Host USB

Juan Carlos Bordachar, Lucas Hernan Enriquez, Rodrigo Medina Maciel

Abstract. La presente investigación tiene por objeto el desarrollo de un driver para el Host USB, integrándolo al sistema operativo Sodium, el cual es un desarrollo con fines educativos del departamento de Ingeniería de la UNLaM. Se investigará acerca del protocolo USB y sus diferentes versiones (1.0, 1.1, 2.0), enfatizando la forma de comunicación que tiene el Host USB con los dispositivos conectados a los puertos. También se analizará cual es la tarea que lleva a cabo la BIOS en la detección de dispositivos USB..

Keywords: Host Controller, USB Controller, Driver, Endpoint, URB, Buffer, PCI Configuration.

1 Introducción

El protocolo del Universal Serial Bus (USB) está destinado principalmente a la interconexión de dispositivos de forma sencilla y confiable. Debido a que la interfaz USB no está patentada por ninguna empresa, y por lo tanto es de libre acceso, ha llegado a convertirse en el estándar para la comunicación de dispositivos periféricos en computación por la reconfiguración de hardware, expansibilidad e interconexión del equipo de forma transparente al usuario.
La inclusión de la capacidad de operar con dispositivos USB en un sistema operativo se considera importante para el mismo.

1.1 Especificaciones Host Controller Interface (HCI).

Un sistema USB tiene una controladora, concentradores (HUBs) y un concentrador raíz (Root HUB), entre otros, y permite tener hasta 127 dispositivos USB incluyendo los concentradores según determina la especificación de Compaq [COM98]. La controladora no es otra cosa que la interfaz de hardware entre el dispositivo USB y el Sistema Operativo. Hoy en día hay un par de interfaces HCI (Interfaz de Controladora de Host) en uso y son:

- Universal Host Controller Interface (UHCI): Especificación creada por Intel.
- Open Host Controller Interface (OHCI): Especificación que tiene su origen en las empresas Compaq y Microsoft. Un controlador compatible con OHCI tiene más inteligencia incorporada en el hardware que uno UHCI, por lo que un driver para un Host Controller OHCI es relativamente más simple.
- Enhanced Host Controller Interface (EHCI): Este es el controlador de Host que admite dispositivos USB 2.0. Suelen venir acompañados de controladores OHCI o UHCI para manejar los dispositivos más lentos.

- Extensible Host Controller Interface (xHCI): Especificación que reemplaza actualmente a las tres anteriores. Aplica a dispositivos USB 3.0 y es la cuarta tasa de transferencia llamada SuperSpeed (5 Gbit/s máxima, 3.2 Gbit/s nominal), manteniendo compatibilidad hacia atrás con dispositivos de estándares anteriores.

A continuación se muestran las capas básicas en las que cualquier subsistema USB está dividido lógicamente desde el punto de vista software.

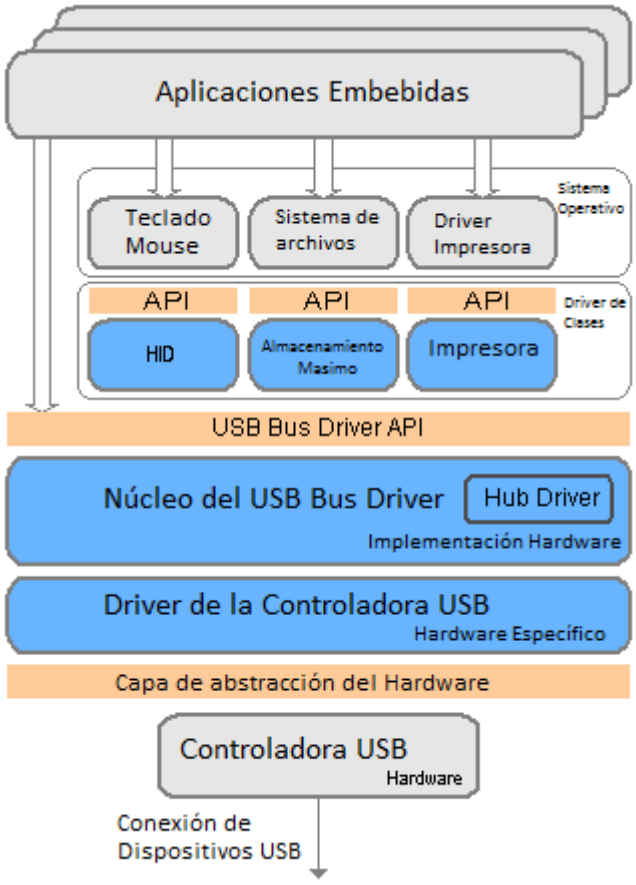

Fig. 1. Capas Subsistema USB.

1.2 Transmisión y codificación.

La especificación USB determina el largo del cable para el conector USB (sin repetir con HUBs) en 5 metros. Esto es porque la alimentación eléctrica es suministrada al dispositivo por medio de la conexión USB, y un voltaje mínimo de 4.4V es garantizado por la especificación. El cableado en grandes distancias puede reducir la potencia suministrada, provocando así la inoperancia del dispositivo. El

largo máximo del cable especificado asegura que un delay de 30 nanosegundos no será excedido. Dado que la información viaja de forma bidireccional a través del cable entre la PC y el dispositivo, una propagación mayor del delay podría resultar en comunicaciones consideradas como no recibidas o bien enviadas por segunda vez antes de recibir respuesta, acusando recibo de los datos.

Los datos son transmitidos en forma serie, en 2 líneas de datos complementarias denominadas D+ y D-. Además se proveen 2 líneas de alimentación y de masa respectivamente, las cuales pueden servir para que el dispositivo tome alimentación del Host (5 V, 500 mA máx.).

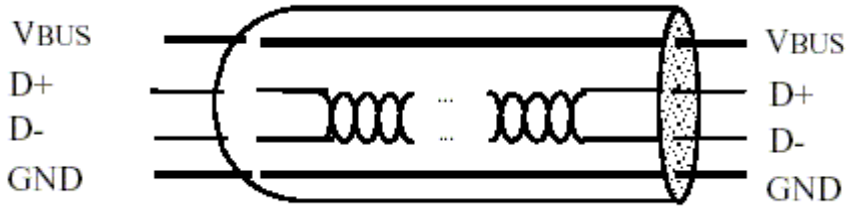

Fig. 2. Cables que componen un cable USB.

Para transmitir los datos en forma serie se utiliza la codificación Non-Return-To-Zero-Inverted o NRZI. En este tipo de codificación, un 0 (cero) se representa sin un cambio de nivel en la tensión, y un 1 (uno) se representa con un cambio de nivel en la tensión.

