Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Sistema de antivirus y antispam implementado en un FPGA

Zetina Moya José Enrique - 206200773
Licenciatura en Ingeniería en Computación
Asesor M. en C. Oscar Alvarado Nava

Julio de 2012

Resumen

El desarrollo de aplicaciones de cómputo sobre dispositivos empotrados, como lo son los FPGAs, tiene gran interés para la industria de la seguridad informática, ya que permite crear sistemas de cómputo orientados a una tarea específica sin la necesidad de un sistema de cómputo completo.

Actualmente, existen multitud de sistemas de seguridad basados en *software*. Algunos proporcionan estabilidad aceptable y están integrados en el sistema operativo, otros por ejemplo, son únicamente interfaces gráficas que facilitan la tarea de gestionar la seguridad de un sistema de cómputo a la mayoría de usuarios y, además, proporcionan seguridad por defecto para un uso básico. Estos últimos, en entornos de mayor exigencia, no proporcionan una respuesta fiable a un ataque informático.

Hoy en día los FPGAs son dispositivos que permiten la implementación de complejos sistemas de seguridad basados en *hardware* o *software* ofreciendo a los desarrolladores una alta gama de posibilidades de implementar sus sistemas en ellos.

Índice general

Índice de figuras

Capítulo 1

Introducción

Con el incremento en el uso de internet han aumentado también las amenazas a los sistemas de cómputo. Hoy en día, un sistema informático debe ser capaz de realizar el filtrado de paquetes maliciosos en el tráfico de red. Sin embargo, las herramientas de seguridad y aplicaciones de usuarios comparten un mismo CPU[1] con recursos para el funcionamiento de herramientas de seguridad basadas en *software*.

Un sistema de cómputo se vuelve inseguro al encenderlo. Los huecos de seguridad en un sistema de cómputo se manifiestan generalmente en dos maneras:

- Huecos de seguridad físicos, el problema sucede cuando se da acceso físico a una computadora a personas sin autorización, lo cual le permitiría realizar cosas que normalmente no sería capaz de hacer.

- Huecos de Seguridad en el Software, donde los problemas son causados por puntos escritos incorrectamente en el software, los cuales pueden ser utilizados para hacer cosas indebidas.

En la actualidad para evitar el acceso remoto de personas no autorizadas a los sistemas de cómputo es necesario robustecer más los sistemas, lo que tiene repercución directa en el tiempo de desarrollo de estos, además de hacerlos mas grandes y complejos.

[1]CPU, acrónimo de *Central Processing Unit* se refiere a la unidad central de proceso de una computadora.

Los sistemas embebidos son dispositivos de propósito específico que se utilizan ampliamente en equipamientos de redes de datos, sensado remoto, comunicaciones, etc. Estos equipos han sido desarrollados para llevar a cabo un conjunto de tareas reducidas y específicas en función de su ámbito de desempeño, característica que los diferencia de las computadoras de propósito general. El sistema operativo embebido es quien le permite realizar sus funciones y en la mayoría de los casos es provisto y desarrolado por el fabricante del hardware.

Implementar un sistema embebido basado en un sistema operativo GNU/Linux que funcione de manera eficiente, robusta y sobretodo que permita modificaciones según necesidades puntuales es una tarea ardua y compleja.

Para resolver dichos problemas, este trabajo propone construir un sistema de seguridad autónomo capaz de manipular y configurar reglas de seguridad en un sistema operativo basado en Linux, lo cual será de gran utilidad para la administración y seguridad de una red de computadoras, ya que permitiría detectar actividades mal intencionadas en el uso de la red.

1.1. Motivación

La información es hoy la materia prima de las organizaciones. Tener información ayuda a tomar decisiones con seguridad y rapidez. Por tanto, proteger la información en todo momento y permitir el acceso a ella sólo para las personas que la necesiten y que, además, sea fiable, es un tema fundamental.

Los virus informáticos son hoy una realidad reconocida por las empresas, quienes saben que es un problema que mina su productividad, ya que sus computadoras están constantemente expuestas a vulnerabilidades de sus sistemas de seguridad. El *spam* afecta también a las organizaciones, pues eliminar todo el *spam* que se recibe toma tiempo y, no sólo eso sino que al ser excesivo provoca lentitud en los sistemas, lo que tiene una repercusión directa en la productividad.

El proyecto realizado describe el diseño de un sistema de seguridad de redes de

computadoras empotrado[2] en un FPGA[3] con un sistema operativo Linux instalado, por lo que es un sistema basado en *hardware* y *software*.

El sistema es capaz de proteger la integridad, confiabilidad y disponibilidad de los datos que ingresan o salen de una red de computadoras, ejecutando aplicaciones y servicios de seguridad a través de un sistema operativo basado en Linux. La red de computadoras que protegerá este sistema puede no estar basada Linux, lo que dará al sistema la posibilidad de proteger redes de computadoras basadas en otros sistemas operativos.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar e implementar un sistema de seguridad para redes de área local en la tarjeta de desarrollo NetFPGA[4] que permita bloquear el tráfico no autorizado a una red y admita al mismo tiempo comunicaciones autorizadas.

1.2.2. Objetivos específicos

- Implementar un sistema operativo basado en Linux[5] en la tarjeta de desarrollo NetFPGA el cual incluya los módulos necesarios para la detección de virus, filtrado de paquetes y acceso remoto.

- Construir en la tarjeta de desarrollo NetFPGA un sistema capaz de detec-

[2]Sistema Empotrado, traducción del inglés *embedded system*, es un sistema diseñado utilizando componentes *hardware* y *software* en un único módulo y para una aplicación específica. Se pretende así conseguir altas prestaciones y gran flexibilidad con unos costos relativamente bajos y un consumo de energía moderado.

[3]FPGA, acrónimo de *Field Programmable Gate Array*, es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad pueden ser configuradas mediante un lenguaje de descripción especializado.

