Visualizador de Estructuras de un Sistema Operativo Educativo

Graciela De Luca, 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

Abstract. El presente trabajo describe los avances conseguidos durante el desarrollo de una aplicación que permitirá la visualización remota de estructuras lógicas y el control de un sistema operativo para uso académico, con el objetivo de facilitar la asimilación de conceptos que, en primera instancia, suelen ser demasiado abstractos. Este será suficientemente flexible como para representar el estado interno de cualquier sistema operativo, aunque inicialmente se basará en el Sistema Operativo SODIUM.
En este documento describiremos el análisis efectuado sobre los chips UART de los dispositivos seriales con la finalidad de optimizar el funcionamiento del driver serie de SODIUM y el diseño preliminar de algunas de las estructuras que el visualizador representará. Finalmente se describirán los mecanismos que se están implementando en SODIUM con el objetivo de conseguir detener su ejecución en forma remota mediante puntos de parada.

Keywords: Interrupciones, instrumentación, puntos de parada, SODIUM, comunicación serial, RSP, GDB-Stub, UART, sistemas operativos, visualización de estructuras, GDT, IDT, PCB, control de ejecución.

1 Introducción

Cualquier sistema visualizador debe poseer la capacidad de generar diagramas en base a grandes cantidades de datos de forma inmediata y de una manera que les permita a los usuarios interpretar fácilmente dicha información [1]. En consecuencia, el visualizador que se está desarrollando en este proyecto procurará recibir información sobre los distintos componentes y estructuras que conforman el sistema operativo durante su ejecución y, en simultáneo, generar gráficos entendibles para que los estudiantes puedan comprender su funcionamiento. La arquitectura Intel presenta distintos componentes que son utilizados por los sistemas operativos para el funcionamiento de una PC. La interrelación entre ambos actores es frecuentemente es de difícil comprensión. Los sistemas operativos arman distintas estructuras para administrar los recursos de una computadora en base a los elementos que proveen sus fabricantes.

El Sistema Operativo SODIUM fue construido con fines educativos para que pueda ser utilizado por estudiantes y docentes para el aprendizaje práctico del funcionamiento de un SO tradicional. Hasta la fecha, SODIUM no presenta un visualizador gráfico que permita observar sus estructuras, componentes y funcionamiento de sus algoritmos. Con propósito de brindar una nueva herramienta de estudio se lleva a cabo este proyecto de investigación, cuya meta es desarrollar un Visualizador gráfico que pueda ser utilizado en un principio por SODIUM y posteriormente por otros Sistemas Operativos.

El presente documento describe los distintos mecanismos que se han implementado en el proyecto para poder ir alcanzando el objetivo planteado al inicio de la investigación

2 Visualizador externo del Sistema Operativo

Como bien se mencionó anteriormente, este proyecto de investigación tiene como objetivo principal generar un software que permita observar diferentes estructuras del Sistema Operativo SODIUM, implementándolo posteriormente para su uso en otros sistemas. La comunicación entre SODIUM y el programa visualizador se efectuará a través de dispositivos seriales. Por consiguiente, fue necesario desarrollar un driver que permita la comunicación entre distintas terminales empleando los puertos COM disponibles en una computadora [7]. En la figura siguiente se detalla cómo se efectuará la implementación final de la aplicación en las distintas unidades.

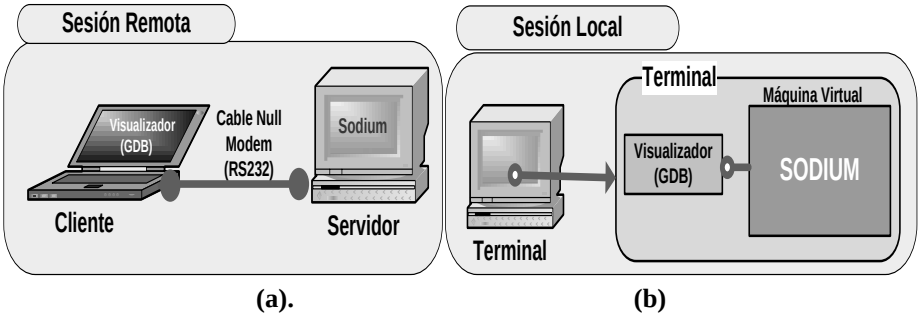

Fig. 1. (a) Representación gráfica de la ejecución del visualizador y SODIUM en dos terminales. **(b)** Representación de la ejecución en la misma terminal

