DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: SISTEMAS EMBEBIDOS.

LUIS CARLOS UYABÁN BECERRA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
BUCARAMANGA
2012

DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: SISTEMAS EMBEBIDOS.

LUIS CARLOS UYABÁN BECERRA

Trabajo de grado para optar al título de Ingeniero Electrónico

Director:
MSc. Jorge H. Ramón Suarez

Codirectores:
MIE. William A. Salamanca B.
MIE(c). Carlos A. Angulo J.
MIE. Carlos A. Fajardo A.
MIE. Sergio A. Abreo C.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
BUCARAMANGA
2012

Dedico este logro a Dios por todo lo bello que me ha brindado, por haberme permitido llegar hasta este punto y por brindarme las herramientas espirituales necesarias para lograrlo. A mis padres Elisa y Santos por todo el amor, enseñanzas, dedicación, entrega y confianza que ha tenido para conmigo. A mi familia que es uno de los grandes pilares de mi vida. Andrea por todo ese apoyo, constancia y amor, que me han permitido no desfallecer y luchar por todo aquello que anhelo. A todos mis amigos, compañeros y a aquellas personas que hicieron valiosos aportes a este proyecto.

AGRADECIMIENTOS

A todas aquellas que permitieron directa e indirectamente la realización de este proyecto. A mi director de proyecto por haberme brindado la oportunidad de trabajar en este proyecto, a mis codirectores que me brindaron su atención, compartieron su tiempo y conocimiento ayudándome a lograr las metas propuestas.
A todos aquellos profesores que participaron en mi proceso de formación. A mis compañeros de los otros proyectos asociados por brindarme los aportes y sugerencias para que este trabajo fuera validado en busca de mejoras. Y todos aquellos compañeros que me acompañaron durante todo proceso.

RESUMEN

TÍTULO: DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE. ASIGNATURA: SISTEMAS EMBEBIDOS[*]

AUTOR: LUIS CARLOS UYABÁN BECERRA[**]

PALABRAS CLAVES: Tarjeta SIE, *Drivers*, Interfaz Gráfica de Usuario y Guías de Laboratorio.

Actualmente se desea incluir en el pensum de Ingeniería electrónica de la E3T la asignatura llamada Sistemas Embebidos. Esta asignatura culminará el proceso de aprendizaje del área de electrónica digital que imparte actualmente la E3T. El desarrollo práctico de la asignatura se basa en el manejo de un sistema de desarrollo llamado SIE el cual cuenta con todos los elementos tanto de software como de hardware para el desarrollo de la asignatura. Este proyecto tiene como objetivo desarrollar una serie de guías prácticas que permitan dar una orientación clara y objetiva sobre los temas teóricos suministrados en la asignatura, y sobre el uso de la tarjeta SIE en la misma. La selección de los temas se realizó teniendo en cuenta el proceso de aprendizaje que se ha venido desarrollando en las asignaturas de Sistemas Digitales, Arquitectura de computadores y procesadores que culminaría brindando los conceptos finales necesarios para que los estudiantes puedan concebir, diseñar, implementar y operar sistemas digitales avanzados.

El desarrollo del proyecto dió como resultado la elaboración e implementación de 10 guías prácticas con temáticas orientadas al desarrollo de drivers y construcción de interfaces gráficas de usuario, que se encuentran consignadas en un documento llamado guías de laboratorio Sistemas embebidos. Además se incluye los requerimientos previos de software y hardware que se deben cumplir y pasos a ejecutar que permiten la realización exitosa de las guías. Las guías quedarán a disposición de los profesores del área de electrónica digital de la E3T para su implementación. Finalmente se incluyen recomendaciones que se deben tener en cuenta al momento de realizar las guías.

[*] Trabajo de grado

[**] **Facultad:** Facultad de Ingenierías Físico Mecánicas, **Escuela:** Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. **Grupo de Investigación:** CPS. **Director:** MSc. Jorge H. Ramón Suarez. **Codirectores:** MIE. William A. Salamanca B., MIE(c). Carlos A. Angulo J., MIE. Carlos A. Fajardo A., MIE. Sergio A. Abreo C.

ABSTRACT

TITLE: DEVELOPMENT OF EDUCATIONAL MATERIAL FOR THE AREA OF DIGITAL SYSTEMS BASED ON THE PLATFORM SIE. SUBJECT: EMBEDDED SYSTEMS [*]

AUTHOR: LUIS CARLOS UYABÁN BECERRA[**]

KEY WORDS: SIE Board , Drivers, Graphical User Interface and Laboratory Guides.

Nowadays the E3T wants to include in the curriculum the Electronic Engineering the subject called Embedded Systems. This course has the purpose culminated a process of learning in the digital system area at the E3T school. The practical development of the course is based on the management of development system called SIE that has all the necessary elements of software and hardware to get the objective of this course. This project aims to develop a serie of practical guides that allow for a clear and objective information provided theoretical issues in the subject, and the use of the SIE card for it. The topics were select taking into account the sequential process of learning that is handled in the field of digital system, providing the final concepts ne-cessary to the students can conceive, design, implement and operate advanced digital systems.

The development of the project concluded in the preparation and implementation of 10 guides with topics designed to develop drivers and building graphical user interfaces, which are contained in a document called Embedded Systems Laboratory guides. It also includes the prere-quisites for software and hardware that must be obtained in order to are going to be execute the successful implementation of the guidelines. The guidelines available to the professors of digital system area at the E3T for a future implementation.

[*] Degree Project

[**] **Faculty:** Physico-Mechanical Engineering Faculty, **School:** School of Electrical, Electronics and Telecommunications Engineering. **Research Group:** CPS. **Director:** MSc. Jorge H. Ramón Suarez. **Codirectors:** MIE. William A. Salamanca B., MIE(c). Carlos A. Angulo J., MIE. Carlos A. Fajardo A., MIE. Sergio A. Abreo C.

