“One language to rule them all.”

Saída com funções do C em x86-64

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

E/S em C

- Em C temos diversas funções para entrada e saída
  - scanf, printf, write, read, …
- As funções na verdade são *wrappers* para chamadas ao Sistema Operacional, que é o real responsável pela E/S

Write

- A função write é uma das mais simples funções de saída
  - Definida dentro de unistd.h
  - ssize_t write(int fd, const void *buf, size_t count);
    - fd é um número identificador do arquivo de saída
      - Em sistemas **UNIX**, o **arquivo 1** representa **STDOUT**
        - Constante STDOUT_FILENO definida em unistd.h
    - *buf é o endereço de um vetor de caracteres
    - count é o número de caracteres total na string
    - Em caso de sucesso
      - A função retorna o número de caracteres impressos
    - Em caso de erro
      - -1 é retornado
      - O erro é armazenado em *errno*

Write

- Para começar, crie o seguinte programa em C, compile e execute
  - Para compilar

    gcc programa.c -o programa

```c
#include<unistd.h>

int main(){
    char ola[] = "Ola Mundo\n";

    write(STDOUT_FILENO, ola, 10);
    return 0;
}
```

Segmento de dados somente leitura

- Em Assembly x86-64 do GAS, para instruir o compilador a inserir os dados na **seção de dados somente leitura**, utilize a diretiva
  .section .rodata
- A diretiva *.string* armazena uma string e a termina com ‘/0’ (0) na memória
  - Exemplo com uma string rotulada de minha_str
    minha_str: .string "um exemplo"

Segmento de dados somente leitura

- Observação
  - .rodata é mapeado para o segmento de texto do programa pelo montador em sistemas UNIX, pois esse segmento é somente leitura
    - Diferente do segmento de dados, que é leitura/escrita

Argumentos

- Em x86-64, consideramos os argumentos da esquerda para a direita, e os colocamos nos seguintes registradores
  - Caso hajam mais parâmetros, o que fazer?

Argumento	Registrador
Primeiro	rdi
Segundo	rsi
Terceiro	rdx
Quarto	rcx
Quinto	r8
Sexto	r9

Argumentos

- Em x86-64, consideramos os argumentos da esquerda para a direita, e os colocamos nos seguintes registradores
  - Caso hajam mais parâmetros, esses são passados via pilha
    - São empilhados em ordem inversa
    - O primeiro parâmetro a ser empilhado é o mais a direita

Argumento	Registrador
Primeiro	rdi
Segundo	rsi
Terceiro	rdx
Quarto	rcx
Quinto	r8
Sexto	r9

- Observação
  - No modo 32 bits, todos os parâmetros são passados via pilha

Argumentos

- Troque os tamanhos dos argumentos conforme necessário
  - Exemplo: se o argumento possui 32 bits, use **e**di ao invés de **r**di

Argumento	**Registrador**
Primeiro	rdi
Segundo	rsi
Terceiro	rdx
Quarto	rcx
Quinto	r8
Sexto	r9

Olá mundo em x86-64

```
        .section  .rodata             #dados somente leitura
ola_mundo:                            #nome da string
        .string "Ola Mundo\n"         #string terminada com \0
        .text                         #seção de texto
        .globl main                   #main visível globalmente
        .type  main, @function
main:                                 #inicio do main
        pushq  %rbp                   #salva stack frame na pilha
        movq %rsp, %rbp               #define novo stack frame
        movl $10, %edx                #terceiro argumento
        movq $ola_mundo, %rsi         #segundo argumento
        movl $1, %edi                 #primeiro argumento
        call write                    #chama a função write
        movl $0, %eax                 #return 0
        movq %rbp, %rsp               #restaura a pilha
        popq %rbp                     #restaura o stack frame
        ret                           #retorna ao chamador
```

Olá mundo em x86-64

Inserindo os parâmetros. Note que *movl $ola_mundo, %esi* carrega o **endereço** onde ola_mundo inicia na memória para %esi

```
        .section  .rodata          #dados somente leitura
ola_mundo:                         #nome da string
        .string "Ola Mundo\n"      #string terminada com \0
        .text                      #seção de texto
        .globl main                #main visível globalmente
        .type  main, @function
main:                              #inicio do main
        pushq  %rbp                #salva stack frame na pilha
        movq %rsp, %rbp            #define novo stack frame
        movl $10, %edx             #terceiro argumento
        movq $ola_mundo, %rsi      #segundo argumento
        movl $1, %edi              #primeiro argumento
        call write                 #chama a função write
        movl $0, %eax              #return 0
        movq %rbp, %rsp            #restaura a pilha
        popq %rbp                  #restaura o stack frame
        ret                        #retorna ao chamador
```

