Universidad Nacional Autónoma de México

Facultad de Ingeniería

TESINA POR TRABAJO PROFESIONAL

Prototipo de un sistema de automatización del proceso de cobro para la utilización de una computadora

Que para obtener el título académico de:
Ingeniero Eléctrico Electrónico

P R E S E N T A:
Omar Backhoff Larrazolo

DIRECTOR:
M.I. Luis Arturo Haro Ruiz

2013

Dedicatoria

Quiero orgullosamente dedicar esta tesis a...

Agradecimientos

Especial agradecimientos a...

Prefacio

Por qué

Siempre una de las grandes satisfacciones en la vida es el obtener un grado escolar; en mi caso, qué mejor grado que el que me otorgaría la Universidad Nacional Autónoma de México con el título de Ingeniero Eléctrico Electrónico. Cuando uno culmina sus créditos, se pregunta ¿Y ahora qué? ¿Cómo me titulo? Bueno, para empezar, cabe decir que uno puede ambicionar por una tesis monumental, titularse por promedio (debió planearse antes), o por cualquier otro método que nos convenga. No hay que menospreciar un modo u otro; no por ser acreedor al título por promedio o por una tesis que cambie al mundo hace menos válido otro método e titulación, de esto me daría cuenta más tarde. Yo elegí titularme por trabajo profesional, y aunque parezca un desapego a la academia, no lo es del todo. Titularse de esta forma involucra muchas cosa, y entre ellas está la de trabajar como académico (al menos en mi caso) debido a que al presentarse un problema uno debe recurrir al estudio, a la investigación, a gente que sabe más que uno, etc. Involucra un proceso más o menos similar al del método científico. Descubrir lo anterior me dio un gran ánimo de seguir adelante y no dejar a un lado la obtención de mi título.

Dónde

Todo empezó cuando un conocido Ingeniero, Juan Valencia, me comenta sobre una empresa para la cual trabajaría. Esta empresa buscaba un grupo de ingenieros que llevaran a cabo un proyecto de automatizar el cobro y el uso de un equipo de cómputo, para un café Internet, los detalles se mencionarán más adelante. Me intrigó mucho la cuestión que me planteaban, así que procedí a indagar en el tema; expuse que mis habilidades serían de utilidad para el proyecto y dije por qué. Me habían ofrecido trabajo. La empresa situada

en Veracruz me exigía visitas regulares para mostrar avances y propuestas; esto significa que podía trabajar en la Ciudad de México mientras cumplía con otros pendientes –perfecto, pensé. De otra forma habría sido mucho más difícil, aunque de todas formas lo fue.

Qué

Después de ser contratado, se me asigna el proyecto hablado: un prototipo de un sistema con un equipo de cómputo capaz de cobrar y controlar el uso de éste; no necesitar de cobradores en las tiendas o en los café Internet. Sería un reto. Las condiciones iniciales del proyecto es que utilizara la mayor cantidad de *software libre* posible, debía ser capaz este sistema de cobrar al cliente con monedas y otorgarle un determinado tiempo de uso del equipo y con determinados servicios, siempre pudiendo restringir lo que sea necesario. Realmente pensé que tendría que aprender acerca de lo más fundamental del sistema operativo a utilizar para lograr lo dicho, no estaba tan equivocado.

Cómo

Cuando mis jefes inmediatos me dijeron que debía utilizar *software libre*, inmediatamente *Linux* apareció en mi mente. Recordé que durante casi toda la carrera trabajé con un sistema operativo llamado *Ubuntu* (basado en *Linux*), y con el cual me di a la tarea de conocerlo minuciosamente; claro, tras haberlo arruinado en múltiples ocasiones, uno aprende. Decidido esto, fue hora de recorrer el camino que me llevaría a hacer algo que trascendería más allá de un salón de clases; llegaba la hora de aplicar conocimientos aprendidos en la facultad, de demostrar al profesionista que la Universidad Nacional es capaz de formar.

Contenido

En este documento resumo el trabajo que realicé durante la realización del proyecto "Prototipo de un sistema de automatización del proceso de cobro para la utilización de una computadora; realizado para la empresa DICIJE"y lo divido en 6 capítulos más un apéndice:

- Capítulo 1: Introducción; describo el objetivo, metas, contexto de participación profesional y metodología del proyecto.

- Capítulo 2: Marco Teórico; describo la teoría que está detrás de los conceptos involucrados en el proyecto.
- Capítulo 3: Desarrollo conceptual; describo el panorama general de lo que se busca desarrollar en el proyecto.
- Capítulo 4: Desarrollo técnico; describo a detalle el cómo se desarrolló y se logró el proyecto.
- Capítulo 5: Resultados y demostración del equipo; muestro el resultado de unir todas las piezas en un producto terminado.
- Capítulo 6: Conclusiones y resultados del proyecto; muestro las observaciones y resultados del trabajo después de concluido el proyecto.
- Apéndice: muestro el código de todos los programas que desarrollé para la realización del proyecto.

Índice general

Índice de figuras

Capítulo 1

Introducción

Si se busca la raíz de lo que se pretende con este documento, probablemente se llegue al siglo I; a los años 10-70 D.C. Su nombre era Herón de Alejandría; conocido ingeniero y matemático de la época. Es considerado uno de los grandes inventores de la historia, destacó en aquella ciudad del imperio Romano en Egipto por sus múltiples invenciones. Su trabajo más importante, quizás, es la primera máquina de vapor, escritos de mecánica e hidráulica y el primer libro de robótica de la historia llamado *los autómatas*, hasta se dice que describió, de forma un tanto arcaica, la ley de acción y reacción de Newton. Si quisiera hacer mención a todas las obras de este genial personaje, me desviaría del tema central, que es el propósito del escrito: una máquina capaz de permitir a un usuario la utilización de un equipo de cómputo, pudiendo cobrar de forma automática y previa al uso. Ahora, si hicimos tanto alboroto con Herón, no fue solo porque es interesante, sino porque él fue quien describió la primera máquina de ventas; esta máquina aceptaba monedas de la época, y a cambio regresaba agua bendita. Este mecanismo era muy sencillo pues la moneda caía sobre una bandeja conectada a una palanca, esta última abría una válvula y dejaba por un instante fluir el agua bendita, hasta que un contrapeso cerraba la válvula de nuevo.

Seguramente después de Herón de Alejandría no se volvió a hablar de una máquina que vendiera algún producto hasta principios de los 1880 en Londres, Inglaterra, durante la revolución industrial. Esta máquina fue diseñada para vender postales. Después en 1888, se vendió goma de mascar por medio de estas máquinas en las estaciones de trenes de Nueva York, EE.UU. De ahí en adelante, la evolución de estas máquinas fue tal que ahora se utilizan para

toda clase de productos; desde una máquina que vende café, otra que vende bebidas y comida chatarra, otra que permite escuchar una canción o jugar un videojuego, y hasta pueden cobrarte el tiempo que usaste un lugar de estacionamiento de automóviles. Éste es el concepto que se quiere implementar en el proyecto.

Cuando se tiene este concepto, cuando todo el proceso de venta es automatizado se presentan numerosas ventajas al evitar el riesgo y costo que representa la mano de obra humana, además invita al consumidor a hacer uso del producto debido a la facilidad que éste presenta. Si bien el costo por equipo aumenta con la integración de este concepto, es pragmático hacerlo; podría darse el caso de tener un equipo operando sin supervisión alguna.

Se tienen grandes beneficios haciendo uso de nuevas tecnologías, beneficios que nos facilitan la existencia y hacen de nuestras tareas algo más eficiente. Estos beneficios traen consigo retos, los cuales hacen del trabajo una aventura que me traerá múltiples obstáculos, pero a su vez un gran enriquecimiento.

1.1. Objetivo

El propósito final, es decir; el objetivo del proyecto es el siguiente:

"Diseñar y desarrollar un prototipo de una máquina que lleve a cabo la automatización de la cobranza y la administración de sesiones trabajo de diversos usuarios de una computadora; ésta con acceso a aplicaciones de escritorio, multimedia y de internet"

Vale la pena añadir que se está involucrado un objetivo personal, que es el de demostrar mis habilidades como Ingeniero al tener que trabajar con distintas disciplinas.

1.1.1. Metas específicas

Además del objetivo, existen una serie de metas a cumplir para resolver problemas, o retos, específicos dentro del proyecto. Estas metas fueron en parte designadas por parte de la empresa y otras fueron mis propias ideas.

- Desarrollar un sistema de créditos, los cuales el usuario puede agregar depositando monedas.

- Desarrollar un programa que administre los créditos y controle el tiempo que los usuarios tienen para utilizar el equipo de cómputo (tiempo de sesión).
- Realizar un programa que sea capaz de mostrarle a los usuarios el tiempo que le resta a su sesión
- Realizar un programa que impida la utilización del equipo cuando no haya créditos.
- Restringir el acceso del usuario para impedir que cambie cualquier clase de configuración del sistema.
- Desarrollar un programa que se encargue de que no exista rastro de que el equipo de cómputo fue utilizado con anterioridad, es decir; que lo restablezca a su estado inicial.

1.2. Contexto de mi participación profesional

He hablado un poco de qué es a lo que quise llegar, también expliqué un poco del dónde trabajé. Pero quiero que quede más que claro el tipo de problema al que me enfrenté, así como bajo que contexto lo realicé, explicando el alcance de éste y su posible relación con otros problemas. Explicaré qué tipo de necesidades se satisfacen y cuáles no, pero también sugiero la combinación de este proyecto con otros para incrementar el alcance al que puede llegar.

1.2.1. Definición del problema

La empresa DICIJE con su marca Multijuegos, localizada en Córdoba, Veracruz, se dedica a fabricar Sinfonolas Digitales y máquinas de videojuegos y me contrató por un periodo de seis meses para empezar un nuevo proyecto. Yo aún llevaba materias en la Facultad de Ingeniería cuando recibí esta propuesta. Aunque sabía que estudiar y trabajar no siempre es la mejor opción, en caso de no ser indispensable, acepté el trabajo con la condición de que este proyecto me sirviera, además de para adquirir experiencia, para titularme de Ingeniero Eléctrico y Electrónico. Ellos aceptaron y pusimos en marcha el trabajo.

Me contrataron como Ingeniero encargado de desarrollar el prototipo de un sistema para una máquina encargada de proporcionar recursos multimedia e Internet por medio de un equipo de cómputo utilizando software libre que

fuera capaz de cobrar por sí misma su utilización, manejando el tiempo de sesión en el cuál ésta dará acceso a sus funciones. Para esta máquina además se piden otros requisitos que en los objetivos se enumeraron. El método de cobranza será un tragamonedas que actuará junto con un microprocesador que comunica a este con el equipo de cómputo para administrar un sistema de créditos que permitirá a cualquiera que pague hacer uso de la máquina.

El fin de esta máquina es independizar a los equipos de cómputo del humano en los café Internet, necesitando el menor mantenimiento posible y maximizando la eficacia del negocio.

1.2.2. Alcance del proyecto

Conocer el alcance al que puede llegar este producto es prometedor; existen un sin fin de negocios a los cuales esta máquina les sería de gran utilidad, ya que las aplicaciones que puede tener son muchas. Se trata, en un principio, de una máquina que facilita la utilización de un equipo de cómputo con acceso a Internet, pero también existe la posibilidad de integrar este concepto a todo tipo procesos que requieran, como por ejemplo el de llenar formatos específicos para ciertos negocios o instituciones. La empresa y yo creemos que el alcance de este proyecto es ilimitado.

Además, en un futuro, la máquina tiene posibilidades de crecer en cuanto a integrar más servicios a ofrecer; ejemplos de esto es contar con periféricos como impresora, cámara web, escaner, etc. Puede crecer este proyecto tanto como uno quiera, hasta convertirlo en un equipo que satisfaga todas las necesidades informáticas del cliente.

El alcance personal que quiero para este proyecto es que sirva como base para que la empresa desarrolle un producto comercial a partir de éste.

1.3. Metodología del proyecto

Incluyo en este capítulo la metodología que utilicé para realizar este proyecto. Esta metodología se divide en tres etapas: en la primera, Estructuración del proyecto, establecí y ordené las tareas que iba a atacar; en la segunda, Criterios de diseño, definí las principales características que el producto final debía poseer; en la tercera, Definición de recursos, repaso los componentes y

herramientas necesarias para trabajar y construir el prototipo que tengo como objetivo.

1.3.1. Estructuración del proyecto

Lo primero que tomé en cuenta para empezar a trabajar, fue que debía llevar una estructura lógica el proyecto. Gracias a esto tomé la decisión de llevar una metodología que consideré apropiada y organizada. Entonces, la forma de iniciar mi trabajo sería la de hacer una lista de los puntos más importantes que debía llevar a cabo para cumplir con mi objetivo, y así poder definir qué hacer y con qué hacerlo. Debía tener en cuenta lo siguiente:

- Tener los conocimientos necesarios para comprender y trabajar en el proyecto.
- Realizar un desarrollo conceptual el cual me permita identificar los retos que el proyecto conlleva.
- Realizar el trabajo técnico que los retos me exigen.
- Integrar el trabajo realizado para conformar un producto; el prototipo del sistema para la máquina que cumpla con el objetivo y las metas establecidas.

Dados los puntos anteriores tuve entonces un camino a seguir para realizar el trabajo que se me asignó: esta lista es la que me guió durante la realización de este proyecto y me permitió llevar una línea lógica y estructurada de trabajo.

1.3.2. Criterios de diseño

Los criterios de diseño me fueron de utilidad para tener claridad y congruencia al trabajar en el proyecto, con ellos el enfoque prevalece y resulta de ayuda para que las piezas embonen y se puedan integrar para obtener un producto realizado.

Los criterios de diseño que dieron enfoque al proyecto son:

- Utilizar un sistema operativo *Linux* .
- Utilizar únicamente software libre (Existe software privativo dentro de *Linux*).

- Utilizar en la mayor medida posible herramientas nativas del sistema operativo con la finalidad de obtener un producto eficaz.
- Integrar tecnología ya utilizada por la empresa por motivos de ahorrar tiempo y gastos.
- Desarrollar el prototipo del sistema de tal forma que su utilización sea amigable y a la vez segura para el usuario.

1.3.3. Definición de recursos

Existen en la actualizad una cantidad numerosa de herramientas disponibles en el mercado, para el proyecto necesité componentes físicos y herramientas de software. Los componentes físicos pertinentes para este trabajo son los necesarios para construir un equipo de cómputo, es decir; una computadora, con la única diferencia de que además necesité los componentes que realizarían el cobro. Las herramientas de software son: el sistema operativo el cual correría en la computadora y los recursos de programación que me permitieron modificar el comportamiento de este sistema operativo para conformar el producto al que quise llegar.

Los recursos más relevantes para realizar este proyecto son:

- *Un equipo de cómputo:* Primeramente necesitamos una computadora para trabajar en el proyecto y el cual formará parte de la máquina o producto terminado, así que se necesita de distintos componentes de hardware para armar un equipo de cómputo, tales como: Monitor, CPU, tarjeta madre, memoria RAM, disco duro, teclado, ratón, entre otros.
- *El sistema operativo:* Teniendo en cuenta los requerimientos del proyecto, se elige *Linux* como sistema operativo, además hay que elegir la distribución a utilizar. En nuestro caso elegí una distribución de *Linux* llamada *Ubuntu*. Lo primero que se debe realizar es una instalación completa de este en nuestro equipo de cómputo. Teniendo esto, se debe conocer el funcionamiento y la estructura del sistema operativo, pues es la base de la parte de software de nuestro proyecto.
- *Recursos de programación:* Contando con el sistema operativo, queda la tarea de adecuar el funcionamiento de esta a partir de diversas herramientas; se pueden utilizar aplicaciones ya existentes para lograr un comportamiento determinado, o programar las necesarias para obtener el resultado deseado. Además se tiene que definir los lenguajes de programación que se adapten mejor al proyecto.

- *Otros periféricos:* Además de los componentes de hardware necesarios para conformar una computadora, se debe incorporar el mecanismo que convierta una moneda en una señal eléctrica que después debe ser interpretada, es decir; un tragamonedas y una interfaz con microcontrolador para comunicar a esta última con la computadora.

Capítulo 2

Marco teórico

2.1. *Linux*

Linux es un sistema operativo basado en *Unix*, y es construido bajo un modelo de software libre y siempre gratuito, aunque hay sus excepciones con desarrollo privado y comercial. El principal componente que define al sistema operativo *Linux* , es el llamado *Linux Kernel*. El primer kernel de *Linux* fue desarrollado por *Linus Trovalds*, un ingeniero en software finlandés, en octubre de 1991.

Este sistema operativo se convertiría en uno de los más versátiles existentes. Puede operar en diversos dispositivos: la computadora portátil es el típico lugar para encontrarlo, pero también puede hacerse uso y ventaja de *Linux* en teléfonos celulares, servidores, súper computadoras, *tablets*, routeadores de red, consolas de videojuegos, etc. En fin, en donde se requiera, puede este sistema operativo adaptarse. Como dato curioso, cabe destacar que alrededor de un 90 % de las supercomputadoras en el mundo corren bajo un sistema basado en *Linux* , y suele usarse en la gran mayoría de los servidores; por su seguridad, confiabilidad y rapidez.

Desarrollar bajo este sistema operativo arroja un mundo de posibilidades; posibilidades que nos permiten ser creativos, libres y eficientes. Creo que lo más importante en el desarrollo de este proyecto, es la libertad. Teniendo libertad para manipular al sistema operativo se abre un panorama ideal para el trabajo que, más adelante, se describirá. Basta con mencionar que se puede hacer uso

del lenguaje de programación que más nos convenga y además no se tiene que pagar por costosas licencias a la hora de dar el salto entre lo experimental a lo comercial. Dos de las buenas razones para elegir *Linux* como base son éstas anteriores.

2.1.1. De dónde vino *Linux*

Resulta que el origen de este sistema operativo se remota a los muy afamados laboratorios de *AT&T Bell*; en estos laboratorios se implemento un sistema operativo llamado *Unix*, en 1970. Este sistema operativo, debido a su portabilidad y disponibilidad, causó que se utilizara mundialmente en instituciones académicas y de negocios. Es muy estable. Otro efecto que tuvo fue que muchos desarrolladores lo tomaran como influencia para nuevos sistemas operativos.

En 1977 la Universidad de California, campus Berkeley, trabajó en un proyecto con la intención de obtener un sistema operativo tipo *Unix*, este fue el *Berkeley Software Distribution* (BSD). El problema que tuvo este proyecto fue que utilizaron una gran parte del código de *Unix*, el cual estaba patentado por *AT&T*; más tarde tendrían problemas legales, lo que complicaría la continuación del desarrollo de este sistema operativo. Actualmente existen versiones nuevas y libres de BSD, aunque no tienen mucho peso en la carrera de los sistemas operativos.

Más tarde, por el año de 1983, un señor llamado *Richard Stallman* –programador y activista de la libertad– inició el proyecto *GNU*, con el cual quería desarrollar un sistema operativo basado en *Unix*. Entre sus trabajos, escribió una licencia de software libre llamada *GNU General Public License* donde se describían sus condiciones para el código abierto. En los 90's, existía prácticamente todo el software requerido para realizar un sistema operativo completo, que fuera libre. Por alguna razón ese proyecto no llamó la atención a los programadores, por lo que quedó incompleto y, casi en su totalidad, olvidado.

