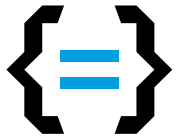


Univerzita Hradec Králové
Fakulta informatiky a managementu

Systémová volání

Mgr. Josef Jan Horálek, Ph.D. & Ing. Tomáš Svoboda, Ph.D.

Systémová volání

- = Systémová volání = volání jádra
  - = základní komunikace aplikačních programů s jádrem
- = Tvůrce programu obvykle oddělen vrstvou standardní knihovny
  - = nepotřebuje tedy znát přesné chování jádra
  - = je však dobré je pochopit

Základní princip

- Jak vypadá vztah uživatelský proces – jádro ?
- Jedná se o dva rozdílné světy:
  - uživatelský proces běží ve svém adresném prostoru (instrukce i data) a vykonává kód programu,
  - někdy potřebuje službu, kterou pro něj vykoná jen jádro – systémové volání.

= Příklad: Otevření a zapsání do souboru v jazyce C

```
FILE * fopen (const char * filename, const char * mode);

int main () { //Velmi zjednodušeno, chybí jakákoliv kontrola chyb FILE *
pFile;

  pFile = fopen("myfile.txt","w"); //Syscall volání v implementaci
  fputs("fopen example",pFile); //Syscall volání v implementaci
  fclose(pFile); // Syscall volání v implementaci

  return 0;
}
```

- Příklad 2:
- Zjištění a nastavení parametrů zařízení – funkce ioctl () .

```
int ioctl (int fd, unsigned long request,…);
```

- Podle konkrétní hodnoty požadavku má volání dva či tři argumenty.
- V jádře pak vypadá prototyp funkce např. takto:

```
long (*unlocked_ioctl)(struct file *filp, unsigned int
cmd, unsigned long arg);
```

