Inżynieria oprogramowania: **optymalizacja pamięciowa programów**

Zdzisław Sroczyński

Politechnika Śląska
Instytut Matematyki
Wydział Matematyki Stosowanej

- Techniki optymalizacji pamięciowej:

- Techniki optymalizacji pamięciowej:
  - ręczne poprawianie kodu (zastępowanie konstrukcji języka, często niebezpieczne ze względu na czytelność kodu)

- Techniki optymalizacji pamięciowej:
  - ręczne poprawianie kodu (zastępowanie konstrukcji języka, często niebezpieczne ze względu na czytelność kodu)
  - wykorzystanie usług oferowanych przez system operacyjny (pamięć wirtualna, nakładkowanie)

- Techniki optymalizacji pamięciowej:
  - ręczne poprawianie kodu (zastępowanie konstrukcji języka, często niebezpieczne ze względu na czytelność kodu)
  - wykorzystanie usług oferowanych przez system operacyjny (pamięć wirtualna, nakładkowanie)
  - zmiana algorytmu

- Techniki optymalizacji pamięciowej:
  - ręczne poprawianie kodu (zastępowanie konstrukcji języka, często niebezpieczne ze względu na czytelność kodu)
  - wykorzystanie usług oferowanych przez system operacyjny (pamięć wirtualna, nakładkowanie)
  - zmiana algorytmu
- optymalizacja pamięciowa istotna:
  - w systemach wbudowanych (urządzenia o niewielkich zasobach pamięci, często rzędu kilku kB)

- Techniki optymalizacji pamięciowej:
  - ręczne poprawianie kodu (zastępowanie konstrukcji języka, często niebezpieczne ze względu na czytelność kodu)
  - wykorzystanie usług oferowanych przez system operacyjny (pamięć wirtualna, nakładkowanie)
  - zmiana algorytmu
- optymalizacja pamięciowa istotna:
  - w systemach wbudowanych (urządzenia o niewielkich zasobach pamięci, często rzędu kilku kB)
  - aplikacjach internetowych (rozmiar przesyłanych danych ma duży wpływ na wydajność aplikacji AJAX, techniczne ograniczenia metody GET)

- Techniki optymalizacji pamięciowej:
  - ręczne poprawianie kodu (zastępowanie konstrukcji języka, często niebezpieczne ze względu na czytelność kodu)
  - wykorzystanie usług oferowanych przez system operacyjny (pamięć wirtualna, nakładkowanie)
  - zmiana algorytmu
- optymalizacja pamięciowa istotna:
  - w systemach wbudowanych (urządzenia o niewielkich zasobach pamięci, często rzędu kilku kB)
  - aplikacjach internetowych (rozmiar przesyłanych danych ma duży wpływ na wydajność aplikacji AJAX, techniczne ograniczenia metody GET)

- Metody poprawiania kodu w celu optymalizacji pamięciowej:

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych
  - przechowywanie powtarzających się wyrażeń w zmiennych pomocniczych

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych
  - przechowywanie powtarzających się wyrażeń w zmiennych pomocniczych
  - zależność powyższych metod od optymalizatora

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych
  - przechowywanie powtarzających się wyrażeń w zmiennych pomocniczych
  - zależność powyższych metod od optymalizatora
  - zastąpienie kilku podobnych funkcji jedną o rozbudowanej liście parametrów

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych
  - przechowywanie powtarzających się wyrażeń w zmiennych pomocniczych
  - zależność powyższych metod od optymalizatora
  - zastąpienie kilku podobnych funkcji jedną o rozbudowanej liście parametrów
  - funkcje wywoływane jednokrotnie – rozwijane w miejscu wywołania *inline*