Tras otros intentos menos destacados, en 1987 apareció un sistema operativo, y quizás el más popular antes de *Linux* , basado en *Unix*, llamado *MINIX* para usos académicos. Aunque el código estaba disponible para aquel que quisiera utilizarlo, las modificaciones y distribución estaban restringidas. Además, al ser un sistema diseñado para funcionar con procesadores de 16-bits, no se adaptaba a la arquitectura del sobresaliente y barato Intel 386, que

cada día adquiría más y más popularidad, debido a su diseño para computadoras personales. Estos factores fueron determinantes para que *Linus Trovalds* empezara su proyecto, después conocido como *Linux* .

2.1.2. Software libre

El término software libre comúnmente es relacionado con lo gratuito, no es del todo correcta esta concepción. Quizás la razón de lo anterior es que el concepto original de esto es *free software*, en inglés; donde la traducción que comúnmente se usa es correcta, pero la palabra *free* del inglés significa gratis, o gratuito, al mismo tiempo que significa libre. También cabe mencionar que el mismo proyecto GNU(Acrónimo recursivo para "GNU No es Unix"), que se ha encargador por muchos años de desarrollar y promocionar el software libre, recomienda entender este concepto de libre como de liberar, y proporciona el ejemplo: "... debería pensarse libre como «libre expresión» , no como en «barra libre»". Entonces, si al hablar de libertad, se debe ser aún más claro; por suerte, el proyecto GNU tiene muy bien definidos los puntos a los que esta libertad se refiere. Son cuatro y son los siguientes:

- La libertad de ejecutar el programa, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo trabaja el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Si lo hace, puede dar a toda la comunidad una oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición necesaria para ello.

Un software es libre si todos los usuarios tienen las libertades anteriores y no tiene que pedir o pagar por un permiso para hacer uso de estas.

La libertad de ejecutar el programa para cualquier propósito significa que su usuario puede utilizarlo en cualquier sistema computacional para cualquier trabajo o fin, sin estar obligado a comunicar esto a su programador o a quien sea propietario. Aquí el obligado que nos importa y del que hablamos es el

de los usuarios, no de quien posee el software o de quien lo programó. Nadie puede imponer los propósitos de uso de este software.

El usuario tiene la libertad de hacer cualquier modificación al programa sin necesidad de pedir permiso, además puede hacerlo en privado si es que lo desea. Si se llegaran a publicar cambios del software en cuestión tampoco se está obligado a notificarle el hecho al propietario o a alguna persona en específico.

La libertad de redistribuir nos habla de la posibilidad con la que se cuenta de entregarle el software o el código a quien sea y como sea, pero incluyendo los binarios o ejecutables del programa (si el lenguaje de programación no incorpora esta característica no es necesaria la condición anterior), también podemos hacer nuestras propias versiones del software; modificando el código y utilizándolo como se nos ocurra, solo hay que tener en cuenta que al distribuirlo se pide que se entregue este también.

El uso comercial

Las cuatro libertades que se han especificado son exactamente las que GNU proporciona, pero hasta ahora no se ha tocado el tema de la comercialización de un producto de software libre. Para que un software sea denominado libre, necesariamente debe cumplir con los puntos ya expuestos, por lo que se puede observar que nunca se habla de precios, costos o dinero. Se puede entonces preguntar ¿Es posible entonces vender el software libre? La respuesta inmediata es sí. La respuesta detallada es que por más que a un software se le asocie un costo puede, o no, ser libre; que un software sea gratuito tampoco significa que sea libre. Hay que recordar que las libertades que ya se nombraron. Otro punto importante es que uno puede vender un equipo de cómputo, con un sistema operativo libre y con alguno que otro software privativo sin ningún problema: la razón de esto es que uno vende hardware y software privativo, quizás, así el producto entero no es libre, pero como la «libertad 0» indica, el sistema operativo –la parte libre de este caso (producto)– puede ser ejecutado para cualquier propósito. Así entonces queda explicado cómo es que se puede utilizar el software libre para un uso comercial.

2.2. Lenguajes de programación

Los lenguajes de programación son idiomas desarrollados para comunicar al ser humano con una máquina, tal como puede ser una computadora, un robot, un microcontrolador, entre otras. Estos idiomas tienen la capacidad de expresar procesos, o instrucciones, que se requieran para controlar un comportamiento lógico de una máquina; esto se puede traducir en operaciones físicas, como lo son las de un robot, lógicas-matemáticas, manejo de información, una combinación de éstas o todas en conjunto. Por lo general son la manera de indicarle a una máquina que lleve a cabo uno o varios algoritmos con el fin de proveer una solución a un problema.

Existe una variedad amplia de lenguajes de programación enfocados a un gran variedad de aplicaciones, sin embargo todos ellos están conformados por una serie de símbolos y reglas que definen su estructura en particular. La clasificación de estos idiomas es muy extensa y difícilmente uno solo se encontraría dentro de una sola categoría; entre los tipos de lenguajes de programación más populares se encuentran:

- Lenguajes estructurados.
- Lenguajes orientados a objetos.
- Lenguajes ensamblador.
- Lenguajes tipo guión (*script*).

Los lenguajes de programación que fueron relevantes para el proyecto caben dentro de las categorías anteriores salvo por los que son orientados a objetos.

2.2.1. Lenguaje C

C es un lenguaje de programación estructurado, diseñado originalmente por Dennis Ritchie, entre los años 1969 y 1973, quien trabajaba en los afamados laboratorios *Bell* de *AT&T*. Una de las principales ventajas que encontró este lenguaje por encima de otros existentes en la época fue que había sido diseñado para cubrir necesidades del día a día con una sintaxis amigable; dejando atrás otros propósitos como ser un lenguaje de prueba o demostrativo.

Este lenguaje requiere de un compilador, que no es más un programa informático que traduce un lenguaje a otro. Los compiladores modernos ayudan

también a detectar errores en el código de un programa y hasta logran hacer optimizaciones al código mismo.

Se suele utilizar este lenguaje hoy en día en todo tipo de aplicaciones, ha influenciado a otros lenguajes de programación probablemente más que ningún otro. Los programas hechos con este lenguaje se caracterizan por ser veloces, y si se tiene la habilidad suficiente pueden ser muy eficientes también, ya que el lenguaje permite que se maneje la memoria que se le asigna al programa de manera directa.

2.2.2. Lenguaje Bash

Bash es más que un lenguaje de programación, pues es también el intérprete de una línea de comandos; de UNIX para ser precisos. Esto quiere decir que todos los sistemas operativos que utilicen UNIX como base entienden este lenguaje, por ejemplo *Linux* y *Macintosh*. Dentro de estos sistemas operativos existen interfaces que le permiten a uno introducir comandos en lenguaje *Bash*. Es un hecho entonces que este es un lenguaje creado para interactuar de manera directa con un sistema operativo, lo que nos proporciona numerosas ventajas si trabajamos sobre uno de ellos. Lo interesante es que además de ser esto, también es un lenguaje tipo guión (*script*), ya que existe la posibilidad de crear un programa basando en este lenguaje y después ser ejecutado e interpretado por el mismo sistema operativo.

Las principales ventajas que nos proporciona este lenguaje son:

- Interpretación directa por parte del sistema operativo con el que se trabaje.
- Posibilidad de acceder a una gran variedad de funciones propias del sistema operativo, que de forma externa sería más complicado.
- Facilidad al momento de programar cuestiones que modifiquen el comportamiento del sistema operativo, ya que esa es su función.
- Documentación extensa respecto a este lenguaje.
- No existe la necesidad de utilizar un compilador ya que es un lenguaje interpretado al momento de su ejecución.

  Bash fue creado por Brian Fox para el proyecto GNU, y fue liberado en 1989. Antes de *Bash* el intérprete que se utilizaba en las plataformas UNIX era uno llamado *Bourne*, denominado así en honor a

su creador Stephen Bourne quien trabajaba para los laboratorios *Bell*. *Bash*entonces remplazó a *Bourne* para convertirse en el lenguaje que por defecto utilizarían las plataformas UNIX.

2.3. Tragamonedas y microcontrolador

Para efectuar el cobro por la utilización de la máquina en cuestión recurro a un mecanismo ya adoptado por la empresa; éste es un tragamonedas, que junto con un microcontrolador logran realizar esta tarea. El tragamonedas es el que recibe monedas y el microcontrolador es el que le comunica a la computadora cuando ésto ha sucedido (ver figura 2.1).

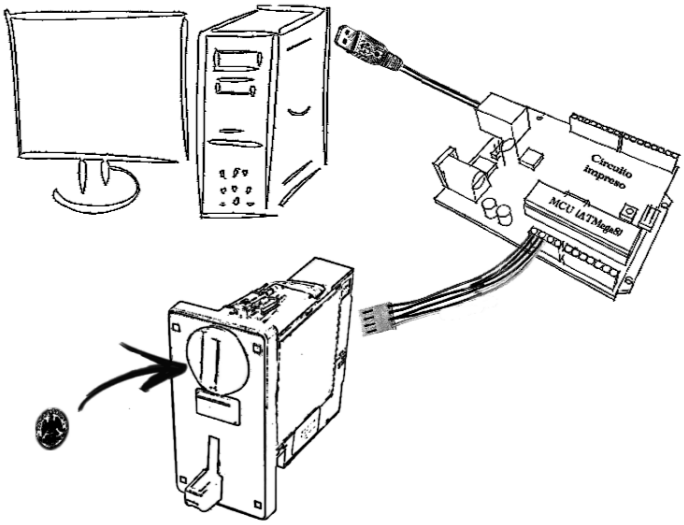

Figura 2.1: Esquema de tragamonedas y microcontrolador.

2.3.1. Tragamonedas

Existe más de un método para realizar un proceso de cobro; desde los más primitivos, como el de que una persona reciba dinero y vigile la utilización de un servicio o producto, hasta dispositivos que aceptan tarjetas bancarias para realizar el cobro y el control de la utilización del servicio o producto sea por

medio de software. Para este proyecto utilizo un método similar al segundo, con la diferencia de que el dispositivo acepta únicamente monedas de una sola denominación.

El funcionamiento de este dispositivo es simple y fácil de interpretar. El tragamonedas debe ser calibrado para la denominación de la moneda que se quiera utilizar. La forma de calibrar un tragamonedas puede variar dependiendo del modelo y sus funcionalidades, el que se utiliza en este proyecto la calibración se realiza simplemente colocando la moneda que se desea utilizar dentro de un compartimiento y el tragamonedas hace el resto.

El tragamonedas se alimenta con una fuente de poder, generalmente de 12 Volts, y cuando una moneda es insertada hace una rápida comparación para ver si ésta coincide con la que se ha calibrado el dispositivo; en caso de ser negativa la comparación devuelve la moneda y en caso de ser positiva emite un pulso eléctrico para que pueda ser interpretado.

Ejemplo 1 (Moneda insertada en el tragamonedas utilizado en el proyecto)**.** *Ejemplifico la señal eléctrica resultante cuando el tragamonedas recibe una moneda y ésta es aceptada (ver figura 2.2). Algunas características del tragamonedas son:*

- *Es alimentado con 12 Volts.*
- *La salida es una señal eléctrica constante a 12 Volts.*
- *El pulso eléctrico emitido es a 0 volts.*

2.3.2. Microcontrolador

Los microcontroladores, o MCU, son circuitos integrados programables. Éstos pueden ejecutar instrucciones guardadas en su memoria. Los MCU se utilizan para resolver problemas específicos y generalmente cumple con una única tarea. Los MCU contienen las partes fundamentales de una computadora en su interior, estas son: unidad de procesamiento, memoria y periféricos de entrada y salida para comunicarse con otros dispositivos.

Los MCU, de fábrica incluyen una numerosa cantidad de funcionalidades con el fin de facilitar la resolución de diversos problemas, por ejemplo; los MCU

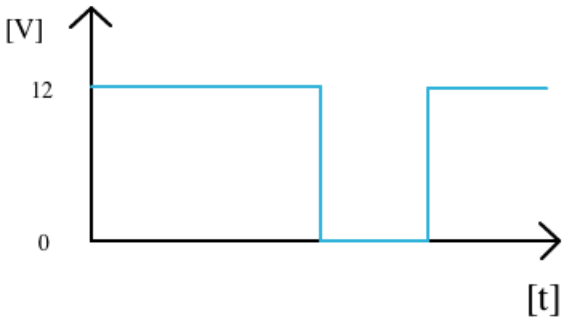

Figura 2.2: Pulso eléctrco al aceptar moneda.

suelen incluir un reloj interno, modulación de ancho de pulsos (PWM), convertidores analógico-digital, así como convertidores digital-analógico, múltiples entradas y salidas, entre otras.

El microcontrolador es una pieza fundamental en el desarrollo de este proyecto debido a que se encarga de comunicar a la computadora que ha recibido un pulso eléctrico proveniente del tragamonedas, es decir; que una moneda ha sido insertada y aceptada. La empresa trabaja con esta tecnología en distintos productos y ya cuenta con una solución a este problema.

Capítulo 3

Desarrollo conceptual

En esta sección describo algunos hechos respecto al proyecto; el qué, y el cómo, pretendí llegar al producto final. En este apartado se describirán los primeros pasos que llevé a cabo para construir la máquina que sería capaz de cobrarle al usuario por la utilización de un equipo de cómputo con las utilerías multimedia básicas para navegar en Internet, procesar texto, hacer presentaciones, hojas de cálculo, etc. (las que el administrador desee).

El proceso de construir esta máquina tuvo complejidades de distintas índoles que en las próximas paginas se van a describir, en lo que más me concentro es en explicar el funcionamiento general del producto; desde el sistema operativo y el software que se utilizó –los programas ya existentes de los que saqué provecho y los que programé de acuerdo a las necesidades que surgieron– hasta el hardware que hace posible el cobro por la utilización del equipo. Entonces, en este apartado no se encuentra el ejercicio más técnico que desarrollé, en vez de eso el enfoque es conceptual, quedándonos con lo fundamental: Ideas, recursos, diagramas de flujo, descripciones generales.

3.1. Elección de herramientas

Quizás lo que le antecede a cualquier clase de desarrollo durante mi proyecto es la toma de decisiones respecto a las herramientas que se utilizaron para poder trabajarlo, estas herramientas definieron la investigación que fue necesaria. Las decisiones de las que hablo fueron desde el sistema operativo

que se iba a utilizar, los lenguajes de programación que me ayudarían a lograr que la máquina funcionara del modo requerido, cuestiones de seguridad para proteger nuestro producto, hardware, etc.

3.1.1. El sistema operativo

Este fue sin duda un tema importante, la decisión que se tomase respecto al sistema que operaría a la máquina es crucial para saber cómo trabajar. Afortunadamente esto no causó mayor problema: desde un principio se me sugirió que el proyecto debía involucrar la mayor cantidad de software libre posible, con esto, la mejor opción que existe, tratándose de software libre, es, sin duda, *Linux* . Por ello la importancia que ocupa *Linux* en el marco teórico.

La decisión de la distribución específica de *Linux* que usaría la tomé de acuerdo a experiencia personal. Hacía algunos años acostumbraba usar y experimentar con una distribución bastante conocida y amigable llamada *Ubuntu*. La principal característica con la que cuenta esta distribución de *Linux* , que me llamó la atención fue la de contar con un desarrollo constante; enfoca el desarrollo de ésta a la facilidad de uso y frecuentemente es actualizada.

Con la decisión tomada, sería cuestión de trabajar en el funcionamiento del sistema; con esto quiero decir que hay que hacerlo lo más estable posible y adecuarlo de tal forma que se comporte como un sistema autosuficiente en cuanto al manejo de sesiones, cobranza, seguridad, etc. Más adelante se describirá este funcionamiento.

Las principales ventajas de utilizar *Linux* en el proyecto son:

- Cuenta con un sistema de archivos que maneja permisos, característica que provee seguridad.
- La flexibilidad que se tiene al ser un sistema operativo abierto abre una infinidad de posibilidades en cuanto a modificar el comportamiento del sistema operativo.
- La rapidez con la que trabaja es fundamental ya que permite tener ahorros en hardware del equipo de cómputo y un iniciado del sistema en cuestión de segundos.
- Existe una comunidad muy grande que da soporte y trabaja entorno a este sistema operativo, lo que provee buena documentación.

3.1.2. Lenguajes de programación

Sin duda uno de los puntos más importantes de la definición de recurso; para fines prácticos el lenguaje que fuera a utilizar no debía ser necesariamente el más eficiente, puesto lo que se buscaba era un prototipo inicial funcional, pero por otro lado el lenguaje tenía que conocerlo lo suficiente para trabajarlo. Otra cuestión a considerar es que no se necesitan procesar muchos datos o hacer cálculos complejos. Después de un poco de investigación llegué a la conclusión de que poseía conocimiento suficiente y utilizaría dos de los lenguajes que más he trabajado, además de que operan a la perfección sobre *Linux* y *Ubuntu*. Los lenguajes que más utilicé en este proyecto fueron *C* y *Bash*. *C* lo aprendí bastante bien durante mi carrera en la Facultad y *Bash* con experiencia personal con *Ubuntu*.

Los lenguajes *C* y *Bash* me ayudarían en gran medida también por la facilidad que presentan al programarse y por ser integrables con procesos propios del sistema operativo, es decir; con ellos pude controlar la forma en que éste se comporta. La gran diferencia que existe entre ellos reside en que *C* crea un ejecutable al momento de ser compilado y *Bash* es interpretado al momento de su ejecución sin crear un archivo ejecutable. Este último funciona con archivos llamados *scripts* que son archivos de texto con el código escrito en ellos, pero con permisos de ejecución –este permiso se lo otorga el usuario para que el sistema operativo, en nuestro caso *Linux* , pueda ejecutar sus instrucciones– para funcionar como pequeños programas. La gran ventaja de este lenguaje es que es el mismo que utiliza *Linux* para realizar muchas de las tareas mismas del sistema operativo; teniendo esto es fácil estudiar cómo trabajaron con *Bash* para correr muchas de las funciones de arranque, cambio de sesión, etc.

3.1.3. Periféricos

Utilicé hardware básico para la operación del equipo de cómputo: un teclado para la introducción de texto, un ratón para maniobrar el puntero, un monitor para ver lo que sucede en el entorno gráfico, un controlador de red inalámbrico (*NIC*) para hacer uso del Internet, etc. Lo que no es usual, y en lo que quiero hacer énfasis, es el uso de un sistema de cobro por moneda; el que nos permitirá introducir créditos a la máquina para hacer uso de ella.

El periférico que permite interactuar con una moneda introducida es, en realidad, la integración de dos dispositivos. El primero, que es el que un

usuario vería, un tragamonedas que funciona mandando una señal eléctrica cuando una moneda, de una denominación en específico, es introducida. La otra parte del periférico es un microcontrolador, que será el que interprete la señal eléctrica mandada por el tragamonedas, la interprete y transmita de tal manera que el sistema operativo en la computadora la pueda entender; esto se hace a través de un puerto USB.

Los periféricos que se necesitan añadir al equipo de cómputo son:

- Teclado.
- Ratón.
- Monitor.
- Tarjeta de red inalámbrica (WiFi).
- Tragamonedas con microcontrolador.

