Exercise 2

a) List and explain the sequence of calls performed by the program in x86-64/prog.s

---

Este programa realiza uma série de chamadas de sistema (SYSCALL) para interagir com o sistema operativo.

Segue-se uma análise do que cada parte do código faz, juntamente com a sequência de chamadas de sistema.

Data Section

```
.section .rodata

data:   .byte  47, 101, 116,  99
        .byte  47, 111, 115,  45
        .byte 114, 101, 108, 101
        .byte  97, 115, 101,   0
        .byte  73,  83,  69,  76
```

- Esta secção define um segmento *read-only* com a *string* "/etc/os-release\0ISEL".
  - 47, 101, 116, 99 -> /etc
  - 47, 111, 115, 45 -> /os-
  - 114, 101, 108, 101 -> rele
  - 97, 115, 101, 0 -> ase\0 (o `\0` representa o terminador nulo)
  - 73, 83, 69, 76 -> ISEL

Text Section

```
.text
.globl _start
_start:
```

- Isto marca o início da secção `.text` e define o ponto de entrada `_start`.

Sequencia de System Calls

1. **Open File Descriptor**

```
movq $-100, %rdi
leaq data(%rip), %rsi
xorq %rdx, %rdx
movq $257, %rax
syscall
```

```
- **System Call**: `openat` (syscall number 257)
- **Argumentos**:
    - `rdi = -100`: AT_FDCWD (current working directory)
    - `rsi = data`: *Pointer* para *file path* "/etc/os-release"
    - `rdx = 0`: Flags (read-only)
```

- **Objetivo**: Abrir o ficheiro "/etc/os-release" em modo de leitura.
- **Return**: O *File descriptor* guardado em `rax`, movido em seguida para `r15` (código

2. **Move file pointer**

```
movq %r15, %rdi
xorq %rsi, %rsi
movq $2, %rdx
movq $8, %rax
syscall
```

- **System Call**: `lseek` (syscall number 0)
- **Argumentos**:
  - `rdi = r15`: *File descriptor*
  - `rsi = 0`: Deslocamento, 0 neste caso, início do ficheiro.
  - `rdx = 2`: Move o valor imediato 2 para o registo %rdx. %rdx é usado para passar o t
- **Objetivo**: Determinar o tamanho do ficheiro. Move o ponteiro do ficheiro para uma nov

3. **Memory Mapping**

```
xorq %rdi, %rdi
movq %r14, %rsi
movq $1, %rdx
movq $2, %r10
movq %r15, %r8
xorq %r9, %r9
movq $9, %rax
syscall
```

- **System Call**: `mmap` (syscall number 9)
- **Argumentos**:
  - `rdi = 0`: Address (NULL, let the kernel choose)
  - `rsi = r14`: Length (number of bytes read)
  - `rdx = 1`: Protection (PROT_READ)
  - `r10 = 2`: Flags (MAP_PRIVATE)
  - `r8 = r15`: File descriptor
  - `r9 = 0`: Offset
- **Objetivo**: Mapear o ficheiro na memoria.
- **Return**: O endereço da área mapeada em `rax`.

4. **Escrever no Standard Output**

```
movq $1, %rdi
movq %rax, %rsi
movq %r14, %rdx
movq $1, %rax
syscall
```

- **System Call**: `write` (syscall number 1)
- **Arguments**:
  - `rdi = 1`: *File descriptor* (stdout)
  - `rsi = rax`: *Buffer* - *poiter* para o endereço da área mapeada.
  - `rdx = r14`: Numero de bytes a escrever
- **Objetivo**: Escreve o número de *bytes* começado no endereço apontado para o *standard

5. **Exit**

```
  xorq %rdi, %rdi
  movq $231, %rax
  syscall
```

```
- **System Call**: `exit_group` (syscall number 231)
- **Arguments**:
    - `rdi = 0`: `Exit status` - código de saída
- **Objetivo**: Terminar o programa.
```

Análise da execução do programa com `strace`

```
execve("./prog", ["./prog"], 0x7ffebfcdcd40 /* 51 vars */) = 0
openat(AT_FDCWD, "/etc/os-release", O_RDONLY) = 3
lseek(3, 0, SEEK_END)                   = 400
mmap(NULL, 400, PROT_READ, MAP_PRIVATE, 3, 0) = 0x79a714280000
write(1, "PRETTY_NAME=\"Ubuntu 24.04.1 LTS\""..., 400) = 400
exit_group(0)                           = ?
+++ exited with 0 +++
```

Print da execução do programa no *stdout*

```
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
```

Resumo

O programa abre o ficheiro "/etc/os-release", em modo de leitura, reposiciona o deslocamento do ficheiro aberto associada ao *file descriptor* até ao final, mapea o conteudo lido para a memória, escreve-o para o *standard output*, e por fim termina (exits).

É de salientar que quando o `mmap` é chamado, este mapeia o arquivo em memória, mas não o carrega para a memória física. Quando o programa tenta aceder À memória mapeada, ocorre um `page fault` e o sistema operativo carrega o a página do ficheiro na memória física.

Isto é uma forma de, por exemplo, mapear ficheiros grandes de forma eficiente e com baixo consumo de memória física, uma vez que só vai ser carregado na memória física as partes realmente necessárias quando é efetuado um acesso às mesmas.

Neste programa, quando o `write` é chamado, o sistema operativo carrega a página do ficheiro na memória física, e o conteúdo é escrito no *standard output*.