INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

SISTEMAS OPERATIVOS

PRÁCTICA 6:

Simulador de memoria virtual

ALUMNOS:

Martínez Ramírez Sergi Alberto – 2020630260 - 2CV15
Meza Vargas Brandon David – 2020630288 - 2CM15
Peña Atanasio Alberto – 2020630367 - 2CV15
Sarmiento Gutiérrez Juan Carlos – 2020630386 - 2CV15

PROFESOR:

José Alfredo Jiménez Benítez

Índice de contenido

Índice de figuras

Glosario de términos

Proceso: programa en ejecución.

Memoria principal: uno de los elementos más importantes y fundamentales de una computadora, destinado al almacenamiento de información y de servir de soporte para la ejecución y gestión de procesos.

Memoria virtual: parte del almacenamiento secundario, que tiene tratamiento de memoria principal y que permite ampliar el número de proceso que concurrentemente pueden ser ejecutados en el sistema.

Paginación: División de un proceso en páginas de igual tamaño y la memoria en marcos de igual tamaño que las páginas.

Segmentación: División de un proceso en segmentos de diferente tamaño, acorde con el tamaño de este.

Listas enlazadas: Una lista es una estructura dinámica de datos. Cada objeto de la estructura está formado por los datos junto con un puntero al siguiente objeto. Al manejar punteros, los datos no tienen por qué estar situados en posiciones consecutivas de la memoria, y lo más normal, es que estén dispersos.

Puntero: Un puntero no es más que una variable estática cuyo contenido es una dirección de memoria.

Contenido

Memoria Virtual

Al hablar de memoria virtual no hacemos referencia a un módulo más de memoria RAM que podemos visualizar entre los componentes de nuestra computadora. La memoria virtual es una técnica de gestión de la memoria del equipo, cuyo uso reside en la utilización conjunta de la memoria principal del sistema y nuestra unidad de almacenamiento, como un disco duro o un ssd. Gracias a la memoria virtual nuestro sistema puede usar parte del almacenamiento como si se tratara de memoria adicional.

En este método, los procesos tendrán sus páginas o segmentos divididos unos en la memoria principal y otros en la memoria virtual, de tal forma que puedan compartir la memoria y por supuesto los demás recursos del sistema.

La incorporación de la memoria virtual involucra la función de intercambio, encargada de establecer qué páginas o segmentos deberán ser removidas de la memoria principal en el caso en que un proceso solicite una página o segmento que no se encuentre en ella.

Si la función de intercambio se realiza de manera frecuente, donde el procesador es monopolizado por la misma sin permitir la ejecución efectiva de procesos, se presenta el fenómeno de hiperpaginación.

Paginación

Esta técnica divide la totalidad de la memoria en fragmentos del mismo tamaño denominados marcos de página. En esta técnica, los procesos son divididos en fragmentos del mismo tamaño de los marcos, los cuales se les denomina páginas.

Así, un proceso tendrá varias páginas asignadas que podrán ocupar marcos de página en la memoria. Este método es la base de la división de la memoria en los sistemas operativos actuales.

Es importante mencionan que se hace necesaria la utilización de estructuras de control que permitan establecer que páginas están asignadas a cada proceso y en que marco, las cuales se denominan tablas de página.

Segmentación

En esta técnica la memoria es dividida de acuerdo con el tamaño de cada proceso en pequeños fragmentos denominados segmentos, el cual tiene un tamaño máximo asignado. En memoria, los segmentos pueden ocupar posiciones no necesariamente contiguas.

Al igual que en la paginación, se hace necesario la utilización de una estructura de datos en la cual se establece la ubicación en memoria de cada uno de los segmentos que componen el proceso denominada tabla de segmentos.

Desarrollo de la práctica

En la práctica se solicita lo siguiente:

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

El código completo de este programa se puede ver en los anexos del presente reporte.

Primeramente, se crea una estructura llamada tabla que servirá como tabla de páginas y la tabla de marcos, esto lo podemos ver en la imagen 1.

```
7  typedef struct tabla{ //se crea la estructura de la tabla de marcos y paginas
8      int indice; //guarda el indice de pagina o marco
9      int referencia; //guarda la pagina o marco al que se hace referencia
0      int estado; //para mantener control de la pagina en la tabla 1: esta, 0: no esta
1      struct tabla *sig; //estrucutra que apunta a la siguiente pagina o marco
2  }tabla;
3
```

Imagen 1. Estructura de la tabla.

Se puede ver que la estructura contiene el índice de la página o del marco, además de una variable entera llamada referencia que servida para indicar el marco o a la página a la que se hace referencia y un estado para saber si está o no en la tabla.

A continuación, se explican las funciones necesarias para que funcione este programa y que hacen posible la simulación de la memoria virtual.

En la imagen 2 tenemos las funciones que realizan el desplazamiento de los bits.

```c
void bitsPagina(int indiceP){ //funcion que hace la conversión de indice a binario
    int bits=3; //indica los bits del indice de pagina
    while(bits >= 0){ //este ciclo while hace la conversion a binario
        if(indiceP & (((long int)1) << bits)) //si el indice y el recorrido regresan verdadero se regresa 1 si no 0
            printf("1");
        else
            printf("0");
        bits--;
    }
}
void bitsMarco(int indiceM){ //funcion que hace la conversión de indice a binario
    int bits=2; //cantidad de bits del indice de marco
    while(bits >= 0){
        if(indiceM & (((long int)1) << bits))
            printf("1");
        else
            printf("0");
        bits--;
    }
}
void desplazamiento(int n){
    int bits = 11;//cantidad de bits de los bits de desplazamiento
    while(bits >= 0){
        if(n & (((long int)1) << bits))
            printf("1");
        else
            printf("0");
        bits--;
    }
}
```

Imagen 2. Funciones de desplazamiento de bits.

Como vemos en la imagen, el caso de la función de bitsPagina(), recibe el índice de la página de la cual se hará su transformación a binario, esto se logra con un ciclo while que ira recorriendo la cantidad de bits de son destinadas para este índice que son 4. Dentro del ciclo while hay una condición donde se hace uso del operador << para desplazar bits a la izquierda y determinar si se imprime un 1 y un 0, de esta forma se va convirtiendo el índice de página a su correspondiente en binario. Esta función es lo mismo que bitsMarco() y desplazamiento(), solo que en estas se usa el índice de marco de página y la cantidad de bits de direccionamiento que son 12, respectivamente.

Posteriormente tenemos la función printDesp() que se encarga de mostrar los 4096 direcciones que tiene la página(4KB), esta parte del código la vemos en la imagen 3.

```c
void printDesp(int indiceP, int indiceM){ //esta funcion realiza los 4096 direccionamientos
    int i;

    for(i=0;i<4096;i++){ //for que va de 0 hasta las 4096 direcciones de la pagina
        bitsPagina(indiceP);
        printf("\t");
        desplazamiento(i);
        printf("\t ------ \t");
        bitsMarco(indiceM);
        printf("\t");
        desplazamiento(i);
        printf("\n");
    }
}
```

Imagen 3. Función printDesp()

Como se ve, esta función manda a llamar a las funciones revisadas en la imagen 2 4096 veces simulando de esa manera el direccionamiento de las paginas correspondientes.

En la imagen 3 podemos ver la función que se encarga de crear las páginas que estarán en la tabla de páginas.

```
65  tabla *crearPag(int i){ //esta funcion crea una pagina
66      tabla *aux = (tabla*) malloc(sizeof(tabla)); //reservo memoria para la pagina
67      aux->indice = i; //se asigna el indice de las paginas en memoria virtual
68      aux->estado = 1; //esta en la tabla de paginas
69      aux->referencia = -1; //de inicio no esta en ningun marco de pagina
70      aux->sig = NULL; //se apunta a la siguiente pagina
71      return aux;
72  }
```

Imagen 4. Función *crearPag().

Como vemos, esta función recibe el índice de la página como parámetro y va estableciendo los valores de cada página, aquí es muy importante decir que solo se crean las páginas y no se agregan a la tabla de páginas, al final se usa el auxiliar para apuntar a la siguiente página y este se retorna.

En la imagen 5 podemos ver la función que se encarga de agregar las paginas a la tabla de páginas.

```
73  tabla *agregarPag(tabla *tblPaginas, tabla *nuevaPag){ //esta funcion agrega la pagina a la tabla de paginas
74      if(tblPaginas == NULL){ //si no hay nada dentro de la tabla se agrega la pagina al inicio
75          tblPaginas = nuevaPag;
76      }else{ //si ya hya una pagina se agrega en el siguiente nodo
77          nuevaPag->sig = tblPaginas;
78          tblPaginas = nuevaPag;
79      }
80      return tblPaginas;
81  }
```

Imagen 5. Función *agregarPag().

