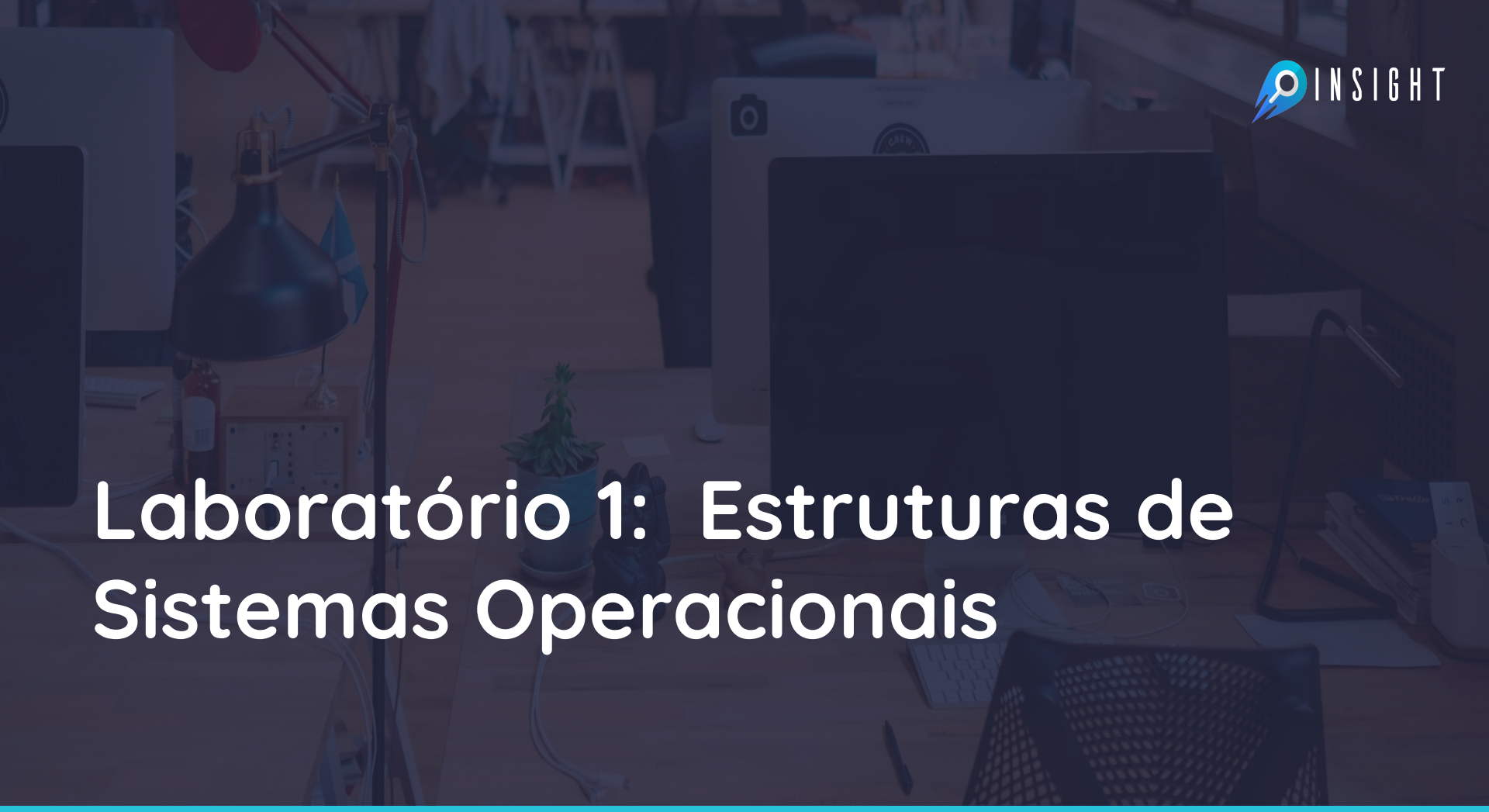


Laboratório 1: Estruturas de Sistemas Operacionais

INSIGHT

DEPARTAMENTO DE COMPUTAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

Objetivo

- Conhecer o sistema operacional Ubuntu
- Entender o que está por trás dos executáveis no sistema operacional Linux

UBUNTU

Linux - Um Pouco de História

- Décadas de 70 e 80: SO Unix da AT&T
  - Primeira versão em 1969 pelos Lab. Bell da AT&T
  - Rodava em computadores de grande porte e estações de trabalho RISC
  - Era utilizado também em ambientes acadêmicos e de pesquisa
  - Tanenbaum desenvolve o MINIX, sistema simples para estudo, teoricamente baseado no Unix mas que não utiliza qualquer linha de código da AT&T
- Em 1991, Linus Torvalds, um estudante de computação finlandês, faz um clone do Minix, projetado para ser um sistema de produção para PC. O sistema chama-se Linux.

