente el sistema a través del soporte de un módulo denominado **gdbstub** ya embebido en la máquina virtual BOCHS, y estando ya familiarizados con su uso y virtudes, consideramos un paso natural extender esta funcionalidad a la ejecución del SODIUM directamente sobre un equipo físico. Otro punto a favor es que los desarrolladores del GDB proporcionan implementaciones básicas de este módulo listas para adaptar a nuevas plataformas o arquitecturas de hardware.

De acuerdo a lo estudiado [1], el protocolo de comunicación remota serie utilizado por GDB (de ahora en más RSP) posee las siguientes características:

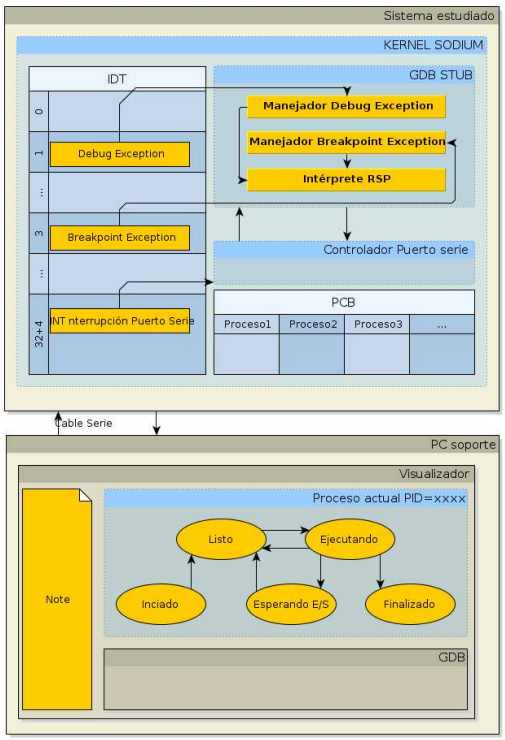
- Basado en el set de caracteres ASCII. Esto significa que todos los caracteres enviados se encuentran dentro del rango imprimible, lo que nos facilita su estudio, documentación, y depuración
- Posee caracteres de inicio y terminación de mensaje claros. La parte útil de cada mensaje comienza por un símbolo “$” y termina con un símbolo “#”
- Las operaciones se definen por un caracter alfabético sensible a las mayúsculas seguido por una lista de argumentos requeridos, separados por coma.
- Posee control de errores. Luego del símbolo terminador de cada mensaje enviado se escribe un número que es el resultado de la suma módulo 256 de la suma del peso de cada uno de los caracteres comprendidos.
- La recepción de un mensaje se confirma inmediatamente por el receptor, indicando “+” si el mismo se recibió íntegramente, “-“ si se detectó un error en el formato del mensaje o durante la ejecución del mismo. Siguiendo a dicho

caracter, si es necesario, se transcribe el resto de la respuesta, o un código y descripción de error.
- Una transacción típica consiste en la emisión de un comando por parte del depurador hacia el sistema remoto y su respectiva respuesta. La única excepción a este comportamiento se da cuando el sistema remoto emite un mensaje hacia el depurador con el objetivo de que este último lo muestre en su salida de consola. Esta acción puede llevarse a cabo en cualquier momento siempre y cuando no exista una transacción todavía en proceso.
- Los datos respondidos por el sistema depurado pueden estar comprimidos mediante la técnica RLE (Run-length encoding) o no, con lo que se puede reemplazar secuencias de más de tres caracteres consecutivos similares por la emisión de un carácter, un signo *, y un contador numérico de repeticiones.

Dentro de los comandos estándares que el RSP contempla, podemos encontrar las siguientes operaciones:
- Iniciar o detener la ejecución de un programa remoto.
- Leer o escribir uno, varios o todos los registros de la CPU a la vez.
- Leer o escribir una cantidad de bytes a partir de una dirección de memoria determinada.
- Ejecutar instrucciones de máquina de a una por vez o ejecutar a toda velocidad hasta solicitar la detención o haber alcanzado el sistema remoto un punto de excepción.

