THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 11

C- Stream I/O

Agenda

- Build a Program Stack
- Use external C-functions in assembly programs
  - fprintf and fscanf
  - Opening a file (fopen function)
  - fprintf
  - Closing a file (fclose function)

- **Example: Adding two Integers using printf**
- **Example: Adding two Integers using local variables**
- **Example: Adding two Integers using scanf and printf**

Build a Program's Stack

- In this lecture, we would like to use "gcc" compiler to compile the assembly language program
- "gcc" provide more compiling and linking options
- "gcc" needs "main" function as a starting point within assembly program
- In that case, we are writing a program similar to C and C++ programs
- We will use "global main" instead "global start"

NASM Program Skelton

```
section .data   (segment .data)
        ; Define variables with initialized values in the data section
        var_name  dx   value     (d for define, x for data type)

section .bss   (segment .bss)
    ; Define variables with uninitialized values in the data section

; Code goes in the text section
section .text       (segment .text)
        global _start
_start:
        enter 0,0
        pusha

        ; Code goes in the text section

        popa
        mov eax, 0
        leave
        ret
```

```
enter   0,0          ——
                        ; setup the program
pusha                ——
;
; Your program here
;
popa                 ——
mov     eax, 0
                        ; cleanup the program
leave
ret                  ——
```

GCC Program Skelton

A general assembly program could have the following format

```
section .data   (segment .data)
        ; Define variables with initialized values in the data section
section .bss     (segment .bss)
     ; Define variables with uninitialized values in the data section
section .text   (segment .text)
        global main          ;  other programs can use this program since main is global
 main:
        push ebp          ; save ebp for whoever called main function (OS or other program)
        mov ebp, esp    ; create our new stack frame

        ; Code goes in the text section


        ; destroy the stack after we done
        mov esp, ebp
        pop epb                ;  restore the old ebp value
        ret
```

```
void main ( )
 {
  codes
 }
```

Use “printf” within assembly program

- For now, we will use the C-program function “printf”
  - Print output from the program
  - Print out all register values
- Include “extern printf” before the main function (since it is external)
- Example:

Assume you would like to print a message,

1- push the message into the new stack that we have created

2- call “printf” as follow:

```
call    printf
```

- Later, we will build our libraries to achieve the previous steps.

Example: Hello World Program

```
section .data   (segment .data)
      msg:     db ‘Display Hello world with printf !’,10, 0
                                ; 10= new line, 0 = end of the message
section .text
     extern printf

 global main
 main:
        push ebp
        mov ebp, esp

       ; Code goes in the text section
        push msg       ; push the memory address of the message to the stack
        call printf      ; printf will display the contents of that memory address

         mov esp, ebp
         pop epb
         ret
```

Compiling and running the HelloP Program

To compile the program:

1- Move to the directory that contains the program

cd /Desktop/AssemblyPrograms

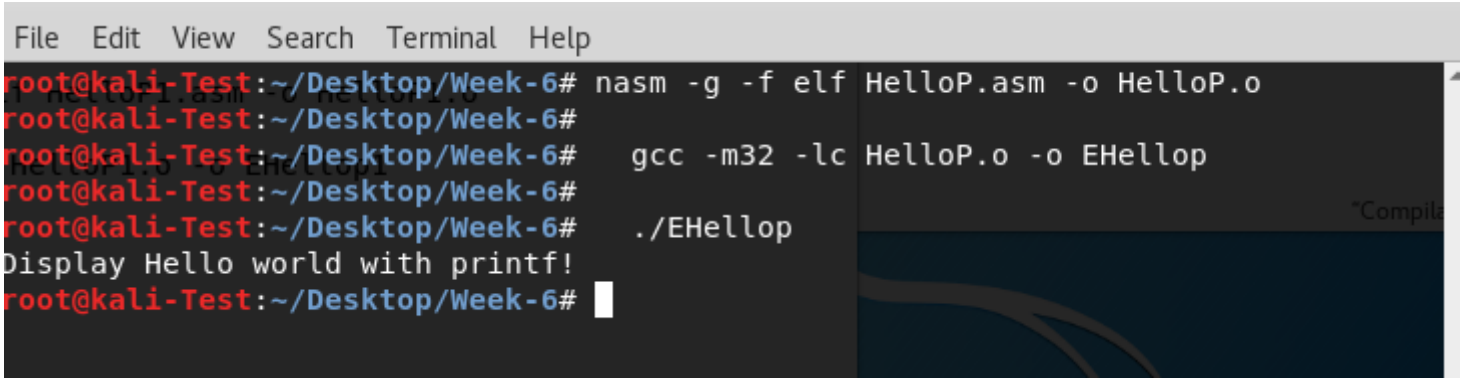
2- Execute the following two steps

nasm -f elf HelloP.asm -o HelloP.o // Create object file

gcc –m32 -lc HelloP.o -o EHelloP // create executable program

To run the program, type

./EHelloP (enter)