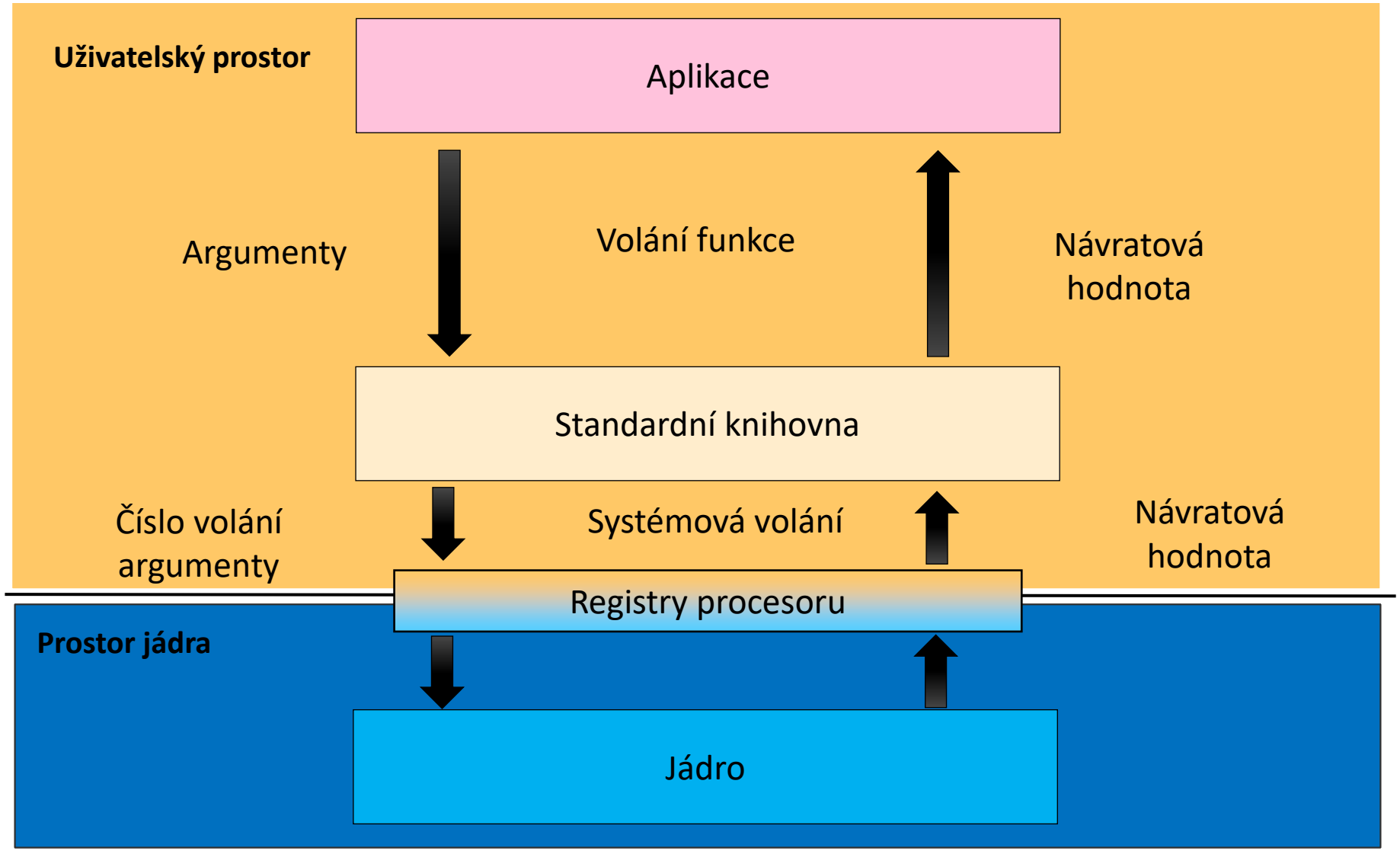
Základní princip

- Můžeme nyní zanedbat odlišné datové typy a další detaily
  - pak se jedno volání funkce **transformuje** na jiné.
- V čem se tedy systémové volání liší od běžné funkce?
  - leží právě na hranici této transformace.

Jak to funguje - například ioctl()

1. Uloží se aktuální obsah registrů
2. Do registrů se uloží číslo systémového volání a parametry
3. Předá se řízení jádru (přepnutí do režimu jádra a začne vykonávat jeho kód)
4. Podle čísla volání se vyhodnotí, jaká funkce v jádře se zavolá
5. Protože jde o volání spojené s otevřeným souborovým deskriptorem, najde se odpovídající objekt souboru
6. Podle objektu souboru se zavolá příslušná výkonná funkce v ovladači
7. Ve výkonné funkci se provede vše potřebné a na konci se vrátí výsledek
8. Do registru se uloží výsledek operace
9. Řízení se předá zpět programu resp. Standardní knihovně (uživatelský režim)
10. Výsledek se přenese z registru do paměti
11. Obnoví se uložený stav registrů

Jak to funguje - například ioctl()