[4]NetFPGA, es una plataforma de bajo costo, diseñada principalmente como una herramienta para la enseñanza de hardware de redes [1].

[5]Linux, es un núcleo libre usado en sistemas operativos de código abierto[7].

tar*malware*[6] como troyanos[7], virus[8] y otras amenazas maliciosas, con la finalidad de proteger los datos que circulan por la red de amenazas o pérdidas.

- Configurar un *firewall* [9] en la tarjeta de desarrollo NetFPGA, para filtrar paquetes y evitar intrusiones en la red.
- Configurar el servicio DHCP[10] en en la tarjeta de desarrollo NetFPGA, que permita contar con una lista de direcciones IP dinámicas y asignarlas a los equipos conforme éstas estan disponibles.
- Configurar en la terjeta de desarrollo NetFPGA el servicio *OpenSSH*[11], para poder acceder a el dispositivo de forma remota.

[6] *Malware*, es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora sin el consentimiento de su propietario.

[7] Troyano, es un software malicioso que se presenta al usuario como un programa aparentemente legítimo e inofensivo pero que al ejecutarlo ocasiona daños.

[8] Virus informático, es un programa con intenciones malignas, que es capaz de propagarse de una computadora a otra.

[9] *Firewall* o cortafuegos, es un dispositivo configurado para permitir, limitar, cifrar y decifrar el tráfico de paquetes en una red de computadoras[8].

[10] DHCP, acrónimo de *Dynamic Host Configuration Protocol*, es un protocolo que permite a dispositivos individuales en una red de direcciones IP obtener su propia información de configuración de red[6].

[11] OpenSSH, acrónimo de *Open Secure Shell*, es un conjunto de aplicaciones que permiten realizar comunicaciones cifradas a través de una red, usando el protocolo SSH[14].

Capítulo 2

FPGA

2.1. Introducción

Las FPGAs son dispositivos semiconductores que puede ser programados después de la fabricación. En lugar de limitarse a una función predeterminada los FPGAs pueden adaptarse a nuevas normas, y reconfigurar el hardware para aplicaciones específicas, incluso después de que el producto ha sido instalado.

Es posible utilizar un FPGA para implementar cualquier función lógica a un circuito integrado y tienen la capacidad de actualizar la funcionalidad después de la primera configuración por lo cual ofrece ventajas para muchas aplicaciones.

Las FPGAs actuales consisten en varias combinaciones de SRAM integrada configurable, transmisores-receptores de alta velocidad, alta velocidad de E/S, bloques lógicos y enrutamiento de paquetes[5].

Como uno de los más versátiles componentes de la tecnología VLSI[1], permite implementar complejos dispositivos electrónicos diseñados mediante lenguajes de descripción de hardware como VHDL[2] o Verilog[3], además muchos de estos también

[1]VLSI, es el acrónimo *Very Large Scale Integration*, integración en escala muy grande. La integración en escala muy grande de sistemas de circuitos basados en transistores en circuitos integrados comenzó en los años 1980, como parte de las tecnologías de semiconductores y comunicación que se estaban desarrollando.

[2]VHDL, es el acrónimo que representa la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de *Very High Speed Integrated Circuit* y HDL es a su vez el acrónimo de *Hardware Description Language*.

[3]Verilog es un lenguaje de descripción de hardware usado para modelar sistemas electrónicos. El lenguaje, algunas veces llamado *Verilog HDL*, soporta el diseño, prueba e implementación de

contienen dentro de sí diversos elementos como unidades de memoria que van desde simples *flip-flops* hasta bloques de memoria dinámica más complejos. También pueden contener multiplicadores, comparadores e incluso procesadores para formar sistemas programables en chips[11].

2.2. NetFPGA

El Proyecto NetFPGA se refiere a un esfuerzo para desarrollar un hardware de código abierto y la plataforma de software que permite la rápida creación de prototipos de dispositivos de red.

NetFPGA se distingue principalmente en dos formas. En primer lugar se encuentra en su enfoque basado en FPGA para dispositivos de red de creación de prototipos. Esto permite a los usuarios desarrollar diseños que son capaces de procesar los paquetes a velocidad de línea, una capacidad general no lograda por los enfoques basados en *software*. La segunda característica distintiva en el NetFPGA es su enfoque en el apoyo a una comunidad de *hardware* de código abierto y desarrolladores de *software* que puedan compartir y construir sobre cada uno de los proyectos y los bloques de construcción[12].

Figura 2.1: Tarjeta de desarrollo NetFPGA

El NetFPGA incluye el todos los recursos de la lógica, la memoria, y interfaces

circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción.

Gigabit Ethernet necesarios para construir un *swtich* completo, el router y o un dispositivo de seguridad. Debido a que la ruta de datos completa se implementa en el hardware, el sistema puede soportar paquetes *back-to-back* a velocidad completa de gigabit y tiene una latencia de procesamiento se mide en sólo unos pocos ciclos de reloj.

Las características con las que cuenta el NetFPGA son:

- Xilinx Virtex-II Pro 50
- 4 puertos de red Gigabit Ethernet
- 4.5 MBytes SRAM
- 64 MBytes DDR2 DRAM
- 2 conectores de entrada y salida SATA
- Slot PCI Estándar
- Conector para cable JTAG Xilinx ChipScope

2.2.1. *PowerPC 405-S Embedded Processor Core*

El procesador PowerPC 405-S de 32-bits es una implementación de la arquitectura PowerPC. Es miembro de la familia *PowerPC 400 series*, contiene componentes esenciales de un subsistema empotrado de alto rendimiento. Entre otras funciones incluye memoria gestión, memoria caché y control de temporizadores y depuración de instalaciones.

La arquitectura del procesador PowerPC 405-S implementa los registros a nivel de usuario, un modelo de programación, tipos de datos, modos de direccionamiento, y 32 bits de operaciones de punto fijo. Las operaciones de punto flotante y las excepciones que estas causan pueden ser emuladas por software, la organización del procesador se puede observar en la figura 2.2.

