RISC5 y Project Oberon para Raspberry Pi

David Camilo Barrera Ribero
Escuela Colombiana de Ingenieria Julio Garavito
Ingeniería de Sistemas
David.barrera-r@mail.escuelaing.edu.co.

***Resumen*- EL procesador RISC5 y Project Oberon fueron desarrollados en el ETH "Eidgenössische Technische Hochschule Zürich" por Niklaus Wirth y Jürg Gutknecht a finales de 1980, lo cual está completamente documentado en su libo "Project Oberon". La idea portar el procesador RISC5 y por ende el sistema Oberon, viene dada por la popularidad de la Raspberry Pi como una plataforma económica y accesible, para dar a conocer el sistema Oberon dentro de la comunidad de la Raspberry Pi, y utilizar Project Oberon y la Raspberry Pi para enseñar como diseñar e implementar un sistema operativo.**

La estrategia utilizada para portar el emulador del procesador RISC5 y el sistema operativo Oberon, fue portar los driver comunes entre el emulador del sistema operativo y la Raspberry Pi, estos fueron la pantalla, mouse "USB", teclado "USB", disco el cual en la Raspberry Pi es una tarjeta SD.

***Palabras Clave*- RISC5,Oberon,portar,emulador.**

I. Introducción

Project Oberon y Commodore 64 son ambos sistemas operativos desarrollados en los años 80. Cuando la raspberry pi fue lanzada en el 2012, esta se presentó como una oportunidad para desarrollar nuevos sistemas operativos orientados a su arquitectura, y portar viejos sistemas operativos orientados a diferentes arquitecturas, como es el caso de Commodore 64. Por otra parte Oberon es un sistema modular de un usuario, de un proceso operativo y multitarea desarrollado en le ETH Zürich utilizando el lenguaje de programación Oberon. Tiene una interfaz de usuario basada en texto visual no convencional "TUI" para la activación de comandos, y fue diseñado para que funcionara sobre un procesador de arquitectura RISC5, el cual es un procesador virtual diseñado por el equipo que desarrollo Oberon.

La idea es portar el sistema operativo Project Oberon basándose en el proyecto Commodore-pi, en el cual se portó el sistema operativo Commodore 64, a través de identificar y portar los periféricos comunes entre el sistema operativo Project Oberon y la raspberry pi. En este documento se explica la implementación y los problemas afrontados al portar el sistema operativo Project Oberon.

A continuación en este documento se presenta como se portó el emulador del procesador RISC5 y el sistema operativo Porject Oberon en una raspberry pi, a través de identificar los periféricos comunes entre la raspberry pi y el sistema operativo Project Oberon. Esto se realizó tomando como referencia y guía los proyectos Commodore Pi, en el cual se portó el sistema operativo Commodore 64, y rpi-boot, el cual es un boot alternativo para la raspberry pi, además de utilizar la librería csud para la implementación de los drivers de USB.

II. Spartan 3 y RISC5

La motivación para el proyecto de diseñar un procesador de arquitectura RISC (del inglés Reduced Instruction Set Computer, en español Computador con Conjunto de Instrucciones Reducidas) y su implementación en un FPGA (del inglés Field Programmable Gate Array), proviene primeramente de diseñar e implementar un pequeño procesador para su uso en sistemas embebidos con varios núcleos interconectados (TMS Tiny Register Machine), y en segundo lugar, es el libro (Compiler Construction) escrito por Niklaus Wirth, el cual trata sobre la construcción de un compilador para una arquitectura de computador hipotética. Esta idea se hizo realista por el advenimiento de los componentes de hardware programables llamados matrices de puertas programables de campo (FPGA).

Gracias a la sencilla arquitectura del procesador RISC, la implantación de este se realizó en un FPGA ya que esta proporciona una cantidad sustancial de libertad para el diseño, para la realización del circuito se utilizó el lenguaje de realización de hardware Verilog el cual provee una ventaja sobre un esquema grafico de un circuito.

La arquitectura del procesador RISC se compone de elementos individuales de direcciones de 8 bits, cuanta con un ALU el cual tiene un banco de 16 registros de 32 bits, cada 32 bits se denomina palabra. Como es característico en los procesadores RISC, los datos pueden transferirse entre la memoria y los registros, por instrucciones de carga y almacenamiento separadas.

El desarrollo del procesador RISC sigue cinco capas de desarrollo, la primera de ellas es propio desarrollo de la arquitectura de este, le sigue el ISC-0 la cual tiene dos memorias distintas, una se utilizan para el programa y la otra para los datos, esta arquitectura permite un buena separación en entre la unidad aritmética y la unidad de control. En la siguiente etapa RISC-1 la BRAM para los datos se sustituye por este SRAM. Para la etapa del RISC-2 se cambia la arquitectura a una arquitectura Von Neumann. En la última etapa RISC-3 se le añaden dos características, interrupciones y acceso a bit.

