UNS DCIC

SISTEMAS OPERATIVOS

#### PRIMER PROYECTO



**Comisión: 6**

**Santiago Maszong. LU: 125932**

29 de Noviembre de 2023 Sistemas Operativos - Comisión N°6

Índice

1. [Experimentación de Procesos y Threads con los Sistemas Operativos 3](#_TOC_250010)
   1. [Procesos, threads y Comunicación 3](#_TOC_250009)
      1. [Banco 3](#_TOC_250008)
      2. [Mini Shell 4](#_TOC_250007)
      3. [Secuencia 9](#_TOC_250006)
      4. [Reserva de Aulas 11](#_TOC_250005)
2. [Problemas 13](#_TOC_250004)
   1. [Lectura 13](#_TOC_250003)
   2. [Problemas Conceptuales 14](#_TOC_250002)
      1. [sistema de gestión de memoria 14](#_TOC_250001)
      2. [sistema de segmentación simple 15](#_TOC_250000)

# Experimentación de Procesos y Threads con los Sistemas Operativos

## Procesos, threads y Comunicación

### Banco

1. Se implementó el problema descrito utilizando threads y semáforos para la sincronización.

Con respecto a la sincronización se utilizaron dos semáforos para cada cola, uno indicando la cantidad actual de clientes en la cola y otro especificando la cantidad de lugares vacíos en la misma. Por ello dichos semáforos se inicializan en 0 y con el valor del tamaño de la cola correspondiente respectivamente.

También se utilizaron semáforos extras para simular la atención de los empleados a los clientes (y que los clientes se bloqueen esperando dicha atención) y para simular la presencia de clientes para atender con el objetivo de que los empleados no hagan espera ocupada mientras esperan clientes sino que se bloquean y son los mismos clientes los que los despiertan.

Además, se utilizaron mutex locks para los casos donde más de un thread puede hacer trywait sobre un mismo semáforo, como es el caso de los semáforos que cuentan la cantidad de clientes políticos y de clientes empresas en las funciones atencionEmpleadoComunes y atencionEmpleadoEmpresas o el semáforo que cuenta la cantidad de espacios vacíos en la cola de entrada, donde varios clientes van a intentar ingresar.

A continuación se describen las funciones implementadas y se indican la cantidad de threads asociados a cada una:

cliente: muchos threads para simular a muchos clientes (un thread es un cliente). Esta función no se bloquea ya que al llenarse la cola de entrada los nuevos clientes se retiran. Para ello se utiliza un trywait sobre el semáforo que cuenta la cantidad de espacios libres en la cola de entrada chequeando si hay lugares disponibles. Como son varios los hilos que hacen trywait de la cola de entrada es necesario hacer exclusividad con un mutex locks para evitar condiciones de carrera como se mencionó previamente.

Luego de entrar en la cola de entrada el cliente trata de ingresar a otra de las colas según su tipo, si hay lugar en dicha cola sale de la cola de entrada y entra a la cola en cuestión (todo esto con los semáforos empty y clientes de cada cola).

Una vez dentro de la cola correspondiente el cliente indica que está listo para ser atendido por algún empleado y se queda esperando por dicha atención. Una vez atendido se libera el lugar que ocupaba en la cola y luego de realizar sus trámites el cliente se retira.

atencionEmpleadoComunes: un thread, representando al empleado que atiende clientes comunes, que también atiende con prioridad a los clientes políticos. Esta función primero chequea si hay algún cliente político o común para atender haciendo trywait sobre el semáforo de clientes para atender y sino se bloquea esperando uno (de esta manera se evita la espera ocupada y los clientes son los que despiertan a los empleados). Si hay clientes para atender primero ve si alguno es político (con trywait) y lo atiende o sino va a buscar clientes comunes (con trywait). Al atender a los clientes primero se los toma de la cola correspondiente (liberando un lugar en la misma) y luego se les da atención

(desbloqueando al cliente). Finalmente vuelve a buscar clientes para atender o se duerme.

atencionEmpleadoEmpresas: dos thread, representando a los dos empleados que atienden clientes empresas, que también atiende con prioridad a los clientes políticos. Es similar a la función atencionEmpleadoComunes a diferencia de que como hay más de un hilo ejecutando la función y por ende haciendo trywait sobre la cola de empresas es necesario agregar un mutex lock para evitar condiciones de carrera sobre dicho trywait.