Do režimu jádra a zpět

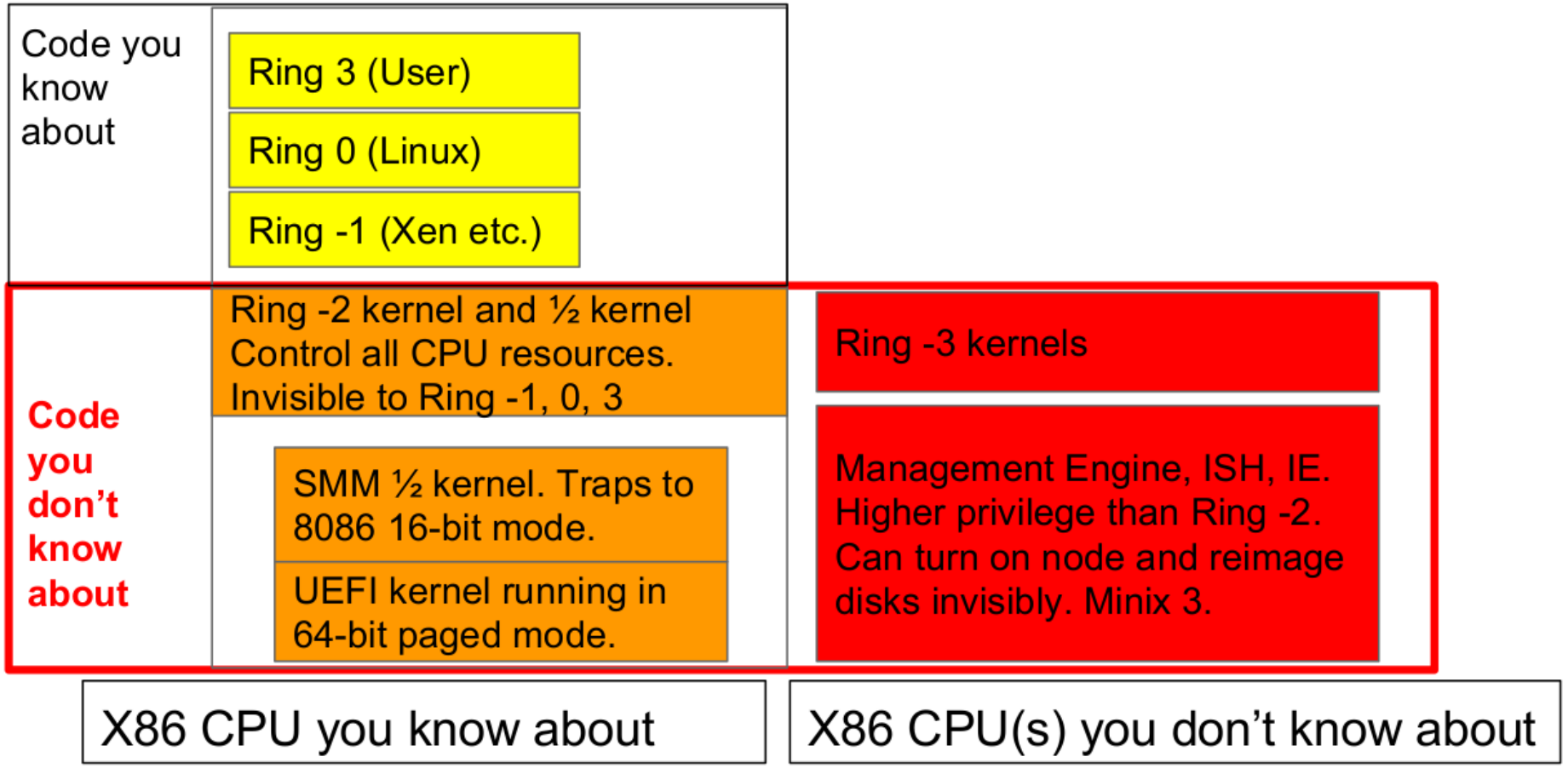
- Jak vlastně probíhá předání řízení jádru a opětovné předání zpět?
- Dvě metody (platformě závislé):
  - speciální instrukce
  - přerušení

- Starší metoda
- Široké použití
  - U jednočipů, kde neběží žádný OS, se používá stále
  - U netypických architektur procesorů také
  - Obecně pomalejší u mikroprocesorů
- Volající proces způsobí softwarové přerušení (platforma x86 nejčastěji 0x80).
- Začne se obsluhovat, procesor přepnut do režimu jádra – provádí kód jádra.
- Po skončení je obnoven stav procesoru – přepnut do uživatelského režimu.