3 Características del driver de puerto serie implementado

Por su simpleza, la comunicación serie fue uno de los medios de comunicación punto a punto entre computadoras y dispositivos más difundidos en los albores de la micro computación. La función principal de esta controladora (UART) es la de tomar cada byte leído del bus de datos paralelo y serializarlo en un tren de bits, enviándolos

de a uno por vez través del canal de comunicación. A su vez, la misma contempla el proceso inverso en su canal de recepción. La norma de comunicación asociada es la RS-232 [5], que especifica las características eléctricas y mecánicas de las interfaces así como velocidades admitidas, control de flujo y uso de bits adicionales para señalización y detección de errores. Esto permite, en particular al chip (16650A), la opción de configurar la generación de interrupciones a la CPU cuando se reciben 1, 4, 8 ó 14 bytes [6].

La norma RS-232 especifica un par de líneas de datos (una para enviar y otra para recibir) y 6 para control, con las que implementa el control de flujo. De esta manera, el equipo receptor puede indicarle al equipo emisor que está listo para recibir, o que se abstenga de continuar enviando datos. Así se logra adaptar la comunicación al estado de ambos equipos.

El driver serial implementado actualmente en SODIUM fue construido sin aprovechar el uso del buffer FIFO que poseen los chips UART más modernos (como el 16550A mencionado anteriormente).

A lo largo de las pruebas efectuadas durante esta fase del desarrollo, se identificó el riesgo de que la tasa inicial de transmisión de datos entre terminales que logramos obtener pudiera no ser suficiente para cubrir nuestras necesidades. Dicho riesgo nos llevó a relevar el modelo del chip UART utilizado en una muestra de computadoras convencionales de distintas marcas, determinando al fin que sí es común encontrar la capacidad de *buffering* en las mismas. Con esta conclusión, sabemos que posteriormente podremos implementar el uso de dicha capacidad en SODIUM mejorando así la tasa de transferencia del driver ya desarrollado.

Según la documentación de VMware, el software ofrece 4 puertos serie a los sistemas virtualizados. Cualquiera de estos puertos puede ser direccionado a un puerto físico real del equipo, o a un archivo, o interconectado entre dos computadoras virtuales. La UART virtualizada es un modelo compatible con el chip 16550A, que ofrece un buffer FIFO de 16 bytes. Los cuatro puertos series comparten las IRQ 3 y 4 de a pares [10].

En el caso de Bochs, también son 4 los puertos serie virtualizados. Permite elegir el modo de operación de manera similar a VMware: null, raw, mouse, file, term (sólo para Unix), FIFO y Socket (sólo para Windows). En todos los casos, el chip emulado ofrece compatibilidad con el muy difundido 16550A [11].

4 Diseño preliminar del Visualizador del Sistema Operativo

El sistema operativo SODIUM fue desarrollado para ejecutar en equipos con arquitectura x86. Dada su naturaleza de sistema operativo de estudio, se buscó ejercitar todas las características que esta arquitectura provee, como ser protección, segmentación fija, segmentación paginada, cambios de contexto por hardware y manejo de interrupciones y excepciones. Por consiguiente, para poder utilizar la memoria, CPU y dispositivos externos se emplean distintas estructuras que están

intrínsecamente relacionadas. Lo que se pretende mostrar por medio del visualizador es el estado de dichos datos y su variación en el tiempo ante distintos eventos en tiempo real.
Cabe destacar, en contraste, que otros sistemas operativos, cuyo objetivo es ser fácilmente portables a distintas arquitecturas de hardware, buscan mantener la mayoría de sus mecanismos de administración de recursos agnósticos a la arquitectura, utilizando únicamente el mínimo denominador común de características presentes en las mismas, como ser paginación. Otro mecanismo provisto por la arquitectura x86 que es frecuentemente ignorado, tanto por compatibilidad como por eficiencia, es el cambio de contexto por hardware.

Seguidamente, se describen algunas de las distintas estructuras que utiliza SODIUM que van a ser representadas gráficamente y cómo se va a exponer su interrelación por el visualizador en su etapa inicial.

