



10. NOVA System call



NOVA – Systémová volání a správa paměti

Domácí příprava

Zadání úlohy

Nápověda

Ladění

Užitečné příkazy gdb

Monitorovací konzole Qemu

Video

Příklad

Materiály

NOVA – Systémová volání a správa paměti

Na tomto cvičení se seznámíte s jádrem miniaturního OS NOVA a implementujete do něj systémové volání `nbrk`. **NOVA** je mikrohypervizor původně vyvíjený na [Drážďanské univerzitě](#), později ve firmě Intel a nyní firmami [GENODE labs](#), [BedRock Systems](#) a [Cyberus Technology](#). Na cvičeních však nebudete pracovat s kompletní verzí jádra NOVA, ale se zjednodušenou verzí pro výuku, která má pouze 2 tisíce řádek kódu.

Domácí příprava

Pro toto cvičení se vám bude hodit:

- aspoň [trochu rozumět kódu v C++](#) (viz také [7. přednáška](#)) a [inline assembleru](#) (viz [8. cvičení](#)),
- vědět, co to jsou [systémová volání](#) a proč/k čemu se používají (viz přednášky),
- vědět, co dělá systémové volání `brk` v Linuxu (budete implementovat podobnou, ale ne stejnou, funkcionalitu),
- vědět, jak CPU překládá virtuální adresy na fyzické (i386 používá dvoustupňové mapování 10-10-12), viz přednášky OSY, APO, [tuto přednášku na YouTube](#), případně další,
- projít si slidy [NOVA Introduction](#),
- stáhnout si, rozbalit a ideálně i vyzkoušet (`make run`) [operační systém NOVA](#)
 - na vlastním PC (Debian/Ubuntu) budete potřeboval balíčky:
 - `libc6-dev-i386`
 - `qemu-system-i386` nebo `qemu-system-x86`

- přečíst hlavní Makefile
- přečíst funkci `Ec::syscall_handler`
- přečíst funkci `Ptab::insert_mapping` (byla na přednáškách)
- podívat se na třídu `Kalloc`
- při práci přes vzdálený přístup použijte

```
ssh -X «login»@postel.felk.cvut.cz
```

přepínač `-X` umožní spouštět i grafické aplikace (QEMU)

- popis instrukce `sysenter` a `sysexit`,
- rozumět rozdílu mezi **uživatelským prostorem a prostorem jádra** (viz přednášky).

Zadání úlohy

V jádře OS Nova implementujte systémové volání `nbrk` s prototypem:

```
void *nbrk(void *address)
```

Toto systémové volání nastaví konec datového segmentu v adresním prostoru procesu (tzv. *program break* nebo jen *break*) na adresu danou parametrem `address`. Tím se zvětší nebo zmenší množství alokované paměti, které může program využívat ke svému běhu. Break je první adresa za koncem namapovaného datového segmentu.

Vaše řešení by mělo splňovat následující požadavky:

- Po úspěšném návratu ze systémového volání je break nastaven na hodnotu `address`. To znamená, že uživatelský program může používat paměť od adresy `0x1000` do adresy o jedna menší než `address`.
- Přístup na stránky začínající na adrese vyšší či rovné `address` nebude programu dovolen.
- Break nesmí být nastaven na nižší hodnotu, než je jeho hodnota při spuštění programu. Tím by se program připravil o část svého kódu nebo dat.
- Break nesmí být nastaven na vyšší hodnotu než `0xBFFFFFF00`. Tím by aplikace mohla přepsat svůj zásobník nebo dokonce jádro, které je namapováno od adresy `0xC0000000` výš.
- Při úspěšném dokončení je vrácena původní hodnota break před vykonáním systémového volání. Při chybě je vrácena hodnota `0`.
- Pokud je `address` rovno `NULL` (`0`), nejedná se o chybu a hodnota break se nemění. Toto volání slouží pouze ke zjištění aktuální hodnoty break.

