Sistemas Operativos

Práctica 3. Introducción a los sistemas operativos Linux y Windows: Interfaz de interrupciones

Prof. Jorge Cortés Galicia

Competencia

El alumno aprende a programar aplicaciones sencillas a nivel ensamblador bajo los sistemas operativos Linux y Windows utilizando la interfaz de interrupciones respectiva de cada sistema, mediante la comprensión de la estructura general e instrucciones para el lenguaje ensamblador del procesador Intel de 32 bits.

Desarrollo

Sección Linux:

1. Inicie sesión en Linux.
2. Cree una carpeta llamada Borrar para que allí mantenga sus programas. Al finalizar su sesión de trabajo respalde sus programas en una memoria usb y elimine esta carpeta con su contenido.
3. Capture el siguiente código:

```
segment  .data                                          ;Segmento de datos
cadena   db  'Programando en ensamblador para Linux',0xA        ;Cadena a imprimir

segment  .text                                          ;Segmento de código
global   _start                    ;Punto de entrada al programa (usado en el enlazador ld)
_start:                            ;Inicio del programa
         mov     edx,38d           ;Longitud de cadena
         mov     ecx,cadena        ;Cadena a escribir
         mov     ebx,1             ;Salida estandar
         mov     eax,4             ;Numero de llamada al sistema "sys_write"
         int     0x80              ;Interrupción de llamadas al sistema del kernel de Linux
         mov     eax,1             ;Número de llamada al sistema "sys_exit"
         int     0x80              ;Interrupción de llamadas al sistema del kernel de Linux
```

4. Guarde su archivo con extensión **.asm** dentro de Borrar y ensamble desde una consola con el siguiente comando:

   nasm –f elf –o nombre_archivo.o nombre_archivo.asm

5. Enlace el código objeto intermedio generado en el paso anterior con el siguiente comando:

   ld -m elf_i386 -o nombre_ejecutable nombre_archivo.o

6. Al final de estos dos pasos obtendrá un programa ejecutable, pruebe el funcionamiento de su aplicación. Dé sus observaciones del código capturado.

7. Capture el siguiente código:

```
segment    .bss                ;Segmento de datos
cadena     resb   50           ;Espacio en memoria para la cadena a almacenar

segment    .text               ;Segmento de código
global     _start              ;punto de entrada al programa (usado con el enlazador ld)
_start:                        ;Inicio del programa
           mov    edx,50d      ;Longitud del bufer
           mov    ecx,cadena   ;Cadena a leer
           mov    ebx,0        ;Entrada estandar
           mov    eax,3        ;Numero de llamada al sistema "sys_read"
           int    0x80         ;Interrupción de llamadas al sistema del kernel de Linux
           mov    edx,50d      ;Longitud de cadena
           mov    ecx,cadena   ;Cadena a escribir
           mov    ebx,1        ;Salida estandar
           mov    eax,4        ;Numero de llamada al sistema "sys_write"
           int    0x80         ;Interrupción de llamadas al sistema del kernel de Linux
           mov    eax,1        ;Número de llamada al sistema "sys_exit"
           int    0x80         ;Interrupción de llamadas al sistema del kernel de Linux
```

8. Ensamble, enlace y ejecute esta aplicación. Dé sus observaciones del código capturado.
9. Programe una aplicación en ensamblador que genere un contador de 0 a 9, mostrando en pantalla el conteo generado. Consejo: revise las instrucciones de ensamblador CMP, JMP, JE, JNE, e INC.
10. Programe una aplicación en ensamblador que copie 5 cadenas dadas (cadena1 a cadena5) a una nueva cadena (cadena6). La copia de las cadenas deberá ser intercalando los caracteres de cada cadena. Las cadenas dadas deben de ser ingresadas por teclado. Muestre en pantalla el contenido de cadena6. Consejo: revise el uso de los registros índice ESI y EDI.
11. Programe una aplicación en ensamblador que muestre en pantalla la longitud de una cadena que haya sido ingresada por teclado. Considere el caso de que la cadena tenga 10 caracteres o más.
12. Programe una aplicación en ensamblador que concatene 5 cadenas (cadena1 a cadena5) ingresadas por teclado, mostrando en pantalla lo siguiente: el contenido de la cadena concatenada, la cadena concatenada en sentido inverso y el número total de vocales que tiene la cadena concatenada.
13. Programe una aplicación en ensamblador que implemente una calculadora con las cuatro operaciones básicas. A través de un menú, dé la posibilidad de seleccionar la operación a realizar. Maneje únicamente dígitos enteros positivos en el intervalo [0, 255]. Consejo: revise las instrucciones de ensamblador ADD, SUB, MUL, y DIV.