GDT (GLOBAL DESCRIPTOR TABLE)

La arquitectura Intel ofrece la posibilidad de utilizar unas tablas de descriptores alojadas en memoria para poder ubicar los segmentos que se utilizan en el sistema. Una de esta estas tablas es la GDT, apuntada por el registro GDTR de la CPU [4]. Dicha estructura contiene esencialmente los descriptores de códigos, datos y stacks utilizados por el sistema operativo para ubicar los segmentos correspondientes en la memoria principal. Para hacer uso de esta herramienta, SODIUM administra esta tabla utilizando una estructura en forma de vector. Cada elemento de dicho vector contiene fundamentalmente Tipo de descriptor (Código, datos o stack), dirección base del segmento, tamaño del segmento y nivel de privilegio DPL. En consecuencia, en base a estos ítems, SODIUM consigue acceder a la información de cada proceso en memoria en un momento determinado. Para esto se utilizan otros elementos que se comentarán seguidamente.

PCB (PROCESS CONTROL BLOCK)

Esta estructura contiene toda la información necesaria de un proceso en particular que un sistema operativo utiliza para poder llevar a cabo su administración durante su tiempo de vida. De forma tal que el SO pueda asignarle eficientemente los recursos que este emplea. Siendo así, SODIUM maneja dicha herramienta para gestionar los distintos procesos que utiliza el sistema. Este componente se encuentra conformado en un vector de estructuras que almacenan distinta información. Entre las más importantes se encuentran el ID del proceso, el ID del padre, estado, nivel de prioridad y los índices de descriptores de segmentos de GDT asignados a código, datos y stack.

La visualización gráfica de esta información es trascendental para que el estudiante pueda comprender el funcionamiento del sistema.

IDT (INTERRUPT DESCRIPTOR TABLE)

Otra tabla que el Sistema Operativo SODIUM debe gestionar, aplicando la tecnología que ofrece Intel, es la IDT. Esta estructura contiene los descriptores de segmentos en donde se encuentran alojadas las rutinas de atención de las interrupciones. Esta tabla consiste en un vector de estructuras de 256 posiciones. Cada posición del vector presenta un selector de segmento para la GDT, Offset, Tipo y nivel de privilegio DPL. En consecuencia, cada vez que ocurre una interrupción en modo protegido, el sistema operativo SODIUM utiliza esa estructura para ubicar la rutina de atención asociada a dicho evento.

MAPA DE MEMORIA

Una de las características importantes que se pretende desarrollar en el visualizador es que este permita observar en tiempo de ejecución la ubicación de los componentes que conforman al sistema operativo y procesos en la memoria principal. La siguiente figura muestra un croquis del diseño preliminar de cómo se pretende mostrar el estado de la memoria en un momento dado.

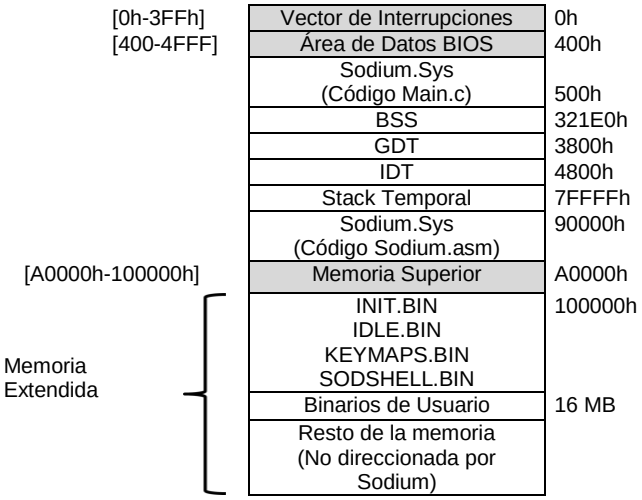

Fig. 2. Croquis del Mapa de Memoria que representará el visualizador

Todas las estructuras anteriormente mencionadas son algunos de los componentes que se pretende que el visualizador de SODIUM muestre inicialmente, dado que es muy importante poder ver gráficamente su contenido. De esta manera, el usuario podrá observar e interactuar con dichos componentes a través los diagramas que los representan [1] [2]. En la figura siguiente se muestra una de las relaciones que se pretenderá graficar entre las tablas PCB, GDT y la ubicación de los distintos componentes en la memoria principal.

