Inicjalizacja i niszczenie obiektów w wybranym języku programowania.

Patryk Lisik

Uniwersytet Łódzki

2024/25

Agenda

- Pojęcia
- Inicjalizacja w językach systemowych.
  - Stos (stack)
  - Sterta (heap)
  - Problem fragmentacji
  - Alokacja pamięci
- Bezpieczeństwo pamięci w językach bez GC
  - Zliczanie referencji(shared_prt)
  - Przekazywanie własności(unique_ptr)
- Problem cyklicznych referencji
- Garbage collectory
  - Taksonomia
  - Działanie Mark/Sweep/Compact

Co to typ?

Typ

Dane i zbiór operacji, które można na nich wykonać [1].

```
class LineFunction {
    double a, b; //dane

    // operacja
    double operator()(double x) const{
        return a * x + b;
    }
};
```

Rysunek 1: Przykład własnego typu w C++

Co to obiekt?

Cykl życia obiektów

Patryk Lisik

Pojęcia

Inicjalizacja w językach systemowych

Bezpieczeństwo pamięci w językach bez GC

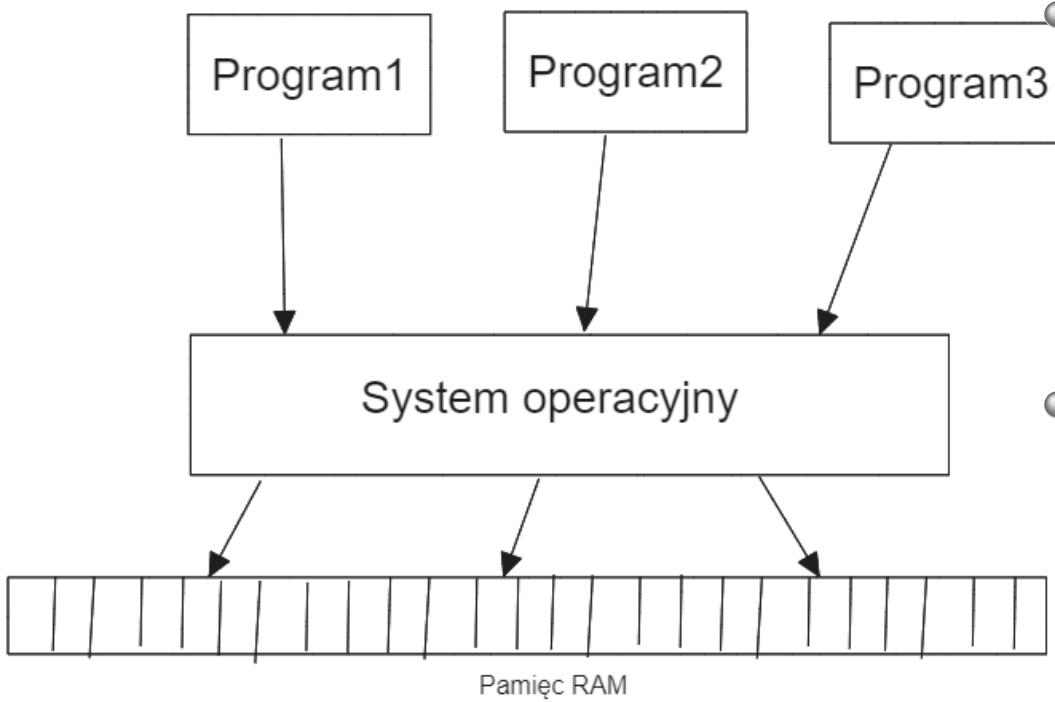
Garbage Collector

Obiekt

Ukonkretnienie typu w postaci zaalokowanej pamięci.

```
LineFunction func{4.5, 2.1};
double d = func(7.2);
```

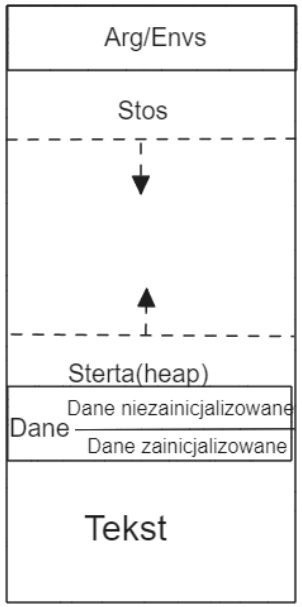
Rola systemu operacyjnego w przydziale pamięci