1. Se implementó el problema descrito utilizando procesos y pipes para la sincronización.

Se utilizaron las mismas funciones, y con respecto a los procesos se utilizaron subprocesos para que ejecuten cada una de las funciones en paralelo de la misma manera y con la misma distribución en que se utilizaron los threads.

En vez de semáforos se utilizaron pipes, dos pipes para cada cola, uno para los clientes actuales y otro para los lugares vacíos. Además de pipes para los semáforos de atención de los empleados y para los de presencia de clientes.

Se siguen utilizando los mutex lock para los read no bloqueantes sobre diferentes pipes, por la misma razón que se usaban para los trywait en la implementación con threads y semáforos.

1. Utilizar threads es más económico que usar procesos, también es un poco más simple de escribir y entender el código hecho con threads y semáforos.

### Mini Shell

Implementamos una minishell en un único archivo llamado minishell.c

Esta Mini Shell es un programa de línea de comandos simple que permite al usuario ejecutar un conjunto limitado de comandos relacionados con la gestión de directorios y archivos en un sistema.

* + - * ayuda():

Esta función muestra una lista de comandos disponibles y una breve descripción de cada uno cuando se llama a "help" proporcionando información sobre cómo utilizar la Mini Shell.

* + - * crearDirectorio(char dir[]):

Esta función crea un directorio con el nombre especificado (dir) en la ubicación actual. Utiliza la función mkdir con permisos de lectura, escritura y ejecución para el propietario, el grupo y otros usuarios. Muestra un mensaje de éxito o un mensaje de error si la creación del directorio no es exitosa.

* + - * eliminarDirectorio(char dir[]):

Esta función elimina un directorio con el nombre especificado (dir) en la ubicación actual. Utiliza la función rmdir para eliminar el directorio.

* + - * crearArchivo(char file[]):

Esta función crea un archivo con el nombre especificado (file) en la ubicación actual. Utiliza la función fopen para crear un archivo en modo escritura ("w+"). Muestra en caso de éxito el correspondiente mensaje o un mensaje de error si la creación del archivo no es exitosa. Finaliza cerrando el archivo después de crearlo.

* + - * listarContenidoDirectorio(char dir[]):

Esta función lista el contenido de un directorio con el nombre especificado (dir) en la ubicación actual. Utiliza la función opendir para abrir el directorio y readdir para obtener y mostrar el contenido. Cierra el directorio después de listar el contenido.

* + - * mostrarContenidoArchivo(char dir[]):

Esta función muestra el contenido de un archivo con el nombre especificado (dir) en la ubicación actual. Utiliza la función fopen para abrir el archivo en modo lectura ("r") y luego muestra su contenido. Finaliza cerrando el archivo después de leerlo.

* + - * modificarPermisos(char dir[], int p):

Esta función modifica los permisos de un archivo o directorio con el nombre especificado (dir). Utiliza la función chmod para cambiar los permisos al valor especificado por p.

Esto permite cambiar los permisos de lectura, escritura y ejecución del archivo o directorio.

* + - * limpiar():

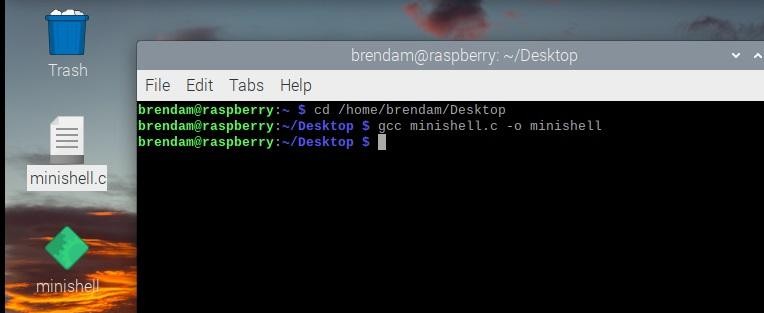
Esta función se utiliza para limpiar el búfer de entrada después de leer una línea de comando.

La función main muestra un mensaje de bienvenida y proporciona la opción "help" para obtener información sobre los comandos disponibles. Espera comandos del usuario. El usuario puede ingresar comandos como "mkdir", "rmdir", "mkfile", "ls", "showfile", "fileperm"

,”help” y "exit". Para cada comando, se realizan llamadas a las funciones correspondientes para llevar a cabo la operación deseada.