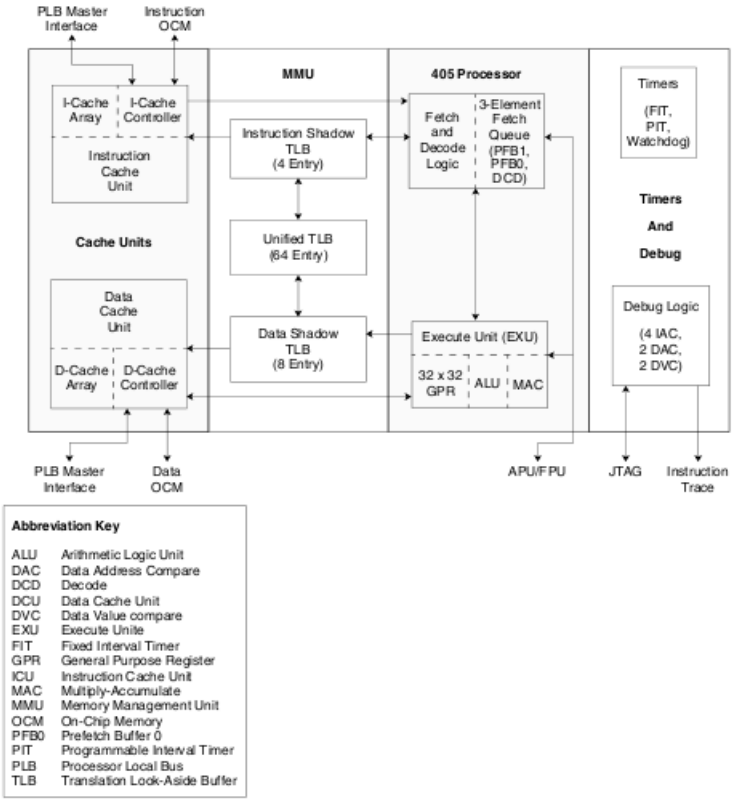

Figura 2.2: Organización del procesador *PowerPC 405-S*

2.3. Compilación y carga del *driver* en Linux

Un controlador de dispositivo, llamado normalmente *driver* es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware. Se puede esquematizar como un manual de instrucciones que le indica al sistema operativo, cómo debe controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el hardware.

Debido a que el software de controladores de dispositivos se ejecuta como parte del sistema operativo, con acceso sin restricciones a todo el equipo, resulta esencial que sólo se permitan los controladores de dispositivos autorizados.

Los *drivers* de la tarjeta de desarrollo NetFPGA se pueden obtener en la siguiente dirección web:

http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Releases

2.3.1. Instalación de dependencias en Debian GNU/Linux

Dependencia significa que un software necesita de otro para que funcione adecuadamente. En Linux es común que se necesiten herramientas o librerías para realizar un trabajo. Este problema se puede resolver, en parte, con programas que se encargan del software instalado y que tratan de resolver las dependencias con información proveída por personas encargadas de los paquetes.

En el caso de la tarjeta de desarrollo NetFPGA es necesaria la instalación de dependencias en el sistema huesped con el fin de poder hacer la compilación del código fuente del *driver* de forma correcta.

A continuación se listan las dependencias requeridas en Debian GNU/Linux 6.0.4:

- **libncurses5-dev**, este paquete contiene los archivos de cabecera, bibliotecas estáticas y los enlaces simbólicos que los desarrolladores que usan ncurses va a necesitar.

```
root@debian:~#apt-get install libncurses5-dev
```

- **libnet1-dev**, proporciona un marco portátil de paquete de red de bajo nivel escritura y manipulación. libnet incluye las interfaces portátiles de creación de paquetes en la capa IP y la capa de enlace, así como una gran cantidad de funcionalidad adicional.

```
root@debian:~#apt-get install libnet1-dev
```

- **libpcap-dev**, proporciona un marco portátil de bajo nivel red de monitoreo. Las aplicaciones incluyen estadísticas de la red recolección, supervisión de la seguridad, la depuración de la red, etc.

```
root@debian:~#apt-get install libpcap-dev
```

Compilación del *driver*

Se compila el *driver*:

```
root@debian:~#cd ~/netfpga/  make
```

Se instala el *driver*:

```
root@debian:~#cd ~/netfpga/  make install >> salida_install.
   txt
```

Se reinicia el sistema:

```
root@debian:~#reboot
```

Una vez que el sistema a reiniciado se verifica la carga del *driver*:

```
root@debian:~#lsmod  | grep nf2
nf2                    11925  0
```

Verificando la interfaces de la tarjeta de desarrollo NetFPGA

Para comprobar que las cuatro interfaces **nf2cX** se han cargado correctamente se utiliza el comando :

```
root@debian:~# ifconfig -a | grep nf2
nf2c0     Link encap:Ethernet  HWaddr 00:4e:46:32:43:00
nf2c1     Link encap:Ethernet  HWaddr 00:4e:46:32:43:01
nf2c2     Link encap:Ethernet  HWaddr 00:4e:46:32:43:02
nf2c3     Link encap:Ethernet  HWaddr 00:4e:46:32:43:03
```

2.3.2. Instalación del *driver* en CentOS

Para hacer sencilla la instalación de los *drivers*, se han creado paquetes *RPM*[4] y un repositorio de *YUM*[5]. Se muestra a continuación los pasos a seguir para la instalación de la tarjeta de desarrollo NetFPGA en sistemas CentOS y Fedora.

