Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Operativos

Proyecto

Fase 5

López Hernández Lissete

Palacios Cabrera Alberto

Pérez Priego Mario Daniel

Grupo: 2CM15

Resumen

El presente proyecto consiste en implementar los mecanismos IPC (Inter-Process Communication) para la realización de un programa ejecutado desde consola que simule el comportamiento de una tienda online con las funciones de registro de usuario (cliente y proveedor), compra de artículos, visualización de catálogo de productos y de carrito de compras.

Para las funciones de registro de usuarios, catálogo de productos y carrito de compras se implementaron listas enlazadas que se almacenan en un archivo con extensión .txt, con el objetivo de no perder los datos con cada ejecución del programa. El registro de usuarios es requisito para iniciar sesión y poder acceder a las demás funciones de la tienda.

Una vez que un usuario inicia sesión debe seleccionar que acción desea hacer y será atendido de acuerdo con la indicación de los semáforos implementados, y de acuerdo con su orden de llegada, dado por una cola.

Para garantizar que todos los usuarios observen el mismo catálogo de productos, se compartió dicho documento mediante memoria compartida.

Cabe mencionar que las funciones previamente descritas son comunicadas al programa mediante la implementación de colas de mensajes, en las que el usuario envía la orden y el programa la respuesta.

I Índice de contenidos

Contenido

II Índice de figuras I
III Índice de tablas II
IV Introducción III
V Descripción del proyecto IV
VI Objetivos que debe cumplir el proyecto V
VII Justificación VI
VIII Marco teórico 1
Sistema de ventas electrónico 1
Linux 1
Procesos 2
Hilos 4
Herramientas del IPC (Inter Process Communication) de UNIX-Linux 5
Semáforos 5
Memoria compartida 6
Colas de mensajes 6
Archivos 7
Dispositivos de entrada y salida 7
IX Análisis y diseño 9
X Implementación y pruebas 12
Ventanas de Ejecución 28
Salida de la terminal para Control 28
Salida de la terminal para Cliente 30
Salida de la terminal para Proveedor 33
XI Conclusiones 38
López Hernández Lissete 38
Palacios Cabrera Alberto 39
Pérez Priego Mario Daniel 40
XII Referencias 42
XIII Anexos 43
Códigos 43
err.h 43
listaProductos.h 43

listaProductos.c 44
listaUsuarios.h 50
listaUsuarios.c 50
listaCarrito.c 56
listaCarrito.h 61
control.h 62
control.c 62
queueMensaje.h 67
mainControl.c 68
mainUsuario.c 72
nodoProductos.h 77
nodoProductos.c 78
nodoUsuarios.h 79
nodoUsuarios.c 79
nodoCarrito.h 80
nodoCarrito.c 81
Usuario.c 82
Usuario.h 83
Protocolo 85
Implementación de un sistema de compras electrónico utilizando herramientas IPC. 85
Trabajo Terminal No. 1 85
1. Introducción 85
2. Objetivo 86
3. Justificación 87
4. Productos o Resultados esperados 87
5. Metodología 88
6. Cronograma 90
7. Referencias 90
8. Alumnos y directores 92

II Índice de figuras

Figura 1 Diagrama de estados de un proceso 2
Figura 2 Ejemplo de lista enlazada de usuarios 9
Figura 3 Ejemplo de lista enlazada de productos 10
Figura 4 Proceso de control y sincronización de usuarios 10
Figura 5 Proceso de compra 11
Figura 6 Estructura usuario 12
Figura 7 Estructura producto 12
Figura 8 Estructura carrito 13
Figura 9 Prototipo dis_existencias 13
Figura 10 Función dis_existencias 13
Figura 11 Estructura mensaje 14
Figura 12 Acciones y estatus del usuario en la cola de mensajes 14
Figura 13 LLave usuario-control 15
Figura 14 Identificador de cola de mensajes. mainControl.c 15
Figura 15 Cola de mensajes control-usuario 15
Figura 16 Creacion de usuario. mainControl.c 15
Figura 17 Memoria compartida Productos. mainControl.c 16
Figura 18 Memoria compartida Carrito mainControl.c 16
Figura 19 Case SESION 16
Figura 20 case VER_PRODUCTOS 17
Figura 21 case ADD_PRODUCTOS 17
Figura 22 case ERROR 17
Figura 23 case BUY_PRODUCTOS 18
Figura 24 case MOD_PRODUCTOS 19
Figura 25 case HIS_PRODUCTOS 19
Figura 26 Menú mainUsuario.c 19
Figura 27 case SESION mainUsuario.c 20
Figura 28 case VER_PRODUCTOS mainUsuario.c 20
Figura 29 case BUY_PRODUCTOS. mainUsuario.c 21
Figura 30 case ADD_PRODUCTOS mainUsuario.c 22
Figura 31 case MOD_PRODUCTOS mainUsuario.c 22
Figura 32 case HIS_PRODUCTOS mainUsuario.c 23
Figura 33 Fragmento de la función crearUsuarios(). 24
Figura 34 Fragmento de la función crearProductos(). 25
Figura 35 fragmento de la función crearCarritos(). 26
Figura 36 Función menu() del archivo Usuario.c 26
Figura 37 Función ini_sesion() del archivo Usuario.c 27
Figura 38 Manejo de las respuestas recibidas por el programa Control. 27
Figura 39 Creación y manejo de los semáforos. 28

III Índice de tablas

Tabla 1 Motivos para la terminación de un proceso 3

IV Introducción

El presente documento pretende describir los procesos de análisis, diseño e implementación de un sistema de compras electrónico que permita el manejo de múltiples operaciones de diversos usuarios de manera concurrente, manteniendo la fiabilidad de los datos y logrando una buena usabilidad para los usuarios del sistema.

En la primera sección del documento se realiza la descripción del proyecto, donde se plantea el contexto y la problemática que planea solucionar el sistema que se desarrollará.

Posteriormente se declaran los objetivos y justificación del proyecto, esto permite fijar el alcance y las limitaciones que se tienen en el desarrollo de este. Además, nos permite visualizar algunos requisitos, requerimientos e incluso posibles reglas de negocio dentro del sistema.

Una vez que se ha definido del proyecto, es necesaria una investigación previa y contextualización de los recursos que serán utilizados dentro del desarrollo. Para ello se encuentra la sección de marco teórico, donde principalmente se puede identificar la descripción de las herramientas que se implementaran en el sistema.

La siguiente etapa consta del análisis y diseño del sistema, donde se muestra cómo se desarrolla el sistema basándose en los requerimientos planteados previamente. En esta sección se puede ver como se hace uso de las diferentes herramientas planteadas durante el marco teórico. Para ello se hace uso de descripciones técnicas detalladas y recursos gráficos que faciliten la comprensión del documento para el lector.

A continuación, se encuentra la sección de implementación y pruebas, en la cual se muestra el desarrollo del sistema a partir de la descripción detallada de los códigos generados en lenguaje C. También se muestran las ejecuciones de dichos códigos para visualizar los resultados que arroja nuestro sistema al usuario.

Por último, la sección de anexos nos proporciona los códigos completos utilizados en el sistema, mostrando así la arquitectura completa de este. Además, también se incluye el protocolo realizado para el desarrollo de este proyecto, y las conclusiones del equipo acerca de la realización del proyecto.

V Descripción del proyecto

Actualmente a causa de la pandemia mundial por Covid-19, ha crecido el desarrollo de diversas herramientas tecnológicas que nos faciliten ciertas actividades a pesar de la disminución de movilidad por parte de la población.

El área del comercio claramente tuvo que dar el salto tecnológico para poder seguir proporcionando bienes básicos, normales y de lujo. Una gran cantidad de empresas grandes ya contaban con servicios web y diversas herramientas tecnológicas que permitieran realizar comercio electrónico, sin embargo, existen pequeñas y medianas empresas (PyMES) que desean dar el salto tecnológico para ajustarse a las necesidades actuales. Al ser empresas con mercados reducidos, los sistemas que requieren para sus empresas deben de dar las herramientas necesarias para su gestión sin la necesidad de tener recursos de altos costos y, sobre todo, que sean de fácil uso para los usuarios.

Por estas razones, en este proyecto se desarrollará un portal de comercio electrónico, que permita al usuario registrarse, buscar artículos, guardarlos en un carrito de compras, para finalmente adquirirlos. Todo con el objetivo de brindar al usuario una experiencia de compra sencilla y rápida. Para ello, se utilizará herramientas IPC (Inter Process Communication) para la sincronización y comunicación programados mediante lenguaje C.

VI Objetivos que debe cumplir el proyecto

Desarrollar un sistema automatizado, robusto y eficiente, mediante el uso de herramientas IPC, que permita la gestión de una tienda electrónica, facilitando los procesos de compra y gestión de productos, tanto para el usuario cliente como proveedor.

Objetivos específicos

- Permitir el acceso de múltiples usuarios concurrentes de manera ágil
- Lograr la consistencia en los datos del sistema multiusuario
- Diseñar una interfaz de uso sencillo para el usuario
- Diseñar el sistema para ser ejecutado en arquitecturas de recursos bajos
- Proteger la integridad del sistema usando herramientas que restrinjan el acceso según el tipo de usuario.

VII Justificación

El comercio electrónico es de los métodos de compra más recurridos en actualidad, debido a la facilidad y comodidad con la que se puede buscar un producto, comprarlo y recibirlo en casa, además de ser fomentado por la nueva normalidad.

Con el objetivo de incentivar a más usuarios mayores de edad a recurrir a este método de compra, es fundamental que la experiencia del usuario sea fluida.

Para ello se propone el desarrollo de una plataforma de compras electrónica sobresaliente por su rapidez de ejecución, con el objetivo de que el usuario pase menos tiempo esperando una respuesta entre cada solicitud, lo que le permite invertir ese tiempo ahorrado en otras actividades o en más búsquedas de productos.

El lenguaje de programación con el que se programará dicha plataforma es lenguaje C, que se caracteriza por ser más rápido que lenguajes más modernos como Java o Python. Las estructuras de programación que se implementarán serán las de control entre procesos IPC, como semáforos, hilos, procesos y memoria compartida. En lo referente a los conocimientos que se aplicarán en este proyecto destacan los de las áreas de Sistemas Operativos y Algoritmia y programación estructurada.

VIII Marco teórico

Sistema de ventas electrónico

Un sistema de ventas electrónico es un software informático que te permite administrar contactos, mejorar el seguimiento de ofertas de ventas volviéndolo más eficiente y con mayor ahorro de tiempo. Por lo tanto, un sistema de ventas es una herramienta que vuelve más fácil gestionar todos los procesos de ventas de una empresa [1].

Algunas de las características más comunes dentro de estos sistemas son:

- Informe de ventas: el sistema permite realizar el registro y análisis de todos los datos de ventas del negocio
- Gestión del inventario: maneja las cantidades de stock de los productos
- Gestión de clientes: permite hacer registro de clientes para dar un seguimiento a los mismos
- Gestión de proveedores: permite llevar en orden la lista de contactos de proveedores para administrar los productos que se tienen en venta y si es necesario solicitar más artículos para el stock.

Linux

Linux es un sistema operativo: un conjunto de programas que le permiten interactuar con el ordenador y ejecutar otros programas.

Un sistema operativo consiste en varios programas fundamentales que necesita el ordenador para poder comunicar y recibir instrucciones de los usuarios; tales como leer y escribir datos en el disco duro, cintas, e impresoras; controlar el uso de la memoria; y ejecutar otros programas. La parte más importante de un sistema operativo es el núcleo. En un sistema GNU/Linux, Linux es el núcleo. El resto del sistema consiste en otros programas, muchos de los cuales fueron escritos por o para el proyecto GNU. Dado que el núcleo de Linux en sí mismo no forma un sistema operativo funcional, preferimos utilizar el término “GNU/Linux” para referirnos a los sistemas que la mayor parte de las personas llaman de manera informal “Linux” [2].

Linux está modelado como un sistema operativo tipo Unix. Desde sus comienzos, Linux se diseñó para que fuera un sistema multi tarea y multi usuario. Estos hechos son suficientes para diferenciar a Linux de otros sistemas operativos más conocidos. Sin embargo, Linux es más diferente de lo que pueda imaginar. Nadie es dueño de Linux, a diferencia de otros sistemas operativos. Gran parte de su desarrollo lo realizan voluntarios de forma altruista.

Linux puede servir como base para casi todos los tipos de iniciativas de TI, incluidos los contenedores, las aplicaciones nativas de la nube y la seguridad. Es la base de algunos de los sectores y empresas más grandes del mundo, desde los sitios web que comparten conocimientos, como Wikipedia y New York Stock Exchange, hasta los dispositivos móviles que utilizan Android (que es una distribución de uso específico del kernel de Linux con software complementario). Con el transcurso de los años, Linux se ha convertido en el estándar "de facto" para las cargas de trabajo fundamentales, de alta disponibilidad y confiabilidad en los centros de datos y las implementaciones de la nube. Tiene varios casos prácticos, distribuciones, sistemas objetivo, dispositivos y capacidades, y todo se basa en las necesidades del usuario y las cargas de trabajo [3].

Procesos

En [4] se define al proceso como un programa en ejecución y, de una forma un poco más precisa, como la unidad de procesamiento gestionada por el sistema operativo. Un proceso incluye también la actividad actual, que queda representada por el valor del contador de programa y por los contenidos de los registros del procesador. Generalmente, un proceso incluye también la pila del proceso, que contiene datos temporales (como los parámetros de las funciones, las direcciones de retorno y las variables locales), y una sección de datos, que contiene las variables globales. El proceso puede incluir, asimismo, un cúmulo de memoria, que es la memoria que se asigna dinámicamente al proceso en tiempo de ejecución.

Insistamos en que un programa, por sí mismo, no es un proceso; un programa es una entidad pasiva, un archivo que contiene una lista de instrucciones almacenadas en disco (denominado archivo ejecutable), mientras que un proceso es una entidad activa, con un contador de programa que especifica la siguiente instrucción que hay que ejecutar y un conjunto de recursos asociados. Un programa se convierte en un proceso cuando se carga en memoria un archivo ejecutable.

Jerarquía de procesos

De acuerdo con [5], la secuencia de creación de procesos genera un árbol de procesos. Para referirse a las relaciones entre los procesos de la jerarquía se emplean los términos de padre, hijo, hermano o abuelo. Cuando el proceso A solicita al sistema operativo que cree el proceso B, se dice que A es padre de B y que B es hijo de A. Bajo esta óptica, la jerarquía de procesos puede considerarse como un árbol genealógico.

Estado del proceso

A medida que se ejecuta un proceso, el proceso va cambiando de estado. El estado de un proceso se define, en parte, según la actividad actual de dicho proceso. Cada proceso puede estar en uno de los estados mostrados en la Figura 1.

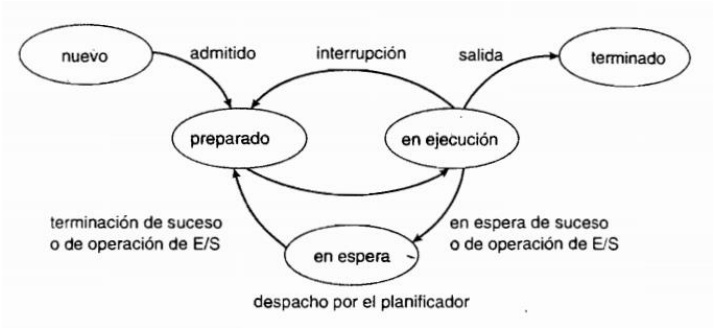

Figura 1 Diagrama de estados de un proceso

- Nuevo: El proceso está siendo creado
- En ejecución: Se están ejecutando las instrucciones
- En espera: El proceso está esperando a que se produzca un suceso (como la terminación de una operación de E/S o la recepción de una señal)
- Preparado: El proceso está a la espera de que le asignen a un procesador
- Terminado: Ha terminado la ejecución del proceso.

Es importante señalar que solo puede haber un proceso ejecutándose en cualquier procesador en cada instante concreto. Sin embargo, puede haber muchos procesos preparados y en espera.

Creación de procesos

Cuando se va a añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso y reserva el espacio de direcciones en memoria principal para el proceso. Estas acciones constituyen la creación de un nuevo proceso [4].

Existen eventos comunes que llevan a la creación de un proceso. En un entorno por lotes, un proceso se crea como respuesta a una solicitud de trabajo. En un entorno interactivo, un proceso se crea cuando un nuevo usuario entra en el sistema. En ambos casos el sistema operativo es responsable de la creación de nuevos procesos. Un sistema operativo puede, a petición de una aplicación, crear procesos.

En [5] se menciona que tradicionalmente, un sistema operativo creaba todos los procesos de una forma que era transparente para usuarios y programas. Esto aún es bastante común en muchos sistemas operativos contemporáneos. Sin embargo, puede ser muy útil permitir a un proceso la creación de otro. Cuando un sistema operativo crea un proceso a petición explícita de otro proceso, dicha acción se denomina creación del proceso. Cuando un proceso lanza otro, al primero se le denomina proceso padre, y al proceso creado se le denomina proceso hijo. Habitualmente, la relación entre procesos necesita comunicación y cooperación entre ellos.

Terminación de procesos

Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización, o que ha completado su tarea. Un trabajo por lotes debe incluir una llamada a un servicio de sistema operativo específica para su terminación. Para una aplicación interactiva, las acciones del usuario indicarán cuando el proceso ha terminado [6]. A continuación, se muestra en la Tabla 1 los motivos más comunes por los que un proceso finaliza.

Tabla 1 Motivos para la terminación de un proceso

Motivo	**Descripción**
Finalización normal	El proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución
Límite de tiempo excedido	El proceso ha ejecutado más tiempo del especificado en un límite máximo. Existen varias posibilidades para medir dicho tiempo. Estas incluyen el tiempo total utilizado, el tiempo utilizado únicamente en ejecución, y, en el caso de procesos interactivos, la cantidad de tiempo desde que el usuario realizó la última entrada.
Memoria no disponible	El proceso requiere más memoria de la que el sistema puede proporcionar.
Violaciones de frontera	El proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.

Error de protección	El proceso trata de usar un recurso, por ejemplo, un fichero, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropia, por ejemplo, escribiendo en un fichero de sólo lectura.
Error aritmético	El proceso trata de realizar una operación de cálculo no permitida, tal como una división por 0, o trata de almacenar números mayores de los que la representación hardware puede codificar.
Límite de tiempo	El proceso ha esperado más tiempo que el especificado en un valor máximo para que se cumpla un determinado evento.
Fallo de E/S	Se ha producido un error durante una operación de entrada o salida, por ejemplo, la imposibilidad de encontrar un fichero, fallo en la lectura o escritura después de un límite máximo de intentos (cuando, por ejemplo, se encuentra un área defectuosa en una cinta), o una operación inválida (la lectura de una impresora en línea).
Instrucción no válida	El proceso intenta ejecutar una instrucción inexistente (habitualmente el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
Instrucción privilegiada	El proceso intenta utilizar una instrucción reservada al sistema operativo.
Uso inapropiado de datos	Una porción de datos es de tipo erróneo o no se encuentra inicializada.
Intervención del operador por el sistema operativo	Por alguna razón, el operador o el sistema operativo ha finalizado el proceso (por ejemplo, se ha dado una condición de interbloqueo).
Terminación del proceso padre	Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
Solicitud del proceso padre	Un proceso padre habitualmente tiene autoridad para finalizar sus propios procesos descendientes.

Hilos

"Un hilo (thread, llamado también proceso ligero) es la unidad de ejecución de un proceso y está asociado con una secuencia de instrucciones, un conjunto de registros y una pila" [7]. Cada que se crea un proceso, el sistema operativo crea un primer hilo, el cual, crea más hilos.

O como lo indica [5], los hilos son objetos dinámicos, los cuales ejecutan una secuencia de instrucciones o pasos que comparten recursos de los procesos.

A diferencia de un proceso, el hilo es el que se ejecuta, mientras que el proceso es el espacio de direcciones donde se encuentra el código que se ejecuta mediante un hilo.

De acuerdo con [7], al igual que los procesos, los hilos pueden tener los siguientes estados:

1) Nuevo: se crea el hilo sin ser activado
2) Preparado: Al activarse un hilo, espera que se le sea asignado un UCP (unidad central de proceso)
3) Ejecución: se le asigna una UCP

4) Bloqueado: es cuando un hilo espera que otro elimine el bloqueo, el cual puede tener los estados:
    a) Dormido: el hilo se bloquea por un periodo de tiempo, para después despertar y pasar al estado 2)
    b) Esperando: el hilo espera a que pase alguna cosa: condición o una operación, para después pasar al estado 2).
5) Muerto: El hilo finaliza su ejecución y en este estado ya no puede pasar a ningún otro.

Herramientas del IPC (Inter Process Communication) de UNIX-Linux

La comunicación entre procesos o Inter Process Communication (IPC) es un mecanismo que permite a los procesos comunicarse con sus iguales y sincronizar sus acciones [8].

Algunas de las cuestiones con que trata es: "cómo un proceso puede pasar información a otro", cómo lograra que dos procesos no se interpongan entre sí, y como obtener la secuencia apropiada de ejecución cuando hay dependencias presentes [6].

De acuerdo con [6] las últimas dos cuestiones se tratan de igual forma en los hilos, es decir, el manejo de dependencias y colisiones. Mientras que el paso de información es más fácil, ya que los hilos "comparten un espacio de direcciones común".

