X86 64bit assembly code to C interface

LBYARCH
Departmet of Computer Technology
College of Computer Studies
De La Salle University Manila

Outline

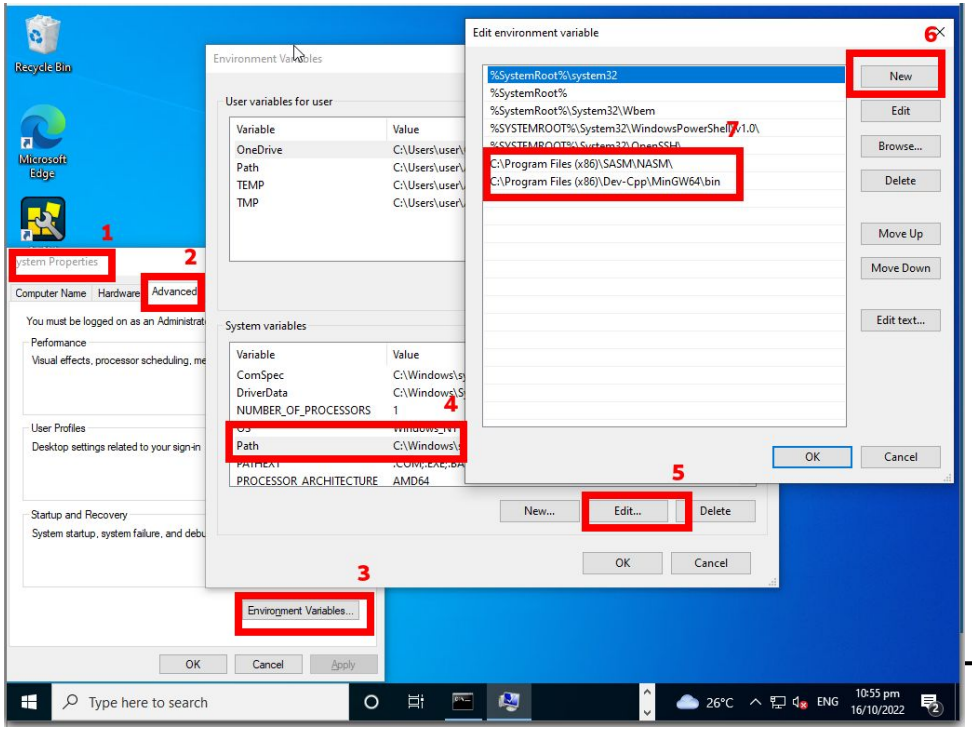
- Environment Configuration
- Call Conventions
- Register Volatility
- Parameter Passing
- x86 call C
- C call x86

Environment Configuration

Environment Configuration

- SASM
- NASM
  - Included in sasm
- Dev-Cpp TDM-GCC
  - Download the installer
- GCC
  - Included in dev-cpp tdm-gcc

Add NASM and GCC(MinGW64) To the system variable Path



- Locate nasm inside installed sasm folder
- Locate mingw bin inside installed dev-cpp

Call Conventions

Caller and Callee

- Caller - call to function
- Callee - the function

x86 assembly call C

- x86 asm caller
- C printf / scanf callee

C call x86 assembly

- C caller
- X86 asm callee

Register Volatility and Preservation

Volatile / caller-saved / scratch register

- Registers usually hold temporary information, that can be overwritten by any subroutine.
- Caller's responsibility to push each of these registers onto the stack, if it would like to restore their values after a subroutine call

Non-volatile / callee-saved / preserve register

- Registers are used to hold long-lived values, that should be preserved across calls.
- Callee's responsibility to both save (push at the beginning) and restore (pop accordingly) them before returning to the caller.

Register type	
Volatile	RAX, RCX, RDX, R8-R11, XMM0-XMM5, YMM0-YMM5
Non-volatile	RBX, RSI, RDI, RBP, RSP, R12-R15, XMM6-XMM15, YMM6-YMM15

Callee Adjustment

Address (by 8)	Data
	Shadow space 4
	Shadow space 3
	Shadow space 2
rsp→	Shadow space 1

The 1st 4 integer parameters are passed from the caller to callee as follows:

- RCX, RDX, R8, and R9 as 1st 2nd 3rd, and 4th parameter
- RAX as return value

The 1st four parameters are stored in a 32 bytes in the stack memory called “shadow space.”