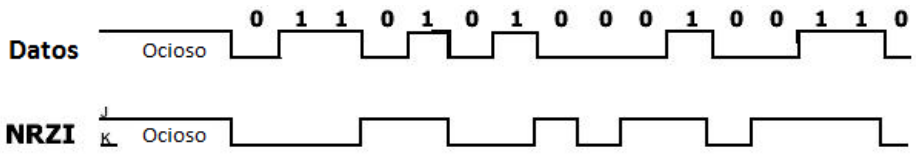

Fig. 3. Relación entre información binaria a enviar y codificación NRZI.

Conjuntamente, se utiliza el bit stuffing, técnica que consiste en insertar un 0 (cero) cada 6 (seis) 1s (unos) consecutivos en el flujo de bits, dado que se pueden producir largos periodos en los que la señal no cambia dando lugar a problemas de sincronización, de ésta forma insertando un 0 (cero) el receptor puede volverse a sincronizar. El relleno bits empieza con el patrón de señal Sync. El "1" que finaliza el patrón de señal Sync es el primer uno en la posible primera secuencia de seis unos. En las señales a velocidad media o baja (low-speed y full-speed), el relleno de bits se utiliza a lo largo de todo el paquete sin excepción. De modo que un paquete con siete unos consecutivos será considerado un error y por lo tanto ignorado.

Teniendo en cuenta que K y J representan respectivamente nivel bajo y nivel alto, el patrón de señal Sync emitido, con los datos codificados, es de 3 pares KJ seguidos de 2 K para el caso de velocidad media y baja.

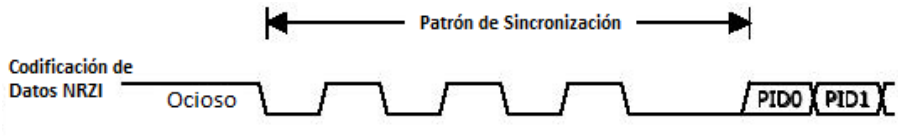

Fig. 4. Patrón Sync codificado en NRZI.

El patrón de señal Sync siempre precede al envío de cualquier paquete, teniendo como objetivo que el emisor y el receptor se sincronicen y se preparen para emitir y recibir datos respectivamente.

A todo paquete le sigue EOP (End Of Packet), cuya finalidad es indicar el final del paquete. En el caso de velocidad media y baja (full-speed y low-speed) el EOP consiste en que, después del último bit de datos en el cual la señal estará o bien en estado J, o bien en estado K, se pasa al estado SE0 durante el periodo que se corresponde con el ocupado por dos bits, finalmente se transita al estado J que se mantiene durante 1 bit. Esta última transición indica el final del paquete.

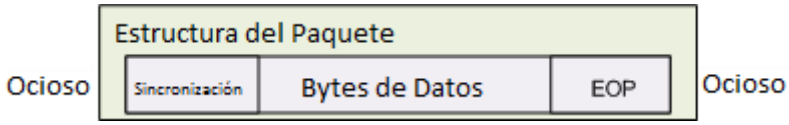

Fig. 5. Estructura de paquete.

Por otro lado, además del bit stuffing y de la codificación NRZI, se utilizan CRCs[1], los cuales se generan después del bit stuffing.

1.3 Transferencias.

Una Transferencia es un bloque de datos que conforma una estructura comprensible para el Host o el Dispositivo. Existen dos tipos básicos de Transferencias relacionadas con los Endpoints:

- Las de Control: El tamaño de éstas es de 8 bytes y se reconocen porque comienzan con un paquete del tipo SETUP.
- Las de Datos: Siempre comienzan con paquetes IN u OUT.
  - Transferencias de Interrupción (Interrupt Data Transfers): Baja tasa de transferencia de datos. Comunicación disponible para aquellos dispositivos que demandan mover muy poca información y poco frecuentemente. Tiene la particularidad de ser unidireccional, es decir del dispositivo al Host, notificando de algún evento o solicitando alguna información. Su paquete de datos tiene las mismas dimensiones que el de las transmisiones de control.

[1] Comprobación de Redundancia Cíclica: Código de detección de error cuyo cálculo es una larga división de computación en el que se descarta el cociente y el resto se convierte en el resultado.

- Transferencias de Bultos (Bulk Data Transfers): Para transferir cantidades relativamente grandes de datos en un solo envío. El ancho de banda disponible para este tipo de transferencia varía en función de la disponibilidad de éste.
- Transferencias Isócronas (Isochronous Data Transfers): Ocupan un ancho de banda del USB, cuya latencia es negociada en la configuración. Sirven para transmitir datos a intervalos regulares de tiempo.

Las Transferencias de Control siempre inician con un paquete SETUP, el cual tiene información sobre la función de configuración que el Host desee que el Dispositivo realice. Son las únicas que utilizan las transacciones de SETUP.

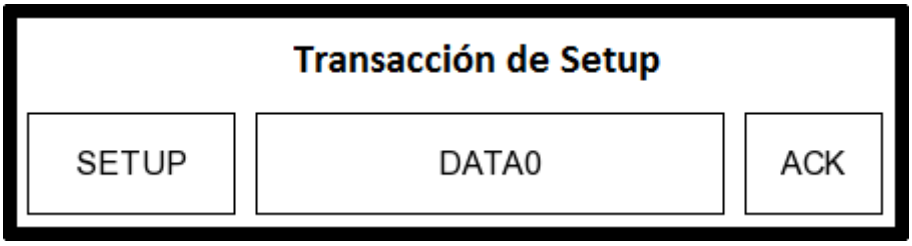

Fig. 6. Transacción de SETUP, la cual consiste en un paquete de SETUP, DATA0 y handshake (en éste caso ACK).

Las Transferencias de Control se llevarán a cabo en un máximo de tres etapas.

- Etapa SETUP: consiste simplemente en una transacción de SETUP.
- Etapa DATA: es opcional. Si se la utiliza puede contener una o más transacciones IN, o una o más transacciones OUT.
- Etapa STATUS: consiste en una sola transacción IN o OUT. Si la etapa DATA se encuentra presente, entonces STATUS utiliza el tipo de transacción opuesto al de DATA. Cuando se omite la etapa DATA, STATUS utiliza solo una transacción IN.

1.4 Solicitudes (Requests).

Todos los dispositivos USB responden a las solicitudes del Host mediante la Default Control Pipe. Las solicitudes se manifiestan en las Transferencias de Control.

Tipo de Solicitudes:

- Solicitudes de Dispositivos Estándar (Standard Device Requests): Son las comunes a todos los tipos de dispositivos USB. Sirven para que el Host identifique y configure un dispositivo durante el proceso de Enumeración.
- Solicitudes de Clase (Class Requests): Son las relativas a cualquier clase particular de dispositivo USB. Cada clase de dispositivo (excepto las clases genéricas o definidas por los fabricantes, [vendor specific]) está definida en una especificación de clase USB. Cada clase tiene sus requisitos propios, tales como tipo y número mínimo de Endpoints; además de otro tipo de Descriptor, que es llamado Descriptor de Reporte (Report Descriptor), el cual indica al Host cómo deben

interpretarse los datos que envía el Dispositivo al Host según la función del Dispositivo.