Rysunek 2: Układ pamięci programu

- Program/Process ma dostęp tylko do pamięci przydzielonej przez OS.
- Odczyt z nie dozwolonej pamięci kończy się wyjątkiem segmentation fault core dumped

Pamięć programu



Rysunek 3: Układ pamięci programu

Argumenty	*argc, zmienne środowiskowe
Stos	Zmienne lokalne programu. ok 1mb
Sterta	Dane alokowane dynamicznie
Dane	Zmienne globalne, vtable,
Tekst	Kod programu

Alokacja na stosie


```
long foo(){
    short d = 7;
    int c = 3
    return d*c;
}
int main(){
    int a = 11;
    long b = foo();
    return 0;
}
```

Szybkość

Kolejna zmienna jest zawsze alokowana na wskaźniku stosu $+1$

Rozmiar

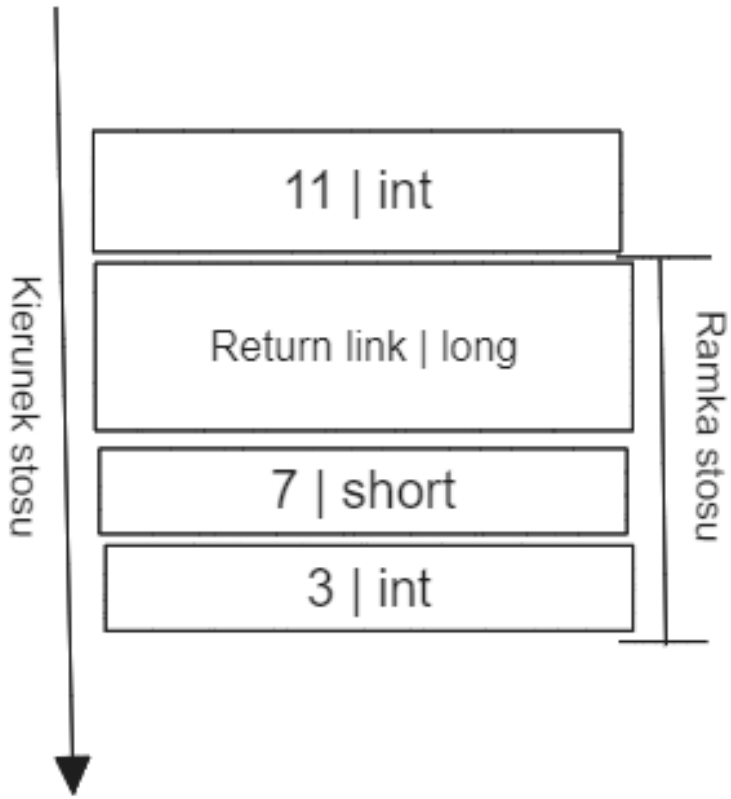
Rozmiar stosu jest stały. Przy jego przekroczeniu występuje wyjątek stack overflow.

Dane

Rozmiar danych przechowywanych na stosie musi być znany w czasie kompilacji.

Dynamiczna alokacja na stosie?


```
long foo(){
    short d = 7;
    int c = 3
    return d*c;
}
int main(){
    int a = 11;
    long b = foo();
    return 0;
}
```

