Správa paměti

Ing. Marek Běhálek
katedra informatiky FEI VŠB-TUO
A-1018 / 597 324 251
http://www.cs.vsb.cz/behalek
marek.behalek@vsb.cz

Obsah přednášky

- Motivace
- Úrovně správy paměti.
- Manuální a automatická správa paměti.
- Metody přidělování paměti.
- Metody regenerace paměti.
- Správa paměti v konkrétních programovacích jazycích.

Motivace

- **Statické přidělení paměti**
  - Známe předem *velikost* i *počet* prvků datové struktury
    ```
    int pole[10]; // 10 * sizeof(int)
    ```
  - Proměnné deklarované na globální úrovni
  - Statické proměnné
- **Dynamické přidělení paměti**
  - Neznámý počet prvků
    ```
    int* pole = new int[pocet];
    ```
  - Rekurze
    ```
    int f(int n) {
      return n == 0 ? 1 : n * f(n-1);
    }
    ```


Úrovně správy paměti

- **Technické vybavení**
  - registry, cache
- **Operační systém**
  - virtuální paměť
  - segmentace, stránkování
- **Aplikace**
  - Přidělování paměti
  - Regenerace paměti
    - Manuální – delete, dispose, free(), …
    - Automatická – garbage collection

Omezení na správu paměti

- **Časová režie**
  - Kolik času navíc zabere správa paměti?
- **Doba pozdržení interaktivity**
  - Na jak dlouho se aplikace „zasekne“ během správy paměti?
- **Paměťová režie**
  - Kolik prostoru spotřebuje správa paměti?
    - Interní fragmentace – zaokrouhlování velikosti bloků
    - Externí fragmentace – nevhodné využití paměti

Správa paměti 5

Problémy správy paměti

- **Předčasné uvolnění paměti**
  - Přístup k paměti, která již byla uvolněna
- **Únik paměti (memory leak)**
  - Neuvolňování nepotřebné paměti
- **Externí fragmentace**
  - Rozdělení volné paměti na mnoho malých bloků
- **Špatná lokalita odkazů**
  - Vliv velikosti cache
- **Chybné předpoklady při návrhu aplikace**

Správa paměti 6

Historie

- 1957 – FORTRAN, statické přidělování
- 1958 – LISP, dynamické přidělování s regenerací
- 1962 – virtuální paměť
- 1965 – vyrovnávací paměti (cache)
- 1969 – Intel – čip 1 kB RAM
- 1974 – Intel 8080 – přístup k 64 kB paměti
- 1975 – Dijkstra - inkrementální regenerace paměti

Manuální správa paměti

- Program sám vrací část paměti, kterou nepotřebuje
- **Přidělování z hromady**
  - Volání funkcí pro přidělování a uvolňování paměti
  - Zodpovědnost je na programátorovi
  - Střídání bloků přidělené a volné paměti
- **Přidělování na zásobníku**
  - Pro lokální proměnné a parametry funkcí
  - Uvolňuje se vždy naposledy přidělená paměť

Automatická správa paměti

- **Vyhledání obsazených bloků paměti**
  - *Živé bloky* – program s nimi dále pracuje (jak to zjistíme??)
  - *Dostupné bloky* – program s nimi je schopen dále pracovat
    - Globální a lokální proměnné, registry – *kořeny dosažitelnosti*
- **Regenerace nevyužité paměti**
  - Paměť je k dispozici pro opakované přidělení

Správa paměti 9

Automatická správa paměti

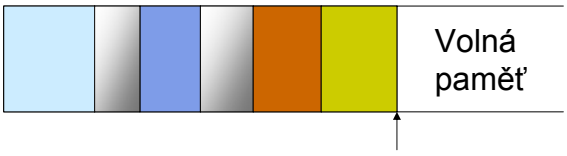
- **Výhody**
  - Programátor se může věnovat jiným problémům
  - Přehlednější rozhraní programových modulů (kdo uvolní přidělenou paměť?)
  - Menší množství chyb spojených s přístupem do paměti
  - Mnohem efektivnější správa paměti
- **Nevýhody**
  - Uvolňuje se pouze nedostupná paměť
  - Není k dispozici ve starších jazycích

Správa paměti 10

Metody přidělování paměti

- Máme k dispozici jistým způsobem organizovanou volnou paměť.
- Dle požadavků aplikace postupně odebíráme a přidělujeme jednotlivým datovým objektům.
- Úkolem přidělování paměti je pro zadanou velikost požadované paměti vyhledat vhodný úsek volné paměti a jeho adresu vrátit.

