Linguagens de Montagem

Chamadas de Sistema
Aula 06

Edmar André Bellorini

Ano letivo 2021

Hello World

```
6   section .data
7       strOla :  db "Ola", 10
8       strOlaL:  equ $ - strOla
9
10  section .text
11      global _start
12
13  _start:
14      mov rax, 1
15      mov rdi, 1
16      lea rsi, [strOla]
17      mov edx, strOlaL
18      syscall
19
20  fim:
21      mov rax, 60
22      mov rdi, 0
23      syscall
```

Hello World

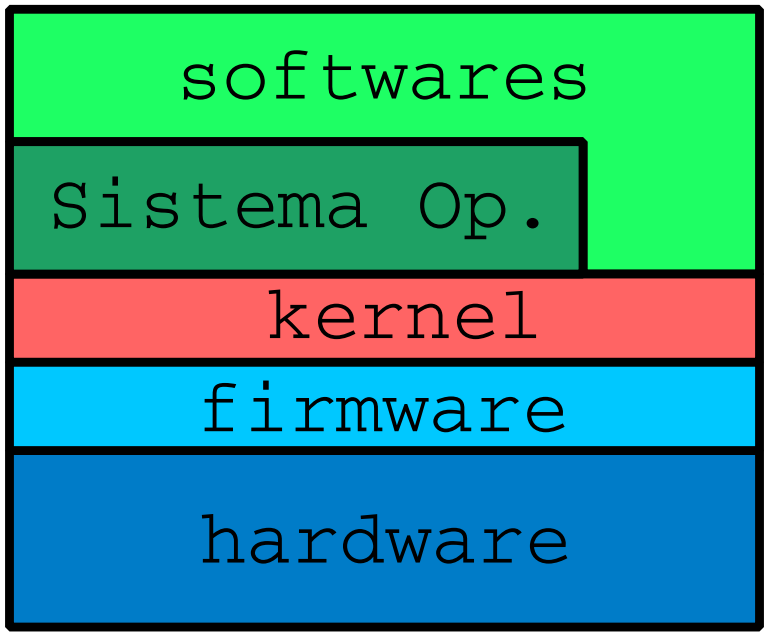
```
6   section .data
7       strOla :  db "Ola", 10
8       strOlaL:  equ $ - strOla
9
10  section .text
11      global _start
12
13  _start:
14      mov rax, 1
15      mov rdi, 1
16      lea rsi, [strOla]
17      mov edx, strOlaL
18      syscall ; instrucao de Chamada de Sistema
19
20  fim:
21      mov rax, 60
22      mov rdi, 0
23      syscall ; instrucao de Chamada de Sistema
```

Interrupções

`syscall`

- É a chamada de sistema padrão dos S.Os x86/Unix x64
- Usada para chamar serviços do `Kernel`
- `Kernel`
  - Componente central de um S.O.
  - Gerencia os recursos de *hardware* disponíveis para os *softwares*
    - Processador
    - Entrada e Saída (monitor/teclado)
    - Memória

Arquitetura (Alta Abstração)



Kernel

- Chamadas de Sistema
  - A partir de um estado da CPU, executa uma determinada operação

```
14      mov rax, 1
15      mov rdi, 1
16      lea rsi, [strOla]
17      mov edx, strOlaL
18      syscall
```

  - Executa a chamada de sistema de impressão (1)

    `mov rax, 1`

    - para a saída padrão (1)

      `mov rdi, 1`

    - do texto que se encontra na posição de memória *strOla*

      `lea rsi, [strOla]`

    - e tem *strOlaL* caracteres

      `mov edx, strOlaL`

Chamadas de Sistemas já conhecidas

- WRITE

```
mov rax, 1          ; syscall WRITE
mov rdi, 1          ; file descriptor
lea rsi, [strOla]   ; *buffer
mov edx, strOlaL    ; count
```

  - SysCall no Kernel

```
ssize_t write(int fd , const void *buf, size_t count);
eax     write(int rdi, const void *rsi, size_t edx  );
```

  - edx

```
strOlaL:  equ $ - strOla
```

    - equ é uma pseudo-instrução de equivalência
    - $ é "aqui"
    - $ - strOla é "aqui" - strOla = tamanho em bytes do texto
  - retorno em rax (eax)
    - Número de caracteres efetivamente escritos

Chamadas de Sistemas já conhecidas

- EXIT

```
mov rax, 60         ; syscall EXIT
mov rdi, 0          ; status
syscall
```

  - SysCall no Kernel

```
void _exit(int status);
void _exit(int rdi   );
```

  - Retorno para o S.O.

```
$: echo $?
```

    - Após a execução de um programa

Uma nova chamada de sistema

- READ
  - SysCall no Kernel

```
ssize_t read(int fd , const void *buf, size_t count);
eax     read(int rdi, const void *rsi, size_t edx  );
```

```
mov rax, 0          ; syscall READ
mov rdi, 1          ; file descriptor
lea rsi, [strLida]  ; *buffer
mov edx, strLidaL   ; count
```

