# *Assembly* do IA-32 em ambiente Linux

## TPC8 e Guião laboratorial

*Alberto José Proença*

---

## Objectivo

A lista de exercícios/tarefas propostos no TPC8 / Guião laboratorial, para execução no servidor, reforça a análise laboratorial (e a ferramenta associada, o depurador `gdb`) referente ao conjunto de **instruções e técnicas para suporte à invocação e execução de funções em C**.
Os exercícios para serem resolvidos antes da aula TP estão assinalados com uma caixa cinza.

---

### *Buffer overflow*

**1.** O seguinte código C mostra uma implementação (de baixa qualidade) de uma função que lê uma linha da *standard input*, copia a *string* lida para uma novo local de memória, e devolve um apontador para o resultado.

```
1 /*  Isto e' codigo de qualidade questionavel.
2     Tem como objectivo ilustrar tecnicas deficientes de programação. */
3 char *getline()
4 {
5     char buf[8];
6     char *result;
7     gets(buf);
8     result = malloc(strlen(buf));
9     strcpy(result, buf);
10    return(result);
11 }
```

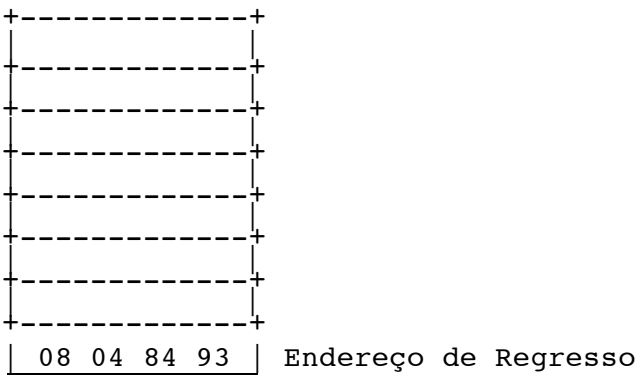
**a)** (A) **Construa** um `main` simples que invoque a função `getline` e compile-o usando a optimização `-O2`; confirme que o programa executável "desmontado" (*disassembled*) até à chamada da função `gets` é semelhante a:

```
1   08048430 <getline>:
2   8048430:    55                push  %ebp
3   8048431:    89 e5             mov   %esp,%ebp
4   8048433:    83 ec 18          sub   $0x18,%esp
5   8048436:    83 ec 0c          sub   $0x0c,%esp
6   8048439:    8d 45 f8          lea   0xfffffff8(%ebp),%eax
7   804843c:    50                push  %eax
8   804843d:    e8 de fe ff ff    call  8048320 <gets@plt>   Invoca gets
```

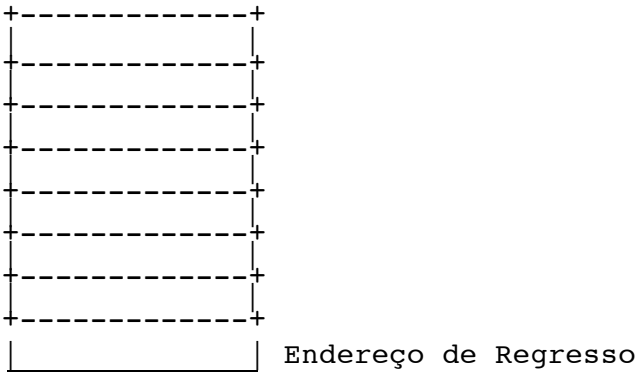
**b)** (A) Considere o seguinte cenário: a função `getline` é invocada com o endereço de regresso `0x8048493`, o registo `%ebp` com o valor `0xbfc7c698`, o registo `%esi` com `0x1`, e o `%ebx` com `0x2` (confirme estes valores ou outros similares); introduz-se a *string* "`012345678901`". Confirme que o programa termina anormalmente.

Pretende-se detectar o local onde ocorreu a anomalia na execução do programa, com o auxílio de um depurador.
Dica: deverá chegar à conclusão que tal aconteceu na execução da instrução `ret` da função `getline`.

**c)** **(A/R)** Considerando que a *stack* "cresce para cima", preencha o diagrama (da *stack frame*) com o máximo de indicações, logo após execução da instrução da linha 5 (no código desmontado em cima). Coloque em cada caixa (que representa 4 *bytes*) o respectivo valor em hexadecimal (se conhecido), à esquerda o endereço mais baixo das 4 células que estão representadas em cada caixa, e à direita uma etiqueta que ajude a esclarecer o conteúdo da *stack* (por ex., "Endereço de Regresso").
Confirme agora a *stack frame* que construiu. Indique a posição de `%ebp`.




**d)** **(R)** Modifique o diagrama para mostrar os valores expectáveis após a invocação da função `gets` (linha 8), e depois confirme esses valores.




**e)** **(R)** Para que endereço acha que o programa está a tentar regressar?
Resp.: _____
Confirme a sua previsão.

**f)** **(R)** Que registo(s) ache que foi(oram) corrompido(s) no regresso da função `getline` e como? Confirme a sua previsão.

**g)** **(B)** Para além do problema de *buffer overflow*, que duas outras coisas estão erradas no código de `getline`?