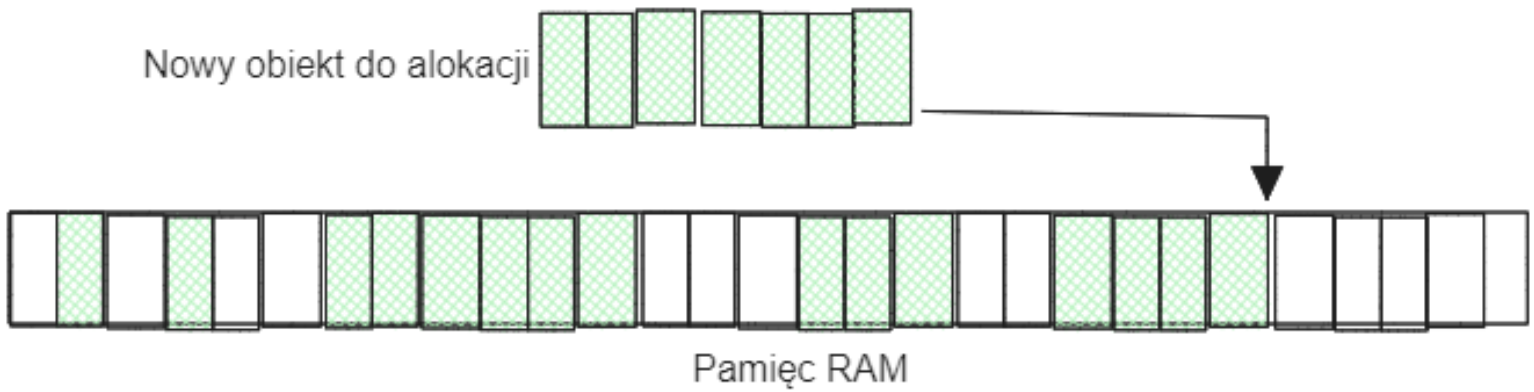


Dynamiczna alokacja na stosie jest niemożliwa, ponieważ elementy nadpisywały by swoje wartości.

Problem fragmentacji

Fragmentacja zewnętrzna

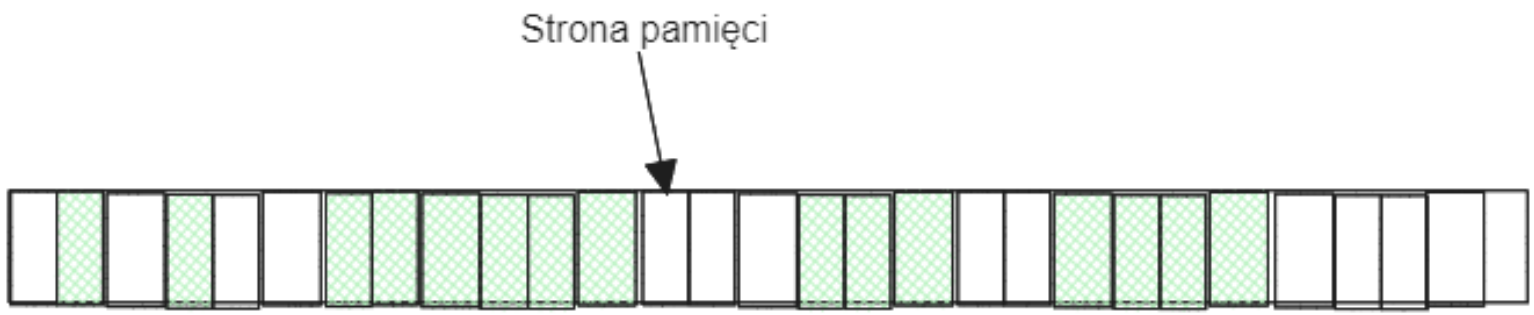
W przypadku pamięci alokowanej dynamicznie nie znamy kolejności alokowania i zwalniania. Możliwe jest, że nowy obiekt nie może zostać umieszczony w pamięci pomimo tego że suma wolnej pamięci jest większa niż jego rozmiar.



Problem fragmentacji

Fragmentacja wewnętrzna

System operacyjny przydziela programom całe bloki pamięci(strony). Zwykle o rozmiarze 8-16 bajtów.