Los mecanismos de la IPC del UNIX System V son: "semáforos que permiten sincronizar procesos; la memoria compartida que permite que los procesos compartan su espacio de direcciones virtuales; y las colas de mensajes, que posibilitan el intercambio de datos con un formato determinado" [9].

Dichos mecanismos presentan características comunes como: "cada mecanismo tiene una tabla cuyas entradas describen el uso que se hace del mismo"; "cada entrada de la tabla tiene una llave numérica elegida por el usuario"; "cada mecanismo dispone de una llamada get para crear una entrada nueva o recuperar alguna ya existente"; "cada entrada de la tabla tiene un registro de permisos que incluye: identificador de usuario y grupo del proceso que ha reservado la entrada, identificador de usuario y de grupo modificados por la llamada de control del mecanismo IPC, un conjunto de bits con los permisos de lectura, escritura y ejecución para el usuario, el grupo y otros usuarios"; "cada mecanismo IPC tiene una llamada de control que permite leer y modificar el estado de una entrada reservada y también permite liberarla".

Semáforos

En [5] se menciona que un semáforo es un mecanismo de sincronización que se utiliza generalmente en sistemas con memoria compartida, bien sea un monoprocesador o un multiprocesador. Su uso en una multicomputadora depende del sistema operativo en particular.

Un semáforo es un objeto con un valor entero, al que se le puede asignar un valor inicial no negativo y al que sólo se puede acceder utilizando dos operaciones atómicas: wait y signal. Las definiciones de estas dos operaciones son las siguientes:

```
wait (s) {
  s = s – 1;
  if (s < 0)
     Bloquear al proceso;
}
signal (s){
  s = s + 1;
```

```
  if (s <= 0)
     Desbloquear a un proceso bloqueado en la operación wait;
}
```

La operación wait también recibe otros nombres como down o P. La operación signal recibe también nombres como up o V.

Cuando el valor del semáforo es menor o igual que cero, cualquier operación wait que se realice sobre el semáforo bloqueara al proceso. Cuando el valor del semáforo es positivo, cualquier proceso que ejecute una operación wait no se bloqueará.

El número de procesos, que en un instante determinado se encuentran bloqueados en una operación wait, viene dado por el valor absoluto del semáforo si es negativo. Cuando un proceso ejecuta la operación signal, el valor del semáforo se incrementa. En el caso de que haya algún proceso bloqueado en una operación wait anterior, se desbloqueara a un solo proceso.

Memoria compartida

De acuerdo con [9], la manera más fácil y rápida de comunicar procesos, es hacer que compartan la zona de memoria, ya que, al escribir dentro de la memoria, automáticamente se hacen disponibles para ser leídas por otro proceso.

Entonces, una memoria compartida permite que dos o más procesos compartan una región dada de memoria. De esta manera, los datos no necesitan ser copiados entre procesos, dando como resultado, la forma más rápida de IPC (Inter-Process Communication).

Pero existe un inconveniente, cuando los datos que usan memoria compartida son muy grandes, probablemente cuando un proceso intente escribir, otro estará leyendo esa parte de la memoria, y si pasa esto, al momento de leer, los datos parecerán incoherentes. Para resolver este problema, se hace uso de semáforos, para que, si un proceso intenta leer o escribir al mismo tiempo que otra, no le permita acceder a ella, hasta que termine el proceso de alguno de los procesos.

Colas de mensajes

Los mensajes, tal como se menciona en [7], proporcionan una solución al problema de la concurrencia de procesos que integra la sincronización y la comunicación entre ellos y resulta adecuado tanto para sistemas centralizados como para distribuidos. Esto hace que se incluyan prácticamente en todos los sistemas operativos modernos y que en muchos de ellos se utilicen como base para todas las comunicaciones del sistema, tanto dentro del computador como en redes de computadoras.

La comunicación mediante mensajes necesita siempre de un proceso emisor y de uno receptor y la información que debe intercambiarse. Las operaciones básicas para una comunicación mediante mensajes que proporciona todo sistema operativo son: enviar (mensaje) y recibir (mensaje). Las acciones de transmisión de información y de sincronización se ven como actividades inseparables.

La comunicación por mensajes requiere que se establezca un enlace entre el receptor y el emisor, la forma de este puede variar de sistema a sistema.

El proceso de denominación de las tareas para la comunicación por mensajes se puede realizar de dos formas: directa e indirectamente.

En la comunicación directa ambos procesos, el que envía y el que recibe, nombran de forma explícita al proceso con el que se comunican. Este método de comunicación establece un enlace entre dos procesos que se desean comunicar, proporcionando seguridad en el intercambio de mensajes, ya que cada proceso debe conocer la identidad de su pareja en la comunicación, pero, por este motivo, no resulta muy adecuado para el diseño de rutinas de servicio de un sistema operativo.

En el método de comunicación indirecta los mensajes se envían y reciben a través de una entidad intermedia (estructura de datos) que recibe el nombre de buzón, puerto o cola de mensajes. Una cola de mensajes es un objeto en el que los procesos dejan mensajes y del cual pueden tomarse por otros procesos. Ofrecen una mayor versatilidad que en el caso del nombramiento directo, ya que permiten comunicación de uno a uno, de uno a muchos, de muchos a uno y de muchos a muchos.

Cada cola de mensajes tiene un identificador que lo distingue. En [4] se define que las operaciones básicas de comunicación tienen la forma:

enviar(cola_de_mensajes, mensaje): envía un mensaje a la cola de mensajes

recibir(cola_de_mensajes, mensaje): recibe un mensaje de la cola de mensajes

La cola de mensajes establece un enlace que puede ser utilizado por más de dos procesos y que al mismo tiempo permite que la comunicación de un proceso con otro se pueda realizar mediante distintas colas de mensajes.

Archivos

Un archivo es una colección de información relacionada, con un nombre, que se graba en el almacenamiento secundario, cada elemento de dicha colección es mapeado por el sistema operativo sobre los dispositivos físicos de almacenamiento, los cuales se caracterizan por ser no volátiles [4].

Los archivos pueden almacenar programas y datos que pueden ser numéricos, alfabéticos, alfanuméricos o binarios, generalmente poseen los siguientes atributos: nombre, identificador único, tipo de archivo, ubicación, protección (control de acceso), tamaño, fecha, hora e identificación del creador [4].

Las operaciones que se pueden realizar con los archivos son: creación, escritura, lectura, reposicionamiento dentro de un archivo, borrado y truncado (borrado de contenido manteniendo atributos).

Existen sistemas de archivos que permiten realizar todas las operaciones previamente mencionadas sin necesidad de un profundo conocimiento del sistema, dentro de dichos gestores se pueden crear directorios (o carpetas) cuyo objetivo es agrupar un conjunto de archivos [6].

Dispositivos de entrada y salida

Actualmente las computadoras están compuestas por una serie de dispositivos capaz de interactuar con la computadora [5].

Los dispositivos de entrada y salida son componentes que se conectan a la computadora que permiten el ingreso y egreso de información.

Los dispositivos de entrada ingresan datos hacia la computadora, como pueden ser teclado, mouse, cámara de video, etc.

Mientras que los dispositivos obtienen información de la computadora que pueden ser traducidos en imagen, video o simplemente acceso de los datos como audífonos, pantalla, impresoras, memorias, etc.

IX Análisis y diseño

El proyecto de la tienda online se desarrollará usando lenguaje C para la codificación del sistema y será ejecutable en sistemas basados en Linux.

El sistema se compone por dos programas: el programa de control y el programa de usuario.

El programa de control es el encargado de hacer la gestión de archivos y la sincronización de usuarios. El programa de usuario es el que permitirá el acceso tanto a clientes como proveedores, permitiendo que hagan compras y gestionen productos respectivamente. El programa de usuario se ejecutará en múltiples terminales permitiendo el acceso multiusuario.

Para que los usuarios puedan acceder al sistema de control es necesario que inicien sesión en el mismo. Para ello se tendrá un archivo con ciertos usuarios predeterminados que estarán descritos de la siguiente forma:

Nombre de usuario – Contraseña – Tipo de usuario

El nombre de usuario y contraseña serán cadenas de caracteres, excluyendo el uso del guión (-). El tipo de usuario es representado por un número "0" en caso de ser un usuario "cliente" y con "1" para usuarios "proveedor", ya que dependiendo del tipo de usuario se pueden realizar diferentes operaciones en el sistema. El uso de guiones dentro de la descripción del usuario sirve para poder separar de manera más fácil los datos y poder extraerlos del archivo.

El programa de control es el encargado de extraer los datos del archivo de usuarios e insertar cada usuario dentro de un nodo para generar una lista enlazada, tener los usuarios en la lista nos permitirá buscar de manera sencilla los usuarios en el proceso de inicio de sesión. Un ejemplo de la lista enlazada de usuarios se muestra en la Figura 2.

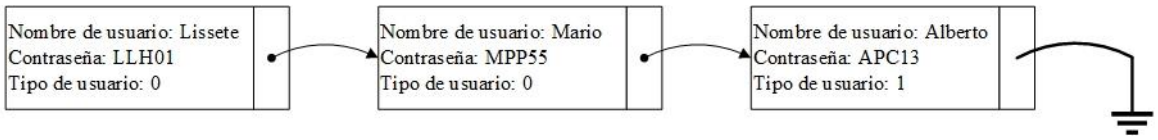

Figura 2 Ejemplo de lista enlazada de usuarios

Una vez que se haya generado la lista, por medio de memoria compartida, cada programa de usuario podrá acceder a la lista y podrá realizar el inicio de sesión con su respectivo proceso de verificación.

Al momento en el que se inicia sesión, el programa usuario podrá acceder a la clave de los semáforos, que consistirá en un carácter almacenado en una variable dentro de la memoria compartida.

Para la gestión de los productos que hay en la tienda se tendrá un archivo de catálogo, en el cual se almacenaran los productos de la siguiente forma:

Nombre de producto – Número de existencias – Nombre del proveedor

El nombre del producto y nombre del proveedor será una cadena de caracteres, donde el nombre del proveedor será asignado según el usuario que esté con la sesión iniciada al momento de agregar el artículo. El número de existencias se representará con un número entero positivo. El usuario proveedor puede aumentar la cantidad de sus productos en el inventario.

A partir del archivo de productos, el programa de control generará una lista enlazada donde cada producto será un nodo, de forma similar a lo que se realizó con los usuarios. De igual manera cada usuario podrá acceder a esta lista por medio de memoria compartida, permitiendo que hagan búsquedas, compras y gestión de productos según el tipo de usuario que sean. Un ejemplo de lista enlazada es mostrado en la Figura 3.

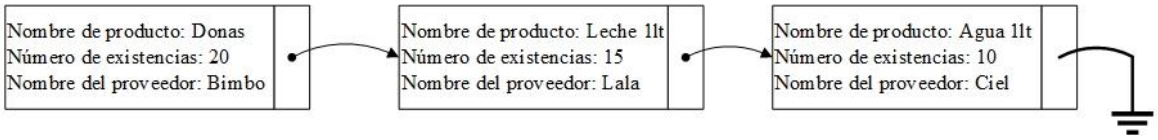

Figura 3 Ejemplo de lista enlazada de productos

Conforme van iniciando sesión los usuarios, comienzan la comunicación con el programa de control a través de una cola de mensajes, indicando las operaciones que desea realizar en el sistema.

Para que cada usuario tenga la oportunidad de realizar sus operaciones en el sistema se hará uso de una sincronización por semáforos, al momento en el que un usuario tiene acceso al sistema, todos los demás usuarios tendrán que estar en espera de su turno. Esto permitirá que los datos se mantengan consistentes ya que como en algunos gestores de bases de datos, no se puede escribir al mismo tiempo por múltiples usuarios. El proceso de control y sincronización de usuarios es representado en la Figura 4.

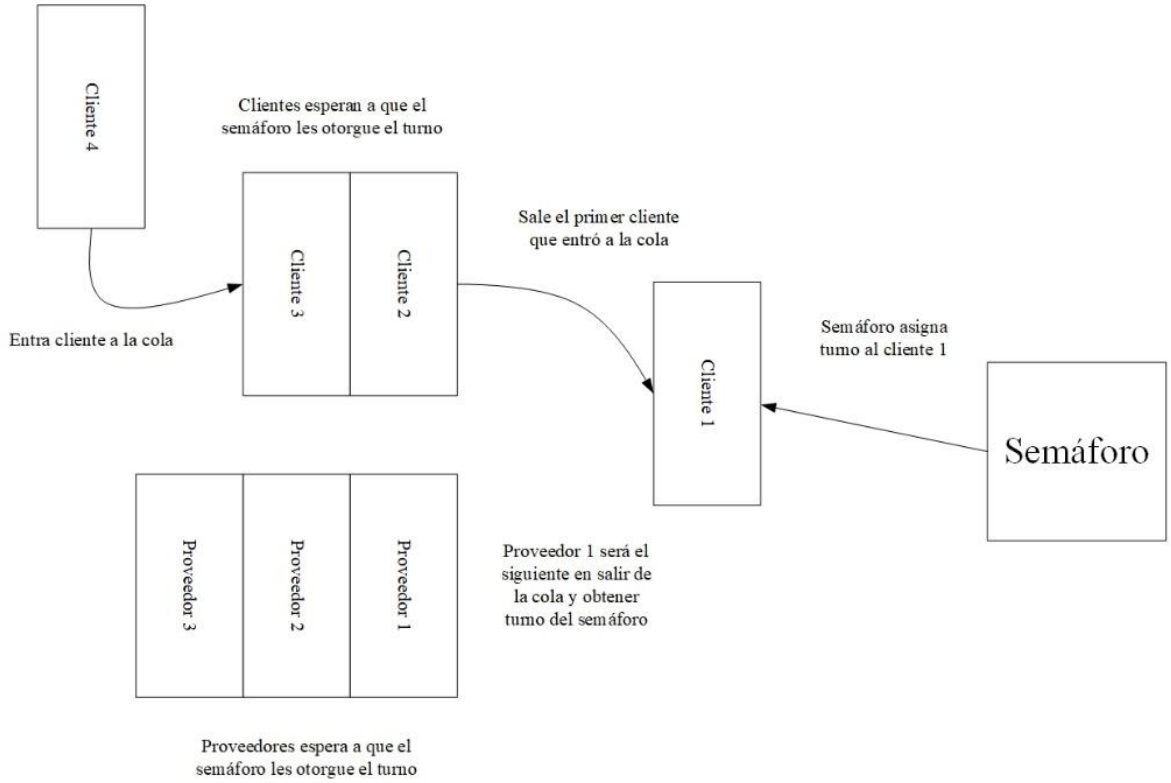

Figura 4 Proceso de control y sincronización de usuarios

Para realizar la compra de productos, un cliente deberá seleccionar el artículo y el número de unidades que desee comprar, dichos datos se almacenaran en un nodo de tipo producto.

Cuando se confirma la compra de los artículos, se procede a actualizar los datos del número de existencia del producto que se compró. Posteriormente, para llevar un registro de todas las ventas realizadas se almacenarán todas las compras efectuadas por los clientes en un archivo de carritos. El archivo de clientes guardará las ventas de la siguiente forma:

Nombre del producto – Nombre de proveedor – Cantidad – Nombre de cliente – Fecha – ID de compra

Estos datos se obtienen respectivamente del archivo de carritos y de la lista de usuarios. La fecha se obtendrá directo del sistema. El ID de compra será un número de 5 dígitos que comenzará desde el "00000" y se incrementará cada que se ingresen las ventas de un nuevo carrito. El proceso de compra es representado mediante la Figura 5.

-

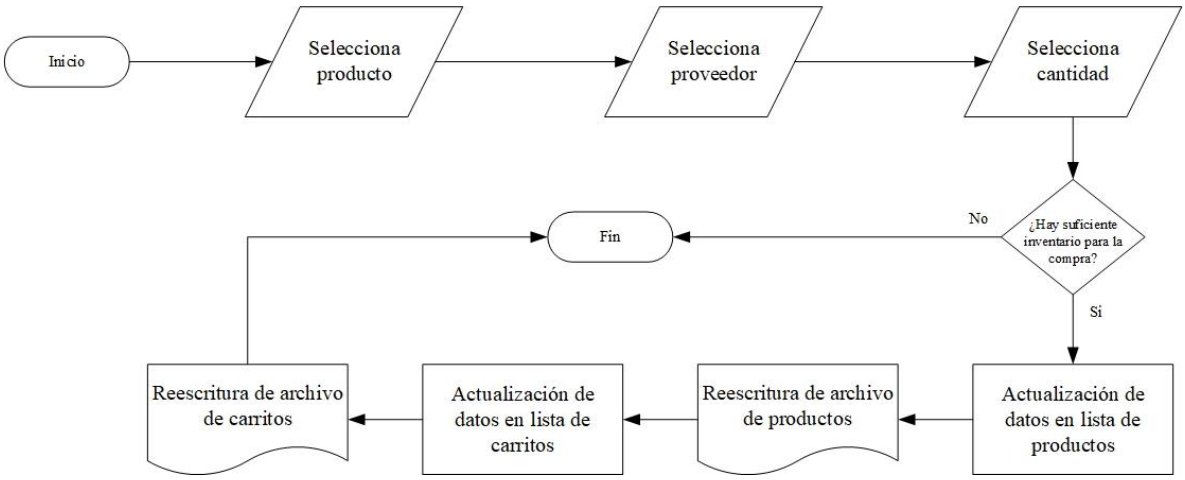

Figura 5 Proceso de compra

X Implementación y pruebas

La escritura del código comenzó con la creación de listas enlazadas que se requieren para la implementación de los archivos de productos, usuarios y carrito de compras.

Se crearon tres listas enlazadas denominadas: listaCarrito, listaUsuario, y listaProductos. La primera es ocupada para almacenar los artículos que un usuario adquiera, la segunda se requiere para almacenar los usuarios registrados en la tienda, y la última lista guarda el catálogo de productos con el respectivo número de existencias de cada artículo ofertado.

Las tres listas realizan operaciones de creación e inicialización de lista, inserción de nodos, y eliminación de nodos. La principal diferencia entre las listas es el tipo de dato que almacena cada nodo.

La listaUsuarios almacena en cada nodo los datos: nombre de usuario, contraseña, tipo de usuario, y si la sesión se encuentra activa o no. Los datos son almacenados en una estructura llamada usuario, cuyo código se muestra en la Figura 6.

```
typedef struct
{
  char nombre_usuario[TAMMAX];
  char password[TAMMAX]; //Contrasenia de inicio de sesion
  int tipo_usuario; //Indica si el usuario es comprador o vendedor
  int sesion_activa; //Indica si el usuario ha iniciado sesion
} usuario;
```

Figura 6 Estructura usuario

La listaProductos almacena en cada nodo los datos: nombre de producto, número de existencias del producto, y el nombre de proveedor. Los datos son almacenados en una estructura llamada producto, cuyo código se muestra en la Figura 7.

```
typedef struct
{
  char nombre_producto[TAMMAX];
  int num_existencia;
  char nombre_proveedor[TAMMAX];
} producto;
```

Figura 7 Estructura producto

La listaCarrito almacena en cada nodo los datos: nombre de producto, nombre de proveedor y la cantidad de artículos comprados. Los datos son almacenados en una estructura llamada producto, cuyo código se muestra en la Figura 8.

```
typedef struct
{
  char* nombre_producto;
  char* nombre_proveedor;
  int cantidad;
} carrito;
```

Figura 8 Estructura carrito

La listaProductos aparte de incluir las operaciones comunes entre las tres listas contiene una función específicamente usada para la operación de compra de artículos. El prototipo de dicha función es representado mediante la Figura 9.

```
int dis_existencias(listaProductos* l, char* nombre_producto,char* nombre_proveedor, int art_comprados);
```

Figura 9 Prototipo dis_existencias

El objetivo de la función dis_existencias es encontrar el artículo especificado por el usuario al momento de realizar una compra y disminuir el número de existencias en la cantidad que el usuario compre.

Para ello se utiliza la sentencia if, que recorre la lista y determina si el nodo analizado es el buscado, una vez encontrado el artículo deseado se procede a disminuir la cantidad de existencias. Como es mostrado en la Figura 10.

```
while (l1)
{
  //En caso de que el articulo buscado sea encontrado
  if (strcmp(nombre_producto, l1->dato->nombre_producto) == 0 && strcmp
  (nombre_proveedor, l1->dato->nombre_proveedor) == 0)
  {
    if (art_comprados > l1->dato->num_existencia)
    {
      printf("\nCantidad invalida :'c\n");
      return CANT_INV;
    }
    else
    {
      //El numero de existencias del articulo disminuira en funcion de la cantidad
      de elementos comprados

      l1->dato->num_existencia = l1->dato->num_existencia - art_comprados;
      return i;
    }
  }
```

Figura 10 Función dis_existencias

Después de crear las estructuras de datos se recurrió a escribir el código de la cola de mensajes que se requiere para comunicar al usuario con el sistema.