Understand of C-printf

- Prints its first argument (format string) to stdout with all formatting characters replaced by the ASCII representation of the corresponding data argument

```c
#include <stdio.h>

int main()
{
    int a = 100;
    int b = 65;
    char ch = 'z';
    char banner[10] = "Hello";
    double pi = 3.14159;
    printf("The variable 'a' in decimal: %d\n", a);
    printf("The variable 'a' in hex: %x\n", a);
    printf("The variable 'a' in binary: %b\n", a);
    printf("'a' plus 'b' as decimal: %d\n", a+b);
    printf("'a' plus 'b' as character: %c\n", a+b);
    printf("A char %c.\t The string %s\nA float value = %f\n", ch, banner, pi);

    return 0;
}
```

```
The variable 'a' in decimal: 100
The variable 'a' in hex: 64
The variable 'a' in binary: %b
'a' plus 'b' as decimal: 165
'a' plus 'b' as character: 
A char z.        The string Hello
 A float value = 3.141590
```

Understand of C-printf

- Example-2

```c
#include <stdio.h>

 main()
  {
   char   char1='a';          /* sample character */
   char   str1[]="Hello World!";   /* sample string */
   int    int1=1234567;       /* sample integer */
   int    hex1=0x6789ABCD;    /* sample hexadecimal */
   float  flt1=5.327e-30;     /* sample float */
   double flt2=-123.4e300;    /* sample double */

   printf("Display All Values: %c %d %X %e %E \n", /* format string for printf */
   char1, str1, int1, hex1, flt1, flt2);

   printf("Display All Values:\n %c\n %d\n %X\n %e\n %E\n \n", /* format string for printf */
   char1, str1, int1, hex1, flt1, flt2);

  }
```

```
Display All Values: a 1334668640 12D687 5.327000e-30 -1.234000E+302
Display All Values:
 a
 1334668640
 12D687
 5.327000e-30
 -1.234000E+302
```

Standard C Library

- I/O commands are not included as part of the C language.
- Instead, they are part of the Standard C Library.
  - A collection of functions and macros
  - Automatically linked with every executable.

- Since they are not part of the language, compiler must be told about function interfaces.
- Standard header files (e.g. < stdio.h) are provided, which contain declarations of functions, variables, etc.

Basic I/O Functions: <stdio.h>

The standard I/O functions are declared in the <stdio.h> header file.

Function	*Description*
putchar	Displays an ASCII character to the screen.
getchar	Reads an ASCII character from the keyboard.
printf	Displays a formatted string,
scanf	Reads a formatted string.
fopen	Open/create a file for I/O.
fread	Read unformatted number of characters from a file and stores in a buffer
fwrite	Write unformatted number of characters from buffer to a file
fprintf	Writes a formatted string to a file.
fscanf	Reads a formatted string from a file.

Formatted I/O

- Printf and scanf allow conversion between ASCII representations and internal data types (e.g. integers).

- Format string contains text to be read/written, and formatting characters that describe how data is to be read/written.

- %d signed decimal integer
  %f signed decimal floating-point number
  %x unsigned hexadecimal number
  %b unsigned binary number
  %c ASCII character
  %s ASCII string

Special Character Literals

- Certain characters cannot be easily represented by a single keystroke, because they
  - correspond to whitespace (newline, tab, backspace, ...)
  - are used as delimiters for other literals (quote, double quote, ...)
- These are represented by the following sequences:

\n	newline
\t	tab
\b	backspace
\\	backslash
\'	single quote
\"	double quote

Display Several Values With “Printf” in Assembly

- The general cyntex to use “printf”

  printf (format, values)

- The general steps to use “printf”
  - Define the print format in the data section
  - Define the variables’ values in the data and/or bss sections
  - Push values to be displayed from write to left
  - Push print format
  - Call printf

```
push value for variable n
push value for variable n-1
-------
push value for variable 1
push print_format
 call printf
```

