Linguagem de Montagem

Chamadas de Sistema
Aula 06

Edmar André Bellorini

Ano letivo 2016

Hello World

```
section .data
   strOla :  db "Ola", 10
   strOlaL:  equ $ - strOla

section .text
global _start

_start:
   mov eax, 4
   mov ebx, 1
   mov ecx, strOla
   mov edx, strOlaL
   int 0x80

   mov eax, 1
   mov ebx, 0
   int 0x80
```

Hello World

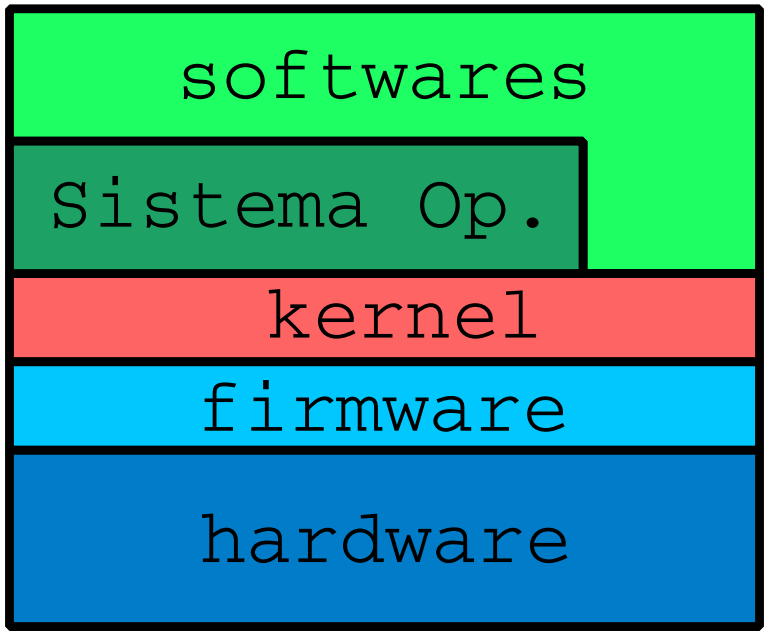
```
1   section .data
2       strOla :  db "Ola", 10
3       strOlaL:  equ $ - strOla
4
5   section .text
6   global _start
7
8   _start:
9       mov eax, 4
10      mov ebx, 1
11      mov ecx, strOla
12      mov edx, strOlaL
13      int 0x80  ; Instrucao de Interrupcao
14
15      mov eax, 1
16      mov ebx, 0
17      int 0x80  ; Instrucao de Interrupcao
```

System Call

```
int 0x80
```

- É a chamada de interrupção padrão dos sistemas x86/Unix
- 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

```
9       mov eax, 4
10      mov ebx, 1
11      mov ecx, strOla
12      mov edx, strOlaL
13      int 0x80  ; Instrucao de Interrupcao
```

  - Executa a chamada de sistema de impressão (4)

    `mov eax, 4`

    - para a saída padrão (1)

      `mov ebx, 1`

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

      `mov ecx, strOla`

    - e tem *strOlaL* caracteres

      `mov edx, strOlaL`

Chamadas de Sistemas já conhecidas

- WRITE

```
mov eax, 4          ; sysCall WRITE
mov ebx, 1          ; file descriptor
mov ecx, strOla     ; *buffer
mov edx, strOlaL    ; count
```

  - SysCall no Kernel

```
ssize_t write(int fd , const void *buf, size_t count);
eax     write(int ebx, const void *ecx, 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
  - eax
    - Número de caracteres efetivamente lidos

Chamadas de Sistemas já conhecidas

- EXIT

```
mov eax, 1          ; sysCall 1
mov ebx, 0          ; status
int 0x80
```

  - SysCall no Kernel

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

  - 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 ebx, const void *ecx, size_t edx  );
```

```
mov eax, 3          ; sysCall Read
mov ebx, 1          ; file descriptor
mov ecx, strLida    ; *buffer
mov edx, strLidaL   ; count
```

  - edx
    - É o número máximo de caracteres lidos
  - 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

```
1   %define maxChars 10
2
3   section .data
4       strOla : db "Hello?", 10
5       strOlaL: equ $ - strOla
6
7       strBye : db "Voce digitou: "
8       strByeL: equ $ - strBye
9
10      strLF  : db 10
11      strLFL : db 1
12  ...
```

