Insper

10 - Prática de Engenharia Reversa

Sistemas Hardware-Software - 2020/1

Igor Montagner

As aulas de hoje e quinta serão dedicadas a revisar conceitos básicos vistos nas últimas 5 aulas. A parte 1 repassa ponteiros e variáveis globais. A parte 2 contém exercícios intermediários que misturam dois conceitos vistos em aula (ex: loops e `LEA` ou funções + condicionais). Está indicado ao lado de cada exercício quais conceitos são exercitados e todos eles serão recebidos via repositório de atividades da disciplina.

Atenção: acesse a aba "Repositório de Atividades" no Teams para indicar seu usuário do blackboard e endereço do repositório privado no Github.

Todo exercício para entrega deverá ser colocado no repositório de atividades da disciplina.

Parte 1 - Ponteiros e variáveis globais

Todos os exercícios da revisão serão feitos com o arquivo `exemplo1` (compilado a partir de `exemplo1.c`).

Ponteiros

```
Dump of assembler code for function muda_valor:
   0x05fa <+0>:     addl   $0x1,0x200a0f(%rip)        # 0x201010 <var_global>
   0x0601 <+7>:     lea    (%rsi,%rsi,2),%eax
   0x0604 <+10>:    add    %eax,%edi
   0x0606 <+12>:    mov    %edi,(%rdx)
   0x0608 <+14>:    lea    (%rdi,%rdi,2),%eax
   0x060b <+17>:    add    %eax,0x2009ff(%rip)        # 0x201010 <var_global>
   0x0611 <+23>:    retq
```

Nas linhas `+7` até `+12` podemos ver claramente a diferença entre a instrução `LEA` e `MOV`

- linha `+7`: é colocado no **registrador** `%eax` o **valor** `%rsi + 2*%rsi`.
- linha `+12`: é colocado na **memória**, no endereço gravado em `%rdx`, o valor do registrador **%edi** (4 bytes)

Ou seja, um `MOV` que se utiliza de parênteses represente o operador *variável apontada por* (*) em *C*. Um `LEA` nunca acessa a memória.

Exercício: Levando as informações acima em conta, faça a tradução das linhas `+7` até `+14` de `muda_valor`

Vamos agora analisar as linhas `+0` e `+17`:

```
0x05fa <+0>:     addl   $0x1,0x200a0f(%rip)        # 0x201010 <var_global>
0x0601 <+7>:     lea    (%rsi,%rsi,2),%eax

0x060b <+17>:    add    %eax,0x2009ff(%rip)        # 0x201010 <var_global>
0x0611 <+23>:    retq
```

O parênteses indica que estamos mexendo na memória e o fato de estarmos usando o registrador `%rip` indica que os dados apontados são globais. Ou seja, eles tem visibilidade no programa todo e existem durante toda a execução do programa. Este cálculo é feito usando deslocamentos relativos ao endereço da instrução atual. Vejamos um exemplo no caso da linha `+0`.

1. Quando a CPU executa a linha `+0` o registrador `%rip` aponta para a linha seguinte (`0x0601`).
2. O resultado do lado direito do `addl` pede acesso a memória na posição `%rip + 0x200a0f`
3. Ou seja, como `%rip = 0x0601`, o valor que queremos acessar está no endereço de memória `0x0601 + 0x200a0f = 0x201010`
4. Note que o *gdb* aponta o valor calculado no lado direito da instrução juntamente com o nome da variável global. Este mesmo nome apareceria quando usamos o comando `info variables`

Logo, a tradução da linha `+0` é simplesmente `var_global++`.

Exercício: Traduza o programa completo abaixo.

Lembre-se de que, ao rodar o programa, os endereços calculados podem mudar. Ou seja, na hora de analisar o programa rodando usando o *gdb* é sempre melhor usar o comando `b` para parar o programa onde você quiser e o comando `x` para mostrar dados na memória.

Parte 2 - exercícios intermediários

Os exercícios desta seção exercitam mais de um conceito ao mesmo tempo. Cada um deles é disponibilizado via um arquivo `exI.o` na pasta `10-revisao-II` no repositório de atividades. A soluções devem ser colocadas no arquivo `solucao_exI.c` correspondente. Veja as instruções em cada arquivo para garantir que está implementando a função correta.

