Trabajo Práctico Wordle

XX.11.2024

—

Hitta Gonzalo Francisco

Lopez Ciro Martin

Martin Godoy

Grupo 10

UNGS

Organización del computador

Comisión 04

Sumario

En el desarrollo de este trabajo práctico haremos uso de los conocimientos adquiridos durante la cursada de la materia Organización del Computador, enfocándonos principalmente lo que se ve durante la segunda parte de la materia, que es el lenguaje de código ensamblador tipo ARMv6 de 32 bits para Raspberry pi v1.2.

Lo que se realizará será un juego de Wordle de consola en el que se busca descubrir una palabra oculta en un número limitado de intentos.

Metas

1. Realizar el programa en el lenguaje Assembly, cumpliendo con las especificaciones de características solicitadas. El juego debe correr en la plataforma.
2. Realizar un informe donde se describa el pseudocódigo del programa y la experiencia/dificultades al realizar dicho programa. (entregarse en formato PDF)

Especificaciones

- El juego debe tomar palabras desde un archivo de texto.
- El juego debe tener un límite de intentos
- El usuario debe poder ingresar una palabra cada intento.
- El juego debe comparar las palabras e informar de las coincidencias por medio de un codigo de color (verde , amarillo, rojo)
- Se debe calcular un resultado en puntuación y ofrecer al jugador de guardarlo con su “Nick name”
- El juego debe ofrecer la posibilidad de volver a jugar o cerrar el programa.

PseudoCodigo

I. .Data

En esta área se albergan los datos, constantes, espacios reservados, punteros y demás utilizados en el programa.
Para este programa se han utilizado las siguientes etiquetas

- **Archivo, ListaPalabras,palabraEnJuego,numeroAleatorio**:Estas etiquetas se encargan de guardar todo lo necesario para seleccionar una palabra aleatoria para poder jugar.
- **cantLetrasPalabra,palabraUsuario,puntuacion:** Estas etiquetas se encargan de guardar la participación del usuario, su palabra intento, su puntuacion y la cantidad de letras de la palabra a adivinar.
- **Volver_a_jugar:** es una bandera para dar la opción al jugador de volver a jugar o salir del programa
- **Ranking_file:** ruta del archivo de texto donde se guardan las puntuaciones.
- **color_rojo, color_verde,color_amarillo,color_reset:** guardan los codigo de color para configurar la consola e imprimier letras con differente color.
- **Mensaje_puntuacion,mensaje_ingreso,mensaje_ingreso2, Mensaje_perdiste,mensaje_volver_a_jugar,mensaje_ganaste, mensaje_NICK:** Almacenan los mensajes para informar el estado del juego al usuario.
- **buffer,bufferNombre,bufferRanking:** se encargan de guardar temporalmente datos a ser utilizados por la lógica del juego antes de ser guardados o descartados.
- **Intentos:** Es una constante que se encargan de limitar el juego ,para poder decidir si el jugador pierde, si no también se usa para calcular puntuación.
- **saltoLinia:** guarda el código de salto de línea /n , por cuestión de alineación de texto por pantalla.

II. .text

En esta sección se escribieron varias funciones , junto a sus subrutinas necesarias para conformar el programa.

