

## APRENDENDO A INTERPRETAR A SYSCALL

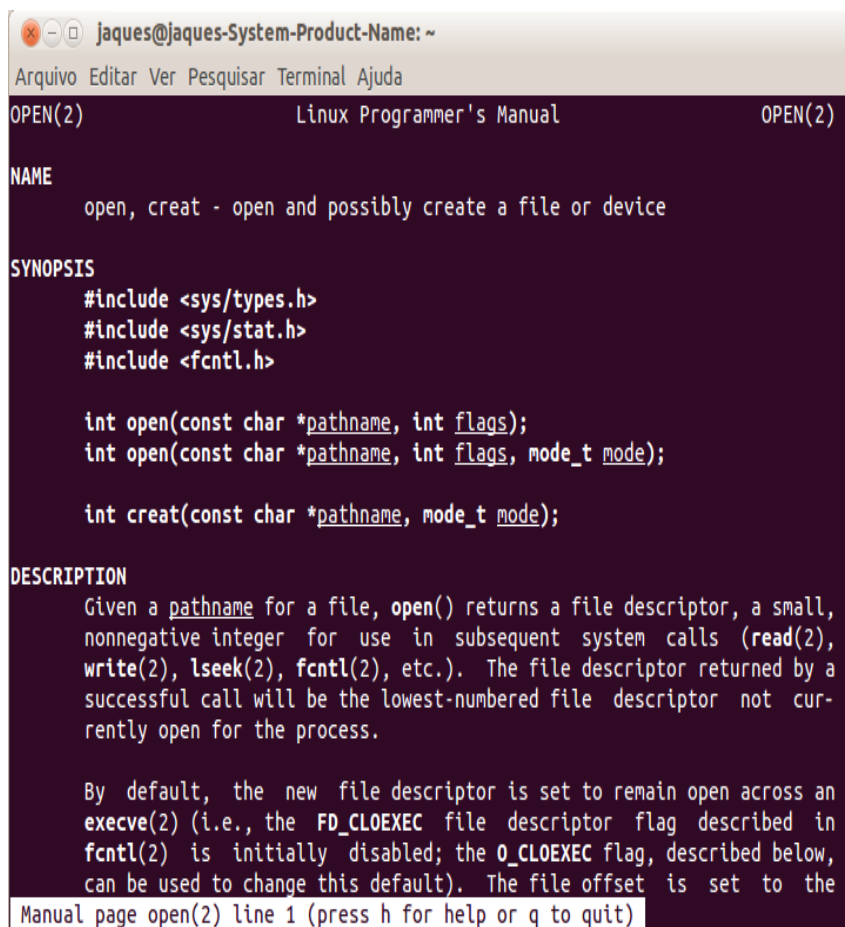
Quando olhamos para um serviço da syscall as vezes temos a impressão que estamos tratando de hieróglifos. Mas vamos tentar entender como funcionaria isso na linguagem C.

Vamos ver um exemplo em que temos que abrir um arquivo em C.

Para isso vamos usar o help do linux sobre as funções já disponíveis para o mesmo. Não esqueçamos que o Unix criou a linguagem C, pois precisava de alguns recursos. E C foi melhorado em C. O comando após o \$, seria:

```
jaques@jaques-System-Product-Name:~$ man 2 open
```

Isto nos dará informações sobre como abrir um arquivo. Veja a figura:



```
jaques@jaques-System-Product-Name: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
OPEN(2) Linux Programmer's Manual OPEN(2)

NAME
    open, creat - open and possibly create a file or device

SYNOPSIS
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>

    int open(const char *pathname, int flags);
    int open(const char *pathname, int flags, mode_t mode);

    int creat(const char *pathname, mode_t mode);

DESCRIPTION
    Given a pathname for a file, open() returns a file descriptor, a small,
    nonnegative integer for use in subsequent system calls (read(2),
    write(2), lseek(2), fcntl(2), etc.). The file descriptor returned by a
    successful call will be the lowest-numbered file descriptor not cur-
    rently open for the process.

    By default, the new file descriptor is set to remain open across an
    execve(2) (i.e., the FD_CLOEXEC file descriptor flag described in
    fcntl(2) is initially disabled; the O_CLOEXEC flag, described below,
    can be used to change this default). The file offset is set to the

Manual page open(2) line 1 (press h for help or q to quit)
```

Você poderá ver a mesma coisa em qualquer livro de C em português. Aqui está em inglês.