Importante: cada exercício estará disponível em uma página do handout de revisão juntamente com questões "padrão" para cada assunto. Essas questões são feitas para ajudar na compreensão dos programas. Faça-as com atenção e facilite sua vida.

Exercício: A função abaixo exercita os assuntos **Variáveis globais** e **Loops**. Seu código completo está disponível no arquivo *ex1-sem-teste*.

```
Dump of assembler code for function ex1:
   0x0616 <+0>:     push   %rbx
   0x0617 <+1>:     mov    $0x0,%ebx
   0x061c <+6>:     jmp    0x62b <ex1+21>
   0x061e <+8>:     mov    $0x0,%eax
   0x0623 <+13>:    callq  0x5fa <faz_algo>
   0x0628 <+18>:    add    $0x1,%ebx
   0x062b <+21>:    cmpq   $0x0,0x2009dd(%rip)        # 0x201010 <var1>
   0x0633 <+29>:    jg     0x61e <ex1+8>
   0x0635 <+31>:    mov    %ebx,%eax
   0x0637 <+33>:    pop    %rbx
   0x0638 <+34>:    retq
```

1. A função acima recebe argumentos? Ela retorna algo? Declare-a abaixo. Se houverem outras funções no arquivo, declare-as também no espaço abaixo.

2. Identifique todos os lugares em que uma variável global é usada. .

3. Use setas nas instruções de *jmp*. Você consegue identificar um loop? Entre quais linhas?

4. Qual a condição testada?

5. Faça uma tradução usando `if+goto` de `ex1`

Usando as perguntas acima preencha o arquivo de solução no repositório e execute os testes. Você pode supor que a função `faz_algo` existe.

Exercício: A função abaixo exercita os assuntos **Ponteiros** e **Condicionais**.

```
Dump of assembler code for function ex2:
   0x05fa <+0>:     mov    (%rdx),%rax
   0x05fd <+3>:     cmp    %rax,(%rdi)
   0x0600 <+6>:     jg     0x60d <ex2+19>
   0x0602 <+8>:     add    $0x8,%rdx
   0x0606 <+12>:    add    (%rdx),%rsi
   0x0609 <+15>:    mov    %rsi,(%rdi)
   0x060c <+18>:    retq
   0x060d <+19>:    lea    (%rsi,%rsi,2),%rsi
   0x0611 <+23>:    jmp    0x606 <ex2+12>
```

1. Quais argumentos são recebidos pela função? Quais são seus tipos? Existe valor de retorno? Declare a função abaixo.

2. Desenhe setas indicando o fluxo do programa. Você consegue identificar a condição testada no `if`?

3. O programa acima é um pouco bagunçado. Qual bloco de código é rodado quando a condição acima é verdadeira?

4. Traduza o programa acima linha a linha. Fique atento às instruções `MOV` cujos operandos usem parênteses!

Com base nas respostas acima escreva seu programa completo no repositório de atividades.

Exercício: A função abaixo exercita os assuntos **Ponteiros** e **Variáveis globais**.

```
Dump of assembler code for function ex3:
   0x05fa <+0>:     mov    0x200a14(%rip),%eax        # 0x201014 <var1>
   0x0600 <+6>:     lea    (%rax,%rax,4),%eax
   0x0603 <+9>:     mov    0x200a07(%rip),%edx        # 0x201010 <var2>
   0x0609 <+15>:    lea    (%rdx,%rdx,4),%ecx
   0x060c <+18>:    lea    (%rcx,%rcx,1),%edx
   0x060f <+21>:    add    %edx,%eax
   0x0611 <+23>:    mov    %eax,(%rdi)
   0x0613 <+25>:    add    0x2009fb(%rip),%eax        # 0x201014 <var1>
   0x0619 <+31>:    mov    %eax,0x2009f1(%rip)        # 0x201010 <var2>
```

1. Identifique quantas variáveis globais existem e onde elas são usadas.
2. A função acima recebe argumentos? Quantos e quais seus tipos? Ela retorna algo? Declare-a abaixo

3. Faça uma tradução linha a linha do programa acima, levando em conta os tamanhos corretos dos dados.

Com base nas respostas acima escreva seu programa completo no repositório de atividades.