INSTITUTO POLITÉCNICO NACIONAL.

ESCUELA SUPERIOR DE CÓMPUTO.

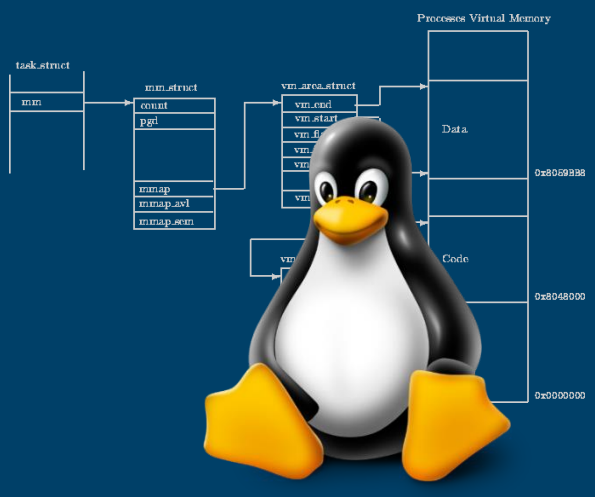
SISTEMAS OPERATIVOS.

PRÁCTICA 6

Simulador de memoria virtual.

Integrantes del equipo:

- Chavarría Vázquez Luis Enrique.
- Juárez Espinoza Ulises.
- Machorro Vences Ricardo Alberto.
- Pastrana Torres Víctor Norberto.



2CM6 Unidad 3

Índice de contenido.

Índice de figuras.

Índice de tablas

Glosario de términos.

Gestión de la memoria.

La gestión de la memoria es una actividad importante que se realiza de forma eficaz en el kernel. La administración de la memoria es el proceso de administrar la memoria de la computadora. es decir, esto incluye la asignación de memoria a varios programas en ejecución para mantener estable el rendimiento del sistema.

La memoria debe asignarse dinámicamente de acuerdo con el requisito y debe liberarse cuando ya no se use para que pueda reasignarse cuando sea necesario. La administración de la memoria depende de la configuración efectiva en el hardware, el sistema operativo y los programas o aplicaciones. En el hardware, la gestión de la memoria implica dispositivos físicos que almacenan los datos. La memoria de acceso aleatorio es un ejemplo de esto. Esto también incluye cachés de memoria y unidades de estado sólido. [1]

En el caso del sistema operativo, la gestión de la memoria implica la asignación de bloques de memoria específicos a programas individuales a medida que cambia la demanda del usuario. A nivel de aplicación, la gestión de la memoria asegura que la memoria demandada por los objetos y las estructuras de datos de cada programa en ejecución esté siempre disponible.

Las computadoras modernas administran la memoria en dos niveles; a nivel de sistema y a nivel de aplicación. La gestión de memoria a nivel de aplicación se clasifica como gestión de memoria automática o manual. En este artículo veremos la gestión de memoria basada en memoria virtual y paginación por demanda. [2]

Memoria virtual.

La memoria virtual es una técnica de gestión de memoria que se puede implementar tanto con hardware como con software. Como su nombre lo indica, agrega memoria virtual a la memoria disponible, por lo que su sistema parecerá tener más memoria de la que realmente existe. La memoria virtual es una capa de direcciones de memoria (direcciones virtuales) que se asignan a direcciones físicas. [2]

Las direcciones de memoria utilizadas por un programa son las direcciones virtuales. Estos espacios de direcciones virtuales y la asignación de memoria real a la memoria virtual son administrados por el sistema operativo. El funcionamiento de la memoria virtual es el siguiente. El sistema informático tiene una cantidad limitada de RAM estática.

Cuando se ejecuta un programa, se carga una instancia del programa en la RAM. Este es el proceso de asignar la memoria para que se ejecuten las instrucciones. Cuando el programa demanda más RAM de la disponible, será asignado a la memoria virtual. Esto evita que el programa carezca de la RAM necesaria para ejecutarse. Esta memoria virtual es en realidad la memoria del disco duro y luego se asigna a la memoria física.

Paginación en demanda.

La paginación por demanda es un tipo de intercambio realizado en sistemas de memoria virtual. En la paginación por demanda, los datos no se copian del disco a la RAM hasta que son necesarios o solicitados por algún programa. Los datos no se copiarán cuando los datos ya estén disponibles en la memoria. [3]

De lo contrario, esto se denomina evaluación perezosa porque solo las páginas de memoria solicitadas se intercambian del almacenamiento secundario (espacio en disco) a la memoria principal. En contraste, durante el intercambio puro, toda la memoria de un proceso se intercambia del almacenamiento secundario a la memoria principal durante el inicio del proceso.

Paginación anticipada.

La paginación anticipada es otro tipo de intercambio en los sistemas de memoria virtual. Aquí, el sistema operativo intenta anticipar los datos que se necesitarán a continuación y los copia en la RAM antes de que realmente se requieran. [4]

Trabajo de búsqueda en demanda.

El trabajo de paginación de demanda se basa en una implementación de tabla de páginas. La tabla de páginas asigna la memoria lógica a la memoria física. La tabla de páginas utiliza un operador bit a bit para marcar si una página es válida o no. Una página válida es aquella que reside actualmente en la memoria principal. Una página no válida se puede definir como la que actualmente reside en la memoria secundaria. [5]

Paginación.

Es una técnica de manejo de memoria, en la cual el espacio de memoria se divide en secciones físicas de igual tamaño, denominadas marcos de página. Los programas se dividen en unidades lógicas, denominadas páginas, que tienen el mismo tamaño que los marcos de páginas. De esta forma, se puede cargar una página de información en cualquier marco de página. [6]

Tabla de páginas.

El número de página virtual se utiliza como índice en la tabla de páginas para buscar la entrada para esa página virtual. En la entrada en la tabla de páginas, se encuentra el número de marco de página (si lo hay). El número del marco de página se adjunta al extremo de mayor orden del desplazamiento, reemplazando el número de página virtual, para formar una dirección física que se pueda enviar a la memoria. [7]

Segmentación.

En este caso, el programa y sus datos asociados se dividen en un conjunto de segmentos. No es necesario que todos los segmentos de todos los programas tengan la misma longitud, aunque existe una longitud máxima de segmento. Como en la paginación, una dirección lógica segmentada consta de dos partes, en este caso un número de segmento y desplazamiento. [8]

Segmentación paginada.

Como su propio nombre lo indica, la segmentación paginada intenta aumentar lo mejor de los dos esquemas. La segmentación proporciona soporte directo a las regiones del proceso y la paginación permite un mejor aprovechamiento de la memoria y una base para construir un esquema de memoria virtual. [9]

Página.

Un bloque contiguo de longitud fija de memoria virtual que reside en el disco. [9]

Memoria física.

la memoria de acceso aleatorio (RAM) de la computadora, generalmente contenida en tarjetas DIMM conectadas a la placa base de la computadora. [1]

Memoria virtual.

La memoria virtual es una parte de un HDD o SSD que se reserva para emular la RAM. La MMU sirve memoria virtual desde el disco a la CPU para reducir la carga de trabajo en la memoria física. [6]

Dirección virtual.

La CPU genera una dirección virtual para cada proceso activo. La MMU asigna la dirección virtual a una ubicación física en la RAM y pasa la dirección al bus. Un espacio de direcciones virtuales es el rango de direcciones virtuales bajo el control de la CPU. [7]

Dirección física.

La dirección física es una ubicación en la RAM. El espacio de direcciones físicas es el conjunto de todas las direcciones físicas correspondientes a las direcciones virtuales de la CPU. Un espacio de direcciones físicas es el rango de direcciones físicas bajo el control de MMU. [5]

Contenido (Investigación)

Memoria virtual.

Generalidades.

La memoria virtual es una técnica de administración de memoria en la que la memoria secundaria se puede utilizar como si fuera parte de la memoria principal. La memoria virtual es una técnica muy común utilizada en los sistemas operativos (SO) de las computadoras.

La memoria virtual utiliza hardware y software para permitir que una computadora compense la escasez de memoria física mediante la transferencia temporal de datos desde la memoria de acceso aleatorio (RAM) al almacenamiento en disco. En esencia, la memoria virtual permite que una computadora trate la memoria secundaria como si fuera la memoria principal.

Hoy en día, la mayoría de las PC vienen con alrededor de 4 GB de RAM. Sin embargo, a veces esto no es suficiente para ejecutar todos los programas que un usuario podría querer usar a la vez. Aquí es donde entra en juego la memoria virtual. La memoria virtual se puede usar para intercambiar datos que no se han usado recientemente y moverlos a un dispositivo de almacenamiento como un disco duro o una unidad de estado sólido (SDD). Esto liberará más espacio en la RAM. [10]

La memoria virtual es importante para mejorar el rendimiento del sistema , la multitarea, el uso de programas grandes y la flexibilidad. Sin embargo, los usuarios no deben confiar demasiado en la memoria virtual, porque el uso de datos virtuales es considerablemente más lento que el uso de RAM. Si el sistema operativo tiene que intercambiar datos entre la memoria virtual y la RAM con demasiada frecuencia, puede hacer que la computadora se sienta muy lenta; a esto se le llama thrashing .

La memoria virtual se desarrolló en un momento en que la memoria física, también denominada RAM, era cara. Las computadoras tienen una cantidad limitada de RAM, por lo que la memoria puede agotarse, especialmente cuando se ejecutan varios programas al mismo tiempo. Un sistema que usa memoria virtual usa una sección del disco duro para emular la RAM. Con la memoria virtual, un sistema puede cargar programas más grandes o múltiples programas ejecutándose al mismo tiempo, permitiendo que cada uno funcione como si tuviera memoria infinita y sin tener que comprar más RAM. [6]

Funcionamiento.

La memoria virtual utiliza tanto hardware como software para funcionar. Cuando una aplicación está en uso, los datos de ese programa se almacenan en una dirección física usando RAM. Más específicamente, la memoria virtual asignará esa dirección a la RAM mediante una unidad de gestión de memoria (MMU). El sistema operativo creará y administrará asignaciones de memoria utilizando tablas de páginas y otras estructuras de datos. La MMU, que actúa como un hardware de traducción de direcciones, traducirá automáticamente las direcciones.

Si en algún momento posterior se necesita espacio RAM para algo más urgente, los datos pueden intercambiarse de la RAM a la memoria virtual. El administrador de memoria de la computadora se encarga de realizar un seguimiento de los cambios entre la memoria física y la virtual. Si esos datos

se necesitan nuevamente, se puede usar un cambio de contexto para reanudar la ejecución nuevamente.

Al copiar la memoria virtual en la memoria física, el sistema operativo divide la memoria en archivos de paginación o intercambia archivos con un número fijo de direcciones. Cada página se almacena en un disco y, cuando se necesita la página, el sistema operativo la copia del disco a la memoria principal y traduce las direcciones virtuales en direcciones reales. [11]

Sin embargo, el proceso de intercambio de memoria virtual a física es bastante lento. Esto significa que el uso de memoria virtual generalmente provoca una reducción notable en el rendimiento.

Variedades.

Hay dos formas de gestionar la memoria virtual: paginada y segmentada .

La paginación divide la memoria en secciones o archivos de paginación, generalmente de aproximadamente 4 KB de tamaño. Cuando una computadora consume su RAM, las páginas que no están en uso se transfieren a la sección del disco duro designada para la memoria virtual mediante un archivo de intercambio. Un archivo de intercambio es un espacio reservado en el disco duro como las extensiones de memoria virtual de la RAM de la computadora.

Cuando se necesita el archivo de intercambio, se envía de vuelta a la RAM mediante un proceso llamado intercambio de página. Este sistema garantiza que el sistema operativo y las aplicaciones de la computadora no se queden sin memoria real.

El proceso de paginación incluye el uso de tablas de páginas, que traducen las direcciones virtuales que el sistema operativo y las aplicaciones usan en las direcciones físicas que usa la MMU. Las entradas en la tabla de páginas indican si la página está en la memoria real.

Si el sistema operativo o un programa no encuentra lo que necesita en la RAM, entonces la MMU responde a la referencia de memoria que falta con una excepción de falla de página para que el sistema operativo vuelva a mover la página a la memoria cuando sea necesario. Una vez que la página está en la RAM, su dirección virtual aparece en la tabla de páginas.

La segmentación también se usa para administrar la memoria virtual. Este enfoque divide la memoria virtual en segmentos de diferentes longitudes. Los segmentos que no se utilizan en la memoria se pueden mover al espacio de memoria virtual en el disco duro.

La información o los procesos segmentados se rastrean en una tabla de segmentos, que muestra si un segmento está presente en la memoria, si se ha modificado y cuál es su dirección física. Además, los sistemas de archivos en segmentación solo se componen de una lista de segmentos mapeados en el espacio de direcciones potencial de un proceso. [12]

La segmentación y la paginación difieren como modelo de memoria en términos de cómo se divide la memoria; sin embargo, también se puede combinar. Algunos sistemas de memoria virtual combinan segmentación y paginación. En este caso, la memoria se divide en cuadros o páginas. Los segmentos ocupan varias páginas y la dirección virtual incluye tanto el número de segmento como el número de página.