3.2. El funcionamiento de la máquina

Para poder abordar el tema acerca del funcionamiento real de la máquina que desarrollé, es necesario entender cómo fue el enfoque que se le dio al proyecto pensando en un producto final; que sería utilizado por un usuario cualquiera con conocimientos mínimos de computación. Este enfoque me permitió tener metas claras a la hora de programar y construir la máquina: en vez de pensar cómo lograr que mis programas hicieran su tarea solamente, me concentré en cómo lograr que mis programas le dieran la menor complicación posible al usuario. Lo anterior me obligó a centrar la idea de la máquina como un producto que alguien, que no fuera yo, lo utilizaría, en vez de dar por hecho que con solo funcionar sería suficiente.

3.2.1. Diagrama conceptual de la máquina

A continuación se muestra un diagrama de flujo de la idea general del funcionamiento de la máquina. Este es apenas una muestra de lo que se quería lograr para el producto final:

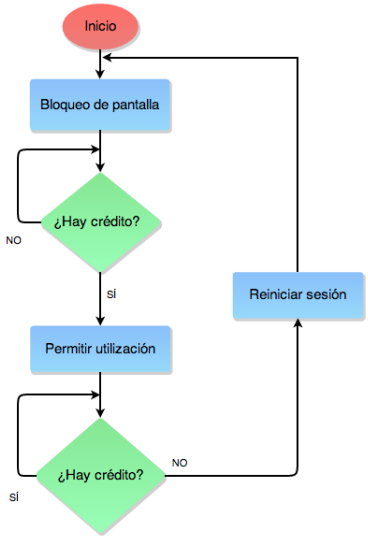

Figura 3.1: Diagrama general de flujo operativo.

Explicación del diagrama

El usuario al estar frente a la máquina lo primero que verá es una pantalla de bloqueo, o *lock screen*. Esta pantalla puede contener alguna imagen o video promocional, o simplemente indicar que se requiere de una moneda para acceder; esta pantalla de bloqueo no permite al usuario realizar función alguna de la máquina y permanecerá hasta que el usuario introduzca alguna moneda. Después, si la requerida moneda es introducida, la pantalla de bloqueo automáticamente desaparecerá y permitirá al usuario ver el escritorio del sistema, que le mostrará las aplicaciones disponibles, quedando lista la máquina para su utilización. Cabe agregar que por cada moneda introducida el sistema debe agregar crédito al sistema que administra el tiempo de duración de la sesión, lo que resultará en tiempo agregado.

3.2.2. Sobre el sistema y el escritorio

Características del escritorio y sistema al que el usuario accede:

- El sistema dará acceso al usuario de utilizar la máquina y ver el escritorio únicamente después de que este inserte una moneda al tragamonedas.

- Escritorio sencillo y visualmente agradable, con accesos directos a las aplicaciones más utilizadas. Estas aplicaciones permiten al usuario realizar tareas comunes como: Navegar por Internet, procesar texto, crear presentaciones, dibujos, jugar y más.
- El escritorio que el usuario ve es limpio, es decir; cada vez que un usuario inicia una sesión en el equipo, éste hace una limpia de los archivos, iconos, configuraciones, o cualquier cosa que el usuario anterior haya creado o modificado. Siempre que se inicie sesión se tendrá un escritorio completamente restablecido, tal y como cuando se prendió la máquina por primera vez.
- Se despliega un reloj contador del tiempo restante de la sesión; este reloj siempre está disponible, el usuario no puede cerrarlo, pero sí moverlo o esconderlo.
- El entorno no permite al usuario acceder a configuraciones del sistema ni permite modificar archivos que puedan causar daños.
- Al acabar el tiempo de sesión, se muestra un recuadro donde te advierte que tienes un tiempo determinado para insertar otro crédito o guardar tu trabajo en Internet o memoria externa. Si se inserta crédito continua la sesión, si no se cierra.

Más adelante se explicarán a detalle las características anteriores, o más bien dicho se explicará cómo es que se logró tales comportamientos del sistema.

3.3. Estados de la máquina

Ya establecí que la gran mayoría de los programas que se realizarían serían *scripts*; en el siguiente capítulo se hablará de ellos de manera más específica. Por ahora me enfocaré a explicar qué pasa con la máquina en cada etapa de su utilización, desde el momento de encendido, el momento en que es utilizada y hasta cuando la sesión es terminada.

3.3.1. Encendido y la máquina bloqueada

Al presionar el botón de encendido del equipo, *Ubuntu* iniciará de forma regular, es decir; El *BIOS* del sistema arranca las tareas específicas del hardware, se realiza la iniciación del kernel, se inicia el gestor gráfico, etc. En el último paso, al iniciarse el gestor gráfico de *Ubuntu*, que es *Gnome*, se hace el

inicio de sesión y se inicializan las aplicaciones definidas por mi. En estas dos etapas deben ocurrir dos cosas que necesitamos:

- Durante el inicio de sesión, se definió que se ejecutará una serie de *scripts* que se encargan de hacer una limpia al escritorio, eliminando todo archivo basura y modificaciones que algún usuario anterior haya podido realizar, para tener siempre el sistema tal y como se definió por primera vez.
- Entre las aplicaciones definidas para el arranque se encuentrán otros *scripts* que cumplen diversas funciones para bloquear el sistema y esperar el ingreso de un crédito o una moneda. En este *script* se realizan funciones tales como:
  - Iniciar o reiniciar el programa que se encarga de manejar los créditos y el tiempo de la sesión. Éste revisa todo el tiempo cuál es el estado de los créditos en el equipo.
  - Ejecuta una aplicación que despliega una imagen en pantalla completa. Esta imagen puede ser publicitaria o simplemente indicar que se debe insertar una moneda para utilizar el equipo.
  - Ejecuta un programa que bloquea cualquier clic del ratón. Esto nos ayuda que el usuario no pueda ingresar a ningún tipo de opciones por medio de este periférico.
  - Ejecuta una serie de comandos que una a una bloquean todas las teclas del teclado que puedan ser utilizadas para iniciar el sistema o ingresar a algún tipo de menú.

  Como se puede apreciar, en esta etapa el sistema queda inutilizable hasta el momento en que estos bloqueos se eliminen. Lo anterior sucedería cuando el equipo detecta que una moneda, o crédito, se ha ingresado.

3.3.2. La máquina en operación y desbloqueada

Está claro que mientras no se haya ingresado una moneda a la máquina, el sistema seguirá bloqueado. Ahora analizaré que sucede cuando hay al menos un crédito vigente.

Cuando por primera vez –es decir; cuando la pantalla está bloqueada– y el sistema detecta que se ha ingresado una moneda en el equipo, este inmediatamente responderá eliminando la pantalla de bloqueo y restableciendo el

ratón y el teclado, mostrará instantáneamente el escritorio limpio y listo para usarse. En este momento ocurrirá lo siguiente:

- Una vez mostrado el escritorio y restablecidos los periféricos (teclado y ratón), se mostrará una clase de reloj o contador de tiempo que le indicará al usuario cuánto tiempo le resta de sesión. Este reloj está programado en *C* utilizando librerías *SDL*. No es posible cerrar el reloj, y en caso de que esto sucediera el contador de tiempo de la sesión seguiría activo; esto es porque el contador real se ejecuta en el sistema de forma invisible.

- El *script* a cargo de manejar el tiempo constantemente actualiza el estado de los créditos existentes, por medio de un banco de créditos, por lo que si el usuario ingresa otra moneda a la máquina, este contador se actualizará (incluido el reloj) mostrando el nuevo tiempo restante de la sesión. Este es el mismo programa que se encarga de restar créditos del banco de créditos una vez que haya transcurrido un determinado número de minutos.

- Cuando el contador llegue a cero, o de otra forma; que se hayan terminado los créditos, se ejecutará otro *script* que despliega una ventana emergente avisando que la sesión está por cerrarse y que, en caso de querer seguir en ella, debe insertar otra moneda. Esta ventana muestra además una barra de progreso que tiene como función indicar al usuario que tiene un tiempo limitado para lo anterior.

- En caso de que el usuario inserte una moneda la ventana emergente desaparecerá, el reloj se actualizará y se podrá seguir utilizando la máquina por un tiempo determinado.

3.3.3. Reinicio de sesión

En caso de que no se inserte crédito cuando la ventana emergente que anuncia que los créditos se han terminado se encuentre desplegada, ocurriría lo siguiente:

1. Se cerrarán de manera forzada todos los programas que se encuentren abiertos.

2. La sesión activa se termina.

3. La sesión se inicia de nuevo.

4. Los archivos y configuraciones del sistema operativo de la sesión en cuestión se restablecen.

5. La máquina queda nuevamente bloqueada y en espera de una moneda nueva.

Esta es la forma en la que una sesión activa termina y es reiniciada, restableciendo el escritorio en su totalidad. Digo escritorio pues es a lo que un usuario tendría acceso.

Capítulo 4

Desarrollo técnico

Después de ver de forma general el concepto y el funcionamiento de la máquina, es tiempo de ver más a detalle lo que hice en este proyecto. En este capítulo se describirán las operaciones de la máquina desde el sentido más técnico pero siempre conservando una claridad para entender qué se ha hecho. Lo más obvio es adentrarse en el código puro y a eso llegaremos, pero algo relevante que no hay olvidar es el cómo se entrelazan, comunican y operan los programas realizados para lograr este comportamiento deseado. El código completo de cada *script*, o programa, no se mostrará como tal en este apartado (salvo los que sean muy cortos), sin embargo es necesario en algunos casos hacer referencia a partes de este código para entender mejor cómo operan los programas. El código de cada programa y *script* se encuentra en el apéndice.

4.1. Preparación del sistema operativo

Dentro de la distribución de *Linux* , *Ubuntu*, existen diversas configuraciones del sistema a las cuales se puede acceder si se tiene permisos de administrador; como trabajé sobre una instalación nueva de esta distribución, tuve toda la libertad de manipular estas configuraciones a mi antojo, también saqué provecho de esto para instalar algunas aplicaciones que me facilitarían labores en algunos casos.

En las siguientes líneas describiré cada uno de los pasos que seguí para obtener un sistema operativo más a mi favor, que estuviera preparado para realizar todas las tareas y ejecutar todos los programas que necesité para

poner en marcha el producto terminado. Aquí explicaré qué configuraciones manipulé qué aplicaciones instalé y qué se necesita para programar y construir este proyecto.

4.1.1. Las aplicaciones y librerías de programación

En materia de programación en *Bash* no se necesita de librerías o aplicaciones adicionales dado que el propio sistema operativo funciona con este lenguaje, y al ser interpretativo (que no se compila) la propia consola del sistema es capaz de ejecutar los programas (*scripts*).

Para *C*

Otra gran ventaja de esta distribución es que utiliza un sistema de administración de paquetes por repositorios llamado *APT*, este viene de su distribución madre *Debian*, y consiste en conectarse a una base de datos remota que contiene un gran número de librerías y aplicaciones para su fácil instalación.

Con un simple comando en la terminal del sistema:

```
$ sudo apt-get install build-essential $
```

Aquí se le indica a *Ubuntu* que instale las librerías y aplicaciones necesarias para programar y compilar en *C* , al mismo tiempo estamos instalando lo necesario para compilar otra clase de paquetes y lenguajes, pero con esto es suficiente para empezar.

Además de instalar estas librerías básicas de *C* , tuve que agregarle a este otras para desarrollar la interfaz gráfica del reloj contador de tiempo. Estas librerías son las SDL, que también de forma sencilla las instalamos en el sistema:

```
 $ sudo apt-get install libsdl1.2-dev libsdl1.2debian $
$ sudo apt-get install libsdl-ttf1.2 libsdl-ttf1.2-dev $
```

Las primeras dos librerías son las de SDL propiamente, y las de abajo son una extensión para utilizar fuentes de texto *ttf*. Como se aprecia, instalar esta clase de paquetes es bastante sencillo con este sistema de gestión de paquetes basado en repositorios.

Aplicaciones adicionales

Son únicamente tres las aplicaciones que utilicé que no estuvieran instaladas por defecto en mi distribución de *Linux* : Geany, Feh y Pessulus.

Geany es un entorno de programación –como lo es Eclipse, Dev C++, CodeBlocks, etc– que es amigable de usar sin dejar de ser muy configurable. En esta aplicación sobre todo programé en los lenguajes *C* , y *Bash*. Para instalar Geany:

```
$ sudo apt-get install geany $
```

Feh es un visualizador de imágenes muy básico; la gracia de este es que me permite desde la consola de comandos desplegar imágenes en pantalla completa o con alguna otra propiedad que desee. Es rápido y efectivo, para instalarlo:

```
$ sudo apt-get install feh $
```

Pessulus es una pequeña aplicación muy útil, esta es un poco distinta a las anteriores pues poco se utiliza pero es de gran ayuda. Resulta que de manera gráfica me permite bloquear el acceso que pueda tener el usuario a las configuraciones del panel de escritorio, impide que el usuario termine la sesión, bloquea una que otra aplicación que pueda interferir con nuestros propósitos, etc. No lo hace todo, sin embargo hace una gran labor por nosotros. Para instalar Pessulus:

```
$ sudo apt-get install pessulus $
```

4.1.2. Configuraciones de cuentas y otras

Cuentas de usuario

Definir el tipo de sesión en la cual se va a trabajar es fundamental, esto es debido a lo siguiente: al tener en mente que la máquina va a ser utilizada por múltiples personas, es necesario restringir el acceso que tienen estas a las configuraciones o archivos del sistema. Este caso es conveniente tener dos cuentas al menos para facilitar la construcción del proyecto; por una parte se

debe contar con una cuenta de administrador para poder realizar todos los cambios necesarios al sistema operativo y por otro lado es necesario una cuenta de "escritorio", "invitado" o alguna que restrinja estos dichosos accesos. Lo definí así con el fin de tener una cuenta de administrador para hacer alguna modificación y otra donde normalmente operará y desde donde no se pueden modificar el comportamiento de la máquina.

Después de una instalación limpia de *Ubuntu*, tendremos una cuenta de administrador, solo se requiere que después se cree la nueva en las configuraciones del sistema.

Sesión

Lo siguiente es indicarle al sistema que al encender la computadora entre de forma automática a la cuenta de invitado (la que tiene los accesos restringidos). Con lo anterior se logra que siempre la máquina acceda a la cuenta y a la sesión que se desea. Los usuarios no tendrán opción de elegir nada acerca de las cuentas del sistema.

Esta configuración se encuentra dentro de las propiedades de inicio de sesión, de igual manera que la anterior, es muy sencillo implementarla.

Protector de pantalla y Energía

Este aspecto del escritorio es mayormente estético y no hay grandes cambios por realizar, sin embargo es importante configurar un par de cosas:

- Deshabilitar la opción de pedir contraseña a la hora de regresar del protector de pantalla al escritorio.

- No permitir que en ningún periodo de tiempo la sesión termine o se llegue a bloquear. Así mismo se debe cancelar la opción de poner al sistema en modo dormido.

4.2. Programación de tareas específicas

He preparado esta sección para explicar cada una de las tareas que el sistema debe cumplir, desde el conteo de créditos, el bloqueo de la pantalla, el contador de tiempo, hasta el restablecimiento del escritorio. Cada tarea

participa de forma fundamental en el comportamiento de la máquina y es determinante para un buen funcionamiento general. El conjunto de estas tareas, que serán programas en *C* y *scripts*en *Bash*, cumplirán su cometido para integrar y formar el proyecto en lo que se pretendió desde un principio.

Los algoritmos que utilicé para desarrollar estas tareas son todos propios, con excepción de uno que más adelante describiré. Hay tareas de todo tipo, que pueden ser desde modificar el comportamiento nativo del sistema operativo hasta agregar otras funciones a éste, pero lo que hace interesante al proyecto es la interacción que tienen cada uno de los pequeños programas que realicé y que en juntos conforman a la plataforma, o sistema, que el objetivo del proyecto describe.

Las primeras tareas que describiré son las fundamentales para el funcionamiento completo de la máquina y dejaré a un lado la estética del producto final, así como algunos refinamientos. Primero explico lo que tiene que ver con la programación y en otra sección describiré los pasos que seguí para los acabados finales.

4.2.1. Restablecimiento del escritorio

Problemática

Esta fue una de las metas más importantes que se me solicitó como característica de la máquina; lo que se me pidió básicamente fue que cada vez que un nuevo usuario, que utilizara la máquina, debería encontrarse con un escritorio limpio, libre de archivos basura de otro usuario de la máquina y con las configuraciones del sistema intactas.

El gran inconveniente es que *Ubuntu* no cuenta con esta opción en ninguna de sus versiones, por lo que tuve que ingeniármela para resolver este dilema y cumplir con esta meta.

Solución

Con un poco de investigación y con algo de conocimiento que tengo sobre los sistemas *Linux* y *Ubuntu*, determiné que todas las configuraciones del usuario se encuentran almacenadas en el directorio por excelencia de cada usuario, es decir; en el directorio *home*.

Ejemplo 2 (Localizar el directorio *home* en un sistema operativo *Linux*)**.**

Supongamos que nuestra cuenta de usuario (usuario del sistema operativo), de la cual se quiere conocer el directorio home, se llama Omar.

Entonces, el directorio que se busca es:

```
/home/Omar
```

Este directorio, además, será el único al que el usuario de la máquina tendrá permisos de escritura. El símbolo "/ "hace referencia al directorio raíz, por lo que de éste derivan todos los demás; incluyendo a /home/Omar.

Sabiendo esto, pensé, que la forma de mantener limpio el escritorio sería logrando que cada vez que un usuario iniciara sesión, el mencionado directorio */home/Omar* contuviera exactamente lo mismo. La manera de solucionar esto sería así; programar un par de *scripts* que me permitirían restablecer el directorio cada vez que se desee.

Son dos programas los que hice para lograr esto:

Script 1 (Respaldo del directorio *home* "bkp.sh")**.**

Tener un respaldo de tal y como es el directorio /home/nomcuenta (nomcuenta es el nombre de la cuenta o usuario del sistema), que es el de la cuenta con permisos restringidos, me permitiría en todo momento regresar al contenido original de éste. La manera por la cual respaldé esto es haciendo un programa, script, que comprima todo el directorio en un archivo.

He aquí el script:

```
1 #! /bin/bash
2
3 rm -rf /root/Desktop/backup.tar
4 tar -cpPf /root/Desktop/backup.tar /home/cliente
```

Este programa lo llamé "bkp.sh".

Descripción del *script* 1

La línea 1 solo contiene la indicación de que el archivo es un *script* y está escrito en lenguaje *Bash*. Las líneas 3 y 4 son las que hacen el trabajo de respaldar el directorio completo –en mi caso fue */home/cliente*– conservando ciertas características.

La línea 3 elimina un archivo comprimido "backup.tar" que es el que contiene respaldo; esta operación la realiza siempre que se quiera respaldar con el fin de nunca tener duplicados. El comando "rm'se encarga de eliminar archivos y las opciones "-rf" indican que se debe eliminar hasta subdirectorios si fuera necesario, es una manera de forzar que se borre por completo .'

En la línea 4 se puede observar que indicamos dos rutas dentro de la instrucción, la primera es el archivo destino (el que estará comprimido) y la segunda es el directorio el cual se va a comprimir. El comando "tar" se encarga de hacer la compresión y las opciones "-cpPf" son las que le indican a la instrucción de que el directorio que se va a comprimir conserve todos los permisos del sistema, incluyendo a los archivos que este contiene y a los subdirectorios. Hay que recordar que al decir permisos, hablamos de los que nos indican quienes pueden a los archivos leer, escribir/modificar, ejecutar, etc.