Los mensajes enviados en la cola de mensajes son definidos en el archivo queueMensaje.h, específicamente en la estructura *mensaje*, cuyo código se muestra en la Figura 11, en la que podemos observar que se requiere de un ID de proceso, un entero *orden* que indique el tipo de acción que el usuario desea realizar, la estructura *usuario* que identifica quien envía el mensaje, la clave del semáforo para el control de los procesos y el producto requerido para la función de compras.

```
typedef struct
{
  long pid; //ID del proceso
  int orden; //Indica el numero de pedido realizado por un usuario
  usuario user;
  producto prod;
  char CLAVE_SEM;//clave que usara el semaforo
} mensaje;
```

Figura 11 Estructura mensaje

En el mismo archivo queueMensaje.h se definen las constantes del 1 al 9, donde: el rango de opciones partiendo del 1 al 6 simbolizan las funciones que puede realizar el usuario; las opciones 7 y 8 son constantes usadas como banderas en la cola de mensajes para reflejar el estatus de la transmisión del mensaje, la última opción (9) se usa como bandera para validar que el usuario haya iniciado sesión. El código de esto es mostrado en la Figura 12.

```
#define SESION          1
#define VER_PRODUCTOS   2
#define BUY_PRODUCTOS   3
#define ADD_PRODUCTOS   4
#define MOD_PRODUCTOS   5
#define HIS_PRODUCTOS   6
#define FIN             7
#define ERROR           8
#define INICIADO        9
```

Figura 12 Acciones y estatus del usuario en la cola de mensajes

Para definir las acciones a realizar con la cola de mensajes se requirieron dos archivos, nombrados: mainControl.c y mainUsuario.c. En el primero se definen las instrucciones desde la vista del sistema, y en el segundo desde la vista de usuario.

En el archivo mainControl.c se comienza por la obtención de la llave para ser usada en cola de mensajes de usuario a control, mediante las líneas de código mostradas en la Figura 13.

```
llave = ftok(FICHERO_LLAVE, CLAVE_USER_CONTROL);
if (llave == -1)
{
    printf("La llave no fue creada");
    exit(0);
}
```

Figura 13 LLave usuario-control

Subsecuentemente se obtiene el identificador de la cola de mensajes con la llave mostrada en la Figura 13. El código para la obtención del ID se representa en la Figura 14.

```
if ((cola_userctrl = msgget(llave, IPC_CREAT | PERMISOS)) == -1)
{
    //En caso de que el proceso de creacion del identificador de la cola de
    mensajes tenga un error.
    perror("El identificador de cola de mensajes de usuarios no fue creado");
    exit(-1);
}
```

Figura 14 Identificador de cola de mensajes. mainControl.c

El mismo código mostrado en la Figura 13 y Figura 14 se usa para la cola de mensajes de control a usuario, como se muestra en la Figura 15.

```
/*Creacion de llave para cola de mensajes de control a usuario*/
llave = ftok(FICHERO_LLAVE, CLAVE_CONTROL_USER);

/*Obtencion del identificador de la cola de mensajes del programa de control*/
if ((cola_ctrluser = msgget(llave, IPC_CREAT | PERMISOS)) == -1)
{
    //En caso de que el proceso de creacion del identificador de la cola de
    mensajes tenga un error.
    perror("El identificador de cola de mensajes de control no fue creado");
    exit(-1);
}
```

Figura 15 Cola de mensajes control-usuario

Se procede a crear un usuario mediante la línea de código mostrada en la Figura 16.

```
users = crearUsuarios();
```

Figura 16 Creacion de usuario. mainControl.c

Luego, se creó la memoria compartida que permite compartir el catálogo de productos entre usuario y sistema. El respectivo código se muestra en la Figura 17.

```c
int shmid;
llave = ftok(FICHERO_MEMORIA, CLAVE_MEMORIA);
shmid = shmget(llave, sizeof(char *), IPC_CREAT | 0600);
cadProductos = shmat(shmid, 0, 0);
```

Figura 17 Memoria compartida Productos. mainControl.c

De igual forma se crea la memoria compartida para la lista de carritos, donde muestra el historial de compras que han hecho los usuarios, como se muestra en la Figura 18.

```c
/*Memoria compartida para Carrito*/
int shmid2;
llave = ftok(FICHERO_MEMORIA_CARRITO, CLAVE_MEMORIA_CARRITO);
shmid2 = shmget(llave, sizeof(char *), IPC_CREAT | PERMISOS);
cadCarrito = shmat(shmid2, 0, 0);
```

Figura 18 Memoria compartida Carrito mainControl.c

Una vez creada la memoria compartida y la cola de mensajes se procede a definir el comportamiento de la última cuando recibe un mensaje. Para ello se recurrió a la sentencia switch que recibe como parámetros los valores de msg.orden que simbolizan las acciones solicitadas por el usuario.

La Figura 19 muestra el caso en el que el usuario solicita iniciar sesión, para ello primero se valida que los datos ingresados sean correctos, en caso de serlo se muestra en la pantalla el mensaje “El usuario fue encontrado”, en caso contrario mostrará un mensaje de error. Para finalizar se envía de regreso un mensaje a la cola de mensajes, el cual incluye la orden que será iniciada sus sesión y la clave para el uso de los semáforos para el control de los procesos, este mensaje es para indicar que la solicitud del usuario fue recibida con éxito.

```c
switch (msg.orden)
{
case SESION:
    printf("\nEl usuario: %lu solicito iniciar sesion\n", msg.pid);
    //Validación de que el usuario exista
    if (validar_lisUsuarios(&users, msg.user.nombre_usuario, msg.user.password, msg.user.tipo_usuario) < 0)
    {
        //En caso de no encontrar los datos de usuario ingresados
        msg.orden = ERROR;
        printf("\nEl usuario no fue encontrado:\n");
        printf("nombre:%d\n", msg.user.tipo_usuario);
    }
    else
    {
        //En caso de que si se encuentren los datos de usuario ingresados
        msg.orden = INICIADO;
        msg.CLAVE_SEM = CLAVE;
        printf("\nEl usuario fue encontrado\n");
    }
    //Regresar mensaje para indicar que fue recibido correctamente
    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 19 Case SESION

La Figura 20 muestra el caso en el que el usuario solicita ver el catálogo de productos, para ello se procede a convertir a cadena la lista de productos, y al finalizar se envía de regreso un mensaje indicando que la solicitud del usuario fue recibida correctamente.

```
case VER_PRODUCTOS:
    //Fin indica la recepcion exitosa del mensaje
    msg.orden = FIN;
    strcpy(cadProductos, "");
    str_lisProductos(cadProductos, &productos);
    printf("\n %s", cadProductos);
    printf("\nEl usuario: %lu solicito ver productos\n", msg.pid);
    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 20 case VER_PRODUCTOS

La Figura 21 muestra el caso en el que el usuario sea un proveedor y necesite agregar productos, para ello se procede a crear una lista temporal que almacenará el nuevo producto y después los elementos de dicha lista serán agregados a la lista de productos, todo ello mediante la función crearProductos(). Posteriormente se convierte a cadena la lista de productos y se envía un mensaje de regreso indicando que la solicitud del usuario fue recibida con éxito.

```
case ADD_PRODUCTOS:
    msg.orden = FIN;
    productos = crearProductos();
    strcpy(cadProductos, "");
    str_lisProductos(cadProductos, &productos);
    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 21 case ADD_PRODUCTOS

En la Figura 22 se contempla el caso de error en el que el usuario no ha iniciado sesión, para advertirle esto se imprime en pantalla el mensaje “El usuario no ha iniciado sesion”.

```
case ERROR:
    msg.orden = ERROR;
    printf("\nEl usuario no ha iniciado sesion\n");
    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 22 case ERROR

En la Figura 23 se define el caso en el que un usuario desea adquirir artículos, para ello se disminuye el número de existencias, si esto se realizó con éxito (es decir que el producto existe y hay suficiente existencia) se obtiene la fecha en la que se compró el producto y se guarda dentro del archivo de carritos con sus respectivos datos; se actualiza la lista de carritos, la lista de productos y el archivo .txt en el que se guardan los datos de la lista de productos. Para finalizar, se envía un mensaje a la cola de mensajes que indica que la solicitud del usuario fue recibida exitosamente.

```
case BUY_PRODUCTOS:
    printf("\nEl usuario: %lu solicito comprar productos\n", msg.pid);
    msg.orden = FIN;
    /*Si hay suficientes existencias del articulo la compra procede*/
    if (dis_existencias(&productos, msg.prod.nombre_producto, msg.prod.nombre_proveedor, msg.prod.
    num_existencia) > 0)
    {
        FILE *car;
        time_t t;
        struct tm *tm;
        char fechayhora[100];
        /*Obtencion de la fecha exacta cuando se realizo la compra*/
        t = time(NULL);
        tm = localtime(&t);
        strftime(fechayhora, 100, "%d/%m/%Y %H:%M", tm);

        car = fopen("carritos.txt", "a");
        if (car == NULL)
            printf("Hubo un error al abrir el archivo .txt");
        fprintf(car, "\n%s-%s-%d-%s-%s-%05d", msg.prod.nombre_producto, msg.prod.nombre_proveedor, msg.
        prod.num_existencia, msg.user.nombre_usuario, fechayhora, ++id);
        //Nombre del producto – Nombre de proveedor – Cantidad – Nombre de cliente – Fecha – ID de compra

        fclose(car);
        carritos = crearCarritos();
        strcpy(cadCarrito, "");
        str_lisCarrito(cadCarrito, &carritos);
        printf("\n %s", cadCarrito);

        strcpy(cadProductos, "");
        str_lisProductos(cadProductos, &productos);
        actualizar_Productos(cadProductos);
    }
    else
    {
        printf("\nNo esta el producto o no hay suficientes\n");
        msg.orden = ERROR;
    }
    /*se envia el mensaje a la cola*/
    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 23 case BUY_PRODUCTOS

La Figura 24 muestra el caso que nos permite hacer modificaciones a la cantidad de existencias de un producto. El primer paso consiste en validar que el usuario que intenta modificar las existencias sea el proveedor de dicho producto. En caso de que no sea el proveedor, se envía por la cola de mensajes una orden de error. Por el contrario, cuando se identifica que el proveedor coincide con el usuario que solicita la modificación, se procede a hacer el aumento de las unidades establecidas por el usuario en el mainUsuario.c. Dado que la función dis_existencias() disminuye las unidades de los productos, se multiplica el parámetro de unidades por un “-1”, logrando así el aumento de las existencias. Por último, es necesario actualizar el archivo de productos, por lo cual es necesario hacer uso de la función actualizar_Productos().

```
case MOD_PRODUCTOS:

    if(buscar_lisProductos(&productos,msg.prod.nombre_producto,msg.prod.
    nombre_proveedor)){
        msg.orden = FIN;
        dis_existencias(&productos, msg.prod.nombre_producto, msg.prod.
        nombre_proveedor, (-1)*msg.prod.num_existencia);
        strcpy(cadProductos, "");
        str_lisProductos(cadProductos, &productos);
        actualizar_Productos(cadProductos);
    }else{
        msg.orden=ERROR;
    }

    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 24 case MOD_PRODUCTOS

En la Figura 25 muestra en el caso que el usuario solicite ver el archivo de carritos (historial de compras), primero limpia la cadena que muestra la lista de carritos, accede a la lista y lo convierte de nuevo a cadena para sobrescribir lo que esté dentro de la memoria compartida.

```
case HIS_PRODUCTOS:
    msg.orden = FIN;
    strcpy(cadCarrito, "");
    str_lisCarrito(cadCarrito, &carritos);
    printf("\n %s", cadCarrito);
    printf("\nEl usuario: %lu solicito ver carrito de compras\n", msg.pid);
    msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
    break;
```

Figura 25 case HIS_PRODUCTOS

En el archivo mainUsuario.c se encuentra un fragmento del menú de usuario, el cual comienza con solicitarle al usuario que indique la operación que desea realizar, el código de esto se muestra en la Figura 26.

```
printf("\nEscribe la opcion que desees:\n");
scanf("%d", &orden);
```

Figura 26 Menú mainUsuario.c

La operación seleccionada por el usuario se almacena en la variable orden, la cual es recibida como parámetro en la sentencia switch.

Para el caso de inicio de sesión se requerirán los datos de usuario y se enviarán como mensaje a la cola de mensajes, todo esto se logra mediante las líneas de código de la Figura 27.

```
case SESION:
    recibir = SI;
    msg.orden = orden;
    userActual = ini_sesion();
    msg.user = userActual;

    printf("nombre:%s\n", msg.user.nombre_usuario);
    printf("pass:%s\n", msg.user.password);
    printf("tipo:%d\n", msg.user.tipo_usuario);
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;
```

Figura 27 case SESION mainUsuario.c

La Figura 28 muestra el caso de ver productos, para el cual se validará que el usuario haya iniciado sesión antes de que el catálogo de productos sea mostrado. Al finalizar se enviará un mensaje a la cola de mensajes con los datos necesarios.

Cuando más de un usuario desea acceder a la misma función, ya sea ver catálogo de productos o comprar algún artículo deberá esperar a que el semáforo le indique su turno, esto con el objetivo de evitar la inconsistencia de datos.

El semáforo implementado permite que solamente un usuario realice la misma función al mismo tiempo, una vez que dicho usuario termine, el semáforo le indicara al siguiente usuario en la cola que es su turno.

```
case VER_PRODUCTOS:
/*Se bloquea el semaforo*/
    down(semaforo);
    if (userActual.sesion_activa)
    {
        /*Visualizacion de los productos en stock*/
        recibir = SI;
        msg.orden = orden;
        printf("\n%s\n", cadProductos);
    }
    else
    {
        /*Error cuando el usuario no ha iniciado sesion*/
        recibir = SI;
        msg.orden = ERROR;
        printf("\nNo has iniciado sesion -_-\n");
    }
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;
```

Figura 28 case VER_PRODUCTOS mainUsuario.c

El caso de comprar productos es representado mediante la Figura 29, en el cual se pide al usuario que ingrese los datos necesarios al usuario para identificar el artículo que desea comprar, y la cantidad. Dichos datos serán mandados al sistema mediante la cola de mensajes.

```
case BUY_PRODUCTOS:
/*Se bloquea el semaforo*/
    down(semaforo);
    /*Se verifica que se haya iniciado sesion y que el usuario sea un cliente*/
    if (userActual.sesion_activa && !userActual.tipo_usuario)
    {
        char producto[TAMMAX];
        char proveedor[TAMMAX];
        int numEx;

        recibir = SI;
        msg.orden = orden;
        /*Recepcion de datos ingresados por el usuario ára señalar que articulo desea comprar*/
        strcpy(msg.user.nombre_usuario, userActual.nombre_usuario);
        strcpy(producto, "");
        strcpy(proveedor, "");
        printf("\nEscribe el nombre del producto: ");
        scanf("%s", producto);
        strcpy(msg.prod.nombre_producto, producto);
        printf("\nEscribe el nombre del proveedor: ");
        scanf("%s", proveedor);
        strcpy(msg.prod.nombre_proveedor, proveedor);
        printf("\nEscribe la cantidad deseada: ");
        scanf("%d", &numEx);
        msg.prod.num_existencia = numEx;
    }
    else
    {
        /*Error arrojado por el sistema cuando un usuario desea comprar articulos sin antes iniciar sesion*/
        recibir = SI;
        msg.orden = ERROR;
        printf("\nNo has iniciado sesion o no eres Cliente\n");
    }
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;
```

Figura 29 case BUY_PRODUCTOS. mainUsuario.c

En la Figura 30 se muestra un fragmento del código del caso en el que se deseen comprar artículos, para ello primero se valida que el usuario haya iniciado sesión previamente, posteriormente se piden los datos de cada producto, como el nombre de producto y la cantidad a ofertar, el proveedor será determinado automáticamente por el sistema en función del usuario que agregue el artículo. Y al final se preguntará al usuario si desea agregar más artículos, en caso de que sí, se requerirán los datos mencionados previamente.

Una vez que se agreguen todos los datos, estos serán enviados al sistema mediante la cola de mensajes.

```c
do
{
    printf("\nEscribe el nombre del producto: ");
    scanf("%s", nombre);

    printf("\nEscribe la cantidad: ");
    scanf("%d", &numEx);

    fprintf(f, "\n%s-%s-%d", nombre, userActual.nombre_usuario,
    numEx);

    printf("\nQuiere agregar otro producto? s/n: ");
    scanf("%s", &continuar);
} while (continuar == 's');
msgsnd(cola_userctrl, &msg, LONGITUD, 0);
fclose(f);
```

Figura 30 case ADD_PRODUCTOS mainUsuario.c

La Figura 31 muestra el código del caso en el que el usuario es un proveedor y selecciona la opción de modificar productos, entonces le es pedido ingresar el nombre de productos y la cantidad de artículos a incrementar.

```c
case MOD_PRODUCTOS:
    if (userActual.sesion_activa && userActual.tipo_usuario)
    {
        char producto[TAMMAX];
        int numEx;

        recibir = SI;
        msg.orden = orden;
        strcpy(producto,"");
        printf("\nEscribe el nombre del producto: ");
        scanf("%s", producto);
        strcpy(msg.prod.nombre_producto,producto);
        strcpy(msg.prod.nombre_proveedor,userActual.nombre_usuario);
        printf("\nEscribe la cantidad a incrementar: ");
        scanf("%d",&numEx);
        msg.prod.num_existencia = numEx;
    }
```

Figura 31 case MOD_PRODUCTOS mainUsuario.c

En el caso que el usuario solicite ver el archivo de carritos, se verifica que el usuario exista, si es el caso, se podrá acceder a la memoria compartida donde está el archivo de carritos y podrá mostrarlo en pantalla, como se muestra en la Figura 32.

```
case HIS_PRODUCTOS:
    if (userActual.sesion_activa)
    {
        recibir = SI;
        msg.orden = orden;
        printf("\n%s\n", cadCarrito);
    }
    else
    {
        /*Error arrojado cuando un usuario intenta acceder
        al historial de compras y no ha iniciado sesion*/
        recibir = SI;
        msg.orden = ERROR;
        printf("\nno has iniciado sesion\n");
    }
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;
```

Figura 32 case HIS_PRODUCTOS mainUsuario.c

Posteriormente, el archivo control.c se utilizó para la definición de las funciones que se usarán el programa mainControl.c.

La primera función creaUsuarios() crea una lista de usuarios temporal a partir de la lectura de un archivo .txt para después ser almacenada en la lista que estamos utilizando, para ello, leemos todo el archivo separando cada cadena encontrada con el formato de nombre-password-tipo_Usuario en datos que contiene la estructura de usuario, es decir, nombre, password y el tipo de usuario para después ser almacenada en la lista de usuarios creada. Como se muestra en la Figura 33.

```
while (!feof(f))
{
    user = (usuario *)malloc((i + 1) * sizeof(usuario));
    //Limpieza del arreglo
    bzero(nombre, TAMMAX - 1);
    char aux;
    aux = '\0';
    for (j = 0; aux != '-'; j++)
    {
        aux = fgetc(f);
        if (aux != '-')
            nombre[j] = aux;
    }
    strcpy(user[i].nombre_usuario, nombre);

    bzero(password, TAMMAX - 1);
    char aux2;
    aux2 = '\0';
    for (j = 0; aux2 != '-'; j++)
    {
        aux2 = fgetc(f);
        if (aux2 != '-')
            password[j] = aux2;
    }
    strcpy(user[i].password, password);
```

Figura 33 Fragmento de la función crearUsuarios().

En la Figura 33 se crea un arreglo dinámico de usuarios, para que cada que encuentre una cadena donde se pueda obtener datos, se incrementa el tamaño del arreglo de usuarios. Después se limpia cada cadena temporal para evitar la impresión de basura con la función bzero().

Para separar las cadenas, se hace uso de ciclos for con condiciones donde solo se obtiene caracteres diferentes al guión que los separa y los saltos de línea. Al tener cada cadena separada, se guardan los datos en las variables temporales y se procede a armar la estructura de usuario para poder ser almacenada en la lista de usuarios.

De la misma forma, la función crearProductos(), crea una lista temporal de productos, para después ser cambiada en la lista original, como se muestra en la Figura 34.

```
listaProductos crearProductos()
{
    char nombre[TAMMAX];
    char proveedor[TAMMAX];
    char numEx[TAMMAX];
    int j, i;

    FILE *f;
    f = fopen("productos.txt", "r");

    listaProductos productos;
    crea_lisProductos(&productos);
    producto *prod;
    while (!feof(f))
    {
        prod = (producto *)malloc((i + 1) * sizeof(producto));
        bzero(nombre, TAMMAX - 1);
        char aux;
        aux = '\0';
        for (j = 0; aux != '-'; j++)
        {
            aux = fgetc(f);
            if (aux != '-')
                nombre[j] = aux;
        }
        strcpy(prod[i].nombre_producto, nombre);
```

Figura 34 Fragmento de la función crearProductos().

Al igual que en la función crearUsuarios(), en la función de la Figura 34 se accede al archivo donde esta almacenado la lista de los productos y se hace uso de un arreglo dinámico de productos para después ser insertado a la lista de productos. Cada que se encuentra un producto, el arreglo incrementa de tamaño y procede a separar las cadenas obtenidas del archivo para poder crear estructuras de usuarios y así insertarlas a la lista de usuarios.

