Pokročilé operační systémy

Jednoúlohový operační systém

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Aktuální stav

Co máme

- zavaděč jádra operačního systému
- zapnutou podporu chráněného 32bitového režimu
- základní obsluhu periferií
  - textový výstup
  - vstup z klávesnice
  - práce s pevným diskem
- souborový systém
- práci s binárními soubory

Aktuální stav

Co nemáme

- obsluhu systémových volání
- standardní knihovnu
- podporu pro spouštění programů
- uživatelské rozhraní
- systémové nástroje
- přepínání mezi privilegovaným a neprivilegovaným kódem

...a zatím nebudeme mít

- ochranu paměti
- multitasking

Obsluha systémových volání

- standardní řešení v podobě přerušení
- přidán handler pro přerušení (0x80)
- chování kopíruje rozhraní Linuxu (možné spouštět programy na obou platformách!)
- narozdíl od běžné obsluhy přerušení je nutné povolit vyvolání přerušení z uživatelského prostoru (mírná změna v IDT entry, viz `irq.c`)
- obsluha přerušení má jinou volací konvenci (než cdecl)
- konverze v `isr_syscall` (`irq-asm.h`) a dále obsluha v `syscalls.c`

Standardní knihovna

- funkce obalující systémovými voláním
- část dle specifikace jazyka C, část dle POSIX
- staticky linkovaná knihovna
- archiv objektových souborů + vygenerovaný index se symboly

```
libc_pops.a: libc_pops.o string.o syscalls.o start.o
        ar -rc libc_pops.a libc_pops.o syscalls.o start.o string.o
        ranlib libc_pops.a
```

- při linkování programů je nutné doplnit cesty k naší implementaci standardní knihovny
  `-L../libc/ -I../libc/include -lc_pops`
- jako vstupní bod se bere funkce `_start`, která zajistí
  - předání argumentů funkci `main`,
  - korektní ukončení systémovým voláním `exit`,
  - viz `external/libc/src/start.asm`.

Podpora pro spouštění programů

- téměř hotová
  - máme prostředky umožňující číst soubory (viz `kmain.c`, varianta 1)
  - umíme přečíst formát ELF a jednotlivé části nahrát do paměti (viz `kmain.c`, varianta 2)
  - zbývá zavolat vstupní bod programu
- předpokládáme, že je program nahráván od 0x00200000

Uživatelské rozhraní OS

- plyne přímo z toho, že máme podporu pro spouštění programů
- ovládání systému může řešit vyčleněný program *command shell* (viz `external/coreutils/cosh.c`)
- command shell je spuštěný
  - po inicializaci systému
  - po skončení každého procesu (systémové volání `exit`)
- práce se systémem formou systémových volání a spouštěním dalších programů/nástrojů (systémové volání `execve`)
- systémové nástroje (např. pro práci se soubory) je možné řešit jako součást command shellu nebo jako samostatné programy

Poznámka

- v tomto bodě máme OS typu MS-DOS nebo CP/M

Přepínání režimu procesoru

Dílčí problémy

1. podpora ring3
2. přepnutí z ring3 do ring0
3. přepnutí z ring0 do ring3
4. nastavení zásobníků
5. rozložení paměti

Podpora ring3

- vyžaduje přidání dvou segmentů do GDT, viz `pm_boot.asm`
- kódový a datový segment s DPL=3 (v rozsahu celého paměťového prostoru)
- ochranu paměti budeme řešit na úrovni stránkování (segmentace má omezenou podporu v překladači)

Přepínání režimu procesoru

Přepnutí z ring3 do ring0

- součástí záznamu v IDT je selektor kódoveho segmentu
- DPL aktuálního kódového segmentu udává CPL
- v obsluze přerušení získáme přepnutí režimu procesoru téměř automaticky
- aktuální pozice CS:EIP a EF uloženy na zásobník (jaký?)

Přepnutí z ring0 do ring3

- trochu ,,tricky" řešení
- jsme v ring0 a přesvědčíme procesor, že v ring3 již byl a chceme se do něj vrátit
- nastavíme obsah zásobníku, jak kdyby došlo k přerušení, a provedeme instrukci `iret`

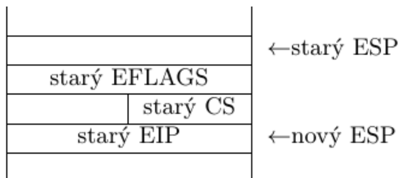
Nastavení zásobníků

- pro každý režim procesoru (ring) potřebujeme jeden zásobník (jádro se nemůže spoléhat na korektní nastavení ESP)
- při přepnutí režimu procesoru nutné změnit zásobník
- k tomu se používá Task State Segment
  - speciální segment popsaný v tabulce GDT
  - selektor uložený v registru TR
  - používal se pro hardwarové přepínání úloh
  - umožňuje definovat adresy zásobníků pro jednotlivé režimy procesoru
  - viz `tasks/tasks-asm.asm`
  - ostatní segmenty nutné řešit ve vlastní režii

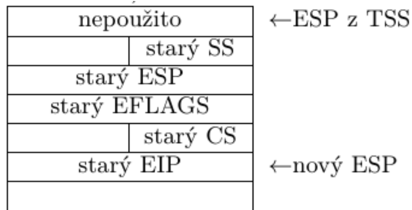
Nastavení zásobníků

- obsah zásobníku se po přerušení liší podle toho, jestli došlo ke změně režimu procesoru nebo ne

- nedošlo ke změně



došlo ke změně



- viz `tasks/tasks-asm.asm`

Rozdělení paměti

- `0x00050000` – zásobník jádra (roste dolů)
- `0x00070000` – začátek kódu jádra
- `0x00080000` – TSS
- `0x000a0000` – záležitosti BIOSu
- `0x00100000` – halda jádra
- `0x00200000` – kód a data programu
- ... break ... (podle velikosti programu, viz systémové volání `brk`)
- `0x003f0000` – zásobník uživatelského prostoru
- prostor pro argumenty programu (viz systémové volání `execve`)