Curso: Arquitectura de computadoras I

Proyecto 1: Visualizador de Datos con Ordenamiento para Linux

PROFESOR

Ronny Giovanni García Ramírez

Grupo #1

ESTUDIANTE
Oscar David Conejo Cantón 2020234423

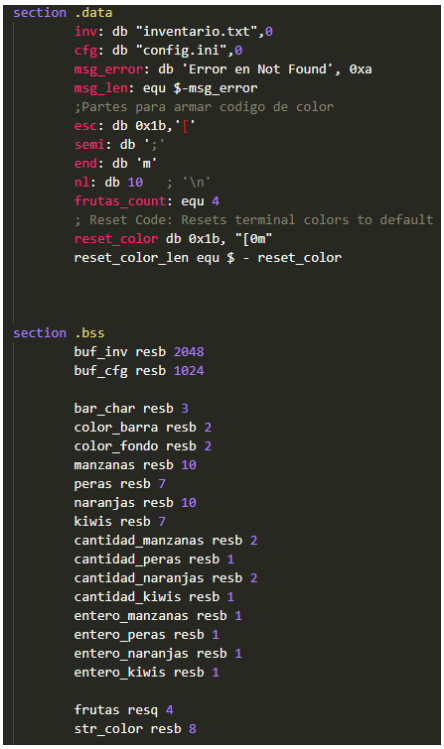
II semestre 2025

Introducción:

El siguiente proyecto presenta el desarrollo de un programa en esamblador x86 utilizando la herramienta de NASM en el entorno de Linux. Este programa procesa dos archivos de texto. En donde el primero contiene datos de inventario que deben ser leídos, ordenados alfabéticamente y ser visualizados en un grafico de barras. El segundo archivo proporciona los parámetros de configuración para visualización.

Datos y variables del programa:

En la Fig.1 se presentan las principales variables y datos que se utilizan a lo largo del programa. Estas se dividen primero en la sección .data. La cual se encarga de definir las variables que ya se conocen y aquí mismo se les puede calcular el tamaño de estas variables. Después esta la sección .bss, la cual se encarga de establecer variables que aun no se conocen. Para esto se les aloja un tamaño predeterminado de bytes a cada variable. Esto para reservar el espacio de cada una en memoria.



```
section .data
        inv: db "inventario.txt",0
        cfg: db "config.ini",0
        msg_error: db 'Error en Not Found', 0xa
        msg_len: equ $-msg_error
        ;Partes para armar codigo de color
        esc: db 0x1b,'['
        semi: db ';'
        end: db 'm'
        nl: db 10    ; '\n'
        frutas_count: equ 4
        ; Reset Code: Resets terminal colors to default
        reset_color db 0x1b, "[0m"
        reset_color_len equ $ - reset_color



section .bss
        buf_inv resb 2048
        buf_cfg resb 1024

        bar_char resb 3
        color_barra resb 2
        color_fondo resb 2
        manzanas resb 10
        peras resb 7
        naranjas resb 10
        kiwis resb 7
        cantidad_manzanas resb 2
        cantidad_peras resb 1
        cantidad_naranjas resb 2
        cantidad_kiwis resb 1
        entero_manzanas resb 1
        entero_peras resb 1
        entero_naranjas resb 1
        entero_kiwis resb 1

        frutas resq 4
        str_color resb 8
```

Figura 1: Section .data y Section .bss

Explicación de rutinas principales:

Extracción de datos de documentos:

En esta pequeña rutina se utilizan los system calls para para poder lograr que el programa extraiga los datos de un documento externo. En esta sección se extrae información de dos documentos: config.ini e inventario.txt. Ambos se hacen de la misma manera, pero caen en diferentes variables para su almacenamiento. Lo primero que se necesita es hacer el system open para poder abrir el archivo de interés. Luego se utiliza el system read, que permite leer y almacenar la información del archivo. Y finalmente se utiliza el system close, el cual cierra el vinculo al archivo y finalizando la extracción de información. Esto se puede apreciar en la Fig.2.

```
;Abrir config.ini
mov rax, 2
mov rdi, cfg
mov rsi, 0
mov rdx,0
syscall

;Leer config.ini
push rax
mov rdi, rax
mov rax, 0
mov rsi, buf_cfg
mov rdx, 1024
syscall

;cerrar config.ini
mov rax, 3
pop rdi
syscall



;Abrir inventario.txt
mov rax, 2
mov rdi, inv
mov rsi, 0
mov rdx,0
syscall

;Leer inventario.txt
push rax
mov rdi, rax
mov rax, 0
mov rsi, buf_inv
mov rdx, 2048
syscall

;cerrar inventario.txt
mov rax, 3
pop rdi
syscall
```

