Zápočtový program

# Garbage collector

Vypracoval: Ondřej Tichavský
Školní rok: 2021/2022
Seminář: Zápočtový program

# Obsah

# 1. Úvod

V tomto dokumentu provedeme popis mého zápočtového programu Garbage Collector.

## 1.1 Zadání

Zadáním je cpp knihovna, jejímž úkolem je provádět správu nad pamětí, přesněji pamětí na haldě.

Knihovna musí nabídnout nějaký prostředek, jakým lze uložit určitou informaci na haldu, a prostředek pro práci s uloženou informací. Aby šlo k informaci nějak přistupovat, upravovat jej...

Knihovna dále musí zaručovat správné uklízení paměti, na které již uživatel stratil odkaz.

# 2. Návod k použití

Přiložený kód obsahuje již konkrétní soubor example.cpp, který již využívá Garbage-Collector. Zároveň obsahuje příklady jeho užití.

# 3. Uživatelská část

## 3.1 Použití

Knihovna funguje od c++ 20.

K celé knihovně lze přistupovat z jediného hlavičkového souboru "GarbageCollector.h". Po inklůdování daného souboru je postaráno o vytvoření statického objektu GarbageCollector gctor.

Samotný garbage collector pracuje pouze s pamětí ve formátu libovolné třídy, nebo struktury, která dědí z pomocné třídy Context. Zároveň k danému objektu přistupuje pomocí třídy GCObject<T>, kde T je typ naší třídy dědící z Context.

### GCObject<T>

GCObject sadu konstruktorů, kdy vždy musí být předána alespoň hodnota odkazu umístění, kde se nachází, buď to je to ukazatel na objekt dědící z Context, nebo je to ukazatel na statický objekt typu Context. Pro vytvoření nového GCObjectu (respektive ukazatele na náš objekt typu T) se používají dvě metody. new_obj<T, Args...>(args...), která mimo to že vrátí GCObject, také vytvoří nový Context, na který vrácený objekt ukazuje.

Druhou metodou je make_ptr(GCObject<T>), který vrátí GCObject odkazující na stejný Context, jako GCObject v argumentu.

K našemu objektu lze přistupovat pomocí metod const T & GCObject<T>::view() čistě pro čtení a T * GCObject<T>::get() pro cokoli dalšího.

Bohužel, pokud chceme předat GCObjekt do parametru funkce. Nelze to provést snadno (a je pro to dobrý důvod, je potřeba získat informaci, v jakém objektu by se nacházel). Do parametru se předá referencí, a až v těle funkce lze vytvořit nový ukazatel na Context.

Vracet GCObjekt můžeme také pomocí reference.

### Invoker

Jedna ze základních funkcí obecného garbage collectoru je uklízení paměti, na kterou se již uživatel nemůže odkazovat. Tuto funkci lze provést voláním funkce gctor.updateMemory(). Obvykle, ovšem, uživatel není nucen starat se o volání uvolňování paměti. Tuto funkci v mém garbage collectoru zastává část knihovny tzv. Invoker. Tato část programu jednou za čas uklidí nepotřebnou paměť.

Moje implementované 2 Invokery pracují na svém vlastním vlákně. Pokud dojde k updatu paměti, ostatní vlákna jsou nucena počkat se svou prací, pokud vyžadují zápis.

Invoker je nutné spusutit pomocí gctor.startNewinvoker<GCDumbInvoker>().

Při práci s invokerem je možné za běhu programu zavolat metodu gc_stop_invoker(). Pokud tato metoda není zavolána během programu, s koncem hlavního vlákna, které bylo zavoláno na main() metodě, se automaticky vlákno Invokeru zastaví zavoláním destruktoru.

**example.cpp**

Pro přehlednost jsem vytvořil soubor example.cpp, který obsahuje řadu příkladů použití knihovny.

Pro lepší přehlednost jsem nechal v programu vypisování běhových hlášek, alespoň v případě, že je speciální příznak gc_print_messages na true.

Při výpisu je každý GCContext označen číslem od 1, kdy 1 označuje statický GCContext.

**problém**

Bohužel způsob implementace GC nebyl zcela ideální, nadruhou stranu jsem jsem nenašel lepší způsob, nicméně, pokud člověk pracuje s objektem uvnitř objektu, musí volat metody make ptr, nebo new obj ze stejného místa, kde je umístěn:

o1.get()->o2 = o1.get()->make_ptr(o2);

Špatně je: o1.get()->o2 = make_ptr(o2); Naštěstí, v případě, že by se uživatel spletl, chyba je ošetřena vyjímkou.

# 4. Informace o programu

## 4.1 GCContext

GCContext je bází naší informace, kterou chceme uložit na haldu. Obsahuje seznam ukazatelů na všechny další objekty GCContext. Všechny je skladuje v unordered_multiset.

Samotná tato třída obsahuje metody make_ptr<...>() a new_obj<...>(), které mají být použity v případě, že chci vytvořit nový objekt, nebo ukazatel uvnitř této třídy.

Mimo třídu GCContext jsou zde také statické funkce make_ptr<...>() a new_obj<...>(), které mají být volány, pokud definuji GCObject mimo potomka GCContext.

## 4.2 GCObject

Má představovat ukazatel, respektive jakoby orientovanou hranu, vedoucí mezi 2 GCContexty. Tyto hrany se pak ukládají u každého GContextu do již zmíněného unordered_multisetu.

Pokud vytvořím nový GCObject mimo jakýkoli GCContext, automaticky se zavolá statická metoda make_ptr<...>(), nebo new_obj<...>(), která uloží danou hranu do statického GCCOntextu tzv. context_static.

To mimo jiné znamená, že pokud bych si vytvářel nové GCObjekty v objektech, které by nedědily z GCContextu, pak bych je považoval za statické a byly by uloženy v paměti až do konce prgramu (pokud by je uživatel sám ručně nezrušil), kdy se uklízí veškerá paměť (to by nebyla chyba programu, ale programátora).

## 4.3 GarbageCollector

Samotná třída GarbageCollector je template s dvěma typy Memory a Invoker, kde Memory musí dědit z GCMemory a Invoker musí dědit z GCInvoker.

GCMemory a Invoker obsahují základní interfacy pro to, co mají představovat, tedy GCMemory musí mít metody pro práci s pamětí, nebo metodu pro volání updateMemory.

### 4.3.1 updateMemory()

Funguje velmi prostě. Jednoduše procházíme celý graf GCContextů, kdy začínáme od statických objektů (v context_static GCContextu). Dále postupujeme a značíme si ty na které jsme narazili.

Na konec vyházíme všechny GCContexty, na které jsme nenarazili. U nich použijeme funkci delete.

## 4.4 Invoker

Mám naimplementované 2 základní. Oba jsou velmi jednoduché. První tzv. GCDumbInvoker po každé vteřině zavolá updateMemory().

Druhý GCBetterInvoker má maximální délku spánku nastavenou na 10 sekund. podle toho, jak moc přibylo paměťi od posledního spánku, tak tak dlouho bude spát další cyklus.

Zastavují práci ostatních vláken pomocí mutexu.

# 5. Závěr

Mám pocit, že můj garbage collector je v podtatě použitelný a funkční. Rozhodně by se dalo dolaďovat například přístupnost k některým datům.

# Přílohy

https://gitlab.mff.cuni.cz/teaching/nprg041/2021-22/drozdik/tichavso/-/tree/master/zapoctovyProgram/GarbageCollector