11. NOVA Malloc

Uživatelský paměťový alokátor pro OS NOVA

Na tomto cvičení si zkusíte implementovat alokátor dynamické paměti, který bude využívat systémové volání `nbrk` z minulého cvičení.

Domácí příprava

Pro toto cvičení budete potřebovat:

- hotovou minulou úlohu
- zkontrolovat si, jestli zdrojové kódy NOVA nebyly od minulé úlohy změněny a případně si stáhnout aktuální verzi
- vědět, co je alokátor dynamické paměti (memory allocator) a způsoby implementace, viz:
  - 7. přednášku
  - http://arjunsreedharan.org/post/148675821737/write-a-simple-memory-allocator
- implementovat spojové seznamy
- rozumět programování v C/C++
- vědět, co dělají funkce malloc a free

Zadání úlohy

Implementujte jednoduchý alokátor paměti pro uživatelský prostor OS NOVA. Váš alokátor bude implementovat funkce `my_malloc` a `my_free` s následujícími prototypy:

```
void *my_malloc(unsigned long size);
int my_free(void *address);
```

Funkce implementujte v souboru `user/mem_alloc.c` (C) nebo `user/mem_alloc.cc` (C++) a na začátku souboru mějte *#include "mem_alloc.h"*. Makefile v adresáři user je na tento

soubor připraven a zkompiluje váš alokátor do knihovny `libc.a`. Tato knihovna se linkuje k programům jako `malloc_test.c`, které můžete použít pro testování vašeho alokátoru.

Od alokátoru budou očekávány následující vlastnosti:

- Bude schopen alokovat a uvolnit paměť.
- Při kompilaci nejsou generována žádná varování.
- Alokátor bude možné používat po přilinkování k aplikaci `user/malloc_test.c` ze zdrojových kódů OS NOVA.
- `my_malloc` alokuje paměť velikosti `size` a vrací adresu začátku alokované paměti. Pokud paměť požadované velikosti nelze alokovat, `my_malloc` vrací `0`.
- `my_free` uvolní paměť alokovanou funkcí `my_malloc`. Parametr address je hodnota dříve vrácená funkcí `my_malloc`. Pokud je paměť úspěšně uvolněna, funkce vrátí `0`, v opačném případě je vrácen nenulový kód chyby, který si můžete nadefinovat jak chcete.
- Pokud je `my_free` zavolána na již uvolněnou paměť nebo na paměť, která nebyla alokována voláním `my_malloc`, jedná se o chybu a funkce by ji měla signalizovat návratovou hodnotou. Můžete předpokládat, že testovací program používající váš alokátor modifikuje pouze paměť (na heapu) vrácenou funkcí `my_malloc`.
- Bude používat systémové volání `nbrk` pro získání paměti pro alokaci.
- Paměťová režie alokátoru pro 16bytové alokace bude maximálně 100%. Tedy pokud např. zavolám 1024krát `my_malloc(16)`, alokátor si od jádra vyžádá voláním `nbrk` maximálně 2×1024×16 = 32 KiB paměti.
- Paměť uvolněná voláním `my_free` půjde znovu alokovat.
- Paměť alokovaná funkcí `my_malloc` bude přístupná minimálně do doby zavolání odpovídajícího `my_free`.
- Žádná část paměti alokované funkcí `my_malloc` nebude znovu alokována dříve, než bude zavoláno odpovídající `my_free`.

Nepovinně (pro plný počet bodů) budou navíc vyžadovány následující vlastnosti:

- Po uvolnění menších sousedních bloků bude možné ve stejné oblasti alokovat jeden velký souvislý blok.

Co se odevzdává: Archiv obsahující vaši implementaci v souboru `user/mem_alloc.c` nebo `user/mem_alloc.cc` a soubor `kern/src/ec_syscall.cc` z minulého cvičení. Archiv můžete vytvořit následujícím příkazem spuštěným z kořenového adresáře NOVY:

```
make hw11
```

Ladění

- Úlohu můžete ladit pomocí Qemu a GDB, jak je popsáno u minulé úlohy.
- Můžete také vyvíjet a ladit alokátor pod Linuxem a až když bude fungovat tam, budete ho překládat a testovat s Novou. V ukázkovém `mem_alloc.c` je totiž i kód, který implementuje volání `nbrk` pomocí Linuxových `brk` a `sbrk`. Svůj alokátor a testovací program můžete přeložit pro Linux například následujícím příkazem:

```
gcc -g -Og -Wall -I./lib -o malloc_test.linux malloc_test.c mem_alloc.c
```

- Stejné programy, kterými testuje BRUTE vaší implementaci si můžete spustit i lokálně:

  1. Stáhněte si objektové soubory testovacích programů.
  2. Rozbalte je do adresáře `nova/user`:

     ```
     tar xf malloc_tests.tgz
     ```

  3. Slinkujte je se svou implementací (stačí použít `make` se jménem cílové binárky):

     ```
     $ make malloc-3m3f malloc-basic ...
     g++ -m32 -g -nostdlib -fomit-frame-pointer -fno-stack-protector -no
     g++ -m32 -g -nostdlib -fomit-frame-pointer -fno-stack-protector -no
     ...
     ```

  4. Spusťte výsledek:

     ```
     qemu-system-i386 -nographic -kernel ../kern/build/hypervisor -initr
     ```

     Můžete přidat i parametry `-s -S` a ladit kód v debuggeru.