III. Emulador del procesador RISC de Oberon

Existen dos emuladores para el procesador virtual RISC5 de Oberon, uno para Windows y otro para Linux, ambos están escritos en C y requieren de SDL2.

Para el caso de Windows el emulador viene con un dll para SDL2 y un ejecutable, con lo cual no es necesario compilar las fuentes del mismo. Por el contrario para Linux es necesario instalar SDL2 y tener un compilador con el estándar C99 para poder compilar las fuentes del emulador, para emular el sistema operativo Oberon es necesario especificar la ruta de RISC.img en el ejecutable que se genera.

En ambos casos el emulador provee acceso a la pantalla, disco, USB mouse y teclado, además de transferir archivos a través de los scripts pcreceive.sh y pcsend.sh. Por ahora el emulador no provee acceso a red.

IV. Sistema Oberon

El sistema Oberon fue diseñado y desarrollado en el ETH de Zürich por Niklaus Wirth y Jürg Gutknech, el desarrollo fue documentado completamente en su libro "Project Oberon the design of an operating system, a compiler, and a computer". Con este libro sus autores justificar el esfuerzo y su motivación de diseñar y construir todo un sistema operativo desde cero, y presentan una serie de conceptos básicos para la construcción de este por capítulos.

Para la realización del sistema operativo Oberon se utilizó Modular-2, aunque este leguaje contaba con la desventaja ligar al sistema operativo a cierto tipo de máquina, lo cual iría en contra de la propiedades esperadas de este sistema operativo. Por lo tanto, Modula-2 se amplió con una característica de extensión de tipo. Al mismo tiempo se reconoció que Modula-2 contenía varias instalaciones que no utilizarían y que no contribuyen realmente a su poder de expresión, pero al mismo tiempo aumentar la complejidad del compilador. Pero el compilador no sólo tendría que ser implementado, sino también debe ser fácilmente descrito, estudiado y entendido. Esto llevó a la decisión de empezar desde cero también en el dominio de diseño del lenguaje, y para aplicar el mismo principio "concentrarse en lo esencial". El nuevo lenguaje, que todavía lleva mucha semejanza con Modula-2, se le dio el mismo nombre que el sistema Oberon.

V. Portar El Procesador Risc5 Y Project Oberon En Una Raspberry Pi

Ya que el sistema Oberon y el procesador virtual RISC5 fueron planeados originalmente para ser ejecutados en un FGPA, en específico en una Sparten 3, la cual esta descontinuada en la actualidad y es difícil adquirir una ya utilizada, se planteó, dada la popularidad de la Raspberry Pi, portar el sistema Operativo en esta. Ya que así se daría a conocer más este sistema operativo, empezando por la comunidad de la Raspberry Pi. A demás como Project Oberon y la Raspberry Pi fueron diseñados y orientados a la enseñanza y aprendizaje, estos proveen una gran oportunidad para facilitar la enseñanza de cómo funcionan los sistemas operativos y como se pude desarrollar uno en un computado de una arquitectura relativamente sencilla.

Se planteó portar el emulador del procesador RISC5 de Oberon porque este al estar escrito en el lenguaje de programación C y utilizar SDL2, se facilita remplazar las funciones que hacen uso de un sistema operativo o del SDL por unas similares que hacen uso directamente de las funciones del procesador ARM.

VI. Portando Los Periféricos

Como se mencionó anteriormente para porta el sistema operativo Project Oberon, se debió portar los periféricos comunes entre el sistema operativo y la raspberry pi, los cuales son la pantalla, USB en este caso mouse y teclado, la tarjeta SD la cual actúa como disco.

Para la implementación del procesador virtual RISC5 y el sistema operativo Oberon se utilizó SDL para la simulación de los drivers de pantalla, mouse y teclado. Un SDL (Simple DirectMedia Layer) es un conjunto de bibliotecas desarrolladas en ellenguaje de programación C que proporcionan funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, además de carga y gestión de imágenes.

A. Pantalla y Framebuffer

Como ya se había mencionado anteriormente, en la implementación del procesador virtual RISC5 y del sistema operativo Oberon se utilizaron funciones SDL para hacer la conexión con la pantalla, a través de una ventana la cual simula el "escritorio" del sistema operativo Oberon. Estas funciones proveen la posición, colores y texturas que se necesitan para que se pueda crear la ventana donde se pude visualizar la pantalla principal de Oberon .Ya que las funciones que proveen las bibliotecas SDL utilizan funciones propias de un sistema operativo, en este caso Linux y Windows ya que el emulador esta implementado para estos dos sistema operativos, por lo tanto fueron remplazadas por funciones que no son dependientes de un sistema operativoy utilizan directamente el procesador risc de la raspberry pi. Tomando como guía Commodore-pi se remplazaron las funciones de pantalla provistas por el SDL por unas equivalentes provistas por el módulo terminal, el cual prepara y se comunica con el módulo framebuffer que es el encargado de la comunicación con la pantalla, este provee las funciones que piden el tamaño de la pantalla "GetSreenSize" y que establece comunicación con el buffer de pantalla "SetupScreen" , para ello framebuffer utiliza el módulo mailbox, el cual es el encargado de hacer la lectura y escritura en el buffer de pantalla. El módulo terminal también hace uso del módulo graphics, el cual provee las funciones "DrawLine", "DrawRectangle", "DrawCircle", "DrawPixel", "DrawCharacterAt","DrawFilledRectangle", "SetVirtualFrameBuffer" las cuales son necesarias para que el sistema operativo grafique.

