Proyecto I:Visualizador de Datos con Ordenamiento

Ronald Duarte Barrantes
Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
2021004089
Sede central Cartago
ronald@estudiantec.cr

I. Introducción

Los lenguajes ensambladores son parte fundamental en la creación de software para hardware, debido a su optimización y su compilación más veloz, lo cual genera algoritmos más eficientes en el uso de recursos y un manejo de valores de memoria y direcciones más rápido y preciso. En este caso se emplea el lenguaje ensamblador x86, el cual es de tipo CISC, lo que significa que presenta un tamaño de instrucciones mayor y variable según el contexto en que se utilice. Además, las instrucciones de lectura y guardado de memoria se encuentran de manera implícita en las instrucciones, sin necesidad de hacer un llamado explícito a las mismas. Para ello se emplean herramientas como Linux 64 y NASM, para la depuración del código.

En cuanto al visualizador de datos con ordenamiento, cumple la tarea de generar una representación gráfica por medio de un histograma con las configuraciones que se obtienen en un archivo config.ini, permitiendo observar los valores que se encuentran dentro de una lista de inventario en orden alfabético, y mostrando su respectiva cantidad en las barras del histograma.

II. Objetivo

La tarea tiene como objetivo diseñar y comprender el desarrollo de un visualizador de datos en orden alfabético, desarrollado en un entorno Linux 64 y mediante el lenguaje ensamblador x86.

III. Instrucciones

Para el desarrollo eficiente del proyecto, se deben tener en cuenta las siguientes caracteristicas fundamentales de diseno que se emplearon en el desarrollo del código.

1. Leer y procesar los valores de config.ini
2. Leer y procesar los valores de inventario.txt
3. Ordenar los datos del inventario alfabéticamente.
4. Dibujar el gráfico usando los datos ya ordenados y los parámetros de configuración.
    - caracter barra:*
    - color barra:94
    - color fondo:40

IV. Funcionamiento del código del Visualizador

En este apartado se describe el desarrollo del código utilizado para la implementación del visualizador, estructurado de manera modular de acuerdo con la función específica que desempeña cada componente del programa.

IV-A. Definición de Datos y Buffers

IV-A1. Seccion.data: En este apartado, el objetivo principal es declarar los mensajes para el manejo de errores, así como los nombres de los archivos a utilizar, como config.ini e inventario.txt, para posteriormente ser leídos y guardar ciertos caracteres necesarios para la generación del histograma en la sección .data, ya que son valores definidos.

```
section .data
    listatxt        db "inventario.txt", 0
    configtxt       db "config.ini", 0

    error_config    db 'No␣se␣encontro␣el␣archivo␣de
        ␣configuracion',0xa
    l_error_config  equ $-error_config

    error_inventario db 'No␣se␣encontro␣el␣archivo␣
        de␣inventario',0xa
    l_error_inventario equ $-error_inventario

    dos_puntos_msg      db ":", 0
    espacio_msg         db "␣", 0
    nueva_linea_msg     db 10, 0

    ansi_esc            db 0x1B
    ansi_open           db "[", 0
    ansi_m              db "m", 0
    ansi_reset_completo db 0x1B, "[0m", 0
    ansi_reset_len      equ $ - ansi_reset_completo
```

IV-A2. Seccion.dss:

IV-B. Reservas de Memoria y Buffers

En este apartado se reservan espacios de memoria que serán utilizados por las funciones del visualizador para almacenar información relevante del archivo de configuración y del inventario. Se incluyen buffers específicos para la creación del histograma, que permiten guardar y recuperar valores de manera eficiente, así como descriptores de archivo para manejar la lectura de los archivos externos.

Se destacan los siguientes elementos:

- **fd, fd1**: Descriptores de archivos que almacenan la dirección de `config.ini` y `inventario.txt`, respectivamente.