Script 2 (Restaurar el directorio *home* "rst.sh")**.**

Una vez que se tiene tenido listo el escritorio y sus configuraciones y lo hemos respaldado con el script 1, nos llega la tarea de restaurarlo, que es el problema que inicialmente me plantee. Para el caso de este pequeño programa, se usa el respaldo que ya se ha creado descomprimiéndolo en el lugar indicado. Aquí el programa completo:

```
1 #! /bin/bash
2
3 rm -fR /home/cliente
4 tar -xpPf /root/Desktop/backup.tar
```

Nota: "-rf" y "-fR" en este caso hacen exactamente lo mismo, la "r" hace referencia a que se eliminan todos los directorios de forma recursiva y la "f" a que se eliminan todos los archivos por la fuerza, es decir; sin preguntar.

A este *script* lo llamé "rst.sh".

Descripción del *script* 2

Claramente hay una similitud con el programa anterior en cuanto a las instrucciones que se ejecutan. Esto es porque no se necesita de algún recurso adicional a los que ya se ha utilizado.

En la línea 3 escribí la instrucción, de nuevo, de eliminar un directorio completo, forzando la operación y haciéndola recursivamente. Hay que tener cuidado con este tipo de comandos porque podríamos eliminar archivos del sistema, dañándolo; para nuestra fortuna el directorio en cuestión únicamente contiene lo respectivo al usuario y no al sistema.

Cuando utilicé el comando "tar" en la línea 4 no es para comprimir algo, de hecho se puede ver que en este caso se tiene una sola ruta, sino para descomprimir el archivo comprimido. La razón por la cual no indiqué la ruta en la cual se va a descomprimir es porque el archivo que ya se ha creado conserva esta información.

4.2.2. Pantalla de bloqueo o *lockscreen*

Problemática

Muy probablemente la máquina, pensando que estuviera terminada y en operación, pasaría una buena parte del tiempo esperando a que se le insertara una moneda para poder cumplir con todos sus propósitos. Durante esta espera la pantalla de la máquina debería mostrar algo y además el sistema debería estar bloqueado de alguna manera para que sea imposible utilizar la máquina mientras no se le inserte una moneda o crédito.

De nuevo con *Ubuntu* no existe forma de realizar esta tarea de forma nativa, y tampoco utilicé ningún programa externo para hacerlo. Entonces la opción que tuve fue programar esto de cero, con una serie de *scripts* y programas que me ayudaron a poner en forma el bloqueo de pantalla.

Solución

Ya que no tenía ninguna manera de completar esta tarea utilizando funciones y programas propios de mi sistema operativo, tuve que aprender cosas nuevas acerca de éste. Las dos principales cosas a las que me enfrenté para hacer esto fueron el bloquear el teclado y bloquear el ratón; esto debido a que poner en la pantalla una imagen o algo que le mostrara al cliente un comercial,

o que inserte una moneda.

Lo que hice en seguida fue preparar un *script* que me permitiera desplegar la imagen en pantalla completa cuando se iniciara la sesión de "cliente"; esto para ver como funcionaría y dejar esa parte terminada para luego seguir con otro reto. Para lograr lo anterior, y como mencioné en la sección 4.1.1, la aplicación de la cual saqué provecho fue una llamada *Feh*: esta sencilla aplicación funciona a través de la línea de comandos, por lo que me permitiría fácilmente hacer lo que deseaba desde un *script* de *Bash*.

Lo anterior se vería logrado con incluir en el *script* la siguiente línea:

```
feh -F /home/lab/.trash/old/lock.jpg
```

Lo que hice con esta línea de código, es indicarle al sistema que debe ejecutar el programa *Feh* con la la opción "-F" indicándole la ruta de un archivo. Se le indica un archivo, que en este caso fue una imagen –porque esta aplicación lo que hace es abrir una imagen–, que es el que se quiere abrir, el que puede ser un comercial o una simple indicación de insertar créditos. La opción en cuestión le ordena a la aplicación a desplegar la imagen en pantalla completa y no en una ventana.

Resuelta la parte de desplegar la imagen con el mismo *script* que fuera para bloquear tanto el teclado como el ratón, me concentré en terminar con esta tarea. Para ese entonces mis opciones eran limitadas, pues por un momento consideré en la posibilidad de buscar una aplicación que cumpliera con las funciones que yo requería para el proyecto, sin embargo la resolución vino con un desarrollo propio.

El principal problema fue el siguiente: tenía una imagen desplegada en pantalla completa, pero eso no sería suficiente para impedir que algún cliente hiciera uso del equipo; en primera porque se podía presionar la tecla de "Escape" y cerrar la imagen, podían presionar otras teclas que ingresarían a algún menú o cambiarían la vista de la imagen, o algo más fácil aún, con el ratón y un clic derecho bastaba para hacer cualquier cosa de las antes mencionadas.

Algo que me fue de mucha ayuda es un programa de código abierto, escrito en lenguaje *C* , que encontré acerca de cómo bloquear el teclado y el ratón por completo del sistema. La condición que el código proponía para regresar

a la normalidad el estado de estos periféricos es la de ingresar una contraseña por medio del teclado; el programa podría recibir la información de las teclas presionadas, pero el sistema no. Lo primero que se me ocurrió fue hacer funcionar el programa y problema resuelto, pero mi inconveniente fue que necesitaba tener habilitada una tecla en específico. En otra sección explicaré el por qué de esto. El punto es que no me servía enteramente ese código, pero sí le sacaría provecho a la hora de bloquear el ratón.

Código para bloquear tecleado

Resulta que en *Linux* existe una herramienta que permite modificar el mapeo que existe entre las teclas del teclado y la función que estas tienen sobre el sistema operativo. Con mapeo me refiero a que cuando una tecla "x" es presionada le corresponde una función "y" que suele ser la misma que la tecla presionada.

Ejemplo 3.

- *Si la tecla que se presiona es la que tiene impresa la letra "M", en el sistema correspondería llamar al caracter correspondiente a la tecla "M"*
- *Si la tecla que se presiona es la que tiene impresa la palabra "enter", en el sistema correspondería llamar a la función correspondiente, que ciertas aplicaciones realiza un salto de línea.*

Entonces si se puede modificar este mapeo, se logran dos cosas:

- Cambiar la función que le corresponde a una tecla, por ejemplo: si se presionara la tecla "M", en vez de llamar al carácter "M", se llamaría a alguna otra función, que podría ser, para este ejemplo, el caracter "P". Entonces, cada vez que se presione la tecla "M" sería como presionar la tecla que le corresponde a "P".
- Cambiar la función que le corresponde a una tecla por una función nula, es decir; mapear la función de una tecla de tal forma que al ser presionada haga absolutamente nada.

De los dos puntos anteriores pude concluir algo importante para esta tarea; teniendo como posibilidad la segunda de las descritas, podía bloquear cada una de las teclas que quisiera, incluyendo el teclado completo, asignando (o mapeando) a ellas con una función vacía, o mejor dicho; quitándoles funcionalidad.

El *script* que había empezado, y que hasta ahora solo contenía lo necesario para desplegar la imagen, debía ser mejorado utilizando esta herramienta para mapear el teclado. Esto lo conseguí agregando como lo siguiente al código de mi programa:

```
1 exec xmodmap -e 'keycode 24 ='&
2 exec xmodmap -e 'keycode 9 ='&
3 exec xmodmap -e 'keycode 119 ='&
4 exec xmodmap -e 'keycode 86 ='&
5 exec xmodmap -e 'keycode 82 ='&
```

Estos son unas de las lineas que el *script* necesita para indicarle al sistema que se desea cambiar la función de una tecla para que haga nada. La instrucción "exec" únicamente se refiere a que el siguiente comando del sistema debe ser ejecutado; "xmodmap -e" es quien nos mapea la expresión que le sigue. La manera en que esta instrucción entiende es por medio de lo que denomina *keycode*, y esto es lo que por defecto se ejecuta cuando se presiona una tecla, por ejemplo: en la línea 2, se aprecia que se refiere al *keycode* 9 y esto en nuestro caso era la tecla de "Escape", que era una de las que nos daba problemas; ahora vemos que esta igualada a nada, por lo que su función será hacer nada.

Como el código de la tecla puede cambiar con el sistema operativo, fue necesario conocer qué teclas son las que se requieren bloquear. Realicé diferentes pruebas y anoté el mapeo de las necesarias para el proyecto. Debido a lo dicho anteriormente, y una vez teniendo las teclas que se desean bloquear, es necesario verificar cuales son los códigos de estas; para hacer esto se debe abrir una terminal, o consola, de *Linux* , luego ejecutar el comando "xev" que mostrará en la terminal el código una vez presionada la tecla de la cual se desea conocer esta información. Se realiza lo anterior hasta completar la lista, así después pude continuar escribiendo nuestro *script*.

Código para bloquear ratón

Para ese momento solo me faltaba bloquear los clics del ratón, y para ello usé el código abierto que encontré en la red de cual no encontré referencia alguna acerca del autor de este código, solo se especificaba que era abierto. Cierto es que tuve que realizar algunas modificaciones al código para lograr que funcionara como yo deseaba, pues el código original incluía un apartado para bloquear el teclado también.

El programa trabaja con unas librerías de X11, que el protocolo para administrar ventanas y entornos gráficos que utiliza *Ubuntu*. Lo que hace es abrir una de "ventana" en nuestro display sobre todas las demás, aunque esta es invisible, después define el control del puntero, o ratón, y no se le asigna ninguna actividad, es decir; si es presionado algún botón de este, nada sucederá.

Aquí unas líneas del código:

```
1  #include <X11/Xlib.h>
2  #include <stdio.h>
3
4  int main()
5  {
6  Display *dpy;
7  XEvent xevent;
8  Window w;
9  const char *password = "112233";
10 int correct_keys = 0;
11
12 dpy = XOpenDisplay(NULL);
13 if (!dpy) {
14 fprintf(stderr, ‘‘Couldn’t open X display%s’’,
15 XDisplayName(NULL));
16 return 1;
17 }
18
19 // Lock mouse
20 w = RootWindow(dpy, DefaultScreen(dpy));
21
22 XGrabPointer(dpy, w, False, 0, GrabModeAsync, GrabModeAsync, None,
```

```
23 None, CurrentTime);
24
25 ...
```

Con estas líneas debería ser suficiente para lograr lo que buscaba desde un principio, realmente el código más relevante se encuentra a partir de la línea 20, que es donde se declara el control del puntero; lo demás son las configuraciones necesarias para lograr esto.

El código restante se enfoca en un método para salir del programa por medio de una contraseña, que nos serviría en caso de que algo saliera mal. La manera en que que programé salir del programa es diferente, por lo que no nos afecta que exista el método de la contraseña, pero es útil para realizar pruebas.

Ahora con los elementos suficientes para terminar el *script* de bloqueo de pantalla: el teclado se bloquea mapeando las funciones que tienen las teclas, los clics del ratón se bloquean con el programa escrito en C , dejando momentáneamente una máquina inutilizable y por último se despliega una imagen en pantalla completa que nos indicaría que se debe instertar una moneda o para anunciar algo.

Entonces el formato del *script* se vería algo así:

Script 3 (Bloqueo de pantalla "lock.sh")**.**

Uniendo todos los elementos necesarios: el mapeo de las funciones del teclado, ejecutar el programa que bloquea el ratón y ejecutar el programa que despliega la imagen en pantalla completa; las líneas del código serían algo como lo siguiente.

```
1  #! /bin/bash
2
3  exec xmodmap -e 'keycode 24 ='&
4  exec xmodmap -e 'keycode 9 ='&
5  exec xmodmap -e 'keycode 119 ='&
6  exec xmodmap -e 'keycode 86 ='&
7  exec xmodmap -e 'keycode 82 ='&
8  exec xmodmap -e 'keycode 80 ='&
9  exec xmodmap -e 'keycode 85 ='&
10 exec xmodmap -e 'keycode 88 ='&
```

```
11 exec xmodmap -e 'keycode 83 ='&
12
13 ...
14 ...
15 ...
16
17 exec /home/lab/.trash/old/m/lockm &
18 feh -F /home/lab/.trash/old/lock.jpg &
19
20 exit 0
```

La primera parte se encarga de bloquear el teclado (los puntos suspensivos indican que hay más código de ese estilo), en la línea 17 ejecutamos el programa que se encarga del ratón y en la 18 se despliega la imagen.

4.2.3. Desbloquear la pantalla

Problemática

Pareciera un tanto absurdo hablar de un programa que bloquee la pantalla y de otro a parte que la desbloquee, en realidad en mi caso fue necesario. El programa que se dedica a hacer lo primero, utiliza una instrucción que cambia la función que tienen las teclas del teclado por una que hace nada; gracias a esto necesité un *script* que revirtiera ese proceso –dado que las otras dos acciones pueden ser revertidas con solo terminar los programas que las realizan.

Solución

La primera parte del asunto es quitar la imagen y desbloquear el ratón de la máquina; esto basta con cerrar las aplicaciones que corren para realizar estas tareas. La parte ingeniosa del ésto es restablecer las funciones de las teclas del teclado que previamente se anularon. La herramienta "xmodmap" cuenta con una opción que me sirvió para lograr esto.

Tras estudiar el manual de dicha herramienta, encontré que con el siguiente comando se puede respaldar el mapa completo de códigos de teclas para poder cargarlo después:

```
$ xmodmap -pke > xmodmap.orig $
```

La opción "-pke" indica a la herramienta "xmodmap" que nos regrese una tabla del mapa completo de códigos de las teclas del teclado. El símbolo ">" es utilizado para indicar en lenguaje *Bash* que debe escribir lo sea regresado de un programa –es decir; lo que desplegaría en la consola un programa después de ser ejecutado– en un archivo: en mi caso fue "xmodmap.orig".

Teniendo el archivo respaldado del mapa de códigos, solo basta indicarle a la herramienta "xmodmap" que utilice este para asignar las funciones correspondientes a las teclas. Logré lo mencionado ejecutando siguiente comando:

```
$ xmodmap xmodmap.orig $
```

En caso de que a la hora de ejecutar el comando anterior no exista el archivo "xmodmap.orig" en la misma carpeta que el *script* se debe incluir la ruta de este archivo, como veremos a continuación.

Script 4 (Desbloqueo de pantalla "kill.sh")**.**
Este programa es un script que tiene como tarea desbloquear la pantalla más una función extra. El desbloqueo de la pantalla es necesario para utilizar la máquina de forma correcta y la función extra es la ejecución de otro programa que más adelante se describirá.

```
1 #! /bin/bash
2
3 killall feh &
4 killall lockm &
5 xmodmap /home/lab/.trash/old/xmodmap.orig
6 exec /home/lab/.trash/old/reloj/reloj1 &
7
8 exit 0
```

Descripción

Las instrucciones que se localizan en las líneas 3 y 4 contienen un formato "killall [aplicación]", el comando "killall" se encarga de cerrar la aplicación que se le indica. Es necesario cerrar las aplicaciones feh y lockm para quitar la imagen en pantalla completa y desbloquear el ratón, respectivamente. En la línea 5 se ejecuta la herramienta "xmodmap", indicándole de qué archivo recuperar el mapeo de las funciones de las teclas del teclado. Por último, en la

línea 6, se ejecuta una aplicación que programé para desplegar el reloj contador de tiempo regresivo.

4.2.4. Administración del tiempo de sesión

Problemática

Quizás este es el punto clave tanto de la máquina que construí como del resto del proyecto. De poco, o nada, nos sirve una máquina que pueda tener un escritorio limpio y restablecido cada vez que se inicia sesión y que en algún momento bloquee el equipo para pedir una moneda si no cuenta con alguna aplicación que administre el tiempo se sesión; sin este cualquiera que inserte un crédito y desbloquee la pantalla usaría sin límite de tiempo la máquina.

Lo que me propusieron fue construir un programa que fuera capaz de asignar tiempo a los créditos insertados de tal modo que cuando éste acabara la sesión, terminara y se restableciera el escritorio. En realidad el problema fue mas allá de lo sencillo que se escuchaba la propuesta, pues ajustar el tiempo de sesión implica varias cosas más: el programa debe desbloquear la pantalla al detectar un crédito, debe mostrar un "reloj", o "contador" del tiempo que le resta a la sesión, también el programa debe actualizarse en caso de que se inserten más créditos, y por último debe de terminar la sesión cuando el tiempo se acabe.

Solución

Resolví que el programa que debía construir tendría que ser un programa maestro, o cerebro, uno que controlara muchas de las operaciones de la máquina, es decir; muchos *scripts*. La función principal de el programa que administraría el tiempo de la sesión sería justamente contar el tiempo, además debería desbloquear la pantalla, terminar la sesión, etc.

Lo primero que realiza el *script* que al final llamé *timer*, que administra el tiempo de sesión es constantemente verificar si se han insertado créditos; esto lo logra constantemente leyendo un archivo que contiene el número de créditos, el archivo se llama *créditos*. Cuando por fin detecta que el archivo de créditos no está en ceros, inicializa las variables de tiempo de forma correcta y entonces puede continuar con la siguiente parte del programa.

A continuación un poco de dicho *script*, el principio de este:

```
#! /bin/bash

while [ 1 ]
do
i=0
ma=0
k=60
while read c
        do

            a=$((${c}*10))
            if [ $c -ge 6 ];
                then
                    i=$((${i}+(${c}/6)))
                    a=$(((${c}-6*(${c}/6))*10))
                    k=0

            fi

            if [ $a -gt 0 ];
                then
                    ma=$((${a}))
                    k=0
            fi

f=${c}

if [ $c -gt 0 ] ...
...

done < "/home/lab/.trash/old/creditos"
clear
done
#exit 0
```

Descripción

En las líneas anteriores inicializo las variables necesarias para contar el tiempo de forma regresiva. Antes, quiero aclarar que en la línea 28 del código existe una condición *if*, esta es la "puerta" que lleva al programa a ejecutar una serie de instrucciones una vez que detecta que existe al menos un crédito en el archivo correspondiente, mientras tanto solo está al pendiente de que se inserte una moneda para inicializar variables y continuar. El *script* empieza con un ciclo *while* eterno – el 1 en su argumento lo indica – con motivo de que nunca finalice el programa de forma autónoma; esto es para que exista siempre la constante verificación de si existen, o no, créditos.

Se declaran las variables que se utilizarán; más adelante veremos que **i** corresponde a las horas, **ma** a los minutos y **k** a los segundos. Con la ayuda de otro ciclo *while* se lee el archivo de créditos, guardando la información de este en una variable temporal **c**.

Dentro del segundo ciclo *while*, el cual lee el archivo de créditos por primera vez, escribí unas líneas de código dedicadas a inicializar las variables de forma adecuada para que así el contador de tiempo regresivo quede establecido en sus condiciones (de tiempo) iniciales. Dos condiciones *if* determinan qué debe suceder; primero se debe cerciorar que el número de créditos sea mayor o igual a seis,esto porque para este prototipo determiné que el tiempo otorgado, en minutos, por crédito sería de 10; así entonces si existen 6 o más créditos el tiempo en minutos equivale a una hora o más. Entonces, de ser así las variables se establecen como se ve en el siguiente ejemplo.