CONTENIDO

pág

INTRODUCCIÓN 14

1. OBJETIVOS 17

1.1. OBJETIVO GENERAL 17

1.2. OBJETIVOS ESPECÍFICOS 17

2. GENERALIDADES 18

2.1. SISTEMA OPERATIVO 18

2.1.1. El sistema operativo Linux 19

2.1.2. Linux Embebido 21

2.1.3. OpenWrt 22

2.2. EL *KERNEL* 23

2.3. *DRIVERS* 25

2.4. INTERFAZ GRÁFICA DE USUARIO 26

2.4.1. Qt 27

2.4.2. Qt/Linux Embebido 27

3. MANUAL DE USUARIO DE LA TARJETA SIE 29

3.1. COMANDOS 29

3.2. INSTALACIÓN DE LAS HERRAMIENTAS DE COMPILACIÓN PARA LA TARJETA SIE 29

3.3. "*FLASHEO*" DEL *KERNEL* Y DEL SISTEMA DE ARCHIVOS RAÍZ . . 30

3.4. CONFIGURACIÓN DE LAS HERRAMIENTAS DE COMPILACIÓN PARA LA CREACIÓN DE APLICACIONES EN QT 30

3.5. CONCEPTOS Y FUNCIONES ESPECIALES PARA LA CREACIÓN DE DRIVERS 31

3.6. CONCEPTOS Y MÉTODOS ESPECIALES PARA LA CREACIÓN DE INTERFACES GRÁFICAS DE USUARIO EN QT 31

4. GUÍAS DE LABORATORIO 32

4.1. GUÍAS DE LABORATORIO DE DRIVERS 32

4.1.1. Primer Módulo 33

4.1.2. Números mayores y menores 34

4.1.3. Estructuras de datos en el Kernel 35

4.1.4. Operaciones de Archivo 35

4.2. GUÍAS DE LABORATORIO PARA Qt 36

4.2.1. Primera aplicación en Qt 36

4.2.2. Ubicación de los *widgets* 37

4.2.3. *Signals* y *slots* 38

5. VALIDACIÓN 39

CONCLUSIONES 41

BIBLIOGRAFÍA 44

LISTA DE TABLAS

pág

1 Funciones del *Kernel* 24

LISTA DE ANEXOS

pág

ANEXO A. MANUAL DE USUARIO DE LA TARJETA SIE: SISTEMAS EMBEBIDOS. 45

ANEXO B. GUÍAS DE LABORATORIO SISTEMAS EMBEBIDOS. 45

INTRODUCCIÓN

Desde hace ya hace varios años la arquitectura de los computadores personales se ha extendido a dispositivos presentes en sistemas de control de máquinas, sistemas de comunicación, automóviles, aparatos electrónicos domésticos, equipos militares, consolas de vídeo juegos, dispositivos médicos, entre otros. Muchos de estos elementos son usados diariamente por los seres humanos para suplir necesidades, para divertirse y como instrumentos de trabajo utilizados para poder desempeñar su labor, sin darse cuenta que en muchos de ellos existe todo un complejo sistema conformado por *software* y *hardware* que permite su funcionamiento. Estos sistemas son llamados Sistemas Embebidos y surgen de diseños desarrollados inicialmente para computadores personales. La diferencia principal entre los sistemas embebidos y un computador personal radica en que los sistemas embebidos no son diseñados para realizar tareas multipropósito, sino para realizar un rango de tareas específicas. Por tal razón estos sistemas poseen dimensiones físicas muy reducidas en comparación con los computadores personales.

Los sistemas embebidos poseen núcleos de procesamiento que pueden ser microcontroladores o procesadores pequeños, además de una serie de componentes que sirven como medio de comunicación con el mundo exterior, memorias y otros elementos que ayudan al núcleo a cumplir la tarea que le fue asignada. Un sistema operativo sencillo o complejo puede ser usado para administrar todos los recursos disponibles tanto de hardware y software con los que cuenta el sistema embebido. Algunos de los sistemas embebidos están provistos de teclados y pantallas que permiten el ingreso de datos y brindan una interfaz amigable con el usuario.

La tarjeta SIE es un sistema de desarrollo creado por el profesor Carlos Camargo de la Universidad Nacional de Colombia que sirve como herramienta práctica para

el diseño, implementación y desarrollo de sistemas embebidos que se requieran en algún ambiente de trabajo específico. Un conjunto de componentes de hardware dentro de la misma placa ofrecen herramientas para construir un diseño específico. La tarjeta posee un procesador de tipo MIPS[1] es administrado por un tipo especial de Linux[2] para sistemas embebidos llamado Linux Embebido utilizado en diferentes sistemas que involucran control de procesos, *routers*, consolas de video juegos, entre otros. El procesador posee soporte para una biblioteca multiplataforma llamada Qt que permite realizar GUI[3] con el uso de una pantalla que puede ser conectada a la tarjeta SIE.

Actualmente se desea incluir en el pensum de la carrera profesional Ingeniería Electrónica la asignatura llamada Sistemas Embebidos, para estar a nivel de muchas Universidades que ya cuentan con asignaturas que cubren los temas más importantes involucrados en el desarrollo de Sistema Embebidos. El proceso comienza desde la primera materia ofrecida de la E3T[4] llamada Sistemas Digitales y culminará con el desarrollo de la asignatura propuesta que presentará los conceptos más importantes involucrados con el desarrollo de *drives* y elaboración de interfaces gráficas de usuario.

Este proyecto va dirigido a aquellos estudiantes que incluyan la asignatura Sistemas Embebidos y deseen complementar su formación profesional, adquiriendo conocimientos sobre el desarrollo de *drivers* e interfaces gráficas de usuario en un Sistema Embebido mediante el uso de la tarjeta SIE. Permitirá establecer un soporte práctico para profesores y directos encargados de la asignatura Sistemas Embebidos ofrecida como Electiva técnica profesional en la E3T.

Este documento describe de forma general los elementos utilizados y actividades

[1] Microprocessor without Interlocked Pipeline Stages (Microprocesador sin bloqueos en las etapas de segmentación)

[2] Sistema operativo de código abierto.

[3] Interfaces gráficas de usuario

[4] Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones

desarrolladas durante la ejecución del proyecto. Como primera parte se presentan las generalidades del mismo, tales como la conceptualización básica y los requerimientos mínimos para su desarrollo. La segunda parte muestra la información referente al manual de usuario y las guías desarrolladas presentadas en los anexos, la información concerniente al proceso de validación de las mismas y finalmente, conclusiones, recomendaciones y anexos.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

Desarrollar material didáctico para la asignatura: Sistemas Embebidos basados en la plataforma SIE.

1.2. OBJETIVOS ESPECÍFICOS

- Elaborar un manual de usuario avanzado para el sistema de desarrollo SIE, orientando su aplicación a la asignatura Sistemas Embebidos.
- Diseñar, implementar y validar prácticas de laboratorio para la asignatura Sistemas Embebidos basadas en el desarrollo de drivers básicos para la plataforma SIE .
- Diseñar, implementar y validar prácticas de laboratorio para la asignatura Sistemas Embebidos basadas en la creación de Interfaces Gráficas de Usuario (GUI) para la plataforma SIE.

2. GENERALIDADES

Algunos Sistemas Embebidos poseen procesadores con sistemas operativos desarrollados especialmente para su arquitectura. Estos sistemas operativos cumplen con labores de administración y control de los recursos de *hardware* y *software* que estén presentes en el sistema. Para el caso de la tarjeta SIE el sistema por defecto es una versión de Linux para sistemas Embebidos llamado Linux Embebido por lo tanto hace falta explorar y comprender algunas de las tareas de administración que cumple este sistema operativo y dar paso a uno de los temas principales de este proyecto: el desarrollo de *drivers*.

