Assembly do IA-32 em ambiente Linux

TPC6 e Guião laboratorial

Baseado no guia de Alberto José Proença

Objectivo e notas

Os exercícios propostos no TPC6 introduzem o **suporte a funções em C**, no IA-32, e a utilização de um depurador (*debugger*). Estes exercícios devem ser realizados no <u>servidor remoto</u> que foi usado na sessão laboratorial anterior (ver TPC4).

O texto de "Introdução ao GDB debugger", no fim deste guião, contém informação necessária para esta sessão, e é uma sinopse ultra-compacta do manual; uma versão integral está disponível no site da GNU.

A resolução deverá ser entregue **impreterivelmente** no início da sessão PL, <u>com a presença do estudante durante a sessão PL</u> para que o TPC seja contabilizado na avaliação por participação. Não serão aceites trabalhos entregues fora da PL.

Passagem de parâmetros a funções

1. a) (TPC) Crie um ficheiro com o nome while_loop.c com seguinte função e execute a sua compilação para assembly, usando o comando gcc -O2 -S while loop.c.

```
int while_loop(int x, int y, int n)
{
    do {
        x += n;
        y *= n;
        n--;
    } while (n > 0);
    return (x+y);
}
```

b) (TPC) Os argumentos x, y e n da função encontram-se, respetivamente, à distância 8, 12 e 16 do endereço de memória especificado em %ebp. Analise o Assembly da função e indique na tabela os registos atribuídos a cada variável, com base na identificação das instruções que efetuam cópias dos valores da pilha (usando %ebp como endereço base) para os correspondentes registos (inclua também essas as instruções na tabela).

Variável	Registo atribuído pelo compilador	Instrução Assembly
Х		
У		
n		

c) (TPC) Confirme essa atribuição dos registos identificando as três instruções no corpo do ciclo do/while que atualizam os valores de x, y e n;

Depuração do programa com gdb

- d) **Utilize o gdb** para depurar o programa e visualizar os valores durante a execução:
 - i. (TPC) Complete o ficheiro while_loop.c acrescentando uma função main que chame a função while loop, passando como argumentos os valores 4, 2 e 3.
 - ii. (TPC) Crie um executável para ser depurado, com o comando gcc -Wall -O2 -g; desmonte o executável com o comando objdump -d e encontre e anote a localização em while_loop, da 1ª instrução de cópia dos valores da pilha para os correspondentes registos, e a 1ª instrução logo a seguir à cópia desses valores.
 - iii. Invocando o *debugger* (com gdb <nome_fich_executável>), insira dois pontos de paragem (*breakpoint*) nos respetivos endereços anotados anteriormente.
 - iv. (TPC) Estime os valores atribuídos aos registos, preenchendo esta tabela sem executar qualquer código (apenas com base na análise do código assembly).

Variável	Registo	Break1	Break_	Break_	Break_	Break_
Х						
У						
n						

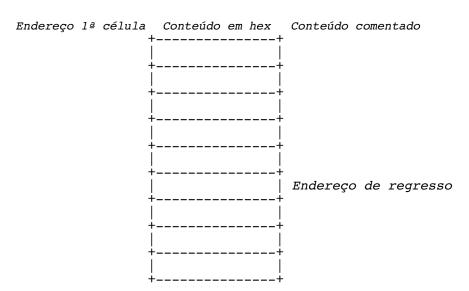
v. Confirme os valores executando o programa dentro do debugger. após cada paragem num breakpoint, visualize o conteúdo dos registos com print \$reg\$ ou com info registers e preencha a tabela em baixo com os valores lidos:

Chamada/regresso de funções e estrutura de ativação (stack frame)

e) Considerando que a stack cresce para cima, pretende-se construir o diagrama da stack frame da função while_loop logo após a execução da instrução antes do breakpoint, com o máx. de indicações (endereços e conteúdos, ver 1ª linha da figura). Comente cada um dos conteúdos da stack frame (por ex., "Endereço de regresso").

Construa assim esse diagrama:

- (i) estime os valores antes da execução do código, e
- (ii) <u>confirme posteriormente</u> esses valores, com o depurador durante a execução do código <u>Nota</u>: neste diagrama, cada caixa representa um bloco de 32-bits em 4 células.



Anexo: Introdução ao GNU debugger

O GNU debugger GDB disponibiliza um conjunto de funcionalidades úteis na análise e avaliação do funcionamento de programas em linguagem máquina, durante a sua execução; permite ainda a execução controlada de um programa, com indicação explícita de quando interromper essa execução – através de breakpoints, ou em execução passo-a-passo – e possibilitando a análise do conteúdo de registos e de posições de memória, após cada interrupção.

Use o GDB para confirmar as tabelas de utilização de registos e o valor dos argumentos nos exercícios. Nota: utilize primeiro objdump para obter uma versão "desmontada" do programa.

A tabela/figura seguinte (de CSAPP) ilustra a utilização de alguns dos comandos mais comuns para o IA-32.

Command	Effect			
Starting and Stopping				
quit	Exit GDB			
run	Run your program (give command line argum. here)			
kill	Stop your program			
Breakpoints				
break sum	Set breakpoint at entry to function sum			
break *0x80483c3	Set breakpoint at address 0x80483c3			
disable3	Disable breakpoint 3			
enable 2	Enable breakpoint 2			
<i>clear</i> sum	Clear any breakpoint at entry to function sum			
delete 1	Delete breakpoint 1			
delete	Delete all breakpoints			
Execution				
stepi	Execute one instruction			
stepi 4	Execute four instructions			
nexti	Like stepi, but proceed through function calls			
continue	Resume execution			
finish	Run until current function returns			
Examining code				
disas	Disassemble current function			
disas sum	Disassemble function sum			
disas 0x80483b7	Disassemble function around address 0x80483b7			
	Disassemble code within specified address range			
print /x \$eip	Print program counter in hex			
Examining data				
print \$eax	Print contents of %eax in decimal			
print /x \$eax	Print contents of %eax in hex			
print /t \$eax	Print contents of %eax in binary			
print 0x100	Print decimal representation of 0x100			
print /x 555	Print hex representation of 555			
print /x (\$ebp+8)	Print contents of %ebp plus 8 in hex			
print *(int *) 0xbffff890	Print integer at address 0xbffff890			
print *(int *) (\$ebp+8)	Print integer at address %ebp + 8			
x/2w 0xbffff890	Examine 2(4-byte) words starting at addr 0xbffff890			
x/20b sum	Examine first 20 bytes of function sum			
Useful information				
info frame	Information about current stack frame			
info registers	Values of all the registers			
help	Get information about GDB			

Figure 3.27: **Example GDB Commands.** These examples illustrate some of the ways GDB supports debugging of machine-level programs.

N° Nome:	Turma:
----------	--------

Resolução dos exercícios (deve ser redigido manualmente)

1. Código em assembly

Transcreva aqui o código assembly de while_loop.s. e classifique cada uma das instruções de acordo com seu propósito:

- (i) copiar valores da pilha para registos
- (ii) atualizar os valores de x, y e n no corpo do ciclo do/while
- (iii) implementação do ciclo do/while
- (iv) chamada/regresso da função e gestão da pilha.
- (v) calcular o valor de retorno da função

Preencha a tabela com os registos atribuídos a cada variável

Variável	Registo
Х	
У	
n	

2. Simulação da execução

Escreva aqui o código C de um programa simples (main) que usa a função while loop:

Coloque aqui a estimativa do que irá estar nos registos após cada ponto de paragem:

Variável	Registo	Break1	Break_	Break_	Break_	Break_
X						
У						
n						