First Parameters Different Data Types

- First few parameters, depending on the data type are stored on different (variant of) registers

parameter	long long int	Int	Short	Char	Float
1st	RCX	ECX	CX	CL	xmm0
2nd	RDX	EDX	DX	DL	xmm1
3rd	R8	R8D	R8W	R8B	xmm2
4th	R9	R9D	R9W	R9B	xmm3
return	RAX	EAX	AX	AL	xmm0

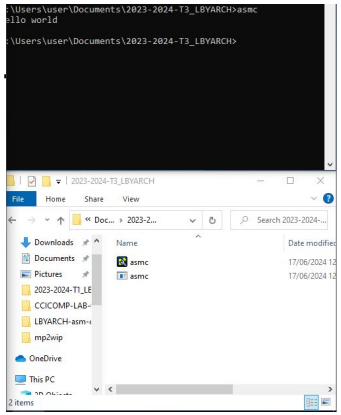
Windows
Based Parameter Passing

- RAX
  - Return value
- RCX, RDX, R8, R9
  - First 4 arguments
- Stack
  - Beyond 4th argument
  - Stack should be aligned in 16 byte boundary
- ***This Lecture is Windows Based***

Unix/MacOS/Linux Based Parameter Passing

- RAX
  - Return value
- RDI, RSI, RCX, RDX, R8, R9
  - First 6 arguments
- Stack
  - Beyond 6th argument
  - Stack should be aligned in 16 byte boundary

x86 asm call C



X86 call C: printf

- X86 assembly code that calls the printf function
- The string address is stored into rcx
- String address can be loaded either: (same effect)
  - lea rcx, [msg] or
  - mov rcx, msg
- Stack space allocated (subtracted to rsp) and freed (add to rsp) for shadow spaces and parameter passing

```
section .data
msg db "hello world",10,0

section .text
bits 64
default rel
global main
extern printf

main:
    sub rsp, 8*5
    lea rcx, [msg]
    call printf
    add rsp, 8*5

    xor rax, rax
    ret
```

Input

Output

hello
world

```
section .text                ; indicates a 64 bit asm code
bits 64
default rel
global main


extern printf                ; printf function
                             ; params passed via rcx, rdx, r8, r9, stack
                             ; Rcx - first parameter - string

main:
        sub rsp, 8*5         ; allocates 8 byte aligned spaces for stack
        …
        add rsp, 8*5
```

Assemble and run

Method 1: in sasm

File > save as .exe

Method 2: 3 commands in cmd

> nasm -f win64 asmfile.asm

> gcc -m64 asmfile.obj -o asmfile.exe

> asmfile.exe

x86 to C: Printf Many Parameter Pass

```
section .data
msg1 db "double %lf %lf %lf %lf %lf %lf",10,0
var1 dq 1.5
var2 dq 2.5
var3 dq 3.5
var4 dq 4.5
var5 dq 5.5
var6 dq 6.5
```

```
section .text
bits 64
default rel
global main
extern printf

main:
    sub rsp, 8*7
    mov rax, [var6]
    mov [rsp+48], rax

    mov rax, [var5]
    mov [rsp+40], rax

    mov rax, [var4]
    mov [rsp+32], rax

    mov r9, [var3]
    mov r8, [var2]
    mov rdx, [var1]
    mov rcx, msg1
    call printf
    add rsp, 8*7

    xor rax, rax
    ret
```

- First 4 parameters
  - Rcx = msg1
  - Rdx = var1
  - R8 = var2
  - R9 = var4
- Parameter 5 onwards
  - stack

```
\Users\user\Documents\2023-2024-T3_LBYARCH>asmc2
uble 1.500000 2.500000 3.500000 4.500000 5.500000 6.500000

\Users\user\Documents\2023-2024-T3_LBYARCH>
```

```
section .data
prompt1 db "enter double: ",0
scanformat db "%lf",0
inputdouble dq 0
prompt2 db "value entered: %lf",10,0
```

```
section .text
bits 64
default rel
global main
extern printf, scanf

main:
    ; print prompt1
    sub rsp, 8*5
    lea rcx, [prompt1]
    call printf
    add rsp, 8*5

    ; scanf
    sub rsp, 8*5
    lea rdx, [inputdouble]
    lea rcx, [scanformat]
    call scanf
    add rsp, 8*5

    ; print prompt2
    sub rsp, 8*5
    mov rdx, [inputdouble]
    lea rcx, [prompt2]
    call printf
    add rsp, 8*5

    xor rax, rax
    ret
```

X86 call C: printf and scanf

Printf

- prints string format with additional parameters
- printf(“value entered %lf”,inputdouble); translated as (see third code block)