Linux - História

- Em 1992, Linus envia, pela Internet, a outros programadores no planeta, o código-fonte (“receita”) do seu Kernel, buscando ajuda para amadurecer aquele embrião.
- Esso é o início da grande “Comunidade Linux”, um grande conjunto de programadores no mundo que mantém e melhora o Linux diariamente.
- Hoje, tornou-se o maior e mais famoso projeto de software livre do mundo
  - É o SO como mais versões para quase todos os tipos de máquinas
  - O seu kernel possui atualmente cerca de 7 milhões de linhas de código.

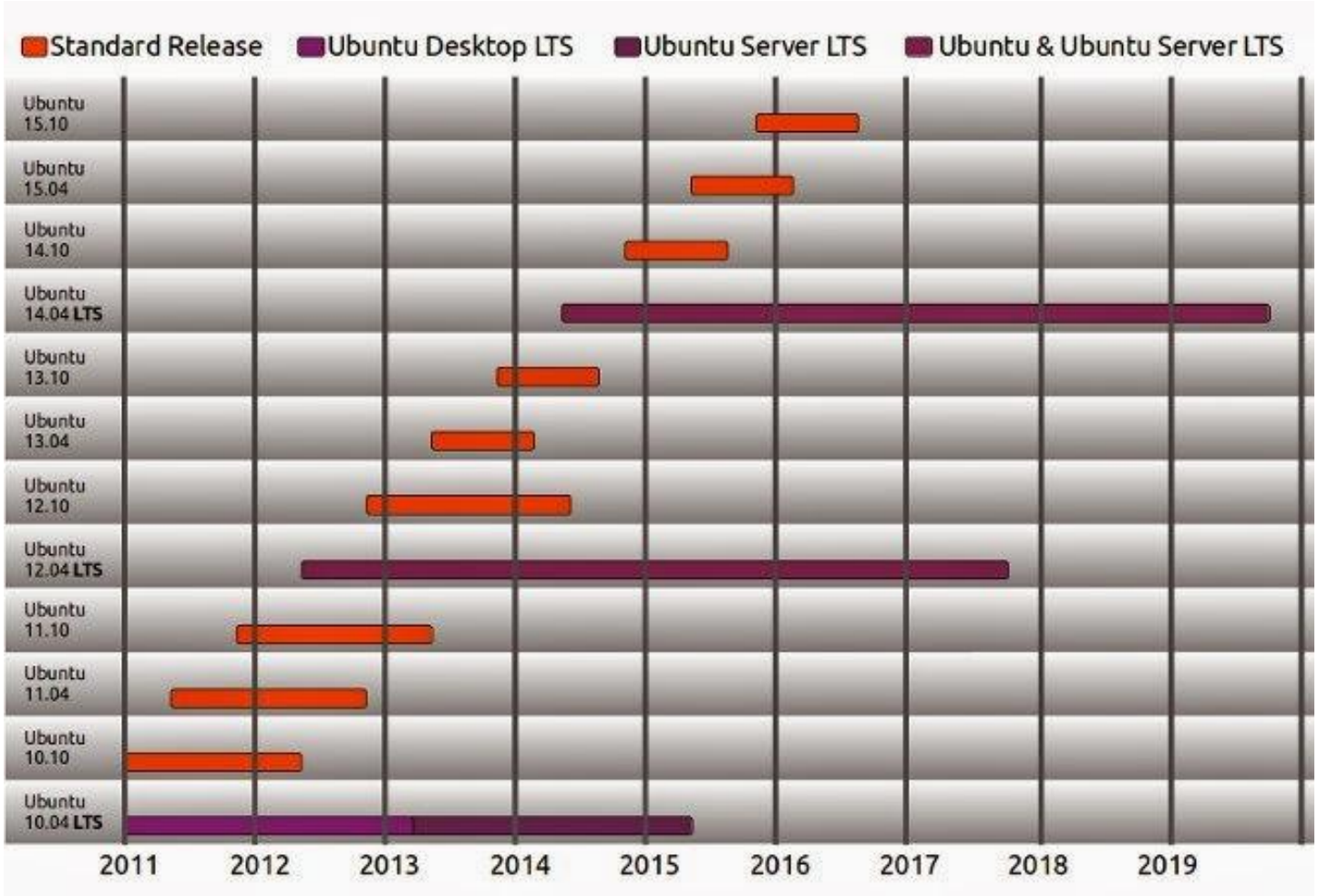
Distribuições

Ubuntu

- Distribuição sul-africana, criada em 2004 pela Canonical Ltd.
  - Baseada na distribuição Debian.
  - Inclui a interface Gnome como padrão; KDE e XFCE são utilizadas nas distribuições derivadas Kubuntu e Xubuntu.
  - A palavra “ubuntu” significa “humanidade para com os outros”, bem como, "eu sou o que sou por causa de quem nós todos somos”

Versões do Ubuntu

- Distribuições do Ubuntu são lançadas a cada abril e outubro
- Ubuntu 4.04 (1º versão do Ubuntu – Abril de 2004)
- Versões LTS (Lançadas de 2 em 2 anos, maior tempo de suporte)



Diretórios

Diretório do root

A primeira hierarquia do sistema de arquivos ou somente:

/

Hierarquia primária

/bin/	Binários principais dos usuários
/boot/	Arquivos do sistema de Boot
/dev/	Arquivos de dispositivos
/etc/	Arquivos de configuração do sistema
/home/	Diretório dos usuários comuns do sistema
/lib/	Bibliotecas essenciais do sistema e os módulos do kernel
/media/	Diretório de montagem de dispositivos
/mnt/	Diretório de montagem de dispositivos - *Mesmo que "media"*
/opt/	Instalação de programas não oficiais da distribuição ou por conta do usuário
/sbin/	Armazena arquivos executáveis que representam comandos administrativos. Exemplo: shutdown
/srv/	Diretório para dados de serviços fornecidos pelo sistema
/tmp/	Diretório para arquivos temporários
/usr/	Segunda hierarquia do sistema, onde ficam os usuários comuns do sistema e programas
/var/	Diretório com arquivos variáveis gerados pelos programas do sistema. Exemplo: logs, spool de impressoras, e-mail e cache
/root/	Diretório do usuário root – usuário root tem total poderes sobre o sistema, podendo instalar, desinstalar e configurá-lo.
/proc/	Diretório virtual controlado pelo Kernel com configuração total do sistema.

ANALISANDO ESTADO DOS PROCESSOS

Exemplo: Acessando o diretório /proc

- `$ cd /proc`
- O /proc é um diretório virtual, mantido pelo kernel onde encontramos a configuração atual do sistema:
  - modelo da cpu,
  - quantidade de memória,
  - dispositivos já montados,
  - interrupções, etc.
- Cada subdiretório tem com o nome que corresponde ao PID (Process ID) de cada processo;
  - Contém diversos arquivos texto
    - Representam uma importante função do programa em execução

Diretório /proc

- `$ cat version` mostra a versão do sistema
- `$ cat cpuinfo` mostra informações do processador
- `$ cat cmdline` mostra os parâmetros passados para o kernel.
- `$ cat pci` lista dispositivos pci do sistema
- `$ cat meminfo` exibe informações sobre a memória no sistema
- `$ cat partitions` exibe as partições do sistema
- `$ cat devices` exibe dispositivos de blocos e caracteres correntemente configurados
- `$ cat filesystems` lista de sistemas de arquivos suportados
- `$ cat interrupts` lista registros de interrupções
- `$ cat iomem` lista o mapa do sistema para cada dispositivo físico.
- `$ cat ioports` lista o endereçamento das portas I/O
- `$ cat misc` lista os drivers registrados em miscellaneous

Diretório /proc/*/status

- Podemos ver os dados do arquivo "status" do subdiretório /proc/12939/ sobre a execução do processo, como:

```
$ cat /proc/12939/status
Name: gnome-terminal
State: S (sleeping)
Tgid: 12939
Pid: 12939
PPid: 1
TracerPid: 0
Uid: 1000 1000 1000 1000
Gid: 1000 1000 1000 1000
```

- **Name** = nome do processo
  State = estado do processo
  PID = identificação do processo
  PPID = ID do processo-pai
  UID = identificação do usuário
  GID = identificação do grupo do usuário

ANATOMIA DE UM PROGRAMA

Criando um simples programa

- Criar um diretório hello –> mkdir hello
- Entrar no diretório e criar um arquivo hello.c
- Conteúdo do hello.c

```
#include <stdio.h>

int main(void)
{
   printf("Hello, World!\n");

   return 0;
}
```

Criando um simples programa (Cont.)

- Compilar com GCC
  - `$ gcc -o hello hello.c`
  - Gera o arquivo `hello`

- Que tipo de arquivo é esse?
  - Comando file – Indica o tipo de arquivo conforme padrão do SO
    - `$ file <arquivo>`
      - Ex: ASCII text, C Program source, directory, ELF-Executable, data, Bourn-again shell-script, JPEG Image File, entre outros.

O que está dentro do executável?

- Para vermos o que há dentro do executável, vamos usar uma ferramenta chamada "objdump"
  - Pode ser usado como um disassembler para ver o arquivo binário na forma de assembly.
  - `$ objdump -f <arquivo binário>`

```
$ objdump -f hello

hello:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400410
```

- A saída nos dá algumas informações importantes sobre o executável.
  - O executável está no formato "ELF64".
  - O endereço inicial é "`0x0000000000400410`"

Formato ELF

- O que é o formato ELF?
  - ELF é o acrônimo de **Executable Linkable Format** (Formato de Execução “Linkeditável”).
  - É um dos vários formatos de arquivos objetos e executáveis usados em sistemas Unix.
  - Cada executável ELF tem um cabeçalho ELF
    - Para saber as informações do cabeçalho ELF do executável, basta usar o comando readelf

