Projeto Big Two

Análise do Código Gerado

Francisco Oliveira Ricardo Neves Vitor Peixoto

20 Maio 2016

Laboratórios de Informática II

Mestrado Integrado em Engenharia Informática

Capítulo 1

Análise do Código Gerado

1.1 Exercício 4

Disassemble contar_valores:

```
080483c4 <contar_valores>:
 80483c4:   push    %ebp
 80483c5:   mov     %esp,%ebp
 80483c7:   push    %edi
 80483c8:   push    %esi
 80483c9:   push    %ebx
 80483ca:   mov     0x8(%ebp),%esi
 80483cd:   mov     0xc(%ebp),%edi
 80483d0:   mov     0x10(%ebp),%ecx
 80483d3:   xor     %ebx,%ebx
 80483d5:   xor     %edx,%edx
 80483d7:   nop
 80483d8:   movl    $0x0,(%ecx,%edx,4)
 80483df:   inc     %edx
 80483e0:   cmp     $0xc,%edx
 80483e3:   jle     80483d8 <contar_valores+0x14>
 80483e5:   xor     %edx,%edx
 80483e7:   cmp     %edi,%edx
 80483e9:   jge     8048405 <contar_valores+0x41>
 80483eb:   nop
 80483ec:   mov     0x4(%esi,%edx,8),%eax
 80483f0:   incl    (%ecx,%eax,4)
 80483f3:   mov     0x4(%esi,%edx,8),%eax
 80483f7:   mov     (%ecx,%eax,4),%eax
 80483fa:   cmp     %ebx,%eax
 80483fc:   jle     8048400 <contar_valores+0x3c>
 80483fe:   mov     %eax,%ebx
 8048400:   inc     %edx
 8048401:   cmp     %edi,%edx
 8048403:   jl      80483ec <contar_valores+0x28>
 8048405:   mov     %ebx,%eax
 8048407:   pop     %ebx
 8048408:   pop     %esi
 8048409:   pop     %edi
 804840a:   leave
 804840b:   ret
```

1.2 Exercício 5

REGISTO	VARIÁVEL	OBSERVAÇÕES
`%esi`	`mao`	– – –
`%edi`	`tam`	– – –
`%ecx`	`*valores`	– – –
`%edx`	`i`	– – –
`%ebx`	`maior`	Variável é movida para `%eax` perto do fim da função.
`%eax`	`mao[i]` → `valores[mao[i]]` → `maior`	Registo armazena `mao[i]` temporariamente, armazenando depois o novo array `valores[mao[i]]`. Por fim é movido o valor da variável `maior`, para ser retornado no fim da função.
– – –	`c`	Variável não é usada no código, logo foi ignorada pelo Assembly.

1.3 Exercício 6

O *breakpoint* irá ser criado na máquina virtual de SC, com recurso à ferramenta *GDB*.

Criar um breakpoint:

```
[a79175@sc ~]$ gcc -O2 contar.c
[a79175@sc ~]$ gdb a.out
(gdb) break *0x08048407
Breakpoint 1 at 0x8048407
```

1.4 Exercício 7

Após o *breakpoint* criado no endereço `0x08048407`. Executamos a função, e no break, pedimos a informação dos registos, com o comando `info registers` onde iremos buscar informação acerca do registo `%ebp`:

Informação dos registos:

```
(gdb) info registers
eax            0x3          3
ecx            0xbfffe800   -1073747968
edx            0xd          13
ebx            0x3          3
esp            0xbfffe7cc   0xbfffe7cc
ebp            0xbfffe7d8   0xbfffe7d8
esi            0xbfffe840   -1073747904
edi            0xd          13
eip            0x8048407    0x8048407 <contar_valores+67>
eflags         0x200246     [ PF ZF IF ID ]
cs             0x73         115
ss             0x7b         123
ds             0x7b         123
es             0x7b         123
fs             0x0          0
gs             0x33         51
```

A partir daqui, sabemos onde estão localizados os parâmetros na stack, nomeadamente o apontador para `valores`. Sabendo isso, saberemos onde estão localizados o *array* de valores na memória. O valor de `%ebp` é então `0xbfffe7d8`. Tendo em conta os conhecimentos da stack, sabemos que o 3º parâmetro (`*valores`) é `0xbfffe7e8`.