- Při žádném volání `nbrk` nesmí dojít k „pádu“ systému.
- *ABI* systémového volání bude následující:
 - vstup: `AL=3` , `ESI=address` ,
 - výstup: `EAX=návratová hodnota` .
- Nově alokovaná paměť bude inicializována na nulu.
- Při snižování hodnoty `break` bude nepřístupná paměť dealokována (a odmapována), aby mohla být opět alokována později.
- Při chybě alokace paměti v průběhu systémového volání nebude adresní prostor uživatelské aplikace změněn a částečně alokovaná paměť bude dealokována.
- Při kompilaci nevypisuje kompilátor žádná varování.

Odevzdává se archiv se souborem `ec_syscall.cc` obsahující vaši implementaci, ideálně vytvořený pomocí:

```
make hw10
```

Nápověda

- Operační systém NOVA s testovací aplikací `user/hello.c` můžete nabootovat buď na fyzickém počítači pomocí zavaděče podporujícího specifikaci [multiboot](#) (např. [GRUB 2](#)), ale pravděpodobně bude efektivnější ho pouštět jako virtuální stroj například v emulátoru Qemu. Pro to stačí spustit příkaz:

```
make run
```

- Pokud se vám při spuštění qemu zobrazuje hláška:

```
Unable to init server: Could not connect: Connection refused
gtk initialization failed
```

znamená to, že nemůžete spouštět grafické aplikace. V tom případě můžete spustit qemu bez podpory grafického rozhraní:

```
qemu-system-i386 -nographic -kernel kern/build/hypervisor -initrd user/he
```



- Pokud úlohu vyvíjíte na Windows ve WSL, je několik možností jak OS NOVA spustit. Nejjednodušší varianta:

1. Qemu **nainstalujte do Windows** a přidejte jeho instalační složku (typicky `C:\Program Files\qemu`) do environment proměnné `PATH` – třeba přes GUI (Start Menu -> vyhledejte `PATH` -> vyberte Edit the system environment variables -> dole tlačítko Environment variables, a pak v části System variables upravte `Path`. Pokud vám hledání `PATH` nedá žádné relevantní výsledky (pravděpodobně kvůli neanglické lokalizaci Windows), vyhledejte místo `PATH` přímo program `SystemPropertiesAdvanced`. Poté zavřete a znovu otevřete WSL terminal, aby se změna v environmentu promítla (shell si drží prostředí, které dostal při spuštění).
2. Otevřete `Makefile` ve složce OS NOVA, a všude kde se volá příkaz `qemu-system-i386` doplňte `.exe` (tedy `qemu-system-i386.exe`). Tím se bude používat Qemu nativně ve Windows místo ve WSL (i když `make` pustíte ve WSL).
3. Otestujte puštěním příkazu `make run`. Výstup seriové linky by se měl zobrazovat v terminálu, a zároveň by se mělo otevřít GUI okno Qemu. Virtuální stroj můžete ukončit buď zavřením Qemu okna, nebo stisknutím `Ctrl-c` v původním WSL terminálu.

- Při zvětšování hodnoty „program break“ musíte v jádře alokovat paměť a namapovat ji do adresního prostoru uživatelské aplikace modifikováním stránkovacích tabulek. Inspirací vám může být funkce `Ec::root_setup()`, která připravuje paměť pro spouštěný program. Funkce čte hlavičky z binárky aplikace, které si můžete zobrazit příkazem `readelf --program-headers hello`.

Na konci funkce nastaví proměnné `Ec::break_min` a `Ec::break_current`, které pravděpodobně budete potřebovat ve své implementaci.