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych
  - przechowywanie powtarzających się wyrażeń w zmiennych pomocniczych
  - zależność powyższych metod od optymalizatora
  - zastąpienie kilku podobnych funkcji jedną o rozbudowanej liście parametrów
  - funkcje wywoływane jednokrotnie – rozwijane w miejscu wywołania *inline*
  - zastąpienie makrodefinicjami lub inline funkcji o niewielkim kodzie, ale parametrach zajmujących dużo miejsca

- Metody poprawiania kodu w celu optymalizacji pamięciowej:
  - niejawna konwersja typów
  - wykorzystanie typów całkowitoliczbowych
  - przechowywanie powtarzających się wyrażeń w zmiennych pomocniczych
  - zależność powyższych metod od optymalizatora
  - zastąpienie kilku podobnych funkcji jedną o rozbudowanej liście parametrów
  - funkcje wywoływane jednokrotnie – rozwijane w miejscu wywołania *inline*
  - zastąpienie makrodefinicjami lub inline funkcji o niewielkim kodzie, ale parametrach zajmujących dużo miejsca

- Optymalizacja struktur danych:

- Optymalizacja struktur danych:
- odniesienie do faktycznego rodzaju reprezentowanych danych

- Optymalizacja struktur danych:
- odniesienie do faktycznego rodzaju reprezentowanych danych
- unikanie narzutu związanego z rozbudowanymi strukturami danych, takimi jak listy i drzewa

- Optymalizacja struktur danych:
- odniesienie do faktycznego rodzaju reprezentowanych danych
- unikanie narzutu związanego z rozbudowanymi strukturami danych, takimi jak listy i drzewa
- zarządzanie pamięcią wirtualną przez aplikację

- Optymalizacja struktur danych:
- odniesienie do faktycznego rodzaju reprezentowanych danych
- unikanie narzutu związanego z rozbudowanymi strukturami danych, takimi jak listy i drzewa
- zarządzanie pamięcią wirtualną przez aplikację

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)
  - kod wykorzystywany przez wiele aplikacji (np. moduły systemu operacyjnego)

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)
  - kod wykorzystywany przez wiele aplikacji (np. moduły systemu operacyjnego)
  - oszczędność miejsca na dysku (potencjalna)

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)
  - kod wykorzystywany przez wiele aplikacji (np. moduły systemu operacyjnego)
  - oszczędność miejsca na dysku (potencjalna)
  - kod biblioteki DLL ładowany jest jednokrotnie i odwzorowywany w przestrzeń adresową danego procesu - oszczędność pamięci fizycznej w systemach wielozadaniowych

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)
  - kod wykorzystywany przez wiele aplikacji (np. moduły systemu operacyjnego)
  - oszczędność miejsca na dysku (potencjalna)
  - kod biblioteki DLL ładowany jest jednokrotnie i odwzorowywany w przestrzeń adresową danego procesu - oszczędność pamięci fizycznej w systemach wielozadaniowych
  - możliwość ładowania potrzebnych w danej chwili bibliotek i niezwłocznego usuwania niepotrzebnych

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)
  - kod wykorzystywany przez wiele aplikacji (np. moduły systemu operacyjnego)
  - oszczędność miejsca na dysku (potencjalna)
  - kod biblioteki DLL ładowany jest jednokrotnie i odwzorowywany w przestrzeń adresową danego procesu - oszczędność pamięci fizycznej w systemach wielozadaniowych
  - możliwość ładowania potrzebnych w danej chwili bibliotek i niezwłocznego usuwania niepotrzebnych
  - ułatwienie pielęgnacji programu - aktualizacje, możliwość dołączania wtyczek/rozszerzeń itp.

- nakładkowanie poprzez wykorzystanie bibliotek łączonych dynamicznie (DLL – ang. *Dynamic-Link Library* w Windows, so w Linux)
  - kod wykorzystywany przez wiele aplikacji (np. moduły systemu operacyjnego)
  - oszczędność miejsca na dysku (potencjalna)
  - kod biblioteki DLL ładowany jest jednokrotnie i odwzorowywany w przestrzeń adresową danego procesu - oszczędność pamięci fizycznej w systemach wielozadaniowych
  - możliwość ładowania potrzebnych w danej chwili bibliotek i niezwłocznego usuwania niepotrzebnych
  - ułatwienie pielęgnacji programu - aktualizacje, możliwość dołączania wtyczek/rozszerzeń itp.