Alokacja na stercie

Cykl życia obiektów

Patryk Lisik

Pojęcia

Inicjalizacja w językach systemowych

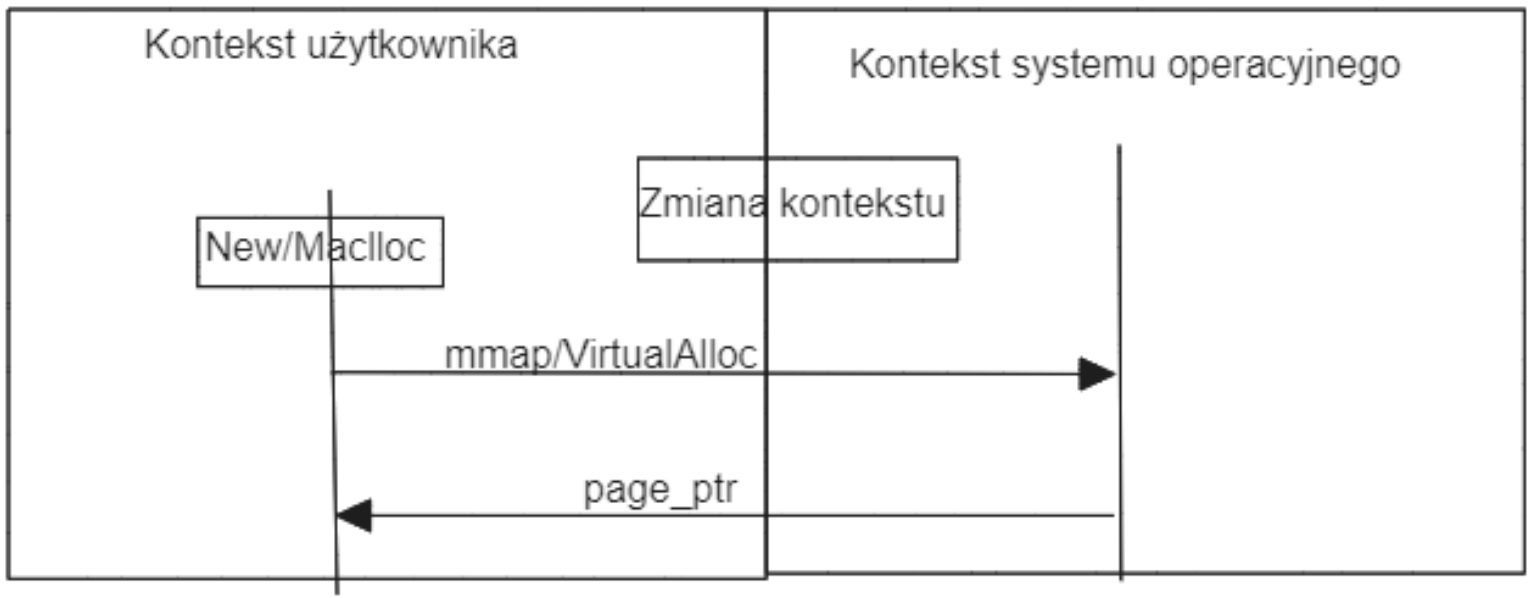
Bezpieczeństwo pamięci w językach bez GC

Garbage Collector

```
int main(){
    int *arr = new int[10];
    return 0;
}
```

Alokacja tablicy 10 intów w C++.

Działanie malloc/new



Działanie new/malloc

Alokacja pamięci

Alokacja pamięci

Alokacja pamięci to rezerwacja przez system operacyjny pamięci dla procesu[2, 3].
(Nie rozpatrujmy sytuacji gdy nie ma systemu operacyjnego)

POSIX

W systemach zgodnych z POSIX alokacja pamięci odbywa się poprzez ‘mmap()‘ [2].

Windows

W systemach Windows alokacja pamięci odbywa się poprzez VirtualAlloc[memoryapi.h] [3].

Zliczanie referencji

Cykl życia obiektów

Patryk Lisik

Pojęcia

Inicjalizacja w językach systemowych

Bezpieczeństwo pamięci w językach bez GC

Garbage Collector

```
template <class T>
class shared_ptr{
    T* value;
    int ref_count;
    shared_prt(shared_ptr& p){ this.ref_count++ }
    ~shared_ptr(){
        ref_count--;
        if(!ref_count){
            delete this.value;
        }
    }
```

std:shared_ptr składa się z licznika referencji i właściwego wskaźnika. Podczas kopi licznik referencji jest zwiększany, a podczas wywołania destruktora zmniejszany. Przy wywołaniu ostatniego destruktora pamięć jest zwlaniana [4].

