CSC 406: Computer System I, 2019 Winter,

Assignment #3

Purpose:

To:

1. Go over the basics of assembly language
2. Go over how to use a debugger
3. Go over the layout of an activation record

Assignment

Please do the following:

1. Download the program called `toAnalyzeCDM.zip` from COL
2. Use an `sftp` program like filezilla to upload it to a ctilinux machine (like `140.192.36.184`) Do *not* bother unzipping it on your local machine.
3. On the same machine unzip it with:
4. `$ unzip toAnalyzeCDM.zip`
5. Do `chmod u+x toAnalyze` to make tell Unix that it is an executable program
6. Analyze it with gdb: `gdb toAnalyze`. It has a structure like:

```
int bar (/* some number of args */)
{
  /* Some number of local vars */

  return( /* something */ );
}


int foo (/* some number of args */)
{
  /* Some if-statement */

  /* Some number of local vars */

```

```
21.  /* Some code, including call(s) to bar() */
22.
23.  return( /* something */ );
24.}
25.
26.
27.int main ()
28.{
29.  /* Some number of local vars, including call(s) to foo() */
30.  return(0);
31.}
```

Answer the following:

1. **(20 Points) Assembly language understanding (1):**

   The assembly language for `bar()` is:

```
(gdb) disass bar
Dump of assembler code for function bar:
   0x00000000004004cd <+0>:     push   %rbp
   0x00000000004004ce <+1>:     mov    %rsp,%rbp
   0x00000000004004d1 <+4>:     mov    %edi,-0x14(%rbp)
   0x00000000004004d4 <+7>:     mov    -0x14(%rbp),%eax
   0x00000000004004d7 <+10>:    add    %eax,%eax
   0x00000000004004d9 <+12>:    mov    %eax,-0x4(%rbp)
   0x00000000004004dc <+15>:    mov    -0x4(%rbp),%eax
   0x00000000004004df <+18>:    pop    %rbp
   0x00000000004004e0 <+19>:    retq
End of assembler dump.
```

   Give a 1-2 sentence description of the purpose of each instruction. **I am more interested in the *why* than the *what*.**

Instruction:	Purpose:
push %rbp	_____

mov %rsp,%rbp	_____
mov %edi,-0x14(%rbp)	_____
mov -0x14(%rbp),%eax	_____
add %eax,%eax	_____
mov %eax,-0x4(%rbp)	_____
mov -0x4(%rbp),%eax	_____
pop %rbp	_____
retq	_____

2. **(10 Points) Assembly language understanding (2):**

   Write a C function that does what `bar()` does.
   You won't be able to figure out the names of my parameters var(s) and local var(s); just make up your own name(s).

3. **(20 Points) Activation Records (1):**

   Stop the program at its *second* call to `bar()`. When I did so I got the following:

```
(gdb) break bar
Breakpoint 1 at 0x4004d1
(gdb) run
Starting program:
/home/instructor/Documents/Academic/DePaul/Classes/CSC373/20178
-4SumI/Assign3/toAnalyze
```

```
Breakpoint 1, 0x00000000004004d1 in bar ()
Missing separate debuginfos, use: debuginfo-install glibc-2.17-
222.el7.x86_64
(gdb) c
Continuing.

Breakpoint 1, 0x00000000004004d1 in bar ()
(gdb) stepi
0x00000000004004d4 in bar ()
(gdb) stepi
0x00000000004004d7 in bar ()
(gdb) stepi
0x00000000004004d9 in bar ()
(gdb) stepi
0x00000000004004dc in bar ()
(gdb) stepi
0x00000000004004df in bar ()
(gdb) info reg
rax            0x4      4
rbx            0x0      0
rcx            0x400560 4195680
rdx            0x2      2
rsi            0x4      4
rdi            0x2      2
rbp            0x7fffffffdbd0   0x7fffffffdbd0
rsp            0x7fffffffdbd0   0x7fffffffdbd0
r8             0x7ffff7dd5e80   140737351868032
r9             0x0      0
r10            0x7fffffffd760   140737488344928
r11            0x7ffff7a30350   140737348043600
r12            0x4003e0 4195296
r13            0x7fffffffdd30   140737488346416
r14            0x0      0
r15            0x0      0
rip            0x4004df 0x4004df <bar+18>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
```

ds	0x0	0
es	0x0	0
fs	0x0	0
gs	0x0	0

Write the activation record for `bar()` when `%rip` gets to 0x00000000004004df.
Under **Value** put the numeric value held at that address.
Under **Purpose** put one of the following:

a. not part of `bar()`'s activation record
b. argument to `bar()`
c. the address inx `foo()` to which `rip` should return
d. the stored `rbp` address for `foo()`
e. local variable to `bar()`
f. in the activation record of `bar()`, but not used

	Address:	Value:	Purpose:
	rbp + 0x10	_____	_____
	rbp + 0xC	_____	_____
	rbp + 0x8	_____	_____
	rbp + 0x4	_____	_____
rbp -->	rbp + 0x0	_____	_____
	rbp - 0x4	_____	_____
	rbp - 0x8	_____	_____
	rbp - 0xC	_____	_____
	rbp - 0x10	_____	_____
	rbp - 0x14	_____	_____
	rbp - 0x18	_____	_____

4. (10 Points) Assembly language understanding (3):

What are the value(s) that `foo()` obtains as arguments from `main()`? In which registers are they passed?

Give offset(s) from `rbp` from within `foo()`'s activation record *or* give the name(s) of the registers.

5. **(10 Points) Assembly language understanding (4):**

   How many *local variables* does `foo()` have?
   Where are they on the stack?
   Give an offset from `rbp` from within `foo()`'s activation record.

6. **(20 Points) Debugger usage (1):**

   `foo()` has a recursive call. Inside of `foo()` what are the values that both its *arguments* and *local variables* take on when rip is at address `0x00400518`? At the *top* of the table give the offset from `rbp` (the hexadecimal number added to `rbp` to get the address of the variable) of the parameter or local variable. (I may have tried to fool you the the number of variables.)

   In the body of the table write the values that that variable has when you hit address *local variables*.

Call:	rbp + _____	rbp + _____	rbp + _____	rbp + _____	rbp + _____	rbp + _____
1	_____	_____	_____	_____	_____	_____
2	_____	_____	_____	_____	_____	_____

7. **(5 Points) Debugger usage (2):**

   What value does `foo()` return to `main()`?

8. **(5 Points) Assembly language understanding (5):**

   `foo()` calls `bar()`. `bar()` starts at address 0x004004CD. If you look at the machine code for `foo()`'s call to `bar()`, however, you'll see that the actual number in the function call is 0xFFFF,FFB8

```
0x04004e1 <foo>:
  . . .
  40050e: 89 c7                 mov    %eax,%edi
  400510: e8 b8 ff ff ff        callq  4004cd <bar>
```

```
400515:89 45 f8                 mov     %eax,-0x8(%rbp)
. . .
```

a. What to what number did the CPU add with 0xFFFF,FFB8 to get the address of `bar()`, 0x0040,04CD?
b. Do this addition. Compute 0x0040,04CD.