1.5 Descriptores.

Son tablas en las que los Dispositivos almacenan información sobre sus características, no siendo modificables por el Host (están grabadas en ROM). Los descriptores son jerárquicos.

Los descriptores que están en los niveles más altos de la jerarquía de Descriptores, informan al Host sobre la existencia de los descriptores que están en los niveles más bajos de la jerarquía de descriptores.

Descriptores comunes a todos los tipos de Dispositivos USB:

- Descriptor de Dispositivo (Device Descriptor): Contiene información sobre el máximo tamaño de paquete que soporta el Endpoint0, cuántas configuraciones soporta el Dispositivo, y otra información. Es el primero que lee el Host.
- Descriptor de Configuración (Configuration Descriptor): Existe uno por cada posible forma de operar del Dispositivo (solo puede haber una configuración activa en un determinado momento). Tiene información sobre cuántas Interfaces existen por configuración.
- Descriptor de Interfaz (Interface Descriptor): Tiene información sobre el número de Endpoints (excepto el Endpoint0) que utiliza la Interfaz y sobre la Clase a la que pertenece.
- Descriptor de punto final (Endpoint Descriptor): Existe uno por cada Endpoint de una Interfaz. Tiene información sobre el número de Endpoint. También sobre el tipo de Transferencia que realiza (Control, Interrupción [Interrupt], Bulk o Isócrono [Isochronous]). Y también tiene información sobre el tamaño máximo de paquete del Endpoint.

A continuación se describen los descriptores básicos de las Clases USB:

- Descriptor de la Clase (Class Descriptor): Especifica a qué Clase USB pertenece una Interfaz. También da información sobre la longitud del Report Descriptor.
- Descriptor de Reporte (Report Descriptor): Tiene información sobre cómo debe el Host interpretar las Transferencias del Dispositivo. Su estructura es totalmente diferente al del resto de los Descriptores. En el contexto del USB, una Transferencia es equivalente a un Report, y es un bloque de datos que el Host puede interpretar.

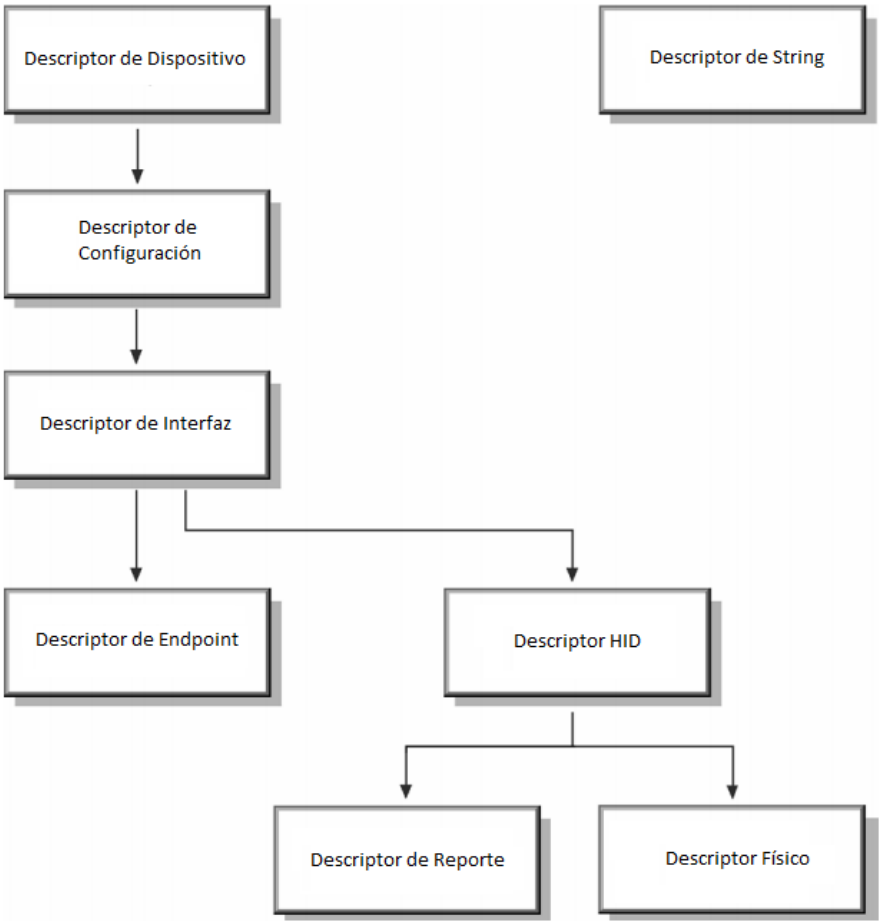

Fig. 7. Descriptores de un Dispositivo.

1.6 Concentrador o HUB

Dispositivo que permite concentrar varios puertos USB, permitiendo la conexión con una máquina mediante un solo bus.

Además del controlador, nos encontramos con el concentrador raíz (Root HUB). Este es el primer concentrador de toda la cadena que permite a los datos y a la energía pasar a uno o dos conectores USB de la PC, y de allí a los 127 periféricos que, como máximo, puede soportar el sistema.

Son elementos claves dentro del Sistema USB. Adicionalmente, simplifican de gran manera la sencillez de la interconexión de dispositivos al computador.

Bajo una óptica eléctrica e informática, los HUBs son concentradores cableados que permiten múltiples conexiones simultáneas. Su aspecto más interesante es la

concatenación, función por la que a un HUB se le puede conectar otro y otro, ampliando la cantidad de puertos disponibles para periféricos.

El HUB está compuesto por dos partes importantes:

- Controlador del HUB: Puede asemejarse a una pequeña CPU de supervisión de las múltiples funciones que deben desempeñar un hub.
- Repetidor del HUB: Tiene la función de analizar, corregir y retransmitir la información que llega al HUB hacia los puertos del mismo. Mantiene una memoria consistente en varios registros de interfaz que el permiten sostener diálogos con el Host y llevar adelante algunas funciones administrativas además de las meramente operativas.

Nos encontramos con dos tipos bien diferenciados según el tipo de fuente de alimentación.

- Sin fuente de alimentación o "Bus-powered": Toma la energía del bus USB, es decir de la interfaz USB del ordenador. No necesita una conexión de energía separada. Estos concentradores pueden tener cuatro puertos como máximo y sólo admiten la conexión de dispositivos de bajo consumo, es decir, que tengan un consumo máximo de 100 mA cada uno, hasta un total de 500 mA (400 mA para los 4 dispositivos más 100 mA para alimentación del propio HUB).
- Con fuente de alimentación o "Self-powered": Tienen su propio alimentador externo, es decir, toma su energía de una fuente de alimentación externa y puede por lo tanto proporcionar plena alimentación a cada puerto. Pueden tener, teóricamente, hasta 255 puertos (sin embargo, el máximo número de dispositivos conectables simultáneamente en un bus USB es de 127). La mayoría de los HUBs tienen cuatro o siete puertos.