De igual forma, para la función de la Figura 35, se accede al archivo de la lista de carritos, para que, con un arreglo dinámico de carritos, se inserten los nuevos elementos a este arreglo. Al igual que crearUsuarios(), en esta función, se incrementa el tamaño del arreglo y se separa los elementos dentro del archivo para crear estructuras, y de esta forma insertarlas de manera correcta a la lista de carritos.

```
listaCarrito crearCarritos()
{
    char nombre[TAMMAX];
    char proveedor[TAMMAX];
    char numEx[TAMMAX];
    char cliente[TAMMAX];
    char fecha[TAMMAX];
    char ID[TAMMAX];

    int j, i;

    FILE *f;
    f = fopen("carritos.txt", "r");

    listaCarrito carritos;
    crea_lisCarrito(&carritos);
    carrito *car;

    while (!feof(f))
    {
        //Obtencion del nombre de producto
        car = (carrito *)malloc((i + 1) * sizeof(carrito));
        bzero(nombre, TAMMAX - 1);
        char aux;
        aux = '\0';
        for (j = 0; aux != '-'; j++)
        {
            aux = fgetc(f);
            if (aux != '-')
                nombre[j] = aux;
```

Figura 35 fragmento de la función crearCarritos().

El archivo Usuario.c se utilizó para la definición de las funciones para el programa mainUsuario.c, donde la primera función es menu(), esta función imprime el menú que se utilizará a lo largo del programa, como se muestra en la Figura 36.

```
void menu()
{
    printf("\n------------Menu-----------\n\n");
    printf("Selecciona una opcion\n\n");
    printf("1 Iniciar sesión\n");
    printf("2 Ver los productos\n");
    printf("3 Comprar un producto\n");
    printf("4 Agregar un producto\n");
    printf("5 Modificar un producto\n");
    printf("6 Mostrar historial de compras \n");
    printf("7 Finalizar programa\n");
    printf("\n---------------------------\n\n");
}
```

Figura 36 Función menu() del archivo Usuario.c.

Las opciones de la Figura 36 indican que dependiendo del tipo de usuario podremos seleccionar opciones como ver, comprar, agregar, modificar o mostrar historial de compras. Y para todo usuario podremos seleccionar las opciones iniciar sesión y ver productos.

La función ini_sesion(), que se muestra en la Figura 37, es el resultado de seleccionar la opción de iniciar sesión en el programa de mainUsuario.c. En esta función se obtienen los datos que escriba el usuario a través del teclado, para ello se hace uso de arreglos de tipo char y una variable int para determinar el tipo de usuario. Una vez adquiridos los datos entrantes por teclado, se procede a copiar los datos a los datos que conforman a la estructura de tipo usuario. Por último, la función retorna la estructura de tipo usuario ya con los datos ingresados por el teclado.

```c
usuario ini_sesion()
{
    usuario user;
    char nombre_usuario[TAMMAX];
    char password[TAMMAX];
    int tipo_usuario=0;;
    strcpy(nombre_usuario,"");
    strcpy(password,"");
    printf("\nEscribe el nombre de usuario: ");
    scanf("%s", nombre_usuario);
    strcpy(user.nombre_usuario,nombre_usuario);

    printf("\nEscribe el password: ");
    scanf("%s", password);
    strcpy(user.password,password);

    printf("\nEscribe el tipo de usuario: ");
    scanf("%d", &tipo_usuario);
    user.tipo_usuario = tipo_usuario;
    return user;
}
```

Figura 37 Función ini_sesion() del archivo Usuario.c

Cada usuario esperará la respuesta que da el programa Control, como se muestra en la Figura 38, cada que Control mande la respuesta de FIN significa que las acciones se realizaron correctamente y de esta forma se desbloquea el semáforo para que los demás usuarios puedan realizar las distintas funciones del programa.

En el caso que se regrese un ERROR, mostrará al usuario que existió un error al realizar algún tipo de acción, y si Control regresa INICIADO, activa la sesión del usuario en cuestión, se crea el semáforo y se le da acceso al mismo para poder hacer uso de las distintas funciones que cuenta el programa.

```c
    /*recibe las respuestas del programa control*/
    msgrcv(cola_ctrluser, &msg, LONGITUD, pid, 0);

    switch (msg.orden)
    {
    case FIN:
    /*Se desbloquea el semaforo*/
        up(semaforo);
        break;

    case ERROR:
        printf("\nQue hubo un error dice\n");
        break;

    case INICIADO:
        /*Si se inicio correctamente, se activa la sesion y se obtiene el
        semaforo del programa Control*/
        userActual.sesion_activa = 1;
        printf("\nInicio correctamente\n");
        printf("\nsemaforo %c\n", msg.CLAVE_SEM);
        /*Se actualiza la llave del semaforo*/
        llave_sem = ftok(FICHERO_SEMAFORO, msg.CLAVE_SEM);
        /*se crea el semaforo*/
        semaforo = Crea_semaforo(llave_sem, 1);
        break;
    default:
        printf("no se reconoce\n");
        break;
    }
} while (!msg.orden);
```

Figura 38 Manejo de las respuestas recibidas por el programa Control.

Para que cada proceso (usuario) esté sincronizado con los otros, se utilizó la herramienta IPC semáforos, como se muestra en la Figura 39, donde la función crea_semaforo crea el semáforo con los argumentos correspondientes y dando valores iniciales, y las funciones up y down, desbloquean y bloquean respectivamente, para la sincronización de los procesos.

```c
int Crea_semaforo(key_t llave, int valor_inicial)
{
    int semid = semget(llave, 1, IPC_CREAT | PERMISOS);
    if (semid == -1)
    {
        return -1;
    }
    semctl(semid, 0, SETVAL, valor_inicial);
    return semid;
}
//Funcion que decrementa el valor del semaforo. Como
argumento recibe el identificador del semaforo en cuestion.
void down(int semid)
{
    struct sembuf op_p[] = {0, -1, 0};
    semop(semid, op_p, 1);
}
//Funcion que incrementa el valor del semaforo. Como
argumento recibe el identificador del semaforo en cuestion.
void up(int semid)
{
    struct sembuf op_v[] = {0, +1, 0};
    semop(semid, op_v, 1);
}
```

Figura 39 Creación y manejo de los semáforos.

Ventanas de Ejecución

Se muestran ejemplos de salidas cuando es cliente y control, después proveedor y control.

Salida de la terminal para Control

```
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$ gcc
Control.c listaCarrito.c listaProductos.c listaUsuarios.c mainControl.c
nodoCarrito.c nodoProductos.c nodoUsuarios.c -o control
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$ ./control


Lista de usuarios
Liss-LLH01-0
Dani-MDPP5-0
Beto-APC13-1
JoseA-JAJ14-1
Idalia-MYL11-0
Ciel-CIE10-1
1
El usuario: 18492 solicito iniciar sesion

El usuario fue encontrado
```

```
1
El usuario: 18458 solicito iniciar sesion

El usuario fue encontrado
2
 Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-25
Galletas-Beto-5
El usuario: 18492 solicito ver productos
3
El usuario: 18492 solicito comprar productos

 Producto-Proveedor-0-Cliente-27/05/2021 00:00-00000
Chocolate-Beto-5-Dani-29/05/2021 18:38-00001
8
Ha ocurrido un error :c
8
Ha ocurrido un error :c
6
 Producto-Proveedor-0-Cliente-27/05/2021 00:00-00000
Chocolate-Beto-5-Dani-29/05/2021 18:38-00001

El usuario: 18492 solicito ver carrito de compras
2
 Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-5
El usuario: 18458 solicito ver productos
8
Ha ocurrido un error :c
4
El usuario: 18458 solicito agregar productos
2
 Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-5
Paleta-Beto-30
Caramelo-Beto-15
El usuario: 18458 solicito ver productos
5
El usuario: 18458 solicito modificar productos
2
```

```
 Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-15
Paleta-Beto-30
Caramelo-Beto-15
El usuario: 18458 solicito ver productos
2
 Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-15
Paleta-Beto-30
Caramelo-Beto-15
El usuario: 18458 solicito ver productos
7
La cola de mensajes ha finalizado
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$
```

Salida de la terminal para Cliente

```
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$ gcc
Usuario.c listaCarrito.c listaProductos.c listaUsuarios.c mainUsuario.c
nodoCarrito.c nodoProductos.c nodoUsuarios.c -o usuario
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$ ./usuario

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
1

Escribe el nombre de usuario: Dani

Escribe el password: MDPP5

Escribe el tipo de usuario: 0
```

```
nombre:Dani
pass:MDPP5
tipo:0

Inicio correctamente

semaforo S

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
2

Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-25
Galletas-Beto-5

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
3

Escribe el nombre del producto: Chocolate

Escribe el nombre del proveedor: Beto
```

```
Escribe la cantidad deseada: 5

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
4

No has iniciado sesion o no tienes los permisos para realizar la accion

Que hubo un error dice

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
5

No has iniciado sesion o no tienes los permisos para realizar la accion

Que hubo un error dice

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa
```

```
----------------------------


Escribe la opcion que desees:
6

Producto-Proveedor-0-Cliente-27/05/2021 00:00-00000
Chocolate-Beto-5-Dani-29/05/2021 18:38-00001


------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
```

Salida de la terminal para Proveedor

```
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$ gcc
Usuario.c listaCarrito.c listaProductos.c listaUsuarios.c mainUsuario.c
nodoCarrito.c nodoProductos.c nodoUsuarios.c -o usuario^C
dani@dani-VirtualBox:~/Documentos/Proyecto/version1/MiTiendita$ ./usuario

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
1

Escribe el nombre de usuario: Beto
```

```
Escribe el password: APC13

Escribe el tipo de usuario: 1
nombre:Beto
pass:APC13
tipo:1

Inicio correctamente

semaforo S

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
2

Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-5

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
3

No has iniciado sesion o no eres Cliente
```

```
Que hubo un error dice

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
4

Escribe el nombre del producto: Paleta

Escribe la cantidad: 30

¿Quiere agregar otro producto? s/n: s

Escribe el nombre del producto: Caramelo

Escribe la cantidad: 15

¿Quiere agregar otro producto? s/n: n

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
2

Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
```

```
Chocolate-Beto-20
Galletas-Beto-5
Paleta-Beto-30
Caramelo-Beto-15

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
5

Escribe el nombre del producto: Galletas

Escribe la cantidad a incrementar: 10

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------


Escribe la opcion que desees:
2

Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-15
Paleta-Beto-30
Caramelo-Beto-15

------------Menu-----------
```

```
Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------

Escribe la opcion que desees:
2

Donas-Bimbo-3
Leche-Lala-0
Agua1L-Ciel-19
COCA-Beto-12
taco-Beto-15
Tortas-Beto-15
Chocolate-Beto-20
Galletas-Beto-15
Paleta-Beto-30
Caramelo-Beto-15

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------

Escribe la opcion que desees:
7

------------Menu-----------

Selecciona una opcion

1 Iniciar sesión
2 Ver los productos
3 Comprar un producto
4 Agregar un producto
5 Modificar un producto
6 Mostrar historial de compras
7 Finalizar programa

----------------------------
```

XI Conclusiones

López Hernández Lissete

Los sistemas operativos y las GUI permiten que los usuarios puedan ocupar una computadora para realizar diversas tareas sin un conocimiento profundo de ella.

Desde el enfoque de manejo de archivos, gracias a los sistemas de archivos podemos realizar operaciones con sus elementos fácilmente.

Por el lado de los procesos, la cantidad de ellos que se ejecutan cuando se abre una aplicación e interactúan con ella no suele ser contemplada por usuarios estándar.

Mientras que, desde la perspectiva de desarrollador, los procesos internos se vuelven más notorios en comparación con un usuario estándar, ya que siguen la idea de “divide y vencerás”, entonces es de esperar que para la realización de una tarea específica la computadora deba realizar varios procesos.

La ya mencionada idea de “divide y vencerás” enseñada en los primeros cursos de programación se encuentra implícita en hilos y procesos, incluso en los semáforos.

Perspectiva que puede ser vista en los procesos, ya que cada uno se divide en subprocesos o hilos que en conjunto realizan la labor encomendada al proceso.

Mientras que en los semáforos se puede apreciar un seccionamiento de los procesos que se pueden ejecutar en un determinado instante y los que no, el cual cambia cada que un proceso finalice su ejecución.

Ya mencionados los conceptos necesarios de algunas herramientas IPC y teniendo una idea clara de lo que se puede realizar con dichas herramientas, podemos empezar a describir el análisis y diseño de un proyecto en el que se puedan aplicar.

En el caso del presente proyecto se pensó implementar semáforos y memoria compartida, donde los archivos serán compartidos por todos los usuarios mediante la ya mencionada memoria compartida, y los semáforos controlarían el acceso a los mismos.

Dicha implementación de herramientas IPC fue el mayor reto que tuvimos en la codificación del proyecto, ya que dichas herramientas tienen el rol de unir el código enfocado al usuario con el enfocado al sistema.

Una vez lograda esta parte, los problemas presentados fueron mayoritariamente de manejo de archivos, ya que las estructuras de datos deben están sincronizadas con los archivos, debido a que en ellos se guarda el registro de catálogo de productos y usuarios, lo que significa que si alguno de los dos tiene diferentes valores generaría inconsistencia en los datos.

Al agregar las últimas funciones de los carritos de compras se implementaron funciones similares a las de listas productos y de usuarios, por ejemplo, las usadas para obtener los datos de un archivo y convertirlo a lista. A diferencia de la lista de productos y usuarios, la lista de carritos agrega datos de ambas, como el nombre del producto y del usuario, cantidad, id de compra y la fecha.

Para obtener el momento en el que el usuario hizo una compra, se utilizó un archivo de cabecera que desconocía, time.h, en el cual tiene funciones que nos permite el manejo del tiempo en C, en el caso de este proyecto, guardar la fecha y la hora en el que se hizo el registro de la compra.

Al terminar las funciones, nos enfocamos en que el proyecto fuera multiusuario, con las colas de mensajes nos permitía conectar los usuarios y el programa Control, pero faltaba la sincronización entre usuarios, para ello, utilizamos los semáforos. La implementación de estos fue relativamente sencilla, lo que estuvo en duda por un momento fue su colocación, al final se determinó que el lugar donde los semáforos mejoraran la sincronización y evitar algún tipo de inconsistencia es colocarlo cuando se ejecuta la función (bloquear semáforo) y desbloquearlo al recibir el mensaje por parte del programa control, de esta forma, otro usuario no puede hacer solicitudes hasta que la comunicación entre un usuario y el control finalice.

Palacios Cabrera Alberto

El primer paso en la elaboración del proyecto final fue un acercamiento a lo que es la administración de procesos y archivos.

Como podemos observar en la información recopilada, el sistema operativo posee múltiples herramientas que permiten manejar los procesos que está ejecutando, de manera que se realicen de la forma más rápida y sin presentar errores durante su ejecución.

El sistema operativo debe de realizar múltiples procesos e hilos desde el momento del arranque y los continuará generando, tanto con las acciones del usuario como por su propio funcionamiento de la computadora. Manejar los procesos e hilos es de fundamental importancia, por ello se implementan los semáforos, colas de mensajes y memoria compartida, que permiten agilizar y controlar el flujo de los mismos.

El uso de archivos es de gran importancia tanto para el usuario final como para los desarrolladores, ya que son los que nos permiten almacenar y preservar la información que necesitemos en nuestro equipo, por lo cual es necesario conocer cómo se trabaja con ellos. Como bien sabemos, la manipulación de archivos se puede realizar desde una GUI o desde la terminal.

Para permitir la comunicación entre el equipo y el usuario, es necesario tener dispositivos con los cuales ingresar información y también con los que recibamos una retroalimentación sobre las operaciones que hemos realizado. Conocer los principios de estos dispositivos será de utilidad ya que son el medio con el cual podemos ingresar y obtener datos de nuestra computadora.

En la siguiente etapa se consolidaron los objetivos y se justificó el proyecto que vamos a realizar. Esto fue posible gracias a la construcción de un protocolo que describiera el proyecto de manera más precisa.

Posteriormente, comenzamos con el análisis y el desarrollo del sistema, requerimos de varias reuniones para lograr definir la mejor arquitectura de nuestro proyecto, tratando de implementar de manera eficiente las herramientas que previamente investigamos. El modelado fue un poco complejo de realizar, ya que debíamos de cumplir con los requerimientos definidos previamente sin complicar demasiado el sistema propuesto, sin embargo, el resultado obtenido fue satisfactorio para el equipo.

La fase de desarrollo continúo con la codificación del sistema. El uso de códigos modulares resultó de gran ayuda en este proyecto, ya que nos permite crear y modificar funciones especificas sin tener que alterar otros aspectos que no requieran de una alteración. Así mismo, nos permite el reciclado de código y una mejor organización dentro del proyecto.

En cuanto a la arquitectura del sistema, el manejo de estructuras de datos y las herramientas del IPC fueron fundamentales, ya que para la comunicación entre los programas se requirió de la

comunicación por colas de mensajes y el uso de diversas estructuras de datos para poder manejar la información necesaria. El uso de archivos nos permitió facilitar una gran cantidad de operaciones ya que podíamos modificar y recuperar datos directo de los mismos.

El sistema ha mostrado consistencia en los datos, permitiéndonos cumplir uno de los principales objetivos que hemos planteado en el proyecto.

Pérez Priego Mario Daniel

El inicio de la práctica da un acercamiento a los conceptos que necesitaremos implementar para el proyecto final, los cuales consisten en procesos, hilos, semáforos, archivos, colas de mensajes, memoria compartida y dispositivos de entrada.

Por un lado, los conceptos de procesos, hilos, semáforos y memoria compartida permiten que en un sistema operativo realice diferentes tareas casi simultáneas.

A partir de estas investigaciones previas, se puede notar que no hay procesos o tareas que la computadora pueda hacer de manera simultánea, sino que realizan procesos uno por uno en una velocidad muy rápida que no es perceptible, lo que aparenta que se realizan al mismo tiempo. Esto es gracias al compartimiento de una sección de memoria, ya que permite que los diferentes procesos puedan tomar información de uno y otro sin tener que hacer una copia o un cambio de dirección para obtenerla.

Por otro lado, los conceptos de archivos, colas de mensajes, dispositivos de entrada y salida van más enfocados en como el usuario puede interactuar con la computadora, y como esta debe realizar una serie de acciones para el correcto almacenamiento y la eficiente comunicación de la información obtenida o que puede regresar de alguna manera hacia el usuario. Los cuales son ingresados u obtenidos de los usuarios a partir de los dispositivos de entrada y salida, por ejemplo, los archivos. Para que esto se realice correctamente, se hace uso de colas de mensajes, como se detalló en la sección correspondiente, permite una eficaz comunicación del sistema operativo, lo que reduce los tiempos y errores para los diferentes procesos que se efectúan dentro de nuestra computadora.

En el caso del presente proyecto, el objetivo es que la plataforma sea de un uso sencillo para el usuario, por lo tanto, solo se mostrarán los datos necesarios, tales como un historial de archivos sin detalles extra como compañía de mensajería, día de llegada, etc., únicamente se mantendrán los detalles como nombre de artículos, cantidad, comprador, proveedor y fecha. Para ello se usarán listas enlazadas, por la facilidad de operaciones que estas permiten. Dichas listas, serán compartidas mediante el mecanismo de memoria compartida, y el control de acceso será indicado mediante semáforos.

En lo referente al proceso de codificación de colas mensajes, y memoria compartida aprendí que el uso de apuntadores como parámetros en dichas herramientas no es permitido, lo cual significó un gran inconveniente. Para solucionarlo se tuvo que recurrir a mandar las estructuras necesarias sin apuntadores y a usar cadenas de caracteres como reemplazo de los apuntadores en ciertas ocasiones.

Se terminó con la función para crear la lista de carritos, a pesar de que era similar con algunas funciones, hubo complicaciones a la hora de codificar, ya que implicaba mayor numero de datos a registrar, y el acceso al archivo generaba errores por datos vacíos o saltos de línea, para resolver esto, se verificó las entradas, las estructuras y la generación de las cadenas a insertar en el archivo.

Para finalizar la codificación del proyecto, se implementó el uso de los semáforos, esto permite la sincronización entre los clientes y los proveedores para que no haya algún tipo de inconsistencia en

los datos. Para ello, se posición la función de down() al iniciar cada función, al realizar las instrucciones correspondientes de cada una, los demás usuarios están en espera, tanto por la cola de mensajes, como los semáforos, y al finalizar la acción del usuario, es desbloqueado el semáforo con la función up() para el siguiente usuario.

Utilizando estas dos herramientas IPC garantizamos la correcta sincronización entre usuarios y el programa Control, ya que la cola de mensajes nos permitió la comunicación entre usuario y control, y los semáforos la organización entre los usuarios para evitar errores con algún dato.

XII Referencias

[1] “Sistema de Ventas y Gestión Empresarial - 【Asesoría Personalizada】”, *Sistema de Ventas*. https://sistema-ventas.com.mx/ (consultado abr. 17, 2021).

[2] “1.2. ¿Qué es GNU/Linux?” https://www.debian.org/releases/jessie/mips/ch01s02.html.es (consultado abr. 17, 2021).

[3] “¿Qué es Linux?” https://www.redhat.com/es/topics/linux (consultado abr. 17, 2021).

[4] A. Silberschatz, P. Baer Galvin, y G. Gagne, *Operating system concepts*, 9a ed. Estados Unidos: Wiley, 2013.

[5] J. C. Pérez, F. García Caballeira, P. De Miguel Anasagasti, y F. Pérez Costoya, *Sistemas operativos. Una visión Aplicada.* Madrid: McGRAW-HILL, 2001.