- **buffer_confg**: Almacena en memoria el contenido completo leído desde `config.ini`.
- **datos_config**: Contiene únicamente los valores de interés extraídos de `buffer_confg`.
- **buffer_lista**: Guarda el contenido completo del archivo `inventario.txt`.
- **Buf_Nombrefruta**: Almacena los nombres de las frutas extraídos de `buffer_lista`.
- **Buf_cantidadfruta**: Contiene las cantidades correspondientes a cada fruta.
- **Buf_numfrutas**: Registra el total de frutas leídas.
- **buf_num_ascii**: Buffer utilizado para almacenar los valores leídos de `config.ini` en formato decimal.
- **caracter_barra**: Almacena el carácter que formará cada barra del histograma.
- **color_barra**: Contiene el color asignado a las barras del histograma.
- **color_fondo**: Contiene el color de fondo del histograma.
- **char_buffer, num_buffer, byte_buffer**: Buffers auxiliares para el almacenamiento temporal de caracteres y números durante la construcción del histograma.

```
section .bss
    buffer_confg      resb 512
    fd                resq 1
    datos_config      resb 100*100
    l_datos_config    resb 64
    buffer_lista      resb 1024
    fd1               resq 1
    datos_lista       resb 64
    l_datos_lista     resb 64

    Buf_Nombrefruta   resb 512
    Buf_frutas_O      resb 512
    Buf_numfrutas     resb 512
    Buf_cantidadfruta resq 100

    buf_num_ascii     resb 64

    caracter_barra     resb 1
    color_barra        resb 1
    color_fondo        resb 1
    char_buffer        resb 1
    num_buffer         resb 20
    byte_buffer        resb 4
```

IV-C. Módulo de Inicio (start)

En este modulo se inicia las variables globales, basicamente hay tres variables globales que tienen como utilidad permitir ver el funcionamiento del del código en los registros mediante la utilización del gdb y la función break ¡variable global¿.

Con respecto al código, en este apartado se identifican tres procesos de gran relevancia. El primero realiza la lectura del archivo `config.ini` y verifica si ocurre algún error durante dicha operación. Posteriormente, se lleva a cabo la lectura del archivo `inventario.txt`, la cual implementa una lógica similar a la empleada en la lectura de `config.ini`, aunque cambiando la posición de memoria en la que se almacenan los datos obtenidos y las llamadas a funciones.

IV-C1. Implementación de la Lectura de Archivos: Para la implementación de la lectura de ambos archivos, se toma en cuenta la dirección en memoria en la que se encuentran mediante el uso de punteros y registros especiales. El procedimiento se realiza a través de llamadas al sistema (`syscalls`) de Linux en lenguaje ensamblador `x86-64`, lo que permite una comunicación directa con el sistema operativo.

En primer lugar, se abre el archivo `config.ini` utilizando la llamada `sys_open` (`mov rax, 2`). El registro `rdi` recibe la dirección del nombre del archivo (`configtxt`) y `rsi` especifica el modo de apertura como solo lectura (`O_RDONLY = 0`). Después de ejecutar la instrucción `syscall`, el descriptor de archivo devuelto por el sistema operativo se almacena en la variable `fd`. Si el valor retornado en `rax` es negativo, el programa finaliza para evitar errores.

Posteriormente, la lectura del contenido se efectúa mediante la llamada `sys_read` (`mov rax, 0`). El descriptor almacenado en `fd` se carga en `rdi`, el buffer `buffer_confg` en `rsi` como destino de los datos y `rdx` indica el tamaño máximo de lectura (512 bytes). Tras ejecutar nuevamente `syscall`, el contenido del archivo queda almacenado en el buffer. Una vez verificado que no existan errores.

La lectura del archivo `inventario.txt` sigue una lógica similar, variando únicamente el nombre del archivo y la posición de memoria en la que se almacena la información, permitiendo así manejar de manera independiente los datos de configuración y los datos de inventario.

IV-C2. Llamadas a funciones: Una parte fundamental para el orden del código es la utilización de *subrutinas*, las cuales permiten generar, por así decirlo, un diseño modular facilitando la creación de bloques de código independientes para tareas específicas en el programa. Esto facilita crear bloques de código para tareas específicas en secciones separadas del `main`, pero que pueden ser llamados cuando se necesiten y reutilizados. En este caso se implementan cinco subrutinas principales, las cuales cumplen papeles esenciales en la creación del proyecto:

- **datos_configuración**: Procesa y organiza la información extraída del archivo `config.ini`.
- **GuardarNombresFrutas**: Almacena en memoria los nombres de las frutas leídas desde `inventario.txt`.
- **Cantidad_fruta**: Registra la cantidad correspondiente a cada fruta detectada en el inventario.
- **OrdenarxNombre**: Ordena alfabéticamente los nombres de las frutas y su respectiva cantidad para su correcta visualización.
- **GenerarHistograma**: Construye el histograma final utilizando los datos procesados y los parámetros de configuración.