Cabeçalho ELF

```
typedef struct
{
unsigned char e_ident[EI_NIDENT]; /* Número mágico e outras informações */
Elf64_Half e_type; /* Objeto de tipo de arquivo */
Elf64_Half e_machine; /* Arquitetura */
Elf64_Word e_version; /* Segmento do objeto de tipo de arquivo */
Elf64_Addr e_entry; /* Ponto de entrada do endereço virtual */
Elf64_Off e_phoff; /* Tabela dos deslocamentos de cabeçalhos dos arquivos
de programas */
Elf64_Off e_shoff; /* Offset do início da seção da tabela de arquivos */
Elf64_Word e_flags; /* Flags específicos do processador */
Elf64_Half e_ehsize; /* Tamanho do cabeçalho ELF em bytes */
Elf64_Half e_phentsize; /* Tamanho da tabela de entrada do cabeçalho do
programa */
Elf64_Half e_phnum; /* Contador da tabela de entradas do cabeçalho do
programa */
Elf64_Half e_shentsize; /* Tamanho da tabela de entrada do cabeçalho da
seção */
Elf64_Half e_shnum; /* Contador da tabela de entradas do cabeçalho da
seção */
Elf64_Half e_shstrndx; /* Índice da tabela de strings do cabeçalho da
seção */
} Elf64_Ehdr;
```

Comando readelf

```
$ readelf -h hello
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400410
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4424 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         30
  Section header string table index: 27
```

O que está dentro do executável? (Cont.)

- Para vermos o que há dentro do executável, vamos usar uma ferramenta chamada "objdump"
  - Pode ser usado como um disassembler para ver o arquivo binário na forma de assembly.
  - `$ objdump -f <arquivo binário>`

```
$ objdump -f hello

hello:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400410
```

  - A saída nos dá algumas informações importantes sobre o executável.
    - O executável está no formato "ELF64".
    - O endereço inicial é "`0x0000000000400410`"

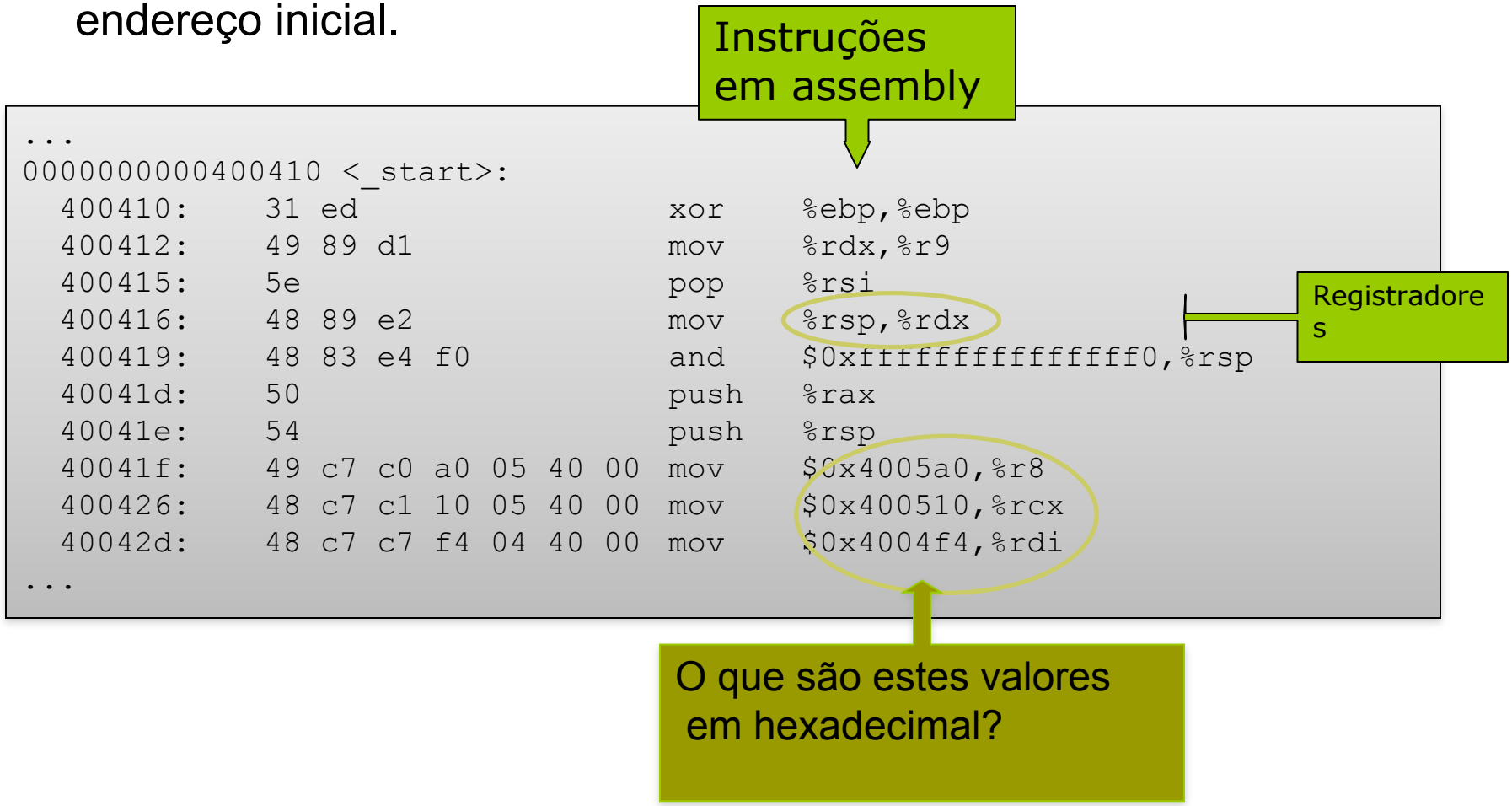
O que está no endereço "0x0000000000400410"?