Debugger de a06e01.asm

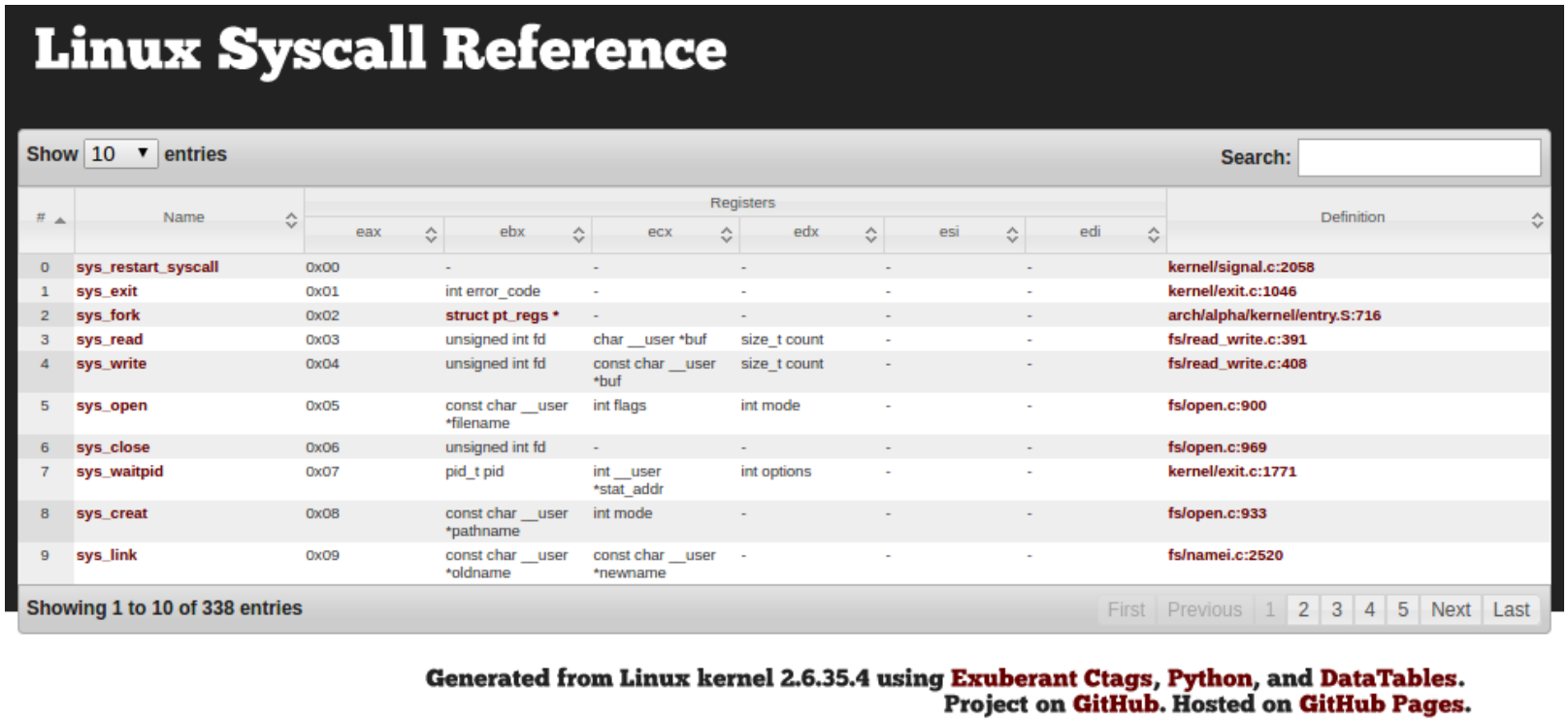
- Breakpoints
  - leitura, resposta e fim
    - Qual é o valor de RAX ao alcançar esses Breakpoints?
- Pré-processador

```
%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 SysCall

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



Linux Syscall Reference

Show 10 entries — Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	-	-	-	fs/namei.c:2520

Showing 1 to 10 of 338 entries — First Previous 1 2 3 4 5 Next Last

Generated from Linux kernel 2.6.35.4 using Exuberant Ctags, Python, and DataTables.
Project on GitHub. Hosted on GitHub Pages.

Alternativa ao site kernelgrok.com

- Página das SysCalls no ▸ man7.org
- Para número da chamada (*rax*) use o comando:

  `cat /usr/include/asm/unistd_32.h | grep nomeChamada`

  por exemplo, para a chamada de sistema write():

  `cat /usr/include/asm/unistd_32.h | grep write`

  Será mostrado todas as chamadas que contenham *write* em seu nome, e logo na sequência, seu número.

```
LM_A06 $: cat /usr/include/asm/unistd_32.h | grep write
#define __NR_write 4
#define __NR_writev 146
#define __NR_pwrite64 181
#define __NR_pwritev 334
#define __NR_process_vm_writev 348
LM_A06 $:
```

Exemplo a06e02.asm

- OPEN

```
int open(const char *pathname, int flags, mode_t mode);
rax open(const char *rbx     , int rcx  , mode_t rdx );
```

  - Abre um arquivo
    - rax: *file descriptor* ou -1 (falha)
    - rbx: caminho do arquivo
    - rcx: *flags**
    - rdx: modo de criação*

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

- CLOSE

```
int close(int fd);
```

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

Exemplo a06e02.asm

```
 ...
section .data
   fileName: db "a06e02.txt"

section .bss
   texto: resb 25
   fileHandle: resd 1

section .text
   global _start

_start:
   mov rax, 5    ; open file
   mov rbx, fileName
   mov rcx, openrw
   mov rdx, userWR
   int 0x80
   ...
```

Exercício de Fixação

- EF0601 - Aplicação suicída: Criar uma aplicação que leia seu 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
  - 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

Relatório

- Somente relatório
  - O modelo de relatório para a disciplina de LM está disponível em anexo da Aula 01
    - Arquivo **modeloRelatorioLM.odt**
    - A data do relatório é a data de entrega (ver moodle)
  - Somente serão aceitos os relatórios em formato .pdf com nome do arquivo seguindo o padrão: **TY.PXX.nome.sobrenome.pdf**
    - TY é o número da turma prática (1, 2, 3 ou 4)
    - PXX é o número da prática, neste caso: P06
    - Anexo: *Flags Open()* - próximo slide

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
  - Valor padrão para -rx-r–r é 644o
    - 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 (RDX)