Además del *hardware* de la tarjeta, una interfaz gráfica de usuario puede ser implementada para permitir a un usuario interactuar con el sistema por medio de elementos visuales mostrados en una pantalla que puede ser conectada a la tarjeta por consiguiente es necesario entender las herramientas necesarias para la creación de una interfaz gráfica para aquellos sistemas digitales elaborados en la tarjeta SIE.

A continuación se presenta una visión general acerca de los aspectos necesarios para la fundamentación teórica que soporta el desarrollo de este proyecto. Cabe resaltar que cada guía cuenta con la debida justificación teórica que permite el entendimiento y desarrollo de cada uno de los procedimientos allí estipulados, necesarios para lograr cada uno de los objetivos planteados en las mismas.

2.1. SISTEMA OPERATIVO

Para muchos autores no es muy claro dar a entender el concepto definitivo concerniente a lo que es un sistema operativo, pero muchos de ellos coinciden en

que desde el punto de vista de software es el encargado de administrar todos los recursos de un sistema, tanto de aplicaciones informáticas como de *hardware*[3]. Un sistema está conformado por una serie de componentes electrónicos comúnmente llamados el *hardware* y un conjunto de programas informáticos que permiten enviar instrucciones al *hardware* en busca de la realización de algún trabajo específico. Un usuario interactúa con el sistema en busca de una respuesta que pueda suplir una necesidad determinada. Pero tanto el *hardware* como el software del sistema, hasta este punto se encuentran aislados totalmente del usuario, así que debe existir un medio por el cual dicho usuario pueda interactuar con el sistema. Este medio o interfaz es suministrada por el sistema operativo. El sistema operativo se encarga de administrar todos los recursos del sistema tanto de *software* y de *hardware*, además de ofrecer una interfaz para que el usuario pueda comunicarse con el sistema. Su unidad central es un núcleo programa principal del sistema operativo. En la actualidad existen muchos sistemas operativos con características funcionales diferentes, algunos de ellos se derivan de otros creados previamente[13].

2.1.1. El sistema operativo Linux

Es un sistema operativo basado en Unix[1] que surgió de la unión de un núcleo llamado *kernel* y un conjunto de herramientas de sistema llamado GNU[2], que funciona bajo la premisa de *software* libre[3]. Fue desarrollado originalmente por Linus Torvalds en la University of Helsinki, en Finlandia. Actualmente se sigue desarrollando con la colaboración de miles de programadores de todo el mundo a través de la Internet[13] medio por el cual es distribuido de forma gratuita.

Dentro de sus características funcionales, Linux es un sistema potente y robusto[15].

[1] Sistema operativo desarrollado en 1969 por Ken Thompson, Dennis Ritchie junto con investigadores de AT&T Laboratories

[2] GNU is not Unix, proyecto iniciado por Free Software Fundation que ofrece aplicaciones de software libre

[3] Aplicaciones que pueden ser utilizadas, modificadas y distribuidas de forma libre

Posee interfaces gráficas de usuario flexibles y poderosas. Aunque muchas personas piensen que Linux no es tan eficiente porque es gratuito, es considerado por muchos un sistema estable y eficiente. Es el sistema operativo preferido para servidores de red[13]. Una de sus características especiales es que es un sistema multitarea o sea que permite realizar varias tareas al mismo tiempo y multiusuario en donde varios usuarios pueden iniciar sesión en el sistema al mismo tiempo. Está equipado con un conjunto de utilidades para desarrollo de programas: compiladores y depuradores de C++ y C.

Linux posee tres componentes fundamentales: el *Kernel*, el entorno y la estructura de archivos[13]. El *Kernel* es el programa central del sistema que se encarga de ejecutar programas y administrar los dispositivos de *hardware*, el entorno proporciona una interfaz para el usuario (la cual puede ser gráfica) y la estructura de archivos que organiza la manera como se almacenan los archivos en un dispositivo de almacenamiento. Los archivos se organizan en directorios, algunos de ellos son reservados para uso estándar del sistema puesto que cada objeto en el sistema es visible como un archivo, incluidos los dispositivos.

Dentro del sistema generalmente se usa los términos espacio del *Kernel* y espacio de usuario para representar cada uno de los entornos que se ven involucrados dentro del mismo. El espacio de usuario designa todas aquellas aplicaciones en donde el usuario participa de una u otra forma y el espacio del *Kernel* provee el medio por el cual el espacio de usuario interactúa con el hardware del sistema.

Debido al gran número y la variedad de herramientas que ofrece el *kernel* que permiten un entorno informático funcional, crear una instalación de Linux desde cero es bastante complicado por lo tanto a través de los años han surgido paquetes[4] comerciales y no comerciales llamados distribuciones[13] dando pie a que diferentes compañías y grupos elaboren sus propias distribuciones de Linux. Al-

[4]Los paquetes en el entorno de Linux son archivos empaquetados que contienen todo el código fuente de una aplicación de *software*

gunas de las distribuciones mas comunes son Red HAt, Fedora, Debian, Ubuntu, etc, cada una de ellas proporciona soluciones a un amplio conjunto de tareas dependiendo de las necesidades de cada usuario utilizando el mismo *kernel*, aunque puede estar configurado de otra forma. Para el caso de sistemas embebidos existe un Linux especial llamado Linux embebido.

2.1.2. Linux Embebido

Es un tipo especial de Linux implementado en un sistema embebido. Este tipo especial de distribución de Linux no necesita de un *Kernel* especial, las versiones oficiales del Kernel son usadas para construir estos sistemas, sin embargo, un *Kernel* utilizado en un sistema embebido se diferencia de uno utilizado para un computador personal por la construcción de su configuración. Un sistema con Linux embebido es un sistema embebido basado en el *Kernel* de Linux, pero esto no implica el uso de una biblioteca específica o herramientas de usuario que vienen con el *Kernel* por defecto[6].

Una distribución de Linux embebido puede incluir: un marco de desarrollo para sistemas con Linux embebido, las aplicaciones de *software* a la medida para el uso de un sistema embebido o ambas cosas[6]. Dentro del marco de desarrollo podemos encontrar varias herramientas que faciliten el progreso de los sistemas embebidos. Algunas de estas herramientas pueden ser: compiladores cruzados, depuradores, software para la gestión de proyectos, entre otros. Estas distribuciones están diseñadas para ser instaladas en un computador personal que comúnmente es llamado computador de desarrollo y servirá para elaborar aplicaciones para el sistema embebido.

El computador de desarrollo puede funcionar con un sistema operativo diferente a Linux pero es recomendable usar Linux ya que se debe estar familiarizado con el entorno de Linux. Además de funcionar con Linux este computador de desarrollo debe contener una serie de aplicaciones que le permita desarrollar aplicaciones para el sistema embebido. Para elaborar dichas aplicaciones se necesitan com-

piladores, enlazadoras, intérpretes, entornos de desarrollo integrados y otras herramientas específicas para la arquitectura del sistema embebido. Cabe mencionar que el computador de desarrollo posee dichas herramientas de desarrollo, pero son propias de su arquitectura y por lo tanto no pueden ser usadas para el sistema embebido. Estas herramientas de desarrollo contienen una serie de programas de desarrollo cruzado que permiten crear una aplicación en una plataforma diferente a donde se ejecutará, es decir, en el computador de desarrollo se crea al aplicación y esta se ejecuta en el sistema embebido.