Funcionamiento general.

El HUB USB tiene la capacidad de detectar si un Dispositivo ha sido conectado a uno de sus puertos, notificando de inmediato al Host, activándose el proceso que desata la configuración del nuevo periférico; adicionalmente, los HUBs también son capaces de detectar la desconexión de un dispositivo, notificando al Host que debe remover las estructuras de datos y programas de administración (drivers) del periférico retirado.

Otra de las funciones importantes de los HUBs es la de aislar a los puertos de baja velocidad de las transferencias a alta velocidad, proceso sin el cual todos los dispositivos de baja velocidad conectados al bus estarían en colapso. La protección de los dispositivos lentos de los rápidos ha sido siempre un problema serio dentro de las redes mixtas, como es USB.

El Root HUB dispone de una cantidad limitada de puertos disponibles para conectar dispositivos, por lo cual para conectar hasta 127 dispositivos necesitamos utilizar HUBs USB con varios puertos; de modo que un Dispositivo se puede conectar directamente al puerto del concentrador raíz o a un puerto de HUB, sin variar para nada su funcionamiento. De hecho, algunos dispositivos pueden funcionar como HUBs al tener conectores USB incorporados, como los teclados. También podemos

conectar un Dispositivo a un HUB, que a su vez esté conectado a otro HUB que está conectado al puerto de Root HUB y el funcionamiento del dispositivo será igual que estando conectado directamente al puerto del Root HUB.

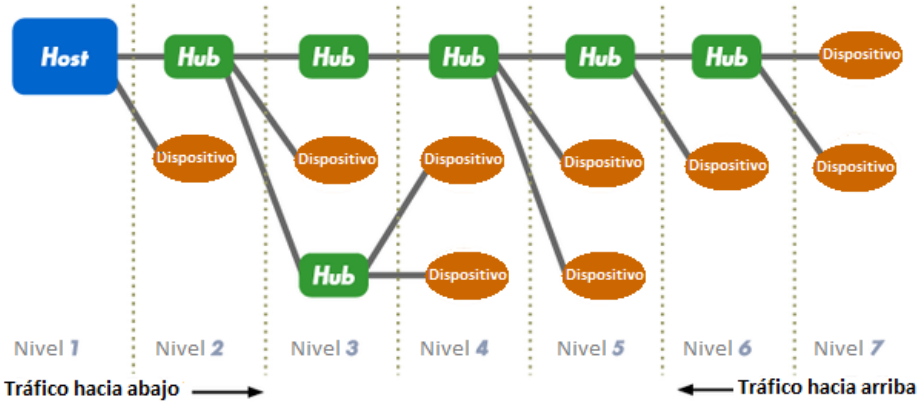

Fig. 8. Topología física USB soportando HUBs.

Una consideración importante es la cantidad de niveles que se pueden tener no debe superar a 7, cuestión fijada por la especificación USB [COM98].

1.7 Dispositivo de Interfaz Humana (Human Interface Device [HID])

Los Dispositivos USB de clase HID utilizan un correspondiente HID Driver para el envío y recepción de los datos hacia y desde el Dispositivo al Host USB. Esto se logra examinando los descriptores de los Dispositivos y la información que estos proporcionan.

Los descriptores de las funciones HID describen que otros descriptores se encuentran presentes y los tamaños de estos. El protocolo HID realiza la implementación de los Dispositivos de una forma muy sencilla. Estos definen sus paquetes de datos y luego presentan un "Descriptor HID" al Host. El descriptor HID es codificado como un arreglo de bytes que describen los paquetes de datos del Dispositivo. Esto incluye cuantos paquetes soporta el dispositivo, que tan grandes son los paquetes, y el propósito de cada byte y bit en el paquete según refiere la especificación de la Organización USB [USB01]. Por ejemplo, un mouse puede decirle al Host que el estado de presionar/soltar el botón izquierdo, es almacenado por ejemplo en el 2° bit del 6 byte en el paquete de datos número 4. Estas ubicaciones de los datos varían según el fabricante, por ejemplo si el mouse es Logitech o Microsoft el click izquierdo es el primer byte mientras que si es Manhattan es el segundo.

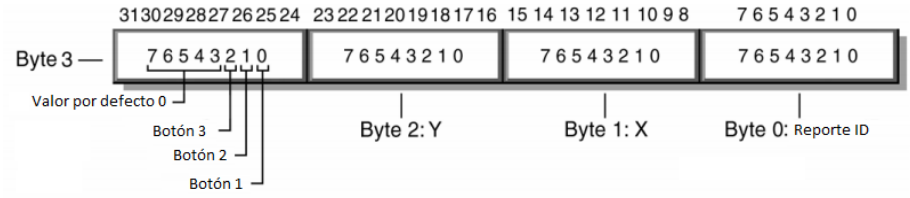

Fig. 9. Paquete de información enviado por Dispositivo mouse.

Un Dispositivo de tipo HID se comunica con el driver utilizando el pipe de Control y el de Interrupción. El primero es utilizado para recibir solicitudes de Control USB y datos de la clase; por otro lado el pipe de Interrupción se utiliza para realizar las transferencias de Interrupción desde el dispositivo y transmitir datos de baja latencia hacia el dispositivo.

1.8 Teclado

Los teclados son algunos de los Dispositivos USB más populares de la case HID.

Un teclado realiza sus funciones mediante un micro controlador. Estos micro controladores tienen un programa instalado para su funcionamiento; estos mismos programas son ejecutados y realizan la exploración matricial de las teclas cuando se presiona alguna, y así determinar cuales están pulsadas.

Para lograr un sistema flexible los micro controladores no identifican cada tecla con su carácter serigrafiado en la misma sino que se adjudica un valor numérico a cada una de ellas que sólo tiene que ver con su posición física. El teclado latinoamericano sólo da soporte con teclas directas a los caracteres específicos del castellano, que incluyen dos tipos de acento, la letra eñe y los signos de exclamación e interrogación. El resto de combinaciones de acentos se obtienen usando una tecla de extensión de grafismos. Por lo demás el teclado latinoamericano está orientado hacia la programación, con fácil acceso al juego de símbolos de la norma ASCII.