Speciální instrukce

- Nevyužívá přerušení.
- Instrukce **`sysenter`** sama zajistí přepnutí do režimu jádra.
- Nastaví registry (CS, EIP atd.) a předá řízení kódu jádra.
- Na závěr se vykoná instrukce **`sysexit`**, která připraví registry pro návrat a přepne do uživatelského režimu.
- Na AMD64 se jedná o instrukce **`syscall`** a **`sysret`**

The operating systems



Reakce na výsledek

- Systémové volání končí opuštěním jádra a předáním výsledku přes registr.
- Konkrétní hodnoty závisí na potřebách volající aplikace, existuje však několik základní možností:
  - úspěšně zakončené volání, které nepřenáší zpět žádnou konkrétní hodnotu, vrací nulu (0) a příslušná hodnota se ukládá do proměnné v paměti `time ()`
  - úspěšně zakončené volání, které přenáší návratovou hodnotu, vrací tuto hodnotu (obecně jakákoli hodnota obvykle nezáporná)
  - neúspěšně zakončené volání vrací zápornou hodnotu odpovídající kódu chyby (`errno`)

Přerušené volání

- Často nastává situace, kdy je vykonáván kód uvnitř implementace systémového volání a v tu chvíli přijde signál, na který se musí reagovat.
- V tuto chvíli může nastat jedna ze tří základních situací:
  - 1) Systémové volání je buď krátké, nebo se nachází již blízko svého konce, tedy nemusí již na nic čekat. Pak se volání normálně dokončí a vrátí standardní výsledek.

Přerušené volání

- 2) Volání bylo přerušeno na začátku, resp. v situaci, kdy buď nebyly v systému provedeny žádné změny, nebo byly provedeny bezpečně reverzibilní změny. Tato situace se řeší opuštěním jádra a vrácením hodnoty.
  - ERESTARTSYS – volání bude opět restartováno; výjimkou je případ, kdy byl signál explicitně obsloužen a neměl nastaven příznak SA_RESTART. Tehdy nastane přerušení volání a návrat do programu
  - ERESTARTNOHAND – volání bude restartováno vždy, když není signál explicitně obsloužen uživatelskou rutinou
  - ERESTART_RESTARTBLOCK – podobná situace s použitím jiného volání
  - ERESTARTNOINTR – volání se restartuje za každých okolností.