La correcta interacción de estas subrutinas garantiza un flujo de trabajo ordenado y modular, facilitando la lectura, depuración y mantenimiento del código ensamblador.

IV-D. Procesamiento de Configuración

En la subrutina **datos_configuración**, se realiza la extracción de los valores definidos en el archivo `config.ini`, los cuales son esenciales para parametrizar el histograma (carácter de barra, color de barra y color de fondo). El

proceso inicia apuntando el registro RSI al espacio de memoria buffer_confg, donde previamente se almacenó el contenido del archivo config.ini, y el registro RDI al espacio datos_config, que servirá como destino de los valores extraídos.

El algoritmo recorre el buffer byte por byte hasta localizar el carácter ':', el cual marca el inicio del dato que se desea obtener. Una vez detectado, se avanza al siguiente byte y comienza la lectura del valor correspondiente, diferenciando entre dos posibles tipos de datos:

- **Valores numéricos**: cuando los caracteres se encuentran en el rango ASCII de los dígitos, se invoca la subrutina ascii_decimal para convertir el número de su representación en texto a su equivalente decimal. Esta conversión se realiza acumulando los dígitos en RAX mediante multiplicaciones sucesivas por 10 y sumando cada nuevo dígito convertido.
- **Valores de tipo carácter**: si el byte leído no corresponde a un dígito, se guarda directamente en datos_config sin conversión, permitiendo almacenar símbolos como el carácter de barra o códigos de color.

El proceso se repite para cada línea del archivo hasta llegar a un salto de línea (0Ah) o al final del buffer (NULL).

Cabe recalcar que la importancia de convertir los códigos de ascci a decimal, es debido a la facilidad en un futuro de poder ordenarlo, por ello se tiene la siguiente función. La subrutina ascii_decimal convierte una cadena de dígitos en formato *ASCII* a su valor decimal. Primero limpia el registro RAX para usarlo como acumulador. Luego, lee byte a byte desde el buffer apuntado por RSI, deteniéndose si encuentra un espacio, un salto de línea o el fin de cadena. Cada carácter numérico se convierte restando '0' para obtener su valor real, se expande a 64 bits y se suma al acumulador, que previamente se multiplica por 10 para desplazar las cifras. Al finalizar, el resultado queda en el byte menos significativo (AL), listo para su uso en el programa.

IV-E. Lectura de Inventario

IV-E1. GuardarNombresFrutas: Para leer el archivo inventario.txt, se utilizan dos elementos clave en el formato (nombre:cantidad). El objetivo principal es separar los nombres de las frutas de sus cantidades y almacenarlos en la memoria temporal Buf_Nombrefruta. Para ello, la dirección de memoria que hace referencia al archivo se guarda en buffer_lista, donde además se inicializa un contador para registrar la cantidad de frutas. Es importante considerar que el separador es ":", ya que todo el texto anterior a este carácter corresponde al nombre de la fruta.

La subrutina GuardarNombresFrutas se encarga de registrar únicamente los nombres presentes en el archivo inventario.txt, ignorando las cantidades. En este proceso, buffer_lista funciona como entrada y Buf_Nombrefruta como área de salida.

El procedimiento inicia controlando los registros RSI (entrada) y RDI (salida), además de limpiar el contador R11, que lleva la cuenta de las frutas leídas. Luego se recorre el contenido del buffer carácter por carácter, copiando cada símbolo al búfer de salida hasta encontrar el delimitador ":" (o un salto de línea), lo que indica el final del nombre. Una vez detectado este carácter, se inserta un NULL para finalizar la cadena y se incrementa el contador de frutas.