Por cada pulsación o liberación de una tecla el micro controlador envía un código identificativo que se llama Scan Code. Para permitir que varias teclas sean pulsadas simultáneamente, el teclado genera un código diferente cuando una tecla se pulsa y cuando dicha tecla se libera. Si el micro controlador nota que ha cesado la pulsación de la tecla, el nuevo código generado (Break Code) tendrá un valor de pulsación incrementado en 128. Estos códigos son enviados al circuito micro controlador donde serán tratados gracias al administrador de teclado, que no es más que un programa de la BIOS y que determina qué carácter le corresponde a la tecla pulsada comparándolo con una tabla de caracteres que hay en el kernel, generando una interrupción por hardware y enviando los datos al procesador. El micro controlador también posee cierto espacio de memoria RAM que hace que sea capaz de almacenar las últimas pulsaciones en caso de que no se puedan leer a causa de la velocidad de tecleo del usuario. Hay que tener en cuenta, que cuando realizamos una pulsación se pueden producir rebotes que duplican la señal. Con el fin de eliminarlos, el teclado también dispone de un circuito que limpia la señal.

Disponer conexión USB en el teclado tiene la ventaja de hacerlo independiente del hardware al que se conecta. El estándar define Scan Codes de 16 bits que se transmiten por la interfaz. Del 0 al 3 son códigos de error del protocolo, llamados NoEvent, ErrorRollOver, POSTFail, ErrorUndefined, respectivamente. Del 224 al 231 se reservan para las teclas modificadoras (LCtrl, LShift, LAlt, LGUI, RCtrl, RShift, RAlt, RGUI).

El teclado USB de la clase HID está normalmente diseñado con un endpoint IN que comunica las pulsaciones al ordenador, y con un endpoint OUT que comunica el estado de los led del teclado desde la computadora al teclado mismo.

El estándar PC87 requiere que la BIOS del ordenador detecte y trabaje con teclados USB de la clase HID para ser utilizados durante el proceso de arranque.

2 Desarrollo.

Los controladores de Host USB se encuentran estandarizados, y cumplen con alguna de las cuatro especificaciones que existen en el mercado (UHCI, OHCI, EHCI, xHCI). Esto se tuvo en cuenta al momento del desarrollo, dado que más allá de conservar compatibilidad hacia atrás los estándares de USB 2.0 y USB 3.0, tanto UHCI y OHCI están en un mismo nivel y uno no contempla al otro. En un principio se optó por desarrollar un controlador para OHCI, pero ante la falta directa de emulación para éste por parte de QEMU[2], se determinó realizar dicho controlador para UHCI tomando como referencia para ello la especificación de Intel [INT96].

El desarrollo del driver para el Host USB se realizó teniendo en cuenta la necesidad de conseguir un acoplamiento bajo por parte de éste respecto a Sodium. Por otro lado, en Sodium se llevaron a cabo una serie de desarrollos/cambios mínimos necesarios para que pueda éste utilizar el driver usb.

Los archivos que se vieron alterados en Sodium fueron:

- herramientas/generar_lanzador.sh: Se agregó el argumento "-usb" en la llamada a QEMU (línea #132).
- include/common/sodheap.h
- include/common/sodstdlib.h
- include/common/sodstring.h
- include/common/system_def.h
- include/common/tipos.h
- include/kernel/scrap.h: Agregado prototipo de función de demonio encargado del polling en background.
- include/kernel/syscall.h
- include/kernel/system.h
- include/kernel/system_asm.h
- include/kernel/drivers/pci.h
- common/sodheap.c
- common/sodstdlib.c

[2] http://www.linux-kvm.org/page/USB

- common/sodstring.c
- kernel/main.c
- kernel/syscall.c
- kernel/system.c: Agregado el Handler para USB y modificada la función que inicializa la IDT para que apunte al Handler.
- kernel/system_asm.asm: Handler en assembler.
- kernel/drivers/pci.c

Los archivos agregados:

- include/drivers/devices.h: Lista con Vendors y Devices de PCIs.
- include/drivers/lsusb.h
- include/drivers/uhci.h
- include/drivers/usb.h
- include/drivers/usbdesc.h
- include/drivers/usbtrans.h
- include/drivers/usb_teclado.h
- kernel/drivers/lsusb.c
- kernel/drivers/uhci.c
- kernel/drivers/usb.c
- kernel/drivers/usbhub.c
- kernel/drivers/usbtrans.c
- kernel/drivers/usb_teclado.c
- usr/bin/lsusb.c

El funcionamiento perseguido para la detección de dispositivos USB conectados al ordenador fue el siguiente:

Mediante una espera pasiva. Se encuentra corriendo un demonio (al mismo nivel que el Shell, para no bloquearlo) el cual se encarga de detectar la conexión de un dispositivo a la máquina. Al ser un demonio, permite que se utilice Sodium de la forma en que el usuario desea sin verse alterada su fluidez por culpa del mismo.

El demonio hace el polling en busca de dispositivos conectados al Host, inicializando los mismos al detectarlos. Se enumera por completo al Dispositivo y se muestra por pantalla una descripción alusiva a la detección de la conexión de un Dispositivo indicando a su vez el nombre del producto. En el caso de ser un Dispositivo HUB se muestra actualmente por pantalla al momento del polling los Dispositivos conectados en cada uno de los puertos del HUB.

Si se desea visualizar los dispositivos ya detectados, se utiliza el comando “lsusb” el cual imprimirá por pantalla la lista de dispositivos que se encuentran actualmente reconocidos por el Host dada su conexión con la PC, permitiendo que el usuario ingrese el número asociado al Dispositivo para obtener una descripción completa por pantalla de todos los descriptores del mismo.

Cabe destacar que se requirieron una serie de pasos obligatorios para inicializar la controladora USB, según se especifica Intel [INT96] y así poder utilizar la funcionalidad provista por ésta mediante una comunicación directa con la misma. Dichos pasos se llevan a cabo junto al inicio de Sodium.

Para comunicarnos con la controladora tuvimos primero que localizarla. Siendo ésta un PCI más, se realizó de forma dinámica la identificación de dispositivos PCI preguntando a cada uno de éstos la clase a la que pertenecen y así determinar si es o no un Host Controller USB (Clase 0x0C; SubClase 0x03).

De ésta forma se localizó el USB_ADDR como conjunción de la dirección PCI_ADDR (index port: 0x0CF8) más un offset dado por el BusNumber, DeviceNumber y FunctionNumber asignados a la controladora USB UHCI (datos obtenidos anteriormente en la identificación de los PCI). Cualquier comunicación u obtención de datos que requerimos siempre se realiza consultando al PCI_ADDR, el cual nos entrega lo solicitado en PCI_DATA (data port: 0x0CFC).

Una vez identificada y localizada la controladora, debimos inicializarla. Cuestión para la cual nos valimos de enviarle a la misma una serie de mensajes seteandole registros internos a fin de cumplir con lo establecido en la especificación Intel [INT96] y lograr así su "encendido".