Ejemplo 4 (Inicializar variables de tiempo si el número de créditos es mayor o igual a 6)**.**

Si la máquina se apaga repentinamente debido a un corte en la energía eléctrica. La persona que utilizaba el equipo tenía 7 créditos restantes; por fortuna el número de créditos queda almacenado en el archivo de créditos. Entonces:

La máquina se enciende y al iniciarse el programa que administra el tiempo (timer) debe reconocer cuántos créditos existen y con ello inicializar las variables de tiempo de forma correcta. Sucedería lo siguiente:

[A partir del ciclo while de la línea 8, donde se lee el archivo de créditos]

1. *Variable temporal;* $\boldsymbol{a} = 7 \times 10 \Rightarrow \boldsymbol{a} = 70$
2. *¿Es* $\boldsymbol{c}$ *mayor o igual a* $\boldsymbol{6}$*?; Sí;*
   - $\boldsymbol{i} = 0 + \frac{7}{6}$ *[nota: en esta forma, la operación división regresará el número entero del cociente; en (7/6) es 1]*
   - $\boldsymbol{a} = [7 - (6 \times \frac{7}{6})] \times 10$
   - $\boldsymbol{k} = 0$
   - $\Rightarrow \boldsymbol{i} = 1$, $\boldsymbol{a} = 10$, $\boldsymbol{k} = 0$ *[nota: el valor de* $\boldsymbol{a}$ *ahora es 10, dejó de ser 70]*
3. *¿Es* $\boldsymbol{a}$ *mayor que* $\boldsymbol{0}$*?; Sí;*
   - $\boldsymbol{ma} = 10$
   - $\boldsymbol{k} = 0$
4. *Variable para almacenar los créditos, ya que* $\boldsymbol{c}$ *es temporal;* $\boldsymbol{f} = 7$
5. *¿Es* $\boldsymbol{c}$ *mayor que* $\boldsymbol{0}$*?: Sí* $\Rightarrow$ *Continua el programa...*

Anteriormente mencionamos lo que significarían las variables: $\boldsymbol{i}$*,el número de horas;* $\boldsymbol{ma}$*, el número de minutos;* $\boldsymbol{k}$ *el número de segundos. Con esto, nuestro contador quedaría iniciado a* ***1 hora con 10 minutos y 0 segundos****, que es lo mismo que* ***70 minutos****.*

En el ejemplo 3 describí cómo trabaja el código para inicializar variables; si el número de créditos fuera distinto, el resultado sería el correcto para determinar el tiempo inicial del contador. El caso más común sería que solo hubiese un crédito, por ese motivo el ejemplo se realizó con el caso mencionado.

Teniendo asignadas las variables de tiempo sus valores iniciales, el programa continua. En la línea 28 del último pedazo de código que venimos analizando existe el condicionante para que el programa pare de intentar inicializar las variables y le permita continuar. Cuando la condición se cumple, lo primero que se manda llamar es una instrucción para desbloquear la pantalla; es necesario esto puesto que con anterioridad expliqué cómo logré deshabilitar el teclado, el ratón y desplegar la imagen.

La instrucción que se ejecuta para desbloquear la pantalla, es en realidad un comando que llama al *script kill.sh* que programé y que se encarga de esto.

El siguiente comando es ejecutado dentro del *timer.sh* en el momento en el que se inserta una moneda o crédito:

```
exec /home/lab/.trash/old/kill.sh
```

Haciendo correr el *script* mencionado desbloquea por completo la pantalla, restableciendo el uso del teclado y el ratón. Una vez hecho esto, la máquina presenta un escritorio limpio y nuevo.

Aprovechando que al mismo tiempo que la pantalla se desbloquea debe empezar el contador de tiempo regresivo, con el *script kill.sh* ejecutamos un programa escrito en *C* que abre una ventana mostrándole al usuario de la máquina el tiempo que le resta de la sesión. Este reloj está sincronizado con el contador de tiempo de la sesión; este se encuentra dentro del mismo *timer.sh*, por lo que el programa de reloj únicamente muestra el tiempo pero no tiene que ver con el manejo de éste.

Debido a que el tiempo inicial del contador ya está asignado a las variables correspondientes, el contador de tiempo regresivo inicia. Este contador es, a grandes rasgos, una cadena de ciclos *for*, lo interesante radica en lo que sucede en cada segundo que transcurre de tiempo.

Por cada segundo transcurrido

En el momento en que el *script* llega al ciclo que maneja los segundos, lo primero que hace es imprimir el tiempo, tanto horas como minutos y segundos en un archivo externo, es el archivo *tiempo*. Este archivo se actualiza con cada segundo que transcurre; el motivo de este es el de proporcionar la conexión entre este programa y otro que será el que despliegue la ventana con el reloj que nos indica el tiempo restante de la sesión. Se logra mediante el siguiente comando:

```
1 echo "${i}
2 ${ma}
3 ${k} "  > '/home/lab/.trash/old/tiempo'
```

Lo único que realizan las líneas de código anteriores es imprimir las variables indicadas, que son las de tiempo, en el archivo con la ruta */home/lab/.trash/old/tiempo*. Si observamos las variables están escritas con una separación de una línea, esto es intencional para que así se impriman en el archivo; el programa que despliega el reloj así las lee.

Un comando sencillo pero más que importante es el que se encarga de hacer que cada iteración sea de exactamente un segundo, sin este el contador no funcionaría en tiempo real. Se logra con la siguiente línea:

```
sleep 1
```

Otra función que desempeña por cada segundo, y quizás de las más importantes que se incluye en el *timer.sh*, es la de verificar si se han agregado, o no, créditos a la máquina. Esto es fundamental debido a que el usuario de la máquina debe poder agregar tiempo a su sesión en cualquier momento y en caso de ser cierto que uno o más créditos fueron agregados, actualizar el contador sumándole el tiempo correspondiente a las monedas ingresadas.

Se consigue lo mencionado anteriormente con las siguientes líneas de código:

```
1  while read b
2    do
3    d=$b
4    x=$((${d}-${f}))
5
6      if [ $x -ge 1 ];
7        then
8        ma=$((${ma}+(10*${x})))
9        f=$d
10     fi
11
12     if [ $ma -eq 60 ];
13       then
14       i=$((${i}+1))
15       ma=0
16     fi
17
18     if [ $ma -gt 60 ];
19       then
20       i=$((${i}+1))
21       ma=$((${ma}-60))
22     fi
23
24 done < "/home/lab/.trash/old/creditos"
```

Descripción

Para demostrar cómo es que estas líneas de código verifican si se ha ingresado una moneda, o un crédito, ejemplifico este proceso.

Ejemplo 5 (Actualizar el tiempo de sesión en caso de existir un nuevo crédito)**.**

Supongamos que un cliente se encuentra en una sesión a la cual le restan ***55*** *minutos de tiempo, se le ocurre que es buena idea insertar una moneda más debido que por lo menos estará una hora más trabajando en la máquina. Como la máquina la programé para que por cada moneda se agreguen* ***10*** *minutos, después de que el cliente inserte la nueva moneda el tiempo de sesión debería actualizarse a 1 hora con 5 minutos y el número de segundos correspondientes. Describo enseguida qué sucede en el segundo próximo a esta acción y cómo este script logra actualizar el tiempo de sesión.*

[A partir de la línea 1 donde se lee el archivo de créditos nuevamente en el programa]

1. *Variable temporal;* $\boldsymbol{d} = 7$ *[Nota: la variable d representa el nuevo número de créditos entrar al ciclo while de cada segundo transcurrido]*
2. $\boldsymbol{x} = 7 - 6$ *[Nota: la variable f representa el número de créditos antes de entrar al ciclo while de cada segundo transcurrido]* $\Rightarrow \boldsymbol{x} = 1$
3. *¿Es* $\boldsymbol{x}$ *mayor o igual a* ***1****?; Sí;*
   - $\boldsymbol{ma} = 55 + (10 \times 1) \Rightarrow \boldsymbol{ma} = 65$
   - $\boldsymbol{f} = 7$
   - $\Rightarrow \boldsymbol{ma} = 65,\ \boldsymbol{f} = 7$
4. *¿Es* $\boldsymbol{ma}$ *igual a* ***60****?;No;*
5. *¿Es* $\boldsymbol{ma}$ *mayor que* ***60****?; Sí;*
   - $\boldsymbol{i} = 0 + 1 \Rightarrow \boldsymbol{i} = 1$
   - $\boldsymbol{ma} = 65 - 60 \Rightarrow \boldsymbol{ma} = 5$
   - $\Rightarrow \boldsymbol{i} = 1,\ \boldsymbol{ma} = 5$

Teniendo entendido que la variable ***i*** *representa a las horas y la variable* ***ma*** *representa a los minutos, nuestro tiempo de sesión restante quedaría actualizado a* ***1 hora con 5 minutos y los segundos correspondientes****. Queda entonces comprobado con este ejemplo que efectivamente el tiempo de sesión restante es actualizado si el cliente de la máquina inserta un crédito a media sesión. Recordando: este proceso ocurre cada segundo, por lo que la actualización es casi en tiempo real.*

Otras tareas del *timer*

Por último, pero no menos importante está el manejo de los créditos reales dentro de este mismo *script*. Si únicamente leyera el archivo que contiene el número de créditos para asignar el tiempo, el programa caería en un ciclo interminable pues el archivo de créditos seguiría intacto; es por esto que escribí una serie de líneas en el código que se encargan de resolver este problema. El método es sencillo, cada vez que transcurre el tiempo asignado por crédito insertado el *script* resta uno al archivo de créditos, de esta forma siempre podemos estar seguros que el crédito restante es el correcto.

Logré lo dicho anteriormente con las siguientes líneas de código:

```
1 r=$((10*${f}-$ma))
2 s=$((10-$r))
3
4 if [ $s -eq 0 ]
5   then
6   f=$((${f}-1))
7   u=$(($f))
8   echo $u > '/home/lab/.trash/old/creditos'
9 fi
```

Descripción

El funcionamiento de estas líneas de código es muy simple, cada minuto que transcurre ejecuta estas instrucciones verificando si la condición *if* se cumple o no. Cuando esta condición se cumple, se realiza una resta –se le quita una unidad a la variable **f**, que es la que contiene el valor de los créditos actuales– para luego imprimir su resultado en el archivo real de créditos. **r** y **s**

son solo variables temporales que sirven para determinar si es, o no, pertinente que se reste uno al archivo de créditos. Con **r** comparamos con la diferencia de 10, ya que 10 es el número de minutos asignados por cada crédito insertado, veces el número actual del archivo de créditos menos los minutos restantes del contador, y cuando esta diferencia sea 10 la variable **s** será cero, lo que provocará que la condición se cumpla y por resultado se actualice el archivo de créditos.

La última tarea que ejecuta este *script*, es la de mostrar un mensaje con cuenta regresiva; este mensaje da una última oportunidad para que el usuario inserte una moneda antes de que la sesión se cierre y pierda todos los cambios que se han realizado en la máquina. Si quisiera conservar su trabajo debería guardar todos los cambios en una memoria externa. Este mensaje también muestra una barra de proceso que indica el tiempo restante para que la sesión acabe de manera definitiva, este tiempo es de aproximadamente 10 segundos pero puede ser modificado.

Esta tarea es ejecutada por el *timer* a través de un *script*, de la siguiente manera:

Script 5 (*salir.sh*)**.** *Este script lo implementé para dar un aviso al usuario de la máquina antes de que esta cierre le sesión y borre toda la información vulnerable que se encontraba dentro de esta. El programa se ejecuta cuando el timer ha llegado a su fin y los créditos se encuentran en ceros, dentro de este script existen dos ciclos while, uno de ellos se encarga de verificar que mientras se despliegue el mensaje se haya insertado una moneda, para terminar este y seguir con el uso de máquina de manera regular. Muestra una barra de proceso con el fin de que quien la vea sepa exactamente cuánto tiempo le resta; esto se logra de la siguiente manera:*

```
zenity --progress --title="Sesion esta por terminar"
    --text="Inserta una moneda si no desea salir"
    --percentage=0 --no-cancel --auto-close
```

El comando "zenity" se encarga de desplegar este mensaje, y en nuestro caso habilitamos la barra de proceso. La barra de proceso esta asociada a un ciclo for que lleva la cuenta (del 0 al 100) que le indica a la barra como progresar, esto aunado a un comando "sleep" de 0.1 segundos, lo que logra que la barra recorra del 0 al 100 en 10 segundos.

Como este script es el que se encarga de cerrar la sesión es importante cerciorarse de que esto se realice. Aplicaciones como "OpenOffice.org" y "firefox" pueden detener un cierre normal de sesión para preguntar si realmente se desea salir sin guardar cambios, o por alguna otra razón; es por esto que incluí en las líneas de código para lograr esto. Las lineas que cierran estas aplicaciones y la sesión son las siguientes:

```
killall soffice.bin
killall firefox-bin
killall Xorg &
killall gnome-session &
```

Las primeras dos líneas son las que cierran las aplicaciones mencionadas y las dos últimas son las que cierran la sesión de forma definitiva, más en específico; la tercera cierra el servidor gráfico y la cuarta cierra propiamente la sesión de Gnome.

4.2.5. Reloj

Problemática

Quien utilice la máquina no solo debe saber que con determinado número de monedas que ingresó se le asigna determinado tiempo, también debe saber el tiempo que le resta en cualquier momento de la sesión; por ello se debe desarrollar una aplicación capaz de mostrar este tiempo restante en todo momento.

Solución

Encontré que una forma muy adecuada para mostrar un reloj que indique el tiempo restante de la sesión es mediante una aplicación que lea constantemente un archivo en donde se encuentre almacenado el tiempo; el archivo será el que se actualice por medio del timer.

El lenguaje que utilicé para resolver esta tarea es *C* , también hice uso de unas librerías llamadas SDL, que son comúnmente utilizadas para programar videojuegos. Al ser un programa simple el código no es muy largo, sin embargo; las librerías SDL requieren algo de código para configurar la ventana que se despliega con el reloj.

Muestro enseguida las primeras líneas del programa:

```
1  #include <SDL/SDL.h>
2  #include <SDL_ttf/SDL_ttf.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  void obtiene_tiempo( char* );
7
8  int main(int argc, char *argv[])
9  {
10   // Variables SDL
11   SDL_Surface *pantalla;
12   SDL_Event evento;
13   int salir = 0;
14
15   // Variables para dibujar fuente
16   TTF_Font* fuente;
17   SDL_Surface* s_fuente;
18   SDL_Color color = { 255, 255, 255, 0 };
19   // SDL_Rect: x, y, ancho, alto
20   SDL_Rect rect = { 0, 0, 0, 0 };
21
22   // Imagen
23   SDL_Surface* imagen;
24
25   char tiempo[ 32 ];
```

En esta parte del código incluyo las librerías que utiliza el programa, las que corresponden a SDL son: la librería principal, que es SDL.h y otra que nos permite importar fuentes de texto TTF, SDL_ttf.h. También declaro una función que servirá para leer el tiempo del archivo que lo contiene (*obtiene_tiempo(char*)*). Después procedo a declarar las variables que SDL necesita para construir una ventana y dibujar el texto. Analizando un poco estas variables se aprecia que son algo intuitivas; SDL_Surface se requiere para cualquier elemento que se vaya a dibujar, ya sea el fondo de la ventana, una imagen o el texto mismo; SDL_Color guarda la información del color que se vaya a utilizar; SDL_Rect nos sirve para dibujar un rectángulo, necesario para fungir como fondo de la ventana y por último SDL_Event, que será la variable que capture

cualquier tecla o botón presionado. La variable **tiempo** que declaro de tipo arreglo de caracteres la utilicé para almacenar el tiempo que se lee del archivo.

Lo que sigue es inicializar la ventana, declarar su tamaño y cargar la fuente que se utiliza para dibujar el texto.

```
1   if ( SDL_Init( SDL_INIT_VIDEO ) < 0 )
2   {
3     printf( "Error: %s\n", SDL_GetError() );
4     exit( 1 );
5   }
6
7   if ( TTF_Init() < 0 )
8   {
9     printf( "Error: %s\n", SDL_GetError() );
10    exit( 1 );
11  }
12
13  pantalla = SDL_SetVideoMode( 150, 60, 0,
14     SDL_ANYFORMAT );
15  if ( pantalla == 0 )
16  {
17    printf( "No se pudo inicializar video: %s\n",
18       SDL_GetError() );
19    exit( 1 );
20  }
21
22  SDL_WM_SetCaption("Tiempo", NULL);
23
24  fuente = TTF_OpenFont( "/home/lab/.trash/old/reloj/
25     default.ttf", 50 );
26  if ( fuente == 0 )
27  {
28    printf( "No se pudo cargar fuente: %s\n",
29       SDL_GetError() );
30    exit( 1 );
31  }
```

El código anterior es fácil de descifrar, pues con las funciones *SDL_Init* y

TTF_Init se inicializa los recursos para construir la ventana y poder dibujar el texto, respectivamente. Con *SDL_SetVideoMode* defino el tamaño de la ventana; para nuestro caso es 150 pixeles de ancho por 60 pixeles de alto. Por último, la función *SDL_WM_Caption* asigna título a la ventana y *TTF_OpenFont* carga en la variable **fuente** la tipografía "default.ttf" e indica que se requiere con un tamaño de 50 puntos. Cabe mencionar que todas estas funciones regresan un número entero que nos permite detectar errores; de hecho las declaraciones *if* prueban que se hayan ejecutado todas estas funciones sin error, de lo contrario el programa termina.

Analizando el ciclo principal del programa:

```
1  while( salir == 0 )
2    {
3      while( SDL_PollEvent( &evento ) )
4      {
5        //if ( evento.type == SDL_QUIT )
6          //salir = 1;
7        //else if ( evento.type == SDL_KEYDOWN )
8        {
9          //if ( evento.key.keysym.sym == SDLK_ESCAPE )
10           //salir = 1;
11       }
12       //else if ( evento.type == SDL_MOUSEBUTTONUP )
13       {
14         //printf( "Se presiono el raton\n" );
15       }
16     }
17
18     obtiene_tiempo( tiempo );
19
20     //printf( "Tiempo: %s\n", tiempo );
21
22     SDL_FillRect( pantalla, NULL, SDL_MapRGB(
23        pantalla->format, 0, 100, 100 ) );
24
25     //SDL_BlitSurface( imagen, NULL, pantalla, NULL )
26        ;
27
```

```
28      s_fuente = TTF_RenderText_Blended( fuente, tiempo
29          , color );
30      SDL_BlitSurface( s_fuente, NULL, pantalla, &rect
31          );
32      SDL_FreeSurface( s_fuente );
33
34      SDL_Flip( pantalla );
35
36      SDL_Delay( 250 );
37    }
```

Este ciclo es ejecutado siempre y cuando la variable **salir** siga siendo igual a cero. De la línea 3 a la línea 16 todas las declaraciones *if* y *else if* se encuentran comentadas; la razón de esto es que no necesitamos capturar ningún evento, incluyendo SDL_Quit que es equivalente a dar un clic a la tache de la ventana, muestro estas líneas para hacer evidente que así se evita que se cierre el programa. *obtener_tiempo(tiempo)* la describo más adelante. Las instrucciones que siguen dibujan en la ventana el rectángulo, que carga el fondo, y el texto que indica el tiempo leído del archivo. También son necesarias SDL_FreeSurface, para liberar memoria y SDL_Flip que ayuda a realizar un "double-buffering", pero esto no es necesario explicarlo para este proyecto. Por último se indica que el retraso entre ciclo y ciclo sea de 250 milisegundos.