Figura 2: Extracción de config.ini e inventario.txt

Find_colon y Found:

Estas rutinas presentadas en la Fig.3 se encargan de recorrer el buffer que contiene la información de config.ini y extraer los valores de interés. Para esto se crea un loop que corre las posiciones del buffer hasta encontrar el símbolo ":". Esto se hace porque después de este símbolo se encuentran los valores de interés de configuración de visualización de la terminal. Una vez que se llega a ":" se hace un jump a la rutina found, en donde se copian los valores a las variables predeterminadas. Estas rutinas tienen 3 versiones: 0, 1, 2. Las tres son iguales en términos de lógica. La única diferencia es la cantidad datos que extraen, ya que el símbolo "[redacted]" necesita 3 bytes para ser almacenado. Y los números "92" y "40" en formato ASCII solamente necesitan 2 bytes para ser almacenados.

Una vez que found0 termina, pasa a find_colon1 pasando así la dirección actual en el buffer mediante rsi. Y este proceso se repite hasta llegar a found2, que termina de extraer los tres datos de interés para configuración.

```
        ;Procesamiento de config.ini
        mov rsi, buf_cfg
find_colon0:
        mov al, [rsi]
        cmp al,0
        je not_found
        cmp al, ':'
        je found0
        inc rsi
        jmp find_colon0

found0:
        inc rsi
        mov al, [rsi]
        mov [bar_char], al
        inc rsi
        mov al, [rsi]
        mov [bar_char+1], al
        inc rsi
        mov al, [rsi]
        mov [bar_char+2], al
        inc rsi
        jmp find_colon1
```

Figura 3: find_colon y found

Copy_z:

La rutina copy_z que se presenta en la Fig.4 se utiliza para copiar dos registros, pero en el programa se utiliza con el fin de concatenar variables de strings. Esto se hace mediante dos registros, se toma la primera posición de uno y se escribe una posición del otro registro y esto se repite hasta copiar totalmente un registro en el otro. Ahora si se utiliza un registro que ya tenia datos almacenados y se empieza copiar datos en las posiciones libres se comienza la concatenación.

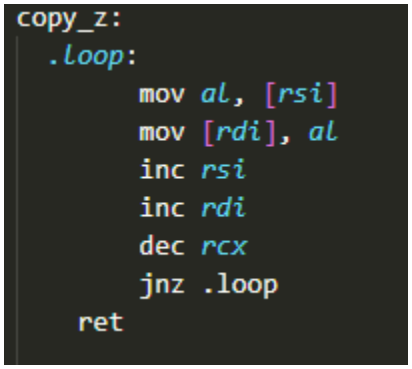


Figura 4: copy_z

Fruta_encontrada:

Esta rutina de la Fig.5 en esencia es muy similar a find_colon y found. Estas dos rutinas trabajan en conjunto para extraer los datos después del “:”. Fruta_encontrada hace lo opuesto, extrae la información que esta antes del “:”. Esto se hace para comenzar con la extracción de información del archivo inventario.txt. Esto con el propósito de extraer el nombre de las frutas, las cuales están siempre antes de un “:”. En esta rutina se hace un loop que compara la posición actual dentro del buffer de inventario y compara a “:”, si no lo es guarda el dato actual en la variable predeterminada y pasa a la siguiente posición en el buffer. Si la posición actual contiene un “:” se pasa a una subrutina llamada .fin la cual introduce un “:0” al final del nombre de la fruta. Esto con el fin de mantener formato y el 0 facilita el manejo sobre esta variable. Una vez arregla el formato se pasa a otra subrutina llamada .cambiar_linea que solamente se encarga de mover la posición en el buffer hasta cambiar de línea. En donde se llama a. salida que incrementa la posición del buffer y se sale del llamado. Esta rutina tiene de fruta_encontrada tiene cuatro versiones: 0, 1, 2, 3. Una para cada fruta: manzanas, peras, naranjas y kiwis.

```
fruta_encontrada0:
 .loop:
        mov al, [rsi]
        cmp al,0
        je not_found
        cmp al, ':'
        je .fin
        mov [manzanas + r12], al
        inc r12
        inc rsi
        jmp .loop

 .fin:
        mov byte[manzanas + r12], ':'
        inc r12
        mov byte[manzanas + r12], 0
        inc rsi
        jmp .cambiar_linea

 .cambiar_linea:
        mov al, [rsi]
        cmp al,0
        je .salir
        cmp al, 10          ; '\n'
        je  .salir
        inc rsi
        jmp .cambiar_linea

 .salir:
        inc rsi
        ret
```