La conexión física entre el sistema embebido y el computador de desarrollo se realiza por medio de un cable que une a los dos sistemas. La comunicación por este cable puede ser serial o una conexión a Ethernet[6]. El sistema embebido posee su propio dispositivo de almacenamiento en donde se encuentra guardado un gestor de arranque, un *Kernel* funcional y un sistema de archivo raíz, elementos esenciales para que el sistema funcione.

2.1.3. OpenWrt

Es una distribución derivada de Debian basada en un *firmware*[1] diseñada para dispositivos embebidos tales como routers[2]. Sus siglas significan Open Wireless Real Time. Inicialmente fue utilizado para el modelo Linkys WRT54G[3] pero debido a su rápida expansión se ha incluido soporte para otros dispositivos[2], como el sistema de desarrollo SIE. Todo el soporte se realiza por medio de red. OpenWrt proporciona un sistema de archivos totalmente modificable que le permite escoger aplicaciones y seleccionar la configuración del dispositivo para que se adapte a las necesidades particulares del usuario[1].

---

[1] Conjunto de instrucciones a nivel máquina grabado en una memoria de solo lectura que controla los circuitos eléctricos de un dispositivo

[2] Es un dispositivo de *hardware* usado para la interconexión de redes informáticas que permite asegurar el direccionamiento de paquetes de datos entre ellas o determinar la mejor ruta que deben tomar.

[3] Router inalámbrico fabricado por la empresa Linksys.

2.2. EL *KERNEL*

El *Kernel* en Linux es el corazón del sistema operativo. Administra el *hardware* de una manera coherente y proporciona abstracciones amigables de alto nivel a los programas del espacio de usuario[9] también se encarga de controlar los dispositivos, los accesos de entrada/salida al sistema, los procesos internos del sistema, los accesos a memoria, la distribución de señales[4] y otras funciones administrativas. Proporciona API s[5] portables entre las diferentes arquitecturas, que permiten el uso de aplicaciones de modo uniforme en cada una de ellas.

En el sistema una serie de procesos especiales se ocupan de diferentes actividades, en donde cada proceso solicita recursos del sistema, ya sea memoria, conectividad a red o cualquier otro recurso acciones que son permitidas por el *Kernel* que se encarga de administrar todas estas solicitudes[9]. La labor del *Kernel* se puede dividir en varias funciones:

El *Kernel* es actualizado constantemente, utilizando un esquema de numeración para identificar cada actualización. Este esquema de numeración utiliza tres números separados por puntos para identificar las actualizaciones. Por ejemplo en la versión 2.6.32, los dos primeros números hacen referencia a la versión como tal, y el tercero indica la actualización. Los números impares, como por ejemplo la versión 2.5, se utiliza para designar un *Kernel* que están en proceso de desarrollo, mientras que los números de versión par, como 2.4, designan un *Kernel* estables. Es recomendable usar sistemas con *Kernel* estable.

---

[4] Una señal es una forma limitada de comunicación entre procesos empleada en Unix y otros sistemas operativos

[5] *Application Programming Interface*, es el conjunto de funciones que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

Tabla 1. Funciones del *Kernel*

FUNCIÓN	DESCRIPCIÓN
Administración de procesos	Crea y elimina procesos, administra la comunicación con el mundo exterior (Entrada/Salida). La comunicación entre procesos es fundamental para el desarrollo eficaz del sistema, esta es administrada por el *Kernel*.
Administración de memoria	La memoria es un recurso esencial para el funcionamiento del sistema. El *Kernel* crea un mapa virtual de la memoria para todos los procesos, permitiéndoles interactuar con ella por medio de funciones especiales definidas en el *Kernel*.
Sistema de archivos	En el sistema casi todo puede ser tratado como un archivo. El *Kernel* crea una estructura de archivos sobre el *hardware*, en donde a menudo abstrae archivos en el sistema.
Control de *hardware*	La mayoría de las operaciones del sistema están asignadas a un dispositivo físico. El *Kernel* debe tener adherido un controlador para cada *hardware* presente en el sistema.
Tareas de Red	Las tareas de red deben ser administradas por el sistema ya que, muchas de las operaciones de red no están asignadas a un proceso específico. El sistema se encarga de enviar paquetes de datos a través de las interfaces de programa y de red, controla la ejecución de los programas de acuerdo con la actividad de red.

2.3. *DRIVERS*

El *Kernel* de Linux está constituido por una red extensa y compleja de código[9] en donde por medio de la construcción de un *driver* se puede entender un poco su funcionamiento. Un *driver* es una especie de "caja negra" que permite que una parte determinada del *hardware* del sistema responda a un interfaz de programación definida. La interacción con el usuario se lleva a cabo por medio de una serie de llamadas estandarizadas que son independientes del controlador por lo tanto la función principal de un *driver* de dispositivo es establecer que estas llamadas particulares interactúen con el *hardware*. La interfaz de programación está hecha de tal forma que los *drivers* puedan ser elaborados como partes separadas del *Kernel*, pero compatibles. Estas partes pueden ser conectadas cuando sea necesario de tal forma que el sistema pueda operar sin ningún problema mientras que esta acción se realiza. Cabe resaltar que es importante tener un buen conocimiento acerca del dispositivo para el cual se quiere elaborar el *driver*.

El *Kernel* posee una característica especial que le permite aumentar o disminuir su funcionalidad mientras está en funcionamiento[9], partes de código pueden ser añadidos o quitados. Estas partes de código son llamadas módulos y los *drivers* son creados como un tipo especial de módulo. Hay que tener en cuenta que los dispositivos dentro del *Kernel* se ven como archivos y pueden ser de tres tipos principalmente:

- Dispositivos de caracteres: Permiten el acceso por medio de un flujo de *bytes*. Incluye llamados a sistema tales como *open, close, read* y *write*, comunes en archivos. Se interactúa con ellos por medio de archivos especiales (llamados nodos) en el sistema de archivos. Estos archivos especiales son creados solo para permitir la transmisión de datos de modo secuencial.
- Dispositivos de bloques: Se accede también por medio de nodos en el sistema de archivos. Se utilizan para alojar sistemas de archivos. Permite operaciones de entrada y salida de datos, lectura y escritura de datos como en un dispositivo de caracteres, pero posee una interfaz de acceso diferente

para el *kernel*.

- Dispositivos de red: Son usados para recibir y transmitir paquetes de datos, que se manejan por medio del subsistema de redes del *Kernel*. Solo recibe y transmite paquetes de red sin importar el tipo de conexión.