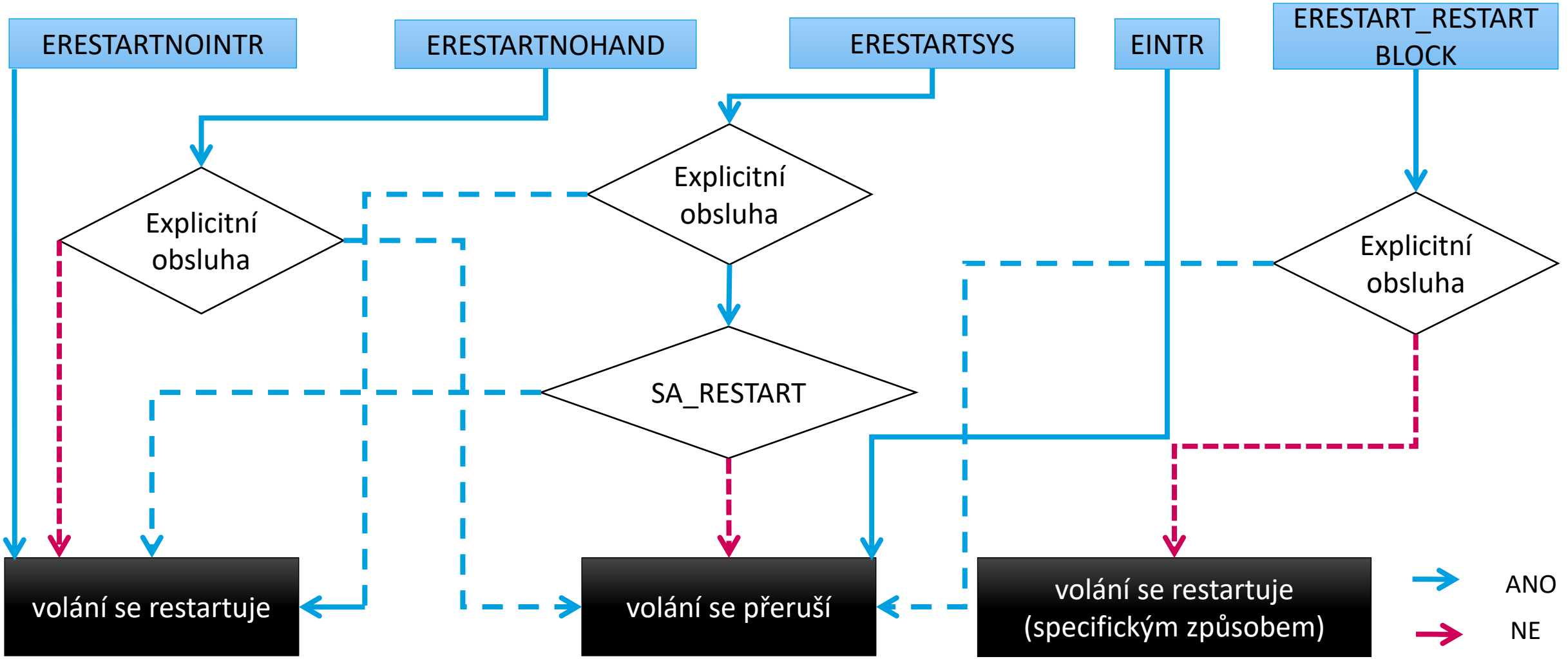
Přerušené volání

- 3) K přerušení volání došlo v situaci, kdy již byly provedeny nevratné změny v systému, například byl do zařízení odeslán manipulační příkaz.
  - Situace se řeší opuštěním jádra s návratovou hodnotu –EINTR.
  - Nejdříve se samozřejmě musí systém dostat do konzistentního stavu.
- EINTR – volání se vždy přeruší.

```
/* These should never be seen by user programs.  To return one of ERESTART*
 * codes, signal_pending() MUST be set.  Note that ptrace can observe these
 * at syscall exit tracing, but they will never be left for the debugged
user
 * process to see.  */
```

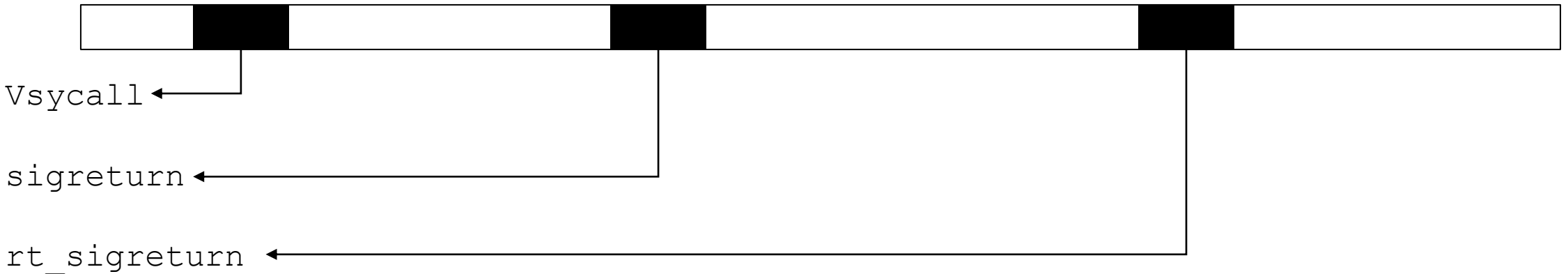
FIM UHK

Reakce na signály a přerušení volání



Knihovna linux–gate.so

- Knihovna – `linux-gate.so` se v systému nevyskytuje
  - Programy ji přesto vyžadují
  - Místo ní je na dané adrese **vsdo** (Virtual Dynamic Shared Object)
    - Obsahuje symboly pro virtuální systémová volání
    - Dva symboly pro návrat z obsluhy signálu