- Para responder esta pergunta, vamos desmontar o executável "hello". Existem diversas ferramentas para desmontar um executável.
  - Usaremos o objdump
    - `$ objdump -d hello`
      - O resultado é um pouco grande. A intenção é ver o que está no endereço `0x0000000000400410`.

```
...
0000000000400410 <_start>:
  400410:    31 ed                   xor    %ebp,%ebp
  400412:    49 89 d1                mov    %rdx,%r9
  400415:    5e                      pop    %rsi
  400416:    48 89 e2                mov    %rsp,%rdx
  400419:    48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
  40041d:    50                      push   %rax
  40041e:    54                      push   %rsp
  40041f:    49 c7 c0 a0 05 40 00    mov    $0x4005a0,%r8
  400426:    48 c7 c1 10 05 40 00    mov    $0x400510,%rcx
  40042d:    48 c7 c7 f4 04 40 00    mov    $0x4004f4,%rdi
...
```

O que está no endereço "0x0000000000400410"?

- Aparentemente o início de uma rotina chamada "_start" está no endereço inicial.



O que são estes valores em hexadecimal?

- Vamos observar a saída do objdump e procurar a resposta!

- Resposta:
  - `0x4005a0` : Este é o endereço de nossa função main().
  - `0x400510` : função _init.
  - `0x4004f4` : função _fini, as funções _init e _fini são funções de inicialização/finalização providas pelo GCC.

Os hexadecimais são todos ponteiros para funções

Função main

```
$ objdump -d hello
...
0000000000 4004f4 <main>:
  4004f4:    55                      push   %rbp
  4004f5:    48 89 e5                mov    %rsp,%rbp
  4004f8:    bf fc 05 40 00          mov    $0x4005fc,%edi
  4004fd:    e8 ee fe ff ff          callq  4003f0 <puts@plt>
  400502:    b8 00 00 00 00          mov    $0x0,%eax
  400507:    5d                      pop    %rbp
  400508:    c3                      retq
  400509:    90                      nop
...
```

Digite: `$ man puts` e veja a definição de puts

```
#include <studio.h>

int main(void)
{
   printf("Hello, World!\n");

   return 0;
}
```

Mais detalhes do programa

- Para gerar o dump completo use o seguinte comando:
  - objdump -Dslx hello

```
hello:     file format elf64-x86-64
hello
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400440

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000400040 paddr
0x0000000000400040 align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off    0x0000000000000238 vaddr 0x0000000000400238 paddr
0x0000000000400238 align 2**0
         ...

Dynamic Section:
  NEEDED               libc.so.6
  INIT                 0x00000000004003e0
....
```

Seção de Dados

- Na seção de dados está armazenado o dado que será impresso pelo função *printf* :

```
hello:     file format elf64-x86-64
hello
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400440

Disassembly of section .rodata:

00000000004005d0 <_IO_stdin_used>:
  4005d0:       01 00                   add    %eax,(%rax)
  4005d2:       02 00                   add    (%rax),%al
  4005d4:       41                      rex.B
  4005d5:       42                      rex.X
  4005d6:       41                      rex.B
  4005d7:       42                      rex.X
  4005d8:       41                      rex.B
  4005d9:       42                      rex.X
  4005da:       41                      rex.B
        ...
```

Mais informações sobre ELF e linkagem dinâmica

- Com ELF, nós podemos montar um executável linkado dinamicamente com bibliotecas.
- Onde "linkado dinamicamente" significa que o processo de “linkagem” acontece em tempo de execução.
- Ao contrário, teríamos que montar um enorme executável contendo todas as bibliotecas chamadas por ele (chamamos isso de um "executável linkado estaticamente").

Quais as dependências para aplicação funcionar?

- Comando ldd – Lista as dependências (bibliotecas) necessárias
  - `$ ldd <arquivo>`
  - O campo 1 diz o nome da dependência, o campo 2 é o divisor e o campo 3 é a localização da dependência.