Kod programu w C:

```
int function_name(int, int, int);
int a, b, c, x;
...
x = function_name(a, b, c);
```

Kod programu w C:

```
int function_name(int, int, int);
int a, b, c, x;
...
x = function_name(a, b, c);
```

Kod w asemblerze:

```
push c                  ; arg 3
push b                  ; arg 2
push a                  ; arg 1
call function_name      ; skok do funkcji
add esp, 12             ; zdjęcie ze stosu
mov x, eax              ; wynik funkcji
```

Tabela sposobów wywoływania funkcji:

Architecture	Calling convention name	Operating system, Compiler	Parameters in registers	Parameter order on stack	Stack cleanup by
16-bit	cdecl			RTL (C)	caller
	pascal			LTR (Pascal)	function
	fastcall	Microsoft (non-member)	ax, dx, bx	LTR (Pascal)	function
	fastcall	Microsoft (member function)	ax, dx	LTR (Pascal)	function
	fastcall	Borland compiler[10]	ax, dx, bx	LTR (Pascal)	function
		Watcom compiler	ax, dx, bx, cx	RTL (C)	function
32-bit	cdecl			RTL (C)	caller
	stdcall			RTL (C)	function
		GCC		RTL (C)	hybrid
	fastcall	Microsoft	ecx, edx	RTL (C)	function
	fastcall	GCC	ecx, edx	RTL (C)	function
	fastcall	Borland/Embarcadero compiler	eax, edx, ecx	LTR (Pascal)	function
	thiscall	Microsoft	ecx	RTL (C)	function
		Watcom compiler	eax, edx, ebx, ecx	RTL (C)	function
64-bit	Microsoft x64 calling convention[7]	Windows (Microsoft compiler, Intel compiler, Embarcadero compiler)	rcx/xmm0, rdx/xmm1, r8/xmm2, r9/xmm3	RTL (C)	caller
	System V AMD64 ABI convention[9]	Linux, BSD, Mac (GCC, Intel compiler)	rdi, rsi, rdx, rcx, r8, r9, xmm0-7	RTL (C)	caller

- Problemy związane z bibliotekami DLL

- Problemy związane z bibliotekami DLL
  - programy mogą korzystać z tej samej biblioteki, ale w różnych wersjach (zależność wyłącznie od **nazwy**)

- Problemy związane z bibliotekami DLL
  - programy mogą korzystać z tej samej biblioteki, ale w różnych wersjach (zależność wyłącznie od **nazwy**)
  - „piekło DLL” – kolejność ładowania bibliotek (system/katalog aplikacji), przechowywanie DLL-i w katalogu aplikacji – koniec z oszczędnością miejsca na dysku

- Problemy związane z bibliotekami DLL
  - programy mogą korzystać z tej samej biblioteki, ale w różnych wersjach (zależność wyłącznie od **nazwy**)
  - „piekło DLL” – kolejność ładowania bibliotek (system/katalog aplikacji), przechowywanie DLL-i w katalogu aplikacji – koniec z oszczędnością miejsca na dysku
  - sposoby łączenia bibliotek DLL z programami wywołującymi: statyczny i dynamiczny

- Problemy związane z bibliotekami DLL
  - programy mogą korzystać z tej samej biblioteki, ale w różnych wersjach (zależność wyłącznie od **nazwy**)
  - „piekło DLL” – kolejność ładowania bibliotek (system/katalog aplikacji), przechowywanie DLL-i w katalogu aplikacji – koniec z oszczędnością miejsca na dysku
  - sposoby łączenia bibliotek DLL z programami wywołującymi: statyczny i dynamiczny
  - łączenie statyczne powoduje załadowanie wszystkich bibliotek przy starcie programu – koniec z oszczędnością pamięci fizycznej