Scanf

- data from command line interpreted using string formatter %d, %s, %f, %lf

C call x86 asm

C code

```c
#include <stdio.h>
extern void asmhello();

int main()
{
    asmhello();
    return 0;
}
```

C to x86: No Parameters

- C - caller
- X86 assembly - callee

Asm code

```
section .data
msg db `hello world \n`,0

section .text
bits 64
default rel
global asmhello
extern printf

asmhello:
    sub rsp, 8*5
    mov rcx, msg
    call printf
    add rsp, 8*5

    ret
```

```
s>cfile
hello world
```

Compile Assemble and Run

```
> nasm -f win64 asmfile.asm

> gcc -c cfile.c -o cfile.obj -m64

> gcc cfile.obj asmfile.obj -o cfile.exe -m64

> cfile.exe
```

C code

```c
int main()
{
    int a=1, b=2, c=3;
    asmfunc(a,b,c);
    return 0;
}
```

C to x86: Few Parameters

Asm code

```
section .data
msg db "params a=%d b=%d c=%d",0

section .text
bits 64
default rel
global asmfunc
extern printf

asmfunc:
    ; parameter received a@rcx, b@rdx, c@r8

    ; printf parameter printf msg, a, b, c
    sub rsp, 8*5
    mov r9, r8
    mov r8, rdx
    mov rdx, rcx
    lea rcx, [msg]
    call printf
    add rsp, 8*5

    ret
```

```
C:\Users\user\Documents\2023-2024-T3_LBYARCH\CToAsm-2-FewParams>compileassemblerun.bat

C:\Users\user\Documents\2023-2024-T3_LBYARCH\CToAsm-2-FewParams>nasm -f win64 asmfile.asm

C:\Users\user\Documents\2023-2024-T3_LBYARCH\CToAsm-2-FewParams>gcc -c cfile.c -o cfile.obj -m64

C:\Users\user\Documents\2023-2024-T3_LBYARCH\CToAsm-2-FewParams>gcc cfile.obj asmfile.obj -o cfile.exe -m64

C:\Users\user\Documents\2023-2024-T3_LBYARCH\CToAsm-2-FewParams>cfile.exe
params a=1 b=2 c=3
C:\Users\user\Documents\2023-2024-T3_LBYARCH\CToAsm-2-FewParams>
```

- Few parameters, 4 and less
  - Parameters in: rcx, rdx, r8,and r9
- Parameters received from c
  - A, b, c residing at rcx, rdx, r8
- Parameters send to printf
  - Rcx = string format
  - Rdx = a, currently residing in rcx
  - R8 = a, currently residing in rdx
  - R9 = a, currently residing in r8

C code

```c
#include <stdio.h>
#include <stdlib.h>

extern int asmsum(int a, int b);

int main()
{
    int a = 1;
    int b = 2;
    int c = asmsum(a, b);
    printf("sum: %d",c);
    return 0;
}
```

Asm Code

```asm
section .text
bits 64
default rel
global asmsum

asmsum:
    ; a@rcx, b@rdx
    mov rax, rcx
    add rax, rdx
    ; return value rax
    ret
```

```
cfile.exe
sum: 3
```

C to x86: Integer Return value

- Parameter passing via registers (and stack)
- Return value always RAX

C to x86: Float Return Value

- For floating point values
- Parameters are xmm0 xmm1 xmm2 xmm3
- Return value is on xmm0

C code

```c
#include <stdio.h>
#include <stdlib.h>

extern double asmsum(double a, double b);

int main()
{
    double a = 1;
    double b = 2;
    double c = asmsum(a, b);
    printf("sum: %lf",c);
    return 0;
}
```

sm Code

```
section .text
bits 64
default rel
global asmsum

asmsum:
    ; a@xmm0, b@xmm1
    addsd xmm0, xmm1
    ; return value xmm0
    ret
```

```
file.exe
sum: 3.000000
```

```c
#include <stdio.h>
#include <stdlib.h>

extern int asmsum(int a, int b, int c, int d, int e, int f);

int main()
{
    int a=1, b=2, c=3, d=4, e=5, f=6;
    int g = asmsum(a,b,c,d,e,f);
    printf("sum: %d",g);
    return 0;
}
```

```asm
section .text
bits 64
default rel
global asmsum

asmsum:
    push rsi
    push rbp
    mov rbp, rsp
    add rbp, 16

    ; a@rcx, b@rdx, c@r8, d@r9, e,f@stack
    mov rax, rcx
    add rax, rdx
    add rax, r8
    add rax, r9
    add rax, [rbp+40]
    add rax, [rbp+48]
    ; return value rax

    pop rbp
    pop rsi
    ret
```