Figura 5: fruta_encontrada

Find_cantidad:

La siguiente rutina aplica la misma lógica que se implemento en find_colon y found. Find_cantidad recorre el buffer de inventario hasta encontrar un “:”. En donde guarda la cantidad de cada fruta. Esto se almacena en una variable predeterminada para cada fruta. Es por esto por lo que hay cuatro versiones de esta rutina: 0, 1, 2 y 3. Que se encargan de continuar en la posición que deja la anterior hasta extraer todos los valores de las cantidades de fruta. Esto se presenta en la Fig.6. Cabe recalcar que lo que se extrae aquí es un valor ASCII que representa al valor numérico. Por lo que este valor no es más que un str y no un valor int.

```
find_cantidad0:
 .loop:
        mov al, [rsi]
        cmp al,0
        je not_found
        cmp al, ':'
        je .found
        inc rsi
        jmp .loop

 .found:
        inc rsi
        mov al, [rsi]
        mov [cantidad_manzanas], al
        inc rsi
        mov al, [rsi]
        mov [cantidad_manzanas+1], al
        inc rsi
        jmp .salir

 .salir:
        ret
```

Figura 6: find_cantidad

Ascii_to_int:

Como se menciono anteriormente las cantidades que se extrajeron anteriormente están en formato ASCII. Por lo que esta rutina se encarga de convertir el valor en ASCII a un valor entero.

```
ascii_to_int:
        xor rax, rax            ; acumulador = 0
.siguiente_digito:
        mov al, [rsi]           ; lee un byte
        cmp al, 0               ; fin de string?
        je  .done
        cmp al, 10              ; '\n'?
        je  .done

        sub al, '0'             ; convierte ASCII a valor
        imul rax, rax, 10       ; acum = acum * 10
        add rax, rdx            ; acum = acum + dígito

        inc rsi
        jmp .siguiente_digito

.done:
        ret
```

Figura 7: ascii_to_int

Ordenar_frutas:

Al tener el nombre de las frutas, estas se deben organizar de forma alfabética. Por lo que se tiene un arreglo de punteros. Donde cada puntero apunta al principio del buffer donde se almacenan los nombres de las frutas. Para el ordenamiento de las palabras se decidió implementar un bubble sort. El cual se encarga de comparar dos cosas y si la primera es mayor que la segunda, se cambian de lugar. Así hasta que todos los ítems estén organizados de menor a mayor. Ahora para implementar esto a esta rutina, se utiliza el valor ASCII de la primera letra de cada palabra. Están se comparan y la mayor es la que va después en el orden alfabético. Esto se puede apreciar dentro de .inner_loop que se toman dos punteros y se extraen las primeras letras de cada uno de los buffers. Estas se comparan y se implementa jbe. Que significa jump if below or equal, lo que determina que si el primer argumento es menor o igual que el segundo haga el jump. Este jump lleva a .no_swap donde no se realiza un cambio. Pero si no se hace el cambio de puntero mediante un registro extra. Lo que se logra que el puntero que iba a un buffer ahora apunta a otro buffer. Esto se repite hasta que no se hace un swap tres veces seguidas. Lo que significa que los punteros están organizados alfabéticamente.

```
ordenar_frutas:
        mov r8d, frutas_count
        dec r8d
 .outer_loop:
        xor edi, edi
 .inner_loop:
        mov rsi, [frutas + rdi*8]
        mov rdx, [frutas + rdi*8 + 8]

        mov al, [rsi]
        mov bl, [rdx]

        cmp al, bl
        jbe .no_swap

        ; swap punteros
        mov rax, rsi
        mov [frutas + rdi*8], rdx
        mov [frutas + rdi*8 + 8], rax

.no_swap:
        inc edi
        cmp edi, r8d
        jb .inner_loop

        dec r8d
        jnz .outer_loop
        ret
```

Figura 8: ordenar_frutas

Strlen:

Esta pequeña rutina se encarga de obtener la longitud de un buffer. Esto lo hace mediante recorrer cada posición del buffer y cada vez que encuentra una posición valida aumenta un contador en rax. Al llegar al final del buffer simplemente se sale y se devuelve. Esto se hace con el propósito de obtener la longitud de un buffer que no se conoce. Esto se va a implementar más adelante para imprimir un buffer que no se esta seguro de su contenido ni su longitud.

```
strlen:
        xor rax, rax
.len_loop:
        cmp byte [rsi+rax], 0
        je  .done
        inc rax
        jmp .len_loop
.done:
        ret
```

Figura 9: strlen

Print_z:

Esta rutina se utiliza para imprimir a una variable que no se conoce. Como no se conoce no se puede imprimir de forma normal. Esta rutina trabaja en conjunto con strlen para poder obtener la longitud del buffer a imprimir.

```
print_z:
        push rsi
        call strlen
        pop rsi
        mov rdx, rax
        mov rax, 1
        mov rdi, 1
        syscall
        ret
```

Figura 10: print_z

Imprimir_frutas:

Este es la primera de dos rutinas que se utilizan para hacer la impresión en la configuración que se solicitó. Primero esta rutina utiliza ebx como un contador de iteraciones, este lo compara con frutas_count que tiene un valor de 4. Ya que hay 4 frutas en la lista. El loop funciona mediante la comparación de ebx y frutas_count, si ebx es menor se continua y si es igual se termino el bucle. Luego de la comparación se obtiene el primero puntero a un nombre de una fruta. Es importante recalcar que en este punto no se conoce a que nombre apunta cada puntero ya que el bubble sort altero estos valores. Si este funciono de forma correcta se tiene una posición esperada, pero esto no se sabe con certeza. Por lo que se llama a la rutina print_z para que imprima el buffer extraído en rsi del primer puntero. Por esta razón se crearon las rutinas print_z y strlen que funcionan para obtener información de un buffer desconocido. Una vez que se print_z imprime el nombre de la fruta en pantalla se llama a imprimir_cantidad. Esta rutina se va a explicar a profundidad en el próximo apartado, pero en resumen imprimir la barra y la cantidad asociada a esa fruta. Luego de que vuelve imprimir_cantidad se imprime un "\n" para hacer un cambio de línea. Se incrementa ebx y se devuelve el loop. Esto se hace 4 veces en total y termina de imprimir en pantalla.

```
imprimir_frutas:
        xor  ebx, ebx
.next:
        cmp  ebx, frutas_count
        jae  .done
        mov  rsi, [frutas + rbx*8]
        call print_z
        call imprimir_cantidad
        mov  eax,1
        mov  edi,1
        lea  rsi, [rel nl]
        mov  edx,1
        syscall
        inc  ebx
        jmp  .next
.done:
        ret
```

Figura 11: imprimir_frutas

Imprimir_cantidad:

Esta es la segunda rutina fundamental para la impresión en la configuración visual solicitada. Esta rutina un buffer almacenado en rsi correspondiente a una de las cuatro frutas. En este momento de la rutina no se sabe cual es la que el buffer contiene. Es por esto por lo que se extrae la primera letra y se compara con alguna de las cuatro posibilidades: m, p, n o k. De esta manera se determina cual fruta contiene el buffer, con esta información se hace un je hacia la subrutina correspondiente .manzanas, .peras, .naranjas, .kiwis. Cada una con la misma lógica solamente ajustadas para cada fruta correspondiente. Una vez se entra a la subrutina correspondiente se cambia el color de la terminal usando la información extraída de config.ini. Después de inicia un loop donde se imprime el carácter de la barra una cantidad de veces igual a la cantidad de esa fruta en el inventario. Tomando el ejemplo1 se imprime el símbolo "▮" cinco veces en el caso del kiwi. Una vez se termina el bucle se pasa a subrutina de salida correspondiente de cada fruta: .donemanzanas, .doneperas, .dondenaranjas y .donekiwis. Todas teniendo la misma lógica, pero adaptadas a la fruta especifica. En todas se restablece el color de la terminal y luego se imprime el valor en ASCII asociado a la cantidad especifica de esa fruta. Después de esto se sale de la rutina y se vuelve al flujo de imprimir frutas hasta acabar con todo el arreglo de punteros.

```
imprimir_cantidad:
        xor r8d, r8d
        mov al, [rsi]
        mov r12, rsi
        cmp al, 'k'
        je .kiwis
        cmp al, 'm'
        je .manzanas
        cmp al, 'n'
        je .naranjas
        cmp al, 'p'
        je .peras

.kiwis:
        ;Cambio de Color
        mov rax,1
        mov rdi,1
        mov rsi,str_color
        mov rdx,8
        syscall

        ;Se hace un bucle de impresion para la cantidad de kiwis
        mov ecx, 5
        cmp r8d, ecx
        je .donekiwis
        ;Impresion de caracter de barra
        mov rax,1
        mov rdi,1
        mov rsi,bar_char
        mov rdx,3
        syscall
        inc r8d
        jmp .kiwis

.donekiwis:
        ; --- Reset the terminal color to default ---
        mov rax, 1
        mov rdi, 1
        mov rsi, reset_color
        mov rdx, reset_color_len
        syscall

        ;Impresion de numero de kiwis
        mov rax,1
        mov rdi,1
        mov rsi, cantidad_kiwis
        mov rdx,1
        syscall
        mov rsi, r12
        ret
```