ENDEREÇO NA MEMÓRIA	**VALOR**	**DESCRIÇÃO**
`0xbfffe7d8`	`0xbfffe8b8`	Antigo `%ebp`
`0xbfffe7db`	`0x0804843f`	Endereço de retorno
`0xbfffe7e0`	`0xbfffe840`	1º Parametro (`mao`)
`0xbfffe7e4`	`0x0000000d` = 13	2º Parametro (`tam`)
`0xbfffe7e8`	`0xbfffe800`	3º Parametro (`*valores`)

Como podemos observar, o valor de `*valores` é `0xbfffe800`. Então iremos analisar os valores que este *array* irá tomar:

Valores do array `valores`:

```
(gdb) x /13wd 0xbfffe800
0xbfffe800:     2       1       1       1
0xbfffe810:     1       2       0       3
0xbfffe820:     0       0       0       1
0xbfffe830:     1
```

Assim descobrimos os valores do *array* `valores`:
valores[0]=2
valores[1]=1
valores[2]=1
valores[3]=1
valores[4]=1
valores[5]=2
valores[6]=0
valores[7]=3
valores[8]=0
valores[9]=0
valores[10]=0
valores[11]=1
valores[12]=1

Sendo que cada inteiro (posição do *array*) ocupa 4 *bytes*, então, o *array* irá ocupar $13 * 4 = 52$ *bytes*.

1.5 Exercício 8

A indexação da matriz ocorre no segundo ciclo `for`. Aqui, a variável `i` toma o valor de todos os números naturais incluindo o 0 (zero) até ao número 12, que é o tamanho do *array* `valores`.

As linhas do código *Assembly* são:

```
0x080483e7:  cmp   %edi,%edx
```
Compara `i` a `tam`.

```
0x080483ec:  mov   0x4(%esi,%edx,8),%eax
```
`%eax` guarda `mao[i]`.

```
0x080483f0:  incl  (%ecx,%eax,4)
```
Incrementa 1 a `[valores[mao[i]]`.

1.6 Exercício 9

Disassemble contar_valores:

```
080483c4 <contar_valores>:
 80483c4:  push  %ebp
 80483c5:  mov   %esp,%ebp
 80483c7:  push  %edi                      | Salvaguarda
 80483c8:  push  %esi                      | os registos
 80483c9:  push  %ebx                      | callee saved.

 80483ca:  mov   0x8(%ebp),%esi            | Vai buscar
 80483cd:  mov   0xc(%ebp),%edi            | os parâmetros
 80483d0:  mov   0x10(%ebp),%ecx           | à stack.

 80483d3:  xor   %ebx,%ebx                 | maior=0
 80483d5:  xor   %edx,%edx                 | i=0
 80483d7:  nop
 80483d8:  movl  $0x0,(%ecx,%edx,4)        | Atribui valor 0 a todas
                                           | as posições do array
                                           | valores.

 80483df:  inc   %edx                      | i++
 80483e0:  cmp   $0xc,%edx                 | Compara a variável "i"ao
                                           | valor absoluto 13.
 80483e3:  jle   80483d8 <cont...+0x14>    | Salta se i≤12[a].


 80483e5:  xor   %edx,%edx                 | i=0
 80483e7:  cmp   %edi,%edx                 | Compara "i" a "tam".
 80483e9:  jge   8048405 <cont...+0x41>    | Salta se i≥tam.
 80483eb:  nop
 80483ec:  mov   0x4(%esi,%edx,8),%eax     | %eax guarda mao[i].
 80483f0:  incl  (%ecx,%eax,4)             | valores[mao[i]]++
 80483f3:  mov   0x4(%esi,%edx,8),%eax     | %eax guarda mao[i].
 80483f7:  mov   (%ecx,%eax,4),%eax        | %eax guarda
                                           | "valores[mao[i]]".

 80483fa:  cmp   %ebx,%eax                 | Compara "maior" e
                                           | "valores[mao[i]]".
 80483fc:  jle   8048400 <cont...+0x3c>    | Salta se
                                           | valores[mao[i]]≤maior
 80483fe:  mov   %eax,%ebx                 | Move valores[mao[i]] para
                                           | o registo %ebx.
 8048400:  inc   %edx                      | i++ (do ciclo for).
```

[a] Este salto refere-se à condição `i<13` que está no primeiro ciclo `for`. Neste caso, o salto é feito, se `i`≤`12` (para continuar no ciclo `for`) que é matematicamente igual a dizer `i<13`.

Disassemble contar_valores: (continuação)

```
8048401:  cmp   %edi,%edx                | Compara "i"a "tam".
8048403:  jl    80483ec <cont...+0x28>   | Se i<tam, salta (entra de
                                         | novo no ciclo for).
8048405:  mov   %ebx,%eax                | Move o valor de
                                         | "maior" para %eax.

8048407:  pop   %ebx                     | Resgata os
8048408:  pop   %esi                     | registos
8048409:  pop   %edi                     | callee saved.
804840a:  leave
804840b:  ret
```