Para permitir la visualización del estado de ejecución del sistema operativo estudiado [2][3], debemos tener en cuenta una solución que nos proporcione la flexibilidad suficiente como para conectar tanto el GDB como nuestro propio programa cliente.

Para que el visualizador esté al tanto en tiempo real acerca de la creación de nuevos procesos, asignaciones de memoria, cambios de estado en el planificador, cambios de contexto, accesos a dispositivos de entrada y salida, accesos al sistema de archivos, y otros, deberemos informar al mismo cada vez que estemos transitando las regiones de código fuente de nuestro kernel asociadas a dichas acciones.
Denominamos a estas regiones “puntos de instrumentación”.
Dentro de las posibles soluciones que estamos planteando se encuentran:
- Utilización de ***Hardware Watchpoints*** implementados a través de los ***Debug Registers*** provistos por la arquitectura i386.
- Invocación a rutinas de log en los Puntos de Instrumentación.



.

Ilustración 1 Diagrama en bloques de la implementación del gdbstub dentro del kernel de SODIUM y representación del estado de un proceso en la interfaz del visualizador [3].

Luego de tomar el control en el punto adecuado, proponemos comunicar al visualizador el cambio de estado. El mensaje deberá ser originado desde el sistema estudiado, de modo que el visualizador deberá soportar la recepción asincrónica del mismo. Para ello, podemos o bien tomar ventaja del mecanismo de salida a consola contemplado en el RSP, o desarrollar el soporte de un nuevo set de operaciones en el gdbstub y en el mismo GDB que contemple la totalidad de los puntos de instrumentación a definir.

Líneas de Investigación, Desarrollo e Innovación

- Investigar protocolos de comunicación serie existentes y adoptar o definir el que utilizaremos, para enviar datos entre la Máquina Cliente y el Servidor, para transferir los datos y las estructuras a visualizar, del sistema operativo.
- Determinar capacidad máxima sostenida de transferencia mensajes para nuestro protocolo de comunicación.
- Investigar la factibilidad de desarrollar un plug-in para máquina virtual capaz de ejecutar acciones especificadas al momento en el que el sistema operativo atraviese un punto de instrumentación.

Resultados y Objetivos

Actualmente hemos concluido el desarrollo del driver del puerto serie para el sistema operativo SODIUM.

Estamos comenzando la etapa de análisis relativa al protocolo RSP y del módulo gdbstub provisto por GDB.

A su vez, nos encontramos sentando las pautas de desarrollo del visualizador, que en una primera etapa se conectará con SODIUM, y en etapas posteriores con otros sistemas operativos de código abierto, para poder analizar el funcionamiento y utilizarlo como elemento didáctico para la enseñanza de Sistemas Operativos.

Formación de Recursos Humanos

El grupo de trabajo consta de dos investigadores de categoría IV, un investigador categoría V, un investigador con 4 años de experiencia y un investigador inicial.

Referencias

[1]Bill Gatliff "*Embedding with GNU: the GDB Remote Serial Protocol*" revista Embedded Systems Programming, Noviembre 1999

[2]Robert P. Bosch Jr., "*Using Visualization to understand the Behavior of Computer Systems, Ph.D. dissertation, Stanford University*", August 2001.

[3]Alharbi, A.; Henskens, F.; Hannaford, M., "*Integrated Standard Environment for the Teaching and Learning of Operating Systems Algorithms Using Visualizations*," Computing in the Global Information Technology (ICCGI), 2010 Fifth International Multi-Conference on , vol., no., pp.205,208, 20-25 Sept. 2010

[4]"*Serial and UART Tutorial*", http://www.freebsd.org/doc/en/articles/serial-uart/

[5]"Puertos Serie" http://wiki.osdev.org/Serial_Ports