  - edx
    - É o número máximo de caracteres lidos
    - "Enter" finaliza entrada e pode ser contato!
  - retorno em rax (eax)
    - Número de caracteres efetivamente lidos

Exemplo a06e01.asm

- Código a06e01.asm em anexo
  - O exemplo a06e01.asm é um código *repeater*
    - Apenas mostra na saída padrão o que foi digitado

```
7   %define maxChars 10
8
9   section .data
10     strOla : db "Hello?", 10, 0
11     strOlaL: equ $ - strOla      ; cuidado: strOlaL "non-existe!" (equ)
12
13     strBye : db "Voce digitou: ", 0
14     strByeL: equ $ - strBye
15
16     strLF  : db 10 ; quebra de linha ASCII!
17     strLFL : db 1
18  ...
```

Debugger de a06e01.asm

- Breakpoints
  - leitura, resposta e fim
    - Qual é o valor de RAX ao alcançar esses Breakpoints?
- Pré-processador

```
%define maxChars 10
```

  - É um avaliador léxico que pode gerar código ou substituir valores pré-definidos.
  - No exemplo, todo texto "maxChars" a partir da linha de definição será substituído pelo valor 10

Referências das SysCalls

- Site syscalls64 ▸ syscalls64 ou ▸ webarchive.org(syscalls64)
  - Contém uma tabela com as chamadas de sistema linux/kernel
    - e link de cada Chamada (*name*) para sua `man-page`
  - Mantido por Paolo Stivanin no GitHub

- Site Linux System Call Table for x86_64 ▸ rchapman.org
  - Mantido por Ryan A. Chapman

Exemplo a06e02.asm

- OPEN

```
int open(const char *pathname, int flags, mode_t mode);
eax open(const char *rdi     , int esi  , mode_t edx );
```

  - Abre um arquivo
    - rax: *file descriptor* ou -1 (falha)
    - rdi: caminho do arquivo
    - esi: *flags**
    - edx: modo de criação*

      *ver últimos slides (Anexo - Flags Open())

- CLOSE

```
int close(int fd);
eax close(int edi);
```

  - Fecha arquivo aberto com Open()
    - rax: 0 em caso de sucesso ou -1
    - edi: *file descriptor*

Exemplo a06e02.asm

```
18   ...
19  section .data
20     fileName: db "a06e02.txt", 0 ; null-terminated string!
21
22  section .bss
23     texto: resb 25
24     fileHandle: resd 1
25
26  section .text
27     global _start
28
29  _start:
30     ; int open(const char *pathname, int flags, mode_t mode);
31     mov rax, 2           ; open file
32     lea rdi, [fileName] ; *pathname
33     mov esi, openrw      ; flags
34     mov edx, userWR      ; mode
35     syscall
36     ...
```

Atividades - slide 1

- a06at01 - Aplicação *sem sentido*: Criar uma aplicação que leia o próprio PID e execute a chamada *kill*
  - PID (Process ID): todo processo em execução recebe um número identificador único
    - é possível executar operações sobre um processo com este PID
    - chamada de sistema **sys_getpid**
  - kill: comando linux que encerra a execução de um processo
    - é possível passar um no. que identifica o motivo do encerramento
    - chamada de sistema **sys_kill**
      Requer **inteiro** como parâmetro, para saber mais use `kill -l` no terminal ou acesse ▸ Kill Commands and Signals
      Recomendado para este caso é o **9**
  - Para saber se o código funcionou, insira um trecho de código após o comando kill que imprima no terminal a frase: "this isn't working!"
    - Ou use o arquivo "ef0601.asm" como esqueleto do seu código

Atividades - slide 2

- a06at02 - Papagali Persistente
  - Use como base o arquivo a06e01.asm
  - A cada execução, o texto entrado pelo usuário deve ser escrito ao final de um arquivo
  - Caso o arquivo não exista, deve ser criado com o nome "papagali.txt"
  - Caso exista, o texto deve ser adicionado ao final (append)
    - Verifique o slide **Anexo - Flags Open()**

Fim do Documento

Dúvidas?

Próxima aula:

- Aula 07: Controle de Fluxo de Execução
  - Fluxo de Execução
  - Labels
  - Laços de Repetições
    finalmente!
  - Desafio!

Anexo - Flags Open()

- O_RDONLY (0), O_WRONLY (1), O_RDWR (2)
- O_CREAT (100)
  - Cria arquivo se o mesmo não existir
  - Para este caso, é necessário definir as permissões do arquivo em RDX (syscall)
  - Valor padrão para -rx-r–r é 0644o
    - ▸ Linux - Permissões de Arquivos
  - Usa-se O_CREAT junto com O_RDONLY (100), O_WRONLY (101), O_RDWR (102)
- O_APPEND (2000)
  - Adiciona conteúdo em arquivo existente
  - Usado em conjunto com O_CREAT + (O_WRONLY ou O_RDWR)
  - Não esquecer das permissões de arquivo (EDX)
- TODOS os valores estão em OCTAL