En la siguiente tabla (tabla 1) presentamos una comparativa, la cual hemos elaborado con el fino propósito de entender las diferencias de una manera mucho más precisa.

Tabla 1 Diferencia entre la administración de memoria por paginación y por segmentación.

Diferencia entre la administración de memoria por paginación y por segmentación.	
Paginación.	**Segmentación.**
Las particiones son de tamaño fijo.	Las particiones son de tamaño variable.
A cada zona de memoria se le llama marco de página y el proceso se divide en trozos iguales llamadas páginas.	A cada zona de memoria se le llama segmento.
Es posible comenzar a ejecutar un programa, cargando solo una parte del mismo en memoria, y el resto cargara bajo la solicitud. se cargará bajo la solicitud.	El programador puede conocer las unidades lógicas de su programa, dándoles un tratamiento particular.
No es necesario que las paginas estén contiguas en memoria, por lo que no se necesitan procesos de compactación cuando existen marcos de páginas libres dispersos en la memoria.	Es posible compilar módulos separados como segmentos el enlace entre los segmentos puede suponer hasta tanto se haga una referencia entre segmentos.
Esta técnica se inventó para obtener un gran espacio de direcciones lineales sin tener que comprar más memoria física.	Esta técnica se inventó para permitir a los programas y datos dividirse en espacios de direcciones lógicamente independientes, ayudando a la compartición y la protección.

Administración.

Los sistemas operativos tienen configuraciones predeterminadas que determinan la cantidad de espacio en el disco duro para asignar a la memoria virtual. Esa configuración funcionará para la mayoría de las aplicaciones y procesos, pero puede haber ocasiones en las que sea necesario restablecer manualmente la cantidad de espacio en el disco duro asignado a la memoria virtual, como con aplicaciones que dependen de tiempos de respuesta rápidos o cuando la computadora tiene múltiples HDD . [1]

Al restablecer manualmente la memoria virtual, se debe especificar la cantidad mínima y máxima de espacio en el disco duro que se utilizará para la memoria virtual. La asignación de muy poco espacio en el disco duro para la memoria virtual puede hacer que la computadora se quede sin RAM. Si un sistema necesita continuamente más espacio de memoria virtual, puede ser conveniente considerar

agregar RAM. Los sistemas operativos comunes generalmente recomiendan a los usuarios que no aumenten la memoria virtual más allá de 1,5 veces la cantidad de RAM.

Sin embargo, administrar la memoria virtual puede ser una experiencia diferente en diferentes tipos de sistemas operativos. Y los profesionales de TI deben comprender los conceptos básicos cuando se trata de administrar la memoria física, la memoria virtual y las direcciones virtuales.

Limitantes. [10]

- El uso de la memoria virtual tiene sus ventajas y desventajas, especialmente con la velocidad. Por lo general, es mejor tener la mayor cantidad de memoria física posible, por lo que los programas funcionan directamente desde la RAM o la memoria física.

- El uso de la memoria virtual ralentiza una computadora porque los datos deben mapearse entre la memoria virtual y la física, lo que requiere soporte de hardware adicional para las traducciones de direcciones.

- El tamaño del almacenamiento virtual está limitado por la cantidad de almacenamiento secundario, así como por el esquema de direccionamiento con el sistema informático.

- Puede que sea necesario un tiempo para cambiar entre aplicaciones que utilizan memoria virtual.

Ventajas de uso. [10]

- Su capacidad para manejar el doble de direcciones que la memoria principal.

- Libera a las aplicaciones de la gestión de la memoria compartida y evita que los usuarios tengan que agregar módulos de memoria cuando se agota el espacio de RAM.

- Mayor seguridad debido al aislamiento de la memoria.

- Se pueden ejecutar varias aplicaciones más grandes simultáneamente.

- La asignación de memoria es relativamente barata.

- Uso efectivo de la CPU.

Paginación y segmentación.

La paginación es una función de administración de la memoria de la computadora que presenta ubicaciones de almacenamiento en la CPU de la computadora como memoria adicional, llamada memoria virtual. Cada dato necesita una dirección de almacenamiento.

La segmentación es un proceso virtual que crea espacios de direcciones de tamaño variable en el almacenamiento de la computadora para datos relacionados, llamados segmentos. Este proceso acelera la recuperación.

La administración de la memoria de la computadora es una función básica del sistema operativo; tanto la paginación como la segmentación son funciones básicas del sistema operativo. Ningún

sistema puede depender de manera eficiente únicamente de la RAM limitada. Entonces, la unidad de administración de memoria (MMU) de la computadora usa el disco de almacenamiento, HDD o SSD , como memoria virtual para complementar la RAM. [3]

Paginación a detalle.

Ahora bien, la función de administración de memoria llamada paginación especifica las ubicaciones de almacenamiento en la CPU como memoria adicional, llamada memoria virtual. La CPU no puede acceder directamente al disco de almacenamiento, por lo que la MMU emula la memoria asignando páginas a marcos que están en la RAM.

En la ilustración 1 que aparece en la parte de abajo podemos ver con más detalle un diagrama que clarifica mucho más el funcionamiento de la paginación.

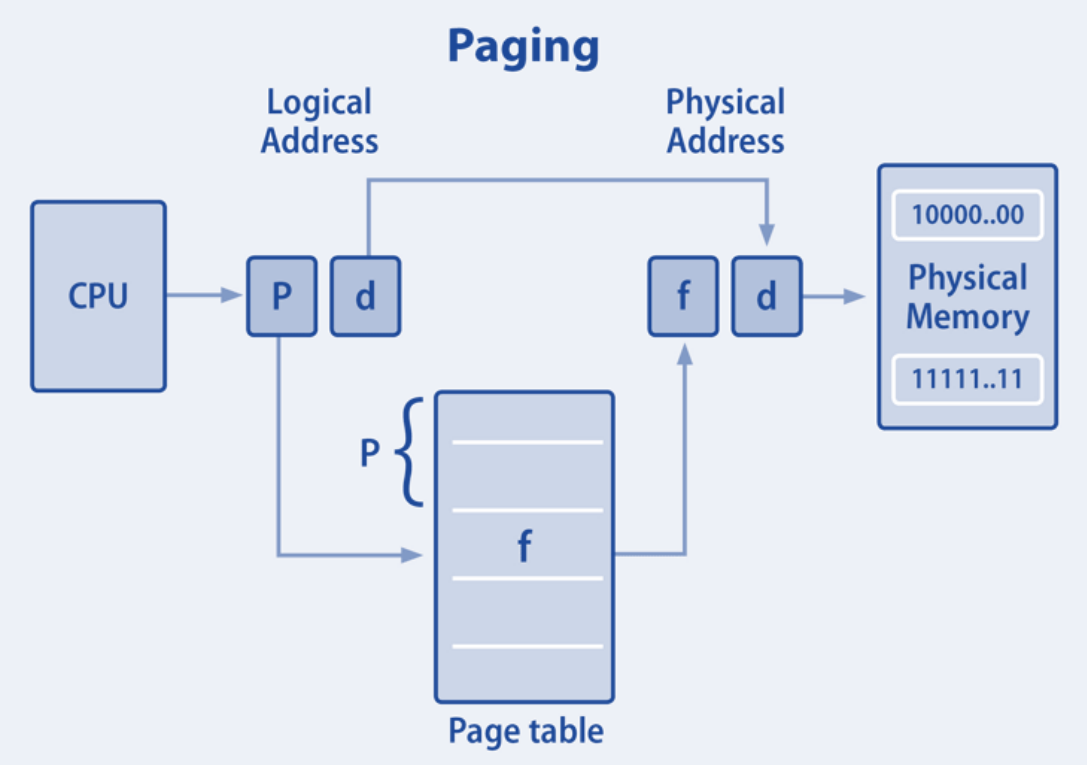


Ilustración 1 Diagrama de paginación.

Cuando asignamos una dirección a un dato mediante una "tabla de páginas" entre la CPU y la memoria física de la computadora, la MMU de una computadora permite que el sistema recupere esos datos cuando sea necesario, esto lo apreciamos con más detalle en la ilustración 1.

Una tabla de páginas almacena la definición de cada página. Cuando un proceso activo solicita datos, la MMU recupera las páginas correspondientes en marcos ubicados en la memoria física para un procesamiento más rápido. El proceso se llama paginación.

La MMU utiliza tablas de páginas para traducir direcciones virtuales a físicas. Cada entrada de la tabla indica dónde se ubica una página: en RAM o en disco como memoria virtual. Las tablas pueden tener una tabla de página de uno o varios niveles, como tablas diferentes para aplicaciones y segmentos.

Sin embargo, las búsquedas constantes en la tabla pueden ralentizar la MMU. Un caché de memoria llamado Translation Lookaside Buffer (TLB) almacena traducciones recientes de direcciones virtuales a físicas para una rápida recuperación. Muchos sistemas tienen múltiples TLB, que pueden residir en diferentes ubicaciones, incluso entre la CPU y la RAM, o entre varios niveles de tabla de páginas.

Hay disponibles diferentes tamaños de marco para conjuntos de datos con páginas más grandes o más pequeñas y marcos de tamaño coincidente. De 4 KB a 2 MB son tamaños comunes, y los marcos de tamaño GB están disponibles en servidores de alto rendimiento.

Un problema llamado fragmentación oculta solía ser un problema en implementaciones de Windows más antiguas. A diferencia de la grave fragmentación externa de la segmentación, la fragmentación interna se produce si cada fotograma no tiene el tamaño exacto del tamaño de la página. Sin embargo, esto no es un problema en el sistema operativo Windows moderno. [10]

Segmentación a detalle.

El proceso conocido como segmentación es un proceso virtual que crea espacios de direcciones de varios tamaños en un sistema informático, llamados segmentos. Cada segmento es un espacio de direcciones virtual diferente que se corresponde directamente con los objetos de proceso.

Cuando se ejecuta un proceso, la segmentación asigna datos relacionados en segmentos para un procesamiento más rápido. La función de segmentación mantiene una tabla de segmentos que incluye direcciones físicas del segmento, tamaño y otros datos.

Ahora en la parte de abajo podemos apreciar una descripción gráfica de la segmentación en la ilustración 2.

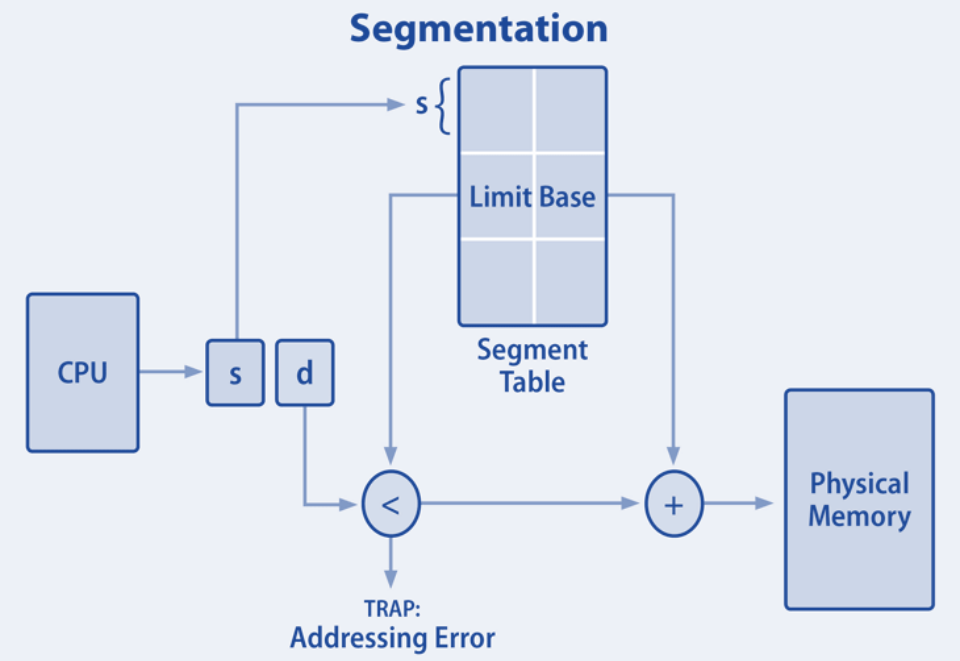


Ilustración 2 Diagrama de segmentación.

La segmentación acelera la recuperación de información de una computadora al asignar datos relacionados en una "tabla de segmentos" entre la CPU y la memoria física. Lo cual se puede apreciar en la ilustración de arriba (ilustración 2).

Cada segmento almacena la función principal del proceso, las estructuras de datos y las utilidades. La CPU mantiene una tabla de mapa de segmentos para cada proceso y bloques de memoria, junto con la identificación de los segmentos y las ubicaciones de la memoria. [13]

La CPU genera direcciones virtuales para ejecutar procesos. La segmentación traduce las direcciones virtuales generadas por la CPU en direcciones físicas que hacen referencia a una ubicación de memoria física única. La traducción no es estrictamente uno a uno: diferentes direcciones virtuales se pueden asignar a la misma dirección física.

