

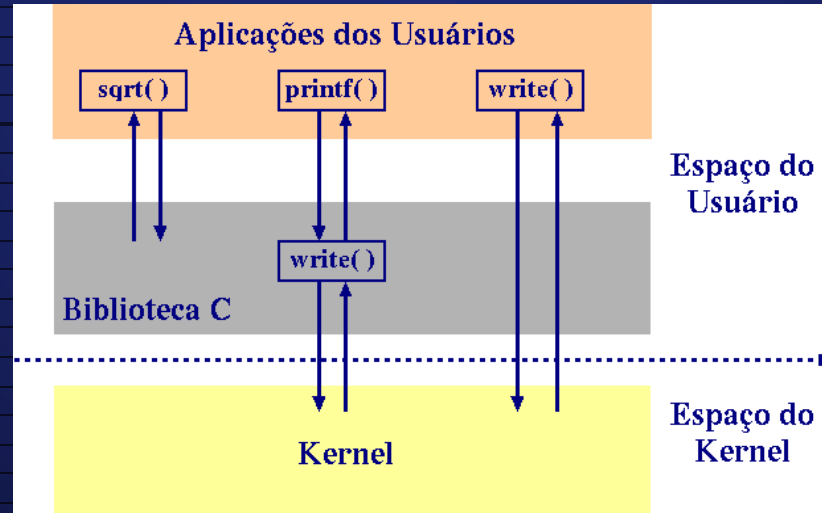
CHAMADAS DE SISTEMA

Terabytes

Definição



As system calls (ou chamadas de sistema) são APIs para criar uma interface entre o *user space* (espaço onde o utilizador tem permissão para executar os seus programas) e o *kernel space* (espaço onde o sistema operacional executa os seus programas que necessitam de um nível de permissão superior).



Espaço de Usuário X Espaço do Kernel

Espaço do Usuário: É a região da memória onde os programas do usuário são executados. Esses programas não têm acesso direto ao hardware ou a áreas de memória críticas do sistema.

Espaço do Kernel: É a região da memória onde o núcleo (ou kernel) do sistema operacional é executado. O kernel tem acesso direto ao hardware e pode executar qualquer instrução da máquina.

Funções das Chamadas de Sistema

As chamadas de sistema são usadas para solicitar serviços do sistema operacional que requerem privilégios mais elevados, como leitura e escrita em arquivos, comunicação de rede, gerenciamento de processos e acesso a dispositivos de hardware.

Sobre o código:

Foi implementado em Linguagem C e Assembly um código que lê um número inteiro e verifica se é par ou ímpar e escreve o resultado no console.

Syscalls usadas no código

Em C:

01

```
write(STDOUT_FILENO, prompt, strlen(prompt));
```

Parâmetros:

- **STDOUT_FILENO**: é o descritor de arquivo para a saída padrão (geralmente o terminal ou console).
- **prompt**: é o buffer de dados que será escrito. Neste caso, é uma string que você deseja imprimir na saída padrão.
- **strlen(prompt)**: é o número de bytes a serem escritos do buffer de dados. **strlen** é uma função da biblioteca padrão C que retorna o comprimento de uma string (não incluindo o caractere nulo de terminação).



02




```
read(STDIN_FILENO, buf, BUF_SIZE);
```

Parâmetros:

- **STDIN_FILENO**: é o descritor de arquivo para a entrada padrão (geralmente o teclado).
- **buf**: é o buffer onde os dados lidos serão armazenados.
- **BUF_SIZE**: é o número máximo de bytes a serem lidos. Em outras palavras, é o tamanho do buffer.

03

```
_exit(0);
```

- **_exit**: é a chamada de sistema. Ela termina imediatamente o processo que a chamou.
- 
- 
- 

Compilação e execução do código

Em C:

```
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ gcc Impar_par.c -o impar_par
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./impar_par
Digite um número inteiro: 20
20 é par.
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./impar_par
Digite um número inteiro: 63
63 é ímpar.
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./impar_par
Digite um número inteiro: 98
98 é par.
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./impar_par
Digite um número inteiro: 135
135 é ímpar.
```


Syscalls usadas no código

Em Assembly:

01 `sys_write:`

```
_start:  
; Escreve a mensagem de prompt na tela  
mov eax, 4 ; sys_write  
mov ebx, 1 ; stdout  
mov ecx, prompt ; endereço da string  
mov edx, 27 ; tamanho da string  
int 0x80 ; chama a syscall
```


02

sys_read:

read_num:

```
; Lê um caractere do teclado  
mov eax, 3; sys_read  
mov ebx, 0; stdin  
mov ecx, char_in  
mov edx, 1  
int 0x80; chama a syscall
```

03

sys_exit:

exit:

```
; Sai do programa  
mov eax, 1 ; sys_exit  
xor ebx, ebx ; código de saída (zero indica sucesso)  
int 0x80 ; chama a syscall
```

Compilação e execução do código

Em Assembly:

```
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ nasm -f elf64 Assembly.asm
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ld Assembly.o -o assembly
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./assembly
Digite um número inteiro: 155
é ímpar.
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./assembly
Digite um número inteiro: 48
é par.
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./assembly
Digite um número inteiro: 53
é ímpar.
filipecorrea@filipecorrea-VirtualBox:~/Downloads$ ./assembly
Digite um número inteiro: 66
é par.
```

CONSIDERAÇÕES FINAIS

Este trabalho explorou as chamadas de sistema (syscalls), essenciais para a interação entre programas e o sistema operacional. Implementamos syscalls em C e Assembly, destacando a eficiência do Assembly e as abstrações úteis do C. Reforçando a importância das syscalls para a segurança, estabilidade e eficiência dos sistemas operacionais.