- **Leer_palabras:** Esta subrutina tiene como función principal carga una lista de palabras desde un archivo de texto y almacenarlas en memoria
- **Leer_palabra:** Lee una palabra ingresada por el usuario y la almacena de un buffer a memoria con un carácter nulo al final.
- **Imprimir_mensaje**: Imprime un mensaje en la pantalla utilizando la syscall de salida.
- **imprimir _mensaje2:** Imprime un mensaje en la pantalla utilizando la syscall de salida.
- **Sortear_palabra:** Genera un número aleatorio en el rango de 0 a 9 utilizando el tiempo del sistema como semilla. Este número se almacena en una variable en memoria
- **almacenarPalabra:** almacena la palabra sorteada en memoria
- **Almacenar_loop:** recorre byte por byte las palabras almacenadas en un buffer hasta llegar a su fin y almacenarlas.
- **incrementarLinea:** Esta rutina se encarga de contar la cantidad de ENTER que nos desplazamos en la lista de palabras para que coincida con nuestro número aleatorio.
- **almacenarDireccion:** Guarda el puntero a donde empieza la palabra a jugar , trae el puntero a donde se va a guardar y guarda 1 byte
- **truncarLoop:** Se encarga de buscar el final de la palabra
- **Truncar:** Se encarga de poner un 0 al final de nuestra palabra elegida
- **Fin_almacenar:** Sale de la función almacenar
- **Verificar_letras:** Recorrer palabraUsuario y palabraEnJuego. Si los caracteres coinciden estando en la misma posición, marcar como verde. Si no coinciden, buscar el carácter en otra posición, si encuentro el caracter en otra posicion lo marco como amarillo. Si no encuentra el carácter en ninguna de las anteriores verificaciones, marcar con rojo. En esta subrutina lo que hacemos es asignar a los registros los valores correspondientes que están guardados en las etiquetas o en otros registros, e inicializamos el índice de palabraUsuario.
- **Verificar_loop:** Loop para verificar si coinciden las palabra en juego con la ingresada por el usuario

- **Buscar_caracter:** hace las comparaciones para llamar acorde a lo necesario según el carácter que se está revisando en base a la cadena a adivinar.
- **Siguiente_busqueda:** Una vez que se hicieron las comparaciones de un carácter se llama a siguiente para evaluar el siguiente.
- **Es_verde:** Es verde se encarga de pintar los caracteres que están en la posición correcta si es adivinada por el usuario, debe mostrar el carácter en verde, luego resetea el color para no pintar la cadena.
- **Es_amarillo:** Se encarga de que si el carácter esta en la cadena pero en una posición incorrecta, lo pinta de amarillo, luego resetea el color para no pintar la cadena entera.
- **Verificar_no_encontrado:** es llamada cuando no se encuentran los caracteres ingresados por el usuario en la cadena, se pintan en rojo y luego se resetea el color para no pintar la cadena entera.
- **Siguiente_caracter:** Avanza al siguiente carácter de palabraUsuario.
- **Fin_verificar:** Vuelve al main.
- **Imprimir_color:** Esta función se encarga de setear el color en la consola
- **Imprimir_caracter:** Esta función se encarga de imprimir el carácter del color indicado en el paso anterior (1 solo carácter a la vez)
- **saltoDeLinea:** mostrar un salto de línea en pantalla
- **imprimir_Perdiste:** Esta función se encarga de anunciar al jugador que perdio al acabarse sus intentos.
- **Calcular_puntos:** Esta función se encarga de calcular los puntos de la partida (intentos * largoPalabra)
- **Dos_digitos:** Si tiene dos dígitos nuestro puntaje hay que separarlos antes de transformarlos en ASCII se divide , multiplica y resta para lograrlo.
- **Div_loop:** Debido a limitaciones con la arquitectura se tuvo que hacer un loop para conseguir el %10 del número.
- **Division_done:** Al finalizar se puede convertir cada carácter en ASCII
- **Un_digito:** Si tiene un solo dígito solo es necesario sumar 48 para pasar a ASCII
- **Imprimir:** Se encarga de imprimir el mensaje junto al resultado.