É claro que o parâmetro `pathname` é uma string onde especificamos o caminho e nome do arquivo que queremos abrir. E o **flags**?

Se continuarmos a pressionar <ENTER> veremos que ele diz que `flags` representa o modo de abertura: `O_RDONLY`, `O_WRONLY`, ou `O_RDWR`. Se somente para leitura, para gravação, ou leitura e gravação. Caso queiramos sair do help temos que pressionar “q”, sem as aspas. Esse “q” é de “quit”. Olhando na última linha da figura já aparece essa dica. Se quisermos algum help pressionaríamos “h”. Experimente pressionar “h”, para ver a lista de comandos que é oferecida.

O **mode** especifica as permissões para uso no caso de um novo arquivo ser criado. Se quiser, veja mais informações no help. Via de regra nosso `mode` será 0.

Então em C seria algo assim:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
```

```

int main()
{
    char *nomeArquivo="teste.txt";
    int fd;
    ...
    fd=open(nomeArquivo, O_RDWR,0); //abrir em modo de leitura e gravação
    if(fd==-1){
        printf("Erro na abertura do arquivo\n");
        exit(1);
    }
}

```

fd é o descritor do arquivo, já comentado anteriormente. Um número que representa o arquivo, e que o sistema operacional já associou ao arquivo que queremos manipular. Caso seja -1, algo deu errado.

### COMO SERIA EM ASSEMBLY?

Antes de tratarmos como seria em Assembly, vamos ver os **flags** de acesso e seus respectivos valores:

```

;=====
;VALORES DE MODO DE ACESSO EM DECIMAL E OCTAL (começa com 0 é octal)
;=====
;
; ACESSO                DECIMAL                OCTAL
;O_RDONLY                0                    00
;O_WRONLY                1                    01
;O_RDWR                 2                    02
;O_CREAT                 64                   0100
;O_EXCL                  128                   0200
;O_TRUNC                 512                   01000
;O_APPEND                1024                  02000
;O_DIRECT                16384                  040000
;O_NOFOLLOW              131072                 0400000
;O_FSYNC                 1052672                04010000
;
;-----
;=====

```

A tabela Syscall é composta dos seguintes parâmetros especificados nos registradores

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
------	-------------	------	------	------	------	-----	-----

A syscalltable mostra o seguinte para a função open

2	sys_open	const char *filename	int flags	int mode
---	----------	----------------------	-----------	----------

Como o comando man 2 open vimos o formato da função open, onde cada parâmetro corresponde a um registrador conforme abaixo:

```
                rdi                rsi                rdx
                ↓                  ↓                  ↓
int open(const char *pathname, int flags, mode_t mode);
```

Os parâmetros internos, correspondem, na ordem, por rax,rdi,rsi,rdx, r10, r8, r9.

A pergunta que não quer calar é “onde diabos retorna o fd?”. Qualquer resultado de conta, operação, resto vai para o acumulador (eax). Portanto, o valor de rax após “syscall” terá o valor do descritor. E será -1 se der algum problema.

O equivalente em assembly seria assim:

```
section .data
    nomeArquivo db "teste.txt",0    ; o 0 é o '\0' da linguagem C
    O_RDWR equ 2    ;definindo a constante
    fd db 0        ;definindo um descritor
    MsgErro db "Erro na abertura do arquivo",10,13
    tamMsgErro equ $-MsgErro
    ...

section .text
    global _start

    _start:
    ;fd=open(nomeArquivo, O_RDWR,0)
    mov rax,2        ;serviço open
    mov rdi, nomeArquivo    ;
    mov rsi, O_RDWR    ; ou mov rsi,2
    mov rdx,0
    syscall
    cmp rax,-1        ; compara com -1

    ;se rax==-1 algo deu errado
    je deuErrado    ; salta para endereço deuErrado, se ax==-1
    mov [fd],eax    ; salva em uma variável fd (existem outros modos)
    .
    .
    .

jmp saidaOk
deuErrado:
    ;uma mensagem de erro aqui write(4)
    mov rax,2
    mov rdi,1        ;grava na tela
    mov rsi,MsgErro
    mov rdx,tamMsgErro
    syscall
    ;exit(1) sai com erro
    mov rax,60
    mov rdi,1
```

