УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Лука Ђелић ПР60-2020
Момчило Мицић ПР69-2020

Индустријски комуникациони протокили

ПРОЈЕКАТ
- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 15.01.2024.

САДРЖАЈ

УВОД

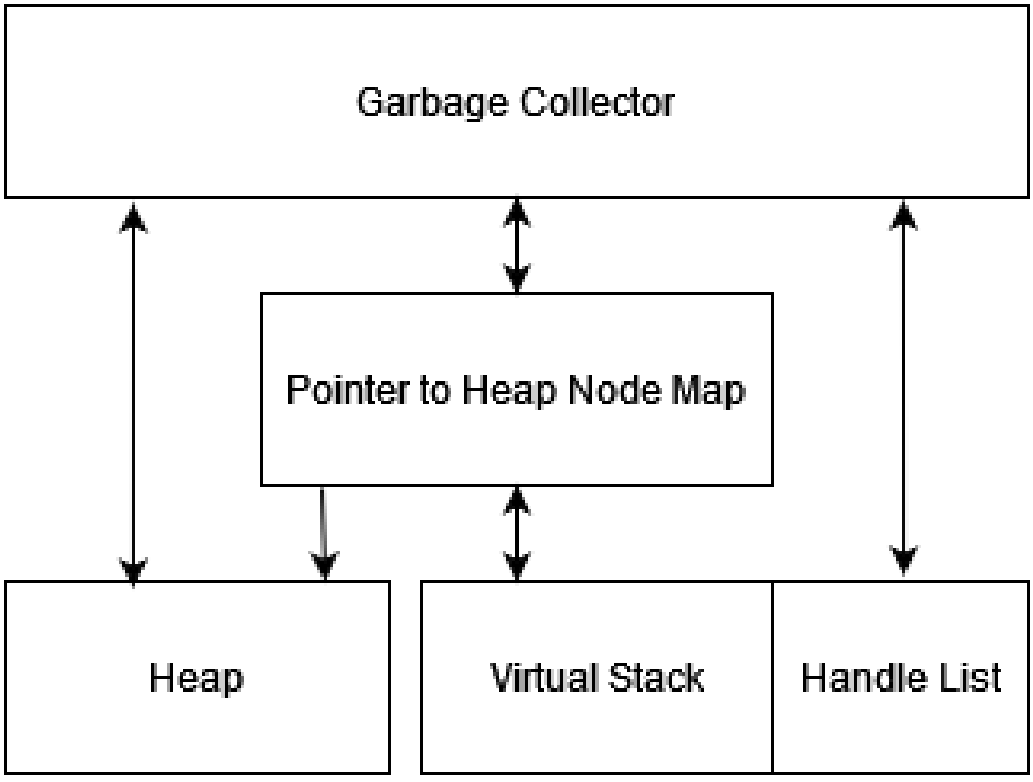
Циљ пројекта је био да се имплементира руководилац Heap-ом који користи Mark and Sweep алгоритам за сакупљање смећа у C програмском језику уз помоћ Windows.h API-ја. Heap треба да буде сегментно организован и да буде thread-safe, што значи да се меморија може безбедно алоцирати и деалоцирати из више нити истовремено.

ДИЗАЈН

Пројекат је осмишљен тако да дизајнирање, разумевање и имплементација буду што лакше, а перформансе на високом нивоу. Поштовали су се принципи конструисања слојевите архитектуре пројекта, где сваки слој управља и/или комуницира само са слојевима директно изнад и испод себе. Тако Garbage Collector рукује Heap-ом, листом нити, и мапом показивача, а свака од тих компоненти има своје методе за руковање сопственом структуром података и враћају GC-ју потребне податке. Методе су обезбеђене критичним секцијама како би се избегло штетно преплитање при руковању подацима.

Хип је, по узору на .NET технологију, подељен на генерације, 0, 1, и 2, где је 0 резервисана за велике објекте (Large Object Heap), а генерације 1 и 2 су за мање објекте (SOH). Сви објекти (HeapNode-ови) из прве генерације који се не почисте у првом циклусу GC-ја, прелазе у другу генерацију. GC се аутоматски активира када понестане места на генерацији 1.

Слика 1. организација пројекта



СТРУКТУРЕ ПОДАТАКА

У пројекту постоји структура GC која садржи показиваче на Heap, листу Handle-ова, тј. референци на нити које су покренуте и мапу показивача и чворова Heap-а (HeapNode-ова).