Para finalizar con esta tarea, muestro el código que escribí para leer el tiempo del archivo que se actualiza gracias al timer:

```
1  void obtiene_tiempo( char* mensaje )
2  {
3    char linea[ 32 ];
4    int nhoras, nminutos, nsegundos;
5
6    FILE* f = fopen( "/home/lab/.trash/old/tiempo", "r"
7        );
8      if ( f )
9      {
10         fgets( linea, 32, f );
11         nhoras = atoi( linea );
12
13         fgets( linea, 32, f );
```

```
14          nminutos = atoi( linea );
15
16          fgets( linea, 32, f );
17          nsegundos = atoi( linea );
18
19          fclose( f );
20          sprintf( mensaje, "%.2d:%.2d:%.2d", nhoras,
21             nminutos, nsegundos );
22    }
23    else
24    {
25      sprintf( mensaje, " " );
26    }
27  }
```

Para leer el archivo en cuestión, utilizo un apuntador tipo FILE y con *fopen* indico qué archivo cargar y qué hacer con él, en este caso solo se necesita leerlo. Si el archivo existe y se cargó sin problemas, se procede a leer línea por línea el contenido de este con la función *fgets* y se convierte a entero con *atoi*; para entonces asignarlo a las variables **nhoras**, **nminutos**, **nsegundos**. Finalmente se cierra el archivo y se escribe el tiempo en el formato deseado (hh:mm:ss) en la variable que se recibe en el argumento de la función.

Ejemplo 6 (Obtener el tiempo restante del archivo *tiempo*)**.** *Si el archivo de tiempo contiene lo siguiente:*
1 45 5

Entonces, si se lee línea por línea, siendo la primera horas, la segunda minutos y la tercera segundos; tendríamos:

- *horas = 1*
- *minutos = 45*
- *segundos = 5*

Como el formato es hh:mm:ss (dos dígitos por parte), el tiempo restante quedaría:

01:45:05

4.2.6. Reinicio de la sesión

Problemática

Terminar la sesión de *Gnome* no es suficiente para el buen y completo funcionamiento de la máquina, pues es necesario reiniciar esta para que la máquina pueda volverse a usar. Es necesario implementar esta tarea pues esta opción no está disponible por defecto en *Ubuntu*. Con conocimiento acerca del sistema operativo es posible modificarlo para lograr lo cometido.

Solución

La mejor forma que encontré para resolver este problema fue modificar un *script* que el administrador de sesiones *GDM* ejecuta al momento en que la sesión actual termina, de esta forma pude controlar lo que sucedía en este punto tan específico de la sesión. Existe, dentro de la carpeta de configuración de *GDM*, un directorio dedicado a este:

`/etc/gdm/PostSession`

A su vez, dentro de este directorio mostrado anteriormente se encuentra un documento, que más que un documento es un *script*, llamado "Default"; es en este donde me involucré para lograr que la sesión se reinicie. Aprovechando que tuve el control de lo que sucedería al término de la sesión, escribí una línea extra antes de indicarle al sistema que la sesión debe empezar de nuevo.

Script 6 (/etc/gdm/PostSession/Default)**.** *Ejecutado tras terminar una sesión de forma regular, por defecto en Ubuntu se encuentra vacío, y modificado de la siguiente manera para lograr que la sesión se reinicie:*

```
1 #!/bin/sh
2
3 echo 0 > '/home/lab/.trash/old/creditos'
4 /sbin/restart gdm
5
6 exit 0
```

Las líneas de código anteriores describen tal cual debe estar el script; como se puede observar, únicamente contiene dos instrucciones: la primera, que mencioné la agregue aprovechando el control que se tiene en ese punto de la sesión, se encarga poner en ceros el archivo de créditos –que ya debiera

estar en ese estado– para asegurar que bajo ninguna circunstancia vaya a ser de otra forma ocasionando que la sesión al reiniciarse desbloquee la pantalla y sea utlilizable la máquina; la segunda instrucción indica que el el servicio "GDM" debe reiniciarse.

4.2.7. Agregar créditos

Este tema para mi proyecto es un tanto superficial, no por el hecho de incorporar un tragamonedas a un equipo de cómputo por medio de un microcontrolador sino porque los detalles son propiedad de la empresa que me contrató y no me correspondió a mi trabajar en ello. Sin embargo el funcionamiento de este aparato que recibe monedas no es muy complejo y puedo describirlo a grandes rasgos sin mayor complicación.

Problemática

El propósito de todo el proyecto es el de desarrollar una máquina que sea capaz de ofrecer al cliente un servicio similar al de un café Internet, pero una de las condiciones que se piden es que sea capaz de realizar el cobro y la administración de tiempo de manera automática. Entonces faltaría aclarar cómo funciona la parte del "autocobro".

Explicación

No llamo a este apartado resolución ya que en realidad, lo relacionado con el tragamonedas, la empresa ha utilizado esta tecnología con anterioridad; yo solamente ajusté un par de cosas para adaptarlo a mi máquina.

El tragamonedas que utilicé es un dispositivo que acepta únicamente un tipo de moneda, este se calibra colocando una muestra de la moneda dentro del tragamonedas. Si intentáramos introducir un tamaño de moneda que no es el que se especificó, el tragamonedas regresaría ésta. Cuando se introduce una moneda que sea la adecuada, el tragamonedas la aceptaría y este emitiría una señal que posteriormente debemos interpretar.

Para hacer uso de la señal que emite el tragamonedas utilicé un microcontrolador –en este caso fue un ATMEGA8–, que a su vez se conecta también a la máquina (al equipo de cómputo). La señal que emite el tragamonedas es un pulso; existe una tensión de 12 volts constantes y el pulso es a 0 volts por una fracción mínima de segundo. Antes de que el microcontrolador reciba la señal,

la salida del tragamonedas debe ser regulada a 5 volts, evitando así daños. El microcontrolador verifica constantemente un puerto de entrada para detectar un cambio (el pulso), una vez detectado el microcontrolador envía una señal de teclado al equipo de cómputo, como si se hubiera presionado la tecla de un teclado. La empresa para la que trabajé desarrolló un programa que le permite al microcontrolador fungir como un teclado conectado por el puerto USB. Esta señal es la que después el sistema operativo recibe para ejecutar un *script* que agrega un crédito al archivo de créditos ("moneda.sh"), que será detallado más adelante.

4.3. Datos de implementación

Resolver las tareas específicas necesarias para el buen funcionamiento de la máquina parecería suficiente, lo cierto es que de poco sirve tener todos los programas y los *scripts* si no se encuentran propiamente integrados, además de que el sistema operativo debe ejecutarlos de forma automática en un momento determinado.

4.3.1. Aplicaciones de inicio

Cuando llegó la hora de implementar las tareas específicas, después de haberlas programado, el propio *Ubuntu* me ayudó un poco. Resulta que más de un *script* debe ser ejecutado al arrancar la sesión, y para esto dentro de la distribución de *Linux* que usé existe una herramienta que nos permite ejecutar algún comando cuando esto sucede; esta herramienta se llama "Aplicaciones de inicio" y se encuentra en las preferencias del sistema. El hecho de que solamente se ejecute un comando no nos afecta ya que ese comando bien puede ser el de ejecutar un *script*. Así entonces resolvemos el problema de ejecutar los *scripts* que sean necesarios para hacer que la máquina funcione adecuadamente.

Los dos *scripts* que incluí en las aplicaciones de inicio son:

- *arranq.sh:* de este *script* no he hablado anteriormente pues solamente se encarga de terminar el administrador de tiempo de la sesión "timer.sh" y volverlo a ejecutar. Esto es necesario para que cada vez que se inicie la sesión se ejecute por primera vez el *timer* y no existan varias instancias de la aplicación corriendo al mismo tiempo. La razón por la cual queremos esto es que cuando el *timer* es ejecutado por primera vez configura su tiempo inicial dadas algunas condiciones iniciales.

- *lock.sh:* la función de este *script* ya la describí con anterioridad, y la razón de que deba ejecutarse al inicio de cada sesión es la misma que el por qué fue programado: se desea que la máquina imposibilite al usuario de usarla hasta que haya insertado una moneda, por lo tanto cada sesión que inicia debe bloquear la pantalla y algunos periféricos como lo son el ratón y el teclado.

Script 7 (iniciador del "timer.sh")**.**

El script cuenta con únicamente dos líneas efectivas de código; una de ellas es la que se encarga, mediante el comando "killall" de terminar el timer, y la segunda tiene como función ejecutar una nueva instancia de este:

```
1 #! /bin/bash
2
3 killall timer.sh
4 /home/lab/.trash/old/timer.sh
5 exit
```

4.3.2. Sobre el restablecimiento del escritorio

Lo que cabe mencionar en este pequeño apartado es acerca de la forma de hacer funcionar a el par de *scripts* que programé para lograr lo deseado. Ya tenemos los dos programas que resolverán el dilema de restablecer el escritorio. Antes que nada hay que colocar estos *scripts* en el directorio donde permanecerán.

Los dos programas (rst.sh y bkp.sh) los coloqué en directorio que hace referencia al escritorio de la cuenta *root*, esta cuenta existe por defecto en el sistema y es la que opera todos los programas y servicios del propio sistema operativo –esta cuenta no es accesible de forma regular a un usuario y se necesita de una contraseña especial, que puede ser distinta de la de administrador, aunque este es el que la asigna por primera vez–. Para hacer esto es necesario copiar el archivo al directorio como superusuario.

El directorio del que hablamos es: `/root/Desktop`

Una vez situados los programas en su lugar, y estando seguros de que el escritorio de la cuenta *cliente* se encuentra tal y como se requiere que siempre

esté, se debe hacer uso de nuestro *script* 1 para realizar el respaldo de nuestro directorio *home*. Se creará el archivo "backup.tar" que contendrá todo lo necesario para hacer el restablecimiento.

Lo siguiente es indicarle al sistema que debe hacer el restablecimiento del escritorio. La pregunta aquí fue ¿Cuándo debe suceder esto? Sabía que no deseaba que se restableciera de forma manual, sino automática; entonces debía hacerse en algún punto de la sesión que ocurriera siempre que un nuevo usuario llega a usar la máquina. Mi solución fue que debía ser en un principio, al momento de abrir la sesión del sistema (no la del usuario cuando inserta un crédito, más bien la propia del gestor gráfico de *Ubuntu*: GDM).

Con motivo de lograr lo anterior, debe modificarse con permisos de superusuario un archivo que se encuentra en el directorio: `/etc/gdm/Init`. El archivo en cuestión es el que lleva el nombre de "Default"

En este archivo se encuentran varias de las instrucciones que se ejecutan cuando el gestor gráfico, GDM, inicia. Al final del archivo, que es un *script*, agregué la siguiente linea:

```
1  #! /bin/sh
2  ...
3  .
4  .
5  ...
6  .
7  .
8
9  /root/Desktop/rst.sh
10
11 exit 0
```

La línea que nos interesa es la 9, es la que indica al *script* de "Default" que ejecute nuestro programa para restablecer el escritorio de nuestra cuenta de *cliente*. La línea 10 es solo parte del propio archivo y es la que termina ese *script*.

Así queda finalizada esta tarea específica, en resumen: se crean dos programas, uno para hacer un respaldo del escritorio por medio de un archivo comprimido y otro que descomprime el archivo creado por el anterior; se co-

locan en el directorio `/root/Desktop`; se ejecuta el primero para hacer el respaldo y por medio del inicio de sesión del sistema se indica que se ejecute el segundo para hacer el restablecimiento.

4.3.3. Hacer uso del tragamonedas

Una vez que tenemos un tragamonedas conectado a un microcontrolador y puede comunicarse a la máquina por medio de una señal de teclado, debe procesarse esta señal en cuestión. En específico, la señal de teclado que es enviada es equivalente a presionar la tecla F11. En realidad la manera más directa para hacer que esto funcionara fue estableciendo un acceso directo de la señal de teclado que recibe la máquina, es decir; si el insertar una moneda equivale a presionar una tecla del teclado, entonces podemos asignar que se ejecute un comando (consecuentemente un *script*) para que la máquina realice lo que se desee. Quiero agregar que al ser un prototipo se tuvo consciencia de que para una versión futura el método debeá ser diferente, puesto que cualquiera podría presionar la tecla saltándose al tragamonedas.

Entonces escribí un *script* que se encarga de sumar créditos; la forma en la que lo hace es muy sencilla: primero lee el archivo de créditos y obtiene el número actual de créditos, después opera ese número para sumarle uno y finalmente escribe el nuevo número de créditos en el archivo. Este es el *script* que se ejecuta cada vez que se inserte una moneda en el tragamonedas.

Así luce este *script*:

Script 8 ("moneda.sh")**.**

```
1  #! /bin/bash
2
3  while read c
4  do
5  a=$(($c+1))
6  done < '/home/lab/.trash/old/creditos'
7  echo $a > '/home/lab/.trash/old/creditos'
8
9
10 while read b
11 do
12 if [$b -eq 0]
```

```
13 then
14 echo 1 > '/home/lab/.trash/old/creditos'
15 fi
16 done < "/home/lab/.trash/old/creditos"
17
18 exit
```

Descripción

En la línea 3 inicia el ciclo *while* de lectura del archivo de créditos y y en la línea 5 se realiza la operación que suma 1 al número actual de créditos; **a** es la variable que guarda el nuevo número de créditos. En la línea 10 inicia otro ciclo *while* que nuevamente lee el archivo en cuestión para después compararlo con 0, y en caso de ser cierta esta condición se escribe 1 al archivo de créditos. Esto último es necesario debido a que como el *script timer.sh* actualiza también este archivo, se llega a dar el caso en que el archivo de créditos contenga -1, y si esto sucede estas líneas aseguran que se agregue el crédito de manera correcta.

4.3.4. Otros detalles

Cierto es que quedan muchos *scripts* y programas por colocar en su lugar. Para el propósito de este proyecto, en su primera fase de desarrollo, bastó con esconder todos estos en algún directorio al que el cliente no tenga fácil acceso. Es por eso que en las líneas de código vemos repetidas veces un directorio:

```
/home/lab/.trash/old
```

Este es el directorio en el que decidí colocar todos los archivos y programas que hacen que la máquina funcione. "lab" es otra cuenta de usuario que cree en el sistema operativo con derechos de administrador, por lo que no es el usuario del sistema "cliente" no tiene acceso a este. Además, una de las carpetas es precedida por un punto (".trash"), en sistemas operativos *Linux* esto significa que la carpeta se encuentra oculta por lo que dificulta aún más el acceso a esta.

4.3.5. Los *scripts*, programas y su interconectividad

Si bien la máquina se encuentra en funcionamiento gracias a todas las modificaciones que realicé al sistema operativo y a los programas y *scripts*

que escribí, vale la pena demostrar el resultado de esto en cuanto a cómo operan en conjunto. En el apartado anterior ya mostré cómo trabaja la máquina en un diagrama de flujo, en este apartado quiero hacer énfasis es la forma en que esto se logra.

Para demostrar lo anterior preparé un diagrama de interconectividad de todos los *scripts* y programas que operan para el buen funcionamiento de la máquina.

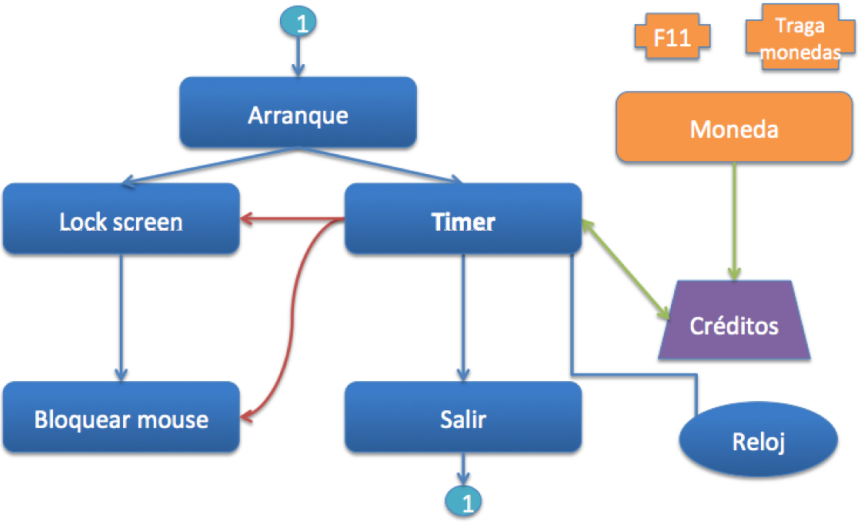

Figura 4.1: Diagrama de interconectividad.

En la figura 4.1 se muestra la conexión que existe con cada uno de los *scripts* que se ejecutan fuera de las operaciones estándar del sistema, es decir; los que forcé a que se ejecutaran de forma automática en la cuenta "cliente". Explico qué sucede en el diagrama con los siguientes puntos:

- El número 1 circunscrito se refiere al estado inicial de la máquina, que en este caso es el inicio de sesión.
- Los rectángulos indican ejecuciones de *scripts*, o programas, que están

relacionados directamente con el estado de la máquina y que causan algún efecto en ella.

- El óvalo es un programa que no causa efecto sobre el estado de la máquina.
- El trapecio es un archivo al que los programas tienen acceso de escritura y lectura.
- Las figuras tipo cruz hacen referencia a las señales externas, provenientes de los periféricos, que causan que se ejecute un *script* en particular.
- Las flechas indican el flujo de influencia que los programas llegan a tener sobre otros.

Más a detalle, justo después de que la máquina se encuentra en su estado inicial (iniciando sesión), ejecuta el *script* de **arranque**, lo que esto provoca que dos programas más se ejecuten: el **bloqueo de pantalla** y el **timer**. A su vez el **bloqueo de pantalla** ejecuta el **bloqueo del ratón** y el **timer** ejecuta el programa que muestra el **reloj**, y cuando llega su tiempo ejecuta el *script* de **salir** de sesión. Se ve que el **timer** también tiene influencia sobre el bloqueo de pantalla y el bloqueo del ratón, en este caso no es para ejecutar esos *scripts* sino para terminarlos.

El **timer** tiene influencia sobre el archivo de **créditos** y a su vez también depende de este, razón por la cual existe la flecha bidireccional entre ellos. Otro *script* que tiene influencia sobre el archivo de **créditos** es el de **moneda**, el cual se ejecuta al momento de recibir la señal directamente del microcontrolador y el **tragamonedas** –para propósitos de prueba, se puede mandar la misma señal a través de la tecla **F11**–.

Capítulo 5

Resultados y demostración del equipo

Si bien los elementos necesarios para construir la máquina están completos –han sido diseñados, programados e implementados–, el trabajo aún no acaba pues a penas empieza una nueva etapa en el desarrollo de este proyecto. En cuanto a mí, solo me resta documentar los resultados que tuve hasta el momento, y digo por adelantado que han sido satisfactorios.

Los resultados los expongo en dos secciones: la primera es en cuanto al funcionamiento interno de la máquina; la segunda es en cuanto a la experiencia del usuario o cliente de la máquina. Estos resultados serán evaluados de manera objetiva, reconociendo aciertos y cuestiones a mejorar.