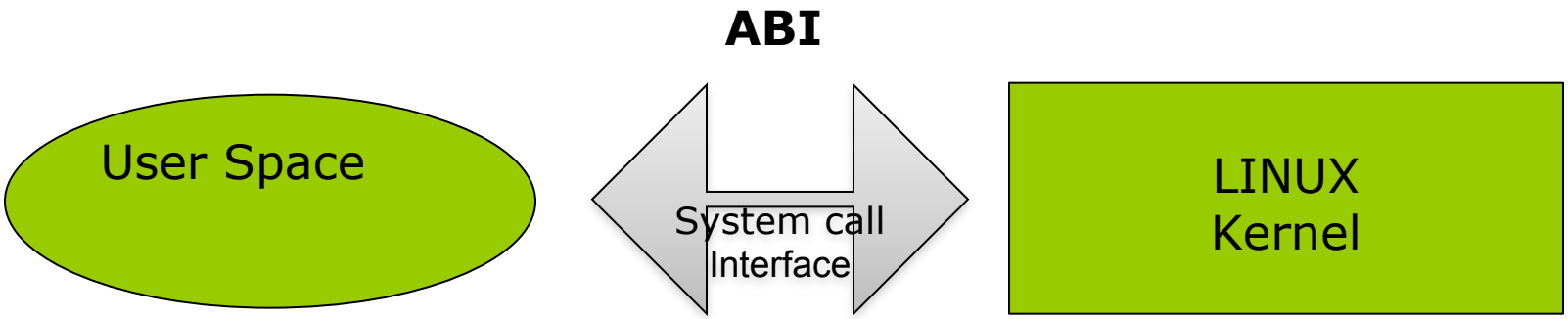
```
$ ldd hello

linux-vdso.so.1 =>  (0x00007fff8a5ff000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f3be8a59000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3be8e2f000)
```

- **linux-vdso.so.1** é o Objeto Compartilhado Dinâmico Virtual do Linux
  - Esta biblioteca fornece a lógica necessária para permitir que os programas do usuário acessem funções do sistema de forma rápida.
- **libc.so.6** contém um ponteiro para /`x86_64-linux-gnu/libc.so.6`
- **./lib64/ld-linux-x86-64.so.2** é o caminho absoluto para outra biblioteca.

ENTENDENDO CHAMADAS AO SISTEMA

A Interface System Call

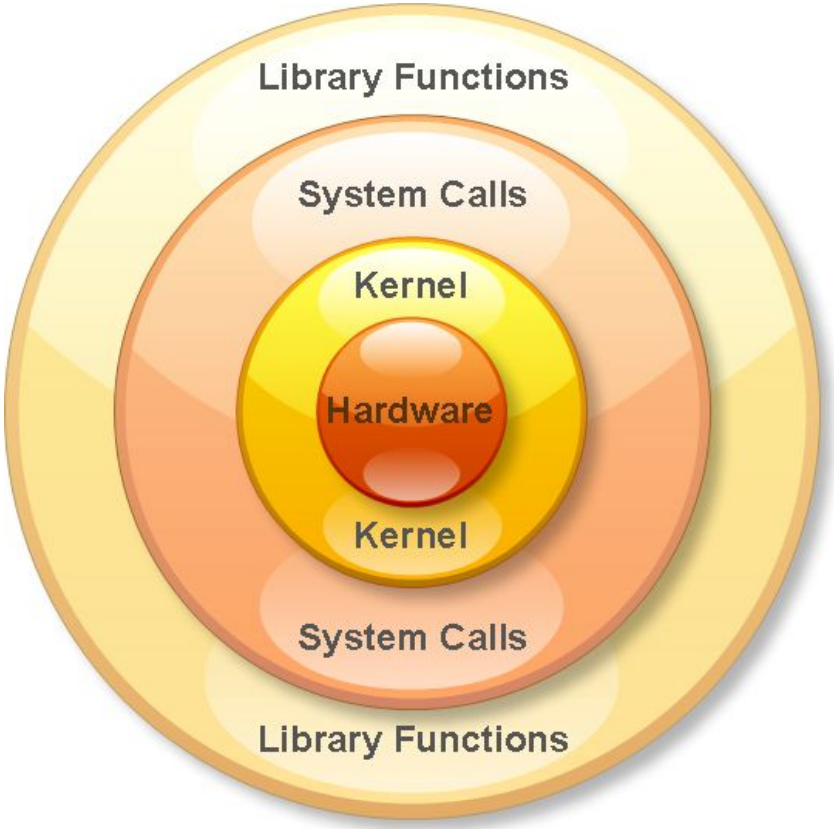


Uma chamada de sistema (***system call***) é o mecanismo usado pelo programa para requisitar um serviço do sistema operacional, ou mais especificamente, do núcleo do sistema operacional.