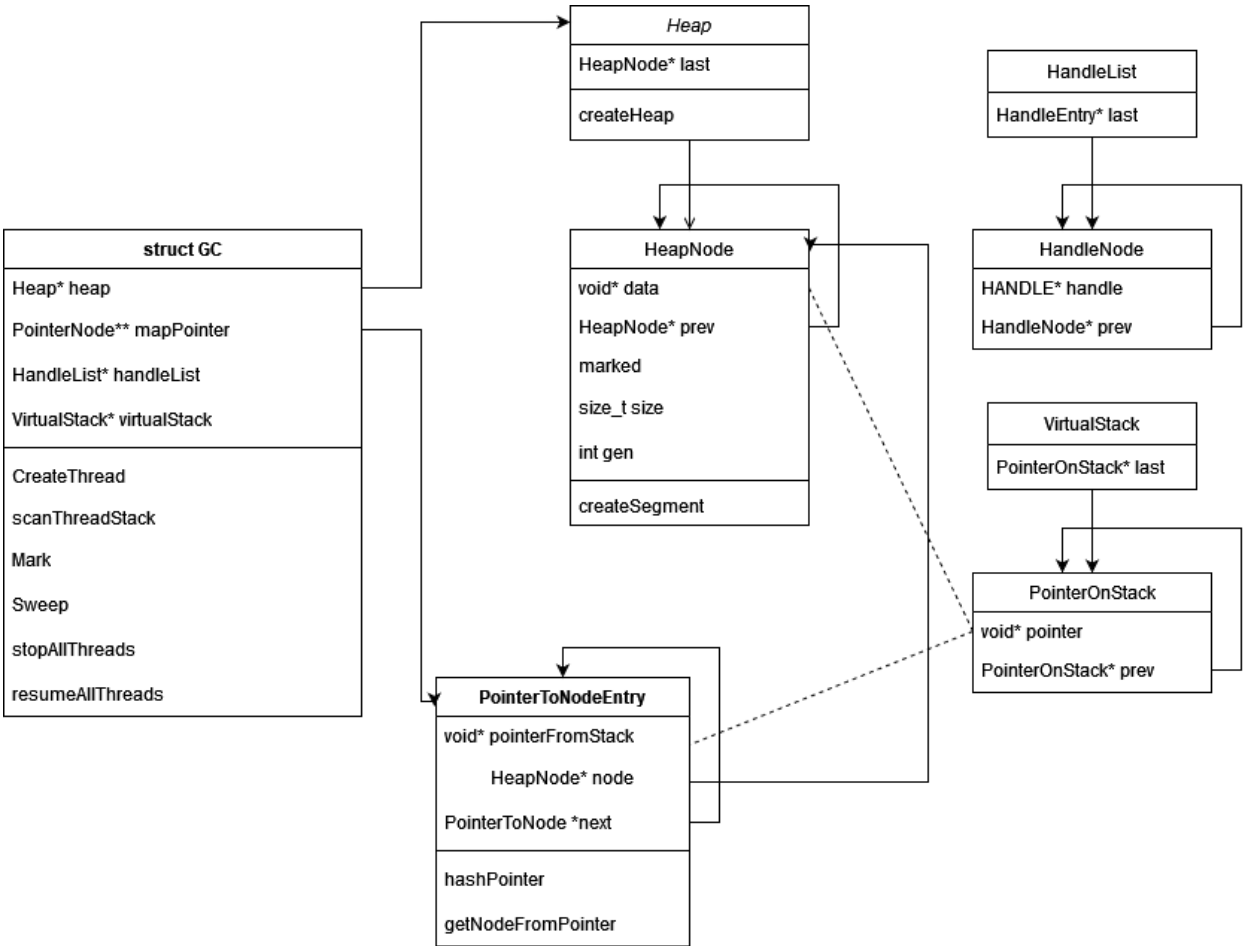
Heap садржи листу чворова а сваки чвор је везан за један показивач који ћемо да алоцирамо, и садржи количину алоцираних бајтова. Листа је реализована тако да се додавање чворова врши на крају листе, на шта имамо показивач, тако да сваки malloc буде O(1) операција и на томе се добија на перформансама. Приликом алоцирања и додавања HeapNode-ова, узимамо показивач на алоциране податке, и додајемо га у мапу која га мапира на HeapNode са којим су ти подаци асоцирани. Ово ће нам помоћи да касније, када будемо пролазили кроз stack нити, проверимо да ли је то што је на стеку показивач, и ако јесте, опет ћемо у O(1) времену наћи чвор хипа који њему одговара и одмах га маркирати.

Мапа је реализована као статички низ чија се величина може подесити променом define-а. Помоћу ње можемо брзо повезати показиваче са стеку са њиховим кореспондирајућим чворовима на Хипу, без да пролазимо кроз цео Хип приликом маркирања, чиме вишеструко скраћујемо време потребно за маркирање.

Листа HANDLE-ова нам служи да итерацијом кроз њу зауставимо све нити, прођемо кроз њихов стек, одрадимо сакупљање смећа и наставимо извршавање нити.

Стек је имплементиран као уланчана листа показивача са стека. Она се аутоматски попуњава приликом алокације, али је потребно поп-овати ствари са стека ручно на крају сваког блока, што смо се договорили да ће бити конвенција.

Слика 2. приказ структура пројекта



ЗАКЉУЧАК

Израда овог пројекта помогла нам је да стекнемо веће искуство у програмирању у C-у, као и да се боље упознамо са Windows API-јем и руковањем нитима помоћу њега. Имали смо проблема приликом проласка кроз стек нити јер из неког разлога ни једна вредност са стека није одговарала показивачима који би требало да буду на стеку. Стога смо имплементирали наш примитивни стек и кроз њега смо пролазили како бисмо чистили меморију.

ПОТЕНЦИЈАЛНА УНАПРЕЂЕЊА

1. Пролазак кроз прави стек нити. То ће нам помоћи да уђемо у срж разумевања процеса, нити, API-ја и да евентуално побољшамо перформансе.
2. Даља оптимизација алгоритма. Додавањем нових структура података могли бисмо додатно убрзати процес чишћења.
3. Динамичко заустављање нити. Наша имплементација тренутно зауставља све нити и пролази кроз њихов стек. Можда постоје неки алгоритми који би нам омогућили да заустављамо само нити које се чисте, а да остале раде свој посао. То би довело до утиска да је рад са апликацијама развијеним у таквом окружењу респонзивнији и са мање ,,штуцања“.
4. Прилагођавање за обе архитектуре. Тренутно је програм рађен за x86 архитектуром, али бисмо могли употребом макроа и претпроцесорских директива да одрадимо и верзију програма за x64 архитектуру.
5. Прављење библиотеке од пројекта. То би омогућило већем броју људи да користе Garbage Collector у оквиру њихових пројеката.
6. Додатна имплементација генерација. Наш GC сада само премешта објекте у другу генерацију. Уколико би се хип правилније изделио на генерације тако да се друга генерација ређе маркира и чисти, тиме бисмо додатно добили на перформансама.