Segmentación paginada

Algunas computadoras modernas usan una función llamada paginación segmentada. La memoria principal se divide en segmentos de tamaño variable, que luego se dividen en páginas más pequeñas de tamaño fijo en el disco. Cada segmento contiene una tabla de páginas y hay varias tablas de páginas por proceso.

Cada una de las tablas contiene información en cada página de segmento, mientras que la tabla de segmento tiene información sobre cada segmento. Las tablas de segmento se asignan a tablas de página y las tablas de página se asignan a páginas individuales dentro de un segmento.

Las ventajas incluyen menos uso de memoria, más flexibilidad en el tamaño de las páginas, asignación de memoria simplificada y un nivel adicional de seguridad de acceso a datos sobre la paginación. El proceso no provoca fragmentación externa. [11]

Comparativas de paginación y segmentación.

A continuación, presentamos un cuadro comparativo entre la segmentación y la paginación para poder resumir los aspectos más esenciales de cada uno de ellos, esto lo podemos ver en la tabla 2.

Tabla 2 Cuadro comparativo (Ventajas y desventajas) paginación y segmentación.

	Segmentación.	Paginación.
Ventajas.	**Sin fragmentación interna.** **Las tablas de segmentos consumen menos espacio en comparación con las tablas de páginas.** **Los tamaños de segmento promedio son más grandes que la mayoría de los tamaños de página, lo que permite que los**	**A nivel de programador, la paginación es una función transparente y no requiere intervención.** **Sin fragmentación externa.** **Sin fragmentación interna en sistemas operativos actualizados.**

	segmentos almacenen más datos de proceso. **Menos gastos generales de procesamiento.** **Es más sencillo reubicar segmentos que reubicar espacios de direcciones contiguos en el disco.** **Las tablas de segmentos son más pequeñas que las tablas de páginas y ocupan menos memoria.**	**Los marcos no tienen que ser contiguos.**
Desventajas.	**Utiliza tecnología heredada en servidores x86-64.** **Linux solo admite la segmentación en microprocesadores 80x86: establece que la paginación simplifica la administración de la memoria al usar el mismo conjunto de direcciones lineales.** **Portar Linux a diferentes arquitecturas es problemático debido al soporte de segmentación limitado.** **Requiere la intervención del programador.** **Sujeto a una grave fragmentación externa.**	**La paginación causa fragmentación interna en sistemas más antiguos.** **Tiempos de búsqueda de memoria más prolongados que la segmentación; remedio con cachés de memoria TLB.**

Ya que hemos analizado las ventajas y desventajas, podemos revisar varios de los aspecto que los diferencian entre sí, con lo cual podemos apreciar con más detalle las diferencias y sobre todo cual es mejor y peor bajo distintos criterios, esto lo podemos apreciar en la tabla 3 que aparece en la parte inferior. [12]

Tabla 3 Cuadro comparativo (Aspectos fundamentales entre la segmentación y la paginación)

	Segmentación.	Segmentación.
Tamaño (Capacidad).	Los segmentos de tamaño variable son especificados por el usuario.	Tamaño de bloque fijo para páginas y marcos. El hardware de la computadora determina los tamaños de página / marco.
Fragmentación.	La segmentación conduce a la fragmentación externa.	Los sistemas más antiguos estaban sujetos a fragmentación interna al no asignar páginas enteras a la memoria. Los sistemas operativos modernos ya no tienen este problema.
Bases.	Las tablas de segmentación contienen información e ID de segmento, y son más rápidas que las búsquedas directas de tablas de paginación.	Las tablas de páginas dirigen a la MMU a la ubicación y el estado de la página. Este es un proceso más lento que las tablas de segmentación, pero la memoria caché de TLB lo acelera.
Disponibilidad.	Los servidores de Windows pueden admitir la compatibilidad con versiones anteriores, mientras que Linux tiene un soporte muy limitado.	Ampliamente disponible en CPU y como chips MMU.

Códigos y ventanas de ejecución.

Programa61.c

Realizar un programa que simule la memoria virtual que cumpla con las siguientes indicaciones:

- La memoria virtual es de 64 KB
- La memoria física es de 32 KB
- El tamaño de página es de 4 KB
- El programa debe indicar que páginas se encuentran en la memoria física
- Se le indicará al programa manualmente que página se desea insertar a la memoria física
- Cuando se le indique deberá hacer la transformación de todas las direcciones de la página mostrando ambas direcciones en cada transformación
- En el momento que se indica la página que se va a mover a la memoria física se debe aplicar un algoritmo de fallo de página
- Las páginas se administrarán mediante una tabla de páginas que debe estar programada mediante listas

Código completo y explicación en general.

Código en texto.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Direcciones en la MaquinaVirtual-Disco Duro
 char dirDisDur [16][5]={
   {'0','0','0','0', '\0'},
   {'0','0','0','1', '\0'},
   {'0','0','1','0', '\0'},
   {'0','0','1','1', '\0'},
   {'0','1','0','0', '\0'},
   {'0','1','0','1', '\0'},
   {'0','1','1','0', '\0'},
   {'0','1','1','1', '\0'},
   {'1','0','0','0', '\0'},
   {'1','0','0','1', '\0'},
   {'1','0','1','0', '\0'},
   {'1','0','1','1', '\0'},
   {'1','1','0','0', '\0'},
   {'1','1','0','1', '\0'},
   {'1','1','1','0', '\0'},
   {'1','1','1','1', '\0'}
};

//Direcciones en la RAM
char dirMemoriaRAM [8][4]={
    {'0','0','0','\0'},
```

```
    {'0','0','1','\0'},
    {'0','1','0','\0'},
    {'0','1','1','\0'},
    {'1','0','0','\0'},
    {'1','0','1','\0'},
    {'1','1','0','\0'},
    {'1','1','1','\0'}
};

int marcosIniciales[8]={5,4,1,6,7,2,3,0};

//Estructura de un elemento de la Tabla de paginacion
struct nodoTablaPag{
   int numPag;
   int marcoPag;
   int  bitPres;
   struct nodoTablaPag *ptrSig;
};

//Creacion Nodo de la tabla de paginacion
struct nodoTablaPag *crearNodoTabla(int numPagEntrada){
  struct nodoTablaPag *ptrNuevo=(struct nodoTablaPag *) malloc (sizeof(struct
nodoTablaPag));
  ptrNuevo->numPag=numPagEntrada;
  ptrNuevo->marcoPag=-1;
  ptrNuevo->bitPres=0;
  ptrNuevo->ptrSig=NULL;
  return (ptrNuevo);
}

//Ingreso de Nodo de la tabla de paginacion
void pushNodoTabla(struct nodoTablaPag *ptrCab,int numeroEntrada){
  struct nodoTablaPag *ptrNuevo=crearNodoTabla(numeroEntrada);
       struct nodoTablaPag *ptrAnt=ptrCab;
       struct nodoTablaPag *ptrDes=ptrCab->ptrSig;
       int band=0;
       if(ptrCab->ptrSig==NULL){
          ptrCab->ptrSig=ptrNuevo;
       }else{
             for(;(ptrDes!=NULL) && (ptrNuevo->numPag < ptrDes-
>numPag);ptrAnt=ptrAnt->ptrSig,ptrDes=ptrDes->ptrSig);
              if(ptrDes==NULL){
                    ptrAnt->ptrSig=ptrNuevo;
              }else{
                    ptrAnt->ptrSig=ptrNuevo;
                    ptrNuevo->ptrSig=ptrDes;
              }

       }
}
//Estructura de un elemento de la Tabla de marcos
struct nodoTablaMar{
   int numMarco;
   int pagMarco;
   struct nodoTablaMar *ptrSig;
```

```
};

//Creacion Nodo de la tabla de Marcos
struct nodoTablaMar *crearNodoMarcos(int numMarEntrada){
  struct nodoTablaMar *ptrNuevo=(struct nodoTablaMar *) malloc (sizeof(struct
nodoTablaMar));
  ptrNuevo->numMarco=numMarEntrada;
  ptrNuevo->pagMarco=-1;
  ptrNuevo->ptrSig=NULL;
  return (ptrNuevo);
}

//Ingreso de Nodo de la tabla de marcos
void pushNodoMarco(struct nodoTablaMar *ptrCab,int numeroEntrada){
  struct nodoTablaMar *ptrNuevo=crearNodoMarcos(numeroEntrada);
       struct nodoTablaMar *ptrAnt=ptrCab;
       struct nodoTablaMar *ptrDes=ptrCab->ptrSig;
       int band=0;
       if(ptrCab->ptrSig==NULL){
          ptrCab->ptrSig=ptrNuevo;
       }else{
             for(;(ptrDes!=NULL) && (ptrNuevo->numMarco < ptrDes-
>numMarco);ptrAnt=ptrAnt->ptrSig,ptrDes=ptrDes->ptrSig);
       //     printf("Paso \n");
              if(ptrDes==NULL){
                    ptrAnt->ptrSig=ptrNuevo;
              }else{
                    ptrAnt->ptrSig=ptrNuevo;
                    ptrNuevo->ptrSig=ptrDes;
              }

       }
}

//Cola de metodo de fallo de página FIFO
struct nodoCola {
   int datoPag;
   struct nodoCola *ptrSig;
};

//Crear nodo de cola
struct nodoCola * crearNodoCola(int dato){
       struct nodoCola *ptrNuevo=(struct nodoCola *) malloc(sizeof(struct
nodoCola));
        ptrNuevo->datoPag=dato;
     ptrNuevo->ptrSig=NULL;
     return (ptrNuevo);
}

//Push de nodo de cola
void pushNodoCola(struct nodoCola *ptrEnt,int datoEnt){
       struct nodoCola *ptrNuevo=crearNodoCola(datoEnt);
       if(ptrEnt->ptrSig==NULL)
          ptrEnt->ptrSig=ptrNuevo;
       else{
```

```
          struct nodoCola *ptrRec=ptrEnt->ptrSig;
          while(ptrRec->ptrSig!=NULL){
               ptrRec=ptrRec->ptrSig;
          }
          ptrRec->ptrSig=ptrNuevo;
       }
}

//Pop de nodo Cola
int popNodoCola(struct nodoCola *ptrEnt){
       struct nodoCola *ptrRef=ptrEnt->ptrSig;
       int datoSal=ptrRef->datoPag;
       ptrEnt->ptrSig=ptrRef->ptrSig;
       free(ptrRef);
       return(datoSal);
}

//Metodo para insertar en la tabla de paginacion un marco
void meterMarcoEnPag(struct nodoTablaPag *ptrCab,int entradaNumPag, int
entradaNumMarco){
    struct nodoTablaPag *ptrRec=ptrCab->ptrSig;
    while(ptrRec!=NULL){
        if(ptrRec->numPag==entradaNumPag){
           ptrRec->marcoPag=entradaNumMarco;
           ptrRec->bitPres=1;
           break;
        }
        ptrRec=ptrRec->ptrSig;
    }
}

//Metodo para sacar en la tabla de paginacion un marco
void sacarMarcoEnPag(struct nodoTablaPag *ptrCab,int entradaNumPag){
    struct nodoTablaPag *ptrRec=ptrCab->ptrSig;
    while(ptrRec!=NULL){
        if(ptrRec->numPag==entradaNumPag){
           ptrRec->marcoPag=-1;
           ptrRec->bitPres=0;
           break;
        }
        ptrRec=ptrRec->ptrSig;
    }
}

//Metodo para insertar en la tabla de marcos una pagina
void meterPagEnMarco(struct nodoTablaMar *ptrCab,int entradaNumMarco, int
entradaNumPag){
    struct nodoTablaMar *ptrRec=ptrCab->ptrSig;
    while(ptrRec!=NULL){
        if(ptrRec->numMarco==entradaNumMarco){
           ptrRec->pagMarco=entradaNumPag;
           break;
        }
        ptrRec=ptrRec->ptrSig;
    }
```

```
}

//Mostrar tabla general
void mostrarEstadoGeneral(struct nodoTablaPag *ptrCabPag,struct nodoTablaMar
*ptrCabMar){
    struct nodoTablaPag *ptrRecPag=ptrCabPag->ptrSig;
    struct nodoTablaMar *ptrRecMac=ptrCabMar->ptrSig;

        printf("         Tabla de paginacion                 Tabla de
marcos\n");
        printf("    ----------------------------");
        printf("             ");
        printf("------------------ \n");
        printf("    | Pagina || Marco || bitOc |");
        printf("             ");
        printf("| Marco | Pagina |\n");
        printf("    ----------------------------");
        printf("             ");
        printf("------------------ \n");

        while(ptrRecPag!=NULL){

         printf("    ----------------------------");
         if(ptrRecMac!=NULL){
             printf("             ");
             printf("------------------");
         }
         printf("\n");
         printf("    |  %s  |",dirDisDur[ptrRecPag->numPag]);
         if(ptrRecPag->marcoPag!=-1){
           printf("|  %s  |",dirMemoriaRAM[ptrRecPag->marcoPag]);
         }else{
           printf("|  xxx  |");
         }
         printf("|   %d   |",ptrRecPag->bitPres);
         if(ptrRecMac!=NULL){
             printf("             ");
             printf("|  %s  |  %s  |",dirMemoriaRAM[ptrRecMac-
>numMarco],dirDisDur[ptrRecMac->pagMarco]);

