Operacijski sistemi

Sistemski klici

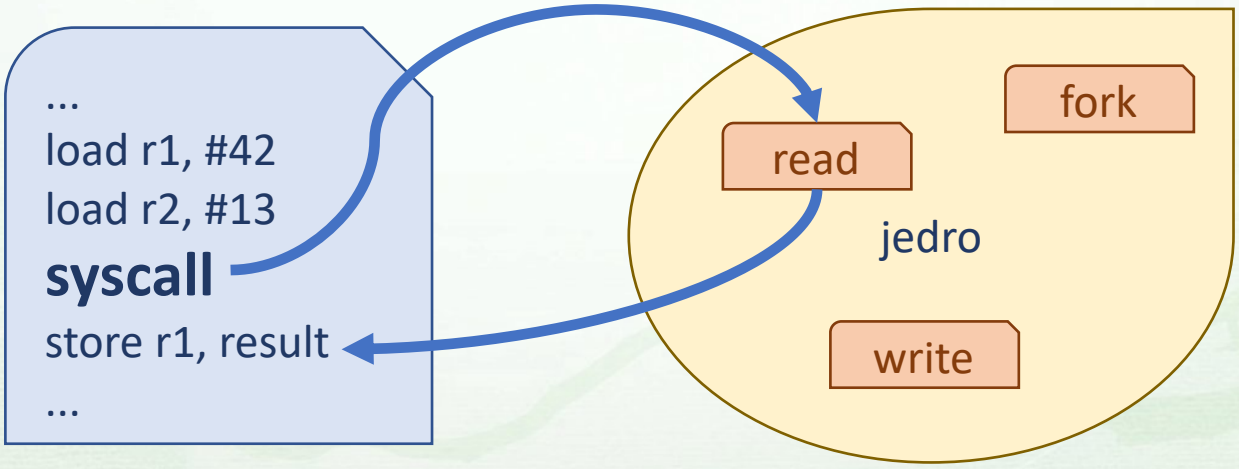
Vsebina

- Sistemski klici
  - Kaj je sistemski klic?
  - Kako ga izvedemo?
  - Ali je njegova uporaba varna?
- Ovojne funkcije
  - Kako čim bolj preprosto uporabljati?
- Standarizacija
  - So sistemski programi prenosljivi?

Sistemski klici

- Kaj je sistemski klic?
  - mehanizem preko katerega **uporabniški program** zahteva **jedrno storitev**
  - klic podprograma v jedru, ki implementira zahtevano storitev
    - uporabniški vidik: podobno navadnemu klicu funkcije

Nekateri OS podpirajo tudi klice tipa „jedro kliče jedro“ (reentrant syscalls)



Sistemski klici

- Sistemski klic oz. sistemska storitev
  - vsak klic ima svojo **številko**
    - npr. 1 … sistemski klic `exit`
    - za isti klic so (lahko) različne številke glede na OS in arhitekturo procesorja
  - prejme lahko tudi **argumente**
    - npr. `exit(32)`
  - prenos številke in argumentov
    - preko registrov
    - preko sklada

Primer: sistemski klic `exit()` v Linuxu

- št. 1: x86 32 bit, ia64, arm, alpha, m68k, mips o32, powerPC
- št. 60: x86 64 bit
- št. 58: mips

Sistemski klici

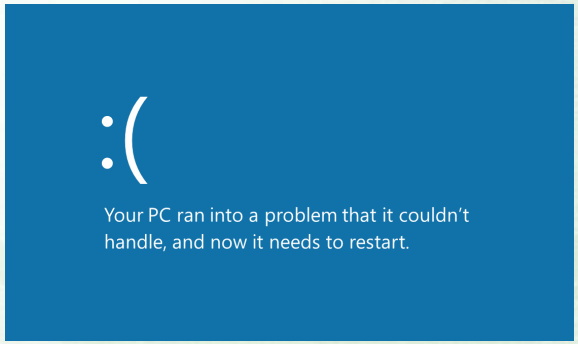
- Tabela rokovalnikov sistemskih klicev
  - *i*-ti element tabele je naslov rokovalnika
    - številka sistemskega klica je enaka indeksu rokovalnika v tabeli
    - vrstni red je fiksen in se v prihodnosti ne spreminja
  - primeri:
    - 1 … exit
    - 2 … fork
    - 3 … read
    - 4 … write
    - …

```
/* arch/x86/entry/syscalls/syscall_32.tbl */

ENTRY(sys_call_table)
  .long sys_restart_syscall
  .long sys_exit
  .long sys_fork
  .long sys_read
  .long sys_write
  .long sys_open
  .long sys_close
  .long sys_waitpid
  .long sys_creat
  .long sys_link
  .long sys_unlink
  .long sys_execve
  .long sys_chdir
  .long sys_time
  .long sys_mknod
  .long sys_chmod
     ...
```