Uniqe_ptr


```
template <class T>
class unique_ptr{
    T* value;
    unique_ptr(unique_ptr& p) = delete; //copy contru
    unique_ptr& operator=(unique_ptr& p)= delete; //c
}
```

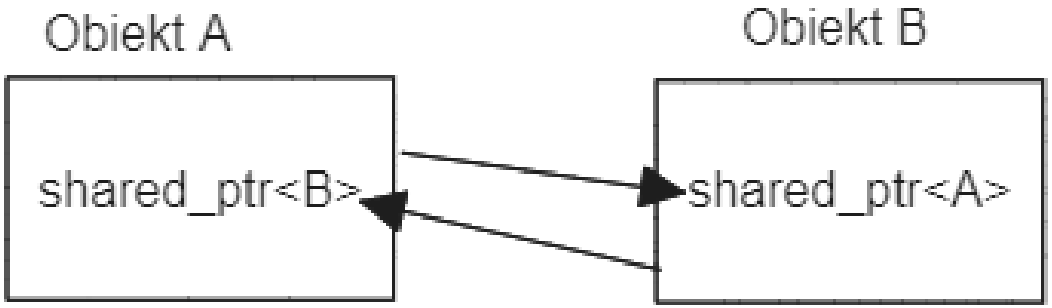
Własność w programowaniu

Własność określa kto jest odpowiedzialny za zwolnienie pamięci [5, 6].

move

std::move informuje kompilator o możliwości przeniesienie zasobów już istniejącego obiektu.

Problem cyklicznych referencji



Rysunek 4: Problem cyklicznych referencji gdzie pamięć nigdy nie jest zwalniana.

std::weak_ptr

Rozwiązaniem problemu cyklicznych referencji jest referencja słaba(std::weak_ptr) która nie powoduje inkrementacji licznika referencji. Nie gwarantuje one że obiekt nie został już usunięty [7, 8].

Typy Garbage collectorów

Rodzaj przetwarzania

Stop the world Zatrzymuje aplikację do działania.

Współbieżny (ang. *Concurrent*) Operuje na wielu wątkach procesach.

Równoległy (ang *Parallel*) Operuje wraz z przetwarzaniem aplikacji.

Fazy przetwarzania

Monolityczny Wykonuje cały process na raz

Inkrementalny Process podzielony na fazy.

Generacyjny

Nowe obiekty są częściej poddawane procesowi GC. Pozwala rzadziej wykonywać defragmentację starszych obiektów.

Oznacz i zmieć (ang. *mark and sweep*)

Faza mark

Przeszukujemy "w głąb" graf obiektów i oznaczamy każdy obiekt do którego dotarliśmy [9].

Faza sweep

Usuwamy obiekty nieoznaczone.

Faza compact(Opcjonalna)

Obiekty przesuwane są blisko siebie w pamięci.

Literatura I

 "Ibm.com – type definitions." https://www.ibm.com/docs/en/epfz/5.3?topic=reference-type-definitions, (dostęp 20-10-2023).

 "mmap(2) — linux manual page." https://man7.org/linux/man-pages/man2/mmap.2.html, (dostęp 31-10-2023).

 "Virtualalloc function (memoryapi.h)." https://learn.microsoft.com/en-gb/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc?redirectedfrom=MSDN, (dostęp 31-10-2023).

 "Cpp reference - std::move." https://en.cppreference.com/w/cpp/utility/move, (dostęp 31-10-2023).

Literatura II


"The rust programming language sekcja [what is ownership?]." https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html#what-is-ownership, (dostęp 31-10-2023).


"Cpp reference - dynamic memory managemnt - std::unique_ptr." https://en.cppreference.com/w/cpp/memory/unique_ptr, (dostęp 31-10-2023).


"Weak pointers and circular references in c++ 11." https://visualstudiomagazine.com/articles/2012/10/19/circular-references.aspx, (dostęp 31-10-2023).

Literatura III


"Cpp reference - dynamic memory management - std::weak_ptr." https://en.cppreference.com/w/cpp/memory/weak_ptr, (dostęp 31-10-2023).


"Cs330 - mark and sweep." https://www.cs.odu.edu/~zeil/cs330/f13/Public/garbageCollection/garbageCollection-htmlsu5.html, (dostęp 31-10-2023).