# **Proyecto Final de Sistemas Operativos 2020 - IT**

Este proyecto consiste en escribir un programa pagingdemand.c que traduzca direcciones lógicas a físicas para un espacio de direcciones virtuales de tamaño 2^16 = 65.536 bytes. El programa leerá de un archivo que contiene direcciones lógicas y, utilizando una tabla de páginas, traducirá cada dirección lógica a su dirección física correspondiente y obtendrá como salida el valor en bytes almacenado en la dirección física traducida.

El objetivo de aprendizaje es utilizar la simulación para comprender los pasos necesarios para traducir direcciones lógicas a físicas. Esto incluirá la resolución de fallas de página mediante la paginación por demanda, sin incluir la administración de un TLB y ni la implementación de un algoritmo de reemplazo de página.

**1. Especificaciones**

El programa leerá un archivo que contiene varios números enteros de 32 bits que representan direcciones lógicas. Sin embargo, la traducción debe considerar los 16 bits más a la derecha de cada dirección lógica. Estos 16 bits se dividen en (1) un número de página de 8 bits y (2) un desplazamiento de página de 8 bits. Por lo tanto, las direcciones se estructuran como se muestra a continuación:



La parte en rojo no se debe considerar.

* Otros detalles incluyen lo siguiente:
* 2^8 entradas en la tabla de páginas
* Tamaño de página de 2^8 bytes
* 256 frames
* Memoria física de 65.536 bytes (256 frames × 256 bytes de tamaño del frame)

Además, el programa solo debe preocuparse por leer direcciones lógicas y traducirlas a sus direcciones físicas correspondientes. No es necesario admitir la escritura en el espacio de direcciones lógicas.

Traducción de direcciones

El programa traducirá direcciones lógicas a físicas utilizando solo una tabla de páginas. De manera que, el número de frame se obtiene de la tabla de páginas o se produce un fallo de página (page fault). El diagrama de Arquitectura de arquitectura de traducción de direcciones muestra este esquema. (ver slides9 diapositiva 29)

Manejo de fallos de página

El programa implementará la paginación por demanda. El almacén de respaldo está representado por el archivo BACKING\_STORE.bin, un archivo binario de tamaño 65,536 bytes. Cuando ocurre una falla de página, leerá una página de 256 bytes del archivo BACKING\_STORE y la almacenará en un frame (marco de página) disponible en la memoria física. Por ejemplo, si una dirección lógica con el número de página 15 resultó en un fallo de página, su programa debería leer la página 15 del BACKING\_STORE (recuerde que las páginas comienzan en 0 y tienen un tamaño de 256 bytes) y la almacenaría en un frame en memoria física. Una vez que se almacena este frame se debe actualizar la tabla de páginas, los accesos siguientes a la página 15 se resolverán mediante la tabla de páginas.

Deberá tratar BACKING STORE.bin como un archivo de acceso aleatorio para que pueda buscar aleatoriamente ciertas posiciones del archivo para su lectura. Se recomienda usar las funciones de la biblioteca estándar de C para realizar operaciones de E/S, incluidas: fopen(), fread(), fseek() y fclose().

El tamaño de la memoria física es el mismo que el tamaño del espacio de direcciones virtuales (65,536 bytes), por lo que no debe preocuparse por los reemplazos de página durante un fallo de página. Recuerde que en sistemas reales hay una menor cantidad de memoria física pero no resolveremos esto en este proyecto.

Archivo de prueba

Se proporciona el archivo addresses.txt, que contiene valores enteros que representan direcciones lógicas que van de 0 a 65535 (el tamaño del espacio de direcciones virtuales). El programa debe abrir este archivo, leerá cada dirección lógica y la traducirá a su dirección física correspondiente, y generará el valor del byte almacenado en la dirección física.