Display Several Values With Printf

- Example: Write assembly code to display a message and its length, an integer, and its memory location using printf

```
SECTION .data                ; Data section
  msg:     db "Display Hello world with printf !",10, 0
  msglen:  equ $-msg
  format1: db "Message Length = %d", 10, 0
  i:       dd  120
  format2: db "Value of the integer is %d", 10, 0
  format3: db "Memory location of my integer is %d", 10, 0
  format4:    db "Display All Values:    Memory Location    = %d, Integer value = %d      Message Length    = %d", 10, 0

 section .text

 extern printf
 global main
 main:
push ebp
mov ebp, esp

push msg
call    printf


; printf (format1, msglen)
push DWORD msglen
push    format1
call    printf
```

```
; printf (format2, value)
push DWORD [i]
push   format2
call    printf

; printf (format3, value)
push DWORD i
push   format3
call    printf
```

; printf(format4, memory location, intger, msglen)

```
push DWORD msglen
push DWORD [i]
push DWORD i
push   format4
call    printf

mov esp, ebp
pop ebp
ret
```

```
File  Edit  View  Search  Terminal  Help
root@kali-Test:~/Desktop/Week-10#  nasm -g -f elf print2.asm -o print2.o
root@kali-Test:~/Desktop/Week-10# gcc -m32 -lc print2.o -o print2
root@kali-Test:~/Desktop/Week-10# ./print2
Display Hello world with printf !
Message Length = 35
Value of the integer is 120
Memory location of my integer is 4231252
Display All Values: Memory Location    = 4231252, Integer value = 120      Message Length    = 35
root@kali-Test:~/Desktop/Week-10#
```

Example: Add Two Integers

- Example: Write assembly code to display the following:

Add two number program

Value of the first Integer Number is 25

Value of the second Integer Number is 13

25 + 13 is 38

Example: Add Two Integers

```
; Add two Integers

SECTION .data                   ; Data section
  msg:     db "Add two numbers program ",10, 0
  N1:      dd  25
  format1: db "Value of the first Integer Number is %d", 10, 0
  N2:      dd  13
  format2: db "Value of the second Integer Number is %d", 10, 0
  ;N3:     dd 0
  format3: db "%d + %d is %d", 10, 0

;section .bss
  N3:   resdd 1

section .text

  extern printf
  global main
  main:
        push ebp
        mov ebp, esp

       push msg
       call    printf

       ; printf (format1, value)
       push DWORD [N1]
       push   format1
       call    printf

       ; printf (format2, value)
       push DWORD [N2]
       push   format2
       call    printf

       mov eax, [N1]
       add eax, [N2]
       mov DWORD [N3], eax
       ; printf (format3, value1, value2, value3)
       push DWORD [N3]
       push DWORD [N2]
       push DWORD [N1]
       push format3
       call    printf

       mov esp, ebp
       pop ebp
       ret
```

```
root@kali-Test:~/Desktop/Week-10#   gcc -m32 -lc Add2N1.o -o Add2N1
gcc: error: Add2N1.o: No such file or directory
root@kali-Test:~/Desktop/Week-10#
root@kali-Test:~/Desktop/Week-10#  ./Add2N1
Add two numbers program
Value of the first Integer Number is 25
Value of the second Integer Number is 13
25 + 13 is 38
```

Global vs Local variables

- Global variables
  - Defined in .data and/or .bss sections
  - They are available for all functions within the program
- Local variables
  - Defined inside a function
  - They are available that function only
  - Can be referenced by ebp register

Global vs Local variables

```
#  Global variables
     int x = 10;              x:  db 10
     int y = 20;              y: db 20;
int main()
{
   #Local variables
   int a = 100;
   int b = 65;
```

Add Two Integers Using Local Variables

```
; Add  two integres using local variables

SECTION .data                   ; Data section
  msg:     db "Add two numbers using local variables",10, 0
  format1: db "Value of the first Integer Number is %d", 10, 0
  format2: db "Value of the second Integer Number is %d", 10, 0
  format3: db "%d + %d is %d", 10, 0


section .text

 extern printf
 global main
 main:
        push ebp
        mov ebp, esp
       sub esp, 12

       mov DWORD [ebp - 4], 123d       ; variable -1
       mov DWORD [ebp - 8], 32d     ; variable-2
       mov DWORD [ebp - 12], 0d     ; variable-3

        push msg
        call    printf
```

```
; printf (format1, value)
push DWORD [ebp-4]
push   format1
call    printf

; printf (format2, value)
push DWORD [ebp-8]
push   format2
call    printf

mov eax, [ebp - 4]
add eax, [ebp - 8]
mov DWORD [ebp -12], eax

; printf (format3, value1, value2, value3)
push DWORD [ebp-12]
push DWORD [ebp-8]
push DWORD [ebp-4]
push format3
call    printf


mov esp, ebp
pop ebp
ret
```

```
root@kali-Test:~/Desktop/Week-10#  nasm -g -f elf Add2N3.asm -o Add2N3.o
root@kali-Test:~/Desktop/Week-10#
root@kali-Test:~/Desktop/Week-10#    gcc -m32 -lc Add2N3.o -o Add2N3
root@kali-Test:~/Desktop/Week-10#
root@kali-Test:~/Desktop/Week-10#    ./Add2N3
Add two numbers using local variables
Value of the first Integer Number is 123
Value of the second Integer Number is 32
123 + 32 is 155
```