[6] A. S. Tanenbaum, *Sistemas operativos modernos*. Madrid, España: Pearson Educación, 2003.

[7] F. J. Ceballos, *Java 2. Curso de programación.*, 4a ed. México: Alfaomega Ra-Ma, 2011.

[8] “Inter Process Communication (IPC)”, *GeeksforGeeks*, ene. 24, 2017. https://www.geeksforgeeks.org/inter-process-communication-ipc/ (consultado mar. 20, 2021).

[9] F. M. Márquez García, *UNIX Programación avanzada*, 3a ed. Madrid, España.: Ra-Ma, 2014.

XIII Anexos

Códigos

err.h

```
#ifndef ERR_H
#define ERR_H
/*********************************************************
Definir errores comunes a programas
*********************************************************/
#include <stdlib.h> /* Para definir NULL */
#include <stdio.h>
#include <string.h>
/*****************C O N S T A N T E S************************/
#define OK 0        /* No hay errores */
#define AP_INV 1    /* Recibe un apuntador nulo */
#define NO_MEM 2    /* malloc regresa NULL*/
#define OVERFLOW 30  /* Si no hay memoria*/
#define UNDERFLOW 31 /* En casos de lista o cola vacia*/
#define POS_INV 32   /* Posicion invalida */
#define TAMDAT 10
#define NO_EXISTE -2 /* El dato buscado no existe*/
#define CANT_INV -3 //*Cantidad invalida*/

/****************M A C R O S**********************/
/* Verifica que un apuntador no tenga direccion nula*/
#define AP_VAL(p) ((p) != NULL)
#endif
```

listaProductos.h

```
#ifndef LISTAPRODUCTOS_H
#define LISTAPRODUCTOS_H
/*********************************************************
Implementa una lista utilizando nodos de una liga.
*********************************************************/
#include "nodoProductos.h"
/*****************M A C R O S*******************************/
#define TIE_DAT(p)  ((p)!=NULL)   /*Si tiene datos*/
#define VAC_LIS(p)  ((p)==NULL)   /* Si es un lista vacia*/
/***************T I P O     D E     D A T O***************/
typedef apnodoProductos listaProductos;
/************** F U N C I O N E S*************************/
int crea_lisProductos(listaProductos* l); /*Inicializa la lista Productos*/
int insf_lisProductos(listaProductos*,DATO_Productos); /*Inserta el ultimo nodo*/
int dest_lisProductos(listaProductos*);      /*Elimina por completo una lista*/
int inse_lisProductos(listaProductos*,int,DATO_Productos);/*inserta nuevos nodos en la lista
Productos*/
```

```
listaProductos ante_lisProductos(listaProductos, int); /*Determina el nodo anterior a la posicion p de la lista Productos*/
int long_lisProductos(listaProductos*);     /*Calcula la longitud de la lista*/
int buscar_lisProductos(listaProductos *l, char* nombre_producto,char* nombre_proveedor);/*Busca la pagina en la lista*/
int elim_lisProductos(listaProductos*,int);  /*Saca un elemento del lista*/
int elip_lisProductos(listaProductos*);     /*Elimina el primer dato*/
char *str_lisProductos(char *s, listaProductos *l);
int dis_existencias(listaProductos* l, char* nombre_producto,char* nombre_proveedor, int art_comprados);
int actualizar_Productos(char *s); /*Actualiza el archivo de productos*/
#endif
```

listaProductos.c

```
#include "listaProductos.h"

/*La funcion crea_lisProductos se encarga de inicializar la lista Productos.
Recibe:
listaProductos *l: apuntador a la lista que almacenara los Productos registrados.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error */
int crea_lisProductos(listaProductos *l)
{
  if (!AP_VAL(l))
    return AP_INV;
  *l = NULL;
  return OK;
}
/*La funcion dest_lisProductos destruye la lista Productos mediante la liberacion de memoria de los nodos que la forman.
Recibe:
listaProductos * l: lista a destruira.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error
*/
int dest_lisProductos(listaProductos *l)
{
  if (!AP_VAL(l))
    return AP_INV;
  while (!VAC_LIS(*l))
    elip_lisProductos(l);
  return OK;
}
/*La funcion ante_lisProductos determina el nodo anterior a la posicion p de la lista Productos.
Recibe:
listaProductos: lista cuyo nodo anterior sera obtenido
int p: posicion de la cual se obtendra su nodo anterior.
Retorna: nodo anterior*/
```

```
listaProductos ante_lisProductos(listaProductos l, int p)
{
 listaProductos aux, ant;
 int pos;
 aux = l;
 ant = aux;
 pos = 0;
 while (aux && pos < p)
 {
  ant = aux;
  aux = aux->sig;
  pos++;
 }
 return ant;
}
/*La funcion long_lisProductos determina la longitud de la lista Productos.
Recibe:
listaProductos *l: lista cuya longitud se determinara
Retorna: longitud de la lista*/
int long_lisProductos(listaProductos *l)
{
 int cont;
 listaProductos aux;
 if (!AP_VAL(l))
  return AP_INV;
 cont = 0;
 aux = *l;
 while (aux)
 {
  cont++;
  aux = aux->sig;
 }
 return cont;
}

/*La funcion inse_lisProductos inserta nuevos nodos en la lista Productos.
Recibe:
listaProductos *l: lista a la que se le agregara el nodo
int pos: la posicion en la que se agregará el nodo
DATOProductos D: dato que se almacenara en dicho nodo.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error*/
int inse_lisProductos(listaProductos *l, int pos, DATO_Productos d)
{
 apnodoProductos nuevo;
 listaProductos aux;
 if (!AP_VAL(l))
  return AP_INV;
 if (!(pos <= long_lisProductos(l) && pos >= 0))
```

```
   return POS_INV;
  if (!(crea_nodProductos(&nuevo, d) == OK))
   return OVERFLOW;
  aux = ante_lisProductos(*l, pos);
  nuevo->sig = aux->sig;
  aux->sig = nuevo;
  return OK;
}

/*La funcion ulti_lisProductos regresa el ultimo nodo de la lista.
Recibe:
listaProductos l: lista cuyo ultimo nodo se buscara
Retorna:
ultimo nodo de la lista*/
listaProductos ulti_lisProductos(const listaProductos l)
{
  listaProductos aux, ant;
  aux = l;
  ant = aux;
  while (aux)
  {
   ant = aux;
   aux = aux->sig;
  }
  return ant;
}

/*La funcion insf_lisProductos inserta un nodo al final de la lista Productos.
Recibe:
listaProductos *l: lista en la que se realizara la operacion de insercion
DATOProductos: dato a insertar
Retorna:
 OK en en caso de exito
Diferente de OK en caso de error*/
int insf_lisProductos(listaProductos *l, DATO_Productos d)
{
  apnodoProductos nuevo, aux;
  if (!AP_VAL(l))
   return AP_INV;
  aux = *l;
  if (!(crea_nodProductos(&nuevo, d) == OK))
   return OVERFLOW;
  if (VAC_LIS(aux)) /*Si la lista esta vacia*/
   (*l) = nuevo;
  else
  {
   aux = ulti_lisProductos(*l);
   aux->sig = nuevo;
  }
  return OK;
```

```
}

/* La funcion elim_lisProductos saca el elemento de la posicion pos de la lista Productos.
Recibe:
listaProductos *l: lista en la que se realizara la operacion de eliminacion
Int pos: posicion que determina el nodo a eliminar.
Retorna:
 OK en en caso de exito
Diferente de OK en caso de error*/
int elim_lisProductos(listaProductos *l, int pos)
{
 listaProductos aux, borrar;
 if (!AP_VAL(l))
   return AP_INV;
 if (!TIE_DAT(*l))
   return UNDERFLOW;
 if (!(pos <= long_lisProductos(l) && pos > 0))
   return POS_INV;

 aux = ante_lisProductos(*l, pos);
 if (aux->sig)
 {
   borrar = aux->sig;
   aux->sig = borrar->sig;
   free(borrar);
 }
 return OK;
}

/*La funcion elip_lisProductos se encarga de borrar el primer nodo de la lista Productos.
Recibe:
listaProductos *l cuyo primer nodo sera borrado
Retorna:
OK en en caso de exito
Diferente de OK en caso de error*/
int elip_lisProductos(listaProductos *l)
{
 apnodoProductos borrar;
 if (!AP_VAL(l))
   return AP_INV;
 if (!TIE_DAT(*l))
   return UNDERFLOW;
 borrar = *l;
 *l = (*l)->sig;
 free(borrar);
 return OK;
}
/*La funcion str_lisProductos convierte a cadena la lista Productos, para que esta pueda ser visualizada
Recibe:
```

```
listaProductos *l: apuntador a la lista Productos cuya cadena sera formada
char *s: apuntador a caracter que almacenara la cadena
Devuelve:
char *str_lisProductos: cadena resultante
*/

char *str_lisProductos(char *s, listaProductos *l)
{
  char aux[TAMMAX + 1];
  char aux1[TAMMAX + 1];
  char aux2[TAMMAX + 1];
  listaProductos l1, l2;
  l2 = NULL;
  l1 = *l;
  while (l1)
  {
    sprintf(aux, "%s", l1->dato->nombre_producto);
    strcat(s, aux);
    sprintf(aux1, "-%s", l1->dato->nombre_proveedor);
    strcat(s, aux1);
    sprintf(aux2, "-%d", l1->dato->num_existencia);
    strcat(s, aux2);
    if(l1->sig != NULL)
      strcat(s, "\n");
    l2 = l1;
    l1 = l1->sig;
  }
  return s;
}

/* La funcion buscar_lisProductos se encarga de buscar un dato en una lista.
Recibe:
listaProductos *l: lista en la que debe buscar
char *nombre_producto: nombre de producto que se busca.
char *nombre_proveedor: nombre de proveedor que se busca.
Regresa:posicion del dato buscado.*/
int buscar_lisProductos(listaProductos *l, char *nombre_producto, char* nombre_proveedor)
{
  if (!AP_VAL(l))
    return AP_INV;
  if (!TIE_DAT(*l))
    return UNDERFLOW;

  listaProductos l1, l2;
  //l2 es una lista auxiliar
  int i = 0;
  l2 = NULL;
  l1 = *l;
  while (l1)
  {
```

```
   if                (strcmp(nombre_producto,l1->dato->nombre_producto)==0                &&
strcmp(nombre_proveedor,l1->dato->nombre_proveedor)==0)
     return i;
   else
   {
    l2 = l1;
    l1 = l1->sig;
    i++;
   }
  }
  return NO_EXISTE;
}
/*dis_existencias: disminuye el numero de existencias de un articulo en una unidad.
Para identificar el articulo cuya existencia será disminuida se utiliza el nombre del producto y
el proveedor.
Recibe:
listaProductos: lista en la que se encuentra el producto cuya existencia sera modificada
char* nombre_producto: nombre del producto que se busca
char* nombre_proveedor: nombre del proveedor que se busca
*/
int dis_existencias(listaProductos *l, char *nombre_producto, char *nombre_proveedor, int
art_comprados)
{
  if (!AP_VAL(l))
    return AP_INV;
  if (!TIE_DAT(*l))
    return UNDERFLOW;

  listaProductos l1, l2;
  //l2 es una lista auxiliar
  int i = 0;
  l2 = NULL;
  l1 = *l;
  while (l1)
  {
    //En caso de que el articulo buscado sea encontrado
    if       (strcmp(nombre_producto,       l1->dato->nombre_producto)       ==       0       &&
strcmp(nombre_proveedor, l1->dato->nombre_proveedor) == 0)
    {
     if (art_comprados > l1->dato->num_existencia)
     {
      printf("\nCantidad invalida :'c\n");
      return CANT_INV;
     }
     else
     {
      //El numero de existencias del articulo disminuira en funcion de la cantidad de elementos
comprados

      l1->dato->num_existencia = l1->dato->num_existencia - art_comprados;
```

```
      return i;
    }
   }
   else
   {
    l2 = l1;
    l1 = l1->sig;
    i++;
   }
  }
 return NO_EXISTE;
}
```

listaUsuarios.h

```
#ifndef LISTAUSUARIOS_H
#define LISTAUSUARIOS_H
/*************************************************************
Implementa una lista utilizando nodos de una liga.
*************************************************************/
#include "nodoUsuarios.h"
/*****************M A C R O S*****************************/
#define TIE_DAT(p)  ((p)!=NULL)   /*Si tiene datos*/
#define VAC_LIS(p)  ((p)==NULL)   /* Si es un lista vacia*/
/***************T I P O     D E     D A T O***************/
typedef apnodoUsuarios listaUsuarios;
/************** F U N C I O N E S*************************/
int crea_lisUsuarios(listaUsuarios* l); /*Inicializa la lista Usuarios*/
int insf_lisUsuarios(listaUsuarios*,DATO_Usuarios); /*Inserta el ultimo nodo*/
int dest_lisUsuarios(listaUsuarios*);      /*Elimina por completo una lista*/
int inse_lisUsuarios(listaUsuarios*,int,DATO_Usuarios);/*inserta nuevos nodos en la lista
Usuarios*/
listaUsuarios ante_lisUsuarios(listaUsuarios, int); /*Determina el nodo anterior a la posicion p
de la lista Usuarios*/
int long_lisUsuarios(listaUsuarios*);      /*Calcula la longitud de la lista*/
int buscar_lisUsuarios(listaUsuarios *l, char* datos_buscado);/*Busca la pagina en la lista*/
int elim_lisUsuarios(listaUsuarios*,int);  /*Saca un elemento del lista*/
int elip_lisUsuarios(listaUsuarios*);      /*Elimina el primer dato*/
char *str_lisUsuarios(char *s, listaUsuarios *l);

int validar_lisUsuarios(listaUsuarios *l, char* usuario, char* password, int tipo);
#endif
```

listaUsuarios.c

```
#include "listaUsuarios.h"

/*La funcion crea_lisUsuarios se encarga de inicializar la lista Usuarios.
Recibe:
listaUsuarios *l: apuntador a la lista que almacenara los usuarios registrados.
Retorna:
OK en en caso de exito
```

```
Diferente de OK en caso de error */
int crea_lisUsuarios(listaUsuarios *l)
{
 if (!AP_VAL(l))
   return AP_INV;
 *l = NULL;
 return OK;
}
/*La funcion dest_lisUsuarios destruye la lista Usuarios mediante la liberacion de memoria de
los nodos que la forman.
Recibe:
listaUsuarios *l: lista a destruir.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error
*/
int dest_lisUsuarios(listaUsuarios *l)
{
 if (!AP_VAL(l))
   return AP_INV;
 while (!VAC_LIS(*l))
   elip_lisUsuarios(l);
 return OK;
}
/*La funcion ante_lisUsuarios determina el nodo anterior a la posicion p de la lista Usuarios.
Recibe:
listaUsuarios: lista cuyo nodo anterior sera obtenido
int p: posicion de la cual se obtendra su nodo anterior.
Retorna: nodo anterior*/
listaUsuarios ante_lisUsuarios(listaUsuarios l, int p)
{
 listaUsuarios aux, ant;
 int pos;
 aux = l;
 ant = aux;
 pos = 0;
 while (aux && pos < p)
 {
   ant = aux;
   aux = aux->sig;
   pos++;
 }
 return ant;
}
/*La funcion long_lisUsuarios determina la longitud de la lista Usuarios.
Recibe:
listaUsuarios *l: lista cuya longitud se determinara
Retorna: longitud de la lista*/
int long_lisUsuarios(listaUsuarios *l)
{
```

```
 int cont;
 listaUsuarios aux;
 if (!AP_VAL(l))
  return AP_INV;
 cont = 0;
 aux = *l;
 while (aux)
 {
  cont++;
  aux = aux->sig;
 }
 return cont;
}

/*La funcion inse_lisUsuarios inserta nuevos nodos en la lista Usuarios.
Recibe:
listaUsuarios *l: lista a la que se le agregara el nodo
int pos: la posicion en la que se agregará el nodo
DATOUsuarios D: dato que se almacenara en dicho nodo.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error*/
int inse_lisUsuarios(listaUsuarios *l, int pos, DATO_Usuarios d)
{
 apnodoUsuarios nuevo;
 listaUsuarios aux;
 if (!AP_VAL(l))
  return AP_INV;
 if (!(pos <= long_lisUsuarios(l) && pos >= 0))
  return POS_INV;
 if (!(crea_nodUsuarios(&nuevo, d) == OK))
  return OVERFLOW;
 aux = ante_lisUsuarios(*l, pos);
 nuevo->sig = aux->sig;
 aux->sig = nuevo;
 return OK;
}

/*La funcion ulti_lisUsuarios regresa el ultimo nodo de la lista.
Recibe:
listaUsuarios l: lista cuyo ultimo nodo se buscara
Retorna:
ultimo nodo de la lista*/
listaUsuarios ulti_lisUsuarios(const listaUsuarios l)
{
 listaUsuarios aux, ant;
 aux = l;
 ant = aux;
 while (aux)
 {
```

```
   ant = aux;
   aux = aux->sig;
  }
  return ant;
}

/*La funcion insf_lisUsuarios inserta un nodo al final de la lista Usuarios.
Recibe:
listaUsuarios *l: lista en la que se realizara la operacion de insercion
DATOUsuarios: dato a insertar
Retorna:
 OK en en caso de exito
Diferente de OK en caso de error*/
int insf_lisUsuarios(listaUsuarios *l, DATO_Usuarios d)
{
  apnodoUsuarios nuevo, aux;
  if (!AP_VAL(l))
    return AP_INV;
  aux = *l;
  if (!(crea_nodUsuarios(&nuevo, d) == OK))
    return OVERFLOW;
  if (VAC_LIS(aux)) /*Si la lista esta vacia*/
    (*l) = nuevo;
  else
  {
    aux = ulti_lisUsuarios(*l);
    aux->sig = nuevo;
  }
  return OK;
}

/* La funcion elim_lisUsuarios saca el elemento de la posicion pos de la lista Usuarios.
Recibe:
listaUsuarios *l: lista en la que se realizara la operacion de eliminacion
Int pos: posicion que determina el nodo a eliminar.
Retorna:
 OK en en caso de exito
Diferente de OK en caso de error*/
int elim_lisUsuarios(listaUsuarios *l, int pos)
{
  listaUsuarios aux, borrar;
  if (!AP_VAL(l))
    return AP_INV;
  if (!TIE_DAT(*l))
    return UNDERFLOW;
  if (!(pos <= long_lisUsuarios(l) && pos > 0))
    return POS_INV;

  aux = ante_lisUsuarios(*l, pos);
  if (aux->sig)
```

```
 {
  borrar = aux->sig;
  aux->sig = borrar->sig;
  free(borrar);
 }
 return OK;
}

/*La funcion elip_lisUsuarios se encarga de borrar el primer nodo de la lista Usuarios.
Recibe:
listaUsuarios *l cuyo primer nodo sera borrado
Retorna:
OK en en caso de exito
Diferente de OK en caso de error*/
int elip_lisUsuarios(listaUsuarios *l)
{
 apnodoUsuarios borrar;
 if (!AP_VAL(l))
  return AP_INV;
 if (!TIE_DAT(*l))
  return UNDERFLOW;
 borrar = *l;
 *l = (*l)->sig;
 free(borrar);
 return OK;
}
/*La funcion str_lisUsuarios convierte a cadena la lista Usuarios, para que esta pueda ser visualizada
Recibe:
listaUsuarios *l: apuntador a la lista Usuarios cuya cadena sera formada
char *s: apuntador a caracter que almacenara la cadena
Devuelve:
char *str_lisUsuarios: cadena resultante
*/

char *str_lisUsuarios(char *s, listaUsuarios *l)
{
 char aux[TAMMAX+1];
 char aux1[TAMMAX+1];
 char aux2[TAMMAX+1];
 listaUsuarios l1, l2;
 l2 = NULL;
 l1 = *l;
 strcpy(s, "\nLista de usuarios\n");
 while (l1)
 {
  sprintf(aux, "%s", l1->dato->nombre_usuario);
  strcat(s, aux);
  sprintf(aux1, "-%s", l1->dato->password);
  strcat(s, aux1);
```

```
   sprintf(aux2, "-%d", l1->dato->tipo_usuario);
   strcat(s, aux2);
   strcat(s, "\n");
   l2 = l1;
   l1 = l1->sig;
  }
  return s;
}
/* La funcion buscar_lisUsuarios se encarga de buscar un dato en una lista de Usuarios.
Recibe:
listaUsuarios *l: lista en la que debe buscar
int datos_buscado:dato que se busca.
Regresa:posicion del dato buscado.*/
int buscar_lisUsuarios(listaUsuarios *l, char* datos_buscado)
{
  if (!AP_VAL(l))
    return AP_INV;
  if (!TIE_DAT(*l))
    return UNDERFLOW;

  listaUsuarios l1, l2;
  //l2 es una lista auxiliar
  int i = 0;
  l2 = NULL;
  l1 = *l;
  while (l1)
  {
    if (datos_buscado == l1->dato->nombre_usuario)
      return i;
    else
    {
      l2 = l1;
      l1 = l1->sig;
      i++;
    }
  }
  return NO_EXISTE;
}

/* La funcion buscar_lisUsuarios se encarga de buscar un dato en una lista de Usuarios.
Recibe:
listaUsuarios *l: lista en la que debe buscar
int datos_buscado:dato que se busca.
Regresa:posicion del dato buscado.*/
int validar_lisUsuarios(listaUsuarios *l, char* usuario, char* password, int tipo)
{
  if (!AP_VAL(l))
    return AP_INV;
  if (!TIE_DAT(*l))
    return UNDERFLOW;
```

```
 listaUsuarios l1, l2;
 //l2 es una lista auxiliar
 int i = 0;
 l2 = NULL;
 l1 = *l;
 while (l1)
 {

  if (strcmp(usuario,l1->dato->nombre_usuario)==0 && strcmp(password,l1->dato->password)==0 &&tipo==l1->dato->tipo_usuario){
    l1->dato->sesion_activa=1;
    return i;
  }
  else
  {
    l2 = l1;
    l1 = l1->sig;
    i++;
  }
 }
 return NO_EXISTE;
}
```