B. USB, Mouse y Teclado

Para los drivers de USB, los cuales son teclado y mouse se utilizó la librería csud, la cual fue desarrollada por Alex Chadiwick, esta librería es un controlador USB originalmente escrito para la raspberry pi (que utiliza un DesignWare USB 2.0 de alta velocidad On-The-Go (OTG HS) Controlador), para ser integrado en cualquier sistema operativo. Csud está diseñado para funcionar ya sea como una sección de código independiente sin dependencias externas, o como más conductor típico, con dependencias externas. En sí CSUD es modular, y por lo podría tener componentes intercambiados o reemplazados. De forma similar a la implementación del controlador de la pantalla del procesador virtual RISC5 y del sistema operativo Oberon se utilizaron funciones SDL para hacer la conexión con el teclado y mouse, lo cual trae el problema de acoplamiento a un sistema operativo, el cual fue descrito en la sección anterior, por ello se remplazaron los llamados a las funciones SDL por aquellas provistas por la librería csud tales como, "KeyboardInitialise", "KeyboardUpdate","KeyboardGetChar","MouseGetPosition X", "MouseGetPositionY", "MouseGetButtonIsPressed".

C. SD, Disk

En la implementación del emulador para el procesador virtual RISC5 y del sistema operativo Oberon, las funciones encargadas de carga y almacenamiento de información en

disco, son realizadas a través de una estructura Disk dentro del módulo Disk, esta estructura tiene la imagen del sistema operativo Oberon, una variable offset, una variable rx_buf la cual indica que comando se debe ejecutar y una variable tx_buf la cual tiene la información a ser guardada en disco, o es la variable en la cual se deposita la información leída del disco. Y el modulo disc dispone de las funciones necesarias para leer y escribir en el disco, de una forma similar a las implementaciones de pantalla y USB, la implementación de disco está acoplada a un sistema operativo ya que utiliza las funciones fopen, fseek, fread, fwrite y malloc. Por lo tanto se volvieron a implementar estas funciones eliminando los llamados a funciones propias de un sistema operativo, las cuales proveían acceso al disco ya fuera para lectura o escritura, y remplazarlas por funciones de lectura y escritura en la SD, ya que esta hace de disco en la raspberry pi. Las funciones fopen, fseek, fread, fwrite y malloc, fueron tomadas del proyecto rpi-boot, el cual es un proyecto de boot alternativo para la raspbery pi, el cual ayuda en el desarrollo de kernels.

VII. CONCLUSIONES

Para el desarrollo de proyectos similares a este, en el cual se trabajan con tecnologías poco conocidas y en donde se espera simular resultados semejantes obtenidos en proyectos con características diferentes, o que nunca se ha realizado un proyecto similar, es la falta de documentación y de expertos en el tema, los factores que más determina el tiempo del proyecto y su posible éxito o fracaso.

El proceso de diseñar, desarrollar e implementar un sistema operativo, es un proceso que requiere gran cantidad de tiempo y conocimiento, en las áreas de hardware como de software, ya que se debe conocer bien las plataformas donde este sistema se va a desplegar y con las que este va a trabajar e interactuar.

REFERENCIAS

[1] Niklaus Wirth.: The Design Of A Risc Architecture And Its Implementation With An Fpga. [11.11.11, rev. 5.10.13]

[2] Niklaus Wirth.: The Risc Architecture. [NW 5.12.10, rev. 1.2.2014].

[3] Niklaus Wirth.: Jürg Gutknecht.: Project Oberon The Design of an Operating System,a Compiler, and a Computer. [ISBN 0-201-54428-8].

[4] Niklaus Wirth.: The Programming Language Oberon. [Revision 1. 10. 90].

[5] CSDU. Consultado el 5 mayo de 2015, de https://github.com/Chadderz121/csud.

[6] Commodore-Pi. Consultado el 5 mayo de 2015, de http://www.commodorepi.co.nr/.

[7] rip-boot. Consultado el 5 mayo de 2015, de https://github.com/jncronin/rpi-boot.

[8] Raspberry pi. Consultado el 5 mayo de 2015, de https://www.raspberrypi.org.

[9] Repository Raspberry pi. Consultado el 5 mayo de 2015, de https://github.com/raspberrypi.