IV-E2. Cantidad fruta: El procedimiento es similar al de los subprogramas GuardarNombresFrutas y datos_C, donde se reutilizan varias funciones. En este caso, el objetivo principal es extraer los valores ubicados detrás del carácter ":" y convertirlos a formato decimal, facilitando su ordenamiento posterior mediante el algoritmo *Bubble-Sort*. Dado que la lógica para la lectura y discriminación de datos es casi la misma que en las rutinas anteriores, no se profundiza en el análisis del código.

En esta subrutina, denominada Cantidad_fruta, se recorre el contenido de buffer_lista para identificar los valores numéricos que aparecen después del delimitador ":". Una vez localizado, se invoca la función ascii_decimal_cantidad para convertir la cantidad en formato ASCII a un número decimal, el cual se almacena en el búfer Buf_cantidadfruta como un dato de 8 bytes. El proceso se repite para cada línea del archivo hasta que todas las frutas han sido leídas, incrementando un contador que registra la cantidad total de elementos procesados.

IV-F. Ordenamiento Alfabético (Ordenamiento_bubble_sort)

Una de las secciones de mayor relevancia corresponde al reordenamiento de los nombres de las frutas y sus respectivas cantidades. Es importante destacar que, cuando se realiza el reacomodo de las cantidades, este depende directamente del nuevo orden alfabético obtenido a partir de los nombres. Para lograrlo, se utilizan los buffers Buf_cantidadfruta y Buf_Nombrefruta, asegurando que cada cantidad permanezca asociada a su fruta correspondiente después del proceso de ordenamiento. Este bloque implementa un **bubble sort** para ordenar las frutas alfabéticamente, asegurando que las cantidades sigan el mismo orden. Primero se prepara el entorno guardando registros y cargando los buffers de nombres (Buf_Nombrefruta) y cantidades (Buf_cantidadfruta), junto con el total de frutas. El ciclo externo reduce el rango de comparación en cada pasada, mientras que el interno compara dos nombres contiguos usando comparar_strings. Si el nombre actual es mayor, se llaman las rutinas intercambiar_nombres y intercambiar_cantidades para mantener la relación nombre–cantidad. El algoritmo termina antes si no hay intercambios, lo que indica que ya está todo ordenado.

Las funciones auxiliares realizan lo siguiente:

- comparar_strings: compara carácter por carácter para definir el orden alfabético.
- intercambiar_nombres: recorre los 16 bytes de cada nombre e intercambia su contenido.
- intercambiar_cantidades: realiza el intercambio de las cantidades (8 bytes) correspondientes.

Para ordenar las frutas alfabéticamente se utiliza el algoritmo *Bubble-Sort*, que compara pares de nombres cercanos

dentro del búfer `Buf_Nombrefruta` (bloques de 16 bytes). Si el nombre actual es mayor que el siguiente, se llama a la rutina `intercambiar_nombres` para cambiar su posición. Bubble Sort ordena una lista comparando elementos contiguos y moviendo los mayores progresivamente al final. [2]

Paralelamente, cada vez que se intercambian dos nombres, se llama a `intercambiar_cantidades`, que realiza el mismo procedimiento para `Buf_cantidadfruta` (bloques de 8 bytes). Esto garantiza que las cantidades asociadas a cada fruta permanezcan correctamente sincronizadas con su nombre después de la clasificación.

IV-G. Generación del Histograma (GenerarHistograma)

Como paso final, se utilizan todas las sub-rutinas vistas anteriormente para la creación del histograma en orden alfabético y de colo cían, marcando el crecimiento de la barra de frutas con *, según lo que pide en el archivo config.ini y según las frutas que se encuentren en inventario.txt.

La subrutina `GenerarHistograma` crea un histograma textual de manera directa y clara. Primero, se cargan los nombres de las frutas desde `Buf_Nombrefruta` y sus cantidades desde `Buf_cantidadfruta`, junto con la configuración de colores y caracteres definidos en `datos_config`. Luego, se recorre cada fruta en un bucle, imprimiendo su nombre, un separador "`:`" y aplicando los colores mediante códigos ANSI.