2.4. INTERFAZ GRÁFICA DE USUARIO

Es un programa informático que permite la comunicación entre el usuario y el sistema operativo proporcionándole elementos gráficos para mostrar la información y posibles opciones de la interfaz. Surgen como evolución de las interfaces de líneas de comandos que inicialmente utilizaron sistemas operativos primitivos. Los elementos gráficos que intervienen en la interfaz gráfica de usuario se denominan objetos gráficos los cuales, dentro de la interfaz pueden ser botones, texto, barras deslizantes, texto dentro de cuadros, ventanas, etc. En lenguajes de programación orientados a objetos creados como instancias de clases.

La forma en que el usuario interactúa con el sistema a través de la interfaz gráfica es por medio de un teclado, mouse o por medio de pantallas táctiles que permiten al usuario seleccionar opciones disponibles dentro de la interfaz y enviar ordenes al sistema.

La interfaz gráfica de usuario en Linux es posible gracias a un programa especial llamado *X Windows System* (sistemas de ventanas X). En Linux Embebido las interfaces gráficas de usuario no son soportadas por el software emphX Windows System como en el Linux normal, en su lugar implementa su propio sistema de ventanas, llamado QWS, lo que permite importantes ahorros de almacenamiento de memoria[6].

2.4.1. Qt

Qt es un kit de herramientas que se podría ver como una clase de librería del lenguaje de programación C++ y un conjunto de herramientas multi-plaforma, usado para elaborar interfaces gráficas de usuario, en donde el enfoque principal es "escriba una vez, compile en cualquier parte"[4]. Es usado en una amplia gama de programas tales como Adobe Photoshop Album, Google Eart, skype, virtual box, entre otros. Fue desarrollado por Haavard Nord y Eirik Chambe-Eng en Noruega. Cuenta con una API intuitiva y poderosa que permite el desarrollo de aplicaciones en múltiples plataformas[4] haciendo uso del lenguaje de programación C++ orientado a objetos de forma nativa, por tal razón la creación de objetos gráficos se realiza por medio de instancias de clases definidas por defecto dentro de kit de herramientas. Es un *software* libre y de código abierto, aunque existen versiones comerciales en la que se debe pagar una licencia para utilizarlas. Qt utiliza una herramienta llamada qmake[1] que permite la compilación de código fuente y posterior creación de la aplicación , en donde el mismo archivo fuente puede ser utilizado para crear aplicaciones en diferentes plataformas.

Qmake crea un archivo de proyecto que contiene la lista de archivos fuentes involucrados en la creación de la aplicación. El archivo de proyecto puede ser de tres tipos: app (aplicación, para aplicaciones autónomas), lib (librerías, para bibliotecas estáticas y compartidas) y subdirs (subdirectorios, para construir subdirectorios recursivos)[9]. Las especificaciones de qmake se encuentran en un directorio de Qt llamado mkspecs.

2.4.2. Qt/Linux Embebido

Es una versión optimizada de Qt para Linux Embebido. Qt/Linux Embebido ofrece la misma API y herramientas de Qt. Puede funcionar en cualquier *hardware* en donde funcione Linux, como mencionamos anteriormente, posee su propio sis-

[1] Herramienta que permite la creación de Makefile, utilizado para la compilación y creación de aplicaciones de *software*

tema de ventanas llamado QWS. Cabe resaltar que el desarrollo de software para funcionar en sistemas embebidos es difícil ya que tienen procesadores más lentos, menos capacidad de almacenamiento, menos memoria y pantallas más pequeñas que los computadores personales.

Al igual que para Linux embebido, es necesario estar dotado de una cadena de herramientas que permitan realizar aplicaciones para esta versión de Qt. Es posible usar la cadena de herramientas de Linux Embebido para crear aplicaciones para Qt/Linux Embebido. A pesar que esta versión de Qt cuenta con la misma API, se debe tener en cuenta que los recursos de hardware de visualización y de entrada de datos por parte del usuario es muy limitada así que se deben crear aplicaciones específicas para estos sistemas.

3. MANUAL DE USUARIO DE LA TARJETA SIE

El manual de usuario contiene los procedimientos necesarios para configurar el computador donde se realizarán las aplicaciones tanto de *drivers* como de aplicaciones gráficas construidas con Qt. Un conjunto de herramientas como compiladores cruzados, depuradores, *software* para la gestión de proyectos, entre otros, permiten la realización de dichas aplicaciones. Estas herramientas deben estar instaladas y configuradas adecuadamente para la puesta en marcha de las guías. Además también hace falta preparar la tarjeta para que contenga los recursos necesarios para ejecutar las aplicaciones que se realicen con las guías. Las temáticas que serán abordadas por este manual de usuario son:

3.1. COMANDOS

Esta sección presenta una explicación breve sobre los comandos de Linux utilizados en el desarrollo de las guías y algunas secciones del propio manual de usuario.

3.2. INSTALACIÓN DE LAS HERRAMIENTAS DE COMPILACIÓN PARA LA TARJETA SIE

En esta sección se muestran los pasos necesarios para descargar una imagen de la distribución de Linux OpenWrt que contiene los elementos necesarios para configurar y preparar el sistema, dando paso a la realización de aplicaciones usando todo un conjunto de herramientas cruzadas en donde se incluyen versiones de programas usados comúnmente en la compilación de código fuente en un

computador personal, pero para la arquitectura del procesador de la tarjeta SIE. Además muestra la manera como se compilan librerías y se incluyen paquetes de *software* para la tarjeta.

3.3. "*FLASHEO*" DEL *KERNEL* Y DEL SISTEMA DE ARCHIVOS RAÍZ

No solo hace falta configurar el computador de desarrollo, también la tarjeta SIE necesita ser configurada con aquellas librerías que compilamos en el conjunto de herramientas. Esta parte del manual presenta los pasos necesarios para realizar esta tarea. Un procedimiento llamado "*flasheo*"[1]. En este proceso se graban las imágenes del *Kernel* y del sistema de archivos Raíz. Estas imágenes son creadas durante el proceso de compilación de la imagen de OpenWrt que se realiza en la sección anterior y son necesarias para poder correr las aplicaciones creadas en el computador de desarrollo. Esta sección del manual de usuario guía al estudiante en este proceso.

3.4. CONFIGURACIÓN DE LAS HERRAMIENTAS DE COMPILACIÓN PARA LA CREACIÓN DE APLICACIONES EN QT

Para la creación de interfaces gráficas de usuario, además de las herramientas de compilación cruzada, hace falta utilizar una herramienta llamada qmake que crea los elementos necesarios para la compilación de la aplicación. Esta sección muestra la forma en la que se debe configurar la herramienta, utilizando las herramientas de compilación instaladas previamente.