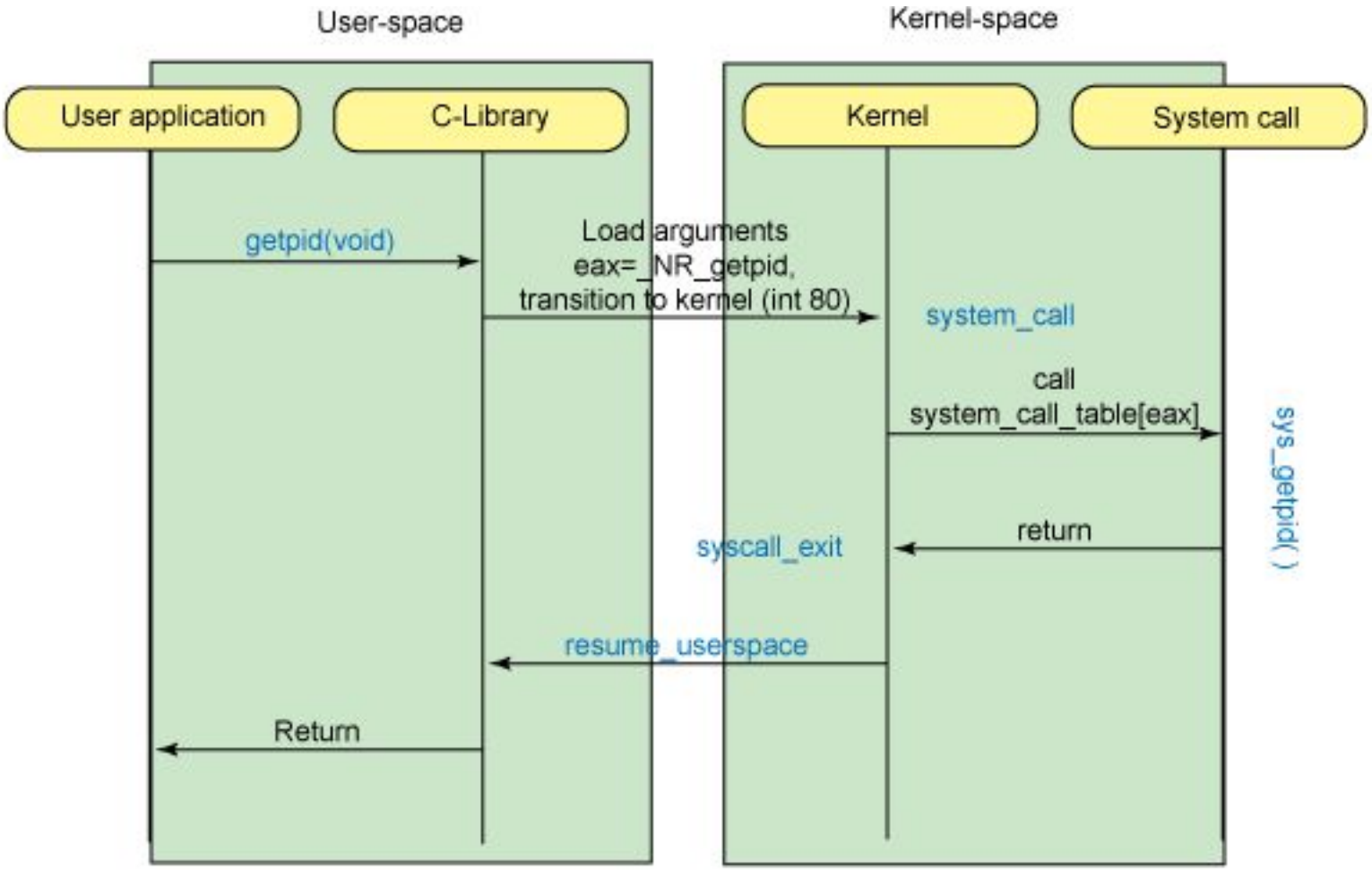
Uma interface binária de aplicação (ABI, do inglês ***Application Binary Interface***) descreve a interface de baixo nível entre uma aplicação e o sistema operacional, entre a aplicação e suas bibliotecas ou entre componentes de uma aplicação.

Bibliotecas, syscalls, kernel e hardware

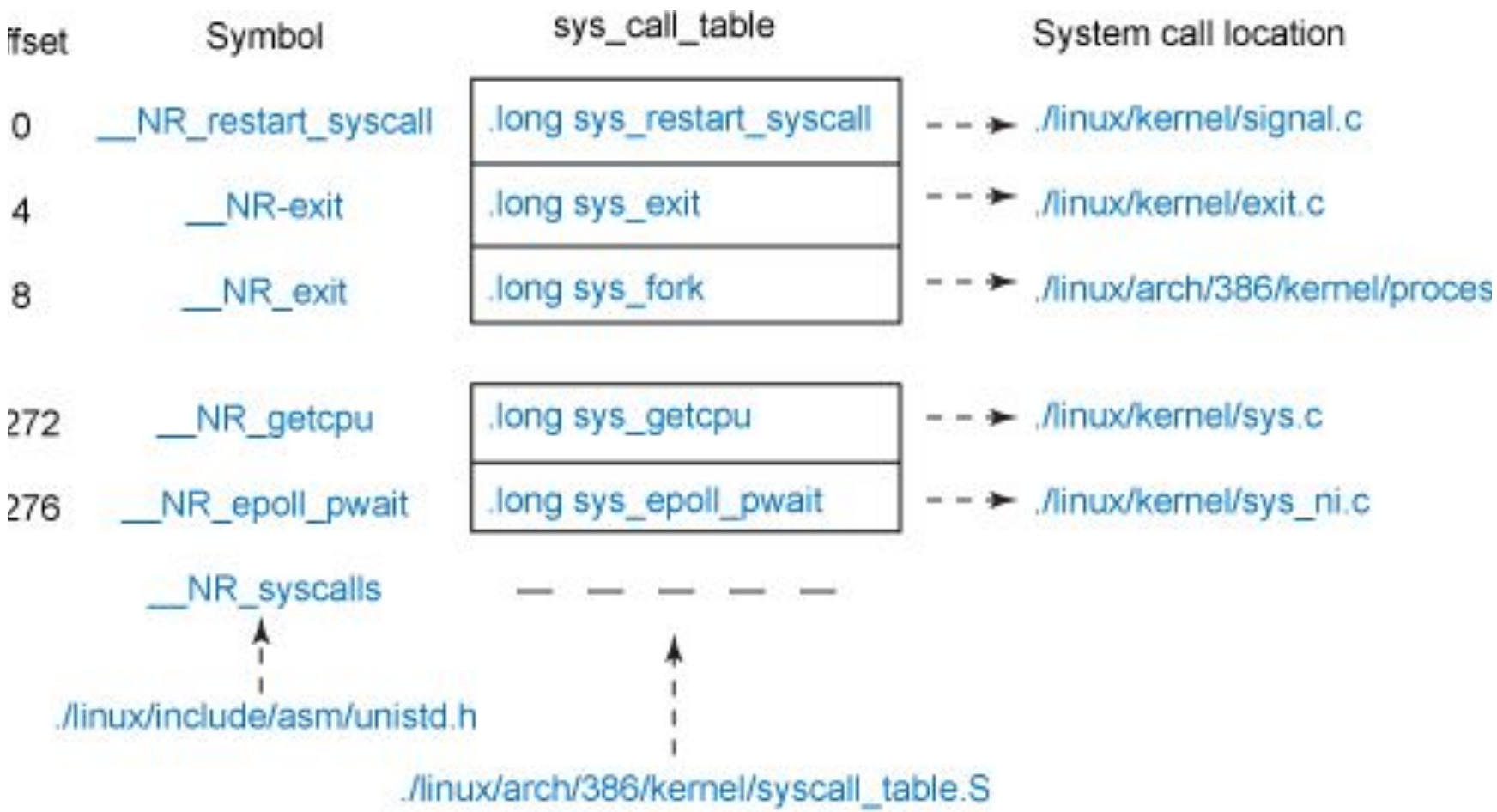
- Relacionamento entre uma Library Function, uma System Call, o Kernel e o hardware.