Como vemos esta función recibe dos variables de tipo tabla, primeramente, se pregunta si la tabla está vacía, si es así, se ingresa una página al inicio de la tabla, si no es así, se ingresa en el siguiente nodo de la tabla, esto se hace con las 16 páginas posibles, al final se regresa la tabla de páginas.

Para el marco de página se hace algo similar con las páginas, primeramente, se crea el marco de página, esto se ve en la imagen 6.

```
109 tabla *insertP(int dire){ //esta funcion solo crea el marco de pagina pero no agrega la pagina
110     if(pgs < 8){
111         tabla *aux = (tabla*) malloc(sizeof(tabla)); //reservo memoria para la pagina
112         aux->indice = pgs; //el indice de marco se va creando conforme las paginas que se van agregando
113         aux->estado = 1; //el estado se vuelve uno si la pagina este en la tabla de marcos
114         aux->referencia = dire; //se asigna el indice de pagina a la que se hace referencia
115         aux->sig == NULL;   //se apunta al siguiente marco
116         pgs++;  //se incrementa la cantidad de paginas en marcos
117         return aux;
118
119     }else{ //si la memoria fisica esta llena, se procede a realizar un remplazo de paginas por FIFO
120         printf("Memoria fisica llena, se hara un remplazo de marco de pagina\n");
121         tabla *aux = (tabla*) malloc(sizeof(tabla));
122         aux = FIFO(aux, dire);
123         pgs++;
124         return aux;
125     }
126
127 }
```

Imagen 6. función insertP().

En primer lugar, se pregunta si hay menos de 8 páginas ya creadas, pues solo pueden existir 8 marcos de páginas según las especificaciones del problema. Si hay espacio se procede a crear el marco de páginas, se agrega su índice con la ayuda de una variable global que va indicando la cantidad de páginas, así como el estado de ese marco de página y la pagina a la que hace referencia el marco, al final se retorna este marco creado.

Si la memoria física está llena, es decir ya existen 8 marcos en la tabla de marcos, se procede a remplazar una página, esto con la función FIFO() cuyo resultados se almacena en la variable aux que es de tipo tabla.

En la imagen 7 podemos ver la función que agrega el marco creado en la función de la imagen 6 a la tabla de marcos.

```
128 tabla *agregarMarco(tabla *tblMarcos, tabla *nuevoMarco){ //aqui se agrega la pagina a un marco de pagina
129     if(tblMarcos == NULL) //si la tabla de marcos esta vacia se agrega al inicio
130         tblMarcos = nuevoMarco;
131     else{//si no esta vacia se agrega en el siguiente nodo
132         nuevoMarco->sig = tblMarcos;
133         tblMarcos = nuevoMarco;
134     }
135     return tblMarcos;
136 }
```

Imagen 7. Función *agregarMarco().

Como podemos ver, se hace algo similar a la función que agrega las paginas a la tabla de páginas, primero se pregunta si la tabla está vacía, si es así se agrega el marco al inicio de la tabla de marcos, pero si ya tiene un marco se añade el nuevo marco en el siguiente nodo de la tabla de marcos.

En la imagen 8 podemos ver la función FIFIO() esta se encarga de verificar cual fue la primera página que se ingresó a un marco de páginas, en este caso, la primera página que entro será aquella con el índice de marco de pagina 0.

```
 98 tabla* FIFO(tabla *tblMarcos, int dire){ //funcion que establece que marco sale y cual entra
 99     tabla *aux;
100     if((tblMarcos !=NULL) && (tblMarcos -> indice == 0)){
101         aux = tblMarcos;
102         aux->referencia = dire; //se asigna la nueva pagina a la que el primer marco hara referencia
103         tblMarcos = aux;
104         // printf("pgs %d %d\n", tblMarcos->indice, tblMarcos->referencia);
105         return tblMarcos;
106     }
```

Imagen 8. Función FIFO();

Como podemos ver se pregunta si la tabla de marcos no está vacía y si el índice de marco de página es igual a 0, de ser verdad estas dos condiciones se devuelve el primer marco que entro en la tabla de marcos y la nueva página a la que hace referencia, pero aún no se modifica en la tabla de marcos.

En la imagen 9 podemos ver la función addFIFO() que se encarga de sobrescribir la nueva página a la que hará referencia el marco de página que entro primero, es decir, el marco de página con el índice 0.

```
tabla *addFIFO(tabla *tblMarcos, tabla *nuevoMarco){
    //se sobreescribe el primer marco ya que fue el primero en ingresar
    tabla* ant, *aux;
    aux = tblMarcos;
        while((aux != NULL) && (nuevoMarco->indice != aux->indice)){
            //se hace un while para recorrer los marcos hasta que haya coincidencia
            ant =aux;
            aux = aux->sig; //se van recorriendo los marcos
            // printf("indi %d marco%d\n",aux->indice, nuevoMarco->referencia);
        }
        if(aux!=NULL){
            //cuando se encuentra una coincidencia y no es nula se modifica la referencia
            aux->referencia = nuevoMarco->referencia;
        }
    // printf("indi %d marco%d\n",nuevoMarco->indice,nuevoMarco->referencia);
    // printf("indi %d marco%d\n",tblMarcos->indice,tblMarcos->referencia);
    return tblMarcos;
}
```

Imagen 9. Función addFIFO().

Esta función recorre todos los marcos de página hasta encontrar el marco de página con índice 0, una vez encontrado se remplaza la página a la que hacía referencia por la nueva. Al final se retorna la tabla de marcos de página con este remplazamiento.

En la imagen 10 podemos ver la función *busca(), esta se encarga de establecer el marco de página en el que esta o al que hace referencia la página movida, de esta forma se elimina la página de la tabla de páginas.

```
tabla *busca(tabla *tblPaginas, int marco, int indi){ //esta funcion busca el indice de marco al que la pagina hara referencia
    tabla *aux, *ant; //se crean auxiliares para no modificar la tabla original
    if(tblPaginas != NULL &&  (marco == tblPaginas->indice)){ //se verifica si el indice de pagina es el mismo que el de pagina
       // printf("indi %d marco%d\n",tblPaginas->indice, marco);
        aux = tblPaginas; //el auxiliar es para no modificar la tabla original
        aux->referencia = indi; //se modifica el marco a l que la pagina hace referencia
        aux->estado = 0;
        tblPaginas = aux; //ahora si se guarda en la tabla original con la referencia actualizada
        printDesp(tblPaginas->indice, indi);// se realiza el direccionamiento de bits
        return tblPaginas;
    }else{
        aux = tblPaginas;
        while((aux != NULL) && (marco != aux->indice)){ //se hace un while para recorrer las paginas hasta que haya coincidencia
            ant =aux;
            aux = aux->sig; //se van recorriendo las tablas de pagina
            //printf("indi %d marco%d\n",aux->indice, marco);
        }
        if(aux!=NULL){ //cuando se encuentra una coincidencia y no es nula se modifica el marco referenciado
            aux->referencia = indi;
            aux->estado = 0;
            printDesp(aux->indice, indi);// se realiza el direccionamiento de bits
        }

        return tblPaginas; //se regresa la tabla de paginas con los marcos a los cuales se referencian
    }
```

Imagen 10. Función *busca().

Como se ve, se recorren todos los índices de página hasta verificar si la página a la que hace referencia el marco de página es igual al marco de página, si esto se cumple, la página que fue movida a la tabla de marcos se ve alterada en la tabla de páginas, cambia su estado a 0 pues esta fue movida de la tabla de páginas y cambia el marco al cual hace referencia. Al final se retorna la tabla de páginas con estas modificaciones a la página.

En la imagen 11, tenemos la función verifica(), esa se encarga de verificar si la página que se desea mover ya ha sido movida de la tabla de páginas o aun está en la tabla de páginas.

```
186 int verifica(tabla *tblPaginas, int dire){ //se verifica si la pagina sigue en la tabla o ya se movio
187     tabla *aux, *ant; //auxiliares para no modificar la tabla original
188     if(tblPaginas != NULL && (dire == tblPaginas->indice)){ //se verifica el estado de la primera pagina
189         if(tblPaginas->estado == 1) //si la pagina esta se regresa un 1 si ya fue movida un 0
190             return 1;
191         else
192             return 0;
193     }else{
194         while((tblPaginas != NULL) && (dire != tblPaginas->indice)){ //recorre todas las paginas
195             tblPaginas = tblPaginas->sig;
196         }
197         if(tblPaginas != NULL){ //si esta la pagina se regresa un 1 si no un 0
198             if(tblPaginas->estado==1)
199                 return 1;
200             else
201                 return 0;
202         }
203     }
204 }
```

Imagen 11. Función verifica().