C to c86: Many Parameters

- Many, 5 and more
  - First 4 parameters in: rcx, rdx, r8, r9
  - Succeeding parameters in stack
- In example: receiving parameters
  - Parameters sent by c code are int a, b, c, d, e, f, g
  - a, b, c, d were in rcx, rdx, r8, r9
  - e, f are in stack
  - Stack starts at rsp(or rbp)+40, then increments of 8
  - 40 = 4*8 shadow addresses bytes + 8 bytes for return address
- Good Practice
  - Preserve rsp, operate on rbp
  - Preserve original rbp by pushing to the stack, then pop afterwards

C code

```c
#include <stdio.h>
#include <stdlib.h>

extern void vecadd(int n, int* arr1, int* arr2, int*arr3);

int main()
{
    int vec1[] = {10, 20, 30, 40};
    int vec2[] = {1, 2, 3, 4};
    int* vec3 = (int*)malloc(4*sizeof(int));
    int n = 4;
    vecadd(n, vec1, vec2, vec3);

    int i;
    for(i = 0; i < n; i++)
        printf("%d ",vec3[i]);
    return 0;
}
```

C to X86: Vector Addition

- Parameters:
  - N number of elements
  - vec1 and vec2 initialized array
  - vec3 allocated result space
- 1st parameter N received at rcx
- 2nd, 3rd, and 4th parameters are addresses of vec1, vec2, and vec3
- Each element are increments of 4 because int is 4 bytes
- Addition result placed on memory space of vec3

Asm Code

```
section .text
bits 64
default rel
global vecadd

vecadd:
    ; n@rcx, vec1@rdx, vec2@r8, vec3@r9
    L1:
        mov rax, [rdx]
        mov rbx, [r8]
        add rax, rbx
        mov [r9], rax
        add rdx, 4
        add r8, 4
        add r9, 4
        loop L1
    ret
```

```
cfile.exe
11 22 33 44
```

cfile.c

```c
#include <stdio.h>
extern void asmfunc();

int main()
{
    int a=1, b=2, c=3, d=4, e=5, f=6, g=7, h=8, i=9;
    asmfunc(a,b,c,d,e,f,g,h,i);
    return 0;
}
```

C to x86: Many Parameters, simultaneos stack use

asmfile.asm | cfile.c

```
8   extern printf
9
10  asmfunc:
11      ; parameter received a@rcx, b@rdx, c@r8, d@r9, e,f,g,h@stack
12
13      ; printf parameter printf msg, a, b, c, d, e, f, h, i
14      sub rsp, 8*10
15
16
17      mov rax, [rsp+150]  ; received as 5th param: rsp + (8*9) + (8*10)
18      mov [rsp+70], rax   ; printf param10 i
19
20      mov rax, [rsp+142]  ; received as 5th param: rsp + (8*8) + (8*10)
21      mov [rsp+62], rax   ; printf param9 h
22
23      mov rax, [rsp+136]  ; received as 5th param: rsp + (8*7) + (8*10)
24      mov [rsp+56], rax   ; printf param8 g
25
26      mov rax, [rsp+128]  ; received as 5th param: rsp + (8*6) + (8*10)
27      mov [rsp+48], rax   ; printf param7 f
28
29      mov rax, [rsp+120]  ; received as 5th param: rsp + (8*5) + (8*10)
30      mov [rsp+40], rax   ; printf param6 e
31
32      mov [rsp+32], r9    ; printf param5 d
33
34      mov r9, r8          ; printf param4 c
35      mov r8, rdx         ; printf param3 b
36      mov rdx, rcx        ; printf param2 a
37      lea rcx, [msg]      ; printf param1 msg
38      call printf
39      add rsp, 8*10
40
41      ret
```

- Simultaneous stack use
  - As receiving parameters
  - As sending parameters to printf
- In example: receiving parameters
  - Parameters sent by c code are int a, b, c, d, e, f, g, h
  - a, b, c, d were in rcx, rdx, r8, r9
  - e, f, g, h, are in stack
  - Stack starts at rsp+120, then increments of 8
    - Calculated with starting rsp +80(new parameters stack adjust), +40(skip old stack shadow and return address)
- In example: sending parameters to printf
  - 1st printf parameter
    - Rcx = string format msg
  - 2nd-4th printf parameter
    - Rdx, r8, r9 = a, b, c residing on rcx, rdx, r8
  - Succeeding parameters
    - Stack = d residing on r9
    - Stack = e, f, g, h residing on stack
    - Pass using the stack +32 (skipping shadow)