1. Ingresar al sistema como usuario *root*

```
[proyecto@CentOS ~]\$ su -
Password:
[root@CentOS ~]#
```

2. Instalar el repositorio *RPMforge*

```
[root@CentOS ~]#rpm --import http://apt.sw.be/RPM-GPG-
   KEY.dag.txt
```

3. Instalar el paquete descargado

```
[root@CentOS ~]#rpm -i rpmforge-release-0.5.2-2.el6.rf
   .*.rpm
```

4. Instalar el repositorio *YUM* NetFPGA

```
[root@CentOS ~]#rpm -Uhv
http://netfpga.org/yum/el5/RPMS/noarch/netfpga-repo-1-1
   _CentOS5.noarch.rpm
```

5. Instalar los paquetes base para la NetFPGA

[4] *RPM Package Manager* (o *RPM*, originalmente llamado *Red Hat Package Manager*) es una herramienta de administración de paquetes pensada básicamente para *GNU/Linux*. Es capaz de instalar, actualizar, desinstalar, verificar y solicitar programas. *RPM* es el formato de paquete de partida del *Linux Standard Base*.

[5] *Yellow dog Updater, Modified* es una herramienta libre de gestión de paquetes para sistemas *Linux* basados en *RPM*.

```
[root@CentOS ~]#yum -y install netfpga-base
```

- NOTA: Es necesario instalar algunas dependencias para el correcto funcionamiento de la tarjeta de desarrollo, YUM se encargará de instarlas automáticamente durante este proceso.

Verificando la interfaces de la tarjeta de desarrollo NetFPGA

Para comprobar que las cuatro interfaces **nf2cX** se han cargado correctamente se utiliza el comando :

```
root@CentOS:~# ifconfig -a | grep nf2
nf2c0     Link encap:Ethernet  HWaddr 00:4e:46:32:43:00
nf2c1     Link encap:Ethernet  HWaddr 00:4e:46:32:43:01
nf2c2     Link encap:Ethernet  HWaddr 00:4e:46:32:43:02
nf2c3     Link encap:Ethernet  HWaddr 00:4e:46:32:43:03
```

Reprogramando CPCI

Una vez instalada la tarjeta de desarrollo en el sistema huesped se debe de reprogramar el *bus CPCI* de la siguiente manera:

```
root@CentOS:~# /usr/local/sbin/cpci_reprogram.pl --all
```

Esperando la salida:

```
Loading the CPCI Reprogrammer on NetFPGA 0
Loading the CPCI on NetFPGA 0
CPCI on NetFPGA 0 has been successfully reprogrammed
```

Es necesario reprogramar el *CPCI* cada vez que se inicie el sistema huesped, esto se puede lograr de forma automatica agregando la siguiente liena al archivo */etc/rc.local*:

```
/usr/local/netfpga/lib/scripts/cpci_reprogram/cpci_reprogram.
   pl --all
```

Con el fin de familiarizarse con la tarjeta de desarrollo durante la realizacón del proyecto se realizaron varias pruebas con proyectos realizados anteriormente por la comunidad de desarrolladores, estas pruebas se encuentran documentadas en el *Apéndice A*.

Capítulo 3

Sistema Operativo

3.1. Introducción

Un sistema de cómputo está compuesto de uno o más procesadores, una memoria principal, discos, interfaces de red y otros dispositivos de entrada/salida. Al ser un sistema complejo es necesario contar con una capa de *software* llamada sistema operativo, cuya labor es administrar todos los dispositivos y proporcionar a los programas de usuario una interfaz sencilla para comunicarse con el *hardware*[15].

Un sistema operativo puede definirse entonces como un conjunto de programas, escrito en uno o más lenguajes de programación, utilizando diferentes paradigmas de programación:

- Lenguaje ensamblador dependiente de la arquitectura objetivo
- Lenguaje de mediano nivel no dependiente de la arquitectura
- Lenguajes interpretados o scripts, no dependiente de arquitectura objetivo

Su objetivo es proporcionar a los programas de usuario un modelo de computadora mejor, más simple y encargarse de la administración de todos los recursos.

El sistema operativo tiene como misión administrar todos los elementos de un sistema complejo, por tanto, el sistema operativo efectúa un reparto controlado de los procesadores, memorias y dispositivos de entrada/salida, entre los diversos programas que compiten por obtener estos recursos.

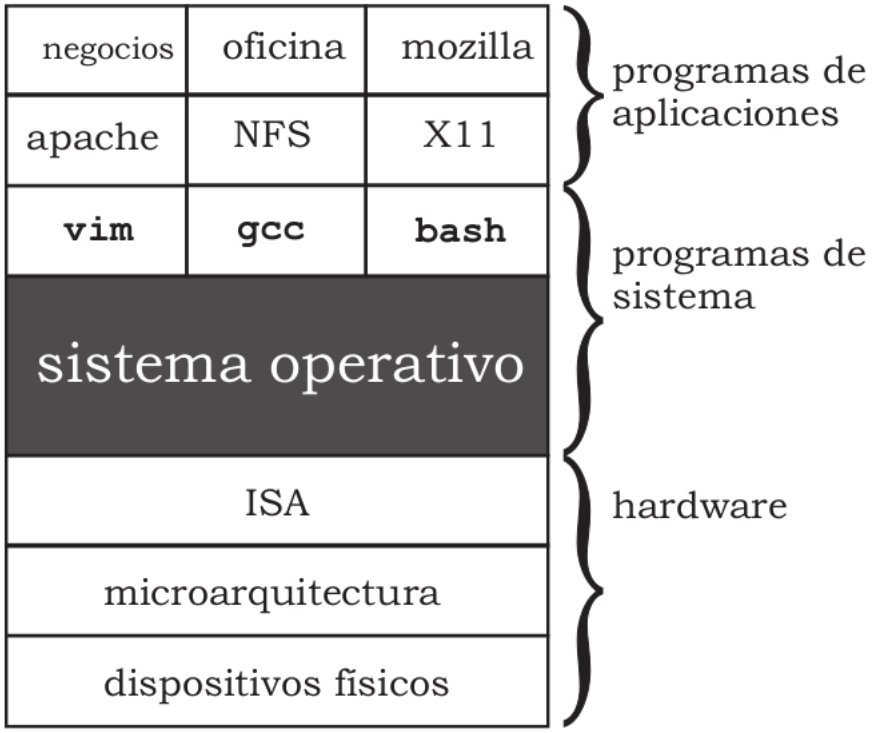

Figura 3.1: Sistema de cómputo en capas

El sistema operativo (SO) permite lanzar aplicaciones a través de procesos[1] o hilos[2] y vigilará que los recursos sean utilizados de forma equitativa entre los procesos.