Primeramente, se verifica la primera página, si esta está en la tabla de páginas se regresa un uno y si no, se regresa un 0, después se recorre toda la tabla de páginas hasta encontrar la página que se quiere mover y se verifica su estado, si esta (estado = 1) se regresa un 1 y si no está (estado = 0) se regresa un 0.

En la imagen 12 podemos ver las funciones mostrarT() y mostrar TM(), que imprimen la tabla de páginas y la tabla de marcos de páginas, respectivamente.

```
82 void mostrarT(tabla *tblPaginas){ //imprime la tabla de paginas ya establecidas al inicio
83     tabla *aux = tblPaginas; //usamos aux para recorrer la tabla e imprimirla
84     while(aux!=NULL){
85         printf("\t\tindice: %d\t\tEstado: %d\t\tMarco referenciado: %d\n", aux->indice, aux->estado, aux->referencia);
86         aux=aux->sig;
87     }
88 }
89 void mostrarTM(tabla *tblMarcos){ //imprime la tabla de marcos de magina
90     tabla *aux = tblMarcos; //usamos aux para recorrer la tabla e imprimirla
91     if(aux ==  NULL) // para verificar si ya se ha movido una pagina a la memoria fisica
92         printf("Aun no se ha movido ninguna pagina a RAM\n");
93     while(aux!=NULL){
94         printf("\t\tindice: %d\t\tPagina referenciada: %d\n", aux->indice, aux->referencia);
95         aux=aux->sig;
96     }
97 }
```

Imagen 12. Funciones mostrarT() y mostrarTM().

Como se puede ver ambas funciones son prácticamente lo mismo, pero se recurrió a hacer una por cada tabla ya que al hacerlas en una misma función ocurría un problema de violación de memoria, investigando un poco nos dimos cuenta de que sucedía por que se intentaba entrar a segmentos de memoria no permitidos o había conflictos en la memoria, haciendo una función para cada tabla solucionó este problema. La única diferencia que se agregó al final es preguntar si la tabla de marcos esta vacía, es decir, si ninguna página ha sido movida a RAM, se imprime un mensaje indicándolo.

Por último veamos el main() de nuestro programa, como podemos ver en la imagen 13, primero creamos las tablas y llenamos la tabla de páginas.

```c
int main(){
    int opc, i, dire, esta;
    tabla *tblPaginas = iniTabla(); //creamos la tabla de paginas
    tabla *tblMarcos = iniTabla();  //creamos la tabla de marcos

    for(i=0;i<16;i++){ //se llena la tabla de paginas automaticamente con 16 paginas
        tabla *nuevaPag = crearPag(i); //se crea la pagina
        tblPaginas = agregarPag(tblPaginas, nuevaPag); //se agrega la pagina a la tabla
    }
```

Imagen 13. Primera parte del main().

Como se ve, se crean las tablas con la función iniTabla(), esta solo regresa un valor NULL para crear la tabla vacía, posteriormente, llenamos con un for la tabla de páginas con exactamente 16 páginas acorde a las especificaciones del problema.

En la imagen 14 vemos una parte del menú de operaciones del programa que son mostrar las tablas, mover página o salir.

```c
while(opc >= 0){ //while para mostrar un menu del programa
    printf("\nOperaciones con la memoria:\n");
    printf("1. Ver tablas\n");
    printf("2. Mover pagina a memoria fisica\n");
    printf("3. Salir\n");
    scanf("%d",&opc);
    if(opc > 3)
        printf("Opcion no valida\n");

    switch (opc)
    {
    case 1:
        system("clear");
        printf("\t\tTabla de paginas\n\n");
        mostrarT(tblPaginas); //funcion que imprime la tabla de paginas
        printf("\n\t\tTabla de Marcos de pagina\n\n");
        mostrarTM(tblMarcos);   //funcion que imprime la tabla de marcos
        break;
```

Imagen 14. Primera parte del menú del programa.

Se hace un ciclo while para mostrar el menú siempre y cuando la opción ingresada la 3 de salir, podemos ver la opción 1 que es mostrar las tablas, tanto la de páginas como la de marcos de páginas.

En la imagen 15 podemos ver la otra parte del menú donde vemos la opción 2 y la 3.

```c
case 2:
    system("clear");
    printf("Ingrese la pagina a mover a RAM 0-15\n");
    scanf("%d", &dire);
    if(dire>15){ //solo se pueden mover las paginas que hay en la tabla
        printf("Solo hay 16 paginas en memoria virtual\n");
        break;
    }

    else{
        esta = verifica(tblPaginas, dire); //verifica si la pagina esta en la tabla de paginas
        if(esta == 1){ //si si esta la pagina se procede a mover
            tabla *nuevoMarco = insertP(dire);

            if(pgs < 9){
                tblMarcos = agregarMarco(tblMarcos, nuevoMarco); //se agrega la pagina a un marco de pagina
                tblPaginas = busca(tblPaginas, tblMarcos->referencia, tblMarcos->indice);
                //funcion que actualiza a que marco hace referencia la pagina movida
            }else{
                tblMarcos = addFIFO(tblMarcos, nuevoMarco); //añade a la tabla el nuevo marco
                //se hace el algoritmo fifo para sobreescribir el primer marco de pagina
                // printf("%d %d\n", tblMarcos->referencia, tblMarcos->indice);
                tblPaginas = busca(tblPaginas, nuevoMarco->referencia, nuevoMarco->indice);
            }

            break;
        }else{
            printf("Ya se ha movido esta pagina\n");
            //se imprime en caso de que se quiera mover una pagina que ya no esta en la tabla
            break;
        }

    }

case 3:
    return 0; //para salir del programa
}
```

Imagen 15. Parte restante del menú del programa.

Como se ve, la opción 2 es para mover la página, en primer lugar, se solicita al usuario ingresar el índice de la página a mover, si esta es mayor a 15 no se mueve ya que solo tenemos 16 páginas. Si la página de mover si está disponible se procede a verificar si esta página ya se movió o sigue en la tabla de páginas, si sigue en la tabla de páginas y hay espacio en la memoria física se mueve la página, en caso de estar llena la tabla de páginas se remplaza la página. Al ultimo vemos la opción 3 para salir del programa.

Ahora veamos la ejecución y funcionamiento del programa, en la imagen 16 podemos ver su correcta compilación y ejecución.

```
brandon@BDMV:~/programas so/practica 6$ gcc pp61.c -o p61
brandon@BDMV:~/programas so/practica 6$ ./p61

Operaciones con la memoria:
1. Ver tablas
2. Mover pagina a memoria fisica
3. Salir
```

Imagen 16. Compilación y ejecución del programa.

Como se ve, se hace la compilación sin ningún tipo de error ni warning, y en primera instancia se muestra el menú de nuestro programa.

En la imagen 17 se puede ver el resultado de mostrar las tablas antes de hacer cualquier cosa.

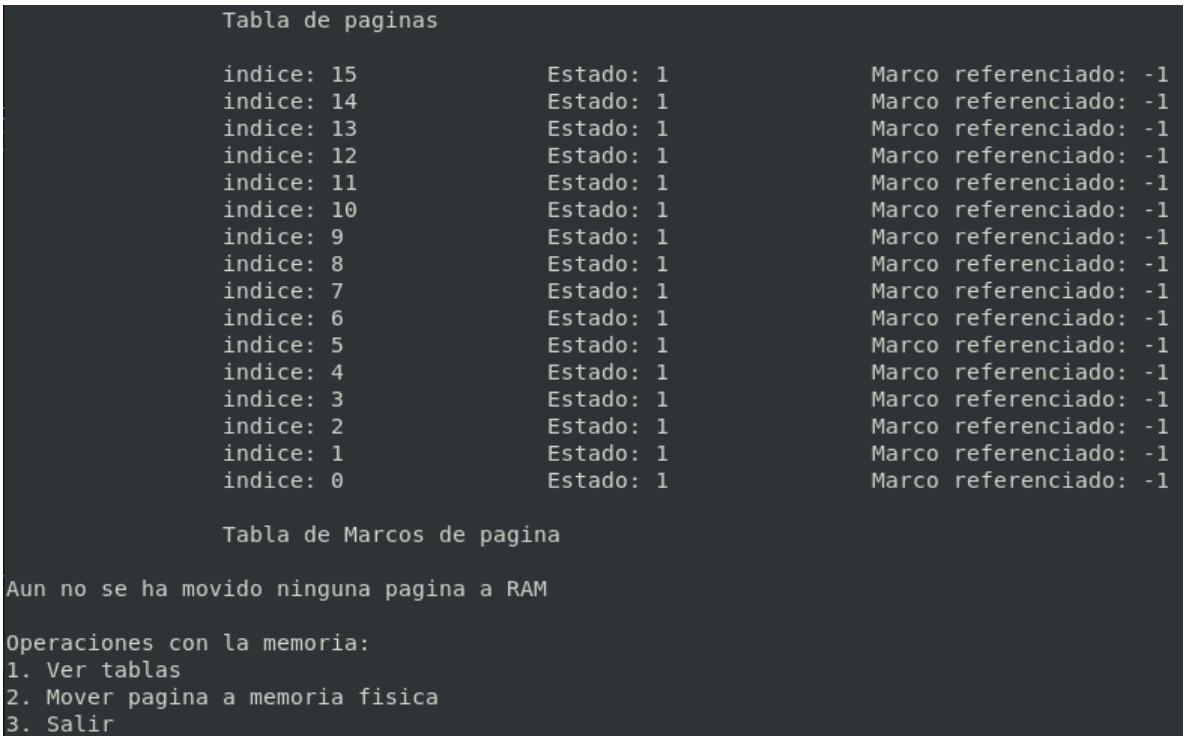


Imagen 17. Mostrando tablas de inicio.