listaCarrito.c

```
#include "listaCarrito.h"

/*La funcion crea_lisCarrito se encarga de inicializar la lista Carrito.
Recibe:
listaCarrito *l: apuntador a la lista que almacenara los carritos registrados.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error */
int crea_lisCarrito(listaCarrito *l)
{
 if (!AP_VAL(l))
   return AP_INV;
 *l = NULL;
 return OK;
}
/*La funcion dest_lisCarrito destruye la lista Carrito mediante la liberacion de memoria de los nodos que la forman.
Recibe:
listaCarrito *l: lista a destruir.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error
*/
int dest_lisCarrito(listaCarrito *l)
{
```

```
 if (!AP_VAL(l))
   return AP_INV;
 while (!VAC_LIS(*l))
   elip_lisCarrito(l);
 return OK;
}
/*La funcion ante_lisCarrito determina el nodo anterior a la posicion p de la lista Carrito.
Recibe:
listaCarrito: lista cuyo nodo anterior sera obtenido
int p: posicion de la cual se obtendra su nodo anterior.
Retorna: nodo anterior*/
listaCarrito ante_lisCarrito(listaCarrito l, int p)
{
 listaCarrito aux, ant;
 int pos;
 aux = l;
 ant = aux;
 pos = 0;
 while (aux && pos < p)
 {
   ant = aux;
   aux = aux->sig;
   pos++;
 }
 return ant;
}
/*La funcion long_lisCarrito determina la longitud de la lista Carrito.
Recibe:
listaCarrito *l: lista cuya longitud se determinara
Retorna: longitud de la lista*/
int long_lisCarrito(listaCarrito *l)
{
 int cont;
 listaCarrito aux;
 if (!AP_VAL(l))
   return AP_INV;
 cont = 0;
 aux = *l;
 while (aux)
 {
   cont++;
   aux = aux->sig;
 }
 return cont;
}

/*La funcion inse_lisCarrito inserta nuevos nodos en la lista Carrito.
Recibe:
listaCarrito *l: lista a la que se le agregara el nodo
int pos: la posicion en la que se agregará el nodo
```

```
DATOCarrito D: dato que se almacenara en dicho nodo.
Retorna:
OK en en caso de exito
Diferente de OK en caso de error*/
int inse_lisCarrito(listaCarrito *l, int pos, DATO_Carrito d)
{
 apnodoCarrito nuevo;
 listaCarrito aux;
 if (!AP_VAL(l))
  return AP_INV;
 if (!(pos <= long_lisCarrito(l) && pos >= 0))
  return POS_INV;
 if (!(crea_nodCarrito(&nuevo, d) == OK))
  return OVERFLOW;
 aux = ante_lisCarrito(*l, pos);
 nuevo->sig = aux->sig;
 aux->sig = nuevo;
 return OK;
}

/*La funcion ulti_lisCarrito regresa el ultimo nodo de la lista.
Recibe:
listaCarrito l: lista cuyo ultimo nodo se buscara
Retorna:
ultimo nodo de la lista*/
listaCarrito ulti_lisCarrito(const listaCarrito l)
{
 listaCarrito aux, ant;
 aux = l;
 ant = aux;
 while (aux)
 {
  ant = aux;
  aux = aux->sig;
 }
 return ant;
}

/*La funcion insf_lisCarrito inserta un nodo al final de la lista Carrito.
Recibe:
listaCarrito *l: lista en la que se realizara la operacion de insercion
DATOCarrito: dato a insertar
Retorna:
 OK en en caso de exito
Diferente de OK en caso de error*/
int insf_lisCarrito(listaCarrito *l, DATO_Carrito d)
{
 apnodoCarrito nuevo, aux;
 if (!AP_VAL(l))
  return AP_INV;
```

```
  aux = *l;
  if (!(crea_nodCarrito(&nuevo, d) == OK))
    return OVERFLOW;
  if (VAC_LIS(aux)) /*Si la lista esta vacia*/
    (*l) = nuevo;
  else
  {
    aux = ulti_lisCarrito(*l);
    aux->sig = nuevo;
  }
  return OK;
}

/* La funcion elim_lisCarrito saca el elemento de la posicion pos de la lista Carrito.
Recibe:
listaCarrito *l: lista en la que se realizara la operacion de eliminacion
Int pos: posicion que determina el nodo a eliminar.
Retorna:
 OK en en caso de exito
Diferente de OK en caso de error*/
int elim_lisCarrito(listaCarrito *l, int pos)
{
  listaCarrito aux, borrar;
  if (!AP_VAL(l))
    return AP_INV;
  if (!TIE_DAT(*l))
    return UNDERFLOW;
  if (!(pos <= long_lisCarrito(l) && pos > 0))
    return POS_INV;

  aux = ante_lisCarrito(*l, pos);
  if (aux->sig)
  {
    borrar = aux->sig;
    aux->sig = borrar->sig;
    free(borrar);
  }
  return OK;
}

/*La funcion elip_lisCarrito se encarga de borrar el primer nodo de la lista Carrito.
Recibe:
listaCarrito *l cuyo primer nodo sera borrado
Retorna:
OK en en caso de exito
Diferente de OK en caso de error*/
int elip_lisCarrito(listaCarrito *l)
{
  apnodoCarrito borrar;
  if (!AP_VAL(l))
```

```
   return AP_INV;
  if (!TIE_DAT(*l))
   return UNDERFLOW;
  borrar = *l;
  *l = (*l)->sig;
  free(borrar);
  return OK;
}
/*La funcion str_lisCarrito convierte a cadena la lista Carrito, para que esta pueda ser visualizada
Recibe:
listaCarrito *l: apuntador a la lista Carrito cuya cadena sera formada
char *s: apuntador a caracter que almacenara la cadena
Devuelve:
char *str_lisCarrito: cadena resultante
*/

char *str_lisCarrito(char *s, listaCarrito *l)
{
  char aux[TAMMAX+1];
  char aux1[TAMMAX+1];
  char aux2[TAMMAX+1];
  char aux3[TAMMAX+1];
  char aux4[TAMMAX+1];
  char aux5[TAMMAX+1];
  listaCarrito l1, l2;
  l2 = NULL;
  l1 = *l;
  while (l1)
  {
   sprintf(aux, "%s", l1->dato->nombre_producto);
   strcat(s, aux);

   sprintf(aux1, "-%s", l1->dato->nombre_proveedor);
   strcat(s, aux1);
   sprintf(aux2, "-%d", l1->dato->cantidad);
   strcat(s, aux2);
   sprintf(aux3, "-%s", l1->dato->nombre_cliente);
   strcat(s, aux3);
   sprintf(aux4, "-%s", l1->dato->fecha);
   strcat(s, aux4);
   sprintf(aux5, "-%05d", l1->dato->ID_compra);
   strcat(s, aux5);
   strcat(s, "\n");
   l2 = l1;
   l1 = l1->sig;
   /*if(l1->sig != NULL)
    */
   //printf("\n%s\n",s);
  }
```

```
 return s;
}
/* La funcion buscar_lisCarrito se encarga de buscar un dato en una lista.
Recibe:
listaCarrito *l: lista en la que debe buscar
int datos_buscado:dato que se busca.
Regresa:posicion del dato buscado.*/
int buscar_lisCarrito(listaCarrito *l, char* datos_buscado)
{
 if (!AP_VAL(l))
   return AP_INV;
 if (!TIE_DAT(*l))
   return UNDERFLOW;

 listaCarrito l1, l2;
 //l2 es una lista auxiliar
 int i = 0;
 l2 = NULL;
 l1 = *l;
 while (l1)
 {
  if (datos_buscado == l1->dato->nombre_producto)
    return i;
  else
  {
    l2 = l1;
    l1 = l1->sig;
    i++;
  }
 }
 return NO_EXISTE;
}
```

listaCarrito.h

```
#ifndef LISTACARRITO_H
#define LISTACARRITO_H
/*************************************************************
Implementa una lista utilizando nodos de una liga.
*************************************************************/
#include "nodoCarrito.h"
/*****************M A C R O S*****************************/
#define TIE_DAT(p)  ((p)!=NULL)   /*Si tiene datos*/
#define VAC_LIS(p)  ((p)==NULL)   /* Si es un lista vacia*/
/***************T I P O     D E     D A T O***************/
typedef apnodoCarrito listaCarrito;
/************** F U N C I O N E S************************/
int crea_lisCarrito(listaCarrito* l); /*Inicializa la lista Carrito*/
int insf_lisCarrito(listaCarrito*,DATO_Carrito); /*Inserta el ultimo nodo*/
int dest_lisCarrito(listaCarrito*);      /*Elimina por completo una lista*/
int inse_lisCarrito(listaCarrito*,int,DATO_Carrito);/*inserta nuevos nodos en la lista Carrito*/
```

```
listaCarrito ante_lisCarrito(listaCarrito, int); /*Determina el nodo anterior a la posicion p de la lista Carrito*/
int long_lisCarrito(listaCarrito*);      /*Calcula la longitud de la lista*/
int buscar_lisCarrito(listaCarrito *l, char* datos_buscado);/*Busca la pagina en la lista*/
int elim_lisCarrito(listaCarrito*,int);  /*Saca un elemento del lista*/
int elip_lisCarrito(listaCarrito*);      /*Elimna el primer dato*/
char *str_lisCarrito(char *s, listaCarrito *l);
#endif
```

control.h

```
#ifndef CONTROL_H
#define CONTROL_H
#include <string.h>
#include <unistd.h>
#include <sys/shm.h>
#include "queueMensaje.h"
#include "listaUsuarios.h"
#include "listaProductos.h"


#define CLAVE      'S'

/**********FUNCIONES***********/
listaUsuarios crearUsuarios();
listaProductos crearProductos();
int actualizar_Productos(char *s); /*Actualiza el archivo de productos*/
listaCarrito crearCarritos();
/******************************/

#endif
```

control.c

```
#include "Control.h"

/*La funcion creaUsuarios se encarga de obtener los datos de usuario del archivo usuarios.txt
y crea una lista de usuarios
Retorna:
users: lista de usuarios*/
listaUsuarios crearUsuarios()
{
   char nombre[TAMMAX];
   char password[TAMMAX];
   char tipo[TAMMAX];

   FILE *f;
   f = fopen("usuarios.txt", "r");

   int j;
```

```
listaUsuarios users;
crea_lisUsuarios(&users);

usuario *user;
int i;
while (!feof(f))
{
   user = (usuario *)malloc((i + 1) * sizeof(usuario));
   //Limpieza del arreglo
   bzero(nombre, TAMMAX - 1);
   char aux;
   aux = '\0';
   for (j = 0; aux != '-'; j++)
   {
      aux = fgetc(f);
      if (aux != '-')
         nombre[j] = aux;
   }
   strcpy(user[i].nombre_usuario, nombre);

   bzero(password, TAMMAX - 1);
   char aux2;
   aux2 = '\0';
   for (j = 0; aux2 != '-'; j++)
   {
      aux2 = fgetc(f);
      if (aux2 != '-')
         password[j] = aux2;
   }
   strcpy(user[i].password, password);

   bzero(tipo, TAMMAX - 1);
   char aux3;
   aux3 = ' ';
   for (j = 0; aux3 != '\n' && !feof(f); j++)
   {
      aux3 = fgetc(f);
      if (aux3 != '-')
         tipo[j] = aux3;
   }

   user[i].tipo_usuario = atoi(tipo);
   user[i].sesion_activa = 0;

   insf_lisUsuarios(&users, &user[i]);
   i++;
}
fclose(f);

return users;
```

```
}

/*La funcion creaProductos se encarga de obtener los datos de usuario del archivo
productos.txt y crea una lista de usuarios
Retorna:
users: lista de usuarios*/
listaProductos crearProductos()
{
   char nombre[TAMMAX];
   char proveedor[TAMMAX];
   char numEx[TAMMAX];
   int j, i;

   FILE *f;
   f = fopen("productos.txt", "r");

   listaProductos productos;
   crea_lisProductos(&productos);
   producto *prod;
   while (!feof(f))
   {
      prod = (producto *)malloc((i + 1) * sizeof(producto));
      bzero(nombre, TAMMAX - 1);
      char aux;
      aux = '\0';
      for (j = 0; aux != '-'; j++)
      {
         aux = fgetc(f);
         if (aux != '-')
            nombre[j] = aux;
      }
      strcpy(prod[i].nombre_producto, nombre);

      bzero(proveedor, TAMMAX - 1);
      char aux2;
      aux2 = '\0';
      for (j = 0; aux2 != '-'; j++)
      {
         aux2 = fgetc(f);
         if (aux2 != '-')
            proveedor[j] = aux2;
      }
      strcpy(prod[i].nombre_proveedor, proveedor);

      bzero(numEx, TAMMAX - 1);
      char aux3;
      aux3 = ' ';
      for (j = 0; aux3 != '\n' && !feof(f); j++)
      {
         aux3 = fgetc(f);
```

```
            if (aux3 != '-')
               numEx[j] = aux3;
         }

         prod[i].num_existencia = atoi(numEx);

         insf_lisProductos(&productos, &prod[i]);
         i++;
      }
      fclose(f);

      return productos;
}

int actualizar_Productos(char *s)
{
      FILE *f;
      f = fopen("productos.txt", "w");
      fprintf(f, "%s", s);
      fclose(f);
      return OK;
}

/* La funcion crearCarritos crea los nodos de la listaCarritos a partir del archivo txt correspondiente.*/
listaCarrito crearCarritos()
{
      char nombre[TAMMAX];
      char proveedor[TAMMAX];
      char numEx[TAMMAX];
      char cliente[TAMMAX];
      char fecha[TAMMAX];
      char ID[TAMMAX];

      int j, i;

      FILE *f;
      f = fopen("carritos.txt", "r");

      listaCarrito carritos;
      crea_lisCarrito(&carritos);
      carrito *car;

      while (!feof(f))
      {
         //Obtencion del nombre de producto
         car = (carrito *)malloc((i + 1) * sizeof(carrito));
         bzero(nombre, TAMMAX - 1);
         char aux;
         aux = '\0';
```

```
for (j = 0; aux != '-'; j++)
{
   aux = fgetc(f);
   if (aux != '-')
      nombre[j] = aux;
}
strcpy(car[i].nombre_producto, nombre);

//Obtencion del nombre del proveedor
bzero(proveedor, TAMMAX - 1);
char aux2;
aux2 = '\0';
for (j = 0; aux2 != '-'; j++)
{
   aux2 = fgetc(f);
   if (aux2 != '-')
      proveedor[j] = aux2;
}
strcpy(car[i].nombre_proveedor, proveedor);

//Obtencion del numero de existencias
bzero(numEx, TAMMAX - 1);
char aux3;
aux3 = '\0';
for (j = 0; aux3 != '-'; j++)
{
   aux3 = fgetc(f);
   if (aux3 != '-')
      numEx[j] = aux3;
}

car[i].cantidad = atoi(numEx);

//Obtencion del nombre del cliente
bzero(cliente, TAMMAX - 1);
char aux4;
aux4 = '\0';
for (j = 0; aux4 != '-'; j++)
{
   aux4 = fgetc(f);
   if (aux4 != '-')
      cliente[j] = aux4;
}
strcpy(car[i].nombre_cliente, cliente);

//Obtencion de la fecha
bzero(fecha, TAMMAX - 1);
char aux5;
aux5 = '\0';
for (j = 0; aux5 != '-'; j++)
```

```
      {
         aux5 = fgetc(f);
         if (aux5 != '-')
            fecha[j] = aux5;
      }
      strcpy(car[i].fecha, fecha);

      //Obtencion del ID
      bzero(ID, TAMMAX - 1);
      char aux6;
      aux6 = ' ';
      for (j = 0; aux6 != '\n' && !feof(f); j++)
      {
         aux6 = fgetc(f);
         if (aux6 != '-')
            ID[j] = aux6;
      }

      car[i].ID_compra = atoi(ID);

      //Insercion del nodo en la listaCarrito
      insf_lisCarrito(&carritos, &car[i]);
      i++;
   }
   fclose(f);

   return carritos;
}
```

queueMensaje.h

```
#ifndef QUEUEMENSAJE_H
#define QUEUEMENSAJE_H
#include "listaUsuarios.h"
#include "listaProductos.h"
#include "listaCarrito.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <string.h>
#include <sys/msg.h>
#include <sys/shm.h>

typedef struct
{
 long pid; //ID del proceso
 int orden; //Indica el numero de pedido realizado por un usuario
 usuario user;
 producto prod;
```

```
} mensaje;
#define LONGITUD (sizeof(mensaje)-sizeof(long))

  //producto product[TAMMAX];
  //char clave_sem;
/**********OPCIONES***********/
#define SESION         1
#define VER_PRODUCTOS   2
#define BUY_PRODUCTOS   3
#define ADD_PRODUCTOS   4
#define MOD_PRODUCTOS   5
#define HIS_PRODUCTOS   6
#define FIN            7
#define ERROR          8
#define INICIADO       9
/*****************************/

/**********CONSTANTES*********/
#define PERMISOS 0666
#define FICHERO_LLAVE "Control"
#define FICHERO_MEMORIA "File1"
#define CLAVE_USER_CONTROL  'K'
#define CLAVE_CONTROL_USER  'L'
#define CLAVE_MEMORIA       'M'

/*****************************/

/**********FUNCIONES***********/
void menu();
/*****************************/
#endif
```