- Problemy związane z bibliotekami DLL
  - programy mogą korzystać z tej samej biblioteki, ale w różnych wersjach (zależność wyłącznie od **nazwy**)
  - „piekło DLL” – kolejność ładowania bibliotek (system/katalog aplikacji), przechowywanie DLL-i w katalogu aplikacji – koniec z oszczędnością miejsca na dysku
  - sposoby łączenia bibliotek DLL z programami wywołującymi: statyczny i dynamiczny
  - łączenie statyczne powoduje załadowanie wszystkich bibliotek przy starcie programu – koniec z oszczędnością pamięci fizycznej

- Tworzenie bibliotek DLL

- Tworzenie bibliotek DLL
  - plik def (dyrektywy LIBRARY, EXPORTS)

- Tworzenie bibliotek DLL
  - plik def (dyrektywy LIBRARY, EXPORTS)
  - oznaczenie funkcji eksportowanych `__declspec(dllexport)`

- Tworzenie bibliotek DLL
  - plik def (dyrektywy LIBRARY, EXPORTS)
  - oznaczenie funkcji eksportowanych `__declspec(dllexport)`
  - przykład 1: **tworzenie biblioteki DLL w C**

- Tworzenie bibliotek DLL
  - plik def (dyrektywy LIBRARY, EXPORTS)
  - oznaczenie funkcji eksportowanych `__declspec(dllexport)`
  - przykład 1: **tworzenie biblioteki DLL w C**

- Używanie bibliotek DLL - łączenie statyczne

- Używanie bibliotek DLL - łączenie statyczne
  - plik lib lub a plus biblioteka DLL – informacje dla linkera

- Używanie bibliotek DLL - łączenie statyczne
  - plik lib lub a plus biblioteka DLL – informacje dla linkera
  - oznaczenie funkcji importowanych `__declspec(dllimport)`

- Używanie bibliotek DLL - łączenie statyczne
  - plik lib lub a plus biblioteka DLL – informacje dla linkera
  - oznaczenie funkcji importowanych `__declspec(dllimport)`
  - przykład 2: **statyczne łączenie biblioteki DLL w C**

- Używanie bibliotek DLL - łączenie statyczne
  - plik lib lub a plus biblioteka DLL – informacje dla linkera
  - oznaczenie funkcji importowanych `__declspec(dllimport)`
  - przykład 2: **statyczne łączenie biblioteki DLL w C**

- Używanie bibliotek DLL - łączenie dynamiczne

- Używanie bibliotek DLL - łączenie dynamiczne
  - funkcje `LoadLibrary`, `GetProcAddress`

- Używanie bibliotek DLL - łączenie dynamiczne
  - funkcje `LoadLibrary`, `GetProcAddress`
  - znaczenie funkcji `DllMain`

- Używanie bibliotek DLL - łączenie dynamiczne
  - funkcje `LoadLibrary`, `GetProcAddress`
  - znaczenie funkcji `DllMain`
  - możliwość kontroli ładowania i usuwania biblioteki – `FreeLibrary`

- Używanie bibliotek DLL - łączenie dynamiczne
  - funkcje `LoadLibrary`, `GetProcAddress`
  - znaczenie funkcji `DllMain`
  - możliwość kontroli ładowania i usuwania biblioteki – `FreeLibrary`
  - przykład 3: **dynamiczne łączenie biblioteki DLL w C**

- Używanie bibliotek DLL - łączenie dynamiczne
  - funkcje `LoadLibrary`, `GetProcAddress`
  - znaczenie funkcji `DllMain`
  - możliwość kontroli ładowania i usuwania biblioteki – `FreeLibrary`
  - przykład 3: **dynamiczne łączenie biblioteki DLL w C**