Sección Windows:

1. Inicie sesión en Windows.
2. Cree una carpeta llamada Borrar en disco duro para que allí mantenga sus programas. Al finalizar su sesión de trabajo elimine esta carpeta con su contenido.
3. Capture el siguiente código:

```
segment    .data
      cadImprimir             db   'Ensamblando en Windows',0xA        ;Segundo argumento para la llamada al sistema _WriteConsoleA()

segment    .bss
      handleConsola           resd   1                                 ;Primer argumento para la llamada al sistema _WriteConsoleA()
      longitudCadena          resd   1                                 ;Tercer argumento para la llamada al sistema _WriteConsoleA()
      caractEscritos          resd   1                                 ;Cuarto argumento para la llamada al sistema _WriteConsoleA()
      ultimoArgumento         resd   1                                 ;Quinto argumento para la llamada al sistema _WriteConsoleA()

segment    .text
global    _main
extern    _GetStdHandle@4                                             ;Acceso externo a la llamada al sistema _GetStdHandle()
extern    _WriteConsoleA@20                                           ;Acceso externo a la llamada al sistema _WriteConsoleA()
extern    _ExitProcess@4                                              ;Acceso externo a la llamada al sistema _ExitProcess()

_main:    push    dword -11                                           ;Argumento pasado por la pila y usado en _GetStdHandle() para la salida estándar
          call     _GetStdHandle@4                                    ;Invocacion de _GetStdHandle()
          mov     [handleConsola],eax                                 ;Devolucion del manejador de consola para escritura en el registro eax

          xor      eax,eax                                            ;Limpieza del registro eax (eax=0)
          mov      eax,23d                                            ;eax=23 caracteres de longitud de la cadena a imprimir
          mov     [longitudCadena],eax                                ;Se guarda la longitud en memoria
          xor      eax,eax                                            ;Limpieza del registro eax (eax=0)
          mov      eax,0d                                             ;eax=0 valor del ultimo argumento de _WriteConsoleA()
          mov     [ultimoArgumento],eax                               ;Se guarda el valor del ultimo argumento en memoria

          push    dword [ultimoArgumento]                             ;Quinto argumento de _WriteConsoleA() pasado por la pila
          push    dword caractEscritos                                ;Cuarto argumento de _WriteConsoleA() pasado por la pila
          push    dword [longitudCadena]                              ;Tercer argumento de _WriteConsoleA() pasado por la pila
          push    dword cadImprimir                                   ;Segundo argumento de _WriteConsoleA() pasado por la pila
          push    dword [handleConsola]                               ;Primer argumento de _WriteConsoleA() pasado por la pila
          call     _WriteConsoleA@20                                  ;Invocacion de _WriteConsoleA()

          xor      eax,eax                                            ;Limpieza del registro eax (eax=0)
          mov      eax,0d                                             ;eax=0 valor del argumento de _ExitProcess()
          mov     [ultimoArgumento],eax                               ;Se guarda el valor del argumento en memoria
          push    dword [ultimoArgumento]                             ;Argumento de _ExitProcess() pasado por la pila
          call     _ExitProcess@4                                     ;Invocacion de _ ExitProcess()
```

4. Guarde su archivo con extensión **.asm** dentro de Borrar y ensamble desde una consola con el siguiente comando:

   nasm –f win32 –o nombre_archivo.obj nombre_archivo.asm

5. Enlace el código objeto intermedio generado en el paso anterior con el siguiente comando:

   ld nombre_archivo.obj -m i386pe -e _main -L “ruta_al_archvivo_kernel32” -l kernel32 -o nombre_ejecutable.exe