Proč existuje knihovna linux–gate.so

- Souvislost s restartem systémových volání po jejich přerušení a šestým parametrem volání
  - Procesory x86 poskytují málo registrů proto:
    - Parametr ukládáme na zásobník (v user mode)
    - Pro následné získání parametru využíváme registr **EBP**, tam před jeho použitím ukládáme obsah registru **ESP** (ukazatel zásobníku)
    - Při využití speciální instrukce pro přepínání režimů dochází k přepsání **EBP**
      - Nutno zajistit obnovení novým zkopírováním **ESP**
    - Virtuální knihovna tak vytváří vstupní bod do jádra
  - Umožňuje využívat tzv. virtuálních volání – např. pro pouhé čtení hodnoty

Systémové volání z pohledu jádra

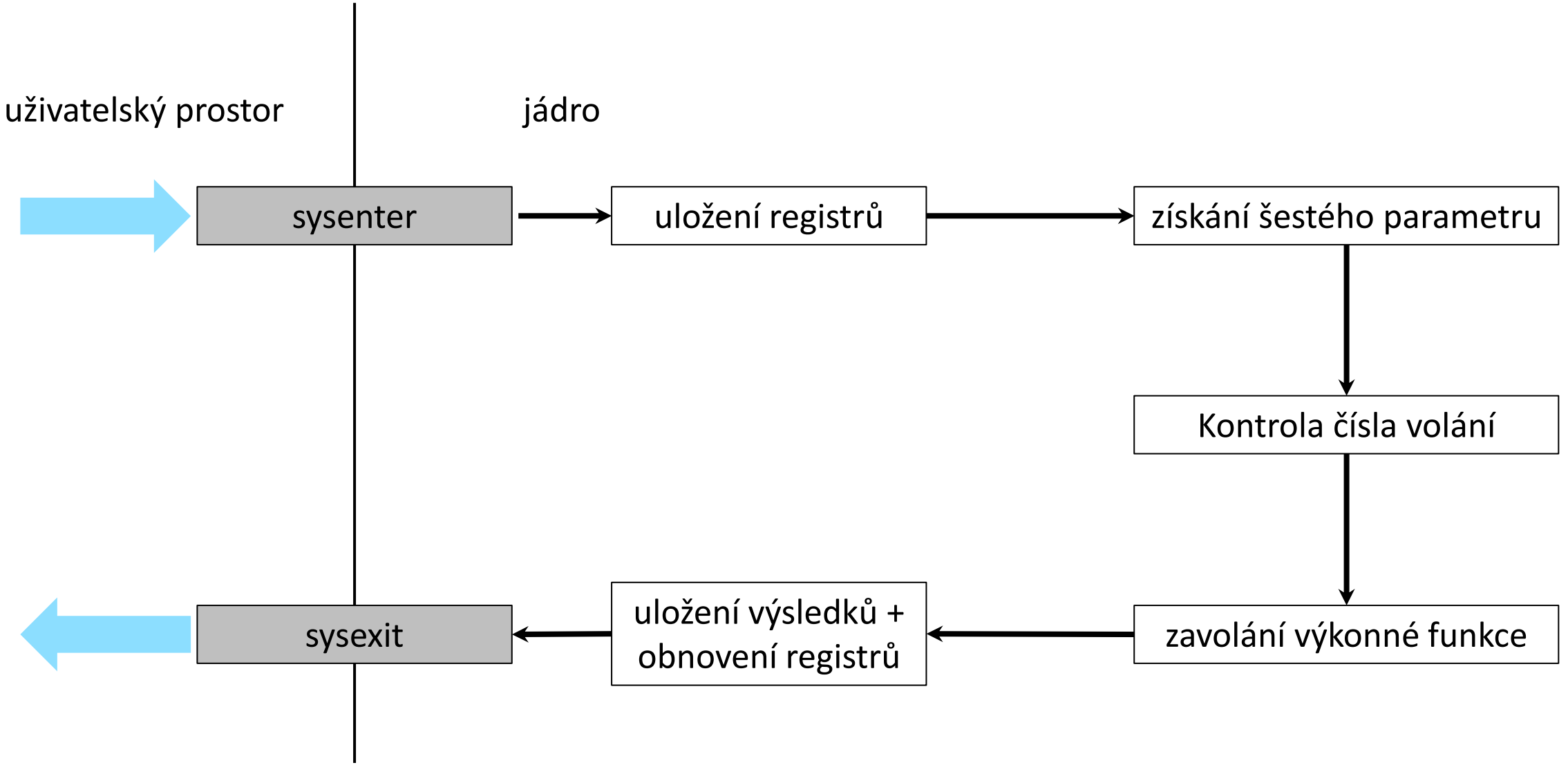
- Případy průchodu volání jádra
  - Běžné systémové volání
    - uživatelská aplikace využila určité systémové volání
  - Návrat z obsluhy signálu
    - návrat z obslužné rutiny signálu

