Igor Dzierwa

RAPORT
Zarządzanie pamięcią, biblioteki, pomiar czasu

Zadanie 1. Kompilator, optymalizacja, pomiar czasu

-> Program 1: `loopfunction.c`

Wywoływanie funkcji w pętli

Flaga	Czas wykonania (s)
	28.039766
O	8.095300
O1	5.504928
O2	5.539241
O3	5.088925

*Program wykonuje iteracje, w każdej iteracji wywołuje funkcję potęgowania, której wyniki są sumowane.
Jego zoptymalizowana postać wykonuje się znacznie szybciej, natomiast różnice między flagami O1, O2 i O3 są niewielkie.

-> Program 2: `bubble.c`

Sortowanie bąbelkowe *(bubble sort)*

Flaga	Czas wykonania (s)
	24.903763
O	9.167446
O1	9.052896
O2	9.062837
O3	9.121642

*Program wykonuje sortowanie bąbelkowe tablicy o rozmiarze 100000, w której liczby całkowite ułożone są w ciągu malejącym *(najgorszy przypadek, gdy złożoność obliczeniowa wynosi O(n^2))*.
Jego zoptymalizowana postać wykonuje się znacznie szybciej, natomiast różnice między flagami O, O1, O2 i O3 są niewielkie.

-> Program 3: `recc.c`

Rekurencja ogonowa

Flaga	Czas wykonania (s)
	21.632954
O	15.842953
O1	16.095077
O2	16.780301
O3	15.550829

*Program oblicza rekurencyjnie ciąg fibonacciego. Liczba wywołań rośnie lawinowo wraz ze wzrostem liczby argumentów *(przyrost wykładniczy - czasochłonny dla dużej liczby argumentów)*.
Jego zoptymalizowana postać wykonuje się szybciej, występują nieznacznie większe różnice między flagami O, O1, O2, O3.

-> Program 4: `fibo.c`

Ciąg fibonacciego

Flaga	Czas wykonania (s)
	46.827064
O	3.877453
O1	3.879167
O2	3.874934
O3	3.838240

*Program oblicza iteracyjne ciąg fibonaccciego. Jego zoptymalizowana postać wykonuje się znacznie szybciej *(10-krotnie)*, natomiast różnice między flagami O, O1, O2, O3 są niewielkie.

-> Wniosek:

*Optymalizacja pozwala na wydłużenie wydajności programu, natomiast różnice pomiędzy flagami bywają niewielkie.

*Wybór optymalizacji powinien być przemyślany, pod kątem tego, jaki efekt chcemy uzyskać.

Zadanie 2. Program korzystający z bibliotek.

Wyodrębniona biblioteka *(`array_operations.h` i `array_operations.c`)* zawiera proste funkcje do operacji na tablicach:

*komunikat sygnalizuje załadowanie biblioteki:

```
void hello_pl(void);
```

*funkcja sprawdza czy tablica jest posortowana:

```
int checkingArraySorted(int arr[], int n);
```

*funkcja zamieniająca kolejność dwóch, leżących obok siebie elementów:

```
void swap(int *xp, int *yp);
```

*funkcja sortująca tablicę bąbelkowo:

```
void bubbleSort(int arr[], int n);
```

*funkcja wypisująca elementy tablicy:

```
void printArray(int arr[], int size);
```

-> Program testowy:

Wyświetla komunikat sygnalizujący załadowanie biblioteki (`void hello_pl(void)`) oraz przeprowadza sortowanie bąbelkowe tablicy o rozmiarze 100000, gdzie liczby całkowite ułożone są w ciągu malejącym.

-> Porównanie czasu działania poszczególnych programów, w zależności od rodzaju zastosowanej biblioteki:

Rodzaj biblioteki	Czas działania (s)
Biblioteka styczna *(static library)*	32.582824
Biblioteka współdzielona *(shared library)*	32.515283
Biblioteka dynamiczna *(dynamically loaded library)*	33.147013

-> *(*dodatkowe)* **Porównanie rozmiaru poszczególnych programów, w zależności od rodzaju zastosowanej biblioteki:**

Rodzaj biblioteki	Rozmiar pliku (KB)
Biblioteka styczna *(static library)*	13
Biblioteka współdzielona *(shared library)*	13
Biblioteka dynamiczna *(dynamically loaded library)*	2

-> Wnioski:

*Czas działania programu korzystającego z biblioteki statycznej jest nieznacznie gorszy od biblioteki współdzielonej.
Użycie bibliotek współdzielonych zamiast statycznych, może przynosić wiele korzyści, ze względu na fakt, że wszystkie programy mogą współdzielić dostęp do nich, jak również do ich różnych wersji.

*Czas działania programu korzystającego z biblioteki dynamicznej jest nieznacznie większy od czasu działania programu korzystającego z biblioteki statycznej oraz współdzielonej. Dużą korzyścią jest natomiast fakt zmniejszenia rozmiaru pliku (2KB - biblioteka dynamiczna, 13KB - biblioteka statyczna i współdzielona) - przy większych aplikacjach może mieć to duże znaczenie.