- Używanie bibliotek DLL - API Widows

- Używanie bibliotek DLL - API Widows
  - różne wersje funkcji np. `MessageBoxW`(Unicode), `MessageBoxA` ANSI

- Używanie bibliotek DLL - API Widows
  - różne wersje funkcji np. `MessageBoxW`(Unicode), `MessageBoxA` ANSI
  - przykład 4: **wywoływanie funkcji API Windows w C**

- Używanie bibliotek DLL - API Widows
  - różne wersje funkcji np. `MessageBoxW`(Unicode), `MessageBoxA` ANSI
  - przykład 4: **wywoływanie funkcji API Windows w C**

- Używanie bibliotek DLL - łączenie z kodem .NET (C#)

- Używanie bibliotek DLL - łączenie z kodem .NET (C#)
  - dyrektywa `[DllImport(_dllLocation)]`, `extern`

- Używanie bibliotek DLL - łączenie z kodem .NET (C#)
  - dyrektywa `[DllImport(_dllLocation)]`, `extern`
  - przykład 5: **łączenie biblioteki DLL z programem w C#**

- Używanie bibliotek DLL - łączenie z kodem .NET (C#)
  - dyrektywa `[DllImport(_dllLocation)]`, `extern`
  - przykład 5: **łączenie biblioteki DLL z programem w C#**

- Używanie bibliotek DLL w Delphi (Object Pascal)

- Używanie bibliotek DLL w Delphi (Object Pascal)
  - słowa kluczowe `library`, `exports`, `stdcall`

- Używanie bibliotek DLL w Delphi (Object Pascal)
  - słowa kluczowe `library`, `exports`, `stdcall`
  - łączenie statyczne, dynamiczne

- Używanie bibliotek DLL w Delphi (Object Pascal)
  - słowa kluczowe `library`, `exports`, `stdcall`
  - łączenie statyczne, dynamiczne
  - przykład 6: **prosta biblioteka DLL w Delphi**

- Używanie bibliotek DLL w Delphi (Object Pascal)
  - słowa kluczowe `library`, `exports`, `stdcall`
  - łączenie statyczne, dynamiczne
  - przykład 6: **prosta biblioteka DLL w Delphi**
  - przykład 7: **prosta biblioteka DLL z Delphi w programie C#**

- Używanie bibliotek DLL w Delphi (Object Pascal)
  - słowa kluczowe `library`, `exports`, `stdcall`
  - łączenie statyczne, dynamiczne
  - przykład 6: **prosta biblioteka DLL w Delphi**
  - przykład 7: **prosta biblioteka DLL z Delphi w programie C#**

- Formularz Delphi w DLL

- Formularz Delphi w DLL
  - możliwość łączenia kodu starszych aplikacji w aplikacjach .NET

- Formularz Delphi w DLL
  - możliwość łączenia kodu starszych aplikacji w aplikacjach .NET
  - przykład 8: **biblioteka DLL z formularzem w Delphi**

- Formularz Delphi w DLL
  - możliwość łączenia kodu starszych aplikacji w aplikacjach .NET
  - przykład 8: **biblioteka DLL z formularzem w Delphi**
  - przykład 9: **formularz DLL z Delphi w programie C#**

- Formularz Delphi w DLL
  - możliwość łączenia kodu starszych aplikacji w aplikacjach .NET
  - przykład 8: **biblioteka DLL z formularzem w Delphi**
  - przykład 9: **formularz DLL z Delphi w programie C#**

- Case Study - wymiana danych pomiędzy różnymi platformami programowymi

- Case Study - wymiana danych pomiędzy różnymi platformami programowymi
  - dodatkowa warstwa pośrednicząca – funkcje eksportowane z przenośnymi parametrami