Esta Mini Shell ofrece una funcionalidad básica de administración de archivos y directorios a través de comandos de línea de comandos simples.

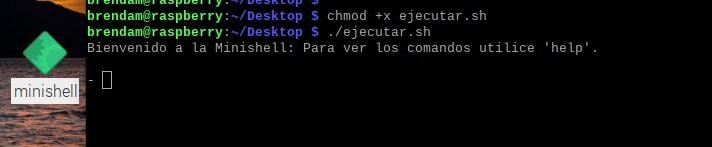
A continuación se muestra la prueba de la minishell correspondiente:

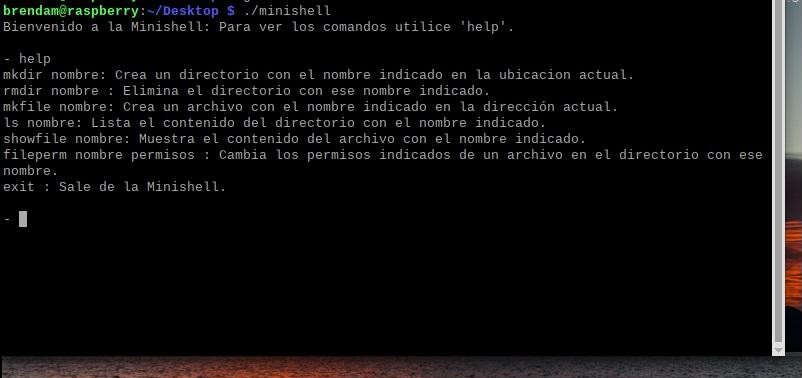
Primero nos ubicamos en la dirección donde se encuentra el archivo minishell.c. utilizamos el comando gcc minishell.c -o minishell para compilar el archivo

para ejecutarlo escribimos ./minishell y nos muestra el siguiente mensaje:



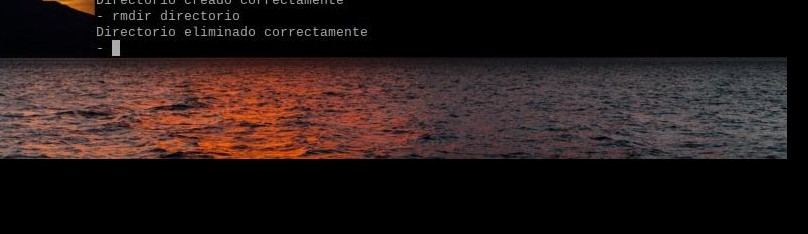
Luego creamos el el script ejecutar.sh, le dimos el permiso de ejecución y lo ejecutamos



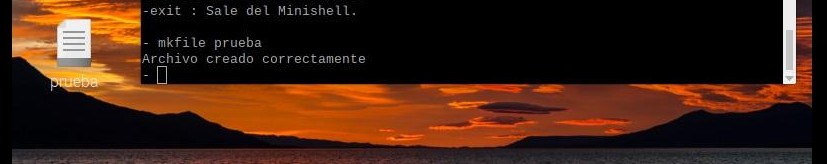
al escribir la palabra help podemos ver la ayuda de como ejecutar cada uno de los comandos

Probando mkdir directorio, vemos que se crea correctamente un directorio con el nombre que le indicamos “directorio”

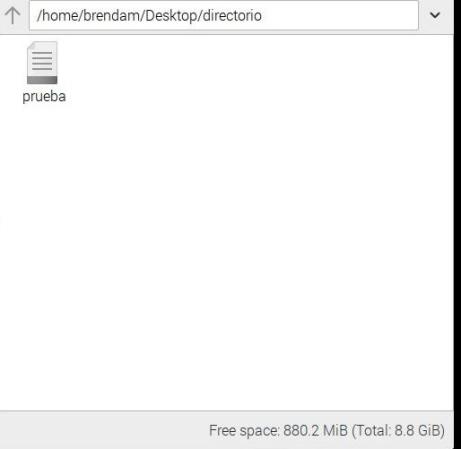
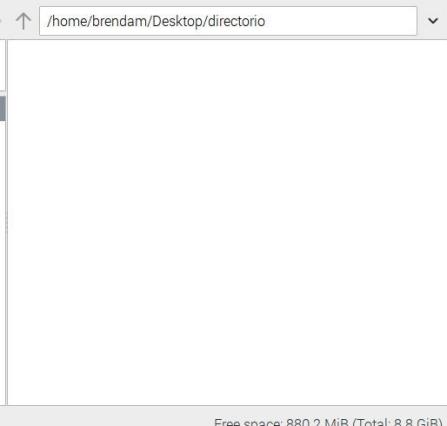