- Při snižování hodnoty „program break“ naopak musíte mapování zrušit a paměť dealokovat.
- Při startu uživatelského programu (např. `hello`) je jeho paměťová mapa následující (trochu zjednodušeno):

Adresy	Obsah
<code>0x00001000 – 0xFFFF000-1</code>	kód programu, viz <code>.text</code> ve výstupu příkazu <code>readelf --sections hello</code>
<code>0xFFFF000 – 0xYYYYY000-1</code>	data programu, viz <code>.data</code> a <code>.bss</code> ve výstupu příkazu <code>readelf --sections hello</code>
<code>0xYYYYY000</code>	program break
<code>0xBFFFF000 – 0xC0000000-1</code>	zásobník (4 kB); počáteční hodnota registru <code>ESP</code> je <code>0xC0000000</code>

- Ke kódu jádra NOVA není žádná dokumentace, ale části, které budete potřebovat, jsou nejsou složité, a měli byste být schopni jim porozumět na základě čtení kódu (a komentářů). Pokud však i přes vaši snahu něčemu nerozumíte, klidně se [zeptejte](#).

Ladění

Při vývoji operačního systému nelze používat debugger tak jednoduše, jak jste zvyklí při vývoji aplikací. Ladit váš kód můžete přidáváním příkazů `printf()` na potřebná místa v kódu. Pokud vám to nestačí můžete použít parametr `-gdb` (případně zkratku `-s`) emulátoru Qemu.

Abychom vám ladění usnadnili, v hlavním `Makefile` jsou připravena pravidla jak pro spouštění Qemu se zmiňovanými parametry, tak pro spouštění debuggeru `gdb` tak, aby šel ladit kód běžící v Qemu:

- V jednom okně spustíte příkaz

```
make rd
```

který spustí Qemu, které po startu počká na připojení debuggeru.

- V druhém okně pak spustíte

```
make du
```

(pokud chcete ladit uživatelský program (nápř. ``hello'`)) nebo

```
make dk
```

pokud chcete ladit jádro.

Můžete v `Makefile` změnit argument příkazu `break` tak, aby se vykonávání programu zastavilo na funkci (či řádku), který potřebujete odladit.

Užitečné příkazy gdb

Příkazy se většinou dají zkrátit na první znak.

- Běh programu: `next (n)`, `step (s)`, `continue (c)`

- Výpis proměnných: `print` (např. `p initialized_var` nebo hexadecimálně: `p/x initialized_var`)
- Výpis paměti: `x` (např: `x/16 0x2000` vypíše 16 slov od adresy `0x2000`, `x/16 &initialized_var` vypíše 16 slov začínající na adrese proměnné `initialized_var` .
- Informace: `info registers` vypíše hodnoty registrů, `info locals` vypíše hodnoty lokálních proměnných atd.
- Zobrazení: `layout src` zobrazí zdrojový kód i příkazové okno, `Ctrl-L` překreslí obrazovku, `tui disable` vypne zobrazování zdrojového kódu.

Monitorovací konzole Qemu

Pro ladění můžete používat i monitorovací konzoli Qemu, která umožňuje zobrazit stav emulovaného CPU.

- Do konzole se přepnete stiskem `Ctrl-Alt-2` v grafickém okně (nebo stiskem `Ctrl-a c` pokud spouštíte `qemu` s přepínačem `-nographic`)
- Příkazy `info mem` nebo `info tlb` vypíší informace o mapování virtuálních adres na fyzické (tj. stránkovací tabulku).
- `info registers` vypíše hodnoty registrů

Video

B4B35OSY: Ladění úloh s jádrem NOVA



Příklad

- Předpokládejme, že `break_start` je na začátku Vašeho procesu nastaven na `0x5000`.
- Při volání funkce `break(0x9000)` dojde k alokovaní stránek `0x5000-0x5FFF`, `0x6000-0x6FFF`, `0x7000-0x7FFF` a `0x8000-0x8FFF`.
- Při dalším volání `break(0x7555)` dojde k uvolnění a odmapování stránky `0x8000-0x8FFF`.
- Při dalším volání `break(0x7666)` se neprovádí žádná alokace, ale vynuluje se paměť `0x7555-0x7665`.
- Při dalším volání `break(0xA000)` se alokuje nová stránka pro `0x8000-0x8FFF` a `0x9000-0x9FFF`, a vynuluje se paměť `0x7666-0x7FFF` (předpokládáme, že nově alokované stránky jsou už vynulované).

Materiály

- [Testovací programy](#)
- [Zdrojové kódy OS NOVA](#)