Como vemos, se muestra la tabla de páginas que fue llenada automáticamente, mostrando su índice, el estado y el marco a l que se hace referencia, como al inicio no hacen referencia a ninguno se pone esta parte con un -1. Como no se ha movido ninguna página, la tabla de marcos de página está vacía.

En las imágenes 18 a 20, podemos ver la segunda opción en marcha, moviendo una página a RAM.

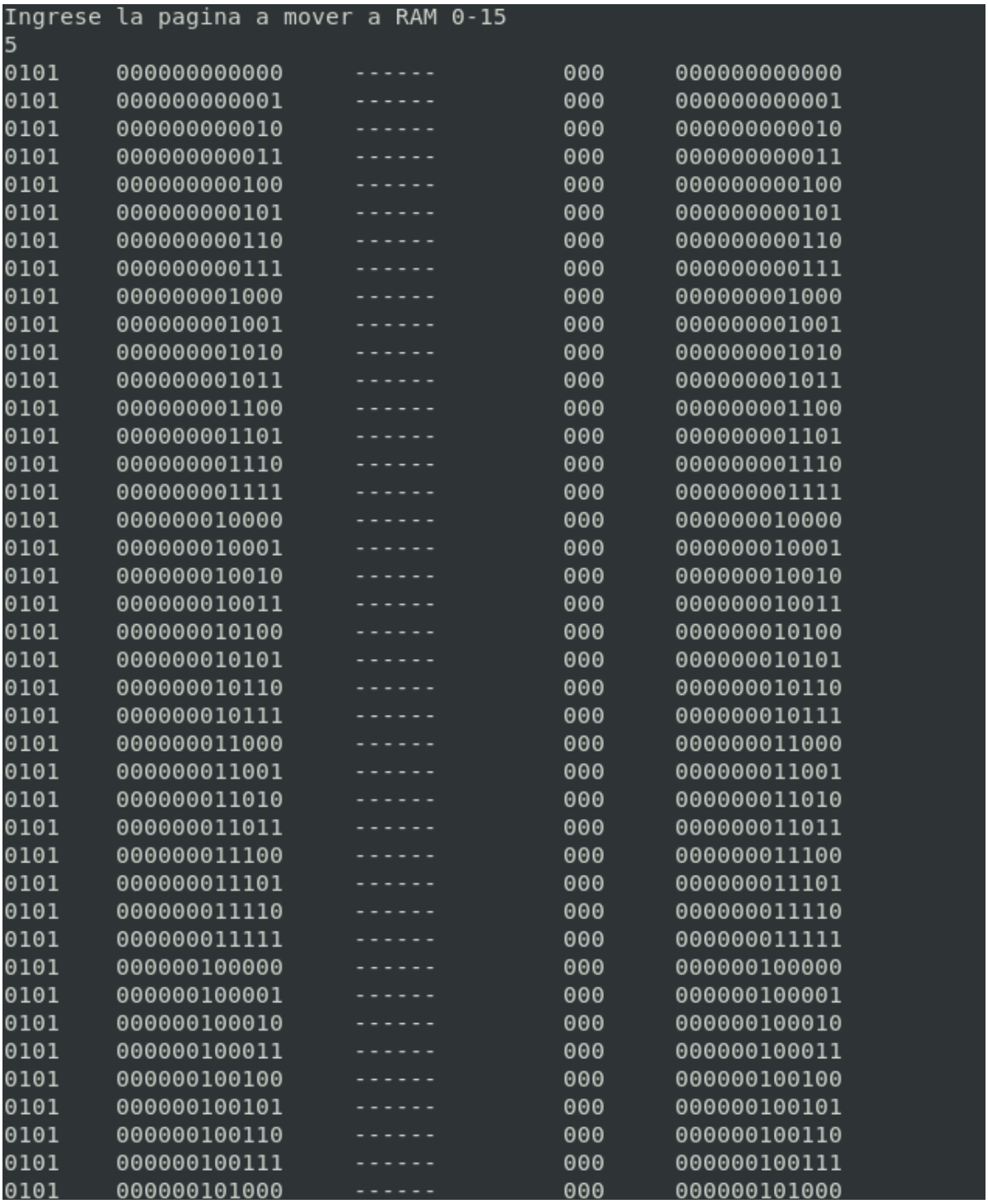
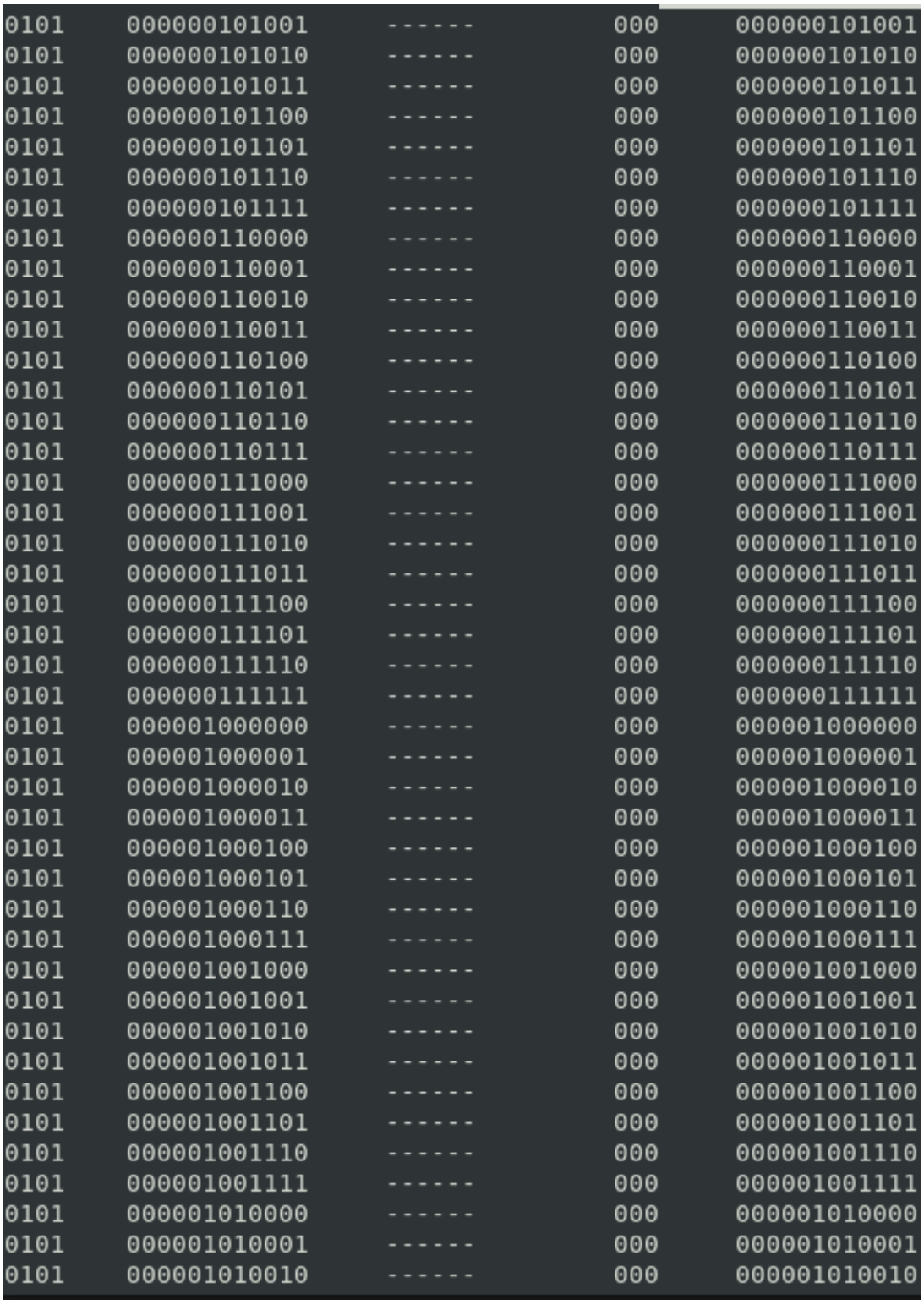