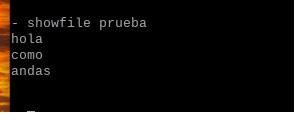
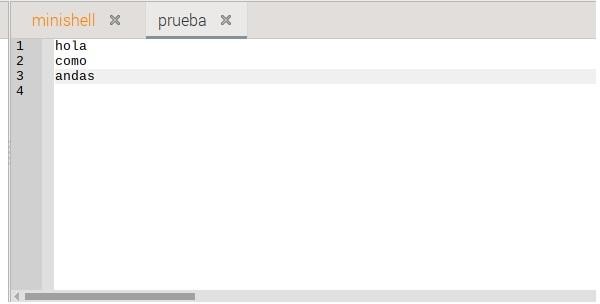
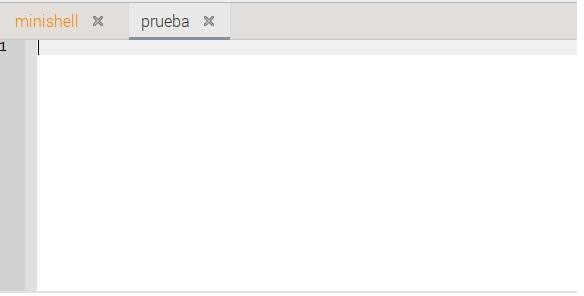
probando rmdir directorio, lo elimina correctamente.

el comando mkfile prueba me crea un archivo con el nombre prueba

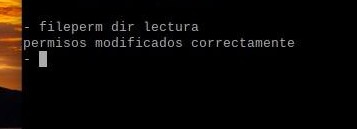


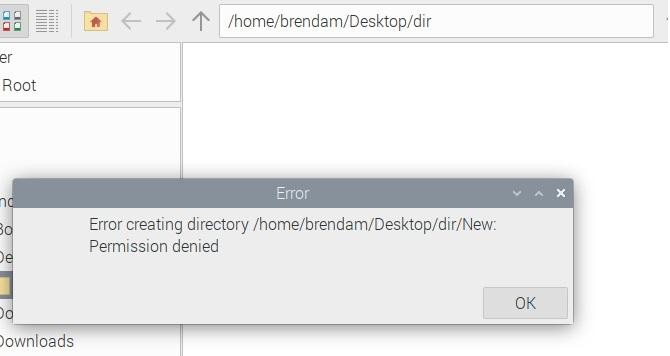
probamos el comando ls directorio, con el directorio vacío y con un archivo llamado prueba dentro del directorio

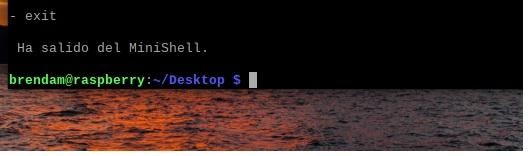


probamos el comando showfile para mostrar el contenido del archivo prueba, primero con el archivo vacío y luego con el archivo con contenido.

Probamos el comando fileperm para cambiarle los permisos al directorio, por lectura, y vemos que no se puede modificar luego de cambiarle el permiso





y por último al ejecutar el comando exit, salimos de la ejecución de la minishell y nos muestra el mensaje de salida

* 1. Sincronización

### Taller de Motos

1. Se utilizaron siete semáforos, uno para cada fase de construcción de una moto (construirNuevaMoto, rueda, chasis, motor, pintura, entregarMoto) y uno más para decidir cuál de los pintores pinta la moto (pintarMoto). Se definieron siete funciones, una para la funcionalidad de cada operario y una extra para manejar la entrega de la moto una vez terminada. Se usaron siete threads, uno para cada función representando a cada operario y a la funcionalidad de entrega de la moto para que se ejecuten concurrentemente. A continuación se explica cómo es la sincronización para obtener la secuencia de construcción de la moto.