```
syscall
```

```
saidaOK:  
    ;exit(0) sai sem erro  
    mov rax,60  
    mov rdi,0  
    syscall
```

## LEITURA OU ESCRITA EM UM ARQUIVO

Antes de ler ou gravar em um arquivo precisamos abri-lo, do mesmo modo que não conseguimos escrever em um caderno com ele dentro da mochila. Precisamos abrir o caderno e então escrever ou ler o que desejamos. Suponha que já abrimos o arquivo e queremos ler o conteúdo. Vamos ao help:

```
jaques@jaques-System-Product-Name:~$ man 2 read
```

Ele mostraria uma tela semelhante a figura já mostrada, e entre um monte de coisas veremos:

```
ssize_t read(int fd, void *buf, size_t count);
```

Aqui a função diz que precisa do descritor *fd* de um buffer *buf* (um vetor de char, ou string), e um tamanho *size\_t*. Precisamos informar de quantos em quantos bytes (caracteres) desejamos ler o nosso arquivo. Isso nós especificamos em *size\_t*. A função retornará em *buf* o que conseguiu ler e em *ssize* a quantidade de caracteres lida. Suponha que eu queira ler de 80 em 80 caracteres mas o número de bytes de meu arquivo não é múltiplo de 80. No final, *ssize* retornará um valor menor que 80, mas será maior que 0 se leu alguma coisa. Quando o mesmo retornar 0 em *ssize*, significa que chegou ao fim do arquivo.

Em C faríamos o seguinte:

```
...  
int main()  
{  
    int tam, fd;  
    char stringLida[80]; //lembrando que ponteiro e vetor é o mesmo  
    ...  
    for(;;)  
    {  
        tam=read(fd,stringLida,80); //lendo em stringLida 80 caracteres  
        if (tam>0) printf("%s",stringLida);  
        else break;  
    }  
    ...  
}
```

Na syscall seria então

```
          rdi      rsi      rdx  
          ↓        ↓        ↓  
ssize_t read(int fd, void *buf, size_t count);
```

Novamente, o retorno do tamanho lido seria em *eax*. Então o código em assembly, segundo a tabela, seria:

```
...  
for_simulado:
```

```

mov rax, 0          ;read
mov rdi, [fd]      ;fd do ARQUIVO
mov rsi, buffLeitura ;buffLeitura é o vetor onde será lido o arquivo
mov rdx, 80 ;tamanho máximo do buffLeitura

syscall

cmp eax,0          ;compara eax com 0
je break          ; salta para break se igual
mov [tam],rax
mov rax,1          ; write
mov rdi, 1         ; tela
mov rsi, buffLeitura
mov rdx, [tam]
syscall
jmp for_simulado ; volta para for_simulado incondicionalmente
break:
...

```

## VALORES DE PERMISSAO DE ARQUIVO

As permissões em Linux utilizam o seguinte formato:

USER | GROUP | OTHER

R W X | R W X | R W X

A permissao é composta de 9 bits, 3 para USER, 3 para GROUP, e 3 para OTHER

Por exemplo, se somente W e X de USER estivessem setados, o valor de permissao seria 011 000 000.

Isso é equivalente em decimal a 0300, coloca-se um zero antes, devido a serem números octais.

Repare que se somente W fosse setado, teria o valor 2 ou 010, se somente R fosse setado, teria o valor 4, se somente X fosse setado, teria o valor 1. Assim 011=3, que corresponde a W e X setado, ou seja, 2+1=3. Associamos os valores

R=4, W=2, X=1, em ordem decrescente de potência de 2. Logo, se quero leitura e escrita e execução para USER e somente leitura e execução para GROUP teríamos 0750. É assim no linux, com a atribuição pelo comando chmod. Exemplo: chmod 0750 nomeArquivo, o chmod a+x, é o mesmo que chmod 0111. a é de all

### ATIVIDADES:

De posse das informações dadas, escrever em código assembly as funções 1, 3, 4, 5, 6, 8, 12 e 13. Aproveitem para ler o último arquivo pdf que enviei em e-mail separado. Descrever o que faz cada função detalhadamente. Entregar na secretaria, em manuscrito, até as 11h.