Imagen 18. Desplazamiento de la página a RAM.



```
0101    000000101001    ------    000    000000101001
0101    000000101010    ------    000    000000101010
0101    000000101011    ------    000    000000101011
0101    000000101100    ------    000    000000101100
0101    000000101101    ------    000    000000101101
0101    000000101110    ------    000    000000101110
0101    000000101111    ------    000    000000101111
0101    000000110000    ------    000    000000110000
0101    000000110001    ------    000    000000110001
0101    000000110010    ------    000    000000110010
0101    000000110011    ------    000    000000110011
0101    000000110100    ------    000    000000110100
0101    000000110101    ------    000    000000110101
0101    000000110110    ------    000    000000110110
0101    000000110111    ------    000    000000110111
0101    000000111000    ------    000    000000111000
0101    000000111001    ------    000    000000111001
0101    000000111010    ------    000    000000111010
0101    000000111011    ------    000    000000111011
0101    000000111100    ------    000    000000111100
0101    000000111101    ------    000    000000111101
0101    000000111110    ------    000    000000111110
0101    000000111111    ------    000    000000111111
0101    000001000000    ------    000    000001000000
0101    000001000001    ------    000    000001000001
0101    000001000010    ------    000    000001000010
0101    000001000011    ------    000    000001000011
0101    000001000100    ------    000    000001000100
0101    000001000101    ------    000    000001000101
0101    000001000110    ------    000    000001000110
0101    000001000111    ------    000    000001000111
0101    000001001000    ------    000    000001001000
0101    000001001001    ------    000    000001001001
0101    000001001010    ------    000    000001001010
0101    000001001011    ------    000    000001001011
0101    000001001100    ------    000    000001001100
0101    000001001101    ------    000    000001001101
0101    000001001110    ------    000    000001001110
0101    000001001111    ------    000    000001001111
0101    000001010000    ------    000    000001010000
0101    000001010001    ------    000    000001010001
0101    000001010010    ------    000    000001010010
```

Imagen 19. Desplazamiento de la página a RAM.

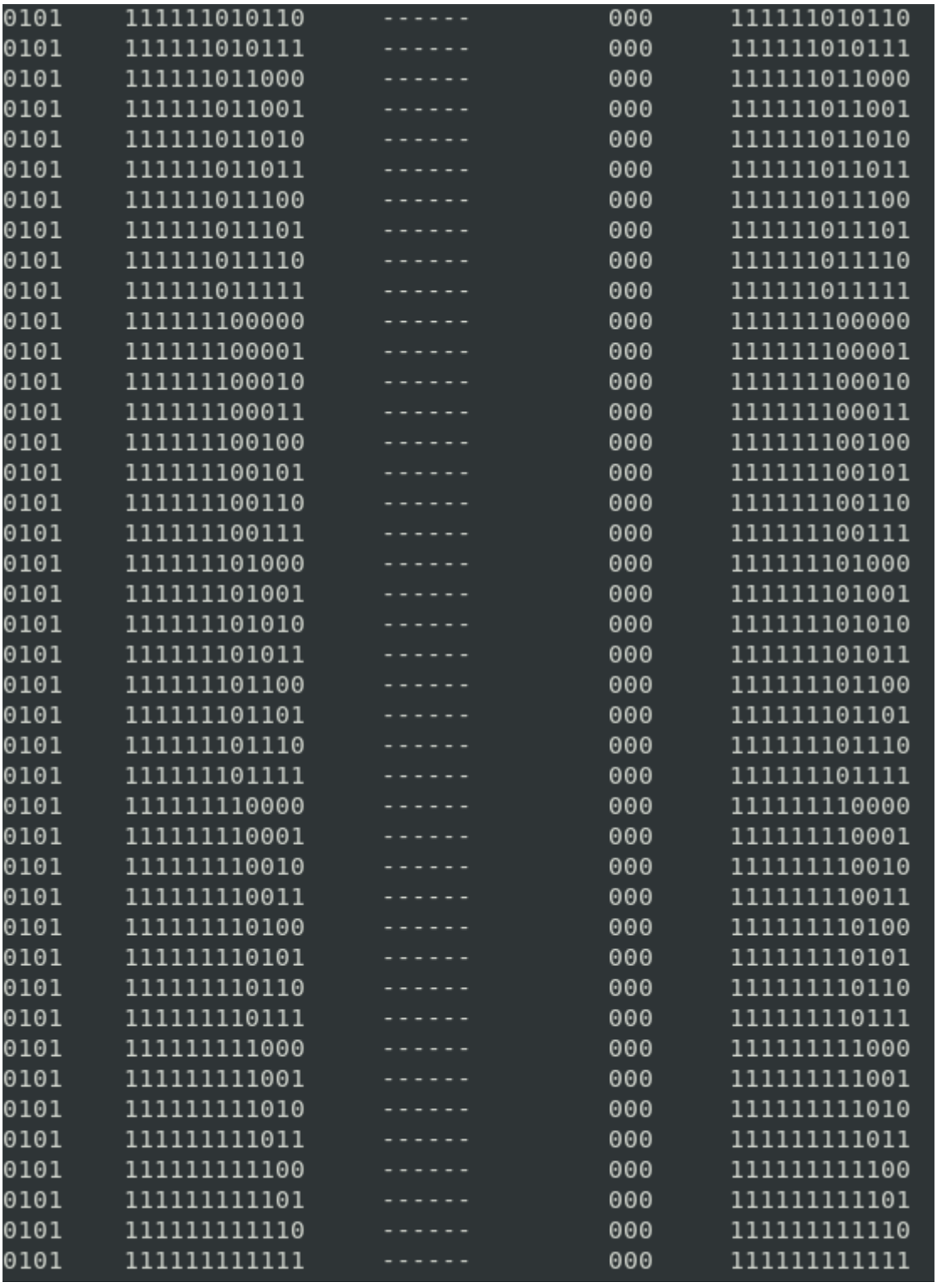


Imagen 20. Desplazamiento de la página a RAM.

Podemos ver en las imágenes anteriores el desplazamiento de las 4096 direcciones de la página, no se mostraron todas, pues son demasiadas, pero en la imagen 18 podemos ver como empieza con su primera dirección que son 12 ceros y en la imagen 20, vemos la última que son 12 unos. Se puede

apreciar que la primera columna es el índice de página, la segunda sus bits de direccionamiento, la tercera solo son guiones para indicar a que marco se está moviendo, la cuarta columna indica el índice de marco al que se está moviendo y la última son los bits de direccionamiento del marco de página.

Se procederá a mover algunas páginas más y en la imagen 21 se muestra la tabla de páginas y el marco de páginas resultantes.

```
Tabla de paginas

indice: 15              Estado: 1               Marco referenciado: -1
indice: 14              Estado: 1               Marco referenciado: -1
indice: 13              Estado: 0               Marco referenciado: 4
indice: 12              Estado: 1               Marco referenciado: -1
indice: 11              Estado: 0               Marco referenciado: 7
indice: 10              Estado: 1               Marco referenciado: -1
indice: 9               Estado: 0               Marco referenciado: 6
indice: 8               Estado: 0               Marco referenciado: 2
indice: 7               Estado: 1               Marco referenciado: -1
indice: 6               Estado: 0               Marco referenciado: 1
indice: 5               Estado: 0               Marco referenciado: 0
indice: 4               Estado: 1               Marco referenciado: -1
indice: 3               Estado: 1               Marco referenciado: -1
indice: 2               Estado: 0               Marco referenciado: 5
indice: 1               Estado: 0               Marco referenciado: 3
indice: 0               Estado: 1               Marco referenciado: -1

Tabla de Marcos de pagina

indice: 7               Pagina referenciada: 11
indice: 6               Pagina referenciada: 9
indice: 5               Pagina referenciada: 2
indice: 4               Pagina referenciada: 13
indice: 3               Pagina referenciada: 1
indice: 2               Pagina referenciada: 8
indice: 1               Pagina referenciada: 6
indice: 0               Pagina referenciada: 5
```

Imagen 21. Movimiento de algunas páginas a RAM.