         }
         printf("\n");
       printf("    ----------------------------");
         if(ptrRecMac!=NULL){
             printf("             ");
             printf("------------------");
         }
           printf("\n");
           if(ptrRecMac!=NULL){
             ptrRecMac=ptrRecMac->ptrSig;
           }
            ptrRecPag=ptrRecPag->ptrSig;

        }
    printf("\n \n");
```

```
}
//Metodo para ver si página esta en memoria RAM
int pagEnMemoriaRAM(struct nodoTablaPag *ptrCabPag,int paginaEnt){
     struct nodoTablaPag *ptrRecPag=ptrCabPag->ptrSig;
     int exixtenciaMemoria=0;
     while(ptrRecPag!=NULL){
        if(ptrRecPag->numPag==paginaEnt){
           if(ptrRecPag->bitPres==1){
               exixtenciaMemoria=1;
           }
        }
        ptrRecPag=ptrRecPag->ptrSig;
     }
     return exixtenciaMemoria;
}

//Metodo buscar Marco con una página en especial
int  buscarMarco(struct nodoTablaMar *ptrCab,int numPagEntrada){
    struct nodoTablaMar *ptrRec=ptrCab->ptrSig;
    int numeroMarcoSalida=-1;
      while(ptrRec!=NULL){
            if(ptrRec->pagMarco==numPagEntrada){
               numeroMarcoSalida=ptrRec->numMarco;
               break;
            }
            ptrRec=ptrRec->ptrSig;
      }
    return numeroMarcoSalida;
}

//Transformación memoria disco a RAM
void transformacionMemoria(int numeroPagina,int numeroMarco){
   int c,k;
   int contador;
   char numeroBase[13];
   printf("\n \n");
   for (int i=0;i<4096;i++){
       contador=0;
       for(c=11;c >= 0; c--){
          k= i >> c;
          if( k& 1){
             numeroBase[contador]='1';
          }else{
             numeroBase[contador]='0';
          }
          contador++;
       }
       contador++;
       numeroBase[contador]='\0';
       printf("Direccion Disco=%s%s ----->Direccion Ram=%s%s \n",dirDisDur
[numeroPagina],numeroBase,dirMemoriaRAM [numeroMarco],numeroBase);
   }
   printf("\n \n");
}
```

```
//Metodo para mover una página seleccionada por el usuario
void usuarioMoverPagina(struct nodoTablaPag *ptrCabPag,struct nodoTablaMar
*ptrCabMar,struct nodoCola *ptrCola,int entraNumPagMover){
     if(pagEnMemoriaRAM(ptrCabPag,entraNumPagMover)==0){
         int pagSalir=popNodoCola(ptrCola);
         int marcoLLenar=buscarMarco(ptrCabMar,pagSalir);
         sacarMarcoEnPag(ptrCabPag,pagSalir);
         meterPagEnMarco(ptrCabMar,marcoLLenar,entraNumPagMover);
         meterMarcoEnPag(ptrCabPag,entraNumPagMover,marcoLLenar);
          pushNodoCola(ptrCola,entraNumPagMover);
          printf("Se mueve la pagina: %d al marco:%d por el metodo de fallo de
pagina FIFO\n",entraNumPagMover,marcoLLenar);
          transformacionMemoria(entraNumPagMover,marcoLLenar);
          printf("\nEstado de las memorias actualizado\n\n ");
          mostrarEstadoGeneral(ptrCabPag,ptrCabMar);

     }else{
       printf("La pagina ya esta en la RAM \n");
     }


}

int main(){
 struct nodoTablaPag *ptrCabPaginas=crearNodoTabla(-1);
 struct nodoTablaMar *ptrCabMarcos=crearNodoMarcos(-1);
 struct nodoCola *ptrPrimeroCola=crearNodoCola(-1);
 for(int i=0;i<16;i++){
    pushNodoTabla(ptrCabPaginas,i);
 }
 for (int i=0;i<8;i++){
     pushNodoMarco(ptrCabMarcos,i);
 }


 for(int i=0;i<8;i++){
     meterMarcoEnPag(ptrCabPaginas,i,marcosIniciales[i]);
     meterPagEnMarco(ptrCabMarcos,marcosIniciales[i],i);
     pushNodoCola(ptrPrimeroCola,i);
 }

printf("Bienvenido al simulador de memoria virtual \n \n");
printf("La RAM es de 32 KB, el disco duro es de 64 KB y las paginas son de 4 kB
\n \n");
printf("Este es el estado de tabla de paginacion y de marcos en este momento
\n\n");
mostrarEstadoGeneral(ptrCabPaginas,ptrCabMarcos);
int estadoSalida=0;
int numeroPagUsuario=0;
int opccion=0;
while(estadoSalida==0){
    printf("Escriba que opccion quiere elegir:\n");
    printf("1-Mover una pagina\n");
    printf("2-Salir de programa\n \n");
    scanf("%d",&opccion);
```

```
    if(opccion==1){
       printf("\nEscriba una pagina de entre 0 a 15 que quiera mover \n \n ");
       scanf("%d",&numeroPagUsuario);
       if(numeroPagUsuario>=0){
            if(numeroPagUsuario<16){

usuarioMoverPagina(ptrCabPaginas,ptrCabMarcos,ptrPrimeroCola,numeroPagUsuario);
}else{
               printf("\nEscriba un numero de pagina valido \n \n");
            }
       }else{
         printf("\nEscriba un numero de pagina valido \n \n");
       }
    }else{
        if(opccion==2){
           printf("Adios\n");
           estadoSalida=1;
        }else{

             printf("\nEscriba una opccion valida \n \n");
        }

    }

}
 return 0;
}
```

Explicación general.

Primero se declaran valores globales para el programa. El primer valor global es un arreglo de 16 strings de longitud cuatro que servirá para dar las direcciones en binario de las páginas del disco duro de manera sencilla. Luego se declara un arreglo de 8 strings de longitud 3 para dar las direcciones en binario de los marcos de la memoria RAM de manera sencilla. Como el último valor global se declara un arreglo con los marcos iniciales para que contengan las primeras 8 páginas en orden.

Luego se crea la estructura para los nodos que conformaran la lista de la tabla de páginas, teniendo los nodos los datos del número de la página, el marco en el que esta, un bit diciendo si está dentro de algún marco y un puntero apuntado al siguiente nodo de la lista.

Para crear los nodos de la tabla de paginación se crea un método llamado crearNodotbala que recibe el número de la página que tendrá el nuevo nodo y que retorna un puntero de nodo de página con el número de página deseado, un numero de marco con el valor -1 diciendo asi que no tiene marco aún, un bit de presencia dentro de un marco con valor cero representando que no está en un marco y un puntero al siguiente nodo con el valor nulo.

Para automatizar el ingreso de nodos en la tabla de paginación se crea un método el pushNodoTabla que recibe el puntero cabera de la lista de la tabla de paginación y el valor de la página deseada. Para esto usa el método para crear nodos de la tabla de paginación y luego lo ordena por medio de un ciclo for si hay más de un nodo en la lista de la tabla de paginación o lo coloca directamente después del nodo de cabera si no hay otro.

Luego se crea la estructura para los nodos para la tabla de marcos con los datos del número del marco, la página que contiene y el puntero al siguiente nodo de la tabla de marcos.

También se crea un método para crear los nodos llamado crearNodoMarcos que solo recibe el número del marco deseado regresando un puntero de un nodo de la tabla de marcos que tiene el número del marco ingresado, un numero de página de valor -1 simbolizando que no tiene página y un puntero con valor nulo del puntero siguiente.

Igualmente, para automatizar el ingreso de nodos en la tabla de marcos se crea un método el pushNodoMarco que recibe el puntero cabera de la lista de la tabla de marcos y el valor del marco deseado. Para esto usa el método para crear nodos de la tabla de marcos y luego lo ordena por medio de un ciclo for si hay más de un nodo en la lista de la tabla de marcos o lo coloca directamente después del nodo de cabera si no hay otro.

Se crea la última estructura para los nodos de la cola de fallo de página con los datos del número de la página insertada y el puntero al siguiente nodo de la cola.

Luego se crea se crea un método para crear los nodos llamado crearNodoCola que solo recibe el número de la página deseada regresando un puntero de un nodo de cola que tiene el número de la página ingresada y un puntero con valor nulo del puntero siguiente.

Después para automatizar el ingreso de nodos en la cola se crea un método el pushNodoCola que recibe el puntero cabera de la cola y el valor de la página de entrada. Para esto usa el método para crear nodos de la cola y luego si hay más de un elemento en la cola por medio de un while se recorre la cola hasta que se llegue al final y se coloca al final el nuevo nodo. En caso de no hay otro nodo se coloca como el puntero siguiente de la nodo cabecera de la cola.

Para hacer la función pop de la cola se crea el método popNodaCola que recibe el puntero cabera de la cola y regresa el valor de la página del nodo que va después del de cabecera, y además asigna al puntero que le sigue del nodo del valor de regreso, como el puntero siguiente del nodo de cabera. Luego libera el espacio de memoria del nodo del valor saliente y regresa el valor.

Después para poner un marco en un especifico en una página en especial de la tabla de paginación se crea el método meterMarcoEnPag en el que lleva el puntero cabecera de la tabla de paginación, la página a buscar y el marco a meter. Con estos datos se recorre los nodos de la tabla de paginación por medio de los punteros de estos y cuando se encuentra el nodo con el valor de página a buscar se le pone en el atributo del marco el valor de entrada en el método y. el bit de ocupación se cambia a valor uno.

Si se quiere sacar el marco se tiene el método de sacarMarcoEnPag que, con el puntero cabecera de la tabla de paginación, la página a buscar y el marco a sacar, también busca de la misma forma el nodo de la página, solo que al encontrarlo el marco de página se le pone el valor del marco como -1 y el bit de ocupación se le pone el valor de cero.

Para poner una página en un especifico en un marco en especial de la tabla de marcos se crea el método meterPagEnMarco en el que lleva el puntero cabecera de la tabla de marcos, el marco a buscar y la página a meter. Con estos datos se recorre los nodos de la tabla de marcos por medio de los punteros de estos y cuando se encuentra el nodo con el valor del marco a buscar se le pone en el atributo de página el valor de entrada en el método.

Ahora para imprimir el estado de la tabla de paginación y la tabla de marcos, se usa el método mostrarEstadoGeneral , donde con los punteros cabeceras de las listas de la tabla de paginación y de marcos se imprime primero los títulos de la tabla y luego por medio de los punteros se recorre la lista de paginación y de marcos hasta que se llegue a punteros nulo y se imprime los valores de que contiene los nodos en turno de estos.

Luego para verificar que no se mueva una página que ya está en RAM se crea el método pagEnMemoriaRAM que toma los valores del puntero cabecera de la tabla de paginación y el número de la página a buscar, para recorrer la lista por medio de los punteros de los nodos por medio de un while hasta que se llegue al punteo nulo, donde en los ciclos se busca si el nodo de la página está en memoria viendo si el valor de ocupación de este es 1 y si lo es se regresa un uno y si no se regresa un cero.

Ahora para buscar un marco con una página en especial se usa el método buscarMarco que usa los valores de entrada del puntero de cabecera para la lista de marcos y el número de la página a buscar recorre la lista de marcos por medio de los punteros de los nodos por medio de un while hasta que se llegue al punteo nulo, donde en los ciclos si se encuentra el marco con la página buscada se guarda una variable el marco encontrado y se retorna

Después se crea un método para simular la transformación de memoria de la del disco a la memoria de RAM, donde se usa un arreglo de longitud 13 para tener los últimos 12 bits de cada dirección. Para obtener eso últimos 12 bits se usa un ciclo for donde se recorre todos los números de cero a 4096 (número máximo que se alcanza con 12 bits) y luego para transformar estos números a binario se usa un ciclo for que va del 11 al 12 para transformar el numero en turno a binario por medio de las operaciones Giro a la derecha (right-shift) a los números del ciclo for y el número a transformar para ver si se pone un cero o un uno. Ya con estos 12 bits solo se le agregan los bits iniciales de cada memoria por medio de las variables globales que tienen las representaciones de cada número de marco y de página en binario.

El último método es para que el usuario mueva la página por medio del método de usuarioMoverPagina que necesita los punteros cabeceras de las tablas de paginación, de la de marcos y la de cola ; y el número de la tabla a mover para esto. Lo hace primero checando si la página está en la memoria que en caso de estar sigue los siguientes pasos:

1. Se saca de la cola la última página ingresada
2. Se busca el marco que tiene esa página
3. Se quita de la tabla de paginación el marco de la página sacada de la cola
4. Se remplaza en la tabla de marcos , en el marco que se buscó la página que se eliminó por la que se quiere mover
5. Se pone en la tabla de paginación, en la página que se mete el marco que se busco
6. Se mete a la cola la página que se movió
7. Se imprime el mensaje de que se movió la página al marco buscado
8. Se muestra la simulación de la traducción de dirección de memorias
9. Se muestra el estado actual de la tabla de paginación y de marcos

En el programa principal se crean los tres nodos de cabecera para la lista de páginas, marcos y la cola para el método de falla de página por FIFO. Después se crean la tabla de paginación con 16 nodos y luego se crea la lista de la tabla de marcos con 8 nodos. Después se llenan los marcos y páginas, además de que en cada ingreso de página esta se ingresa en la cola. Luego se manda un mensaje

explicando en que consiste la simulación y el programa. Luego un while repite los siguiente pasos hasta que se cambie el estado de la variable estadoSalida a cero , que es cuando se da a la opción de terminar el programa:

1. Pregunta si se quiere mover una página o terminar el programa
2. Si se quiere mover la página, pregunta que página dentro del rango de entre 0 a 15 se quiere mover
3. Si no se ingresa dentro del rango da el mensaje de que se necesita poner dentro del rango y si está dentro del rango llama al método usuarioMoverPagina

Código por partes y explicación detallada de cada parte.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Direcciones en la MaquinaVirtual-Disco Duro
char dirDisDur [16][5]={
  {'0','0','0','0', '\0'},
  {'0','0','0','1', '\0'},
  {'0','0','1','0', '\0'},
  {'0','0','1','1', '\0'},
  {'0','1','0','0', '\0'},
  {'0','1','0','1', '\0'},
  {'0','1','1','0', '\0'},
  {'0','1','1','1', '\0'},
  {'1','0','0','0', '\0'},
  {'1','0','0','1', '\0'},
  {'1','0','1','0', '\0'},
  {'1','0','1','1', '\0'},
  {'1','1','0','0', '\0'},
  {'1','1','0','1', '\0'},
  {'1','1','1','0', '\0'},
  {'1','1','1','1', '\0'}
};

//Direcciones en la RAM
char dirMemoriaRAM [8][4]={
   {'0','0','0','\0'},
   {'0','0','1','\0'},
   {'0','1','0','\0'},
   {'0','1','1','\0'},
   {'1','0','0','\0'},
   {'1','0','1','\0'},
   {'1','1','0','\0'},
   {'1','1','1','\0'}
};
```