[1] Término que se utiliza para hacer referencia a la programación de la memoria *flash* que posee la tarjeta, en esta memoria se encuentran grabadas las imágenes del Kernel, el sistema de archivos raíz y el gestor de arranque del sistema operativo

3.5. CONCEPTOS Y FUNCIONES ESPECIALES PARA LA CREACIÓN DE DRIVERS

En el desarrollo de drivers existen funciones especiales definidas en el *Kernel* que ofrecen una serie de servicios que permiten al *driver* interactuar con el sistema y tomar recursos de él. Además, una serie de estructuras permiten representar el dispositivo a controlar dentro del sistema. Esta parte del manual presenta los conceptos básicos necesarios sobre Linux y su *kernel*, enfatiza en la definición de cada una de funciones y las estructuras dentro del *kernel* necesarias para la elaboración del *driver*.

3.6. CONCEPTOS Y MÉTODOS ESPECIALES PARA LA CREACIÓN DE INTERFACES GRÁFICAS DE USUARIO EN QT

La programación en Qt es totalmente diferente a la utilizada para la creación de *drivers*. La programación orientada a objetos es utilizada por Qt para crear programas de interfaz gráfica por medio de instancias de clases y el uso de métodos, definidas en clases predeterminadas de Qt. Las clases básicas que se necesitan en Qt son definidas en esta sección. Los métodos son una serie de funciones definidas en una clase utilizadas por los objetos de una clase, algunos métodos especiales permiten ubicar los objetos visuales de una aplicación llamados *widgets* y otras permiten darle funcionalidad a estos objetos. Esta sección del manual profundiza en estos temas.

4. GUÍAS DE LABORATORIO

Las guías elaboradas durante el desarrollo del proyecto están consignadas en el anexo GUÍAS DE LABORATORIO DE SISTEMAS EMBEBIDOS[1], las cuales presentan una estructura general entendible y concisa que permiten un claro entendimiento sobre los pasos a seguir para la construcción de *drivers* básicos e interfaces gráficas de usuario en Qt, que permiten cumplir con los objetivos planteados en cada guía. La organización de cada una de la guías es la siguiente: Título de la guía a desarrollar, Introducción acerca de los procedimientos y conceptos que permiten mostrar de forma general la temática principal de la guía, objetivos que se quieren alcanzar después del desarrollo de la guía, Marco teórico que explica los conceptos básicos sobre los temas que se abordarán en el desarrollo de las guías, un procedimiento general que muestra de forma resumida aquellas actividades que se van a realizar en la guía, un procedimiento detallado que explica paso a paso las actividades a efectuar para cumplir con los objetivos propuestos, un resumen que permite explicar brevemente lo que se desarrollo en la guía y unas preguntas de prueba que permiten comprobar que los objetivos propuestos se cumplieron.

4.1. GUÍAS DE LABORATORIO DE DRIVERS

Las guías de laboratorio de *drivers* abordan los temas fundamentales de programación en el *Kernel*, que permitirán entender el funcionamiento de los *drivers* en el *Kernel* de Linux por medio de la elaboración de un *driver* sencillo pero funcional

[1] La guías 1,2,3,4,5 y 6 del anexo B hacen referencia al desarrollo de *drivers* y las guías 7, 8, 9 y 10 hacen referencia al desarrollo de interfaces gráficas de usuario en Qt

que permite interactuar con algún *hardware* de la tarjeta SIE desde el espacio de usuario. El desarrollo de las guías se llevara a cabo por estudiantes que matriculen la asignatura Sistema Embebidos que dará fin a la rama de sistemas digitales ofrecida por la E3T y que le permitirán crear *drivers* para aquellos periféricos que se deseen agregar a la tarjeta SIE.

Los conceptos abordados de forma secuencial permiten dar un ambiente de aprendizaje a los estudiantes que vienen trabajando con la tarjeta SIE en asignaturas anteriores. Dentro de las prácticas se abordaron cuatro temas fundamentales en la creación de *drivers* en donde solo hace falta tener un computador con el conjunto de herramientas apropiadas y una tarjeta SIE preparado para el proceso.

La creación de *drivers* en el espacio del *Kernel* se realiza utilizando el lenguaje de programación C, pero esto no es limitante para los estudiantes ya que dentro del pensum de Ingeniería Electrónica de la E3T existe una asignatura que es ofrecida en los primeros niveles de formación, que aborda los conceptos teóricos y prácticos de este lenguaje de programación. Además no se necesita ser un experto programador del *Kernel* para poder realizar las guías, ya que estas orientan al estudiante y le proporcionan los conceptos necesarios para entender cada paso del proceso.

A continuación se describen los temas más importantes de las guías propuestas:

4.1.1. Primer Módulo

El objetivo principal de esta guía es elaborar un módulo que permitirá dar inicio al desarrollo de *drivers*. Un módulo es una fracción de código del *Kernel* que puede ser añadido o retirado del mismo, mientras éste está funcionando. Los *drivers* son módulos, aunque no todos los módulos son *drivers*. El módulo se elabora mediante la creación de un archivo fuente con código escrito en lenguaje C. Dentro de este archivo fuente deben existir una función especial que inicializa el módulo en

el momento de la instalación y otra que es llamada cuando el módulo se retira. El proceso de compilación se realiza por medio de la ejecución de un Makefile[1] que finalmente crea el módulo. Además de crearlo hace falta ejecutar unos comandos predeterminados por el sistema para añadirlo en el *Kernel*.

La guía orienta paso a paso este proceso, explicando cada una de las líneas involucradas en el archivo fuente del módulo. Muestra que debe contener el Makefile necesario para la compilación del código fuente y el uso de los comandos implicados en la instalación del *driver* en el sistema. Dentro de la justificación teórica se explican los conceptos implicados en la guía.

4.1.2. Números mayores y menores

El objetivo de esta guía es mostrar como se reservan números de dispositivo y como se crea un nodo para el dispositivo a controlar. Además de crear el módulo con las funciones de inicialización/limpieza e instalarlo en el *Kernel* hace falta registrarlo e identificarlo dentro del sistema. Un par de números llamados "número mayor" y "número menor" son los encargados de esta tarea, el primero identifica al *driver* como tal y el segundo se encarga de distinguir los dispositivos a cargo del *driver*. Son asignados al *driver* por medio de una función especial del *Kernel* que revisa en el sistema que números están disponibles y los reserva. Un archivo especial llamado nodo abstrae en el espacio de usuario el dispositivo para el cual se crea el *driver*.

Esta guía permite mostrar en el sistema de archivos de la tarjeta en donde se encuentran registrados estos números y los nodos de los dispositivos que están siendo controlados actualmente en el sistema. Se complementa el archivo fuente que se creó en la guía anterior. Cabe resaltar que el archivo fuente creado en la primera guía será complementado a medida que se vayan desarrollando las

[1] Un *Makefile* es un archivo especial que contiene una serie de instrucciones que debe seguir un programa llamado *make*. *make* es un programa que permite crear aplicaciones u otras acciones definidas en el *Makefile*

guías. Los conceptos y los pasos para la realización de la reserva de los números de dispositivo se encuentran consignados en la guía.