- Case Study - wymiana danych pomiędzy różnymi platformami programowymi
  - dodatkowa warstwa pośrednicząca – funkcje eksportowane z przenośnymi parametrami
  - przykład 10: **biblioteka DLL wyświetlająca wzory matematyczne w MathML**

- Case Study - wymiana danych pomiędzy różnymi platformami programowymi
  - dodatkowa warstwa pośrednicząca – funkcje eksportowane z przenośnymi parametrami
  - przykład 10: **biblioteka DLL wyświetlająca wzory matematyczne w MathML**
  - przykład 11: **program win32 używający biblioteki**

- Case Study - wymiana danych pomiędzy różnymi platformami programowymi
  - dodatkowa warstwa pośrednicząca – funkcje eksportowane z przenośnymi parametrami
  - przykład 10: **biblioteka DLL wyświetlająca wzory matematyczne w MathML**
  - przykład 11: **program win32 używający biblioteki**
  - przykład 12: **wersja .NET – użycie biblioteki w programie C#**

- Case Study - wymiana danych pomiędzy różnymi platformami programowymi
  - dodatkowa warstwa pośrednicząca – funkcje eksportowane z przenośnymi parametrami
  - przykład 10: **biblioteka DLL wyświetlająca wzory matematyczne w MathML**
  - przykład 11: **program win32 używający biblioteki**
  - przykład 12: **wersja .NET – użycie biblioteki w programie C#**

- DLL na platformie .NET

- DLL na platformie .NET
  - przykład 13: **tworzenie biblioteki**

- DLL na platformie .NET
  - przykład 13: **tworzenie biblioteki**
  - przykład 14: **wykorzystanie biblioteki**

- DLL na platformie .NET
  - przykład 13: **tworzenie biblioteki**
  - przykład 14: **wykorzystanie biblioteki**

- Pliki odwzorowane w pamięci

- Pliki odwzorowane w pamięci
  - funkcje `CreateFile`, `CreateFileMapping`, `MapViewOfFile`

- Pliki odwzorowane w pamięci
  - funkcje `CreateFile`, `CreateFileMapping`, `MapViewOfFile`
  - przykład 15: **tworzenie pliku mapowanego w pamięci**

- Pliki odwzorowane w pamięci
  - funkcje `CreateFile`, `CreateFileMapping`, `MapViewOfFile`
  - przykład 15: **tworzenie pliku mapowanego w pamięci**
  - przykład 16: **możliwość wymiany danych między procesami**

- Pliki odwzorowane w pamięci
  - funkcje `CreateFile`, `CreateFileMapping`, `MapViewOfFile`
  - przykład 15: **tworzenie pliku mapowanego w pamięci**
  - przykład 16: **możliwość wymiany danych między procesami**
  - sprawdzanie czy aplikacja została uruchomiona w jednym egzemplarzu

- Pliki odwzorowane w pamięci
  - funkcje `CreateFile`, `CreateFileMapping`, `MapViewOfFile`
  - przykład 15: **tworzenie pliku mapowanego w pamięci**
  - przykład 16: **możliwość wymiany danych między procesami**
  - sprawdzanie czy aplikacja została uruchomiona w jednym egzemplarzu

- optymalizacja pamięciowa zależna od optymalizatora wbudowanego w translator

- optymalizacja pamięciowa zależna od optymalizatora wbudowanego w translator
- wykorzystanie funkcji systemu operacyjnego

- optymalizacja pamięciowa zależna od optymalizatora wbudowanego w translator
- wykorzystanie funkcji systemu operacyjnego
- intensywna optymalizacja kodu celowa dla urządzeń wbudowanych, niektórych aplikacji internetowych

- optymalizacja pamięciowa zależna od optymalizatora wbudowanego w translator
- wykorzystanie funkcji systemu operacyjnego
- intensywna optymalizacja kodu celowa dla urządzeń wbudowanych, niektórych aplikacji internetowych

Dziękuję za uwagę

Następny temat:
projektowanie interfejsu użytkownika