El Host Controller UHCI dispone tal como lo marca la especificación Intel [INT96] un concentrador raíz, el cual cuenta con solo dos puertos. Sobre cualquiera de éstos se conectarán los Dispositivos (HUBs y/o funciones).

Para reconocer los Dispositivos conectados, se censa cada puerto y se pregunta por el estado del mismo, el cual nos determina si hay o no un Dispositivo conectado que no ha sido aún "configurado" por el Host (cuando se lleva a cabo la enumeración de un dispositivo, se le asigna a éste una dirección la cual es única dentro de todos los dispositivos conectados, a fin de poder identificar unívocamente los destinatarios de los mensajes a enviar/recibir). En tal caso (Dispositivo conectado aún no "configurado") se lleva a cabo la enumeración con el mismo, obteniendo de éste los datos necesarios/requeridos para poder operar correctamente con el mismo; se guarda dicha información en una estructura generada para tal fin. Cada dispositivo detectado tiene una variable del tipo estructura asociada allocada en memoria, dicha variable se almacena en un array.

El caso especial en el que el Dispositivo conectado es un HUB, se llevan a cabo una serie extra de pasos más allá de la enumeración. Estos consisten básicamente en determinar la cantidad de puertos que dispone y el tipo de HUB que es (Bus-powered o Self-powered) y poder así no solo agregar la referencia a cada puerto del HUB a la lista de puertos a censar en busca de Dispositivos cada vez que se haga un polling, sino que además consultar en el caso de los Bus-powered la energía requerida por cada uno y configurar de ésta forma la asignación de la misma, encendiendo así cada uno de los puertos dejándolo listo para la conexión de un Dispositivo. Inmediatamente después de hecho el "power-on" de los puertos se chequea cada uno de éstos en busca de un Dispositivo, cubriendo así la situación en la que se conecta directamente un HUB ya con dispositivos conectados a éste.

Cada vez que se debe realizar el polling en busca de dispositivos, se chequea el estado de los puertos propios del concentrador raíz y los puertos de cada uno de los concentradores detectados. En el caso que se detecte en un puerto una variación indicando que se produjo una desconexión, se realizan los pasos necesarios considerando el caso en que no solo un Dispositivo fue desconectado, sino varios dada la desconexión en sí de un HUB con Dispositivos conectados a éste.

Cada estructura representativa a cada Dispositivo conectado dispone de un puntero el cual indica el "padre" de ese dispositivo, es decir, sobre el puerto de quien está conectado; permitiendo así la determinación en cualquier momento del nivel de anidamiento en que se encuentra cada Dispositivo, más allá que a fines funcionales el Host trata a todos los Dispositivos por igual asumiéndolos en un mismo nivel lógico.

QEMU fue el emulador utilizado para realizar las pruebas dado que Bochs presentó problemas en cuestiones referidas a controladores USB. Así mismo también se utilizó VMware para realizar pruebas más exhaustivas en cuestiones referidas a conexión de dispositivos. De todos modos las pruebas en máquina real fueron las que realmente nos marcaron el paso y nos condicionaron cualquier avance seguro.

2.1 Situación

A la fecha, la investigación ya cuenta con las siguientes funcionalidades, capacidades desarrolladas:

- Detección de la ubicación física del controlador USB UHCI.
- Inicialización del controlador USB UHCI. Más allá de su detección por medio del offset más PCI_ADDR es necesario inicializarlo, lo cual conlleva una serie de pasos los cuales ya están resueltos..
- Demonio que se encarga de realizar la tarea de censar todos los puertos determinando si se conectó o no un dispositivo (hot-plug[3]).
- Secuencia de pasos necesarios para realizar la enumeración de los Dispositivos conectados y detectados al ordenador.
- La información obtenida de los dispositivos se almacena en una lista ubicada en memoria de forma dinámica. Tal información además se muestra por pantalla con el formato correspondiente para identificar unívocamente cada valor/dato.
- Programa de usuario encargado de listar los dispositivos USB detectados. Mostrando inicialmente al ejecutarlo la lista de dispositivos reconocidos, permitiendo que mediante el ingreso del número asociado al dispositivo se visualice una completa y detalla información obtenida de sus descriptores.
- Acciones necesarias para dejar listos los Dispositivos del tipo HUB a la espera de la conexión de otros Dispositivos.
- Reconocimiento del anidamiento de Dispositivos mediante HUB. Se reconoce para cada Dispositivo el nivel físico ([Fig.8.]) en el que se encuentra conectado y lo muestra al listar la información del mismo.
- Anidamiento de Dispositivos HUB.
- Mensaje por pantalla indicando el suceso ocurrido, determinando correctamente si se trata de una conexión o desconexión de un Dispositivo.

[3] El usuario podrá conectar y desconectar los dispositivos USB las veces que quiera sin que tenga que apagar y encender la máquina. Cabe aclarar que no es lo mismo que "Plug and Play", donde si se conecta una impresora, cámara fotográfica, o scanner a través de la interfase USB, no es necesario apagar el equipo ni hacer que el sistema busque el nuevo Hardware ya que el sistema automáticamente reconoce el dispositivo conectado e instala los controladores adecuados.

- Implementación de semáforo a fin de evitar conflictos al momento de consultar la variable que contiene la identificación de cada Dispositivo.
- Implementación inicial del Driver para Mouse USB, el cual permite obtener del mismo información propia a los movimientos realizados y botones pulsados. Es una versión inicial la cual presenta falencias en determinados aspectos (como ser por ejemplo el manejo de diferentes modelos de Mouse).
- Implementación inicial de Driver para Teclado USB. La misma presenta dificultades para su comprobación dado el problema presentado al conectar un teclado a la máquina virtual.

Todo lo antes descripto implementa el sistema de documentación doxygen, así como también respeta la nomenclatura utilizada en el código de Sodium para funciones, estructuras de datos y variable, y cumple con un indentado de 3 espacios en reemplazo a cada tabulación.

Restan por refinar las siguientes características:

- Driver mouse USB.
- Driver teclado USB.

Para la implementación del driver que controle un mouse USB en Sodium y según describe la especificación provista por la organización USB (Usb.org - [USB01]) se implementaron una serie de funciones las cuales realizan solicitudes y recepción de datos de control para la configuración del dispositivo. Así como la generación de descriptores de transferencia que luego serán leídos y de estos se obtendrán los datos de posición o botones presionados o soltados.

Para comprobar el correcto funcionamiento del driver para el mouse USB, se refleja en la consola de Sodium en la parte inferior derecha los datos leídos del mouse, como ser movimientos y pulsación de los botones izquierdo y derecho del Dispositivo. De igual forma se realiza en lo que respecta a teclado USB.

Dado que no se manejan interrupciones para tomar los datos tanto del Mouse como del Teclado, se hizo que el mismo proceso que se encarga de hacer el polling de los Dispositivos cuando se encuentra ocioso realice el polling en busca de información en los dispositivos HID que se encuentran conectados (Teclado y Mouse).