Como se ve, llenamos el espacio en la tabla de marcos de páginas pues hay 8 marcos dentro. Podemos observar la página a la que hace referencia cada marco, y en la tabla de páginas, se ve el marco al que hace referencia y el estado de las páginas que ya han sido movidas.

En la imagen 22 podemos ver que ocurre cuando queremos mover una página cuando la tabla de marcos de página ya está llena.

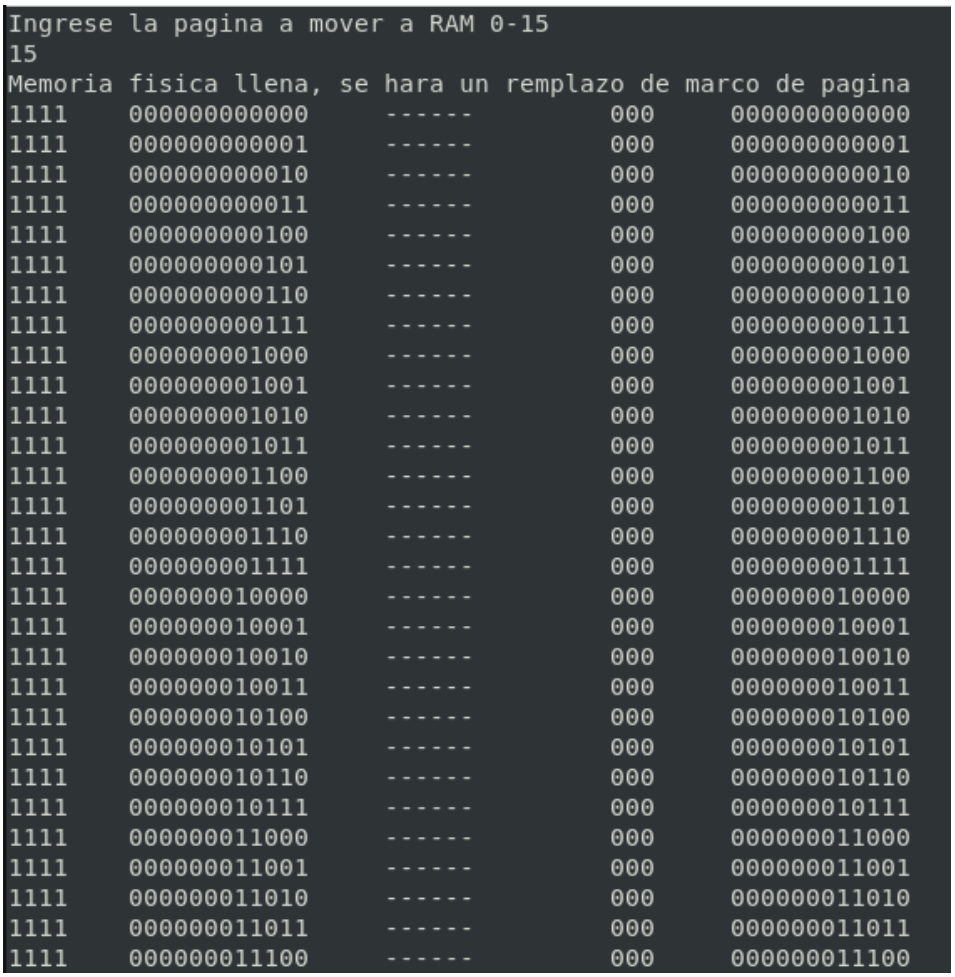


Imagen 22. Moviendo página a tabla de marcos llena.

Como se ve, se indica al usuario que la memoria física está llena y se hará un remplazo de páginas. Abajo podemos ver cómo se van moviendo las direcciones de la página 15 al marco 0, se mueven al 0 ya que este fue el primero en ingresar a la tabla de marcos.

En la imagen 23 vemos de nuevo las tablas ya moviendo algunas páginas más.

```
Tabla de paginas

indice: 15              Estado: 0               Marco referenciado: 0
indice: 14              Estado: 1               Marco referenciado: -1
indice: 13              Estado: 0               Marco referenciado: 4
indice: 12              Estado: 1               Marco referenciado: -1
indice: 11              Estado: 0               Marco referenciado: 7
indice: 10              Estado: 0               Marco referenciado: 0
indice: 9               Estado: 0               Marco referenciado: 6
indice: 8               Estado: 0               Marco referenciado: 2
indice: 7               Estado: 0               Marco referenciado: 0
indice: 6               Estado: 0               Marco referenciado: 1
indice: 5               Estado: 0               Marco referenciado: 0
indice: 4               Estado: 1               Marco referenciado: -1
indice: 3               Estado: 1               Marco referenciado: -1
indice: 2               Estado: 0               Marco referenciado: 5
indice: 1               Estado: 0               Marco referenciado: 3
indice: 0               Estado: 1               Marco referenciado: -1

Tabla de Marcos de pagina

indice: 7               Pagina referenciada: 11
indice: 6               Pagina referenciada: 9
indice: 5               Pagina referenciada: 2
indice: 4               Pagina referenciada: 13
indice: 3               Pagina referenciada: 1
indice: 2               Pagina referenciada: 8
indice: 1               Pagina referenciada: 6
indice: 0               Pagina referenciada: 7
```

Imagen 23. Tablas con movimientos.

Una vez más vemos las tablas actualizadas una vez que se han movido más paginas a la tabla de marcos de página.

En la imagen 24 vemos lo que sucede si queremos mover una página que ya hemos movido anteriormente.

```
Ingrese la pagina a mover a RAM 0-15
2
Ya se ha movido esta pagina

Operaciones con la memoria:
1. Ver tablas
2. Mover pagina a memoria fisica
3. Salir
```

Imagen 24. Mover página que ya se movió antes.

Como se ve, arroja un mensaje de que la pagina ya ha sido movida, es decir, que ya no esta en la tabla de páginas.

Por último, en la imagen 25 vemos la salida del programa.

```
Operaciones con la memoria:
1. Ver tablas
2. Mover pagina a memoria fisica
3. Salir
3
brandon@BDMV:~/programas so/practica 6$
```

Imagen 25. Salida del programa.

Como se ve, ingresamos la opción numero 3 mostrada en el menú del programa para finalizar con la ejecución de nuestro programa.

Conclusiones

Martínez Ramírez Sergi Alberto

En esta práctica pude aprender un poco más acerca de la memoria virtual, ya que me habían quedado un par de dudas de los vídeos, pero con la investigación que realizamos pude resolver varias de mis dudas. Con respecto a la práctica creo que fue lo más complicado de realizar y no por el mal entendimiento del tema (que, si bien pudo afectar, no es el principal motivo de la dificultad) sino porque las herramientas que se utilizaban para realizarla (como las listas enlazadas, definir estructuras, etc.) hace mucho tiempo que no las programaba y tenía dificultades al hacerlo, así que tuve que repasar un poco algunos temas de mi curso de estructuras de datos. Al entender los temas mencionados anteriormente el siguiente problema fue el entender como programar ambas memorias. Sin embargo, me parece que esta práctica fue un reto muy agradable de resolver y sobre todo me hizo ver de una forma más abstracta lo que es la memoria virtual.

Meza Vargas Brandon David

Como es bien sabido la memoria virtual es un recurso muy importante en nuestras computadoras, pues esta nos permite simular un espacio de memoria mayor que el que tiene la memoria física de nuestra computadora y de esta forma hacer que los programas se ejecuten sin tener en cuenta el tamaño exacto de la memoria física y que no tengan problemas en su ejecución.

Con la realización de esta práctica el tema quedo muy claro y lo entendí muy bien gracias al programa solicitado, el cual fue simular la memoria virtual y como se hace la paginación en la memoria.

Fue un programa muy interesante, en principios el programa se me hizo muy confuso y algo complicado de entender, pero con algunas explicaciones e investigaciones por mi cuenta y por parte del equipo me di cuenta de que en realidad no era tan complejo. El mayor problema que tuve, y en general el equipo fue que no recordábamos de manera exacta como trabajar con listas enlazadas en c, pero investigando este tema se resolvió esta problemática.

Hablando sobre el programa realizado, fue un reto, pero se logró, de inicios habíamos pensado en crear una estructura por cada tabla, lo cual al final no fue así, pues nos dimos cuenta de que con una era necesaria. De las cosas que más causaron conflictos fue el saber que páginas y marcos serían referenciados, pero como se vio en el desarrollo de esta práctica se resolvió recorriendo las tablas hasta llegar a una condición para determinar estas referencias.