Ilustración 3 Código por partes primera sección.

En la ilustración 3 podemos ver que primero se declaran valores globales para el programa. El primer valor global es un arreglo de 16 strings de longitud cuatro que servirá para dar las direcciones en binario de las páginas del disco duro de manera sencilla. Luego se declara un arreglo de 8 strings de longitud 3 para dar las direcciones en binario de los marcos de la memoria RAM de manera sencilla. Como el último valor global se declara un arreglo con los marcos iniciales para que contengan las primeras 8 páginas en orden.

```c
int marcosIniciales[8]={5,4,1,6,7,2,3,0};

//Estructura de un elemento de la Tabla de paginacion
struct nodoTablaPag{
   int numPag;
   int marcoPag;
   int  bitPres;
   struct nodoTablaPag *ptrSig;
};

//Creacion Nodo de la tabla de paginacion
struct nodoTablaPag *crearNodoTabla(int numPagEntrada){
  struct nodoTablaPag *ptrNuevo=(struct nodoTablaPag *) malloc (sizeof(struct nodoTablaPag));
  ptrNuevo->numPag=numPagEntrada;
  ptrNuevo->marcoPag=-1;
  ptrNuevo->bitPres=0;
  ptrNuevo->ptrSig=NULL;
  return (ptrNuevo);
}
```

Ilustración 4 Código por partes segunda sección.

Luego en la ilustración 4 se crea la estructura para los nodos que conformaran la lista de la tabla de páginas, teniendo los nodos los datos del número de la página, el marco en el que esta, un bit diciendo si está dentro de algún marco y un puntero apuntado al siguiente nodo de la lista. Para crear los nodos de la tabla de paginación se crea un método llamado crearNodoTabla que recibe el número de la página que tendrá el nuevo nodo y que retorna un puntero de nodo de página con el número de página deseado, un numero de marco con el valor -1 diciendo asi que no tiene marco aún, un bit de presencia dentro de un marco con valor cero representando que no está en un marco y un puntero al siguiente nodo con el valor nulo.

```c
//Ingreso de Nodo de la tabla de paginacion
void pushNodoTabla(struct nodoTablaPag *ptrCab,int numeroEntrada){
  struct nodoTablaPag *ptrNuevo=crearNodoTabla(numeroEntrada);
    struct nodoTablaPag *ptrAnt=ptrCab;
    struct nodoTablaPag *ptrDes=ptrCab->ptrSig;
    int band=0;
    if(ptrCab->ptrSig==NULL){
       ptrCab->ptrSig=ptrNuevo;
    }else{
        for(;(ptrDes!=NULL) && (ptrNuevo->numPag < ptrDes->numPag);ptrAnt=ptrAnt->ptrSig,ptrDes=ptrDes->ptrSig);
        if(ptrDes==NULL){
           ptrAnt->ptrSig=ptrNuevo;
        }else{
           ptrAnt->ptrSig=ptrNuevo;
           ptrNuevo->ptrSig=ptrDes;
        }

    }
}
//Estructura de un elemento de la Tabla de marcos
struct nodoTablaMar{
   int numMarco;
   int pagMarco;
   struct nodoTablaMar *ptrSig;
};
```

Ilustración 5 Código por partes tercera sección.

En la ilustración de la parte superior (Ilustración 5) podemos ver como para automatizar el ingreso de nodos en la tabla de paginación se crea un método el pushNodoTabla que recibe el puntero cabecero de la lista de la tabla de paginación y el valor de la página deseada. Para esto usa el método para crear nodos de la tabla de paginación y luego lo ordena por medio de un ciclo for si hay más de un nodo en la lista de la tabla de paginación o lo coloca directamente después del nodo de cabera si no hay otro. Luego se crea la estructura para los nodos para la tabla de marcos con los datos del número del marco, la página que contiene y el puntero al siguiente nodo de la tabla de marcos.

```
//Creacion Nodo de la tabla de Marcos
struct nodoTablaMar *crearNodoMarcos(int numMarEntrada){
  struct nodoTablaMar *ptrNuevo=(struct nodoTablaMar *) malloc (sizeof(struct nodoTablaMar));
  ptrNuevo->numMarco=numMarEntrada;
  ptrNuevo->pagMarco=-1;
  ptrNuevo->ptrSig=NULL;
  return (ptrNuevo);
}

//Ingreso de Nodo de la tabla de marcos
void pushNodoMarco(struct nodoTablaMar *ptrCab,int numeroEntrada){
  struct nodoTablaMar *ptrNuevo=crearNodoMarcos(numeroEntrada);
    struct nodoTablaMar *ptrAnt=ptrCab;
    struct nodoTablaMar *ptrDes=ptrCab->ptrSig;
    int band=0;
    if(ptrCab->ptrSig==NULL){
       ptrCab->ptrSig=ptrNuevo;
    }else{
        for(;(ptrDes≠NULL) && (ptrNuevo->numMarco < ptrDes->numMarco);ptrAnt=ptrAnt->ptrSig,ptrDes=ptrDes->ptrSig);
    //  printf("Paso \n");
        if(ptrDes==NULL){
           ptrAnt->ptrSig=ptrNuevo;
        }else{
           ptrAnt->ptrSig=ptrNuevo;
           ptrNuevo->ptrSig=ptrDes;
        }

    }
}
```

Ilustración 6 Código por partes cuarta sección.

Igualmente, en la ilustración 6 se aprecia que, para automatizar el ingreso de nodos en la tabla de marcos se crea un método el pushNodoMarco que recibe el puntero cabecero de la lista de la tabla de marcos y el valor del marco deseado. Para esto usa el método para crear nodos de la tabla de marcos y luego lo ordena por medio de un ciclo for si hay más de un nodo en la lista de la tabla de marcos o lo coloca directamente después del nodo de cabera si no hay otro. Se crea la última estructura para los nodos de la cola de fallo de página con los datos del número de la página insertada y el puntero al siguiente nodo de la cola.

```c
//Cola de metodo de fallo de pagina FIFO
struct nodoCola {
   int datoPag;
   struct nodoCola *ptrSig;
};

//Crear nodo de cola
struct nodoCola * crearNodoCola(int dato){
    struct nodoCola *ptrNuevo=(struct nodoCola *) malloc(sizeof(struct nodoCola));
    ptrNuevo->datoPag=dato;
    ptrNuevo->ptrSig=NULL;
    return (ptrNuevo);
}

//Push de nodo de cola
void pushNodoCola(struct nodoCola *ptrEnt,int datoEnt){
    struct nodoCola *ptrNuevo=crearNodoCola(datoEnt);
    if(ptrEnt->ptrSig==NULL)
       ptrEnt->ptrSig=ptrNuevo;
    else{
       struct nodoCola *ptrRec=ptrEnt->ptrSig;
       while(ptrRec->ptrSig≠NULL){
          ptrRec=ptrRec->ptrSig;
       }
       ptrRec->ptrSig=ptrNuevo;
    }
}
```

Ilustración 7 Código por partes quinta sección.

En la imagen de arriba (ilustración 7), se crea la última estructura para los nodos de la cola de fallo de página con los datos del número de la página insertada y el puntero al siguiente nodo de la cola.

Luego se crea se crea un método para crear los nodos llamado crearNodoCola que solo recibe el número de la página deseada regresando un puntero de un nodo de cola que tiene el número de la página ingresada y un puntero con valor nulo del puntero siguiente.

Después para automatizar el ingreso de nodos en la cola se crea un método el pushNodoCola que recibe el puntero cabero de la cola y el valor de la página de entrada. Para esto usa el método para crear nodos de la cola y luego si hay más de un elemento en la cola por medio de un while se recorre la cola hasta que se llegue al final y se coloca al final el nuevo nodo. En caso de no hay otro nodo se coloca como el puntero siguiente del nodo cabecera de la cola. Cabe recalcar que lo ya mencionado se aprecia en la ilustración 7.

```c
//Pop de nodo Cola
int popNodoCola(struct nodoCola *ptrEnt){
    struct nodoCola *ptrRef=ptrEnt->ptrSig;
    int datoSal=ptrRef->datoPag;
    ptrEnt->ptrSig=ptrRef->ptrSig;
    free(ptrRef);
    return(datoSal);
}

//Metodo para insertar en la tabla de paginacion un marco
void meterMarcoEnPag(struct nodoTablaPag *ptrCab,int entradaNumPag, int entradaNumMarco){
    struct nodoTablaPag *ptrRec=ptrCab->ptrSig;
    while(ptrRec≠NULL){
        if(ptrRec->numPag==entradaNumPag){
            ptrRec->marcoPag=entradaNumMarco;
            ptrRec->bitPres=1;
            break;
        }
        ptrRec=ptrRec->ptrSig;
    }
}
```

Ilustración 8 Código por partes sexta sección.

Se observa en la ilustración 8 que para hacer la función pop de la cola se crea el método popNodaCola que recibe el puntero cabecera de la cola y regresa el valor de la página del nodo que va después del de cabecera, y además asigna al puntero que le sigue del nodo del valor de regreso, como el puntero siguiente del nodo de cabera. Luego libera el espacio de memoria del nodo del valor saliente y regresa el valor.

Después para poner un marco en un especifico en una página en especial de la tabla de paginación se crea el método meterMarcoEnPag en el que lleva el puntero cabecera de la tabla de paginación, la página a buscar y el marco a meter. Con estos datos se recorre los nodos de la tabla de paginación por medio de los punteros de estos y cuando se encuentra el nodo con el valor de página a buscar se le pone en el atributo del marco el valor de entrada en el método y. el bit de ocupación se cambia a valor uno; lo ya mencionado se mostró ilustración 8 de la parte superior.

```c
//Metodo para sacar en la tabla de paginacion un marco
void sacarMarcoEnPag(struct nodoTablaPag *ptrCab,int entradaNumPag){
    struct nodoTablaPag *ptrRec=ptrCab->ptrSig;
    while(ptrRec!=NULL){
        if(ptrRec->numPag==entradaNumPag){
            ptrRec->marcoPag=-1;
            ptrRec->bitPres=0;
            break;
        }
        ptrRec=ptrRec->ptrSig;
    }
}

//Metodo para insertar en la tabla de marcos una pagina
void meterPagEnMarco(struct nodoTablaMar *ptrCab,int entradaNumMarco, int entradaNumPag){
    struct nodoTablaMar *ptrRec=ptrCab->ptrSig;
    while(ptrRec!=NULL){
        if(ptrRec->numMarco==entradaNumMarco){
            ptrRec->pagMarco=entradaNumPag;
            break;
        }
        ptrRec=ptrRec->ptrSig;
    }
}
```

Ilustración 9 Código por partes séptima sección.