Para obtener dicha secuencia de símbolos fue necesario inicializar el semáforo de construirNuevaMoto en 1 para desbloquear el hilo del operario de ruedas y poder comenzar con la construcción ya que todos los otros hilos están bloqueados.

Una vez desbloqueado el hilo de operario de ruedas el mismo “construye” en serie dos ruedas, aumentando el valor del semáforo ruedas en 2.

En este punto se desbloquea el hilo del operario del chasis, que necesitaba los 2 recursos del semáforo ruedas, y procede a armar el chasis aumentando el valor del semáforo chasis a 1 y desbloqueando al hilo del operario motor. Este, de manera análoga al anterior, coloca el motor y aumenta el valor del semáforo motor a 2 y el valor de pinturaMoto a 1.

Al aumentar el valor del semáforo del motor a 2 se desbloquean los dos hilos de los pintores (el que pinta de rojo y el que pinta de verde), para hacer la elección aleatoria de qué pintor realiza la pintura se utilizó la llamada no bloqueante try\_wait sobre el semáforo pinturaMoto en ambas funciones , sin mutex lock, de tal manera que se provoque una condición de carrera entre ambos para obtener el recurso. Finalmente ambos hilos se bloquean al esperar por el recurso del semáforo motor, sin el semáforo motor los hilos estarían haciendo busy waiting hasta que el try\_wait en alguno de los 2 sea exitoso.

Cuando uno de las dos funciones accede al recurso del semáforo pinturaMoto procede a pintar la moto del color correspondiente y aumenta el recurso de los semáforos “pintura” y “entregarMoto” en 1 indicando que ya se realizó la misma.

El hilo que gestiona la entrega de las motos se desbloquea cuando consigue 2 recursos del semáforo entregarMoto, y como dicho semáforo se inicializa en 1, la primera moto es entregada luego de ser pintada (donde se agrega el segundo recurso necesario para desbloquear la entrega) omitiendo así el agregado del equipamiento extra en la primera moto. Para la segunda moto, luego de pintarla los valores de los semáforos “pintura” y “entregarMoto” son 2 y 1 respectivamente, aquí no se desbloquea la entrega de la moto todavía, sino que se desbloquea el último operario (el encargado del equipamiento extra) el cual requiere 2 recursos del semáforo “pintura”. Dicho operario agrega el equipamiento extra y aumenta en 2 los recursos del semáforo “entregarMoto”, el cual tiene ahora un valor de 3, permitiendo la entrega de la moto (consume 2) y dejando 1 recurso para que se repita el ciclo indefinidamente. De esta forma se obtiene que 1 de cada 2 motos se le agrega equipamiento extra alternadamente.

Cabe destacar que cada uno de las funciones que ejecutan los hilos mencionados tienen un ciclo while que vuelve a ejecutar el mismo código indefinidamente, con la particularidad de que se bloquea el hilo ya que la primer instrucción dentro del ciclo while es una llamada wait por un recurso de un semáforo entonces se bloquea hasta obtener dicho recurso como se explicó previamente, sin hacer busy waiting.

### Santa Claus

Primero se definieron constantes arbitrarias para la cantidad de renos (9) y para la cantidad de elfos (6).

Se utilizaron siete semáforos, uno para cada fase de construcción de una moto (construirNuevaMoto, rueda, chasis, motor, pintura, entregarMoto), se definieron siete funciones, una para la funcionalidad de cada operario y una extra para manejar la entrega de la moto una vez terminada. Se usaron siete threads, uno para cada función representando a cada operario y a la funcionalidad de entrega de la moto para que se ejecuten concurrentemente. A continuación se explica cómo es la sincronización para resolver el problema.

# Problemas

## Lectura

Realizamos una presentación de Linux embebido en donde exponemos las características, ventajas y usos más importantes que este sistema presenta.

Un sistema embebido es un conjunto de hardware y software de computadora basado en un microcontrolador o microprocesador, controlado por un sistema operativo en tiempo real o RTOS, memoria limitada, y que puede variar tanto en tamaño como en complejidad. Linux embebido es un tipo de sistema operativo/núcleo Linux que fue diseñado para ser instalado y utilizado en dispositivos o sistemas embebidos.

A diferencia de las distribuciones de Linux de propósito general utilizadas en computadoras de escritorio y servidores, Linux embebido se ajusta a las restricciones de recursos de los dispositivos embebidos, como limitaciones de memoria, capacidad de almacenamiento y potencia de procesamiento.