6. Al final de estos dos pasos obtendrá un programa ejecutable, pruebe el funcionamiento de su aplicación. Dé sus observaciones del código capturado.
7. Capture el siguiente código:

```
segment    .bss
      handleConsola          resd    1              ;Primer argumento para la llamada al sistema _ReadConsoleA()
      cadLeer                resb    30             ;Segundo argumento para la llamada al sistema _ReadConsoleA()
      longitudCadena         resd    1              ;Tercer argumento para la llamada al sistema _ReadConsoleA()
      caractLeidos           resd    1              ;Cuarto argumento para la llamada al sistema _ReadConsoleA()
      ultimoArgumento        resd    1              ;Quinto argumento para la llamada al sistema _ReadConsoleA()

segment    .text
global    _main
extern    _GetStdHandle@4                           ;Acceso externo a la llamada al sistema _GetStdHandle()
extern    _WriteConsoleA@20                         ;Acceso externo a la llamada al sistema _WriteConsoleA()
extern    _ReadConsoleA@20                          ;Acceso externo a la llamada al sistema _ReadConsoleA()
extern    _ExitProcess@4                            ;Acceso externo a la llamada al sistema _ExitProcess()

_main:    push      dword -10                 ;Argumento pasado por la pila y usado en _GetStdHandle() para la entrada estandar
          call       _GetStdHandle@4          ;Invocacion de _GetStdHandle()
          mov       [handleConsola],eax       ;Devolucion del manejador de consola para lectura en el registro eax

          xor       eax,eax                   ;Limpieza del registro eax (eax=0)
          mov       eax,30d                    ;eax=30 caracteres de longitud de la cadena a leer
          mov       [longitudCadena],eax      ;Se guarda la longitud en memoria
          xor       eax,eax                   ;Limpieza del registro eax (eax=0)
          mov       eax,0d                    ;eax=0 valor del ultimo argumento de _ReadConsoleA()
          mov       [ultimoArgumento],eax     ;Se guarda el valor del ultimo argumento en memoria

          push      dword [ultimoArgumento]   ;Quinto argumento de _ReadConsoleA() pasado por la pila
          push      dword caractLeidos        ;Cuarto argumento de _ReadConsoleA() pasado por la pila
          push      dword [longitudCadena]    ;Tercer argumento de _ReadConsoleA() pasado por la pila
          push      dword cadLeer             ;Segundo argumento de _ReadConsoleA() pasado por la pila
          push      dword [handleConsola]     ;Primer argumento de _ReadConsoleA() pasado por la pila
          call      _ReadConsoleA@20          ;Invocacion de _ReadConsoleA()

          xor       eax,eax
          push      dword -11                 ;Argumento pasado por la pila y usado en _GetStdHandle() para la salida estandar
          call      _GetStdHandle@4           ;Invocacion de _GetStdHandle()
          mov       [handleConsola],eax       ;Devolucion del manejador de consola para escritura en el registro eax

          xor       eax,eax                   ;Limpieza del registro eax (eax=0)
          mov       eax,30d                   ;eax=30 caracteres de longitud de la cadena a imprimir
          mov       [longitudCadena],eax      ;Se guarda la longitud en memoria
          xor       eax,eax                   ;Limpieza del registro eax (eax=0)
          mov       eax,0d                    ;eax=0 valor del ultimo argumento de _WriteConsoleA()
          mov       [ultimoArgumento],eax     ;Se guarda el valor del ultimo argumento en memoria

          push      dword [ultimoArgumento]   ;Quinto argumento de _WriteConsoleA() pasado por la pila
          push      dword caractLeidos        ;Cuarto argumento de _WriteConsoleA() pasado por la pila
          push      dword [longitudCadena]    ;Tercer argumento de _WriteConsoleA() pasado por la pila
          push      dword cadLeer             ;Segundo argumento de _WriteConsoleA() pasado por la pila
          push      dword [handleConsola]     ;Primer argumento de _WriteConsoleA() pasado por la pila
          call      _WriteConsoleA@20         ;Invocacion de _WriteConsoleA()

          xor     eax,eax                     ;Limpieza del registro eax (eax=0)
          mov     eax,0d                      ;eax=0 valor del argumento de _ExitProcess()
          mov     [ultimoArgumento],eax       ;Se guarda el valor del argumento en memoria
          push    dword [ultimoArgumento]     ;Argumento de _ExitProcess() pasado por la pila
          call     _ExitProcess@4             ;Invocacion de _ ExitProcess()
```

8. Ensamble (ignore la advertencia generada en caso de que se muestre), enlace y ejecute. Dé sus observaciones acerca del funcionamiento del programa.
9. Programe las aplicaciones de los puntos 9 al 13 de la sección de Linux usando el ensamblador para Windows.