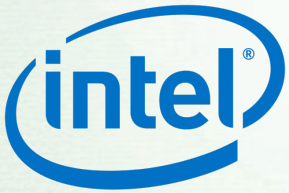
Sistemski klici

- Preklop nivoja zaščite procesorja
  - **jedro**, ki nudi storitev
    - teče v privilegiranem načinu
  - **klicoči program**, ki zahteva storitev
    - teče v zaščitenem načinu
    - direkten klic podprograma v jedru bi sprožil izjemo
  - potrebna je pomoč **strojne opreme**
    - za izvedbo preklopa v privilegirani način in klic podprograma
    - za preklop nazaj v zaščiteni način in vrnitev na mesto klica
  - Kako torej klicati funkcijo v jedru?



Sistemski klici

- Sistemski vmesnik – preklop v jedro
  - **namenski strojni ukaz**
    - za sistemski klic in tudi za vrnitev iz njega
    - procesor naredi preklop in pokliče nameščeni rokovalnik sistemskih klicev v jedru



Primer: arhitektura x86
- Intel: `sysenter` / `sysexit`
- AMD: `syscall` / `sysret`

Sistemski klici

- Sistemski vmesnik – preklop v jedro
  - **programska prekinitev**
    - programska: lahko jo sprožimo programsko
    - prekinitev: procesor izvede preklop v jedrni način
    - rokovalnik prekinitve: podprogram v jedru, ki skrbi za obdelavo prekinitve
    - procesor naredi preklop in pokliče nameščeni rokovalnik prekinitve v jedru
    - vsak OS ima svojo izbrano številko prekinitve

Primer: arhitektura x86
- BIOS: INT 0x10, 0x13, …
- DOS: INT 0x21
- Windows: INT 0x2E
- Linux: INT 0x80

Primer: arhitektura ARM
- SWI no

Sistemski klici

- *Sistemski vmesnik – preklop v jedro
  - ostali mehanizmi
  - **klicna vrata** (call gate)
    - x86 specifično, uporaba v OS/2
    - zahteva oddaljeni klic v drug segment, CALL FAR
    - zahtevna inicializacija segmentov
    - možno preklapljanje tudi v manj privilegiran nivo
  - **pomnilniška vrsta**
    - uporabno za veliko št. klicev
    - sistemske klice postavimo v vrsto
    - jedro periodično pregleduje vrsto

Sistemski klici

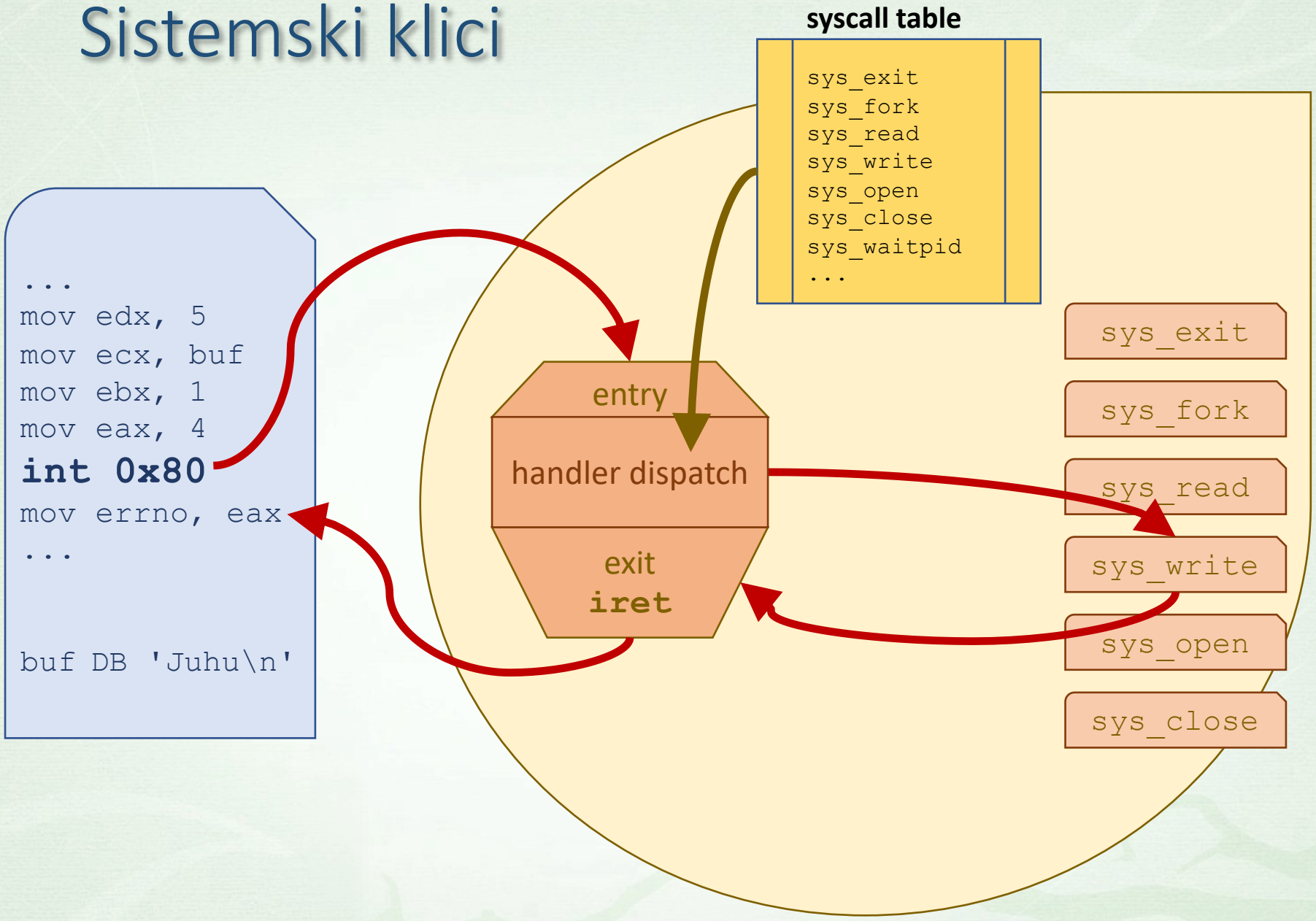
- Primer: arhitektura x86 in OS Linux
  - programska prekinitev
    - strojni ukaz `int 0x80`
  - št. sistemskega klica:
    - register `eax`
  - parametri klica:
    - registri `ebx, ecx, edx, esi, edi, ebp`
  - vračanje iz klica:
    - strojni ukaz `iret`