Linux embebido es el sistema operativo por excelencia utilizado en dispositivos o sistemas embebidos.Se ha convertido en la elección preferida para diseñadores, desarrolladores y organizaciones que buscan crear sistemas embebidos eficientes y de alto rendimiento.

Esto se debe a la amplia cantidad de ventajas que posee Linux embebido, entre ellas podemos encontrar:

* Código Abierto: Como sistema de código abierto, Linux embebido proporciona acceso completo al código fuente. Esto significa que los desarrolladores pueden inspeccionar, modificar y extender el sistema. Lo que permite una mayor transparencia y control.
* Costos bajos: Dado que Linux embebido es de código abierto, no hay costos de licencia asociados. Esto puede resultar en un ahorro significativo en comparación con sistemas operativos propietarios.
* Flexibilidad y personalización :Linux embebido se puede adaptar a las necesidades específicas de un dispositivo o sistema, lo que lo hace muy versátil.

Como sistema operativo de código abierto,es altamente personalizable, lo que lo hace adecuado para una amplia variedad de aplicaciones.

Podemos instalar un sistema muy ligero e ir añadiendo todo lo necesario posteriormente o según lo vayamos necesitando.

La capacidad de leer, modificar y redistribuir el código fuente, junto con un enfoque de bloques de construcción altamente modular, brinda flexibilidad en el diseño y desarrollo de sistemas embebidos.

* Múltiples proveedores de software, desarrollo y soporte: La comunidad de Linux es grande y diversa, lo que significa que hay una amplia gama de proveedores, desarrolladores y recursos de soporte disponibles.

Al existir numerosos proveedores de software y soporte para Linux embebido, brinda a los desarrolladores una amplia gama de recursos.

La comunidad de desarrolladores de Linux es vasta y activa. Esto garantiza un flujo constante de actualizaciones, parches de seguridad y soporte técnico. También significa que se pueden encontrar soluciones y recursos fácilmente en línea.

* Velocidad de desarrollo: La disponibilidad de componentes listos para usar, como una pila TCP/IP, el kernel de Linux y bibliotecas, acelera el desarrollo de productos. Al ser un sistema operativo de código abierto , Linux embebido permite una alta velocidad de desarrollo de productos.Proporciona piezas de código listas para usar que los desarrolladores pueden incorporar en sus aplicaciones embebidas, lo que reduce la complejidad y el tiempo de desarrollo.

Linux embebido debido a todas las cualidades presentadas, es utilizado en una amplia gama de dispositivos y aplicaciones entre los cuales podemos mencionar : Telecomunicaciones y Redes

Teléfonos móviles y tablets Dispositivos médicos

Electrodomésticos y dispositivos domésticos inteligentes Automóviles Inteligentes

Aeroespacial y Defensa

Incluso Microsoft Windows cuenta con componentes de Linux como parte del Subsistema de Windows para Linux o Windows Subsystem for Linux (WSL).

El sistema operativo Android es un tipo de Linux embebido, personalizado para ser utilizado en teléfonos inteligentes . En esencia, Android ejecuta el mismo tipo de kernel de Linux que el de los sistemas embebidos y comparte los mismos controladores, la gestión de la memoria y las capacidades de red. Pero también incluye algunas modificaciones del núcleo que lo hacen ideal para los sistemas móviles.

Por todo lo investigado vemos que Linux embebido es esencial en la actualidad debido a su versatilidad, eficiencia y confiabilidad. Su capacidad para adaptarse a una amplia gama de aplicaciones y su comunidad activa de desarrolladores lo convierten en un pilar en el mundo de los dispositivos y sistemas embebidos.

## Problemas Conceptuales

### 2.2.1 sistema de gestión de memoria

### 2.2.2 sistema de gestión de memoria

Las direcciones lógicas y físicas de 16 bits se componen por los bits de número de página/marco (más significativos) y los bits de desplazamiento dentro de la página (menos significativos). Ya que las páginas tienen un tamaño de 4096 bytes, equivalente a 2^12 bytes, y como el direccionamiento es al byte entonces son necesarios 12 bits para direccionar el desplazamiento dentro de cada página. Entonces de los 16 bits totales de la dirección los 12 menos significativos son los de desplazamiento y los 4 más significativos son los de numero de página/marco. Esto se corrobora al ver que la tabla de páginas tiene 16 entradas o páginas y que dichos bits se utilizan para navegar en la tabla.