5.1. Resultados del funcionamiento interno

Con funcionamiento interno me refiero al cómo la máquina trabaja con todos sus elementos de manera integrada. En este apartado muestro estos resultados comentando al respecto. Es fundamental que muestre con ilustraciones esto a lo que me he referido, así la comprensión se facilita al mismo tiempo que la explicación. Entre las ilustraciones incluyo diagramas de flujo, un diagrama relacional y sus respectivas explicaciones.

5.1.1. Ciclo operativo de la máquina

En cuanto al ciclo que recorre la máquina que construí cabe destacar que su autonomía es lo que considero más importante. El proyecto entero se concentra en que ésta sea capaz de operar bajo poca, o ninguna, supervisión humana, entonces el hecho de haber logrado esta autonomía fue crucial para la satisfacción general de mi trabajo.

Muestro a continuación un diagrama de flujo operativo de la máquina:

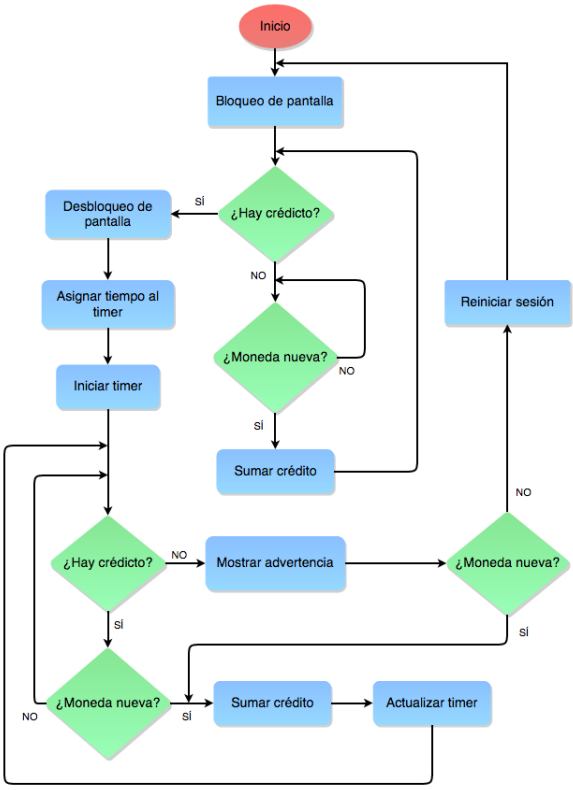

Figura 5.1: Diagrama operativo.

Mencioné que la máquina es autónoma, en la figura 5.1 se observa que una vez iniciado el proceso –iniciar este proceso quiere decir encender la máquina

y que inicie sesión– el flujo operativo nunca termina, en vez de eso regresa a un estado inicial dejándola preparada para recorrer el ciclo nuevamente.

Algo que merece la pena mencionar es que el proceso es muy sencillo, por lo que la máquina no sufre mayor afectación en cuanto a rendimiento, las operaciones se efectúan de manera casi instantánea, limitadas únicamente por el tiempo en que el sistema se actualiza, que es cada segundo.

Posibles mejoras

Aunque en términos globales estoy muy satisfecho con el producto final, el funcionamiento interno de esta máquina tiene un par de posibles mejoras:

- Por alguna razón el tragamonedas falla de vez en cuando, por fortuna esto no ocurre seguido. La posible causante del problema es la calidad de este producto, y podría ser arreglado ya sea con un tragamonedas de mayor calidad o quizás con algún parche en el código del microcontrolador.

- Algunas aplicaciones pueden detener el proceso de cerrar sesión, aunque detecté un par de estas valdría la pena hacer más pruebas buscando esta clase de fallo.

5.2. Usabilidad de la máquina

Esta sección trata de todo lo relacionado con lo que el usuario final de la máquina llega a percibir, esto incluye a las aplicaciones multimedia instaladas en el equipo, la velocidad de la máquina, la apariencia del escritorio, etc. Es importante revisar este asunto debido a que el usuario final es quien tiene la última palabra para decidir si usar o no la máquina. Existen diversos factores que influyen en el funcionamiento y la apariencia de la máquina, factores que pueden ser adecuados para maximizar la eficacia de la máquina y así gustar más. Entre los factores más destacables se encuentran la velocidad de cerrar e iniciar sesión una vez terminados los créditos, que las aplicaciones a las que vaya a tener acceso el usuario sean las adecuadas para cubrir las necesidades y expectativas básicas de quien use la máquina, y por último la apariencia misma del escritorio, es decir; la interfaz del usuario.

5.2.1. Velocidad y respuesta

Este factor es un tanto impreciso, resulta que lo que influencía más en esto es el hardware con el que esté equipada la máquina. El procesador, la memoria RAM, el procesador de gráficos, etc. Sin embargo, existe también influencia en el sistema que programé para el rendimiento de la máquina. Si bien el sistema responde con mucha fluidez –*Linux* es un sistema operativo muy veloz y estable– el hecho de que se cierre la sesión y vuelva a iniciar cada vez que un usuario termine con sus créditos afecta a la experiencia del usuario.

Algunas cuestiones a mencionar sobre el reinicio de sesión:

- Es necesario para cumplir con la propiedad de restablecimiento del escritorio.
- No es reversible, como no lo es el escritorio restablecido.
- Tarda al menos unos cuatro segundos en completar. Esto aumenta si existían muchas aplicaciones abiertas.
- Proporciona un escritorio limpio a cada usuario que utilice la máquina.
- Asegura que no exista rastro de las operaciones realizadas por usuarios anteriores; esto implica una seguridad mayor.

5.2.2. Seguridad

Quizás la característica más importante de la máquina que construí es la seguridad. Usualmente cuando se presta, o renta, un equipo de cómputo, éste cuenta con métodos de seguridad muy rudimentarios, me explico; es normal que una computadora guarde archivos descargados de la Internet, historial de búsqueda en el navegador, caché del mismo y hasta cambios en opciones del sistema. Lo anterior no ocurre con esta máquina gracias a que al terminar una sesión el sistema restablece el escritorio para dejarlo como si nunca hubiera sido utilizado anteriormente. El usuario tiene un acceso limitado a modificar archivos o configuraciones del sistema, pero de cualquier manera con cada sesión nueva el escritorio es restablecido junto con todas aquellos cambios que hayan podido ocurrir en el sistema.

De esta forma evitamos sucesos como los siguientes:

- Dejar almacenada información privada y sensible en el equipo de cómputo, comprometiendo datos personales.

- Que otros usuarios modifiquen preferencias del sistema o de aplicaciones, perjudicando así la experiencia final de otros usuarios.
- Que por alguna razón un archivo se corrompa y alguna aplicación deje de funcionar, o inclusive el sistema (bastaría con reiniciar la sesión para que todo vuelva a la normalidad).

La seguridad es un tema muy importante en cualquier servicio que se proporcione. Además del reinicio de sesión como método de seguridad, existe el bloqueo de pantalla, ratón y teclado para que cuando esté inactiva la sesión no pueda alguien interferir con el funcionamiento de la máquina; también hay que recordar que dentro de la sesión el usuario no tiene acceso a las configuraciones del sistema y tampoco puede alterar archivos importantes del sistema gracias al sistema de permisos que el propio sistema operativo soporta.

5.2.3. Interfaz y escritorio

Este es otro de los temas importantes en la máquina, es lo que el usuario final va a ver con la primer moneda que inserte. Que exista una interfaz funcional y agradable a la vista es fundamental para que quien utilice la máquina se sienta cómodo y con ganas de volverla a usar. Este aspecto no fue una tarea de primera prioridad en el proyecto, pues se pensaba trabajar más a futuro como muchas otras cosas. Sin embargo, al ser el primer prototipo de esta máquina, queda en mí presentar un producto de calidad que dé la sensación de que ha sido trabajado hasta en el aspecto estético, además de lo funcional.

Pantalla de bloqueo

Cuando la máquina y la pantalla se encuentran bloqueadas, en espera de una moneda, se deja ver una imagen en pantalla completa pidiéndole al usuario que inserte una moneda. Para esta imagen existe la posibilidad de poner cualquier material gráfico, ya sean imágenes, video o ambas. Por ejemplo, podría mostrarse la publicidad que el propietario de la máquina desee.

Otra posibilidad para la pantalla de bloqueo es que se muestre algún video corto de demostración sobre cómo operar la máquina con motivo de atraer a más clientes. Por otro lado, si un cliente potencial no tiene el conocimiento de cómo utilizar una computadora, este video demostrativo ayudaría a informar sobre cómo realizar tareas básicas, como por ejemplo abrir el procesador de textos o navegar en Internet.

5.3. Demostración

En esta sección mostraré algunas imágenes representantes del proyecto; entre ellas incluyo capturas de pantalla y fotografías de la maquina construida. Esto lo hago con la intención de demostrar cómo los *scripts* y programas que realicé trabajan en conjunto con el sistema operativo y los periféricos de la máquina para obtener el producto final que se buscaba.

La demostración la presento en dos partes: la primera, el hardware y la segunda, el software. Con hardware me refiero a los periféricos que caracterizan a esta máquina, es decir; el tragamonedas y el circuito impreso con microcontrolador; y con software hablo del sistema operativo, aplicaciones y, sobre todo, la interfaz que se le muestra al usuario, es decir el escritorio del que he hablado a lo largo del proyecto.

5.3.1. Hardware

Tragamonedas

Parte indispensable para el concepto de auto-administración de la máquina; el cobro autónomo por la utilización de la máquina no sería posible sin un dispositivo que sea capaz de recibir monedas, billetes, fichas, o la mercancía que se desee emplear. En mi caso utilicé uno de los tragamonedas más sencillos que se puede encontrar; éste solo recibe monedas de una única denominación que es configurable.

El tragamonedas es alimentado con 12 volts provenientes directamente de la fuente de alimentación del equipo de cómputo, que también proporciona su respectiva tierra.

Microcontrolador

Este dispositivo funge como una interfaz que interpreta la señal del tragamonedas para que el equipo de cómputo pueda entender que un crédito ha sido, o no, insertado. El diseño del circuito impreso y el programa que corre el microcontrolador pertenecen a la empresa. En la imagen 5.3 se aprecia que la salida hacia el equipo de cómputo es por medio de una conexión USB.

El microcontrolador es alimentado por 5 volts provenientes del equipo de cómputo también por el cable de la conexión USB, que también provee la tierra del circuito.

Figura 5.2: Tragamonedas utilizado.

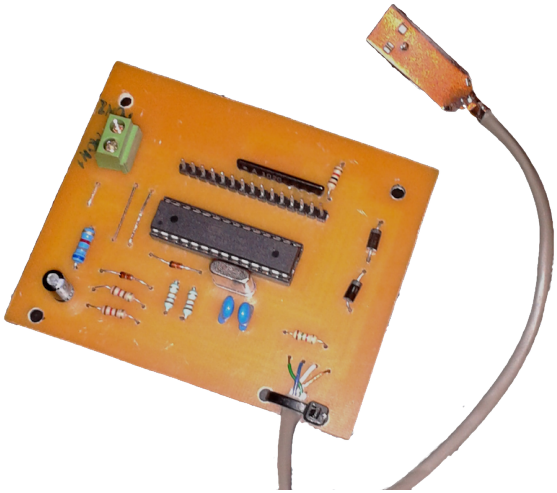

Figura 5.3: Microcontrolador utilizado.

5.3.2. Software

A continuación muestro de manera ordenada imágenes respecto a la experiencia del usuario en términos de software, es decir; lo mostrado a continuación representa lo que se vería al sentarse alguien a utilizar la máquina.

Escritorio

Al acceder a la sesión en la máquina, es decir; después de haber ingresado al menos un crédito, tendremos en pantalla un escritorio limpio y entonces estaría la máquina preparada para ser utilizada.

Figura 5.4: Escritorio en una nueva sesión.

Aclaro que este escritorio es casi el mismo que se puede encontrar en la distribución de *Linux Ubuntu*, salvo por las modificaciones que realicé para restringir determinados accesos a configuraciones. Dentro de este escritorio se puede notar la existencia de algunos elementos y características fundamentales para el buen funcionamiento de la máquina, tales como los siguientes:

- Interfaz limpia con íconos agrandados para su fácil localización.
- Barra de tareas en la parte inferior del escritorio para una sencilla administración de ventanas, así como una visualización de información básica del sistema.
- Ícono con la leyenda "PENDRIVE"que indica que una memoria *flash* externa está siendo reconocida por el sistema.
- Un reloj que indica el tiempo que le resta a la sesión antes de ser terminada.

Es necesario mostrar que la utilización de memorias externas es posible puesto a que es indispensable la existencia de un método para almacenar información personal, puesto a que cada vez que una sesión se reinicia todo cambio en el sistema es eliminado; incluyendo archivos ajenos a este. También es útil que se le indique al usuario cuanto tiempo queda para ser utilizada la máquina, este es el motivo de que se muestre un reloj que presente esta información.

Reloj

El reloj que muestra el tiempo restante de la sesión es una parte fundamental del software con el que cuenta la máquina. Sin este, el concepto de administración de tiempo según los créditos que se tengan, no tendría sentido puesto a que el usuario que utilice la máquina tendría que confiar en que el producto está haciendo bien su trabajo. En la figura 5.4 se aprecia la primera vista que tenemos del escritorio cuando se inicia sesión y en éste se encuentra la aplicación del reloj.

Figura 5.5: Reloj de tiempo restante.

La aplicación del reloj que desarrollé es sencilla y vistosa (ver figura 5.5), no hace más que mostrar el tiempo que falta para que la sesión termine; tarea fundamental.

Aplicaciones

Uno de los servicios que se provee con la máquina es el poder hacer uso de aplicaciones multimedia, Internet, procesadores de texto, etc. Existen algunos quioscos que permiten el acceso a Internet, impresión de fotografías, entre otras; la mayoría de estos quioscos permiten la utilización de únicamente una aplicación con un propósito muy bien determinado. En la máquina que construí, por poseer un entorno tipo escritorio tiene la característica ser multitarea, es decir; se puede utilizar más de un aplicativo al mismo tiempo.

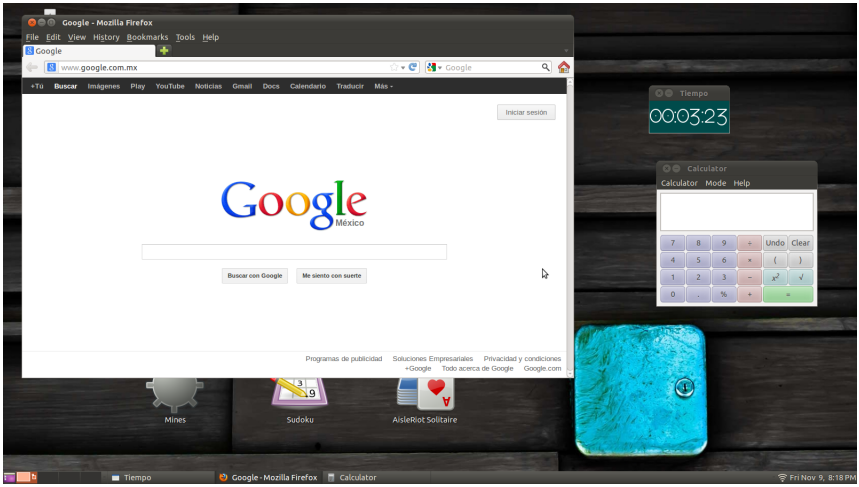

Figura 5.6: Escritorio con múltiples ventanas abiertas.

En la figura 5.6 vemos que es posible contar, en este caso, con tres aplicaciones abiertas. Digo tres aplicaciones porque estoy contando el reloj que muestra el tiempo restante de la sesión, que cabe aclarar que también es una aplicación. En esa misma figura se puede apreciar también que se cuenta con un navegador de Internet y una calculadora abiertos.

Advertencia mostrada

Una vez que se ha tenido activada la sesión en la máquina, se han utilizado las distintas aplicaciones por el tiempo que determinó según el número de créditos que se hayan ingresado (o monedas), el reloj deberá terminar su cuenta regresiva. Cuando el tiempo finaliza la sesión no termina inmediatamente, puesto que si lo hiciera se perdería el estado actual de la máquina (aplicaciones y archivos) sin previo aviso. Por lo anterior, implementé una solución que advierte al usuario de que su sesión está por terminar; al acabarse el tiempo de la sesión se despliega una ventana de notificación que pide insertar una moneda en caso de que no se desee terminar la sesión.

En la figura 5.7 se aprecia mejor el detalle. Es una ventana simple que contiene una barra de progreso que tarda un tiempo corto determinado para completarse. Esta es la última oportunidad que tiene el usuario para insertar otro crédito antes de salir de la sesión.

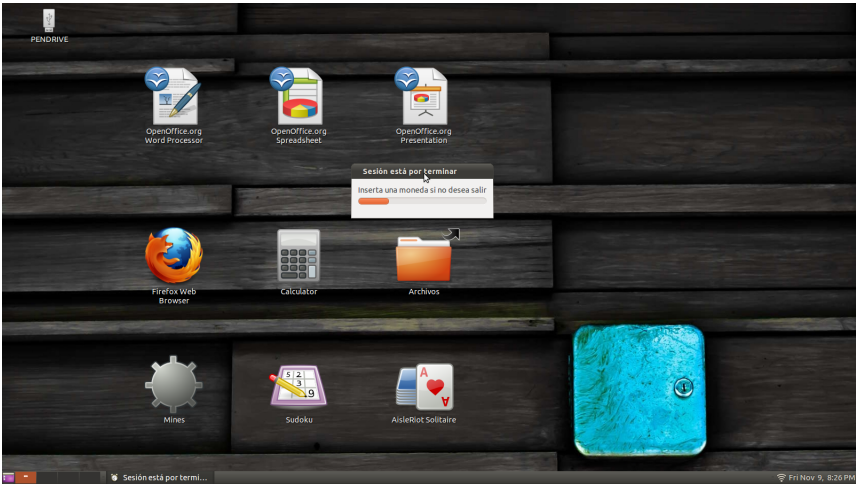

Figura 5.7: Escritorio al terminar el tiempo de sesión.

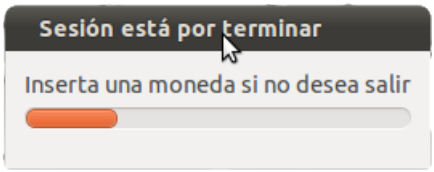

Figura 5.8: Advertencia de fin de sesión.

5.3.3. Proyección del producto terminado

El prototipo del sistema que permite automatizar el proceso de cobro por la utilización de un equipo de cómputo queda terminado y da como resultado un producto sólido y funcional; una máquina en sí misma que cumple un objetivo específico. A continuación se muestra una proyección del producto terminado:

Figura 5.9: Proyección de la máquina completa.

Capítulo 6

Conclusiones y resultados del proyecto

Durante la realización de este proyecto surgieron distintos problemas para lograr cumplir con el objetivo final. Estos problemas fueron un desafío a mis habilidades como ingeniero y que con perspicacia y dedicación resolví. Es un orgullo para mi presentar las conclusiones de este trabajo.

En general, el objetivo del proyecto se cumplió en su totalidad. Como dice en el primer capítulo de esta tesina, el objetivo del proyecto es:

"Diseñar y desarrollar un prototipo de una máquina que lleve a cabo la automatización de la cobranza y la administración de sesiones trabajo de diversos usuarios de una computadora; ésta con acceso a aplicaciones de escritorio, multimedia y de internet"

En particular, hubieron diversas metas específicas que se complementaban con el cumplimiento del objetivo del proyecto.

