Programowanie II R

Zadania – seria 14.

Programowanie współbieżne i asynchroniczne.

Zadanie 1. `sumasync` – Współbieżne sumowanie elementów kolekcji.

Napisz szablony trzech funkcji

```
template <typename It>
auto Max1(It begin, It end)

template <typename It>
auto Max2(It begin, It end)

template <typename It>
auto Max3(It begin, It end)
```

zwracających sumę elementów zawartych pomiędzy dwoma iteratorami `begin` i `end`. Załóż, że iteratory wskazują na obiekty, dla których możliwe jest wykonanie operacji dodawania (tzn. istnieje przeciążony operator `+`); jeśli chcesz, możesz dodać koncept narzucający wykonalność dodawania.

Funkcje generowane na podstawie szablonu `Max1` powinny wykonywać swoje zadanie w sposób sekwencyjny, tzn. bez wykorzystania mechanizmu współbieżności, funkcje tworzone na podstawie szablonu `Max2` – współbieżnie, operując bezpośrednio na wątkach, zaś funkcje generowane w oparciu o szablon `Max3` – współbieżnie i asynchronicznie.

Korzystając z tych szablonów napisz program `sumasync`, który utworzy wektor liczb całkowitych lub rzeczywistych o odpowiednio dużym rozmiarze, wypełni go losowymi liczbami, a następnie trzykrotnie obliczy sumę jego elementów, posługując się jednym z przygotowanych przez Ciebie szablonów – za każdym razem innym – i dodatkowo mierząc czas wykonywania obliczeń w każdym przypadku. Program powinien na zakończenie wypisać na standardowe wyjście wynik i czas wykonywania każdego z trzech sumowań.

Wskazówka. Zamiast rozwiązywać zadanie od zera, możesz posłużyć się przykładami z notatek do zajęć, wprowadzając w nich odpowiednie modyfikacje.

Zadanie 2. `callspeed` – Czas sekwencyjnego i współbieżnego wywołania funkcji.

Napisz szablony trzech funkcji

```
template <typename F>
auto FCall1(F f, int n)

template <typename F>
auto FCall2(F f, int n)

template <typename F>
auto FCall3(F f, int n)
```

które przyjmują jako argumenty pewną funkcję o sygnaturze `void f(void)` oraz dodatnią liczbę całkowitą n. Funkcje tworzone na podstawie tych szablonów powinny n-krotnie wywołać funkcję przekazaną im jako pierwszy argument i zwrócić łączny czas trwania wszystkich n wykonań tej funkcji.

Funkcje generowane na podstawie szablonu `FCall1` powinny wykonywać swoje zadanie w sposób sekwencyjny, tzn. bez wykorzystania mechanizmu współbieżności, funkcje tworzone na podstawie szablonu `FCall2` – współbieżnie, operując bezpośrednio na wątkach, zaś funkcje generowane w oparciu o szablon `FCall3` – współbieżnie i asynchronicznie.

Przygotuj ponadto testową funkcję o sygnaturze `void f(void)` realizującą dowolne czasochłonne zadanie (np. tworzenie dużej listy liczb pseudolosowych i sortowanie jej za pomocą odpowiedniego algorytmu biblioteki standardowej, interakcję z systemem plików lub internetem).

Następnie napisz program `callspeed`, który wykona napisaną przez Ciebie testową funkcję na trzy sposoby, używając do tego za każdym razem innego z szablonów `FCall1`, `FCall2` i `FCall3` z pewną, wybraną przez Ciebie liczbą wykonań n, i wypisze na ekranie czas trwania tych wykonań w każdym przypadku. Poeksperymentuj z różnymi funkcjami testowymi.

Zadanie 3. `imgscraper` – Asynchroniczne pobieranie obrazów ze strony WWW.

Napisz program `imgscraper`, przyjmujący dwa argumenty wywołania: łańcuch tekstowy reprezentujący adres URL strony internetowej oraz łańcuch tekstowy reprezentujący ścieżkę do lokalnego folderu. Program powinien przeanalizować kod HTML strony o podanym adresie i odnaleźć w nim wszystkie obrazy (reprezentowane przez elementy `img`, adres pliku z obrazem jest wartością atrybutu `src`), a następnie pobrać każdy z nich i zapisać we wskazanym folderze.

Program powinien wykonywać operacje wejścia/wyjścia współbieżnie i asynchronicznie – dotyczy to komunikacji z analizowaną stroną internetową oraz pobierania i zapisywania plików obrazów.

Wskazówka. Przewodnik omawiający tworzenie programu analizującego kod strony internetowej z wykorzystaniem bibliotek *cpr* i *Gumbo* można znaleźć na przykład na stronie

`https://www.webscrapingapi.com/c-web-scraping#building-a-web-scraper-with-c++`

Przykład pobierania pliku za pomocą biblioteki *cpr* można znaleźć między innymi w dokumentacji tej biblioteki, pod adresem

`https://docs.libcpr.org/advanced-usage.html#download-file`

Opracowanie: Bartłomiej Zglinicki.