Understand “scanf” Function

- Reads ASCII characters from stdin, matching characters to its first argument (format string), converting character sequences according to any formatting characters, and
- storing the converted values to the addresses specified by its data pointer arguments.
-
```
char name[100];
int Month, Day, Year;
float pi;

scanf("%s %d/ %d/ %d/ %1f",  name, &Month, &Day, &Year, &pi);
```

%d	Reads until first non-digit.
%x	Reads until first non-digit (in hex).
%s	Reads until first whitespace character.

Add two integers using scanf and printf

```
; Add  two integres using scanf and printf

SECTION .data                ; Data section
  msg:     db "Add two numbers using local variables and using scanf and printf",10, 0
  msg1: db " Please enter the first Integer Number", 10, 0
  format1: db "Value of the first Integer Number is %d", 10, 0
  formatScan: db "%d", 0     ;  for scanf
  msg2: db " Please enter the second Integer Number", 10, 0
  format2: db "Value of the second Integer Number is %d", 10, 0
  format3: db "%d + %d is %d", 10, 0

  N1:      dd 0
  N2:      dd 0
  N3:      dd 0


SECTION .text

 extern printf
 extern scanf
 global main
 main:
        push ebp
        mov ebp, esp
```

```
root@kali-Test:~/Desktop/Week-10# nasm -g -f elf Add2NScan.asm -o Add2NScan.o
root@kali-Test:~/Desktop/Week-10# gcc -m32 -lc Add2NScan.o -o Add2NScan
root@kali-Test:~/Desktop/Week-10# ./Add2NScan
Add two numbers using local variables and using scanf and printf
 Please enter the first Integer Number
33
 Please enter the second Integer Number
44
33 + 44 is 77
```

```
push msg
call    printf

push msg1
call    printf

; scanf (formtScan, value)
push N1
push formatScan
call scanf

push msg2
call    printf

; scanf (formtScan, value)
push N2
push formatScan
call scanf

mov eax, [N1]
add eax, [N2]
mov DWORD [N3], eax

; printf (format3, value1, value2, value3)
push DWORD [N3]
push DWORD [N2]
push DWORD [N1]
push format3
call    printf


mov esp, ebp
pop ebp
ret
```

Previous example with local variables

```
; Add  two integres using scanf, printf and local variables

SECTION .data                ; Data section
 msg:     db "Add two numbers using local variables and using scanf and printf",10, 0
 msg1: db " Please enter the first Integer Number", 10, 0
 format1: db "Value of the first Integer Number is %d", 10, 0
 formatScan: db "%d", 0     ;  for scanf
 msg2: db " Please enter the second Integer Number", 10, 0
 format2: db "Value of the second Integer Number is %d", 10, 0
 format3: db "%d + %d is %d", 10, 0


SECTION .text

 extern printf
 extern scanf
 global main
 main:
        push ebp
        mov ebp, esp
        sub esp, 12

        push msg
        call    printf
```

```
root@kali-Test:~/Desktop/Week-10# nasm -g -f elf Add2NScanlocal.asm -o Add2NScanlocal.o
root@kali-Test:~/Desktop/Week-10# gcc -m32 -lc Add2NScanlocal.o -o Add2NScanlocal
root@kali-Test:~/Desktop/Week-10# ./Add2NScanlocal
Add two numbers using local variables and using scanf and printf
 Please enter the first Integer Number
44
 Please enter the second Integer Number
22
44 + 22 is 66
```

```
push msg1
call    printf

; scanf (formtScan, value)
push DWORD [ebp-4]
push formatScan
call scanf

push msg2
call    printf

; scanf (formtScan, value)
push DWORD [ebp-8]
push formatScan
call scanf

mov eax, [ebp - 4]
add eax, [ebp - 8]
mov DWORD [ebp -12], eax

; printf (format3, value1, value2, value3)
push DWORD [ebp-12]
push DWORD [ebp-8]
push DWORD [ebp-4]
push format3
call    printf


mov esp, ebp
pop ebp
ret
```