Durante el desarrollo del programa nos surgió un problema que al menos yo no había visto antes, se trata del error de violación de segmento, esto resulto útil por que al momento de investigar por que sucedía nos dimos cuenta de los errores que cometíamos que hacían aparecer este mensaje. Uno de ellos es que estábamos haciendo un mal uso del operador & y *, apuntando a direcciones de memoria incorrecta, además en la parte de imprimir las tablas este error ocurría por un conflicto en la función printf.

Al ir desarrollando la practica me di cuenta de que algunas funciones eran parecidas y no hubo mucha complicación en esa parte.

También esta práctica me hizo conocer los operadores para trabajar con bits en c, al menos dos, los cuales son << y >>, estos nos ayudan a desplazar bits a la izquierda o la derecha. Nunca había usado estos operadores antes y gracias a esta práctica los pude conocer y aplicar de manera correcta.

En conclusión, esta práctica fue muy útil para comprender al cien el tema de memoria virtual, además de que el investigar ciertas cosas y usar cosas nuevas en lenguaje c me pueden resultar de mucha ayuda en práctica futuras, sin duda una práctica muy completa a pesar de ser solo un programa.

Peña Atanasio Alberto

En general la práctica fue un reto, un tanto difícil, debido a dos aspectos principales, el primero es que ya no recordaba cómo utilizar las estructuras de datos, más específicamente las listas enlazadas, lo cual fue el tema central de esta práctica, por lo anterior tuve que recurrir a mis programas hechos y tutoriales de estructuras de datos. El segundo hecho que complicó el desarrollo de la práctica fue que en los videos de la Unidad 3 no fue explicado ningún código fuente, lo cual torno a dicha práctica en un poco abstracta y teórica. Pero, nada que una sesión de dudas con el profesor y la creación de un código fuente, no pudiese resolver.

Se comprendió, por medio de una simulación, cómo es que interactúan los procesos dentro de la memoria RAM y el Disco Duro y cómo es que estos son esenciales para que el sistema operativo administre la memoria. Todo lo anterior, se pudo implementar por medio de listas enlazadas que simulaban una administración de memoria y una asignación de espacios de memoria a ciertos procesos, tomando en cuenta los huecos y las unidades de asignación de memoria, teniendo muy presente los algoritmos que permitían una optimización, como lo es el FIFO (Primero en entrar, primero en salir). Aquí me llevo una gran lección debido a que mi equipo y yo nos percatamos de una violación de memoria, lo cual es muy similar a querer utilizar una dirección de memoria vacía. Fue una confusión en cuanto al uso de operadores de memoria y la función printf().

También se entendió de mejor manera cómo es que funciona el proceso de la paginación por medio de las tablas de páginas y sus índices, marcos de memoria y sus elementos binarios. En la parte binaria, se volvió a trabajar con los operadores de corrimiento, que en principio costó trabajo entender, pero se recurrió, nuevamente, a programas desarrollados en anteriores semestres.

Considero que con esta práctica logré reconocer la importancia de la memoria virtual en los Sistemas Operativos y comprendí de buena forma cómo ocurre el intercambio de memoria entre la RAM y el Disco Duro, además de la paginación. Para finalmente saber cómo se administra la memoria.

Sarmiento Gutiérrez Juan Carlos

En esta ocasión nos hicimos frente con conceptos sumamente importantes del sistema operativo, en este caso la memoria virtual, la paginación y la segmentación. En mi caso especial no me basto con ver los videos proporcionados por el profesor así que decidí buscar más contenido de forma tal que fuese capaza de complementar la información. En este caso donde más rellene “huecos” fue en la parte de la memoria virtual ya que no comprendía del todo la forma en la que esta se subdividía del disco duro, y después como es que la RAM le proporcionaba una especie de diccionario de rutas para almacenamiento. Además, entendí más con el código las distintas formas en las cuales podemos implementar la paginación, desde definir un algoritmo que nos haga las dos operaciones de reemplazo en el mismo método o recibir un archivo de configuración para definir el tamaño de página que para esta práctica fue de 4Kb, pero pudimos realizar mejoras dinámicas como hacer el tamaño de 8Kb, 2Kb, o incluso 16Kb. Por otra parte, creo que el aprendizaje más fuerte por así decirlo lo obtuve en la parte de la salida del programa ya que hay se veía de forma clara los indicadores básicos tanto en la tabla de páginas como en la tabla de marcos de página tales como el estado, el marco y pagina referenciado etc. Finalmente me di un gran repaso de una herramienta que se utilizó durante la codificación, hablo de las estructuras de datos, para ser más específicos las listas ya que como se dejó

claro en los videos es la mejor y más eficaz estructura para mantener comunicada memoria, procesos, datos y demás información del sistema. Sin duda una práctica muy buena para comprender la memoria virtual y como es que se implementa a un nivel bajo.

Bibliografía

[1] O, Mendoza, "Administración de la memoria y el almacenamiento", 2017. [En línea]. Disponible: http://virtual.umng.edu.co/distancia/ecosistema/odin/odin_desktop.php?path=Li4vb3Zhcy9pbmdlbmllcmlhX2luZm9ybWF0aWNhL3Npc3RlbWFzX29wZXJhdGl2b3MvdW5pZGFkXzQv#slide_6

[2] W, Gunnar, "Administración de memoria: Memoria virtual". [En línea]. Disponible: http://gwolf.sistop.org/laminas/12-memoria-virtual.pdf

[3] P, González, "Listas Enlazadas en C", 2015. [En línea]. Disponible: http://www.pedrogonzalezruiz.net/listas/listas.html

Anexos

Código del programa 61

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MARCOS 8;
#define PAGINAS 16;

typedef struct tabla{ //se crea la estructura de la tabla de marcos y paginas
    int indice; //guarda el indice de pagina o marco
    int referencia; //guarda la pagina o marco al que se hace referencia
    int estado; //para mantener control de la pagina en la tabla 1: esta, 0: no esta
    struct tabla *sig; //estrucutra que apunta a la siguiente pagina o marco
}tabla;


int pgs; //indica el total de paginas en marcos

void bitsPagina(int indiceP){ //funcion que hace la conversión de indice a binario
    int bits=3; //indica los bits del indice de pagina
    while(bits >= 0){ //este ciclo while hace la conversion a binario
        if(indiceP & (((long int)1) << bits)) //si el indice y el recorrido regresan verdadero se regresa 1 si no 0
            printf("1");
        else
            printf("0");
        bits--;
    }
}
void bitsMarco(int indiceM){ //funcion que hace la conversión de indice a binario
    int bits=2; //cantidad de bits del indice de marco
    while(bits >= 0){
        if(indiceM & (((long int)1) << bits))
            printf("1");
        else
            printf("0");
        bits--;
    }
}
void desplazamiento(int n){
    int bits = 11;//cantidad de bits de los bits de desplazamiento
    while(bits >= 0){
        if(n & (((long int)1) << bits))
            printf("1");
        else
            printf("0");
        bits--;
    }
}
void printDesp(int indiceP, int indiceM){ //esta funcion realiza los 4096 direccionamientos
    int i;

    for(i=0;i<4096;i++){ //for que va de 0 hasta las 4096 direcciones de la pagina
        bitsPagina(indiceP);
        printf("\t");
        desplazamiento(i);
        printf("\t ------ \t");
        bitsMarco(indiceM);
        printf("\t");
        desplazamiento(i);
        printf("\n");
    }
}
tabla *iniTabla(){ //funcion que crea la tabla
    return NULL;
}

tabla *crearPag(int i){ //esta funcion crea una pagina
    tabla *aux = (tabla*) malloc(sizeof(tabla)); //reservo memoria para la pagina
    aux->indice = i; //se asigna el indice de las paginas en memoria virtual
    aux->estado = 1; //esta en la tabla de paginas
    aux->referencia = -1; //de inicio no esta en ningun marco de pagina
    aux->sig = NULL; //se apunta a la siguiente pagina
    return aux;
}
tabla *agregarPag(tabla *tblPaginas, tabla *nuevaPag){ //esta funcion agrega la pagina a la tabla de paginas
    if(tblPaginas == NULL){ //si no hay nada dentro de la tabla se agrega la pagina al inicio
        tblPaginas = nuevaPag;
    }else{ //si ya hya una pagina se agrega en el siguiente nodo
        nuevaPag->sig = tblPaginas;
        tblPaginas = nuevaPag;
    }
    return tblPaginas;
}
void mostrarT(tabla *tblPaginas){ //imprime la tabla de paginas ya establecidas al inicio
    tabla *aux = tblPaginas; //usamos aux para recorrer la tabla e imprimirla
    while(aux!=NULL){
        printf("\t\tindice: %d\t\tEstado: %d\t\tMarco referenciado: %d\n", aux->indice, aux->estado, aux->referencia);
```