- **Fin_del_juego:** Se encarga de linkear diferentes subrutinas para cuando el jugador pierde.
- **Opcion_volver_a_jugar:** Se encarga de mostrar por pantalla el mensaje para volver a jugar y tomar la repuesta por teclado. S jugar N cerrar programa.
- **Preguntar:** Mensaje para volver a jugar, si el usuario ingresa ‘n’, se finaliza el juego, si ingresa ‘s’ se resetean las variables necesarias para jugar y se llama a jugar.
- **Ganaste:** Si el jugador ganó se muestra un mensaje por pantalla.
- **cerrarArchivo:** Cierra el archivo de las palabras después de utilizarlo al principio del juego.
- **Error:** Muestra un mensaje de error se no se pudo cerrar adecuadamente.
- **Resetear_color:** Esta función se encarga de resetear la configuracion de consola al color base
- **Pedir_nombre:** Hace la syscall necesaria para mostrar un mensaje en pantalla pidiéndole al jugador que ingrese su nombre y luego hace la syscall para leer, permitiendo ingresarlo, almacena el nombre en el buffer.
- **Grabar_ranking:** abre un archivo donde escribe el nombre ingresado y se guarda en este con su puntuación.
- **Mostrar_ranking:** Abre el archivo y lo muestra por pantalla.
- **Main:** Con subrutinas lee las palabras del archivo, las sortea y almacena el resultado, también cierra el archivo e inicializa los intentos.
- **Jugar:** Punto de inicio de la parte jugable del juego, utilizado principalmente para volver a jugar
- **Main_loop:** loop utilizado para administrar los intentos del jugador y que pueda ingresar multiples palabras intento.
- **Ganador_post_game:** Secion del main para llamar las rutinas que competen solo a los que ganaron, sea guardar ranking ver puntos,etc
- **Finalizar:** Esta funcion es lo mismo que el usual fin, se encarga de llamar al sistema para cerrar el programa.

Experiencia (dificultades, cosas a destacar)

- Siendo el primer proyecto grande de assembly, fue laborioso al principio, costó organizarse y entender el flujo/control requerido para afrontar dicho proyecto.
- Debido a limitaciones de tiempo, no se le ha podido dedicar toda la atención a errores y detalles como nos hubiera gustado.
- Una de las mayores dificultades encontradas fue aprender el funcionamiento de las distintas instrucciones específicas del sistema como el file descriptor , syscalls, y otras instrucciones las cuales solo con la documentación oficial no alcanzó para comprender completamente su funcionamiento. Todo lo relacionado con manejo de archivos externos fue complicado.
- Fue difícil comprender el funcionamiento de las cadenas de texto cuando estas requieren modificaciones y formateo dependiendo los sistemas dentro del juego.
- El elegir el tipo correcto de datos y su tamaño reservado fue algo a tener en cuenta a lo largo de todo el proyecto.
- Se intentó hacer el random utilizando una syscall para obtener los valores del reloj, pero no logramos hacerlo andar por lo que optamos por unas semillas y una fórmula matemática.
- Al guardar los valores del ranking del archivo se sobrescribián los anteriores valores.
-

- Este grupo se formó al dejar la materia dos integrantes del grupo 14 y uno del 10, por lo que cada grupo tenía códigos distintos bastante progresados. Pudimos reutilizar bastante código que ya teníamos hecho de antemano y así avanzar más rápido.

Herramientas usadas, Bibliografía.

- Se utilizó tanto **Putty** como **winscp** para conectarse al servidor y manejar archivos. También control de versión sincronizando con una carpeta local.
- Se utilizó como editores: principalmente **Vscode** , pero también se usó **notepad++ , nano, notepad**.
- Se utilizó **telegram y Discord** para la comunicación con profesores y compañeros.
- Se utilizó la documentación: Específica de la materia Organización del computador. (wiki.educabit.ar) ,https://developer.arm.com/, foros de https://stackoverflow.com/, **chatGPT** fue usado para contestar dudas surgidas al leer la documentación. Por ejemplo para tener una explicación más específica o al contrario más simplificada de algunas instrucciones. Por ejemplo “file descriptor”. **NO SE UTILIZO** para copiar y pegar código. Todo código ajeno fue usado como referencia, y o tenido en cuenta para crear una versión propia adaptada al proyecto. Como por ejemplo el código aleatorio proporcionado por el profesor Mariano Vargas. También se puede agregar el hecho de que algunos comentarios fueron creados usando autocompletar o generado. Lo que creo más molestia que beneficio en algunos casos.
- Se han mirado los videos del campus virtual como otros videos en youtube para comprender algunos conceptos.
- Cabe destacar que algunas partes de código fueron escritas primero en **java** para facilitar el entendimiento a la hora de resolver la implementación de algunas funciones.