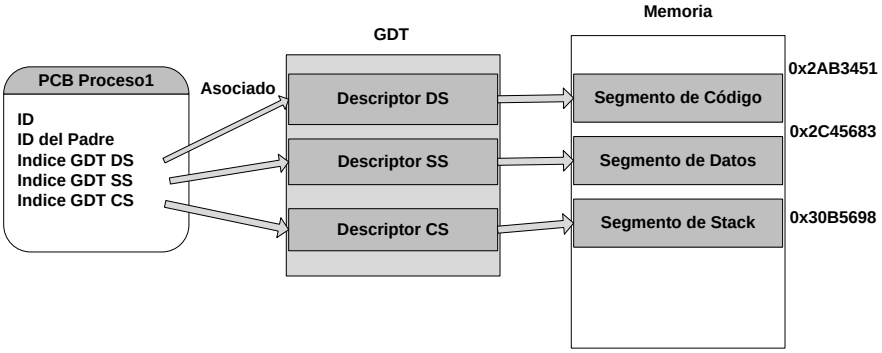

Fig. 3. Relación entre estructuras de SODIUM

Logueo al inicio de SODIUM

Se desarrolló en SODIUM una funcionalidad que permite enviar mensajes arbitrarios a la terminal remota con un formato específico de logueo. Este módulo fue confeccionado basándose en el comando FTRACE de Linux. Estos mensajes tienen como fin notificar al usuario la ocurrencia de distintos eventos de SODIUM. Actualmente, se informan en la terminal remota los acontecimientos ocurridos desde el inicio de SODIUM hasta que la consola del sistema operativo es activada. Sin embargo, se pretende que a futuro utilizar este tipo de mensajes para informar al visualizador de la ocurrencia de sucesos del sistema operativo. Denominaremos "Puntos de Instrumentación" a las regiones de código que generen este tipo de mensajes.

El formato de logueo utilizado es el siguiente:

Criticidad	*Fecha*	*Hora*	*Evento*	*Descripción*
0	24/07/2014	21:23:33	1024	ES_LA

Fig. 4. Formato de logueo remoto

- **Criticidad:** Indica el nivel de importancia del mensaje de log. Varía de 0 a 5 siendo 5 el nivel más crítico.
- **Fecha:** Indica la fecha en que se generó el evento en el sistema operativo.
- **Hora:** Indica la hora en que ocurrió el suceso.
- **Evento:** Es el tipo de evento. Este número entero es conocido tanto por el sistema operativo como por el visualizador. Por ejemplo, el número 1024 puede indicar el evento de carga satisfactoria de un mapa de caracteres para la consola activa.
- **Descripción:** Contiene la descripción del hecho acontecido. Siguiendo con el ejemplo anterior, puede contener el nombre de la distribución de teclado cargada.

5 Detención de la ejecución del Sistema Operativo

Uno de los principales objetivos planteados desde el inicio del proyecto consistió en que el usuario tuviera la posibilidad de detener la ejecución del Sistema Operativo desde la interfaz de visualización, a fin de poder observar el estado de las distintas estructuras que lo componen en un momento determinado. Esta característica es muy importante, dado que otorga a los estudiantes la facultad de interactuar con SODIUM en forma remota, pudiendo así reafirmar los conocimientos teóricos impartidos al respecto.

Utilización de puntos de parada

En la investigación en curso, se hizo hincapié en la detención del Sistema Operativo SODIUM en forma remota. Por dicho motivo, primeramente se analizó el funcionamiento de otros depuradores existentes y de los mecanismos que éstos utilizan para implementar puntos de parada en distintos programas de usuario. En consecuencia, se observó que algunas de estas herramientas emplean dos tipos diferentes de puntos de parada [9][3]: los implementados por software y los implementados por hardware

Si bien en SODIUM se desarrollaron las rutinas de manejo de excepciones generadas tanto por breakpoints de software como de hardware, se determinó que era conveniente trabajar únicamente con el primer tipo en la etapa inicial del desarrollo de la detención remota del Sistema Operativo. Esta decisión se fundamentó en que los puntos de parada desarrollados por software no presentan limitaciones de uso e implementación respecto de su contraparte.

Inicialmente las invocaciones de la excepción número 3, asociada a los puntos de parada por software, podían ser realizadas únicamente desde el código del Kernel del Sistema Operativo SODIUM. Esta restricción se debió a que las excepciones eran ejecutadas con un nivel de privilegio con valor 0, correspondiente al DPL de la Interrupt Gate asociada. En consecuencia, un programa de usuario no podía utilizar un punto de parada de este tipo durante su ejecución. Por este motivo, se determinó cambiar el nivel de privilegio del DPL de esta excepción asignándole el valor número 3. De esta manera se consiguió que un programa de usuario pueda ejecutar una instrucción que genere una interrupción de punto de parada durante su ejecución en SODIUM.