Para representar la cantidad de cada fruta, se imprime un carácter de barra repetido tantas veces como indica el valor almacenado en `Buf_cantidadfruta`. Si la cantidad es cero, se omite la barra. Tras imprimir las barras, se restablecen los colores de la terminal y se imprime el número correspondiente a la cantidad. [1]

El proceso se repite fila por fila hasta procesar todas las frutas. Internamente se utilizan funciones auxiliares como `imprimir_string`, `imprimir_caracter`, `imprimir_numero`, y `imprimir_codigos_ansi_corregido` para controlar la salida de texto y colores, garantizando que cada barra y número se impriman correctamente en la misma línea correspondiente a su fruta. Así, cada línea del histograma combina el nombre, la barra indicativa de cantidad y el valor numérico, generando una representación visual sencilla y ordenada.

IV-H. Manejo de Errores y Salida

Este apartado permite verificar el progreso en el manejo de los *buffers*, confirmando que los valores se almacenan correctamente. Incluye además un control de errores para asegurar la correcta apertura de los archivos a analizar, lo cual resulta fundamental. Finalmente, se ejecuta la rutina de salida del sistema. Esta sección del código implementa el manejo de errores y la verificación de datos en memoria. Los bloques `Fin_programa` y `Fin_programa_dos` muestran mensajes en pantalla (`sys_write`) cuando ocurre un fallo en la configuración o en la apertura del archivo de inventario, y luego redirigen la ejecución a la salida del sistema.

La parte marcada como `funcional` (comentada) sirve para imprimir en consola distintos *buffers* de trabajo, como los datos de configuración, el inventario completo, los nombres de las frutas y sus cantidades, permitiendo comprobar que la información se guarda correctamente en memoria.

V. Flujo del sistema

Se presenta un diagrama con el flujo que sigue el programa para la ejecución correcta del código, el cual permite el ordenamiento e impresión de los datos en pantalla.