Přidělování na zásobník



- **Přidělení paměti**
  - Posune ukazatel o velikost přiděleného prostoru
- **Uvolnění paměti**
  - Prázdná operace
  - Vyplnění prostoru spec. vzorem – pro ladění

Přidělování na zásobník

```
class SimpleAllocator {
public SimpleAllocator(char* memAddr, unsigned memSize) {
  m_addr = memAddr;
  m_size = memSize;
}
public char* alloc(unsigned size) {
  if( size > m_size ) throw new NoMemoryException();
  char* addr = m_addr;
  m_addr += size;
  return addr;
}
public void free(char* addr, unsigned size) {}
// Aktuální začátek volné paměti
protected char* m_addr;
// Aktuální velikost volné paměti
protected unsigned m_size; }
```

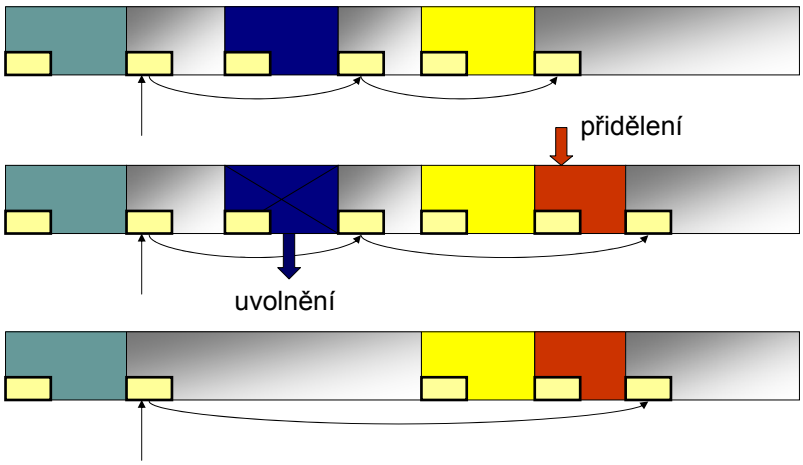

Přidělování na zásobník

- velmi rychlá metoda
- často se požívá v rámci bloku, který získáme jinou metodou – sublokátor
- přidělování paměti pro aktivační záznamy (C/C++/Pascal)
- jednou z modifikací je zavedení operací *mark* a *release*

Přidělování ze seznamu volných bloků

- **Volné bloky paměti seřazené do seznamu**
- **Přidělení paměti**
  - Vyhledání bloku vhodné velikosti
  - Zařazení nevyužité části volného bloku zpět do seznamu
- **Uvolnění paměti**
  - Zařazení bloku do seznamu
  - Spojení s navazujícími bloky – snížení fragmentace
  - Přesun volných bloků do jednoho celku (defragmentace paměti) – co s ukazateli?

Přidělování ze seznamu volných bloků



Metody pro přidělování volných bloků

- First fit
  - Prochází seznam volných bloků a vybere první, jehož velikost je větší nebo rovna požadované velikosti.
  - Blok je rozdělen na přidělenou paměť a zbylé volné místo
  - Při uvolňování je potřeba volný blok opět zařadit do seznamu
  - Nevýhoda – na začátku seznamu vznikne mnoho malých bloků, jejich sekvenční procházení může podstatně zpomalit operaci přidělování paměti.

Správa paměti 17

Metody pro přidělování volných bloků

- Různě strukturovaný seznam
  - Zařazení uvolněného bloku na začátek/konec seznamu – složitější slévání sousedních volných bloků.
  - Použití uspořádaného seznamu podle adres bloku.
  - Použití složitějších struktur – například stromy.
  - Lze použít u všech prezentovaných metod

Správa paměti 18

Metody pro přidělování volných bloků

- Next fit
  - vyhledávání začíná na pozici, kde předchozí vyhledávání skončilo
  - zabrání hromadění menších bloků na začátku seznamu
  - Nevýhoda – přidělené bloky od sebe mohou být značně vzdálené
  - Celkově vede k menší efektivitě

Správa paměti 19

Metody pro přidělování volných bloků

- Best fit
  - snažíme se najít volný blok, jehož velikost je větší nebo rovna požadované
  - Nevýhody
    - pro velké množství objektů bude prohledávání a tedy i přidělování paměti pomalé
    - velké množství nepoužitelných malých zbytků paměti
  - Optimalizace
    - více seznamů pro bloky přibližně stejné délky
    - indexace + seřazení bloků podlé délky (nebo složitější datové struktury)

Správa paměti 20

Metody pro přidělování volných bloků

- Worst fit
  - Je použit největší volný blok paměti.
  - Nově vzniklý blok (zbytek po přidělení paměti) bude dostatečně velký.