4.1.3. Estructuras de datos en el Kernel

Dentro del *Kernel* existen unas series de estructuras especiales que permiten abstraer los dispositivos. Además del nodo, una estructura se encarga de representar el dispositivo en el *Kernel* y otra de representarlo cuando se abre como un archivo. Hace falta crear estas estructuras y definir llamados al sistema de modo que un usuario o aplicación en el espacio de usuarios pueda acceder al dispositivo. Por último se reserva memoria del sistema para el dispositivo.

El objetivo de esta guía es presentar la forma como se representan los dispositivos en el sistema para su posterior control. La definición básica de llamados a sistema tales como *open, release, read* y *write* son presentados inicialmente con una rutina muy básica para exhibir su comportamiento.

4.1.4. Operaciones de Archivo

Finalmente debe existir una rutina en cada uno de los llamados al sistema definidos para que el usuario o la aplicación del espacio de usuario puedan intercambiar datos con el dispositivo por medio del nodo. Funciones especiales permiten leer y escribir en los registro de entrada y salida de la memoria del procesador de la tarjeta SIE. El objetivo de esta guía es elaborar las rutinas necesarias para que una aplicación desde el espacio de usuario interactué con el dispositivo, por medio del nodo.

La guía propone una forma, como ejemplo para hacer usa de los llamados a sistema, por medio de la manipulación de un dispositivo conectado a la tarjeta por defecto. La explicación de cada uno de los llamados del sistema se hace evidente en la sección del marco teórico, como la explicación de cada una de las líneas que

contiene el código fuente del driver. Cabe resaltar que las guías culminan con la creación de un *driver* básico que contiene todos los elementos esenciales para su funcionamiento, pero existen otros temas que no son cubiertos por las guías y que son necesarios para el desarrollo de un driver completo. Los alcances de este trabajo de grado no permiten cubrir con estos tópicos asía que se deja un punto de inicio y una puerta abierta para el posterior estudio de estos temas.

4.2. GUÍAS DE LABORATORIO PARA QT

Las guías de laboratorio para la elaboración de interfaces gráficas de usuario con Qt abordan los temas necesarios para que el estudiante que matricule la asignatura Sistemas Embebidos pueda obtener las capacidades necesarias en la creación de interfaces gráficas para algún sistema implementado en la tarjeta SIE. El desarrollo de interfaces gráficas de usuario en Qt está orientado al uso de programación orientada a objetos en el lenguaje de programación C++ pero esto no es limitante al igual que sucede en las guías de *drivers* ya que en la misma asignatura ofrecida en niveles iniciales de la carrera de Ingeniería Electrónica también se abordan los temas referentes al lenguaje de programación C++.

A continuación se describen los temas más importantes de las guías propuestas:

4.2.1. Primera aplicación en Qt

Las aplicaciones en Qt se elaboran por medio de clases personalizadas elaboradas en el lenguaje de programación C++, consignadas en un serie de librerías integradas en las herramientas de Qt por tal motivo se hace necesario el uso de herramientas de compilación cruzada y del programa qmake para compilar el código fuente de la aplicación y crear el archivo ejecutable que permitirá ejecutar

la aplicación. Los objetos visuales dentro de la aplicación y la aplicación misma son creados por instancias de clases predefinidas en Qt, así que es fundamental definir una instancia para la aplicación que permitirá el uso de otras clases para incluir los *widgets*. La palabra *widget* se utiliza en Qt para hacer referencia a los objetos visuales dentro de la aplicación.

La guía pretende presentar al estudiante un primer ámbito que le permita tener una idea clara de lo que debe contener de primera mano el archivo fuente de su aplicación. Además muestra los pasos necesarios para la compilación del archivo fuente. Esta guía por su simplicidad permite comprobar que el estudiante configuro de manera correcta su conjunto de herramientas de compilación para le creación de aplicaciones en Qt.

4.2.2. Ubicación de los *widgets*

Los *widgets* son ubicados de forma conveniente por Qt, de acuerdo al espacio disponible dentro de la aplicación. Como muchas de las aplicaciones que vemos diariamente, están conformadas por varios objetos visuales que deben ser organizados de forma adecuada en busca de un impacto visual adecuado y que permitan mostrar la información completa al usuario. Dejar este trabajo a Qt no es la manera más adecuada ya que muchas veces no lo hace como se esperaría que lo hiciera. Por tal motivo Qt está equipado con una serie de clases especiales que permiten ubicar los *widgets* de acuerdo a las necesidades que tenga el programador.

La finalidad de esta guía es presentar una serie de clases básicas que le permitan organizar los *widgets* de varias maneras por medio del uso de las clase especiales ofrecidas por Qt para esta tarea. Los conceptos y las herramientas necesarias para tal fin son abordados por la guía presentando varias opciones para hacerlo.

4.2.3. *Signals* y *slots*

En las aplicaciones de interfaz de usuario los *widget* deben comunicarse entre ellos para cumplir con una labor especifica, ya sea cerrar la aplicación con un *click* de un *mouse* o pulsador, enviar datos al sistema, configurar algún dispositivo por medio de una orden, etc. Por tal motivo Qt está equipado de una serie de herramientas llamadas *signal* y *slot* que permite la comunicación entre objetos de una clase. La *signal* se encarga de avisar que se realizo una acción por parte del usuario a un ente que le pueda interesar y el *slot* se encarga de realizar una actividad específica.

La guía ofrece los fundamentos teóricos de las *signals* y *slots*, que le permiten entender su funcionamiento dando paso a que el estudiante las pueda utilizar adecuadamente. Orienta en el uso de *slots* que por defecto trae Qt e incita al estudiante a crear *slots* personalizados.

5. VALIDACIÓN

La validación de las guías fue desarrollada por medio de una asignatura que dio el ambiente necesario para realizar dicha validación. Esta asignatura permitió dar un preámbulo a la creación de la asignatura Sistemas Embebidos. Matriculada por compa-ñeros de proyectos afines al uso de la tarjeta SIE y otros estudiantes que adelantan trabajos de grado con el grupo de investigación CPS, quienes se encargaron de poner en práctica cada una de las guías propuestas para la materia. En ella se explicaron los conceptos necesarios para el diseño de drivers y aplicaciones GUI para la tarjeta SIE y posteriormente se entregaban las guías para que los estudiantes las desarrollaran.