Adaptación de GDB-Stub

Con la finalidad de controlar la ejecución del Sistema Operativo SODIUM en forma remota, se analizó el funcionamiento del módulo que ofrece GDB, denominado GDB-Stub [8]. Se indagó en la comprensión de sus funcionalidades, con el objeto de obtener los conocimientos necesarios para emplearlos luego en el proyecto. En base al estudio llevado a cabo, se determinaron los cambios necesarios que se debieron efectuar en SODIUM a los fines de conseguir detener su ejecución desde otra

terminal. En los siguientes apartados se describirán los mecanismos que se están empleando con este propósito.

Proceso de detención de SODIUM en forma remota

A raíz del análisis mencionado, se descubrió que la tarea principal de GDB-Stub se centra en su función "handle_exception", cuya funcionalidad elemental consiste en detener el sistema y luego esperar en forma activa las peticiones de los usuarios que estén conectados remotamente a ella. En consecuencia, se desarrolló en SODIUM un módulo que emula su funcionamiento. En la figura 5 se visualiza el funcionamiento básico de este componente:

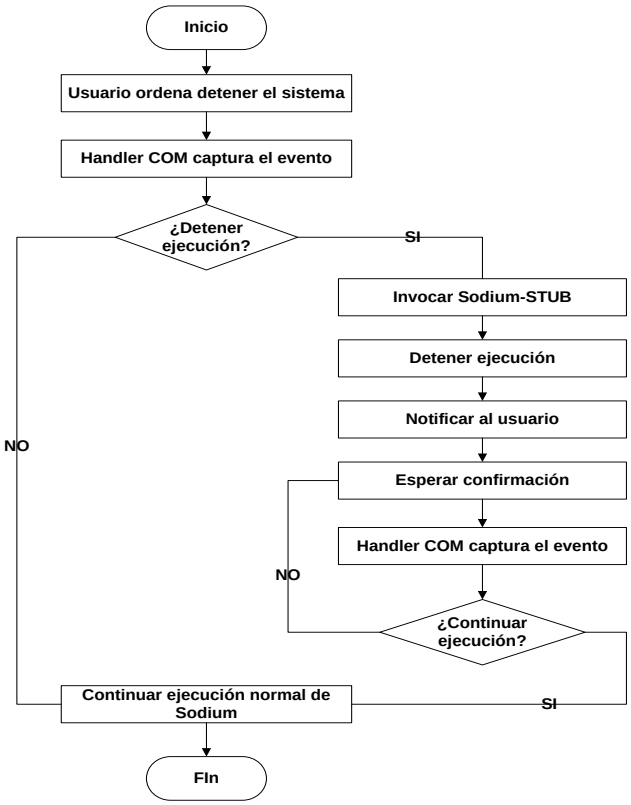

Fig. 5. Diagrama de flujo sobre la detención de la ejecución del sistema

En el gráfico anterior se puede observar el diagrama de flujo esencial de la detención del sistema en forma remota. Primeramente el usuario envía, desde otra terminal, un mensaje a SODIUM indicándole que detenga su ejecución. Como la comunicación entre el cliente y el sistema operativo se efectúa de acuerdo al protocolo RS-232, este aviso es recibido por el controlador de puerto serie [7].

Cuando la UART recibe un byte desde otro dispositivo, genera una interrupción que es capturada por el manejador asociado al puerto COM que se está utilizando. Actualmente en esta investigación se está empleando el puerto COM1, por lo que se estará activando la IRQ (Interrupt Request) número 4. Una vez capturada esta interrupción, se determina si la señal enviada por el cliente es de detención. En caso afirmativo, se invoca al módulo desarrollado en SODIUM que emula el funcionamiento de "handle_exception" (de ahora en más SODIUM-Stub). Esta función detiene la ejecución del Sistema Operativo y envía un mensaje de notificación al usuario comunicándole que el sistema se detuvo y queda en espera de las peticiones del cliente. Finalmente, el usuario envía un mensaje solicitando reanudar la ejecución del sistema operativo, y este es recibido por el manejador del puerto serie. Acto seguido, el manejador invoca a SODIUM-Stub y éste restablece la ejecución normal de SODIUM.