El tema de tratamiento de HUB está finalizado, más allá que resta hacer correcciones en diferentes mensajes arrojados por pantalla para que quede el soporte de USB en Sodium de una manera menos intrusiva en lo que respecta a mensajes propios de debug en la interfaz consola.

Actualmente Sodium muestra por pantalla el reconocimiento e inicialización exitosa del Host Conroller UHCI. Queda a la espera pasiva de un dispositivo conectado (corre demonio). Ante la conexión de un dispositivo se advierte por pantalla tal situación. Dispositivos del tipo HUB son configurados correctamente encendiendo los puertos que disponen éstos. Ante la conexión de un Mouse USB, se muestran por pantalla los datos obtenidos de éste al momento de su uso. El comando lsusb permite visualizar los dispositivos que se encuentran actualmente detectados.

De cada Dispositivo se obtiene la información contenida en todos descriptores del mismo, sea cual fuere el tipo de descriptor.

En versiones anteriores existía el comando “pusb” el cual permitía lanzar de forma manual la acción necesaria para realizar el polling sobre los puertos y detectar así cualquier cambio (conexión o desconexión de Dispositivos), igual tarea a la que realiza actualmente por sí solo el demonio. Dado esto se eliminó el comando para realizar el polling de forma manual, mismo para evitar problemas al momento de compartir el acceso a la variable que aloja las instancias de todos los Dispositivos ya detectados.

Mismo en comparación a versiones anteriores, se aumentaron los tiempos de delay para permitir así una mayor estabilidad al Subsistema USB. También se incluyeron mayores controles respecto a la liberación de variables para evitar conflictos con la memoria.

3 Problemas Presentados.

Los pasos para el desarrollo del driver Host USB, se tornaron por momento dificultosos. A continuación se detallan algunos de los problemas a sortear.

3.1 Funcionalidades faltantes en Sodium.

Dado que Sodium es un sistema operativo con fines educativos aún en desarrollo, existen muchas funcionalidades propias de un SO que actualmente se encuentran en desarrollo o bien aún no han sido planificadas. Mismo, muchas funcionalidades sobre las que no se trabaja activamente y se dan por “finalizadas” no cuentan con la estructura y estabilidad correspondiente y necesaria. El diseño y desarrollo de un driver Host USB sin disponer de determinadas funcionalidades provistas por el SO o sobre funcionalidades inestables se torna dificultoso y requiere de pensar alternativas no muy formales ni recomendables en situaciones ideales.

Algunas funcionalidades faltantes que podemos mencionar son:

- Subsistema SCSI.
- Manejo de semáforos.

Funcionalidades que no presentan una robustez absoluta:

- Manejo de memoria.

Pasamos a describir cuales son dichas alternativas pensadas para cada una de las funcionalidades necesarias faltantes.

Subsistema SCSI.

Es necesario solo para Dispositivos USB de almacenamiento masivo, para escribir o leer datos, no así para posibilitar la detección del mismo (enumeración); por lo cual inicialmente se considera que no será un problema, aunque llegado el momento de realizar pruebas más allá de la detección del Dispositivo si será una traba. Es por ello que se opta por trabajar fundamentalmente con Dispositivos HID (Human Interface Device) básicos (por ejemplo: teclado, mouse).

Manejo de semáforos.

Tanto el proceso polling que se encarga de censar los puertos como así también el proceso encargado de mostrar por pantalla la información de los Dispositivos conectados, utilizan una misma variable global al Subsistema USB, la cual almacena la información propia de cada Dispositivo conectado. Dicho uso compartido de la variable hace que, si se consulta a través del comando "lsusb" la descripción de alguno de los Dispositivos y justo el planificador da permiso en ese momento al proceso encargado de la detección de Dispositivos para que tome el control del procesador, se genere una excepción por violación en el uso de recursos compartidos.

Se implementó en forma precaria el uso de semáforos a fin de minimizar éste problema.

Manejo de memoria.

Por especificación [INT96] al momento de allocar memoria para determinadas estructuras explicitas para el uso de UHCI es necesario hacerlo con una alineación especifica de memoria. Dicha cuestión es contemplada en funciones semejantes a memalign disponibles en bibliotecas libc de Linux. Dado que no se puede utilizar dichas librerías, tal funcionalidad debía estar implementada en alguna de las funciones propias de Sodium. El problema fue que no existía nada semejante, con lo cual se debió generar una función que alinee memoria considerando el manejo que se hace de la misma en el Sistema Operativo en cuestión. Tal situación no fue fácil y de hecho presentó muchos inconvenientes dado que se debió analizar cómo estaban resueltas determinadas cuestiones en Sodium referentes a memoria, ajenas a lo que es el Subsistema USB a implementar. Aún no se encuentra solventado del todo éste problema, provocando que la detección de dispositivos se limite actualmente a solo 2, ya que más de esa cantidad provoca errores de memoria los cuales se encuentran en análisis para su corrección.

3.2 Investigación teórica escasa.

Quedaron demasiadas cuestiones a analizar más en detalle para llevar a cabo adelante el desarrollo con la menor cantidad posible de obstáculos/inconvenientes, las cuales obviamente fueron descubiertas al ser cruzadas forzosamente durante el desarrollo.

Se subestimo por momentos la complejidad propia del desarrollo de un Subsistema USB.

Se considera que la investigación también debió haber abarcado temas referidos a PCI fundamentalmente (ej: PCI BIOS Specification) y Host Controllers entre otros.

Los resultados obtenidos en la búsqueda de material en buscadores científicos no arrojaron claridad en determinados asuntos. Esto pudo deberse a una falta de conocimiento en búsqueda de material científico, pero lo concreto es que muchas patentes fueron arrojadas como resultados ante búsquedas, las cuales poca relevancia técnica proveían.

3.3 Accesibilidad a PCI desde Modo Protegido.

Inicialmente se intentó conseguir los datos del controlador USB, el cual es visto como un dispositivo PCI más, desde Modo Real para lo cual se debió desarrollar todo en assembler (en el archivo sodium.asm). Esto se debió a la información recibida respecto a la inaccesibilidad a la BIOS desde Modo Protegido, la cual era necesaria para realizar la búsqueda del dispositivo PCI en cuestión, USB Host Controller.