Se aprecia en la ilustración 9 como si se quiere sacar el marco se tiene el método de sacarMarcoEnPag que, con el puntero cabecera de la tabla de paginación, la página a buscar y el marco a sacar, también busca de la misma forma el nodo de la página, solo que al encontrarlo el marco de página se le pone el valor del marco como -1 y el bit de ocupación se le pone el valor de cero y del mismo modo para poner una página en un especifico en un marco en especial de la tabla de marcos se crea el método meterPagEnMarco en el que lleva el puntero cabecera de la tabla de marcos, el marco a buscar y la página a meter. Con estos datos se recorre los nodos de la tabla de marcos por medio de los punteros de estos y cuando se encuentra el nodo con el valor del marco a buscar se le pone en el atributo de página el valor de entrada en el método.

```c
//Mostrar tabla general
void mostrarEstadoGeneral(struct nodoTablaPag *ptrCabPag,struct nodoTablaMar *ptrCabMar){
    struct nodoTablaPag *ptrRecPag=ptrCabPag->ptrSig;
    struct nodoTablaMar *ptrRecMac=ptrCabMar->ptrSig;

        printf("         Tabla de paginacion                  Tabla de marcos\n");
        printf("    ----------------------------");
        printf("             ");
        printf("------------------ \n");
        printf("    | Pagina || Marco || bitOc |");
        printf("             ");
        printf("| Marco | Pagina |\n");
        printf("    ----------------------------");
        printf("             ");
        printf("------------------ \n");

        while(ptrRecPag≠NULL){

        printf("    ----------------------------");
        if(ptrRecMac≠NULL){
            printf("             ");
            printf("------------------");
        }
        printf("\n");
        printf("    |  %s  |",dirDisDur[ptrRecPag->numPag]);
        if(ptrRecPag->marcoPag≠-1){
          printf("|  %s  |",dirMemoriaRAM[ptrRecPag->marcoPag]);
        }else{
          printf("|  xxx  |");
        }
        printf("|   %d   |",ptrRecPag->bitPres);
        if(ptrRecMac≠NULL){
            printf("             ");
            printf("|  %s  |  %s  |",dirMemoriaRAM[ptrRecMac->numMarco],dirDisDur[ptrRecMac->pagMarco]);

        }
        printf("\n");
       printf("    ----------------------------");
        if(ptrRecMac≠NULL){
            printf("             ");
            printf("------------------");
        }
          printf("\n");
          if(ptrRecMac≠NULL){
            ptrRecMac=ptrRecMac->ptrSig;
          }
           ptrRecPag=ptrRecPag->ptrSig;

        }
    printf("\n \n");
}
```

Ilustración 10 Código por partes octava sección.

Ahora en la ilustración 10, para imprimir el estado de la tabla de paginación y la tabla de marcos, se usa el método mostrarEstadoGeneral , donde con los punteros cabeceras de las listas de la tabla de paginación y de marcos se imprime primero los títulos de la tabla y luego por medio de los punteros se recorre la lista de paginación y de marcos hasta que se llegue a punteros nulo y se imprime los valores de que contiene los nodos en turno de estos.

```
//Metodo para ver si pagina esta en memoria RAM
int pagEnMemoriaRAM(struct nodoTablaPag *ptrCabPag,int paginaEnt){
    struct nodoTablaPag *ptrRecPag=ptrCabPag->ptrSig;
    int exixtenciaMemoria=0;
    while(ptrRecPag!=NULL){
        if(ptrRecPag->numPag==paginaEnt){
            if(ptrRecPag->bitPres==1){
                exixtenciaMemoria=1;
            }
        }
        ptrRecPag=ptrRecPag->ptrSig;
    }
    return exixtenciaMemoria;
}

//Metodo buscar Marco con una pagina en especial
int  buscarMarco(struct nodoTablaMar *ptrCab,int numPagEntrada){
    struct nodoTablaMar *ptrRec=ptrCab->ptrSig;
    int numeroMarcoSalida=-1;
    while(ptrRec!=NULL){
        if(ptrRec->pagMarco==numPagEntrada){
            numeroMarcoSalida=ptrRec->numMarco;
            break;
        }
        ptrRec=ptrRec->ptrSig;
    }
    return numeroMarcoSalida;
}
```

Ilustración 11 Código por partes novena sección.

Luego en la ilustración 11, para verificar que no se mueva una página que ya está en RAM se crea el método pagEnMemoriaRAM que toma los valores del puntero cabecera de la tabla de paginación y el número de la página a buscar, para recorrer la lista por medio de los punteros de los nodos por medio de un while hasta que se llegue al punteo nulo, donde en los ciclos se busca si el nodo de la página está en memoria viendo si el valor de ocupación de este es 1 y si lo es se regresa un uno y si no se regresa un cero y ahora para buscar un marco con una página en especial se usa el método buscarMarco que usa los valores de entrada del puntero de cabecera para la lista de marcos y el número de la página a buscar recorre la lista de marcos por medio de los punteros de los nodos por medio de un while hasta que se llegue al punteo nulo, donde en los ciclos si se encuentra el marco con la página buscada se guarda una variable el marco encontrado y se retorna.

```c
//Transformacion memoria disco a RAM
void transformacionMemoria(int numeroPagina,int numeroMarco){
   int c,k;
   int contador;
   char numeroBase[13];
   printf("\n \n");
   for (int i=0;i<4096;i++){
      contador=0;
      for(c=11;c >= 0; c--){
        k= i >> c;
        if( k& 1){
          numeroBase[contador]='1';
        }else{
          numeroBase[contador]='0';
        }
        contador++;
      }
      contador++;
      numeroBase[contador]='\0';
      printf("Direccion Disco=%s%s ----->Direccion Ram=%s%s \n",dirDisDur [numeroPagina],numeroBase,dirMemoriaRAM [numeroMarco],numeroBase);
   }
   printf("\n \n");
}

//Metodo para mover una pagina seleccionada por el usuario
void usuarioMoverPagina(struct nodoTablaPag *ptrCabPag,struct nodoTablaMar *ptrCabMar,struct nodoCola *ptrCola,int entraNumPagMover){
    if(pagEnMemoriaRAM(ptrCabPag,entraNumPagMover)==0){
        int pagSalir=popNodoCola(ptrCola);
        int marcoLLenar=buscarMarco(ptrCabMar,pagSalir);
        sacarMarcoEnPag(ptrCabPag,pagSalir);
        meterPagEnMarco(ptrCabMar,marcoLLenar,entraNumPagMover);
        meterMarcoEnPag(ptrCabPag,entraNumPagMover,marcoLLenar);
         pushNodoCola(ptrCola,entraNumPagMover);
         printf("Se mueve la pagina: %d al marco:%d por el metodo de fallo de pagina FIFO\n",entraNumPagMover,marcoLLenar);
         transformacionMemoria(entraNumPagMover,marcoLLenar);
         printf("\nEstado de las memorias actualizado\n\n ");
         mostrarEstadoGeneral(ptrCabPag,ptrCabMar);

    }else{
      printf("La pagina ya esta en la RAM \n");
    }


}
```

Ilustración 12 Código por partes décima sección.

Después en la ilustración 12 se aprecia claramente como se crea un método para simular la transformación de memoria de la del disco a la memoria de RAM, donde se usa un arreglo de longitud 13 para tener los últimos 12 bits de cada dirección. Para obtener eso últimos 12 bits se usa un ciclo for donde se recorre todos los números de cero a 4096 (número máximo que se alcanza con 12 bits) y luego para transformar estos números a binario se usa un ciclo for que va del 11 al 12 para transformar el numero en turno a binario por medio de las operaciones Giro a la derecha (right-shift) a los números del ciclo for y el número a transformar para ver si se pone un cero o un uno. Ya con estos 12 bits solo se le agregan los bits iniciales de cada memoria por medio de las variables globales que tienen las representaciones de cada número de marco y de página en binario.

También se aprecia en la misma ilustración, como el último método es para que el usuario mueva la página por medio del método de usuarioMoverPagina que necesita los punteros cabeceras de las tablas de paginación, de la de marcos y la de cola; y el número de la tabla a mover para esto. Lo hace primero checando si la página está en la memoria que en caso de estar sigue los siguientes pasos:

1. Se saca de la cola la última página ingresada
2. Se busca el marco que tiene esa página
3. Se quita de la tabla de paginación el marco de la página sacada de la cola
4. Se remplaza en la tabla de marcos, en el marco que se buscó la página que se eliminó por la que se quiere mover
5. Se pone en la tabla de paginación, en la página que se mete el marco que se busco
6. Se mete a la cola la página que se movió
7. Se imprime el mensaje de que se movió la página al marco buscado

8. Se muestra la simulación de la traducción de dirección de memorias
9. Se muestra el estado actual de la tabla de paginación y de marcos

```
int main(){
 struct nodoTablaPag *ptrCabPaginas=crearNodoTabla(-1);
 struct nodoTablaMar *ptrCabMarcos=crearNodoMarcos(-1);
 struct nodoCola *ptrPrimeroCola=crearNodoCola(-1);
 for(int i=0;i<16;i++){
    pushNodoTabla(ptrCabPaginas,i);
 }
 for (int i=0;i<8;i++){
     pushNodoMarco(ptrCabMarcos,i);
 }
 for(int i=0;i<8;i++){
     meterMarcoEnPag(ptrCabPaginas,i,marcosIniciales[i]);
     meterPagEnMarco(ptrCabMarcos,marcosIniciales[i],i);
     pushNodoCola(ptrPrimeroCola,i);
 }

printf("Bienvenido al simulador de memoria virtual \n \n");
printf("La RAM es de 32 KB, el disco duro es de 64 KB y las paginas son de 4 kB \n \n");
printf("Este es el estado de tabla de paginacion y de marcos en este momento \n\n");
mostrarEstadoGeneral(ptrCabPaginas,ptrCabMarcos);
int estadoSalida=0;
int numeroPagUsuario=0;
int opccion=0;
while(estadoSalida==0){
    printf("Escriba que opccion quiere elegir:\n");
    printf("1-Mover una pagina\n");
    printf("2-Salir de programa\n \n");
    scanf("%d",&opccion);
    if(opccion==1){
       printf("\nEscriba una pagina de entre 0 a 15 que quiera mover \n \n ");
       scanf("%d",&numeroPagUsuario);
       if(numeroPagUsuario≥0){
           if(numeroPagUsuario<16){
               usuarioMoverPagina(ptrCabPaginas,ptrCabMarcos,ptrPrimeroCola,numeroPagUsuario);
           }else{
              printf("\nEscriba un numero de pagina valido \n \n");
           }
       }else{
         printf("\nEscriba un numero de pagina valido \n \n");
       }
    }else{
        if(opccion==2){
           printf("Adios\n");
           estadoSalida=1;
        }else{
            printf("\nEscriba una opccion valida \n \n");
        }
    }
}
 return 0;
}
```

Ilustración 13 Código por partes onceava sección.

En el programa principal (dentro de la ilustración 13), se crean los tres nodos de cabecera para la lista de páginas, marcos y la cola para el método de falla de página por FIFO. Después se crean la tabla de

paginación con 16 nodos y luego se crea la lista de la tabla de marcos con 8 nodos. Después se llenan los marcos y páginas, además de que en cada ingreso de página esta se ingresa en la cola. Luego se manda un mensaje explicando en que consiste la simulación y el programa. Luego un while repite los siguiente pasos hasta que se cambie el estado de la variable estadoSalida a cero, que es cuando se da a la opción de terminar el programa:

1. Pregunta si se quiere mover una página o terminar el programa
2. Si se quiere mover la página, pregunta que página dentro del rango de entre 0 a 15 se quiere mover
3. Si no se ingresa dentro del rango da el mensaje de que se necesita poner dentro del rango y si está dentro del rango llama al método usuarioMoverPagina

Ejecución (Imágenes y explicación).

Para realizar la ejecución del programa primero debemos compilarlo, para esto utilizamos el comando gcc Programa61.c -o Programa61, en esta ocasión no necesitamos pasar algún parámetro más para la compilación. Una vez hecho eso procedemos a la ejecución, esto mediante el comando ./Programa61. El programa nos mostrara un mensaje de bienvenida, el tamaño de la "memoria RAM" , "disco Duro", el tamaño de cada página, el estado de la tabla de paginación y de marcos. Lo ya descrito se aprecia en la ilustración 14.