El procedimiento anterior describe el funcionamiento actual del módulo encargado de la detención del Sistema Operativo. Cabe señalar que este componente aún continúa optimizándose, por lo que puede sufrir variaciones en un futuro.

Visualización de estructuras cuando SODIUM se encuentre detenido

Se pretende que el sistema operativo transfiera el estado de cada una de sus estructuras internas al visualizador en el instante en que el usuario detenga su ejecución. De tal forma, el usuario podrá interactuar con las mismas observando en detalle su composición. Con esta característica se pretende generar una herramienta educativa que consiga transferir a los estudiantes los conocimientos teóricos y prácticos acerca del funcionamiento interno de un Sistema Operativo tradicional.

Manejadores y puntos de parada

Al inicio del desarrollo de la detención remota del sistema, se había planificado que SODIUM ejecute automáticamente un punto de parada por software al recibir un mensaje de solicitud de parada desde una terminal remota. Luego, una vez que el manejador capturara la excepción 3 generada, invocaría al módulo SODIUM-Stub para llevar a cabo las peticiones del usuario. No obstante, posteriormente se determinó que era más eficiente llamar directamente al módulo SODIUM-Stub, en lugar invocarlo a través de la generación de una excepción. Esto fue debido a que se producía una disminución en el rendimiento del sistema dado por el anidamiento de manejadores. Por dicho motivo, se decidió evitar la utilización de la excepción 3 al momento de recibir un mensaje de detención y que en su lugar se invoque directamente a la función SODIUM-Stub, tal como se muestra en la figura 4.

Es importante mencionar que el estado actual de este mecanismo todavía presenta una merma en la productividad del sistema, dado que aún se continúa ejecutando con anidamientos de manejadores. Por dicho motivo, actualmente se está trabajando para conseguir subsanar dichas dificultades. Además, cabe señalar que, si bien no se emplearán en la detención del sistema operativo los mecanismos desarrollados para el

manejo de puntos de parada, estos serán resguardados en el código fuente de SODIUM para futuros proyectos.

6 Conclusión:

Hasta la fecha, en la investigación en curso, se consiguió desarrollar una característica fundamental en el funcionamiento del Visualizador que consiste en lograr la posibilidad de detención del Sistema Operativo SODIUM en forma remota. De esta manera se le otorga al usuario la posibilidad de detener el sistema en el preciso momento en que desee analizar el estado del mismo. Además de lo anteriormente dicho, se pudo establecer preliminarmente el tipo de estructuras que el Visualizador mostrará al usuario por medio de diferentes representaciones gráficas. Conjuntamente, a través del envío de mensajes asincrónicos hacia el visualizador, se busca informar al usuario en forma descriptiva la ocurrencia de distintos eventos del sistema. Como se mencionó anteriormente, estas funcionalidades deben seguir optimizándose, de forma tal que puedan ser empleadas satisfactoriamente por los estudiantes y educadores. No obstante, se consiguió construir una parte fundamental en el desarrollo del Visualizador del sistema Operativo.

7 Referencias

1. Robert P. Bosch Jr.:"Using Visualization to Understand The Behavior of Computer System": pp 13-16 (2001).
2. Farzaneh Zareie y Mahsa Najaf-Zadeh: "OSLab: A Hand-on Educational Software for Operating System Concepts Teaching and Learning": Research WebPub: pp 1-3 (2013)
3. Prasad Krishnan: "Hardware Breakpoint (or watchpoint) usage in Linux Kernel", IBM Linux Technology Center, Canada: pp 1-10 (2009)
4. Intel: "Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide": pp 65-6,413 (2011)
5. Dallas Semiconductor:"Fundamentals of RS-232 Serial Communications, Application note 83", Sunnyvale, (1998)
6. David S.Lawyer: "Serial HowTo" Greg Hankin: Capítulo 18: (2011)
7. 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", Congreso WICC (2014)
8. The GNU Project Debugger, https://sourceware.org/gdb/onlinedocs/gdb/Remote-Debugging.html#Remote-Debugging
9. http://x86asm.net/articles/debugging-in-amd64-64-bit-mode-in-theory/index.html
10. VMware Inc., http://www.dpunkt.de/leseproben/1686/Kapitel_2.pdf
11. Bochs Emulator Project, http://bochs.sourceforge.net/doc/docbook/user/bochsrc.html