Opening a file (fopen)

- Before using a file, you have to open it using fopen function

  fopen (char* name, char* mode);

- First argument: name
  - The name of the physical file, or how to locate it on the storage device.
- Second argument: mode
  - How the file will be used:

    "r" -- read from the file

    "w" -- write, starting at the beginning of the file

    "a" -- write, starting at the end of the file (append)

- Example

```
FILE *infile1,      *outfile2;
infile1 = “mydate.txt”;
outfile2 = “outfile.txt”;
fopen (infile1, “r”);    // open “mydata.txt” for reading
fopen (infile2, “w”);    // open “outfile.txt” for writing
fopen (infile2, “rw”);    // open “outfile.txt” for reading and writing
```

```
    fopen (infile1, “r”);    // open “mydata.txt” for reading
    fopen (infile2, “w”);    // open “outfile.txt” for writing
```

- Example in Assembly

```
section .data
        infile1  db  “mydate.txt”, 0
        inmode db “r”, 0
        outfile2 db  “outfile.txt”, 0
        outmode db “w”, 0
Section .text
    extern fopen
Main:
        push inmode
        push infile1
        call fopen
            ----
           -----
        push outmode
        push outfile2
        call fopen
```

Closing a file (fclose)

- After using a file, you have to close it using "fclose" function to notify the OS that you are done with the file

  fclose (char* name);

- First argument: name
  - The name of the physical file, or how to locate it on the storage device.

- Example

```
FILE *infile1,        *outfile2;
infile1 = "mydate.txt";
outfile2 = "outfile.txt"
fopen (infile1, "r");    // open "mydata.txt" for reading
fopen (infile2, "w");    // open "outfile.txt" for writing
fopen (infile2, "rw");    // open "outfile.txt" for reading and writing
  -----
  -----
fclose infile1;
fclose outfile2;
```

Open, read, and close file

```
        section .data
                openFile:       db 'File1.txt',0
                fileModeR:      db 'r',0
                buffer:         dd 0        ;  points to a memory location
        section .bss
                bufLength       resb 10     ; memory location with length of 10 characters
Section .text
        extern fopen, fgets, fclose
Main:
         ; To open file
                push fileModeR              ; push read mode
                push openFile               ; push file name
                call fopen
                mov ebx, eax                ; save the file handle in ebx
        ; to read the file
                push ebx                    ; push the file handle as a parameter
                push bufLength              ; push the length of the buffer as parameter
                push buffer                 ; push the buffer location as a parameter
                call fgets                  ; get the characters in the file
        ; close file and tear down the stack
                push ebx
                call fclose
                mov esp, ebp
                pop ebp
                ret
```

Open, write, and close file

```
        section .data
                writeFile:      db 'File2.txt',0
                fileModeW:      db ‘w’,0
                buffer:         dd 0            ; points to a memory location
                format1         db ‘%s, 0       ; write string format to a file
Section .text
        extern fopen, fprintf, fclose
Main:
         ; To open file
                push fileModeW              ; push read mode
                push writeFile              ; push file name
                call fopen
                mov ebx, eax                ; save the file handle in ebx

        ; to write to the file
                push buffer                 ; push the buffer location as a parameter
                push format1                ; push the writing format for fprintf
                push ebx                    ;push the file handle as a parameter
                call fprintf                ; get the characters in the file

        ; close file and tear down the stack
                push ebx
                call fclose
                mov esp, ebp
                pop ebp
                ret
```

Putting It All Together

You should know:

- Build a Program Stack
- Use external C-functions in assembly programs
  - fprintf and fscanf
  - Opening a file (fopen function)
  - fprintf
  - Closing a file (fclose function)

- **Adding integers using local variables**
- **Adding integers using scanf and printf**

Questions?

Coming Next Week
System Call:
Build I/O Assembly functions

Week 11 Assignments

- **Learning Materials**

  1- Week 11 Presentation

  2- Read Pages 452-462 & 487-493: Ch.13: Duntermann, Jeff. Assembly Language Step by Step, Programming with Linux

- **Assignment**

  1- Complete “Lab 11” by coming Sunday 11:59 PM.