Sistemski klici

- Primeri sistemskih klicev v OS Linux, x86
  - `fork()`
    - sistemski klic št. 2 brez argumentov
    - eax=2, int 0x80
  - `exit(42)`
    - sistemski klic 1 z argumentom 42
    - eax=1, ebx=42, int 0x80
  - `write(1, "Juhu\n", 5)`
    - sistemski klic št. 4, deskriptor 1, niz dolžine 5
    - eax=4, ebx=1, ecx=@"Juhu", edx=5, int 0x80

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

Sistemski klici



Sistemski klici

- Izvedba sistemskega klica
  - priprava na sistemski klic
    - podajanje št. sistemskega klica in argumentov
  - vstop v jedro
    - preko sistemskega vmesnika, preklop v privilegiran način, proženje rokovalnika
  - izvedba rokovalnika sistemskega vmesnika
    - preverjanje št. klica in klic specifičnega rokovalnika
  - izvedba rokovalnika sistemskega klica
    - navaden klic rutine znotraj jedra
  - izstop iz jedra
    - preklop nazaj v uporabniški način

Sistemski klici

Zakaj sploh imeti sistemske klice? Zakaj ne bi uporabljali kar klicev funkcij?

- Primerjava: *sistemski klic* in *klic funkcije*
  - sistemski klic je počasnejši
    - zahteva preklop nivoja zaščite
  - izvedba samega rokovalnika klica je zahtevnejša
    - npr. preverjanje argumentov, uporaba tabele rokovalnikov, klic rokovalnika
  - podpora procesorja
    - običajen funkcijski klic
      - strojni ukaz `call`, `jmp` , `jal` ipd.
    - poseben mehanizem za sistemski klic
      - programska prekinitev `int`, `swi` ipd.

Sistemski klici

- OS kot programska knjižnica
  - če bi uporabljali le klice funkcij za izvedbo sistemskih klicev
  - vprašljiva varnost programov
    - programi bi morali delovati pravilno
    - programi bi se morali obnašati pošteno
  - luknja v takem sistemskem klicu
    - lahko sesuje celoten OS
    - lahko omogoči vdor v jedro in posledično vdor v katerikoli program
  - luknja v funkcijskem klicu (običajna knjižnica)
    - sesujemo le svoj program

Sistemski klici

- Varnost sistemskih klicev – zaščita sistema
  - sistemski klic je tudi mehanizem zaščite
    - za nadzor operacij, ki jih izvajajo procesi
    - ne more vsakdo delati vsega
    - avtorizacija uporabnikov/procesov/opravil
  - primer preverjanja
    - preverjanje številke klica
    - preverjanje podanih argumentov klica
    - preverjanje ali ima proces dovoljenje za uporabo zahtevanih virov
    - preverjanje ali ima proces dovoljenje za izvedbo zahtevane operacije