Durante la puesta en marcha del desarrollo de las guías se presentaron los siguientes puntos más importantes:

- Diversos errores referentes a partes de código mal escritos y falta de claridad en algunos de los pasos de las mismas.
- Vacíos dentro los conceptos necesarios para la puesta en marcha de las guías.
- Inconvenientes en el manejo de los lenguajes de programación usados para la elaboración de drivers e interfaces gráficas de usuario con Qt.
- Falta de claridad en algunos puntos importantes que se deben tener en cuenta para el adecuado desarrollo de las guías.
- Comandos de Linux utilizados incorrectamente.
- Falta de claridad respecto a donde se debían ejecutar (computador de desarrollo o tarjeta SIE) los comandos citados en la guías.
- Algunos estudiantes empezaron a desarrollar las guías y por uno u otro motivo interrumpieron este proceso si haber terminado completamente la guía, este hecho generó una serie de problemas ya que la guía tiene una estructura secuencial en donde algunos de los pasos subsiguientes dependen de pasos anteriores.

Cabe resaltar que en la implementación de la asignatura Sistemas Embebidos existe un periodo de transición mientras que a la asignatura llegan las personas que vienen trabajando con la tarjeta desde la asignatura de Sistemas Digitales, así que es entendible que se presenten vacíos conceptuales con respecto a algunos temas sobre el uso del sistema de desarrollo SIE.

CONCLUSIONES Y RECOMENDACIONES

Dado al actual uso de sistemas embebidos tanto en la vida cotidiana como en los ambientes laborales y de la academia surge la necesidad de buscar herramientas que permitan el aprendizaje y desarrollo acerca de la estructura, funcionamiento e implementación de estos sistemas. Muchos de los sistemas embebidos funcionan bajo un sistema operativo que se encarga de administrar todos los recursos tanto de *hardware* y *software* que el sistema posee. Este proyecto suple la necesidad de crear *drivers* que controlen los dispositivos conectados al sistema y una interfaz gráfica que permita al usuario visualizar información sobre el proceso que está realizando el sistema por medio de una serie de guías implementadas en un sistema de desarrollo llamado SIE que permite la construcción de sistemas embebidos de uso específico.

El sistema de desarrollo SIE ofrece un ambiente que contiene todos los recursos tanto de *software* como de *hardware*, que permiten diseñar e implementar sistemas avanzados que desarrollen trabajos específicos. Dado el entorno por parte de la tarjeta SIE hacía falta elaborar una serie de documentos que permitan sacar provecho de los recursos ofrecidos por el sistema y enseñen a los estudiantes como hacerlo, ya que no existía uno como tal. En este proyecto se elaboraron una serie de documentos:

- Manual de usuario de la tarjeta SIE: en donde están consignados los requerimientos y configuraciones necesarias tanto en el computador desarrollador como en la tarjeta SIE, que permiten la ejecución de las guías. Además contiene los conceptos más importantes de cada guía.
- Guías de laboratorio *drivers*: en donde por medio de cuatro guías prácticas se abordan los temas básicos involucrados en la elaboración de un driver funcional para los periféricos conectados al procesador de la tarjeta, por alguno de sus registros de Entrada/Salida. Cabe aclarar que el tema de drivers es muy extenso, así que da cabida para la realización de otro proyecto que com-

plemente a este, para que finalmente permita a la asignatura de Sistemas Embebidos ofrecer una enseñanza completa en el desarrollo de drivers.

- Guías de laboratorio interfaces gráficas de usuario con Qt: en donde tres guías explican el uso de las clases más fundamentales de Qt que permiten la construcción de interfaces gráficas de Usuario, permitiendo que los estudiantes puedan diseñar GUI con elementos visuales agradables que permitan sofisticar más el diseño de su sistema embebido.

Por medio de una asignatura se realizó el proceso de validación de las guías, en donde se encontraron una serie de errores de código, vacios conceptuales, falta de claridad en muchos conceptos y procedimientos que fueron corregidos inmediatamente. Esto permitió que las guías fueran retroalimentadas y perfeccionadas de modo tal que ofrecieran una herramienta confiable en el proceso de aprendizaje de los temas tratados.

Se sugiere que cuando se implemente la asignatura de sistemas embebidos se dicte una clase sobre lenguajes de programación tanto de C como de C++, antes de poner en práctica las guías ya que se evidenció muchas falencias en el uso de estos lenguajes de programación, sobre todo en el tema de programación orientada a objetos. Esta clase permitirá recordar aquellos conceptos fundamentales de cada lenguaje, ya que muchos estudiantes en el transcurso de su proceso de formación dejan de usarlos llegando con conceptos débiles y olvidados.

Se recomienda que los estudiantes una vez comiencen a desarrollar las guías le den término en la misma sesión donde se ejecuta la práctica. En el proceso de validación se presentó esta situación que dio paso a una serie de errores inesperados durante el desarrollo de la práctica.

Las guías fueron elaboradas pensando en servir como apoyo práctico a cada uno de los temas involucrados en la asignatura de Sistemas Embebidos. Así que se aconseja implementar una explicación teórica acerca de los conceptos que se verán comprometidos en el desarrollo de las guías para que así el estudiante

pueda desarrollar la guía con más confianza y pueda alcanzar sin ningún problema los objetivos propuestos.

BIBLIOGRAFÍA

[1] https://openwrt.org/, Abril 2012.

[2] WEEKS, Roger; DUMBILL, Edd and JEPSON, Brian. Linux Unwired, O Reilly, 2004.

[3] CAMPDERRICH, Benet. Ingenieria del Software. UOC. 2003.

[4] BLANCHETTE, Jasmin and SUMMERFIELD, Mark. C++ GUI Programming with Qt 4, Prentice Hall, 2008.

[5] EZUST, Alan and EZUST, Paul. An Introduction to Design Patterns in C++ with Qt 4, Prentice Hall, 2007.

[6] YAGHMOUR, Karim. Building Embedded Linux Systems, O Reilly, 2003.

[7] RUBINI, Alessandro; KROAH-HARTMAN, Greg and CORBET, Jonathan. Linux Device Drivers, O Reilly, 2005.

[8] P. BOVET, Daniel and CESATI, Marco. Understanding the Linux Kernel, O Reilly, 2000.

[9] VENKATESWARAN, Sreekrishnan. Essential Linux Device Drivers, Prentice Hall, 2008.

[10] M. DIETEL, Hardvey and J. DEITEL, Paul. Como programar en C/C++, Prentice Hall, 1994.

[11] W. KERNIGHAN, Brian and M. RITCHIE, Dennis. El lenguaje de programación C, Pearson Educación, 1991.

[12] CEBALLOS, Francisco Javier. Enciclopedia del lenguaje C++, Alfaomega, 2009.

[13] PETERSEN, Richard. Linux: Manual de referencia, MCGRAW-HILL, 2008.

[14] SIEVER, Ellen; FIGGINS, Stephen; LOVE, Robert y ROBBINS, Arnold. Linux, La guía definitiva de Linux, O REILLY, 2010.

[15] ADELSTEIN, Tom y LUBANOVIC, Bill. Administración de sistemas Linux, Anaya Multimedia, 2007.