Para poder realizar estas tareas es necesario tener una jerarquía de ejecución, esta jerarquía puede establecerse con:

- Procesos con privilegos limitados
- Procesos con privilegos otorgados temporalmente
- Procesos con mayores privilegios

Para lograr que el sistema operativo logre su objetivo, es necesario que el procesador tenga almenos dos modos de operación:

[1]Proceso, unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistemas asociados[15]

[2]Hilo, es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

- Modo *kernel*, o supervisor, permite accesar a todo el *hardware* y permite la ejecución de cualquier instrucción.

- Modo usuario, o modo real, solo permite la ejecución de un conunto reducido de instrucciones[13].

3.2. Linux

Linux tiene su origen en Unix, éste apareció en los años sesenta, desarrollado por los investigadores Dennis Ritchie y Ken Thompson, de los Laboratorios Telefónicos Bell.

Andrew Tanenbaum desarrolló un sistema operativo parecido a Unix (Minix) para enseñar a sus alumnos el diseño de un sistema operativo. Debido al enfoque docente de Minix, Tanenbaum nunca permitió que éste fuera modificado, ya que podrían introducirse complicaciones en el sistema para sus alumnos.

Un estudiante finlandés llamado Linus Torvalds, constatando que no era posible extender Minix, decidió escribir su propio sistema operativo compatible con Unix.

En aquellos momentos el proyecto GNU (*GNU's Not Unix*), que Richard Stallman había iniciado hacía ya casi diez años, comprendía un sistema básico casi completo. La excepción más importante era el *kernel* o núcleo, que controla el *hardware*.

Torvalds decidió aprovechar el sistema GNU y completarlo con su propio núcleo, que bautizó como Linux (*Linux Is Not UniX*).

El *kernel* es el componente central del sistema operativo. Su funciones son principalmente administrar el *hardware* de manera coherente y justa mientras se le otorga un nivel de abstracción familiar, a través de las APIs[3], a las aplicaciones de nivel de usuario.

Entre otras tareas relevantes de un sistema operativo, el *kernel* de Linux maneja dispositivos, administra los acceso de E/S, controla los procesos y administra el uso compartido de memoria. Dentro del *kernel*, la interfaz de bajo nivel es específica para cada configuración de hardware, sobre la cual, el *kernel* ejecuta y provee control

[3] API acrónimonimo de *Application Programming Interface*es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

directo de los recursos hardware.

Típicamente, los servicios de bajo nivel manejan operaciones específicas de el CPU, operaciones de memoria específicas a la arquitectura, y provee interfaces básicas para dispositivos. Los capa de alto nivel provee abstracciones comunes a todos los sistemas Unix, incluyendo procesos, archivos, sockets y señales. Este nivel de abstracción se mantiene constante aunque difiera el hardware.

Entre estos dos niveles de abstracción, el *kernel* necesita lo que se denomina componentes de interpretación para comprender e interactuar con datos estructurados provenientes de, o hacia ciertos dispositivos. Los diferentes tipos de sistemas de archivos y los protocolos de red son ejemplos de fuentes de datos estructurados. El *kernel* necesita interpretarlos e interactuar a fin de proveer acceso a los datos provenientes desde estas fuentes o hacia las mismas.

Los servicios brindados por el *kernel* no son soporte suficiente para cargar y ejecutar las aplicaciones. Es necesario contar con librerías, éstas proveen APIs familiares y abstracciones de servicios que interactúan con el *kernel* en nombre de las aplicaciones para obtener la funcionalidad deseada.

La librería principal, utilizada en la mayoría de las aplicaciones Linux, es la librería C GNU (**glibc**). Típicamente las librerías son enlazadas dinámicamente en el momento en el que se ejecutan las aplicaciones. Esto es, no son parte de las aplicaciones binarias, sino que se cargan dentro del espacio de memoria de las aplicaciones durante el inicio de las mismas. Esto permite a varias aplicaciones utilizar una misma instancia de una librería en vez de realizar una copia en memoria por cada aplicación que se ejecuta.

Según lo expuesto anteriormente es lógico pensar la conveniencia de enlazar dinámicamente las librerías, sin embargo, en los sistemas empotrados esto no es del todo cierto. El motivo radica en que las aplicaciones no utilizan la librería C en forma completa, sino que dependiendo de la aplicación puede utilizar partes de la librería y no otras. De este modo, en algunas aplicaciones parte de la librería se encuentra en la misma aplicación binaria. Este es el fundamento por el cual es preferible utilizar un enlazamiento estático, sin embargo nos encontramos con un inconveniente, para sistemas Linux empotrados la librería **glibc** consume demasiados recursos de la memoria RAM del sistema, por este motivo, reemplazar esta librería puede significar un

ahorro de espacio en memoria. Usualmente se la reemplaza por librerías alternativas diseñadas para sistemas empotrados.

3.2.1. El *kernel* de Linux

El *kernel* Linux es distribuido bajo la licencia GNU GPL por lo que su capacidad de evolución es una cualidad que posee desde su surgimiento, hecho por el cual su desarrollo es muy activo, brindando soporte para cientos de protocolos de red, decenas de arquitecturas de *hardware* y por supuesto, obteniendo un rendimiento eficiente y robusto[9].