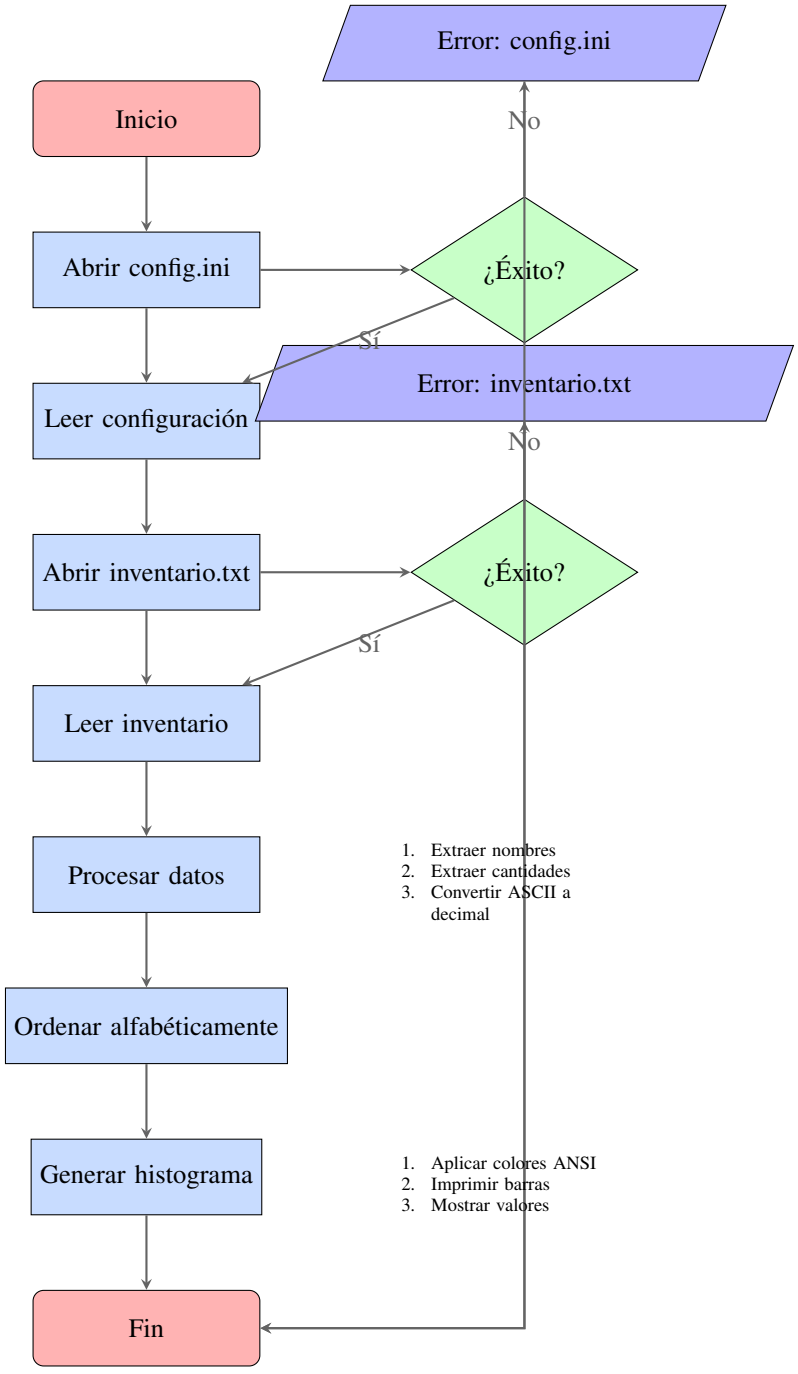

Figura 1. Diagrama de flujo del sistema de procesamiento de inventario

VI. Funcionamiento

Para comprobar el funcionamiento, se deben tomar en cuenta los siguientes elementos y su comportamiento en el histograma.

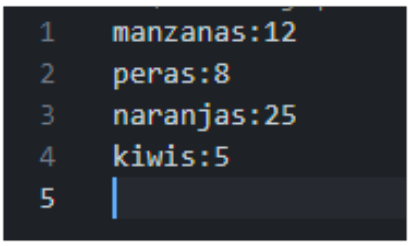

Figura 2. Inventario de las frutas

En este apartado se presenta el inventario mostrado en la Fig. VI para ordenar, por lo cual es importante recalcar que al visualizarlo se tendrá el siguiente resultado: kiwis, manzanas, naranjas y peras. Esta es la distribución correcta, y cada ordenamiento de frutas tendrá su respectiva cantidad, representada por un asterisco (*) y una barra.

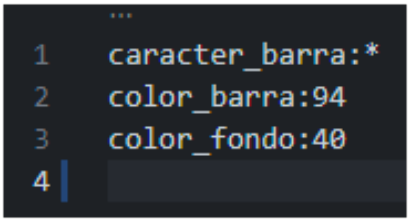

Figura 3. Configuración del histograma

La configuración vista en la fig.VI representa los parámetros a seguir de nuestro visualizador, por ejemplo presenta una barra creciente compuesta por * y con un color celeste, con un fondo negro y al final se pueden apreciar los valores numéricos que representan la cantidad de las frutas.

Figura 4. Resultado final del visualizador

La configuración mostrada en la Fig. VI representa los parámetros a seguir en nuestro visualizador. Por ejemplo, presenta una barra creciente compuesta por asteriscos (*) de color celeste sobre un fondo negro.

VII. Conclusión

- En el análisis, el orden y el uso adecuado de los espacios de memoria y de los registros son esenciales al diseñar un código en lenguaje ensamblador, ya que permiten almacenar correctamente los valores a analizar.
- El algoritmo de ordenamiento Bubble Sort, por su simplicidad y eficiencia, es adecuado para esta implementación, ya que permite trabajar tanto con valores numéricos como con cadenas de texto.
- El diseño del histograma se puede modificar dependiendo de las frutas que haya y de los valores que se introduzcan en el archivo de configuración, generando una gran versatilidad y reutilización.

VIII. Repositorio

A continuación se presenta el repositorio donde se almacena el código correspondiente a la implementación de este proyecto. Dentro de la carpeta `code_asm`, en el subdirectorio `codigo_final`, se encuentra el archivo `proyecto_1.asm`.

https://github.com/Ronald-exe/Proyecto$_u no_V isualizador$

Referencias

[1] M. Poostchi, K. Palaniappan, D. Li, M. Becchi, F. Bunyak, and G. Seetharaman, “Fast integral histogram computations on GPU for real-time video analytics,” arXiv preprint arXiv:1711.01919, 2017.

[2] R. Sedgewick, *Algorithms in C*, 3rd ed., Reading, MA: Addison-Wesley, 1998, pp. 123–130.