mainControl.c

```
#include "Control.h"
#include "queueMensaje.h"

int main(int argc, char const *argv[])
{
   int cola_userctrl, cola_ctrluser, pid;
   int opcion;
   int id = 0;
   mensaje msg;
   key_t llave;

   listaCarrito carritos;
   listaUsuarios users;
   listaProductos productos;
   char cadUsers[TAMMAX];
   char *cadProductos;
   char *cadCarrito;
```

```
  /*Creacion de llave para ser usada en cola de mensajes de usuario a control*/
  llave = ftok(FICHERO_LLAVE, CLAVE_USER_CONTROL);
  if (llave == -1)
  {
    printf("La llave no fue creada");
    exit(0);
  }
  /*Obtencion del identificador de la cola de mensajes de los usuarios*/
  if ((cola_userctrl = msgget(llave, IPC_CREAT | PERMISOS)) == -1)
  {
    //En caso de que el proceso de creacion del identificador de la cola de mensajes tenga un
error.
    perror("El identificador de cola de mensajes de usuarios no fue creado");
    exit(-1);
  }
  /*Creacion de llave para cola de mensajes de control a usuario*/
  llave = ftok(FICHERO_LLAVE, CLAVE_CONTROL_USER);

  /*Obtencion del identificador de la cola de mensajes del programa de control*/
  if ((cola_ctrluser = msgget(llave, IPC_CREAT | PERMISOS)) == -1)
  {
    //En caso de que el proceso de creacion del identificador de la cola de mensajes tenga un
error.
    perror("El identificador de cola de mensajes de control no fue creado");
    exit(-1);
  }

  users = crearUsuarios();
  //Impresión de la lista de usuarios
  str_lisUsuarios(cadUsers, &users);
  printf("\n %s", cadUsers);

  //Creacion de memoria compartida de la cadena de Productos
  int shmid;
  llave = ftok(FICHERO_MEMORIA_PRODUCT, CLAVE_MEMORIA_PRODUCT);
  shmid = shmget(llave, sizeof(char *), IPC_CREAT | PERMISOS);
  cadProductos = shmat(shmid, 0, 0);

  /*Memoria compartida para Carrito*/
  int shmid2;
  llave = ftok(FICHERO_MEMORIA_CARRITO, CLAVE_MEMORIA_CARRITO);
  shmid2 = shmget(llave, sizeof(char *), IPC_CREAT | PERMISOS);
  cadCarrito = shmat(shmid2, 0, 0);

  productos = crearProductos();

  while (1)
  {
    /*Recepcion de mensajes de la cola usuario-control, el mensaje se guarda en &msg*/
    msgrcv(cola_userctrl, &msg, LONGITUD, 0, 0);
```

```
    char a[TAMMAX];
    printf("%d", msg.orden);
    switch (msg.orden)
    {
    case SESION:
      printf("\nEl usuario: %lu solicito iniciar sesion\n", msg.pid);
      //Validación de que el usuario exista
      if (validar_lisUsuarios(&users, msg.user.nombre_usuario, msg.user.password,
msg.user.tipo_usuario) < 0)
      {
        //En caso de no encontrar los datos de usuario ingresados
        msg.orden = ERROR;
        printf("\nEl usuario no fue encontrado:\n");
        printf("nombre:%d\n", msg.user.tipo_usuario);
      }
      else
      {
        //En caso de que si se encuentren los datos de usuario ingresados
        msg.orden = INICIADO;
        msg.CLAVE_SEM = CLAVE;
        //msg.clave_sem = CLAVE_SEM;
        printf("\nEl usuario fue encontrado\n");
      }
      //Regresar mensaje para indicar que fue recibido correctamente
      msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
      break;

    case VER_PRODUCTOS:
      //Fin indica la recepcion exitosa del mensaje
      msg.orden = FIN;
      strcpy(cadProductos, "");
      str_lisProductos(cadProductos, &productos);
      printf("\n %s", cadProductos);
      printf("\nEl usuario: %lu solicito ver productos\n", msg.pid);
      msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
      break;

    case BUY_PRODUCTOS:
      printf("\nEl usuario: %lu solicito comprar productos\n", msg.pid);
      msg.orden = FIN;
      /*Si hay suficientes existencias del articulo la compra procede*/

      if (dis_existencias(&productos, msg.prod.nombre_producto,
msg.prod.nombre_proveedor, msg.prod.num_existencia) > 0)
      {
        FILE *car;
        time_t t;
        struct tm *tm;
        char fechayhora[100];
```

```
            t = time(NULL);
            tm = localtime(&t);
            strftime(fechayhora, 100, "%d/%m/%Y %H:%M", tm);

            car = fopen("carritos.txt", "a");
            if (car == NULL)
               printf("No puelo toy chikito");
            else
               printf("Si puelo");

            fprintf(car, "\n%s-%s-%d-%s-%s-%05d", msg.prod.nombre_producto,
msg.prod.nombre_proveedor, msg.prod.num_existencia, msg.user.nombre_usuario,
fechayhora, ++id);
            //Nombre del producto – Nombre de proveedor – Cantidad – Nombre de cliente –
Fecha – ID de compra

            fclose(car);
            carritos = crearCarritos();
            strcpy(cadCarrito, "");
            str_lisCarrito(cadCarrito, &carritos);
            printf("\n %s", cadCarrito);

            strcpy(cadProductos, "");
            str_lisProductos(cadProductos, &productos);
            actualizar_Productos(cadProductos);
         }
         else
         {
            printf("\nno esta el producto o no hay suficientes xd\n");
            msg.orden = ERROR;
         }
         msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
         break;
         /*Agregar productos al stock*/
      case ADD_PRODUCTOS:
         printf("\nEl usuario: %lu solicito agregar productos\n", msg.pid);
         msg.orden = FIN; //Solicitud recibida
         productos = crearProductos();
         strcpy(cadProductos, "");
         str_lisProductos(cadProductos, &productos); //Mostrar listado de productos
         msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
         break;
         /*Modificar productos*/
      case MOD_PRODUCTOS:
         printf("\nEl usuario: %lu solicito modificar productos\n", msg.pid);
         if (buscar_lisProductos(&productos, msg.prod.nombre_producto,
msg.prod.nombre_proveedor))
         {
            msg.orden = FIN;
```

```
            dis_existencias(&productos,                    msg.prod.nombre_producto,
msg.prod.nombre_proveedor, (-1) * msg.prod.num_existencia);
            strcpy(cadProductos, "");
            str_lisProductos(cadProductos, &productos);
            actualizar_Productos(cadProductos);
          }
          else
          {
            msg.orden = ERROR;
          }

          msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
          break;
          /*Historial de productos*/
        case HIS_PRODUCTOS:
          msg.orden = FIN;
          strcpy(cadCarrito, "");
          str_lisCarrito(cadCarrito, &carritos);
          printf("\n %s", cadCarrito);
          printf("\nEl usuario: %lu solicito ver carrito de compras\n", msg.pid);
          msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
         break;
          /*Caso que indica la existencia de errores*/
        case ERROR:
          msg.orden = ERROR;
          printf("\nHa ocurrido un error :c\n");
          msgsnd(cola_ctrluser, &msg, LONGITUD, 0);
          break;

        case FIN:
          msgctl(cola_userctrl, IPC_RMID, 0);
          msgctl(cola_ctrluser, IPC_RMID, 0);
          printf("\nLa cola de mensajes ha finalizado\n");
          exit(0);
          break;
        }
    }

    return 0;
}
```

mainUsuario.c

```
#include "Usuario.h"
int main(int argc, char const *argv[])
{
    int cola_userctrl, cola_ctrluser, pid;
    listaCarrito lisCarritoDani;
    listaUsuarios users;
    listaProductos productos;
    char cadUsers[TAMMAX];
```

```
char *cadProductos;
char *cadCarrito;

mensaje msg;
key_t llave;
enum
{
   NO,
   SI
} recibir = NO;

/*Creacion de las colas de mensajes*/
llave = ftok(FICHERO_LLAVE, CLAVE_USER_CONTROL);

if ((cola_userctrl = msgget(llave, IPC_CREAT | PERMISOS)) == -1)
{
   perror("msget");
   exit(-1);
}

llave = ftok(FICHERO_LLAVE, CLAVE_CONTROL_USER);

if ((cola_ctrluser = msgget(llave, IPC_CREAT | PERMISOS)) == -1)
{
   perror("msget");
   exit(-1);
}
/*Memoria compartida para Productos*/
int shmid;
llave = ftok(FICHERO_MEMORIA_PRODUCT, CLAVE_MEMORIA_PRODUCT);
shmid = shmget(llave, sizeof(char *), IPC_CREAT | PERMISOS);
cadProductos = shmat(shmid, 0, 0);
msg.pid = pid = getpid();

/*Memoria compartida para Carrito*/
int shmid2;
llave = ftok(FICHERO_MEMORIA_CARRITO, CLAVE_MEMORIA_CARRITO);
shmid2 = shmget(llave, sizeof(char *), IPC_CREAT | PERMISOS);
cadCarrito = shmat(shmid2, 0, 0);

usuario userActual;
while (1)
{
   menu();
   int orden;
   printf("\nEscribe la opcion que desees:\n");
   scanf("%d", &orden);
   switch (orden)
   {
      /*Caso para iniciar sesion*/
```

```
case SESION:
   recibir = SI;
   msg.orden = orden;
   userActual = ini_sesion();
   msg.user = userActual;

   printf("nombre:%s\n", msg.user.nombre_usuario);
   printf("pass:%s\n", msg.user.password);
   printf("tipo:%d\n", msg.user.tipo_usuario);
   msgsnd(cola_userctrl, &msg, LONGITUD, 0);
   break;
   /*Visualizacion de los productos en stock*/
case VER_PRODUCTOS:
down(semaforo);
   if (userActual.sesion_activa)
   {
      recibir = SI;
      msg.orden = orden;
      printf("\n%s\n", cadProductos);
   }
   else
   {
      recibir = SI;
      msg.orden = ERROR;
      printf("\nno has iniciado sesion -_-\n");
   }
   msgsnd(cola_userctrl, &msg, LONGITUD, 0);
   break;
   /*Comprar productos*/
case BUY_PRODUCTOS:
down(semaforo);
   if (userActual.sesion_activa && !userActual.tipo_usuario)
   {
      char producto[TAMMAX];
      char proveedor[TAMMAX];
      int numEx;

      recibir = SI;
      msg.orden = orden;
      strcpy(msg.user.nombre_usuario, userActual.nombre_usuario);
      strcpy(producto, "");
      strcpy(proveedor, "");
      printf("\nEscribe el nombre del producto: ");
      scanf("%s", producto);
      strcpy(msg.prod.nombre_producto, producto);
      printf("\nEscribe el nombre del proveedor: ");
      scanf("%s", proveedor);
      strcpy(msg.prod.nombre_proveedor, proveedor);
      printf("\nEscribe la cantidad deseada: ");
      scanf("%d", &numEx);
```

```
      msg.prod.num_existencia = numEx;
   }
   else
   {
      recibir = SI;
      msg.orden = ERROR;
      printf("\nNo has iniciado sesion o no eres Cliente\n");
   }
   msgsnd(cola_userctrl, &msg, LONGITUD, 0);
   break;
   /*Agregar productos, para ello se necesita que el usuario sea de tipo proveedor*/
case ADD_PRODUCTOS:
down(semaforo);
   if (userActual.sesion_activa && userActual.tipo_usuario)
   {
      FILE *f;
      f = fopen("productos.txt", "a");
      char continuar = 's';
      char nombre[TAMMAX];
      int numEx;

      recibir = SI;
      msg.orden = orden;

      do
      {
         printf("\nEscribe el nombre del producto: ");
         scanf("%s", nombre);

         printf("\nEscribe la cantidad: ");
         scanf("%d", &numEx);

         fprintf(f, "\n%s-%s-%d", nombre, userActual.nombre_usuario, numEx);

         printf("\n¿Quiere agregar otro producto? s/n: ");
         scanf("%s", &continuar);
      } while (continuar == 's');
      fclose(f);
   }
   else
   {
      recibir = SI;
      msg.orden = ERROR;
      printf("\nNo has iniciado sesion o no tienes los permisos para realizar la accion\n");
   }
   msgsnd(cola_userctrl, &msg, LONGITUD, 0);
   break;
   /*Modificar productos*/
case MOD_PRODUCTOS:
down(semaforo);
```

```
    if (userActual.sesion_activa && userActual.tipo_usuario)
    {
       char producto[TAMMAX];
       int numEx;

       recibir = SI;
       msg.orden = orden;
       strcpy(producto, "");
       printf("\nEscribe el nombre del producto: ");
       scanf("%s", producto);
       strcpy(msg.prod.nombre_producto, producto);
       strcpy(msg.prod.nombre_proveedor, userActual.nombre_usuario);
       printf("\nEscribe la cantidad a incrementar: ");
       scanf("%d", &numEx);
       msg.prod.num_existencia = numEx;
    }
    else
    {
       recibir = SI;
       msg.orden = ERROR;
       printf("\nNo has iniciado sesion o no tienes los permisos para realizar la accion\n");
    }
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;
    /*Historial de productos del usuario*/
  case HIS_PRODUCTOS:
    if (userActual.sesion_activa)
    {
       recibir = SI;
       msg.orden = orden;
       printf("\n%s\n", cadCarrito);
    }
    else
    {
       recibir = SI;
       msg.orden = ERROR;
       printf("\nno has iniciado sesion\n");
    }
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;

  case FIN:
    recibir = NO;
    msg.orden = FIN;
    msgsnd(cola_userctrl, &msg, LONGITUD, 0);
    break;
  }
   default:
    recibir = NO;
    printf("no se reconoce\n");
```

```
            break;
        }
        if (recibir == SI)
        {
            do
            {
                msgrcv(cola_ctrluser, &msg, LONGITUD, pid, 0);

                switch (msg.orden)
                {
                case FIN:
                up(semaforo);
                    break;

                case ERROR:
                    printf("\nQue hubo un error dice\n");
                    break;

                case INICIADO:
                    userActual.sesion_activa = 1;
                    printf("\nInicio correctamente\n");
                    printf("\nsemaforo %c\n", msg.CLAVE_SEM);
                    llave = ftok(FICHERO_SEMAFORO, msg.CLAVE_SEM);
                    semaforo = Crea_semaforo(llave, 1);
                    break;
                default:
                    printf("no se reconoce\n");
                    break;
                }
            } while (!msg.orden);
        }
    }

    return 0;
}
```

nodoProductos.h

```
#ifndef NODOPRODUCTOS_H
#define NODOPRODUCTOS_H
/*********************************************************
Declara un nodo con una liga
*********************************************************/
#include "err.h"
/***************C O N S T A N T E S***********************/
#define DATO_Productos producto *
#define FORMATOVirtual " %p"
#define TAMMAX       1000
/*************** T I P O     D E     D A T O*****************/
typedef struct
{
```

```
  char nombre_producto[TAMMAX];
  int num_existencia;
  char nombre_proveedor[TAMMAX];
} producto;
typedef struct _nodoProductos
{
  DATO_Productos dato;
  struct _nodoProductos *sig;
} nodoProductos;
typedef nodoProductos *apnodoProductos;
/***************F U N C I O N E S ****************************/
int crea_nodProductos(apnodoProductos *p, DATO_Productos d); /*Crear un nodo*/
int elim_nodProductos(apnodoProductos *p);              /*Eliminar un nodo*/
char *str_nodProductos(char *, const apnodoProductos);    /*Convertir nodo a cadena*/
#endif
```

nodoProductos.c

```
#include "nodoProductos.h"
/* Convierte un nodo en una cadena
Recibe:
  Cadena que almacenara la representacion del nodo
  Apuntador a nodo a ser convertido
Regresa:
  El mismo apuntador que recibe
*/
char *str_nodProductos(char *s, const apnodoProductos ap)
{
  if (!(AP_VAL(s) && AP_VAL(ap)))
    return s;
  sprintf(s, FORMATOVirtual, ap->dato);
  return s;
}
/* Inicializa un nodo
Recibe:
  Apuntador a nodo a ser inicializado
  Dato para inicializar el nodo
Regresa
  Codigo de error
*/
int crea_nodProductos(apnodoProductos *p, DATO_Productos d)
{
  if (!AP_VAL(p))
    return AP_INV;
  *p = (nodoProductos *)malloc(sizeof(nodoProductos));
  if (!AP_VAL(*p))
    return NO_MEM;
  (*p)->dato = d;
  (*p)->sig = NULL;
  return OK;
}
```

```
/* Elimina un nodo
Recive
 Apuntador a nodo a ser eliminado
Regresa
 Codigo de error
*/
int elim_nodProductos(apnodoProductos *p)
{
 if (!AP_VAL(p))
   return AP_INV;
 free(*p);
 *p = NULL;
 return OK;
}
```

nodoUsuarios.h

```
#ifndef NODOUSUARIOS_H
#define NODOUSUARIOS_H
/*************************************************************
Declara un nodo con una liga
*************************************************************/
#include "err.h"
#include <stdio.h>
#include <string.h>
#define TAMMAX 1000
/***************C O N S T A N T E S************************/
#define DATO_Usuarios usuario*
#define FORMATOVirtual " %p"
/*************** T I P O    D E    D A T O*****************/
typedef struct
{
 char nombre_usuario[TAMMAX];
 char password[TAMMAX]; //Contrasenia de inicio de sesion
 int tipo_usuario; //Indica si el usuario es comprador o vendedor
 int sesion_activa; //Indica si el usuario ha iniciado sesion
} usuario;
typedef struct _nodoUsuarios
{
 DATO_Usuarios dato;
 struct _nodoUsuarios *sig;
} nodoUsuarios;
typedef nodoUsuarios *apnodoUsuarios;
/***************F U N C I O N E S ****************************/
int crea_nodUsuarios(apnodoUsuarios *p, DATO_Usuarios d); /*Crear un nodo*/
int elim_nodUsuarios(apnodoUsuarios *p);              /*Eliminar un nodo*/
char *str_nodUsuarios(char *, const apnodoUsuarios);    /*Convertir nodo a cadena*/
#endif
```

nodoUsuarios.c

```
#include "nodoUsuarios.h"
```

```
/* Convierte un nodo en una cadena
Recibe:
 Cadena que almacenara la representacion del nodo
 Apuntador a nodo a ser convertido
Regresa:
 El mismo apuntador que recibe
*/
char *str_nodUsuarios(char *s, const apnodoUsuarios ap)
{
 if (!(AP_VAL(s) && AP_VAL(ap)))
   return s;
 sprintf(s, FORMATOVirtual, ap->dato);
 return s;
}
/* Inicializa un nodo
Recibe:
 Apuntador a nodo a ser inicializado
 Dato para inicializar el nodo
Regresa
 Codigo de error
*/
int crea_nodUsuarios(apnodoUsuarios *p, DATO_Usuarios d)
{
 if (!AP_VAL(p))
   return AP_INV;
 *p = (nodoUsuarios *)malloc(sizeof(nodoUsuarios));
 if (!AP_VAL(*p))
   return NO_MEM;
 (*p)->dato = d;
 (*p)->sig = NULL;
 return OK;
}
/* Elimina un nodo
Recibe
 Apuntador a nodo a ser eliminado
Regresa
 Codigo de error
*/
int elim_nodUsuarios(apnodoUsuarios *p)
{
 if (!AP_VAL(p))
   return AP_INV;
 free(*p);
 *p = NULL;
 return OK;
}
```

nodoCarrito.h

```
#ifndef NODOCARRITO_H
#define NODOCARRITO_H
```

```
/*********************************************************
Declara un nodo con una liga
*********************************************************/
#include "err.h"
#include <stdio.h>
#include <string.h>
/***************C O N S T A N T E S************************/
#define DATO_Carrito carrito *
#define FORMATOVirtual " %p"
/*************** T I P O    D E    D A T O*****************/
typedef struct
{
  char* nombre_producto;
  char* nombre_proveedor;
  int cantidad;
} carrito;
typedef struct _nodoCarrito
{
  DATO_Carrito dato;
  struct _nodoCarrito *sig;
} nodoCarrito;
typedef nodoCarrito *apnodoCarrito;
/***************F U N C I O N E S ****************************/
int crea_nodCarrito(apnodoCarrito *p, DATO_Carrito d); /*Crear un nodo*/
int elim_nodCarrito(apnodoCarrito *p);              /*Eliminar un nodo*/
char *str_nodCarrito(char *, const apnodoCarrito);    /*Convertir nodo a cadena*/
#endif
```

nodoCarrito.c

```
#include "nodoCarrito.h"
/* Convierte un nodo en una cadena
Recibe:
  Cadena que almacenara la representacion del nodo
  Apuntador a nodo a ser convertido
Regresa:
  El mismo apuntador que recibe
*/
char *str_nodCarrito(char *s, const apnodoCarrito ap)
{
  if (!(AP_VAL(s) && AP_VAL(ap)))
    return s;
  sprintf(s, FORMATOVirtual, ap->dato);
  return s;
}
/* Inicializa un nodo
Recibe:
  Apuntador a nodo a ser inicializado
  Dato para inicializar el nodo
Regresa
  Codigo de error
```

```
*/
int crea_nodCarrito(apnodoCarrito *p, DATO_Carrito d)
{
 if (!AP_VAL(p))
   return AP_INV;
 *p = (nodoCarrito *)malloc(sizeof(nodoCarrito));
 if (!AP_VAL(*p))
   return NO_MEM;
 (*p)->dato = d;
 (*p)->sig = NULL;
 return OK;
}
/* Elimina un nodo
Recive
 Apuntador a nodo a ser eliminado
Regresa
 Codigo de error
*/
int elim_nodCarrito(apnodoCarrito *p)
{
 if (!AP_VAL(p))
   return AP_INV;
 free(*p);
 *p = NULL;
 return OK;
}
```

Usuario.c

```
#include "Usuario.h"
/*La función Menu, muestra las distintas opciones que puede seleccionar el usuario*/
void menu()
{
   printf("\n------------Menu-----------\n\n");
   printf("Selecciona una opcion\n\n");
   printf("1 Iniciar sesión\n");
   printf("2 Ver los productos\n");
   printf("3 Comprar un producto\n");
   printf("4 Agregar un producto\n");
   printf("5 Modificar un producto\n");
   printf("6 Mostrar historial de compras \n");
   printf("7 Finalizar programa\n");
   printf("\n----------------------------\n\n");
}

/*Funcion ini_sesion, solicita al usuario los datos para el ingreso al sistema
Retorna:
user: una estructura de usuario
*/
usuario ini_sesion()
```

```
{
   usuario user;
   char nombre_usuario[TAMMAX];
   char password[TAMMAX];
   int tipo_usuario=0;;
   strcpy(nombre_usuario,"");
   strcpy(password,"");
   printf("\nEscribe el nombre de usuario: ");
   scanf("%s", nombre_usuario);
   strcpy(user.nombre_usuario,nombre_usuario);

   printf("\nEscribe el password: ");
   scanf("%s", password);
   strcpy(user.password,password);

   printf("\nEscribe el tipo de usuario: ");
   scanf("%d", &tipo_usuario);
   user.tipo_usuario = tipo_usuario;
   return user;
}
//Funcion que crea el semaforo que media la secuencia de procesamiento de los procesos
int Crea_semaforo(key_t llave, int valor_inicial)
{
   int semid = semget(llave, 1, IPC_CREAT | PERMISOS);
   if (semid == -1)
   {
      return -1;
   }
   semctl(semid, 0, SETVAL, valor_inicial);
   return semid;
}
//Funcion que decrementa el valor del semaforo. Como argumento recibe el identificador del
semaforo en cuestion.
void down(int semid)
{
   struct sembuf op_p[] = {0, -1, 0};
   semop(semid, op_p, 1);
}
//Funcion que incrementa el valor del semaforo. Como argumento recibe el identificador del
semaforo en cuestion.
void up(int semid)
{
   struct sembuf op_v[] = {0, +1, 0};
   semop(semid, op_v, 1);
}
```

Usuario.h

```
#ifndef USUARIO_H
#define USUARIO_H
```

```
#include "queueMensaje.h"
/***************variables globales semaforo************/
key_t llave;
int semaforo;
/***************FUNCIONES*****************/
/*imprime menu*/
void menu();
/*inica sesion del usuario*/
usuario ini_sesion();
/************SEMAFOROS***************/
/*crea los semaforos*/
int Crea_semaforo(key_t llave, int valor_inicial);
/*bloquea el semaforo*/
void down(int semid);
/*desbloquea el semaforo*/
void up(int semid);
#endif
```

Protocolo

Implementación de un sistema de compras electrónico utilizando herramientas IPC.