Recuerde que en el deber de programación 4 ya se ha solicitado obtener el número de página y el desplazamiento (offset) por lo que puede incluir esta parte en el programa solicitado. Tanto para la tabla de páginas como la memoria principal se recomienda utilizar arreglos de 256 elementos y para acceder al archivo binario el uso de un puntero a file de la librería estándar de C.

**2. Especificaciones de las salidas**

El programa debería ejecutarse de la siguiente manera, ya no use ./build:

|  |
| --- |
| ./pagingdemand addresses.txt data.txt |

Donde address.txt es el nombre del archivo que contiene las direcciones lógicas y data.txt es el nombre del archivo de salida que contiene lo que se describe más abajo.

El programa leerá el archivo address.txt, que contiene N direcciones lógicas que van de 0 a 65535. También, debe traducir cada dirección lógica a una dirección física y determinar el contenido del byte almacenado en la dirección física traducida. (Recuerde que en el lenguaje C, el tipo de dato char ocupa un byte de almacenamiento, por lo que se sugiere usar este tipo).

Este programa deberá generar la salida en un archivo de texto (en el ejemplo el archivo es de nombre data.txt). El formato del archivo de salida es el siguiente: cada línea es la traducción de una dirección virtual (véase la figura la figura), y contiene:

|  |
| --- |
| Virtual address: 16916 Physical address: 20 Value: 0  Virtual address: 62493 Physical address: 285 Value: 0  Virtual address: 30198 Physical address: 758 Value: 29  Virtual address: 53683 Physical address: 947 Value: 108  Virtual address: 40185 Physical address: 1273 Value: 0  ... |

1. La dirección lógica que se está traduciendo (el valor entero que se lee de address.txt).
2. La dirección física traducida (la traducida por el programa).
3. El valor del byte almacenado en la memoria física en la dirección traducida.

En repositorio del proyecto encontrará el archivo correct.txt, que contiene los valores de salida correctos para el archivo address.txt, que también estará disponible. Debe utilizar este archivo para determinar si su programa está traduciendo correctamente direcciones lógicas a físicas.

**Entregables**

Siga todos los requisitos y pautas de presentación exactamente como se describe arriba, entre otras cosas: estructurar y comentar adecuadamente el código, sin incluir el código objeto o ejecutables en su envío, nombrando el archivo MAKE exactamente makefile. Además del programa deberá incluir un reporte que debe describir los antecedentes del problema, el uso, parámetros del programa, limitaciones que tuvo al desarrollarlo, cómo resolvió estas limitaciones y mostrar salidas de pantallas con las pruebas realizadas para comprobar el funcionamiento del programa.

Sus programas deben compilarse con gcc versión 7 o superior sin warnings y ejecutarse correctamente en Ubuntu Linux, o alguna otra distribución basada en UNIX. Habrá una penalización si no corre en Ubuntu, de manera que deben asegurarse que corra en esta distribución. Coloque todos los archivos y documentos necesarios en un archivo comprimido usando tar y gzip (.tar.gz o .tgz) o, alternativamente, un archivo zip (.zip) (no utilice el formato rar, bzip2). OJO: no comprima directamente la carpeta que contiene los archivos solicitados, sino que seleccione los archivos y a partir de la selección cree el comprimido.

Subir este archivo en la tarea de Sidweb correspondiente antes de la fecha límite anterior. Las entregas tardías se marcarán de acuerdo con la política de retrasos publicada en la página web del curso.

No muestre, intercambie ni copie código con nadie, y no busque una solución en la Web! El plagio será detectado y severamente penalizado. Esto debe seguir siendo un esfuerzo individual.

Mantenga convenciones de organización, diseño y codificación del código, ya que estas también serán parte importante de la nota. Todos sus fuentes deberán tener datos del autor y fecha de creación, entre otras cosas y deben incluirse en carpeta raiz. Si utiliza código externo, es decir no desarrollado por usted, y que lo incluya en las directivas, deberá colocarlo dentro de en una carpeta de nombre /utils.

¡Gracias y buena suerte!