Správa paměti 21

Přidělování s omezenou velikostí bloku (buddy system)

- Hierarchické dělení paměti na části podle pevných pravidel
- Spojovat lze pouze sousední bloky v hierarchii (buddies)
- Výsledek pak bude patřit do bezprostředně vyšší úrovně hierarchie.
- Souseda bloku najdeme jednoduchým výpočtem – nižší režie (1 bit volný/obsazený)
- Pro každou možnou velikost bloků se udržuje samostatný seznam volných bloků.

Správa paměti 22

Binární přidělování

- Nejznámější buddy systém.
- Velikost bloků je vždy mocnina dvou.
- Při dělení se blok rozdělí na dvě stejné části.
- Všechny bloky jsou „zarovnané“ na mocninu dvou.
- Přístup k blokům je tedy zajištěn na jednoduchém výpočtu pomocí bitových operací.

Example of Buddy System

1 Mbyte block	1 M					
Request 100 K	A = 128 K	128 K	256 K	512 K		
Request 240 K	A = 128 K	128 K	B = 256 K	512 K		
Request 64 K	A = 128 K	C = 64 K	64 K	B = 256 K	512 K	
Request 256 K	A = 128 K	C = 64 K	64 K	B = 256 K	D = 256 K	256 K
Release B	A = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Release A	128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Request 75 K	E = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Release C	E = 128 K	128 K	256 K	D = 256 K	256 K	
Release E	512 K			D = 256 K	256 K	
Release D	1 M					

Fibonacciho přidělování

- f(1)=1, f(2)=2, f(n)=f(n-1)+f(n-2)
  - 1, 2, 3, 5, 8, …
- Tato varianta se snaží snížit vnitřní fragmentaci bloků využít kompaktnější sady možných velikostí bloků.
- Každý blok lze beze zbytku rozdělit na dva bloky, jejichž velikosti jsou opět prvky řady.
- Problémem je přidělování několika bloků stejné velikosti.

Správa paměti 25

Metody regenerace paměti

- Nepoužité bloky paměti chceme znovu využít
- **Jak vyhledat nepoužité bloky?**
  - Čítače odkazů
  - Sledování odkazů – značkování
- **Inkrementální metody**
  - Úklid paměti se střídá s během aplikace
  - Lze využít pro systémy pracující v reálném čase

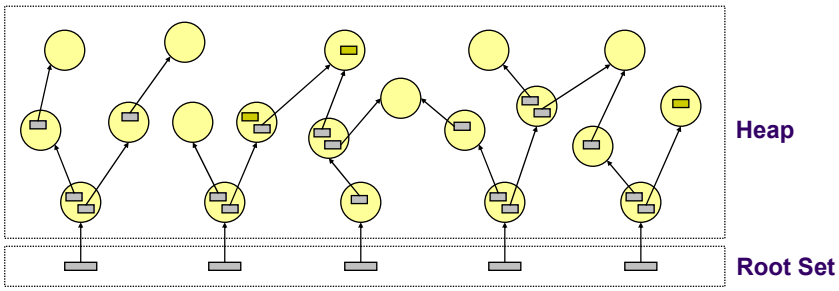
Správa paměti 26

Metoda dvoufázového značkování (mark and sweep)

- **Předpoklady:**
  - Známe kořeny dosažitelnosti
    - Globální a lokální proměnné
    - Registry
  - Známe strukturu dat (kde jsou další ukazatele)
- **Postup:**
  - Označkujeme všechny bloky dosažitelné z kořenů
  - Neoznačené bloky uvolníme

Správa paměti 27

Metoda dvoufázového značkování (mark and sweep)



Správa paměti 28

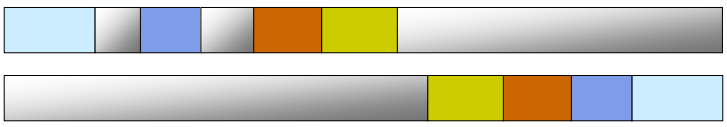
Metoda dvoufázového značkování (mark and sweep)

- Čas potřebný na prozkoumání dostupnosti objektů je závislý na velikosti hromady, ne na množství „smetí“ (garbage).
- Jeden „průchod“ ušetříme, pokud použijeme *regeneraci s kopírováním*.