Fluxo de uma chamada a System Call



System Call Table



Rescrevendo o conteúdo do hello.c

- Vamos usar a função `write` ao invés de `printf`
- Objetivo: Trocar a função write por uma syscall
- O que usar no include?
  - Ver no comando `$ man 2 write`

```
#include <unistd.h>

int main(void)

{
   write(1, "Hello, World!\n", 14);

   return 0;
}
```

Compilando e desmontando o novo hello

```
$ gcc -o hello hello.c
$ ./hello
Hello, World!
$ objdump -d hello
```

```
0000000000004004f4 <main>:
  4004f4:       55                      push   %rbp
  4004f5:       48 89 e5                mov    %rsp,%rbp
  4004f8:       ba 0e 00 00 00          mov    $0xe,%edx
  4004fd:       be 0c 06 40 00          mov    $0x40060c,%esi
  400502:       bf 01 00 00 00          mov    $0x1,%edi
  400507:       e8 e4 fe ff ff          callq  4003f0 <write@plt>
  40050c:       b8 00 00 00 00          mov    $0x0,%eax
  400511:       5d                      pop    %rbp
  400512:       c3                      retq
```

Substituindo a função por uma syscall

- Existe uma syscall no Linux, chamada `sys_write`.
  - Podemos chamá-la diretamente no código.
    - Ver includes através de `$ man syscall`

```
#include <unistd.h>
#include <sys/syscall.h>

int main(void)
{
   syscall(SYS_write,1, "Hello, World!\n", 14);

   return 0;
}
```

Compilando e desmontando o novo hello

```
$ gcc -o hello hello.c
$ ./hello
Hello, World!
$ objdump -r hello
```

```
000000000004004f4 <main>:
  4004f4:       55                      push   %rbp
  4004f5:       48 89 e5                mov    %rsp,%rbp
  4004f8:       b9 0e 00 00 00          mov    $0xe,%ecx
  4004fd:       ba 0c 06 40 00          mov    $0x40060c,%edx
  400502:       be 01 00 00 00          mov    $0x1,%esi
  400507:       bf 01 00 00 00          mov    $0x1,%edi
  40050c:       b8 00 00 00 00          mov    $0x0,%eax
  400511:       e8 ea fe ff ff          callq  400400 <syscall@plt>
  400516:       b8 00 00 00 00          mov    $0x0,%eax
  40051b:       5d                      pop    %rbp
  40051c:       c3                      retq
```

Rastreando SysCall

```
$ strace ./hello
```

```
execve("./hello", ["./hello"], [/* 19 vars */]) = 0
brk(0)                                  = 0x1547000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f1ebcda0000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=20925, ...}) = 0
mmap(NULL, 20925, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1ebcd9a000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3,
"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\320\37\2\0\0\0\0\0"..., 832) =
832
fstat(3, {st_mode=S_IFREG|0755, st_size=1845024, ...}) = 0
...
write(1, "ABABABA\n", 8ABABABA
)                 = 8
exit_group(0)                           = ?
+++ exited with 0 +++
```

Referencias

- http://www.ibm.com/developerworks/linux/library/l-system-calls/

Fim do Laboratório 1