Trabajo Terminal No. 1

Alumnos: López Hernández Lissete, Palacios Cabrera Alberto, Pérez Priego Mario Daniel
Director: Jiménez Benítez José Alfredo
e-mail:
llopezh1600@alumno.ipn.mx

Resumen – En este proyecto se desarrollará un portal de comercio electrónico, que permita al usuario registrarse, buscar artículos, guardarlos en un carrito de compras, para finalmente adquirirlos. Todo con el objetivo de brindar al usuario una experiencia de compra sencilla y rápida. Para ello, se utilizará herramientas IPC (Inter Process Communication) para la sincronización y comunicación programados mediante lenguaje C.

Palabras clave – hilos, memoria compartida, semáforos, procesos.

1. Introducción

En los últimos años el comercio electrónico ha crecido exponencialmente. Principalmente a partir del año 2020, como consecuencia de la Pandemia Mundial por Covid-19.

En medio del cierre de negocios y la disminución de la movilidad social por la pandemia del Covid-19, las ventas online en México crecieron 81% en 2020 [1]. Se estimó que más de un 39% de la población mexicana adquiría bienes o servicios en línea. Solo tres años antes, en 2017, el porcentaje de compradores digitales no superaba el 30%. Se calcula que esta tendencia al alza continúe en los próximos años, rozando el 58% de penetración en 2025 [2].

A partir de ello los desarrolladores se han centrado en el modelado e implementación de tiendas virtuales. Dichos sistemas requieren de arquitecturas robustas que permitan el manejo de múltiples usuarios que puedan ejecutar diversas operaciones dependiendo de su rol dentro del sistema.

Los entornos más utilizados por sus múltiples beneficios en el área de servicios son los basados en Linux. Esto se debe a que es uno de los sistemas operativos más robustos, estables y rápidos. Además, al ser un sistema operativo libre, se tiene libertad de copia y de distribución, por lo que tener acceso a una distribución de manera gratuita es bastante sencillo. Al ser un software de código libre, también se tiene la posibilidad de modificar el código fuente para adaptar el sistema operativo a nuestras necesidades específicas [3].

Como se ha mencionado, una tienda en línea requiere de una arquitectura robusta que permita el correcto funcionamiento del sistema, por lo cual, Linux es una de las opciones más fiables en este ámbito, ya que, entre sus muchos beneficios, se caracteriza por ser un sistema operativo multitarea y multiprocesos.

A nivel mundial existen múltiples empresas que poseen estas características y se han posicionado como los líderes en e-commerce gracias a la experiencia que ofrecen a sus usuarios. Forbes en [4] nos menciona algunas de las empresas más grandes y dominantes en el ámbito del e-commerce. En la Tabla. 1 podemos ver algunas de las empresas más relevantes.

Tabla. 1 Principales empresas de e-commerce

NOMBRE	DESCRIPCIÓN	INGRESOS
Amazon	• Gran catálogo de productos • Buena experiencia al cliente • Ventas internacionales • Entregas rápidas	• Ingresos de 41,600 millones de dólares en 2020 [5]
Ebay	• Ventas en precio fijo o subasta • Ventas internacionales	• Ingresos de 5,667 millones de dólares en 2020 [6]
Costco	• Gran catálogo de productos • Requiere membresía • Productos exclusivos	• Ventas netas de 166,761 millones de dólares en 2020 [7]
Walmart	• Gran catálogo de productos • Tienda minorista más grande a nivel mundial	• Ingresos de 134,710 millones de dólares en 2020 [8]
Mercado Libre	• Gran catálogo de productos • Venta sin comisiones, costo por publicidad • Permite pago de servicios • Ventas internacionales	• Ingresos de 575 millones de dólares en 2020 (México) [9]

Forbes México menciona en [4] que entre los principales retos del comercio electrónico en México destaca la falta de información al consumidor y una mayor facilidad en el proceso de compra, por lo tanto tener un enfoque en la eficiencia y facilidad del procesos de compra es algo de gran importancia para el éxito de un sistema de compras.

2. Objetivo

Desarrollar un sistema que permita la gestión de una tienda electrónica, el cual facilitará los procesos de compra y gestión de productos, tanto para el usuario cliente como proveedor. El sistema debe de controlar el flujo de datos y la sincronización de múltiples procesos, brindando un software robusto y eficiente que apoye a la automatización de un comercio. Para ello se hará uso de herramientas IPC para su aplicación

Objetivos específicos

- Permitir el acceso de múltiples usuarios concurrentes de manera ágil
- Lograr la consistencia en los datos del sistema multiusuario

- Diseñar una interfaz de uso sencillo para el usuario
- Diseñar el sistema para ser ejecutado en arquitecturas de recursos bajos
- Proteger la integridad del sistema usando herramientas que restrinjan el acceso según el tipo de usuario.

3. Justificación

El comercio electrónico es de los métodos de compra más recurridos en actualidad, debido a la facilidad y comodidad con la que se puede buscar un producto, comprarlo y recibirlo en casa, además de ser fomentado por la nueva normalidad.

Con el objetivo de incentivar a más usuarios mayores de edad a recurrir a este método de compra, es fundamental que la experiencia del usuario sea fluida.

Para ello se propone el desarrollo de una plataforma de compras electrónica sobresaliente por su rapidez de ejecución, con el objetivo de que el usuario pase menos tiempo esperando una respuesta entre cada solicitud, lo que le permite invertir ese tiempo ahorrado en otras actividades o en más búsquedas de productos.

El lenguaje de programación con el que se programará dicha plataforma es lenguaje C, que se caracteriza por ser más rápido que lenguajes más modernos como Java o Python. Las estructuras de programación que se implementarán serán las de control entre procesos IPC, como semáforos, hilos, procesos y memoria compartida. En lo referente a los conocimientos que se aplicarán en este proyecto destacan los de las áreas de Sistemas Operativos y Algoritmia y programación estructurada.

4. Productos o Resultados esperados

El procedimiento que sigue este sistema se detalla de manera esquemática en la Figura. 1 en la cual se describen los diferentes módulos o procesos que trabaja nuestro sistema.

A. Solicitar Carrito: en este módulo el cliente podrá acceder al archivo de carritos, dónde podrá solicitar un carrito.
B. Guardar Carrito: registro de los productos a comprar para poder generar un carrito.
C. Archivo de carritos: recopila y almacena la información acerca de los carritos generados.
D. Proceso de Control: el cliente podrá acceder al sistema proporcionando los datos, si el usuario existe, el proceso de control le proporcionará la clave para poder hacer uso de los semáforos. También creará los diferentes hilos para cada usuario ingresado.
E. Búsqueda de productos: realiza la búsqueda para confirmar la existencia de un producto en el archivo de catálogo de productos.
F. Agregar artículo: en este módulo el proveedor podrá ingresar un nuevo artículo al archivo de productos.
G. Agregar existencia: al existir un producto, el proveedor podrá agregar un número de artículos en existencia de dicho producto.
H. Búsqueda de artículo: a diferencia del cliente, el proveedor podrá acceder directamente con el archivo de catálogo ya que tendrá desde un inicio las claves necesarias para poder usar un semáforo.
I. Archivo de clientes: almacena la información acerca de los clientes.
J. Archivo catálogo: recopila información acerca de los productos de la tienda.

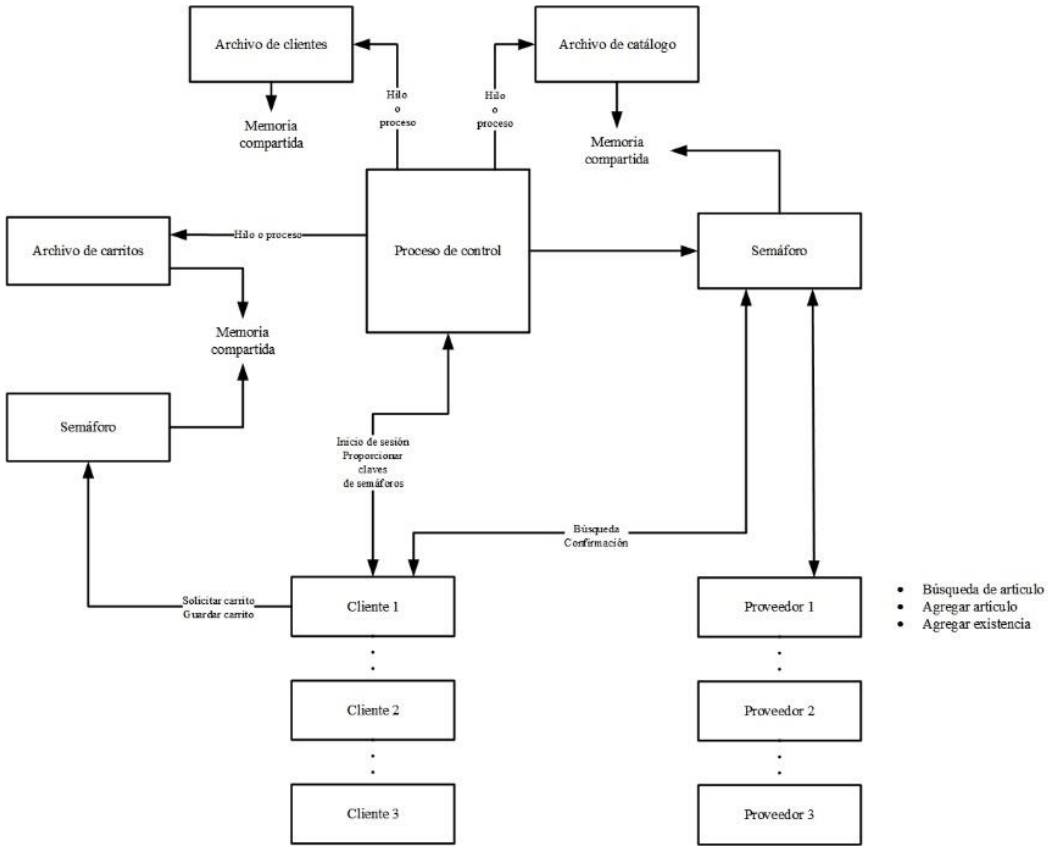

Figura. 1 Arquitectura del sistema.

5. *Metodología*

Se empleará en el proyecto la metodología SCRUM al ser una de las más utilizadas para el desarrollo de software en la actualidad. La norma que otorga las métricas que se usarán es la ISO 9126, que determina la calidad del software en base a su funcionalidad.

SCRUM

En [10] se define a Scrum como una metodología de desarrollo ágil utilizada en el desarrollo de Software basada en procesos iterativos e incrementales. Scrum es un marco ágil adaptable, rápido, flexible y eficaz que está diseñado para entregar valor al cliente durante todo el desarrollo del proyecto. El objetivo principal de Scrum es satisfacer las necesidades del cliente a través de un entorno de transparencia en la comunicación, responsabilidad colectiva y progreso continuo. El desarrollo parte de una idea general de lo que se necesita construir, elaborando una lista de características ordenadas por prioridad (cartera de productos) que el propietario del producto quiere obtener.

Scrum es precisamente una evolución de Agile Management. La metodología Scrum se basa en un conjunto de prácticas y roles muy definidos que deben estar involucrados durante el proceso de desarrollo de software.

En Scrum, el equipo se enfoca en construir software de calidad. El propietario de un proyecto Scrum se centra en definir cuáles son las características que debe tener el producto para construir (qué

construir, qué no y en qué orden) y superar cualquier obstáculo que pueda entorpecer la tarea del equipo de desarrollo.

El equipo Scrum consta de los siguientes roles:

- Scrum Master: La persona que lidera el equipo guiándolos a cumplir con las reglas y procesos de la metodología. El Scrum Master se encarga de mantener Scrum actualizado, brindando coaching, mentoring y capacitación a los equipos en caso de que lo necesite.
- Proyect Owner (PO): Es el representante de las partes interesadas y los clientes que utilizan el software. Trasladan la visión del proyecto al equipo, validan los beneficios en historias para incorporarlos al Product Backlog y los priorizan de forma periódica.
- Miembro del Equipo de Desarrollo: Grupo de profesionales con los conocimientos técnicos necesarios que desarrollan el proyecto de manera conjunta llevando a cabo las historias con las que se comprometen al inicio de cada sprint.

Cada uno de los eventos Scrum facilita la adaptación de algunos de los aspectos del proceso, el producto, el progreso o las relaciones.

- Sprint: Sprint es la unidad básica de trabajo de un equipo Scrum. Esta es la característica principal que marca la diferencia entre Scrum y otros modelos de desarrollo ágil.
- Planificación del Sprint: El objetivo de la Planificación del Sprint es definir qué se hará en el Sprint y cómo se hará. Esta reunión se realiza al inicio de cada Sprint y se define cómo abordará el proyecto proveniente de las etapas y plazos del Product Backlog. Cada Sprint se compone de diferentes funciones.
- Daily Scrum: El objetivo del Daily Scrum es evaluar el progreso y tendencia hasta el final del Sprint, sincronizando las actividades y creando un plan para las próximas 24 horas. Es una reunión breve que tiene lugar a diario durante el período de Sprint. El Scrum Master debe intentar solucionar los problemas u obstáculos que surjan.
- Revisión de Sprint: El objetivo de la revisión de Sprint es mostrar qué trabajo se ha completado con respecto a la acumulación de productos para futuras entregas. Se revisa el sprint terminado y ya debe haber un avance claro y tangible en el producto para presentar al cliente.
- Retrospectiva del Sprint: El equipo revisa los objetivos completados del Sprint finalizado, anota lo bueno y lo malo, para no repetir los errores nuevamente. Esta etapa sirve para implementar mejoras desde el punto de vista del proceso de desarrollo. El objetivo de la retrospectiva del Sprint es identificar posibles mejoras en el proceso y generar un plan para implementarlas en el próximo Sprint.

Los Artefactos SCRUM están diseñados para garantizar la transparencia de la información clave en la toma de decisiones.

- Backlog del producto (PB): El backlog del producto es una lista que recopila todo lo que el producto necesita para satisfacer a los clientes potenciales. Lo elabora el product owner y se priorizan las funciones según lo que es cada vez menos importante para el negocio. El objetivo es que el propietario del producto responda a la pregunta "Qué se debe hacer".
- Sprint Backlog (SB): Es un subconjunto de elementos del backlog del producto, que son seleccionados por el equipo para realizar durante el sprint en el que van a trabajar. El equipo

establece la duración de cada Sprint. Por lo general, la acumulación de sprints se muestra en tableros físicos llamados tablero Scrum, que hace que el proceso de desarrollo sea visible para todos los que ingresan al área de desarrollo.

- Incremento: El Incremento es la suma de todas las tareas, casos de uso, historias de usuario, backlogs de productos y cualquier elemento que se desarrolló durante el sprint y que estará disponible para el usuario final en forma de Software.

ISO-9126

La ISO, bajo la norma ISO-9126, ha establecido un estándar internacional para la evaluación de la calidad de productos de software el cual fue publicado en 1992 con el nombre de "Information technology –Software product evaluation: Quality characteristics and guidelines for their use", en el cual se establecen las características de calidad para productos de software [11].

El estándar ISO-9126 establece que cualquier componente de la calidad del software puede ser descrito en términos de una o más de seis características básicas, las cuales son: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad; cada una de las cuales se detalla a través de un conjunto de subcaracterísticas que permiten profundizar en la evaluación de la calidad de productos de software.

6. Cronograma

Ver los anexos 1, 2 y 3.

7. Referencias

[1] R. Noguez, "Confinamiento impulsa ventas online en México: crecen 81% en 2020", *Forbes México*, ene. 27, 2021. https://www.forbes.com.mx/negocios-ventas-online-mexico-2020/ (consultado mar. 25, 2021).

[2] "Porcentaje de compradores online México 2017-2025", *Statista*. https://es.statista.com/previsiones/703404/tasa-penetracion-comercio-electronico-mexico (consultado mar. 26, 2021).

[3] Ciberaula, "Por qué utilizar Linux?" https://linux.ciberaula.com/articulo/pq_linux/ (consultado mar. 26, 2021).

[4] F. Staff, "15 sitios que dominan al e-commerce en el mundo • Forbes México", *Forbes México*, abr. 02, 2013. https://www.forbes.com.mx/15-sitios-que-dominan-al-e-commerce-en-el-mundo/ (consultado mar. 26, 2021).

[5] "Amazon saldrá de la pandemia con crecimientos de más del 20%", *Expansión*, ago. 13, 2020. https://expansion.mx/tecnologia/2020/08/13/amazon-saldra-de-la-pandemia-con-crecimientos-del-20 (consultado mar. 26, 2021).

[6] F. Staff, "EBay triplica ganancias en 2020 con beneficios de 5,667 mdd", *Forbes México*, feb. 03, 2021. https://www.forbes.com.mx/tecnologia-ebay-triplica-ganancias-2020-5667-mdd/ (consultado mar. 26, 2021).

[7] "Estado de resultados de Costco", *Investing.com México*. https://mx.investing.com/equities/costco-whsl-corp-new-income-statement (consultado mar. 26, 2021).

[8] Reuters, "Walmart reporta mayores ganancias tras aumento de 79% en sus ventas en línea", *El Economista*. https://www.eleconomista.com.mx/mercados/Walmart-reporta-mayores-ganancias-tras-aumento-de-79-en-sus-ventas-en-linea--20201117-0028.html (consultado mar. 26, 2021).

[9] “Ingresos de MercadoLibre en México 2020”, *Statista*. https://es.statista.com/estadisticas/1114514/mexico-mercado-libre-ingresos-netos/ (consultado mar. 26, 2021).
[10] “What Is Scrum Methodology? & Scrum Project Management”, sep. 30, 2019. https://www.digite.com/agile/scrum-methodology/ (consultado abr. 15, 2021).
[11] “Calidad en la Industria del Software. La Norma ISO-9126”. Consultado: abr. 16, 2021. [En línea]. Disponible en: http://recursosbiblioteca.utp.edu.co/tesisd/textoyanexos/0053L864e_anexo.pdf.

8. Alumnos y directores

López Hernández Lissete.- Alumno de la carrera de Ing. en Sistemas Computacionales en ESCOM, Especialidad en Sistemas, Boleta: 2020630213, Tel.5567470858 , email: llopezh1600@alumno.ipn.mx

Firma:_____

Palacios Cabrera Alberto.- Alumno de la carrera de Ing. en Sistemas Computacionales en ESCOM, Especialidad en Sistemas, Boleta: 2020630360, Tel. 5537072600, email: apalaciosc1601@alumno.ipn.mx

Firma:_____

Pérez Priego Mario Daniel.- Alumno de la carrera de Ing. en Sistemas Computacionales en ESCOM, Especialidad en Sistemas, Boleta: 2020630349, Tel. 5560501122, email: mperezp1603@alumno.ipn.mx

Firma:_____

Jiménez Benítez José Alfredo.- Ing. en Electrónica con especialidad en Sistemas de la UAM, Maestro en Tecnología Avanzada con especialidad en Tiempo Real y Control de IPN, Dr. en Formación Docente de IPN, Profesor de ESCOM/IPN, Áreas de interés: Electrónica, Inteligencia emocional, Psicología, email: jose.jimenez.benitez@outlook.com

Firma:_____

Anexo 1

CRONOGRAMA Nombre del alumno(a): López Hernández Lissete_____ TT No.1
Título del TT: Implementación de un sistema de compras electrónica utilizando herramientas IPC.

Actividad	MAR	ABR	MAY	JUN
Investigación de herramientas IPC.				
Estructuración de la documentación.				
Análisis de requisitos.				
Planificación de iteraciones.				
Diagrama de flujo del módulo clientes.				
Ejecución de iteraciones.				
Sprint review (demostración de requisitos).				
Sprint Retrospective (Retrospectiva).				
Refinamiento de la lista de requisitos y cambios en el proyecto.				
Generación del reporte técnico.				
Video de presentación.				

Anexo 2

CRONOGRAMA Nombre del alumno(a): Palacios Cabrera Alberto_____ TT No.1
Título del TT: Implementación de un sistema de compras electrónica utilizando herramientas IPC.

Actividad	MAR	ABR	MAY	JUN
Investigación de herramientas IPC.				
Estructuración de la documentación.				
Análisis de requisitos.				
Diagrama de flujo del módulo proveedores.				
Ejecución de iteraciones.				
Sprint review (demostración de requisitos).				
Sprint Retrospective (Retrospectiva).				
Refinamiento de la lista de requisitos y cambios en el proyecto.				
Generación del reporte técnico.				
Video de presentación.				

Anexo 3

CRONOGRAMA Nombre del alumno(a): Pérez Priego Mario Daniel_____ TT No.1
Título del TT: Implementación de un sistema de compras electrónica utilizando herramientas IPC.

Actividad	MAR	ABR	MAY	JUN
Investigación de herramientas IPC.				
Estructuración de la documentación.				
Análisis de requisitos.				
Diagrama de flujo del módulo proceso de control.				
Ejecución de iteraciones.				
Sprint review (demostración de requisitos).				
Sprint Retrospective (Retrospectiva).				
Refinamiento de la lista de requisitos y cambios en el proyecto.				
Generación del reporte técnico.				
Video de presentación.				