Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Sistemas Operativos

Kernel & User Mode

Kernel mode vs user mode

Em cada momento, o CPU encontra-se num determinado **nível de privilégio**.

Os CPUs da família x86-32 (IA-32) e x86-64 (AMD64/Intel64) suportam 4 níveis de privilégio, designados **rings** (0 .. 3).

Os sistemas operativos Windows e Linux usam essencialmente os dois níveis extremos:

- Ring 0 - *kernel mode* : nível de privilégio máximo, em que é possível executar qualquer instrução, aceder a todos os registos e a toda a memória.
- Ring 3 - *user mode* : nível de privilégio mínimo, com algumas instruções, alguns registos e parte do espaço de endereçamento inacessíveis.

Execução em kernel ou user mode

O código central do sistema operativo, designado *kernel*, bem como os controladores dos dispositivos *hardware* (*device drivers*), executam-se com o nível de privilégio máximo, em ***kernel mode***.

- △ acesso directo a toda a funcionalidade, mas
- ▽ um erro de execução compromete todo o sistema

As aplicações instaladas pelo utilizador e vários utilitários de sistema executam-se com o nível de privilégio mínimo, em ***user mode***.

- △ erros graves de execução comprometem apenas a aplicação, mas
- ▽ todas as acções sobre o sistema têm de ser pedidas ao *kernel*

Transições entre user mode e kernel mode

O CPU arranca sempre em *kernel mode* (ring 0), executando-se o código de arranque do sistema.

Quando as inicializações de sistema estão concluídas, é lançado o primeiro processo para execução em *user mode*. A partir desse momento, há três motivos para regressar a *kernel mode*:

1. Atendimento de pedidos de interrupção em linhas de *hardware*
2. Tratamento de excepções do CPU (instruções ilegais, endereços de memória inacessíveis, divisão por 0, etc.)
3. Chamadas de sistema (*system calls*): invocação de operações do *kernel*

Mecanismo uniforme de entrada em kernel mode

O atendimento de pedidos de interrupção de linhas de *hardware* exige que esteja previsto um mecanismo para transitar de *user mode* para *kernel mode* quando surge um pedido de interrupção.

Quando é executada uma instrução ilegal, uma acesso a um endereço de memória inválido, um divisão por 0 ou outro caso que o CPU não permita, é emitida uma excepção, na forma de um pedido de interrupção originado internamente pelo próprio CPU, que o *kernel* deve atender em *kernel mode*.

Para uniformizar as entradas em *kernel mode*, a execução de uma interrupção de *software* (instrução *int*) permite executar uma chamada de sistema (*system call*).

API de sistema

O sistema operativo define o conjunto de operações que as aplicações de *user mode* podem requerer ao *kernel* que execute. Exemplos:

0	read	sys_read	fs/read_write.c
1	write	sys_write	fs/read_write.c
2	open	sys_open	fs/open.c
3	close	sys_close	fs/open.c
22	pipe	sys_pipe	fs/pipe.c
33	dup2	sys_dup2	fs/file.c
39	getpid	sys_getpid	kernel/sys.c
57	fork	stub_fork	kernel/fork.c
59	execve	stub_execve	fs/exec.c

Fonte: https://filippo.io/linux-syscall-table/

Chamada de sistema

rax: identificador da operação de sistema

rdi, rsi, rdx, r10, r8, r9: argumentos da operação

Instrução **syscall** provoca entrada em *kernel mode*, para a rotina configurada previamente pelo sistema operativo em registos especiais (MSR) do CPU.

Execução em *kernel mode* suportada por *stack* específico para *kernel*.

Saída de kernel mode com **sysret**, com o valor de retorno deixado em **rax**.

Originalmente (ainda em x86-32) era usada a instrução **int <n>**, com efeito semelhante, mas de execução mais complexa e demorada.

Exemplo de chamada de sistema (getpid)

```
        .text
        .global xgetpid
xgetpid: movq $39, %rax
        syscall
        ret
        .end
```