Una clasificación generalmente utilizada para clasificar a los sistemas operativos tiene que ver con el modo de compartir el espacio de memoria. Se diferencian tres tipos, de tiempo real, monolíticos y microkernel. De forma resumida las características son las siguientes:

- *Realtime*, El espacio de direcciones es plano o lineal, no posee protección de memoria entre las aplicaciones y el *kernel*, es decir, el nucleo del *kernel*, el subsitema del *kernel* y las aplicaciones comparten el mismo espacio de memoria. Se denominan *Realtime* debido a que no hay sobrecarga por llamadas al sistema, pasaje de mensajes o copia de datos.

- Monolítico, está diferenciado el espacio de memoria de usuario y *kernel*. Las aplicaciones que operan en el espacio de usuario lo hacen sobre direcciones de memoria virtuales por lo tanto no pueden corromper la memoria de otras aplicaciones o del *kernel*. Sin embargo, los componentes del *kernel* comparten el mismo espacio de direcciones y por ende, un *drive* o módulo mal programado puede causar la inestabilidad del sistema. La mayor ía de os sistemas operativos Unix son de este tipo.

- Microkernel, hace uso de un pequeño SO que provee los servicios básicos y el resto del *kernel* se ejecuta como aplicaciones. La clave del microkernel surge a partir de un esquema robusto de paso de mensajes.

Cuando un programa es ejecutado en modo usuario este no puede acceder directamente a programas o estructuras de datos del *kernel*, sin embargo, cuando el

mismo se ejecuta en modo *kernel* estas restricciones no existen.

3.2.2. Sistema de archivos

El sistema de archivos es el encargado de realizar la organización y almacenamiento de los archivos en los diferentes dispositivos disponibles en el sistema. En función de las características del dispositivo de almacenamiento y del tipo de información que se va a guardar es preferible utilizar un sistema de archivos u otro.

Linux da soporte a varios sitemas de archivos, dentro de los más utilizados se encuentran ext2, ext3, etc. Estos sistemas de archivos son manejados por una capa denominada Sistema de Archivos Virtual (VFS[4]). Esta capa de abstracción provee una visión consistente de los datos almacenados en diferentes dispositivos del sistema. Esta visión es lograda separando el nivel de usuario de los sistemas de archivos, utilizando llamadas estándar al sistema, permitiendo sistemas de archivos lógicos sobre cualquier dispositivo físico. Por lo tanto esta capa abstrae los detalles físico del dispositivo permitiendo un acceso a los mismos a través de archivos de una maner consistente.

Por debajo de esta capa VFS, el *kernel* interactúa con dispositivos de E/S a través de controladores de dispositivos. Estos controladores se encuentran incluidos en el *kernel* y consisten en estructuras de datos y funciones que controlan uno o más dispositivos como discos rígidos, teclados, mouses, monitores, interfaces de red, dispositivos SCSI.

Uno de los propósitos fundamentales de los controladores de dispositivos es aislar los programas de usuario del acceso a estructuras de datos críticas del *kernel* y dispositivos de *hardware*. Además, un controlador de dispositivo oculta al usuario la complejidad y variabilidad de un dispositivo *hardware*. Por ejemplo, un programa que quiere escribir datos en un disco rígido no tiene en cuenta si el mismo posee sectores de 512 bytes o 1024 bytes. El usuario simplemente abre el archivo y realiza

[4]VFS, es una capa de abstracción encima de un sistema de archivos más concreto. El propósito de un VFS es permitir que las aplicaciones cliente tengan acceso a diversos tipos de sistemas de archivos concretos de una manera uniforme. Puede ser utilizada para tender un puente sobre las diferencias en los sistemas de archivos de Windows, de Mac OS y Unix, de modo que las aplicaciones pudieran tener acceso a archivos en los sistemas de archivos locales de esos tipos sin tener que saber a qué tipo de sistema de archivos están teniendo acceso.

el comando de escritura. El controlador manejará los detalles y aislará al usuario de las complejidades y riesgos de programar directamente sobre el dispositivo de *hardware*. Estos controladores proveen la representación de los dispositivos a través de archivos, en GNU/Linux y sistemas operativos Unix todo *hardware* es representado por un archivo.

Linux posee la capacidad de agregar y quitar componentes del *kernel* en tiempo de ejecución. Como hemos descripto anteriormente, el *kernel* Linux posee una estructura de *kernel* monolítico, con una interfaz para agregar y quitar módulos de controladores de dispositivos dinámicamente luego del arranque del mismo. Esta característica no solo agrega flexibilidad al usuario, sino que además, en sistemas empotrados adquiere una especial importancia debido a su capacidad de actualización y adaptación a dispositivos de E/S nuevos[4].

Todo dispositivo, ya sea que se encuentre en un sistema empotrado o una PC de escritorio, necesita al menos un sistema de archivos. La principales razones de esto son:

- Las aplicaciones poseen programas separados, independientes por ende necesitan espacio de almacenamiento en un sistema de archivos.
- Los dispositivos de bajo nivel también son accedidos mediante archivos.

3.2.3. *Embedded Linux*

Linux empotrado, del inglés: *Embedded Linux* se refiere al uso de un sistema operativo basado en Linux en un sistema empotrado, como por ejemplo teléfonos móviles, robots, servidores, dispositivos electrónicos y aplicaciones industriales con microcontroladores y microprocesadores[3].

Para implementar un sistema Linux empotrado en un dispositivo de *hardware*, se debe conocer en términos generales la arquitectura del mismo, es decir, el tipo de micro-procesador que posee, la cantidad de memoria, los buses que soporta, los componentes que posee la tarjeta de desarrollo, etc. Esta información es de vital importancia, ya que al preparar el sistema Linux que se ejecutará en el dispositivo, se debe compilar con soporte para esas características. La tarea de construir un

sistema operativo basado en Linux que se ejecute en este dispositivo objetivo se debe realizar mediante compilación cruzada.