Základní činnosti v rámci volání

- Pro kód, který implementuje systémová volání je charakteristické:
  - Společný
  - Nízkoúrovňový
  - Napsaný v assembleru
    - Pro zajištění kontroly jaké strojové instrukce se dostávají do jádra
  - Závislý na hardwarové architektuře

FIM UHK

Vykonávání systémového volání



Vykonávání systémového volání - začátek volání

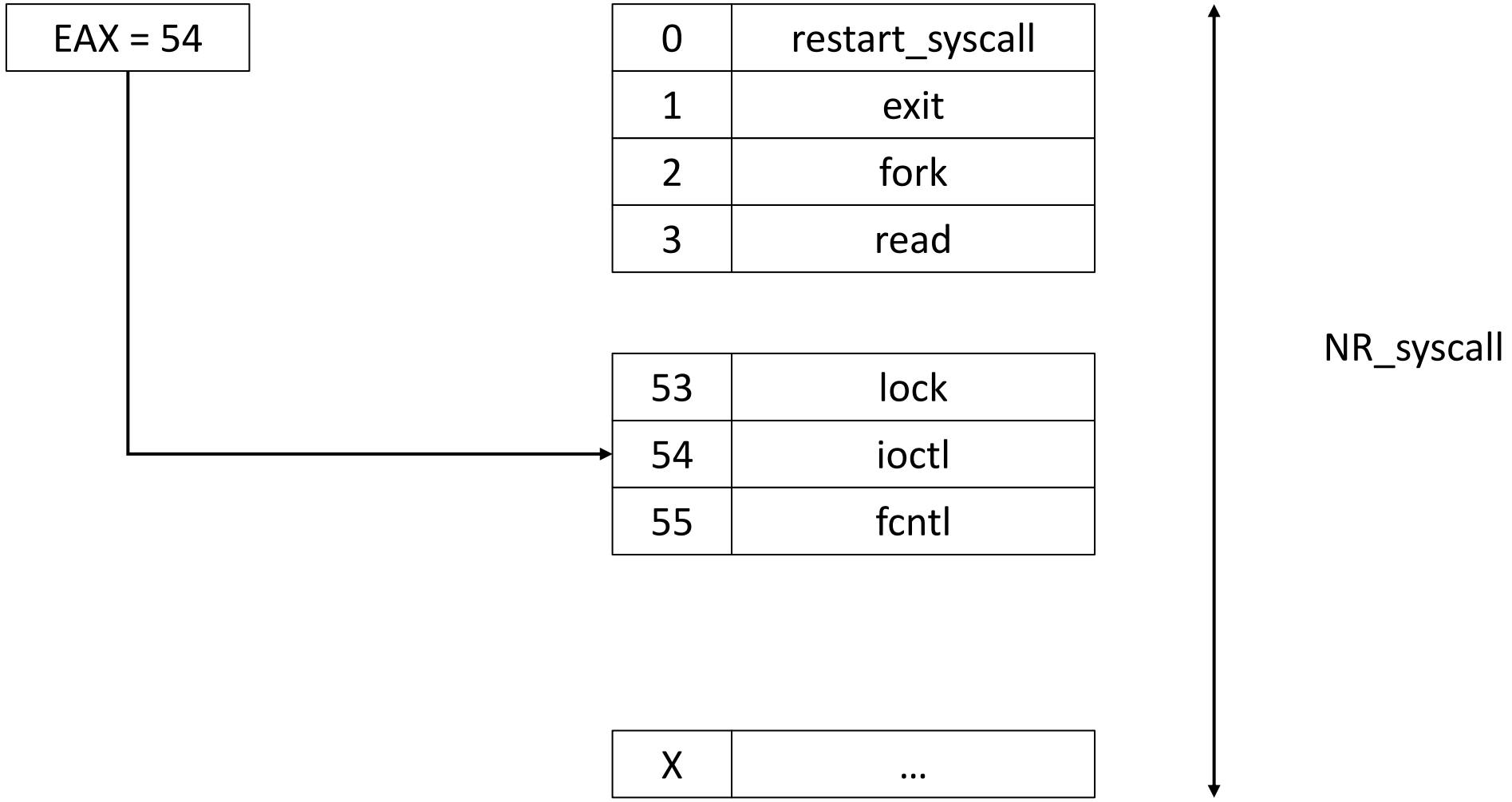
- Z pohledu jádra volání začíná ve vstupním bodě, kam se dostává přes instrukci `sysenter`.
  - Používá se pro něj označení `sysenter_entry`.
  - Provedením instrukce `sysenter` se:
    - adresa bodu `sysenter_entry` objeví v registru EIP (ukazatel instrukce)
    - adresa zásobníku jádra v registru EXP (ukazatel zásobníku)