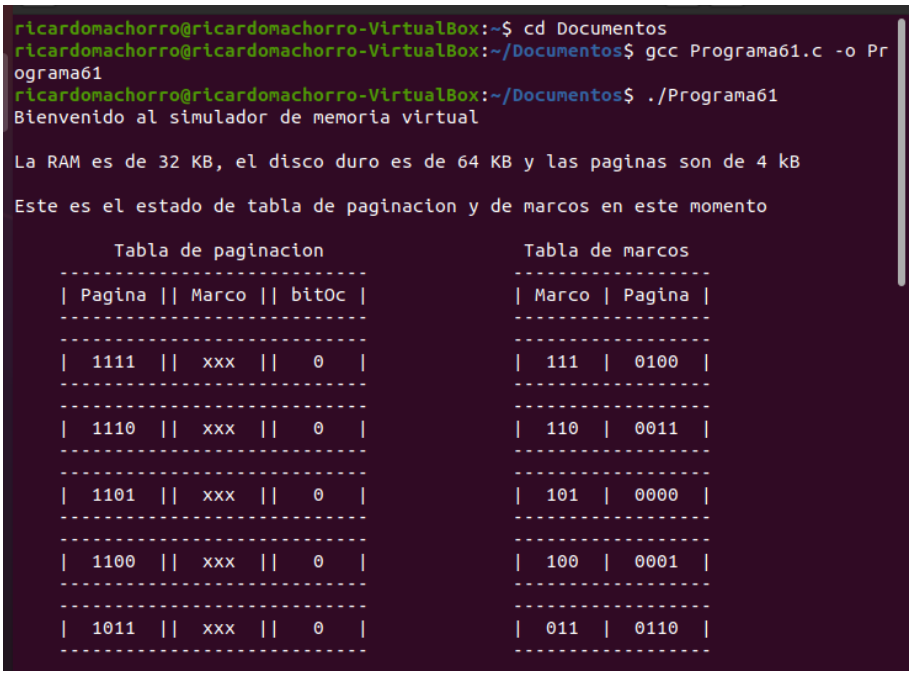


Ilustración 14 Primera parte ejecución.

La tabla de paginación consiste en 3 columnas, la primera columna nos muestra el numero de página, la siguiente el marco en donde se encuentra alojada esa página, si en esta sección aparecen los caracteres "xxx" significa que la pagina no se encuentra cargada en la memoria RAM, la última columna es un bit que indica si la pagina está dentro de algún marco. La tabla de marcos indica el número de página que se encuentra cargada en la memoria RAM y el marco donde se encuentra. Aclaramos que lo mencionado puede verse en la ilustración 15 con el diseño de nuestra tabla en ella.

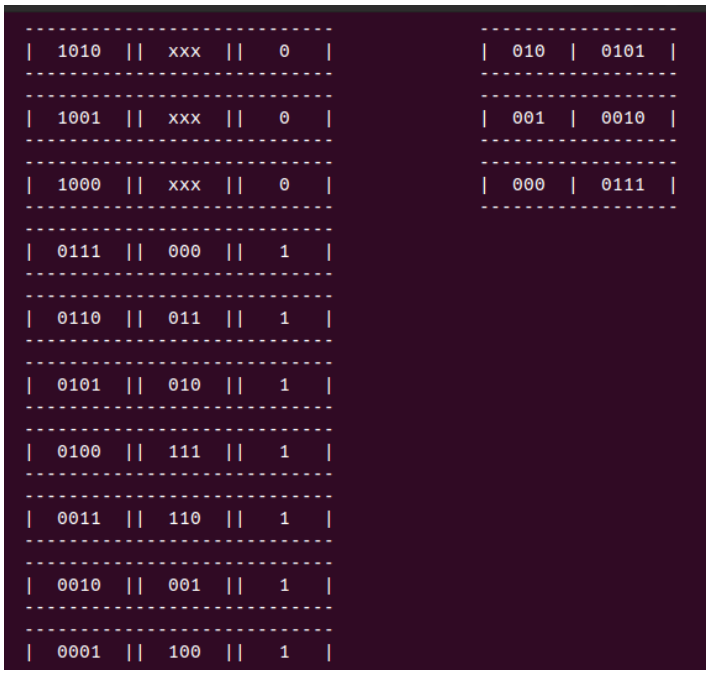


Ilustración 15 Segunda parte ejecución.

Al final de las tablas se muestra el menú de opciones donde podemos mover alguna pagina dentro de la memoria RAM o terminar el programa, justo como se observa en la imagen de la parte inferior (ilustración 16).

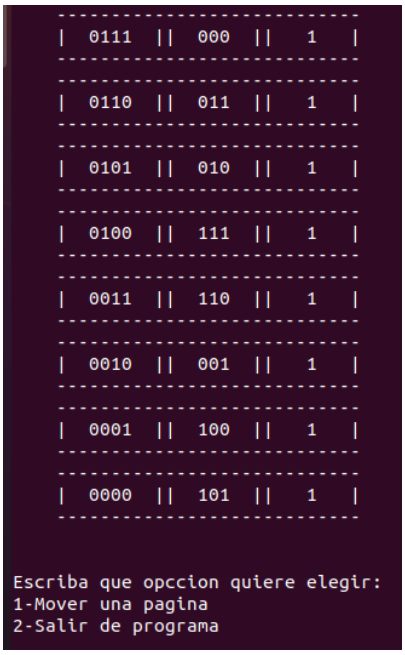


Ilustración 16 Tercera parte ejecución.

Si se le da a la opción uno se mostrará el mensaje que pedirá que página se quiere mover, en caso de ser una que ya está en memoria RAM, mandara un mensaje solicitando al usuario que seleccione una página entre 0 a 15 y volverá a mostrar el primer menú tal como se muestra en la ilustración 17.

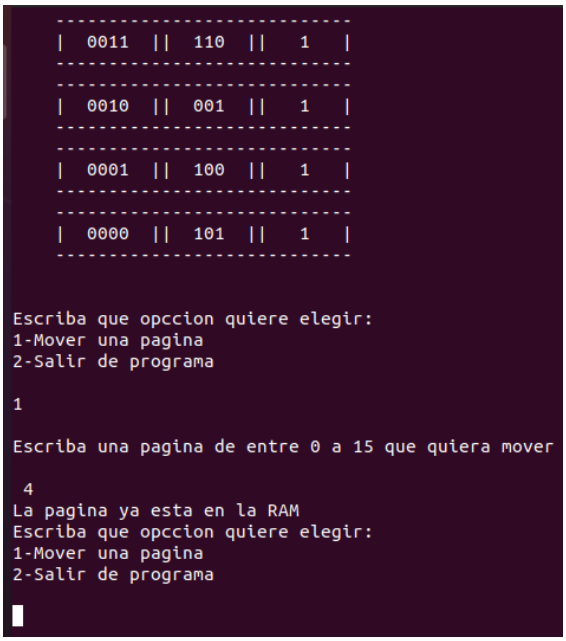


Ilustración 17 Cuarta parte ejecución.

Si se pide mover una que no está en la RAM mostrara el mensaje de que se pasara la página seleccionada al marco con la página más antigua metida, además del proceso de traducción de direcciones de memoria como se muestra en la ilustración 18.

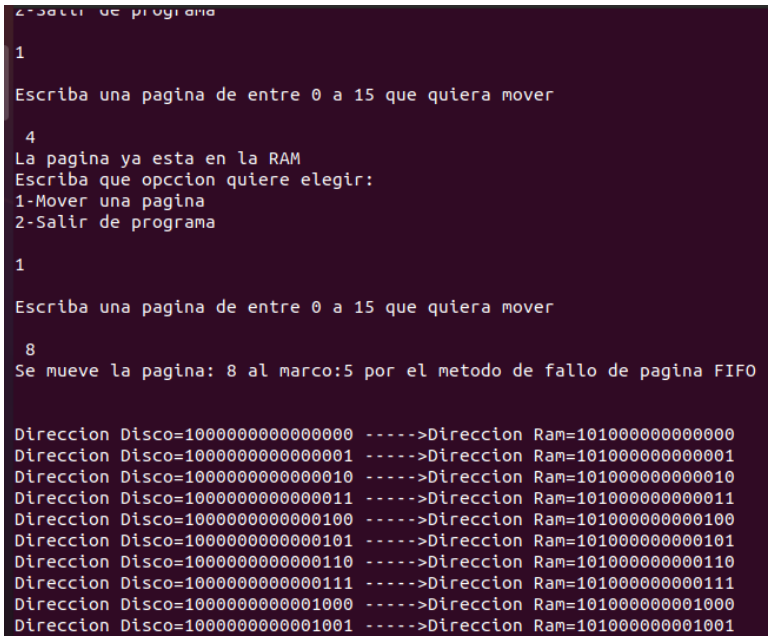


Ilustración 18 Quinta parte ejecución.

Cuando termine la traducción se mostrará la tabla de paginación y de marcos actualizada, para volver a mostrar después el primer menú como se muestra en las ilustraciones 19, 20 y 21 ubicadas en la parte inferior.

Ilustración 19 Sexta parte ejecución.

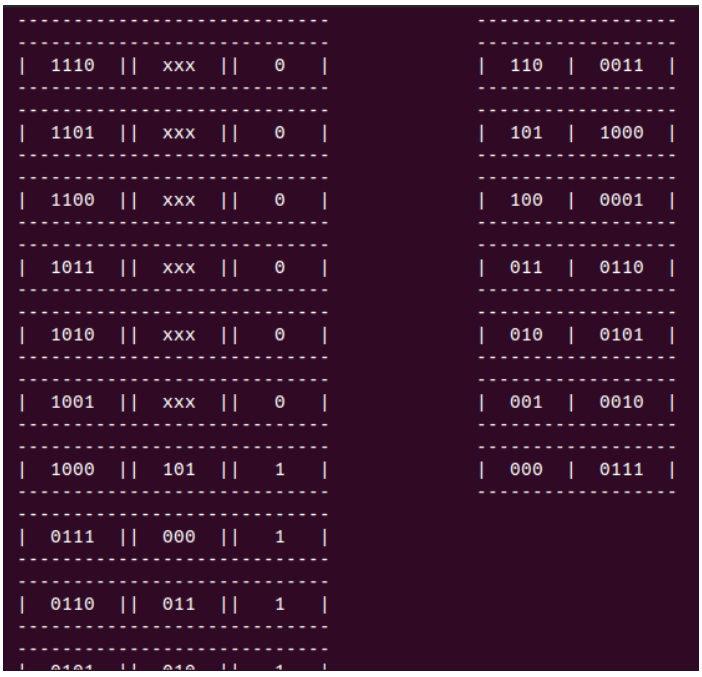


Ilustración 20 Séptima parte ejecución.

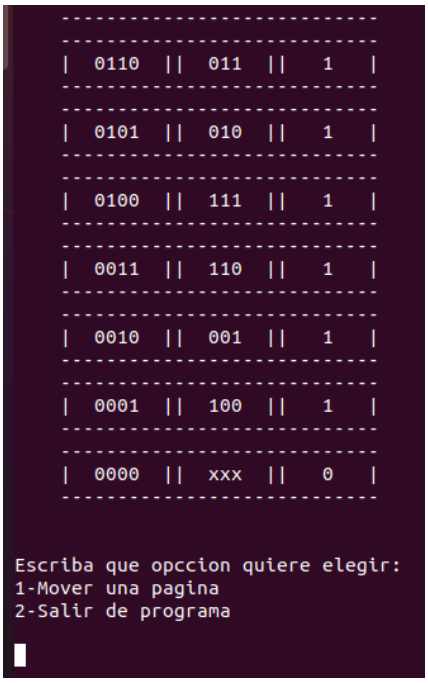


Ilustración 21 Octava parte ejecución.

Por último, pero no menos importante, si se quiere salir del programa cuando aparece el primer menú se da a la opción número 2 y el programa se despide como se ve en la ilustración 22.

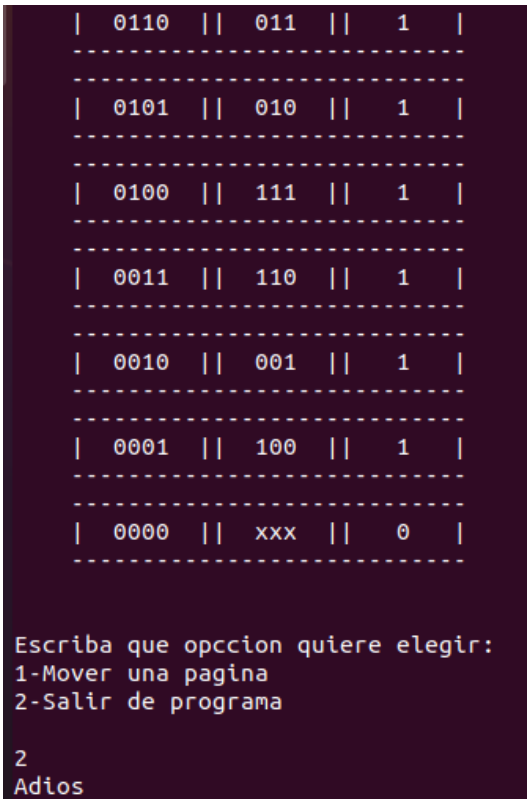


Ilustración 22 Novena parte ejecución.

Conclusiones.

Chavarría Vázquez Luis Enrique.