Capítulo 4

Compilación del *kernel*

4.1. Introducción

La compilación se refiere al proceso de traducción de un lenguaje de alto nivel a otro funcionalmente equivalente, expresado en lenguaje ensamblador de una arquitectura específica[10]. Durante este capítulo se explica la técnica para crear una toolchain que genere binarios para la plataforma PPC405.

4.2. El proceso de compilación

El proceso de compilación es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje máquina. Además de un traductor, se pueden necesitar otros programas para crear un programa objeto ejecutable. Un programa fuente se puede dividir en módulos almacenados en archivos distintos. La tarea de reunir el programa fuente a menudo se confía a un programa distinto, llamado preprocesador. El preprocesador también puede expandir abreviaturas, llamadas a macros, a proposiciones del lenguaje fuente.

Este proceso se compone internamente de varias etapas mostradas a continuación:

- Análisis léxico, se encarga de la reducción del texto del programa en *tokens*:
  - identificadores
  - separadores operadores

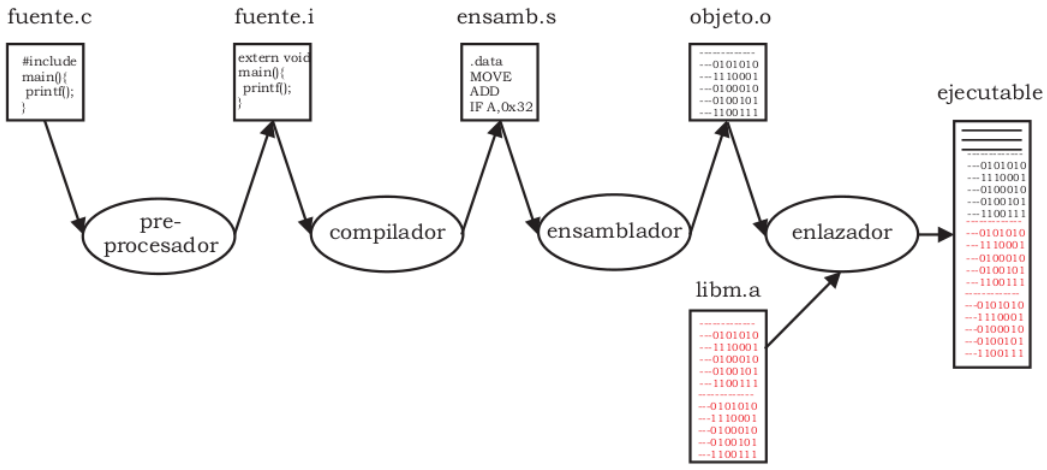

Figura 4.1: El proceso de compilación

  - constantes

- Análisis sintáctico, se encarga del análisis de símbolos para reconocer la estructura de programa:

  - indentificador = expresión
  - indentificador + constante

- Análisis semántico, realiza la asociación de identificadores con zonas de memoria y la asociación de tipos de datos.

- Generación de código, Asocia las sentencias con secuencias de instrucciones.

- Optimización de código, consiste en mejorar el código intermedio, de modo que resulte un código máquina más rápido de ejecutar.

4.3. Compilación cruzada

Si un compilador es capaz de compilar un programa para otra arquitectura en la cual se está ejecutando, se dice que es un compilador cruzado. En este proceso se identifica al equipo que realiza la compilación mediante el término *Host* o Huesped y al dispositivo que ejecuta el software, como sistema *Target* u Objetivo[17].

4.3.1. Sistema huesped

La implementación de un entorno de compilación cruzada nos brinda la posibilidad de aprovechar los recursos que disponemos en una PC. Esta tarea se ha llevado a cabo sobre una PC de escritorio con las siguientes características:

- Procesador Intel(R) Pentium(R) 4 CPU 3.00GHz
- Memoria 764MB
- Sistema Operativo Debian GNU/Linux 6.0.4
- Kernel Linux 2.6.32-5-686 (i686)

4.3.2. Sistema objetivo

El sistema objetivo es la tarjeta de desarrollo NetFPGA, dentro de ella se cuenta con un FPGA Xilinx Virtex-II Pro 50 con las siguientes características:

- Dos procesadores PowerPC 405 (PPC405) a 300Mhz con 16kB para datos y 16kB para instrucciones en su memoria cache
- 8 Mb de bloques RAM de doble vía
- 20 Entradas-Salidas de alta velocidad (RocketIO), con una velocidad máxima de operación de 3.125 Gbps
- 444 multiplicadores de 18 x 18 bits
- 12 relojes para control de bloques
- 1040 puertos de entrada-salida de usuario

Estas características proporcionan un gran número de posibilidades para el desarrollo de aplicaciones y, puesto que existen hoy en día herramientas de software que ayudan a la programación, compilación, sıntesis, simulación y depuración tanto de hardware como de software, se obtiene una alta flexibilidad de desarrollo, permitiendo a los usuarios centrarse en el diseño y tomando la responsabilidad de dicho diseño para obtener el máximo provecho de los recursos[16].