Finalmente resultó que se requiere de la BIOS para consultar los dispositivos PCI si se lo hace a través de las interrupciones que proporciona la misma, lo cual sí es válido que solo desde Modo Real se lo puede hacer; pero no así si se utiliza el PCI_ADDR y PCI_DATA para realizar la lectura del puerto y determinar el offset correspondiente al dispositivo buscado. Dicha consulta al puerto PCI puede realizarse tanto en Modo Protegido como Real. Desde luego que en Modo Protegido se tiene una mayor facilidad para realizar debug del código ante problemas y cosas por el estilo, dado que es factible utilizar código en lenguaje C. Ante lo dificultoso que se tornaba cada cambio o agregado a hacer en el código assembler para la lectura de los PCI, se resolvió migrar todo lo ya hecho en Modo Real a Protegido, debiendo modificar mucha lógica ya que en el primer modo mencionado se utilizaban interrupciones de la BIOS, mientras que desde el segundo solo se consulta el puerto PCI (0x0CF8 y 0x0CFC).

3.4 Tratamiento de HUB.

No se había considerado que el Dispositivo HUB requería un tratamiento especial por sobre el común de los demás Dispositivos posibles de conexión. Con lo cual la investigación no profundizó cuestiones propias del mismo. Es así que se omitieron en principio tareas semejantes al encendido de cada puerto del concentrador para permitir el uso de los mismos. Esto generó que la investigación y recolección de información sobre el mismo se realice de forma tardía, encontrando que era mucha la información a nivel conceptual y técnico sobre HUBs ([COM98] Capitulo 11). Tal situación requirió dedicar mucho tiempo al tema e hizo que no sea tan sencillo concluir el tratamiento y manejo de éste tipo de Dispositivo de una forma simple y rápida.

3.5 Inexistencia de HUB en Máquina Virtual.

Al realizar las pruebas de conexión de dispositivos para comprobar el funcionamiento correcto del driver desarrollado, teníamos dos alternativas para llevar a cabo las mismas; la primera era utilizar una maquina virtual en VMware y la otra utilizar una máquina real. Para llevar a cabo ambas era necesario instalar Sodium en una unidad de diskete, dependiendo el caso que fuere se hacía en una unidad virtual si se corría la prueba en VMware o bien en un diskete físico si se corría sobre máquina real. Ésta última era una tarea mucho más lenta y tediosa de llevar a cabo. La opción preferida fue VMware por una mera cuestión de simplicidad y velocidad en llevar a cabo la prueba. El problema se presentó cuando al conectar un Dispositivo HUB a la

máquina física donde corría el emulador, no se visualizaba éste (el dispositivo) como opción para conectar a la máquina virtual. Situación que no ocurría con cualquier otro Dispositivo de otra clase. Fue así que para llevar a cabo las pruebas con HUB se realizaron cada una de éstas en máquina real, con las dificultades ya mencionadas que ésta presentaba.

Promediando el desarrollo específico para HUB fue cuando se descubrió que VMware "no ve" a los Dispositivos HUB como tales, pero sí realiza la identificación de los dispositivos que están conectados a través de éste, permitiendo conectarlos a las máquinas virtuales. Por otro lado, lo más importante para nuestra simplicidad en las tareas de pruebas, fue descubrir que VMware emula un dispositivo HUB con 7 puertos; brindándonos esto una alternativa interesante para realizar las pruebas sin tener que llevar las mismas a máquina real pasando por la instalación de Sodium en un diskete 3½ fisico.

3.6 Teclado USB en Máquina Virtual.

Por razones ajenas, VMware nos presenta problemas al momento de realizar la conexión del Teclado USB. Impidiendo así que la implementación del Subsistema USB identifique correctamente a éste y por consiguiente realice el trato adecuado para permitir su uso.

4 Conclusión.

El desarrollo del driver para el Host USB se encuentra finalizado en una versión inicial, en la cual se demuestra la comunicación con los dispositivos que se conecten. Desde luego hay cosas por mejorar y perfeccionar, pero no son cuestiones que impidan observar los logros obtenidos a lo largo del desarrollo.

El tema propuesto de investigación es por sí mismo demasiado complejo y amplio en las posibilidades que brinda de investigación resultándole justo, y hasta tal vez escaso, el tiempo disponible para su elaboración/desarrollo.

Lo que se logró es identificar a la controladora USB, inicializarla y configurarla, armar las estructuras necesarias para llevar a cabo la comunicación, transferencias y almacenamiento de los datos recabados; pasos explicitados para la enumeración, muestra por pantalla de datos de dispositivos, programa de usuario, programa corriendo en background haciendo el polling de los puertos, drivers básicos para mouse y teclado.

Se presentaron muchos y diversos problemas de diversas índoles. Realmente por momentos se tornó complicado y sin visión clara de alcanzar el objetivo propuesto; pero gracias al esfuerzo y dedicación se resolvieron en su mayoría los problemas que estaban a nuestro alcance de ser solucionados.

5 Bibliografía.

[01] http://sopa.dis.ulpgc.es/ii-dso/leclinux/drivers/usb/LEC_USB.pdf

[02] http://www.linuxjournal.com/node/5604/print

[AXE09] Axelson Jan, "USB Complete – The Developer's Guide, Fourth Edition", 06/2009

[COM08] ComBlock, "USB 2.0 Interface User Manual", 12/2008, http://www.comblock.com/ download/USB20_UserManual.pdf

[COM98] Compaq, Intel, Microsoft, NEC, "Universal Serial Bus - Specification", 23/09/1998, revision 1.1

[COM99] Compaq, Microsoft, "OpenHCI, Open Host Controller Interface Specification", National Semiconductor, 14/09/1999, revisión 1.0a, ftp://ftp.compaq.com/pub/supportinformation/papers/hcir1_0a.pdf

[CYPXX] Cypress Semiconductor, "EZ-USB® Technical Reference Manual", 198 Champion Court, San Jose, CA 95134-1709.

[INT10] Intel, "eXtensible Host Controller Interface for Universal Serial Bus (xHCI)", 21/05/2010, http://www.intel.com/technology/usb/download/xHCI_Specification_for_USB.pdf

[INT96] Intel, "Universal Host Controller Interface (UHCI), Design Guide", 03/1996, revisión 1.1, http://download.intel.com/technology/usb/UHCI11D.pdf

[TRO01] Trodden, Jay, "EHCI, A Survey Of Major Features", MindShare, Inc, 10/09/2001, http://www.mindshare.com/files/resources/MindShare_EHCI_whitepaper.pdf

[USB01] usb.org, "Device Class Definition for Human Interface Devices (HID)", Firmware Specification, 27/06/2001, versión 1.11, http://www.usb.org/developers/devclass_docs/HID1_11.pdf

[USS08] Usselmann R., "USB 2.0 function core", OpenCores.org, 12/2008, http://www.opencores.org/projects.cgi/web/usb

[VEN08] Venkateswaran, Sreekrishnan, "Essential Linux Device Drivers", Prentice Hall, 2008

[WIN00] Windows Platform Design Notes, "Keyboard Scan Code Specification", 16/03/200, revision 1.3a

[XIL08] Xilinx, "OPB universal serial bus device", 12/2008, revisión 1.00a, http://www.xilinx.com/support/documentation/ip_documentation/usb_ds591.pdf