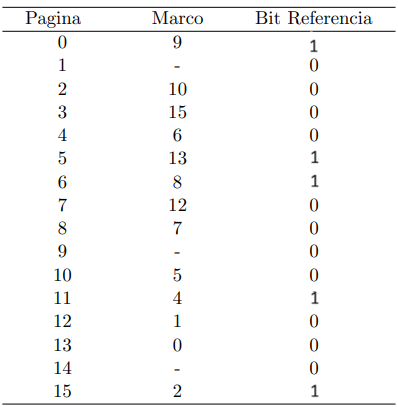
a) Para obtener las direcciones físicas equivalentes debo tomar los primeros 4 bits de la dirección lógica ya que los mismos indican el número de página.

Para el caso de la dirección 0x621C, que en binario es 0110 0010 0001 1100, los 4 bits más significativos representan al número 6. Localizo dicha página en la tabla y obtengo el número de marco que es 8. Este número de marco se lo concateno a los 12 bits restantes de la dirección lógica (bits de desplazamiento) para obtener la dirección física final de 0x821C.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dirección**  **Virtual** | **Número de**  **Página** | **Marco** | **Desplazamiento** | **Dirección Física** |
| 0x621C | 6 | 8 | 0x21C | 0x821C |
| 0xF0A3 | 15 (F16) | 2 | 0x0A3 | 0x20A3 |
| 0xBC1A | 11 (B16) | 4 | 0xC1A | 0x4C1A |
| 0x5BAA | 5 | 13 (D16) | 0xBAA | 0xDBAA |
| 0x0BA1 | 0 | 9 | 0xBA1 | 0x9BA1 |

Así obtengo todas las direcciones físicas y los bits de referencia para dichas páginas se setea en 1, indicando que se ha accedido recientemente a los marcos de la memoria física indicados en la tabla.

La tabla queda así:



b) Una dirección lógica que produzca un fallo de página es, por ejemplo la dirección 0x1A23 que representa a la entrada 1 en la tabla de páginas y cuyo referenciamiento a un marco en memoria principal no está definido ya que muestra un guión ( - ).

c) En este caso, si hay lugar en memoria principal se puede cargar dicha página en un marco en la misma y referenciar su dirección en un hueco libre de la tabla de páginas. Sino, como el reemplazo es local, el proceso que causa el fallo de página deberá reemplazar una página de un marco perteneciente a su propio conjunto de marcos asignados. Y como el algoritmo de reemplazo es LRU la página a reemplazar debe ser una no recientemente usada. Dicha página se puede obtener observando el bit de referencia, y reemplazarla por la nueva página que necesita cargarse en memoria.

En definitiva el marco de página seleccionado para realizar el reemplazo va a ser alguno del conjunto de marcos asignados al proceso en cuestión y además que tenga el bit de referencia en 0.

1. 0, 228: indica que es en el segmento 0 y con un desplazamiento en dicho segmento de 228 direcciones.

Como el desplazamiento (228) es menor al largo del segmento (346) entonces no se produce un fallo de segmento.

La dirección física es 1058 y se obtiene sumando a la dirección inicial del segmento (830) con el desplazamiento (228).

1. 2, 648: indica que es en el segmento 2 y con un desplazamiento en dicho segmento de 648 direcciones.

Como el desplazamiento (648) es mayor al largo del segmento (408) entonces se produce un fallo de segmento.

1. 3, 776: indica que es en el segmento 3 y con un desplazamiento en dicho segmento de 776 direcciones.

Como el desplazamiento (776) es menor al largo del segmento (812) entonces no se produce un fallo de segmento.

La dirección física es 1546 y se obtiene sumando a la dirección inicial del segmento (770) con el desplazamiento (776).

1. 1, 98: indica que es en el segmento 1 y con un desplazamiento en dicho segmento de 98 direcciones.

Como el desplazamiento (98) es menor al largo del segmento (110) entonces no se produce un fallo de segmento.

La dirección física es 746 y se obtiene sumando a la dirección inicial del segmento (648) con el desplazamiento (98).

1. 1, 240: indica que es en el segmento 0 y con un desplazamiento en dicho segmento de 240 direcciones.

Como el desplazamiento (240) es mayor al largo del segmento (110) entonces se produce un fallo de segmento.