Olá mundo em x86-64

Chama a função write e **empilha o endereço de retorno**. Write é definido em “unistd.h”. O GCC vai ter que encontrar essa função no PATH do sistema e fazer a linkedição.

```
        .section  .rodata           #dados somente leitura
ola_mundo:                          #nome da string
        .string "Ola Mundo\n"       #string terminada com \0
        .text                       #seção de texto
        .globl main                 #main visível globalmente
        .type  main, @function
main:                               #inicio do main
        pushq  %rbp                 #salva stack frame na pilha
        movq %rsp, %rbp             #define novo stack frame
        movl $10, %edx              #terceiro argumento
        movq $ola_mundo, %rsi       #segundo argumento
        movl $1, %edi               #primeiro argumento
        call write                  #chama a função write
        movl $0, %eax               #return 0
        movq %rbp, %rsp             #restaura a pilha
        popq %rbp                   #restaura o stack frame
        ret                         #retorna ao chamador
```

Montando o programa

- Monte o programa
  as olaMundo.s --gstabs -o olaMundo.o
- Faça a linkedição com a ajuda do GCC
  gcc olaMundo.o -no-pie -o olaMundo
  - -no-pie desabilita a geração de executáveis de posição independente
    - O pie permite o S.O. carregar as dependências em posições aleatórias da memória a cada execução do programa
      - **Recurso de segurança** do S.O.
    - Não vamos conseguir usar devido a forma que estamos gerando nosso programa

Exercício

1. Monte e execute o programa do exemplo anterior

2. Execute passo a passo no GDB, e analise o que está acontecendo nos registradores e memória
   - Dicas do GDB
     - Para imprimir o conteúdo você pode usar a sintaxe do C. Exemplos:
       printf "%s", &ola_mundo (endereço do label)
       printf "%s", 0x400594 (passando um endereço de memória diretamente)
     - *si* executa a próxima instrução
       - Caso seja uma chamada a função, entra na função
     - *ni* faz o mesmo que *si*, **mas não entra na função**

Tornando mais legível

- A diretiva .equ pode ser utilizada para definir nomes a expressões
  .equ NOME, EXPRESSÃO
  - Na etapa de montagem, o montador substitui as ocorrências de NOME pelo valor de EXPRESSÃO
  - **Similar a um #define em C**

Tornando mais legível

Um ponto (.) significa “esse endereço”. Como o . está após a string, carregamos o end. final da string. Como o label ola_mundo marca o endereço de início da string, a conta sendo feita é endereçoFinal-endereçoInicial-1, o que dá o tamanho da string. O -1 é necessário pois a string é terminada com ‘\0’ por padrão com a diretiva .string.

```
        #constantes
        .equ STDOUT,1
        .section  .rodata             #dados somente leitura
ola_mundo:                            #nome da string
        .string "Ola Mundo\n"         #string terminada com \0
        .equ ola_mundoSz, .-ola_mundo-1
        .text                         #seção de texto
        .globl main                   #main visível globalmente
        .type  main, @function
main:                                 #inicio do main
        pushq  %rbp                   #salva stack frame na pilha
        movq %rsp, %rbp               #define novo stack frame
        movl $ola_mundoSz, %edx       #terceiro argumento
        movl $ola_mundo, %esi         #segundo argumento
        movl $STDOUT, %edi            #primeiro argumento
        call write                    #chama a função write
        movl $0, %eax                 #return 0
        movq %rbp, %rsp               #restaura a pilha
        popq %rbp                     #restaura o stack frame
        ret                           #retorna ao chamador
```

Exercício

3. Modifique o exemplo feito em aula para que a string seja impressa utilizando *printf*. **Atenção**: antes de chamar o *printf*, **carregue 0 para rax**. Veremos o motivo para isso na próxima aula.

Referências

- Bob Plantz. **Introduction to Computer Organization: A Guide to X86-64 Assembly Language and GNU/Linux.** 2011.
- **Intel® 64 and IA-32 Architectures Software Developer's Manual.** Intel, 2019.
- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**: **a Interface Hardware/Software.** 5a Edição. Elsevier Brasil, 2017.
- STALLINGS, W. **Arquitetura e Organização de Computadores.** 10 ed. Prentice Hall. São Paulo, 2018.
- M. Matz, J. Hubička, A. Jaeger, M. Mitchell. **System V Application Binary Interface AMD64 Architecture Processor Supplement.** 2014.