Vykonávání systémového volání - obecný kód

- Od vstupního bodu začíná obecný kód
  - Ukládání registrů a návratové hodnoty na zásobník
    - Registr EBP obsahuje původní obsah registru ESP (zkopírován před vstupem do volání)
    - Přenos šestého parametru volání z uživatelského prostoru
      - Přenos z uživatelského zásobníku prostřednictvím registru EBP (použijeme adresu uloženou v registru a následně je vložena samotná hodnota parametru)
  - Kontrola čísla volání (registr FAX) s ohledem na podporovaný rozsah
  - Vyhledání a zavolání funkce odpovídající číslu volání z tabulky `sys_call_table`

FIM UHK

Vykonávání systémového volání - výběr výkonné funkce

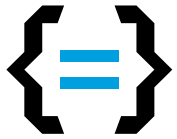


Vykonávání systémového volání - návrat z funkce

- Po návratu z funkce se:
  - Obsah registru EAX zkopíruje do zásobníku
  - Do registru EDX je umístěna návratová adresa
    - Při návratu do uživatelského prostoru se přesune do EIP
  - Do registru ECX je umístěn ukazatel na uživatelský zásobník ESP
- Po celou dobu manipulaci s registry jsou zakázána přerušení
- Kopírování do cílových registrů a přepnutí do uživatelského režimu řeší `sysexit`.
- Konec systémového volání

Vykonávání systémového volání - práce při návratu z funkce

- Těsně po návratu z výkonné funkce volání se kontroluje, zde před návratem do uživatelského prostoru není nutné realizovat další činnosti, které typicky jsou:
  - Přepnutí kontextu – naplánování jiné úlohy
  - Obsluha čekajících signálů
    - Po jejich zpracování je realizována test na potřebu realizace dalších činností.
- Není-li již nutné řešit další úkoly dochází k obnovení registrů ze zásobníku a za využití `sysexit` se vlákno vrací do uživatelského prostoru.



Univerzita Hradec Králové
Fakulta informatiky a managementu

Děkuji za pozornost

Další téma: Signály