Figura 12: imprimir_cantidad parte1

```
.manzanas:
        ;Cambio de Color
        mov rax,1
        mov rdi,1
        mov rsi,str_color
        mov rdx,8
        syscall

        ;Se hace un bucle de impresion para la cantidad
        mov ecx, 12
        cmp r8d, ecx
        je .donemanzanas
        ;Impresion de caracter de barra
        mov rax,1
        mov rdi,1
        mov rsi,bar_char
        mov rdx,3
        syscall
        inc r8d
        jmp .manzanas

.donemanzanas:
        ; --- Reset the terminal color to default ---
        mov rax, 1
        mov rdi, 1
        mov rsi, reset_color
        mov rdx, reset_color_len
        syscall

        ;Impresion de numero de manzanas
        mov rax,1
        mov rdi,1
        mov rsi, cantidad_manzanas
        mov rdx,2
        syscall
        mov rsi, r12
        ret

.naranjas:
        ;Cambio de Color
        mov rax,1
        mov rdi,1
        mov rsi,str_color
        mov rdx,8
        syscall

        ;Se hace un bucle de impresion para la cantidad
        mov ecx, 25
        cmp r8d, ecx
        je .donenaranjas
        ;Impresion de caracter de barra
        mov rax,1
        mov rdi,1
        mov rsi,bar_char
        mov rdx,3
        syscall
        inc r8d
        jmp .naranjas

.donenaranjas:
        ; --- Reset the terminal color to default ---
        mov rax, 1
        mov rdi, 1
        mov rsi, reset_color
        mov rdx, reset_color_len
        syscall

        ;Impresion de numero de naranjas
        mov rax,1
        mov rdi,1
        mov rsi, cantidad_naranjas
        mov rdx,2
        syscall
        mov rsi, r12
        ret
```

Figura 13: imprimir_cantidad parte2

```
.peras:
        ;Cambio de Color
        mov rax,1
        mov rdi,1
        mov rsi,str_color
        mov rdx,8
        syscall

        ;Se hace un bucle de impresion para la cantidad
        mov ecx, 8
        cmp r8d, ecx
        je .doneperas
        ;Impresion de caracter de barra
        mov rax,1
        mov rdi,1
        mov rsi,bar_char
        mov rdx,3
        syscall
        inc r8d
        jmp .peras

.doneperas:
        ; --- Reset the terminal color to default ---
        mov rax, 1
        mov rdi, 1
        mov rsi, reset_color
        mov rdx, reset_color_len
        syscall

        ;Impresion de numero de peras
        mov rax,1
        mov rdi,1
        mov rsi, cantidad_peras
        mov rdx,1
        syscall
        mov rsi, r12
        ret
```

Figura 14: imprimir_cantidad parte3

Comprobación de funcionamiento:

A continuación, se presentan las comprobaciones de funcionamiento para los ejemplos 1 y 2. La diferencia entre ambos siendo el carácter utilizado para la barra y el color a modificar de la terminal. Esto demuestra la flexibilidad y modularidad del programa desarrollado.

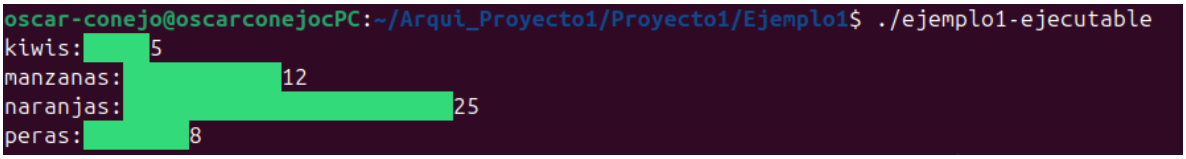


Figura 15: Comprobación Ejemplo 1

Figura 16: Comprobación Ejemplo 2