Si bien es cierto, a pesar de que las capacidades de las computadoras actuales, de los equipos de sobremesa, portátiles y personales en general son bastante superiores a lo que teníamos hace algunos años atrás, en muchas ocasiones se necesitan tener sistemas para la gestión y administración correcta la memoria, debido a que existen ciertos programas y ciertas implementaciones a problemas que requieren tener un flujo de datos bastante grande y por tanto precisan de un manejo de la memoria soberbio; podemos afirmar que existen una gran variedad de dispositivos y por tanto una gran variedad de clientes, por lo cual en la medida de lo posible se debe garantizar que los usuarios puedan utilizar sin importar las circunstancias de su equipo las soluciones que las empresas han implementado o los desarrolladores de software han ideado, evidentemente no conseguiremos que todos los programas y todas las funcionalidades ejecuten en equipos de bajos recursos, pero más sin en cambio existen algunos sistemas de gestión de memoria que nos permiten poder extender el uso de software bastante útil y aprovechar al máximo los recursos que tiene la computadora sin importar sus capacidades técnicas, lo cual está genial para muchos usuarios que usualmente no cuentan con equipos con demás sea potencia ya que recordemos que usualmente cierto tipo de computadoras están dedicadas a nichos en específico.

A decir verdad, a lo largo de esta práctica pudimos implementar una simulación bastante buena y en mi opinión muy didáctica para poder ver el funcionamiento de la memoria virtual, del mismo modo me resultó bastante útil poder ver los vídeos que se nos mostrarán largo de esta unidad número tres, debido a que en la parte teórica se pudo apreciar bastante bien como es que tenemos que implementarlo, pero a decir verdad también tuvimos algunas dificultades que afortunadamente pudimos resolver trabajando y dialogando todos en conjunto.

La verdad es que la mayor parte de nuestras dificultades estuvieron en torno al uso de estructuras que a decir verdad no habíamos implementado ya hace algún tiempo, pero como lo mencione fue una cuestión de apoyarnos en equipo y poder hacer propuestas de solución efectivas, desde luego pudimos ver los procesos y conceptos de la gestión de memoria básica y como esta puede tomar a consideración muchos aspectos que nos ayudan a mejorar la optimización y sobre todo evitar cometer errores en la parte de la optimización de nuestras implementaciones de software.

Considero que lo más fundamental de esta práctica, fue el poder entender qué en ocasiones necesitamos hacer mejoras en la optimización para poder ofrecer sobre todo, un sistema robusto y desde luego que cumpla con los parámetros básicos de calidad del software, además el manejo adecuado de la memoria nos ayuda a poder planificar la implementación de software en diversos dispositivos y ecosistemas, con lo cual se puede tener una mayor flexibilidad y garantizar la escalabilidad de nuestro software, lo que desde mi humilde punto de vista pienso es esencial en una industria que en términos generales cuenta con una variedad tremenda de usuarios y dispositivos.

Juárez Espinoza Ulises.

En los diferentes sistemas operativos existe un proceso que se encarga de administrar la memoria de la computadora, particularmente en mi caso nunca me había puesto a analizar cómo funciona y como esta tarea que se dice sencilla tienen su grado de complejidad, esta práctica me ayudo a comprender algunos conceptos importantes indispensables para hacer posible esta tarea.

El primero es la asignación, cuando en el día a día ejecutamos programas en la computadora el sistema operativo se encarga de asignarle memoria a cada uno mediante su ejecución parece algo hasta cierto punto lógico sin embargo no es tan sencillo como parece, ya que si no se asigna de manera correcta el rendimiento del sistema se vería afectado.

Para lograr esto se dice que se debe de asignar dinámicamente, es decir se asigna solo lo que el proceso requiere y se debe liberar cuando ya no se esté usando e inclusive reasignarse si es necesario. La práctica me ayudo a comprender el concepto de memoria virtual y la razón por la que se implementa hoy en día, esto se debe a que la mayoría de las computadoras actuales cuenta con pocos GB de RAM y no resulta suficiente para cubrir las necesidades que los usuarios pueden llegar a requerir, es aquí donde la memoria virtual tiene un papel importante, gracias a que permite intercambiar datos que no se usaron de manera reciente, moviéndolos a un dispositivo de almacenamiento, lo cual permite liberar espacio en la RAM.

Creo que un muy importante esta función de la memoria virtual, porque no solo nos permite ejecutar más programas de los que normalmente podemos, además nos ayuda a mejorar el rendimiento de nuestro sistema, poder usar programas grandes, entre otros, algo que me pareció interesante es que no debe abusar del intercambio de datos entre la memoria virtual y la RAM ya que la computadora se alentara.

Dicho esto en la práctica se realizó un simulador de memoria virtual en lenguaje C, en donde se emplearon diversas estructuras básicas que este lenguaje ocupa, tales como los punteros, las pilas, las colas y las listas, además que al momento de implementar la memoria virtual, además comprendí otros conceptos que utiliza la memoria virtual para funcionar, tales como las páginas y marcos de página, asi mismo las tablas de página y las tablas de marco de página y las tablas de paginación en donde se muestra el intercambio entre las dos otras tablas.

Machorro Vences Ricardo Alberto.

En esta práctica se trató uno de los elementos que, a pesar de muchas veces ser olvidado en la actualidad, sigue teniendo una gran importancia a la hora de desarrollar un sistema o ejecutar un simple programa. Este elemento es la memoria ya que, aunque en estos tiempos no hay tanto problema con esta como lo era en el pasado por el hecho de que los equipos modernos tienen una capacidad de memoria mayor, sigue siendo importante a tomar en cuenta. En esta práctica aprendí como es que verdaderamente la memoria de la RAM, veloz pero limitada, trabaja con la memoria de disco duro, abundante pero lenta, para así dar la ilusión de tener mucha memoria y rápida.

En las investigaciones que se hizo para las clases de esta unidad se vio diferentes formas y herramientas para la administración de la memoria, siendo la más investigada la de paginación con el fin de poder hacer esta práctica de manera más sencilla. En los videos se ejemplifico como se puede usar estructuras de datos como las lista para la manipulación correcta de las páginas y ver de manera sencilla cual ya está en memoria RAM y cual no.

Esta técnica además fue acompañada por los diferentes algoritmos para la selección de remplazo de una página por otra siendo el ejido para esta práctica la de la cola o mejor conocida como FIFO por su sencillez y por el hecho de que la estructura de las colas ya es usada en otros ámbitos por lo tanto hay más forma de poder recrear esta para la práctica.

Esta práctica además de ayudarme a recordarme como se usan las estructuras básicas para la programación y dejarme usar viejos códigos de la vocacional, también me ayudo a practicar uno de los elementos más importantes del lenguaje C, que son los punteros ya que, aunque muchas veces se pueden ver como algo molesto son muy útiles para crear las estructuras como listas, colas y pilas.

En la creación de esta práctica como conocimiento o habilidad extra me ayudo a ver como se puede crear de manera sencilla una especie de interfaz de usuario rudimentaria por mi propia cuenta, ya que en muchos lenguajes con los que estoy acostumbrado a trabajar ya tienen uno ya implementado ya sea por clases o librerías de estos.

En resumen, esta práctica me ayudo a ver todo los procesos y conceptos que tiene la administración básica de la memoria y como esta tiene que tomar muchas cosas en cuenta para no cometer errores que afecten el procesamiento de un programa o proceso.

Pastrana Torres Victor Norberto.

Durante el desarrollo de la practica recordamos conceptos que hemos visto en cursos pasados, uno de los principales fue el de colas, que en este caso es uno de los factores de mayor importancia ya que en este momento estamos realizando una aplicación directa de ese tema, mediante su uso para la selección de páginas que queríamos cargar a la memoria RAM. Otro tema que recordamos y que en su momento no considere importante fue el de listas, esta estructura de datos es utilizada en los niveles más “bajos” del sistema operativo ya que aquí únicamente trabajamos con bits para gestionar el uso de la memoria.

Durante el repaso de la teoría vimos que otra estructura utilizada en esta sección, es la de los mapas de bits. Durante el curso de programación estructura yo consideraba innecesaria esta estructura además de que me pareció muy frustrante de utilizar, pero ahora que estoy cursando Sistemas Operativos entiendo el porqué de las cosas, ahora entiendo que esta estructura es utilizada por la rapidez que ofrece pero comparado con listas, tiene desventajas ya que solo indica si la página esta ocupada o no, por el contrario listas a pesar de ser más lenta en la búsqueda de algún proceso nos ofrece más información.

La administración de memoria es algo primordial hoy en día, cada vez las aplicaciones y programas pueden realizar cosas impresionantes, pero para lograr todo eso es necesario de recursos suficientes, una de las frases que más recuerdo de las clases es que “los programadores siempre quieren más memoria”, me identifico totalmente con esta frase. Durante toda mi estancia en la Escuela Superior de Computo nunca me había cuestionado si mis programas estaban diseñados de la mejor manera posible, siempre ignoraba si contaba con los recursos necesario para que mi programa se ejecute, pero esto no siempre fue así, en los inicios de lo que hoy conocemos como ordenadores los recursos con los que se disponían eran muy bajos, incluso si comparamos la capacidad de una computadora de los años 2000 con un teléfono móvil actual, las diferencia son impresionantes.

Puedo decir que este tema lo comprendí con facilidad porque el dispositivo móvil con el que cuento tiene bajos recursos y con mucha frecuencia el almacenamiento se llenaba pero yo no entendía muy bien porque, yo sabía que la memoria donde se ejecutan los programas o aplicaciones es las memoria RAM, no el almacenamiento interno pero ahora gracias a esta unidad comprendo que precisamente debido a los recursos limitados con los que cuentan algunos dispositivos, fue necesario buscar de alguna alternativa para que estos dispositivos tuvieran la posibilidad de ejecutar aplicaciones de grandes tamaños, pero incluso esta solución no es suficiente porque siguen existiendo dispositivos que no pueden ejecutar aplicaciones debido a los pocos recursos con los que se cuentan.

Bibliografía

[1] D. Martínez, «Sistemas Operativos. Repositorio Universidad del Nordeste de Argentina.,» de *Sistemas Operativos. Repositorio Universidad del Nordeste de Argentina.*, Buenos Aires, U.N.N.E, S/F.

[2] interserver, «interserver,» 2016 Noviembre 29. [En línea]. Available: https://www.interserver.net/tips/kb/virtual-memory-demand-paging/#:~:text=This%20virtual%20memory%20is%20actually,mapped%20into%20the%20physical%20memory.&text=Demand%20paging%20is%20a%20type,being%20demanded%20by%20some%20program..

[3] geeksforgeeks, «geeksforgeeks,» 16 Agosto 2019. [En línea]. Available: https://www.geeksforgeeks.org/virtual-memory-in-operating-system/.

[4] search server virtualization, «search server virtualization,» Noviembre 2013. [En línea]. Available: https://searchservervirtualization.techtarget.com/definition/memory-paging. [Último acceso: 2020].

[5] unne, «unne,» 10 Julio 2002. [En línea]. Available: http://exa.unne.edu.ar/informatica/SO/SO3.htm.

[6] M. Alegre Ramos, «SISTEMAS OPERATIVOS MONOPUESTO.,» 2010. [En línea]. Available: https://books.google.com.ec/books?id=4zjxk81LgKIC&pg=PA70&dq=diferencia+entre+paginacion+y+segmentacion+de+memoria&hl=es&sa=X&ved=0ahUKEwiYysjAzPLQAhWEBsAKHTecBHgQ6AEIMzAF#v=onepage&q=diferencia%20entre%20paginacion%20y%20segmentacion%20de%20memoria&f=fal.

[7] rcujilan, «Diseño de Sitemas RCUJILAN,» 3 Abril 2017. [En línea]. Available: http://rcujilan.blogspot.com/2017/04/diferencia-entre-la-administracion-de.html.

[8] A. Tanenbaum, Sistemas Operativos Modernos, México CDMX, 2009.

[9] A. Villar, «Introducción a la informática y al uso de y manejo de aplicaciones comerciales.,» España, 2006.

[10] G. Wolf, «SISTOP,» Facultad de ingenieria UNAM, 2016. [En línea]. Available: http://gwolf.sistop.org/laminas/12-memoria-virtual.pdf. [Último acceso: 2020].

[11] EHU, «Segmentación paginada [SILB94],» 2012. [En línea]. Available: https://lsi.vc.ehu.eus/pablogn/docencia/manuales/SO/TemasSOuJaen/ADMINISTRACIONDELAMEMORIA/5.3SegmentacionPaginada.htm.

[12] enterprise storage forum, «Paginación y segmentación,» eWeek, https://www.enterprisestorageforum.com/storage-hardware/paging-and-segmentation.html#:~:text=Paging%20is%20a%20computer%20memory,data%20needs%20a%20storage%20address., 2020.

[13] DCC UCHILE, «users.dcc.uchile.cl,» users.dcc.uchile.cl, S/F. [En línea]. Available: https://users.dcc.uchile.cl/~jpiquer/Docencia/SO/aps/node28.html. [Último acceso: 2020].