Správa paměti 29

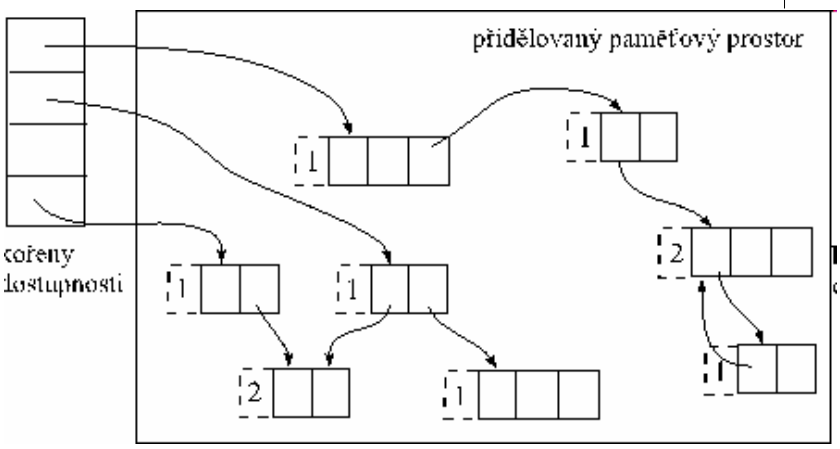
Regenerace s kopírováním

- Všechny obsazené bloky zkopírujeme do jiné souvislé oblasti paměti (vyhledáme je stejně jako u metody Mark and Sweep).
- Je třeba přesměrovat odkazy na kopírované objekty
- Při kopírování se snažíme vytvořit jeden kompaktní blok (má-li A odkaz na B, dáme je za sebe…)



Správa paměti 30

Regenerace s počítáním odkazů (reference counting)



Regenerace s počítáním odkazů (reference counting)

- **Postup**
  - Každý objekt obsahuje počitadlo odkazů
  - L := R zvýší počet odkazů na R a sníží počet odkazů na předchozí hodnotu L
  - Při snížení počitadla na nulu se paměť uvolní
- **Problémy**
  - Nefunguje pro cyklické odkazy
  - Nebezpečí tranzitivního uvolňování – odezva
  - Velká režie při častých změnách – lok. proměnné

Generační regenerace paměti (Generational)

- Většina objektů má krátkou životnost.
- Objekty rozděleny do několika oblastí dle jejich „stáří“.
- Pokud objekt „přežije“ regeneraci paměti je zvětšena hodnota určující jeho stáří.
- Objekty v oblastech pro „staré“ objekty není nutné tak často testovat.

Správa paměti v jazyce C

- Standardní knihovní funkce
  - void* malloc(size_t size)
  - void* calloc(size_t num, size_t size)
  - void* realloc(void* ptr, size_t size)
  - void free(void* ptr)
- Implementace obvykle seznamem volných bloků

```
long* buffer;
buffer = (long*)calloc(40, sizeof(long));
buffer = (long*)realloc(buffer,
                        100 * sizeof(long));
```

Správa paměti v jazyce C++

- Součást syntaxe jazyka
  - T* ptr = **new** T()
  - **delete** ptr;
- Přidělování paměti pro pole
  - long* buffer = new long[40];
  - delete[] buffer;
- Nekombinovat malloc/new a free/delete!

Správa paměti v jazyce C++

- Vlastní přidělování/uvolňování pro třídu
  - class C {
    public:
    C(char* s);
    void* operator new(size_t size, int arg);
    void operator delete(void* ptr, size_t size);
    …
    }
  - C* cp = new(10) C(“abc”);

Správa paměti v jazyce Java

- **Automatická regenerace paměti**
  - (záleží na výrobci, SUN, IBM - mark & sweep, prázdné bloky spojovány kopírováním, generational GC
  - Neexistuje destruktor (kdy by se měl volat?)
- **Inkrementální regenerace**
  - Samostatné vlákno s nízkou prioritou
  - Před uvolněním paměti volá metodu: `protected void finalize() throws Throwable`
  - Můžeme přímo volat garbage collector: `System.gc()`
- Pro vyhledání referencí se využívají *metadata*

Správa paměti 37

Správa paměti v jazyce C# (1)

- **Automatická regenerace paměti**
  - Používá algoritmus next fit, regenerace s kopírováním, využívá generací.
    - Udržuje pointer na další volné místo: NextObjPointer.
    - Při regeneraci paměti jsou objekty na hromadě „setříděny“ dle jejich vzdálenosti od kořenů.
    - Udržuje tři generace.
      - Vytvořené objekty
      - Objekty které přežily jeden průchod GC.
      - Objekty které přežily více průchodů GC.

Správa paměti 38

Správa paměti v jazyce C# (2)

- **Inkrementální regenerace**
  - Dva vlákna běžící na pozadí
    1. Používá metodu dvoufázového značkování a identifikuje „garbage“.
    2. Volá `finalize` a uvolňuje pamět.

- Můžeme explicitně uvolnit paměť: `System.GC.Collect` – uvolní všechny
- Pro jednotlivé instance nutno použít rozhraní `System.IDispose`

Správa paměti 39