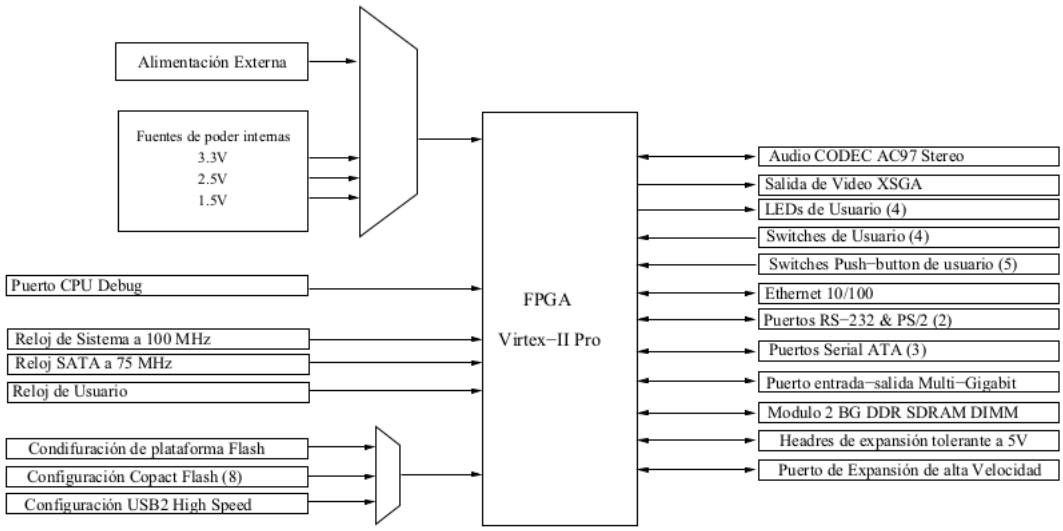

Figura 4.2: Diagrama a bloques de la tarjeta Virtex-II Pro 50

4.3.3. GNU toolchain

GNU toolchain es un término que agrupa a una serie de proyectos que contienen las herramientas de programación producidas por el proyecto GNU. Estos proyectos forman un sistema integrado que es usado para programar tanto aplicaciones como sistemas operativos.

El GNU toolchain es un componente vital en el desarrollo del núcleo Linux, el desarrollo del BSD y software para sistemas empotrados.

Apéndice A

Familiarización con la tarjeta de desarrollo NetFPGA

A.1. Prueba de autocomprobación

El *Selftest* es un bitfile que garantiza que todos los componentes de la plataforma de desarrollo son totalmente funcionales. El autodiagnóstico se compone tanto de un bitfile que contiene la lógica y las interfaces de los componentes externos, como de un software que muestra los resultados. El *selftest* prueba todo el hardware en paralelo. La prueba continúa funcionando en repetidas ocasiones hasta que sea terminado por el usuario.

El bitfile realiza pruebas rigurosas de la *SRAM* y *DRAM DDR2* para asegurarse de que todas las líneas de la memoria pueden ser adecuadamente escritas y leídas . En la prueba de red se envían ráfagas de paquetes en las interfaces Ethernet y los cables de *loopback* se ponen en marcha para que los paquetes pueden ser leídos y en comparación con los datos que se transmiten.

La prueba de *loopback SATA* transmite los datos mediante las lienas de entrada y salida Multi-Gigabit líneas (MGIOs) para asegurar que los datos pueden ser transmitidos de forma fiable en la alta velocidad de interfaces de entrada y salida .

La prueba de DMA ejerce el controlador *PCI (CPCI), VirtexII*, y el bus *PCI* para asegurarse de que grandes bloques de datos pueden ser enviados entre la tarjeta de desarrollo NetFPGA y memoria de la computadora huesped.

Las siguientes instrucciones asumen que se ha instalado con éxito una tarjeta de desarrollo NetFPGA con CentOS.[2].

Bibliografía

[1] C. Adam. *Introduction Guide*. Stanford University, 2006. URL http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide#Introduction.

[2] C. Adam. *VerifyHardwareAndSoftware*. Stanford University, 2011. URL http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/VerifyHardwareAndSoftware.

[3] American Arium. *Embedded Linux Debug Tools, A New Paradigm*. RTC magazine, 2003. URL http://www.rtcmagazine.com/articles/view/100131.

[4] D.P. Bovet y M. Cesatí. *Understanding The Linux Kernel*. O'Reilly Series. O'Reilly, 2005. ISBN 9780596005658.

[5] Altera Corporation. *What is an FPGA?* Altera Corporation, 2012. URL http://www.altera.com/products/fpga.html.

[6] R. Droms. *Dynamic Host Configuration Protocol*. IEEE, 1997. URL http://www.ietf.org/rfc/rfc2131.txt.

[7] Linux Foundation. *What Is Linux: An Overview of the Linux Operating System*. Linux Foundation, 2009. URL https://www.linux.com/learn/resource-center/376-linux-is-everywhere-an-overview-of-the-linux-operating-system.

[8] N. Freed. *Behavior of and Requirements for Internet Firewalls*. IEEE, 2000. URL http://www.ietf.org/rfc/rfc2979.txt.

[9] Emiliano P. López. *Estudio e Implementación de un Router MIPS Mediante Linux Embebido*. Proyecto Fin de Carrera, Universdad Nacional del Litoral, 2007. URL http://linuxemb.wikidot.com/tesis.

[10] Oscar Alvarado Nava. Proceso de compilación. Notas del curso Arquitectura de Computadoras.

[11] Oscar Alvarado Nava. *Implementación en FPGAs de Algoritmos de Compresión-Descompresión para Dispositivos Móviles*. Proyecto Fin de Carrera, CINVESTAV, 2007.

[12] NetFPGA. *NetFPGA Technical Specifications*. LRD Group, 2012. URL http://www.netfpga.org/php/specs.php.

[13] J.Ñiño y J.N. Camazón. *Sistemas operativos monopuesto*. Ciclos Formativos. Editorial Editex, 2011. ISBN 9788497719711.

[14] OpenSSH. *OpenSSH 6 has just been released*. OpenSSH, 2012. URL http://www.openssh.com/txt/release-6.0.

[15] A.S. Tanenbaum y R.E. García. *Sistemas Operativos Modernos*. Pearson Educación. Pearson Educación, 2003. ISBN 9789702603153.

[16] Usiel Alvarado Villafranca. *Sistema para la Programación de un FPGA Mediante Interfaz Gráfica de Usuario en Linux*. Proyecto Fin de Carrera, UAM Unidad Azcapotzalco, 2010.

[17] K. Yaghmour, J. Masters, G. Ben-Yossef, y P. Gerum. *Building Embedded Linux Systems*. O'Reilly Series. O'Reilly, 2008. ISBN 9780596529680.