```
86          aux=aux->sig;
87      }
88  }
89  void mostrarTM(tabla *tblMarcos){ //imprime la tabla de marcos de magina
90      tabla *aux = tblMarcos; //usamos aux para recorrer la tabla e imprimirla
91      if(aux ==  NULL) // para verificar si ya se ha movido una pagina a la memoria fisica
92          printf("Aun no se ha movido ninguna pagina a RAM\n");
93      while(aux!=NULL){
94          printf("\t\tindice: %d\t\tPagina referenciada: %d\n", aux->indice, aux->referencia);
95          aux=aux->sig;
96      }
97  }
98  tabla* FIFO(tabla *tblMarcos, int dire){ //funcion que establece que marco sale y cual entra
99      tabla *aux;
100     if((tblMarcos !=NULL) && (tblMarcos -> indice == 0)){
101         aux = tblMarcos;
102         aux->referencia = dire; //se asigna la nueva pagina a la que el primer marco hara referencia
103         tblMarcos = aux;
104         // printf("pgs %d %d\n", tblMarcos->indice, tblMarcos->referencia);
105         return tblMarcos;
106     }
107 
108 }
109 tabla *insertP(int dire){ //esta funcion solo crea el marco de pagina pero no agrega la pagina
110     if(pgs < 8){
111         tabla *aux = (tabla*) malloc(sizeof(tabla)); //reservo memoria para la pagina
112         aux->indice = pgs; //el indice de marco se va creando conforme las paginas que se van agregando
113         aux->estado = 1; //el estado se vuelve uno si la pagina este en la tabla de marcos
114         aux->referencia = dire; //se asigna el indice de pagina a la que se hace referencia
115         aux->sig == NULL;   //se apunta al siguiente marco
116         pgs++;  //se incrementa la cantidad de paginas en marcos
117         return aux;
118 
119     }else{ //si la memoria fisica esta llena, se procede a realizar un remplazo de paginas por FIFO
120         printf("Memoria fisica llena, se hara un remplazo de marco de pagina\n");
121         tabla *aux = (tabla*) malloc(sizeof(tabla));
122         aux = FIFO(aux, dire);
123         pgs++;
124         return aux;
125     }
126 
127 }
```

```c
tabla *agregarMarco(tabla *tblMarcos, tabla *nuevoMarco){ //aqui se agrega la pagina a un marco de pagina
    if(tblMarcos == NULL) //si la tabla de marcos esta vacia se agrega al inicio
        tblMarcos = nuevoMarco;
    else{//si no esta vacia se agrega en el siguiente nodo
        nuevoMarco->sig = tblMarcos;
        tblMarcos = nuevoMarco;
    }
    return tblMarcos;
}
tabla *addFIFO(tabla *tblMarcos, tabla *nuevoMarco){
    //se sobreescribe el primer marco ya que fue el primero en ingresar
    tabla* ant, *aux;
    aux = tblMarcos;
        while((aux != NULL) && (nuevoMarco->indice != aux->indice)){
            //se hace un while para recorrer los marcos hasta que haya coincidencia
            ant =aux;
            aux = aux->sig; //se van recorriendo los marcos
            // printf("indi %d marco%d\n",aux->indice, nuevoMarco->referencia);
        }
        if(aux!=NULL){
            //cuando se encuentra una coincidencia y no es nula se modifica la referencia
            aux->referencia = nuevoMarco->referencia;
        }
    // printf("indi %d marco%d\n",nuevoMarco->indice,nuevoMarco->referencia);
    // printf("indi %d marco%d\n",tblMarcos->indice,tblMarcos->referencia);
    return tblMarcos;
}


tabla *busca(tabla *tblPaginas, int marco, int indi){ //esta funcion busca el indice de marco al que la pagina hara referencia
    tabla *aux, *ant; //se crean auxiliares para no modificar la tabla original
    if(tblPaginas != NULL &&  (marco == tblPaginas->indice)){ //se verifica si el indice de pagina es el mismo que el de pagina
       // printf("indi %d marco%d\n",tblPaginas->indice, marco);
        aux = tblPaginas; //el auxiliar es para no modificar la tabla original
        aux->referencia = indi; //se modifica el marco a l que la pagina hace referencia
        aux->estado = 0;
        tblPaginas = aux; //ahora si se guarda en la tabla original con la referencia actualizada
        printDesp(tblPaginas->indice, indi);// se realiza el direccionamiento de bits
        return tblPaginas;
    }else{
        aux = tblPaginas;
        while((aux != NULL) && (marco != aux->indice)){ //se hace un while para recorrer las paginas hasta que haya coincidencia
            ant =aux;
            aux = aux->sig; //se van recorriendo las tablas de pagina
            //printf("indi %d marco%d\n",aux->indice, marco);
        }
        if(aux!=NULL){ //cuando se encuentra una coincidencia y no es nula se modifica el marco referenciado
            aux->referencia = indi;
            aux->estado = 0;
            printDesp(aux->indice, indi);// se realiza el direccionamiento de bits
        }

        return tblPaginas; //se regresa la tabla de paginas con los marcos a los cuales se referencian
    }


}
int verifica(tabla *tblPaginas, int dire){ //se verifica si la pagina sigue en la tabla o ya se movio
    tabla *aux, *ant; //auxiliares para no modificar la tabla original
    if(tblPaginas != NULL && (dire == tblPaginas->indice)){ //se verifica el estado de la primera pagina
        if(tblPaginas->estado == 1) //si la pagina esta se regresa un 1 si ya fue movida un 0
            return 1;
        else
            return 0;
    }else{
        while((tblPaginas != NULL) && (dire != tblPaginas->indice)){ //recorre todas las paginas
            tblPaginas = tblPaginas->sig;
        }
        if(tblPaginas != NULL){ //si esta la pagina se regresa un 1 si no un 0
            if(tblPaginas->estado==1)
                return 1;
            else
                return 0;
        }
    }
}


int main(){
    int opc, i, dire, esta;
    tabla *tblPaginas = iniTabla(); //creamos la tabla de paginas
    tabla *tblMarcos = iniTabla();  //creamos la tabla de marcos

```

```c
    for(i=0;i<16;i++){ //se llena la tabla de paginas automaticamente con 16 paginas
        tabla *nuevaPag = crearPag(i); //se crea la pagina
        tblPaginas = agregarPag(tblPaginas, nuevaPag); //se agrega la pagina a la tabla
    }
    //tblPaginas=tblPaginas->sig;
    //printf("1aver %d\n", tblPaginas->indice);

    while(opc >= 0){ //while para mostrar un menu del programa
        printf("\nOperaciones con la memoria:\n");
        printf("1. Ver tablas\n");
        printf("2. Mover pagina a memoria fisica\n");
        printf("3. Salir\n");
        scanf("%d",&opc);
        if(opc > 3)
            printf("Opcion no valida\n");

        switch (opc)
        {
        case 1:
            system("clear");
            printf("\t\tTabla de paginas\n\n");
            mostrarT(tblPaginas); //funcion que imprime la tabla de paginas
            printf("\n\t\tTabla de Marcos de pagina\n\n");
            mostrarTM(tblMarcos);   //funcion que imprime la tabla de marcos
            break;

        case 2:
            system("clear");
            printf("Ingrese la pagina a mover a RAM 0-15\n");
            scanf("%d", &dire);
            if(dire>15){ //solo se pueden mover las paginas que hay en la tabla
                printf("Solo hay 16 paginas en memoria virtual\n");
                break;
            }

            else{
                esta = verifica(tblPaginas, dire); //verifica si la pagina esta en la tabla de paginas
                if(esta == 1){ //si si esta la pagina se procede a mover
                    tabla *nuevoMarco = insertP(dire);

                 if(pgs < 9){
                    tblMarcos = agregarMarco(tblMarcos, nuevoMarco); //se agrega la pagina a un marco de pagina
                    tblPaginas = busca(tblPaginas, tblMarcos->referencia, tblMarcos->indice);
                    //funcion que actualiza a que marco hace referencia la pagina movida
                 }else{
                    tblMarcos = addFIFO(tblMarcos, nuevoMarco); //añade a la tabla el nuevo marco
                    //se hace el algoritmo fifo para sobreescribir el primer marco de pagina
                    // printf("%d %d\n", tblMarcos->referencia, tblMarcos->indice);
                    tblPaginas = busca(tblPaginas, nuevoMarco->referencia, nuevoMarco->indice);
                 }

                 break;
                }else{
                    printf("Ya se ha movido esta pagina\n");
                    //se imprime en caso de que se quiera mover una pagina que ya no esta en la tabla
                    break;
                }

            }

        case 3:
            return 0; //para salir del programa
        }
    }

    return 0;
}
```

Imagen 26. Código del programa 61.