Ovojne funkcije sistemskih klicev

- Izziv: zahtevnost izvedbe sistemskega klica
  - neposredna izvedba je težavna
  - programiranje v zbirniku
  - potrebno rokovanje z registri in vstop v jedro

```
write(1, "Juhu\n", 5)
```

```
...
mov edx, 5
mov ecx, buf
mov ebx, 1
mov eax, 4
int 0x80
mov errno, eax
...

buf    DB 'Juhu\n'
```

Ovojne funkcije sistemskih klicev

- Ovojna funkcija (wrapper function)
  - njen namen je klic druge funkcije oz. (v našem primeru) izvedba sistemskega klica
    - priprava in preverjanje argumentov in vstop v jedro
  - v standardni knjižnici
    - stdlib.h, stdio.h, unisdt.h, itd.
  - primer: fork
    - knjižnica `unistd.h`
    - ovojna funkcija `fork()`
    - sistemski klic številka `2`
    - rokovalnik sis. klica `sys_fork()`

uporabniški prostor

jedrni prostor

Ovojne funkcije sistemskih klicev

- Operacijski sistem Windows NT
  - v dinamični knjižnici *ntdll.dll* (Native API)
    - se navadno ne uporablja neposredno
    - vsebuje ovojne funkcije sistemskih klicev
    - precej nedokumentirano
  - v dinamični knjižnici *kernel32.dll* (Windows API)
    - ovojne funkcije, ki kličejo ovojne funkcije iz *ntdll.dll*
    - osnovni API operacijskega sistema Windows
  - primer: CreateProcess
    - knjižnica `processthreadsapi.h`
    - ovojna funkcija `CreateProcessA()`

Windows

Ovojne funkcije sistemskih klicev

- Izvedba sistemskega klica
  - **neposredno**
    - nastavitev registrov in vstop v jedro v zbirniku
  - preko **specifične ovojne funkcije**
    - uporaba predpripravljenih ovojnih funkcij iz knjižnice
  - preko **splošne ovojne funkcije** `syscall()`

```
syscall(__NR_write, 1, "Pozdravljen, svet!\n", 19);
```

Kje najdem številke sistemskih klicev?
- V /usr/include/asm/unistd.h

  - **posredno** preko **ostalih funkcij**
    - npr. `printf()`

Ovojne funkcije sistemskih klicev

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTime() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Standardi

- Aplikacijski programski vmesnik
  - API – application programming interface
  - vmesnik za uporabo programskih knjižnic
  - temelji na simbolični predstavitvi
    - npr. imena funkcij, razredov itd.

- Aplikacijski dvojiški vmesnik
  - ABI – application binary interface
  - temelji na številski predstavitvi
    - npr. številska oznaka funkcije, prenos argumentov v registrih

Standardi

Portable Operating System Interface
X v POSIX predstavlja Unix

- POSIX – standard IEEE 1003
  - prenosljivi vmesnik operacijskega sistema
    - definira programski vmesnik med aplikacijami in OS
    - predpisuje funkcije, ukazno lupino, okoljske spremenljivke, orodja lupine, datotečno hierarhijo, knjižnico za nitenje, …
  - vzrok standardizacije
    - obstaja veliko različnih UNIXov
    - standard omogoča prenosljivost programov
    - POSIX aplikacije le ponovno prevedemo (teorija)

Glej tudi: https://pubs.opengroup.org/onlinepubs/9699919799/nfindex.html

Standardi

- Razvoj POSIX standarda
  - standardi do leta 1997
    - POSIX.1 … procesi, datoteke, pomnilnik, cevi, signali, standardna C knjižnica, …
    - POSIX.1b … razvrščanje, sporočila, semaforji, ključavnice, …
    - POSIX.1c … nitkanje pthreads
    - POSIX.2 … ukazi v okolju
  - sedaj POSIX.1, 2013 edition
    - praktično enako kot Single UNIX Specification

Standardi

Single UNIX Specification

- Single UNIX Specification
  - Open Group je lastnik blagovne znamke UNIX™
    - od leta 1994
  - standard SUS je razvila delovna skupina Austin Group: IEEE + Open Group
    - izhaja iz IEEE POSIX, vendar IEEE ni lastnik UNIX™
    - Open Group pa je lastnik UNIX™

Standardi

Single UNIX Specification

- Skladnost s SUS standardom
  - **certificirani Unix** sistemi
    - možna uporaba blagovne znamke UNIX™
    - AIX, EulerOS (Linux-based), HP-UX, macOS, …
  - **ne-certificirani Unix podobni** sistemi
    - pogosto podpirajo standard skorajda v celoti
    - vendar niso certificirani
      - drago certificiranje, nestrinjanje z posameznimi delčki
    - Linux, BSDs
  - **ostali ne-certificirani** sistemi
    - Windows Subsystem for Linux (delno)
    - Cygwin