Algunas de las principales metas fueron:

- Elegir y justificar el sistema operativo a emplear.
- Diseñar un sistema de administración de sesiones.
- Diseñar un programa de temporización (*script timer.sh*).

- Diseñar un programa de control de periféricos.
- Mostrar físicamente el prototipo funcionando.

En el tercer capítulo de esta tesina dí una justificación detallada del por qué se elegí *Ubuntu Linux* como plataforma hacia los usuarios de la computadora. Esta resultó ser una decisión acertada; estratégica para lograr las diferentes metas y finalmente el objetivo.

En el cuarto capítulo de esta tesina doy una explicación detallada del código desarrollado para cumplir con los mecanismos de administración de sesiones, temporización y el control de periféricos. Desarrollé un diagrama que muestra cómo interactúan los diferentes programas (scripts) desarrollados.

Finalmente, en el quinto capítulo mostré los diagramas de flujo de la operación del dispositivo (sistema). Parte de este mismo capítulo trata de la demostración del prototipo funcional; muestro a detalle el diagrama de bloques de la operación de la máquina, así como los diagramas y esquemas que se usarán como base para la demostración.

Resumo del cumplimiento las metas lo siguiente:

La máquina que desarrollé logra automatizar el cobro para su utilización, así como administrar el tiempo de duración del uso de la misma. Esto es el objetivo principal y me da gusto expresar que lo he cumplido en su totalidad. De igual manera, me propuse una serie de metas que complementan al objetivo y al proyecto mismo, las cuales presentaron los retos y que gracias a ellos el desarrollo de la máquina se logró de manera ordenada y me ayudaron a nunca desviar el camino para lograr el objetivo final. La máquina, como producto final del proyecto cuenta con una plataforma que acepta monedas y otorga créditos por medio de un tragamonedas, una interfaz que lo interpreta y un programa en el equipo de cómputo que determina la equivalencia entre las monedas insertadas y los créditos otorgados. También posee con una plataforma que convierte el número de créditos que se tenga en tiempo, mismo que determina cuántos minutos se utilizará la máquina. Cuenta también con un sistema que impide la utilización de la máquina si no se cuenta con créditos, y bloquea el acceso a múltiples configuraciones del sistema para evitar daños a la máquina. Y por último, para garantizar la seguridad, tanto de la máquina como del usuario, cada vez que termine una sesión el sistema se restablecerá dejando un escritorio limpio y libre de rastros de uso.

Algunas conclusiones adicionales a las que se Llegué:

- El sistema operativo *Linux* resultó ser muy adecuado para desarrollar una máquina de este tipo por diversas razones:
  - Robusto manejo de permisos.
  - Tiempo corto de reinicio de sesión.
  - Sistema abierto, flexible y seguro.
- Una característica distinctiva de esta máquina, es que al iniciar una sesión provee una interfaz límpia, sin dejar rastros de sesiones pasadas; provee seguridad a los usuarios.
- El utilizar lenguaje *Bash* fue un gran acierto, ya que me facilitó el desarrollo de programas que interactúan con el sistema operativo.

En general, se concluye la viabilidad de desarrollar una máquina que automatice la cobranza al uso de un equipo de cómputo. La compañía Multijuegos ha dado seguimiento a los desarrollos obtenidos en este Proyecto y actualmente está comercializando soluciones relacionadas con el mismo. Mi principal contribución a este proyecto como Ingeniero Electrónico, fue el integrar mis conocimientos de distintas disciplinas (programación, electrónica, informática) para diseñar y desarrollar un producto final, que cumplió con el objetivo y las metas predeterminadas.

No considero que cumplir con los objetivos y las metas propuestas sea el cierre de este proyecto, por el contrario considero que esta experiencia me abre muchas puertas para seguir mejorando mi formación como ingeniero, además el haber estado a cargo de diseñar y desarrollar este prototipo incrementa el entusiasmo y empeño que como profesionista ejerzo en cualquier actividad que tenga de frente. La máquina puede mejorar todavía más, existen muchas ideas que puedo implementar, doy por hecho que mi trabajo no esta cerca de haber acabado, mi realización profesional a penas empieza; es por esto que reafirmo que el haber adquirido esta experiencia tiene un valor indescriptible.

Apéndice

A: bkp.sh

```
#! /bin/bash

rm -rf /root/Desktop/backup.tar
tar -cpPf /root/Desktop/backup.tar /home/cliente
```

C: rst.sh

```
#! /bin/bash

rm -fR /home/cliente
tar -xpPf /root/Desktop/backup.tar
```

D: Init/Default

```
#!/bin/sh
# Stolen from the debian kdm setup, aren't I sneaky
# Plus a lot of fun stuff added
#  -George

PATH="/usr/bin:$PATH"
OLD_IFS=$IFS

#if [ -x '/usr/bin/xsplash' ];
#then
#         /usr/bin/xsplash --gdm-session --daemon
#fi

/sbin/initctl -q emit login-session-start
    DISPLAY_MANAGER=gdm

gdmwhich () {
  COMMAND="$1"
  OUTPUT=
  IFS=:
  for dir in $PATH
  do
    if test -x "$dir/$COMMAND" ; then
      if test "x$OUTPUT" = "x" ; then
        OUTPUT="$dir/$COMMAND"
      fi
    fi
  done
```

```
    IFS=$OLD_IFS
    echo "$OUTPUT"
}

sysresources=/etc/X11/Xresources

# merge in defaults
if [ -f "$sysresources" ]; then
    xrdb -merge "$sysresources"
fi

sysmodmap=/etc/X11/Xmodmap

XMODMAP=`gdmwhich xmodmap`
if [ "x$XMODMAP" != "x" ] ; then
  if [ "x$GDM_PARENT_DISPLAY" = "x" ]; then
    if [ -f $sysmodmap ]; then
      $XMODMAP $sysmodmap
    fi
  else
    ( DISPLAY=$GDM_PARENT_DISPLAY XAUTHORITY=
        $GDM_PARENT_XAUTHORITY $XMODMAP -pke ) |
        $XMODMAP -
  fi

  #
  # Switch Sun's Alt and Meta mod mappings
  #

  UNAME=`gdmwhich uname`
  PROCESSOR=`$UNAME -p`
  if [ "x$PROCESSOR" = "xsparc" ]; then
    if $XMODMAP | /usr/bin/grep mod4 | /usr/bin/grep Alt
        > /dev/null 2>/dev/null
    then
      $XMODMAP -e "clear␣Mod1" \
               -e "clear␣Mod4" \
               -e "add␣Mod1␣=␣Alt_L" \
               -e "add␣Mod1␣=␣Alt_R" \
               -e "add␣Mod4␣=␣Meta_L" \
```

```
                -e "add␣Mod4␣=␣Meta_R"
    fi
  fi
fi

SETXKBMAP=`gdmwhich setxkbmap`
if [ "x$SETXKBMAP" != "x" ] ; then
  # FIXME: is this all right? Is this completely on
      crack?
  # What this does is move the xkb configuration from
      the GDM_PARENT_DISPLAY
  # FIXME: This should be done in code. Or there must
      be an easier way ...
  if [ -n "$GDM_PARENT_DISPLAY" ]; then
    XKBSETUP=`( DISPLAY=$GDM_PARENT_DISPLAY XAUTHORITY=
        $GDM_PARENT_XAUTHORITY $SETXKBMAP -v )`
    if [ -n "$XKBSETUP" ]; then
      XKBKEYMAP=`echo "$XKBSETUP" | grep '^keymap' | awk
          '{ print $2 }'`
      XKBTYPES=`echo "$XKBSETUP" | grep '^types' | awk
          '{ print $2 }'`
      XKBCOMPAT=`echo "$XKBSETUP" | grep '^compat' | awk
          '{ print $2 }'`
      XKBSYMBOLS=`echo "$XKBSETUP" | grep '^symbols' |
          awk '{ print $2 }'`
      XKBGEOMETRY=`echo "$XKBSETUP" | grep '^geometry' |
          awk '{ print $2 }'`
      if [ -n "$XKBKEYMAP" ]; then
        $SETXKBMAP -keymap "$XKBKEYMAP"
      elif [ -n "$XKBTYPES" -a -n "$XKBCOMPAT" -a -n "
          $XKBSYMBOLS" -a -n "$XKBGEOMETRY" ]; then
        $SETXKBMAP -types "$XKBTYPES" -compat "
            $XKBCOMPAT" -symbols "$XKBSYMBOLS" -geometry
            "$XKBGEOMETRY"
      elif [ -n "$XKBTYPES" -a -n "$XKBCOMPAT" -a -n "
          $XKBSYMBOLS" ]; then
        $SETXKBMAP -types "$XKBTYPES" -compat "
            $XKBCOMPAT" -symbols "$XKBSYMBOLS"
      elif [ -n "$XKBSYMBOLS" ]; then
        $SETXKBMAP -symbols "$XKBSYMBOLS"
```

```
        fi
      fi
    fi
fi
/root/Desktop/rst.sh

exit 0
```

E: PostLogin/Default

```
#!/bin/sh
#
# Note: this is a sample and will not be run as is.
    Change the name of this
# file to <gdmconfdir>/PostLogin/Default for this script
     to be run.  This
# script will be run before any setup is run on behalf
    of the user and is
# useful if you for example need to do some setup to
    create a home directory
# for the user or something like that.  $HOME, $LOGIN
    and such will all be
# set appropriately and this script is run as root.

killall timer.sh &
exec /root/Desktop/timer.sh &
```

F: PostSession/Default

```
#!/bin/sh

echo 0 > '/home/lab/.trash/old/creditos'
/sbin/restart gdm

exit 0
```

G: arranq.sh

```
#! /bin/bash

killall timer.sh
/home/lab/.trash/old/timer.sh
exit
```

H: kill.sh

```
#! /bin/bash


killall feh &
killall lockm &
#killall reloj1 &
xmodmap /home/lab/.trash/old/xmodmap.orig
exec /home/lab/.trash/old/reloj/reloj1 &

exit 0
```

I: lock.sh

```
#! /bin/bash

exec xmodmap -e 'keycode 24 =' &
exec xmodmap -e 'keycode 9 =' &
exec xmodmap -e 'keycode 119 =' &
exec xmodmap -e 'keycode 86 =' &
exec xmodmap -e 'keycode 82 =' &
exec xmodmap -e 'keycode 80 =' &
exec xmodmap -e 'keycode 85 =' &
exec xmodmap -e 'keycode 88 =' &
exec xmodmap -e 'keycode 83 =' &
exec xmodmap -e 'keycode 106 =' &
exec xmodmap -e 'keycode 53 =' &
exec xmodmap -e 'keycode 58 =' &
exec xmodmap -e 'keycode 55 =' &
exec xmodmap -e 'keycode 107 =' &
exec xmodmap -e 'keycode 25 =' &
exec xmodmap -e 'keycode 63 =' &
exec xmodmap -e 'keycode 66 =' &
exec xmodmap -e 'keycode 94 =' &
exec xmodmap -e 'keycode 133 =' &
exec xmodmap -e 'keycode 64 =' &

exec /home/lab/.trash/old/m/lockm &
feh -F /home/lab/.trash/old/lock.jpg &


exit 0
```

J: moneda.sh

```
#! /bin/bash

while read c
do
a=$(($c+1))
done < '/home/lab/.trash/old/creditos'
echo $a > '/home/lab/.trash/old/creditos'


while read b
do
if [$b -eq 0]
then
echo 1 > '/home/lab/.trash/old/creditos'
fi
done < "/home/lab/.trash/old/creditos"

exit
```

K: salir.sh

```
#! /bin/bash

while read c
do
  x=$c
  if  [ $x -lt 1 ];
  then
    for ((i=0;i<100;i++)); do
      echo ${i}
      sleep 0.1

      while read b
      do
        a=$b
        if [ $a -gt 0 ];
        then
          killall salir.sh
        fi
      done < "/home/lab/.trash/old/creditos"
    done | zenity --progress --title="Sesión está por
      terminar" --text="Inserta una moneda si no desea
      salir"    --percentage=0 --no-cancel --auto-
      close
  sleep 1
  echo terminaste
  #killall timer.sh
  killall soffice.bin
  killall firefox-bin
```

```
    killall Xorg &
    killall gnome-session &
    fi

done < "/home/lab/.trash/old/creditos"
exit
```

L: timer.sh

```
#! /bin/bash

while [ 1 ]
do
i=0
ma=0
k=60
while read c
      do

      a=$((${c}*1))
      if [ $c -ge 6 ];
        then
          i=$((${i}+(${c}/6)))
          a=$(((${c}-6*(${c}/6))*1))
          k=0

        fi


      if [ $a -gt 0 ];
        then
          ma=$((${a}))
          k=0
        fi

f=${c}
if [ $c -gt 0 ]
```

```
    then
    exec /home/lab/.trash/old/kill.sh &
for ((i;i>=0;i--));
  do

    for ((ma;ma>=0;ma--));
    do

    if [ $c -gt 0 ];
    then
      echo si3
      for ((k;k>=0;k--));
      do

      clear
      echo "${i}
${ma}
${k}␣" > '/home/lab/.trash/old/tiempo'
echo "${i}
${ma}
${k}␣"
      sleep 1
        while read b
        do
        d=$b
        x=$((${d}-${f}))

          if [ $x -ge 1 ];
          then
          ma=$((${ma}+(1*${x})))
          c=$d
          f=$((${b}))
          fi

          if [ $ma -eq 60 ];
          then
          i=$((${i}+1))
          ma=0
          fi
```

```
            if [ $ma -gt 60 ];
            then
            i=$(( ${i}+1))
            ma=$(( ${ma}-60))
            fi

          done < "/home/lab/.trash/old/creditos"


        done
        k=$(( ${k}+60))


        fi



        r=$(( 1*${f}-$ma))
        s=$(( 1-$r))

        if [ $s -eq 0 ]
        then
        f=$(( ${f}-1))
        u=$(( $f))
        echo $u > '/home/lab/.trash/old/creditos'
        fi


      done


   ma=$(( ${ma}+60))
   done
killall -9 reloj1
/home/lab/.trash/old/salir.sh

fi

done < "/home/lab/.trash/old/creditos"
clear
done
```

```
#exit 0
```

M: reloj.c

```
#include <SDL/SDL.h>
#include <SDL_ttf/SDL_ttf.h>
#include <stdio.h>
#include <stdlib.h>

void obtiene_tiempo( char* );

int main(int argc, char *argv[])
{
  // Variables SDL
  SDL_Surface *pantalla;
  SDL_Event evento;
  int salir = 0;

  // Variables para dibujar fuente
  TTF_Font* fuente;
  SDL_Surface* s_fuente;
  SDL_Color color = { 255, 255, 255, 0 };
  // SDL_Rect: x, y, ancho, alto
  SDL_Rect rect = { 0, 0, 0, 0 };

  // Imagen
  SDL_Surface* imagen;

  char tiempo[ 32 ];

  if ( SDL_Init( SDL_INIT_VIDEO ) < 0 )
  {
```

```
    printf( "Error: %s\n", SDL_GetError() );
    exit( 1 );
}

if ( TTF_Init() < 0 )
{
    printf( "Error: %s\n", SDL_GetError() );
    exit( 1 );
}

pantalla = SDL_SetVideoMode( 150, 60, 0, SDL_ANYFORMAT
    );
if ( pantalla == 0 )
{
    printf( "No se pudo inicializar video: %s\n",
        SDL_GetError() );
    exit( 1 );
}

SDL_WM_SetCaption("Tiempo", NULL);

fuente = TTF_OpenFont( "/home/lab/.trash/old/reloj/
    default.ttf", 50 );
if ( fuente == 0 )
{
    printf( "No se pudo cargar fuente: %s\n",
        SDL_GetError() );
    exit( 1 );
}

imagen = SDL_LoadBMP( "/home/lab/.trash/old/reloj/
    jukinux.bmp" );
if ( imagen == NULL )
{
    printf( "No se pudo cargar imagen: %s\n",
        SDL_GetError() );
    exit( 1 );
}
//SDL_SetColorKey( imagen, SDL_SRCCOLORKEY, SDL_MapRGB
    ( pantalla->format, 0, 0, 0 ) );
```

```
SDL_SetAlpha( imagen, SDL_SRCALPHA, 64 );
while( salir == 0 )
{
  while( SDL_PollEvent( &evento ) )
  {
    //if ( evento.type == SDL_QUIT )
      //salir = 1;
    //else if ( evento.type == SDL_KEYDOWN )
    {
      //if ( evento.key.keysym.sym == SDLK_ESCAPE )
        //salir = 1;
    }
    //else if ( evento.type == SDL_MOUSEBUTTONUP )
    {
      //printf( "Se presionó el ratón\n" );
    }
  }

  obtiene_tiempo( tiempo );

  //printf( "Tiempo: %s\n", tiempo );

  SDL_FillRect( pantalla, NULL, SDL_MapRGB( pantalla->
      format, 0, 100, 100 ) );

  SDL_BlitSurface( imagen, NULL, pantalla, NULL );

  s_fuente = TTF_RenderText_Blended( fuente, tiempo,
      color );
  SDL_BlitSurface( s_fuente, NULL, pantalla, &rect );
  SDL_FreeSurface( s_fuente );

  SDL_Flip( pantalla );

  SDL_Delay( 250 );
}

SDL_FreeSurface( imagen );
TTF_Quit();
SDL_Quit();
```

```
  return 0;
}

void obtiene_tiempo( char* mensaje )
{
  char linea[ 32 ];
  int nhoras, nminutos, nsegundos;

  FILE* f = fopen( "/home/lab/.trash/old/tiempo", "r" );
    if ( f )
    {
        fgets( linea, 32, f );
        nhoras = atoi( linea );

        fgets( linea, 32, f );
        nminutos = atoi( linea );

        fgets( linea, 32, f );
        nsegundos = atoi( linea );

        fclose( f );
        sprintf( mensaje, "%.2d:%.2d:%.2d", nhoras,
            nminutos, nsegundos );
  }
  else
  {
    sprintf( mensaje, "␣" );
  }
}
```

Bibliografía

[1] Banahan M. y Brady D. y Doran M. *The C Book*. Addison Wesley, 2nd edition, 1991.

Referencias

[2] GNU. Vendiendo software libre.
http://www.gnu.org/philosophy/selling.es.html.

[3] GNU. The gnu bash reference manual.
http://www.cs.cmu.edu/ awb/linux.history.html, 2010.

[4] Wikimedia Foundation Inc. Linux. http://en.wikipedia.org/wiki/Linux, Última actualización: 16 Enero 2013.

[5] Wikimedia Foundation Inc. Hero of alexandria.
http://en.wikipedia.org/wiki/Hero_of_Alexandria, Última actualización: 26 Enero 2013.

[6] Wikimedia Foundation Inc. Vending machine.
http://en.wikipedia.org/wiki/Vending_machine, Última actualización: 27 Enero 2013.

[7] Wikimedia Foundation Inc. Linus trovalds.
http://en.wikipedia.org/wiki/Linus_Torvalds, Última actualización: 28 Enero 2013.

[8] Linus Trovalds. Linux history.
http://www.cs.cmu.edu/ awb/linux.history.html, 1992. Mail letters.

[9] GNU with Luis Miguel Arteaga Mejía. ¿qué es software libre?
http://www.gnu.org/philosophy/free-sw.es.html, 2001.