Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Laboratorium 5

Cel: Nabycie umiejętności tworzenia i implementacji programów równoległych w środowisku *Pthreads*

Zajęcia:

1. Utworzenie katalogu roboczego (np. *lab_5*), skopiowanie ze strony przedmiotu paczki *pthreads_suma.tgz,* rozpakowanie w podkatalogu (np. *zad_1*), analiza kodu tworzenia wątków i wykonywania operacji przez wątki
   a) w programie obliczania sumy występują dwa sposoby uzyskiwania wyniku ostatecznego na podstawie wyników pracy poszczególnych wątków
      - jednym jest wykorzystanie zmiennej globalnej (lub statycznej) i bezpieczne (w sekcji krytycznej) sumowanie wyników cząstkowych
      - drugim jest wpisanie wyników cząstkowych do tablicy i dokonanie sumowania przez wątek główny
      - rozwiązanie pierwsze jest prostsze (nie wymaga dodatkowych struktur danych) i bardziej uniwersalne
      - dwa warianty uzyskiwania wyniku - z sekcją krytyczną i bez sekcji krytycznej należy powtórzyć w samodzielnie tworzonym programie obliczania całki
   b) sens przeglądania plików z paczki *pthreads_suma.tgz* polega m.in. na możliwości skopiowania niektórych rozwiązań (fragmentów kodu) w drugiej części laboratorium
2. Uruchomienie programu (**w miejsce dostarczonej biblioteki *libpomiar_czasu.a* - skompilowanej w innym systemie - być może trzeba umieścić własną bibliotekę utworzoną w ramach laboratorium „pomiar_czasu”**)
   a) każdorazowo *(dotyczy to wszystkich programów w ramach wszystkich laboratoriów)* należy sprawdzać poprawność działania – wersja równoległa ma dawać identyczne wyniki jak wersja sekwencyjna (jeśli ze względu na sposób dekompozycji wyniki nieznacznie się różnią należy wyjaśnić skąd biorą się różnice)
   b) **sens obliczeń równoległych polega na skróceniu czasu realizacji zadania – każdorazowo *(dotyczy to wszystkich programów w ramach wszystkich laboratoriów)* należy sprawdzić czy udało się uzyskać w wersji równoległej skrócenie czasu działania w stosunku do wersji sekwencyjnej, jeśli użycie rosnącej liczby wątków/procesów nie powoduje skracania czasu działania programu należy wyjaśnić dlaczego tak się dzieje)**
   c) w przypadku prostego programu obliczania sumy wyrazów tablicy eksperymentem testującym sensowność użycia wersji równoległej może być porównanie czasu wykonania sekwencyjnego i równoległego (np. z użyciem 2 wątków) dla dwóch rozmiarów tablic: 1000 i 100 000 000 – kiedy zastosowanie wersji równoległej ma uzasadnienie? (wnioski powinny znaleźć się w sprawozdaniu) **(ocena)**

- *->Jak wygląda prosty wzorzec zrównoleglenia pętli? (podaj odpowiedni fragment procedury wątku i omów działanie wątku, zwracając uwagę na rodzaj dekompozycji pętli) Jakie uzyskuje się czasy obliczeń zwiększając liczbę wątków? Czy zależy to od wielkości zadania? Jaki można zaobserwować narzut związany z wykonaniem równoległym – w wersji z mutex'em i bez mutex'a? (narzut będzie szczególnie widoczny w przypadku małego zadania)*

3. Pobranie ze strony przedmiotu programu obliczania całki (w postaci paczki *pthreads_calka.tgz - w* paczce funkcja *main* oraz funkcje obliczania całki z dekompozycją pętli i dekompozycją obszaru są umieszczone w 3 różnych plikach). Rozpakowanie np. w nowym katalogu *zad2* i wstępne uruchomienie (**w miejsce dostarczonej biblioteki *libpomiar_czasu.a* - skompilowanej w innym systemie - być może trzeba umieścić własną bibliotekę utworzoną w ramach laboratorium „pomiar_czasu”**). Pliki źródłowe programu zawierają szkielet do testowania wariantów zrównoleglenia obliczania całki w zadanym przedziale metodą trapezów (dla ułatwienia sprawdzenia poprawności działania domyślnie obliczana jest całka z *sinus(x)* w przedziale [0,PI] – wynik powinien być przybliżeniem 2.0). Plik z funkcją *main* (sterującą wykonaniem całego programu i mierzącą czas poszczególnych wariantów) zawiera wersję sekwencyjną obliczania całki w funkcji *calka_sekw*, podobną do algorytmu:

```
calka=0.0; dx = (b-a)/N;
for(i=0;i<N;i++){
      x1=a+i*dx;
      calka+=0.5*(funkcja(x1)+funkcja(x1+dx))*dx;
}
```

uwaga: w kodzie każda wartość funkcji (z wyjątkiem skrajnych) jest liczona dwa razy, a mnożenie przez stałą podstawę odbywa się wielokrotnie w sumowaniu – kod nie jest optymalny

4. Kompilacja (*make*) i uruchomienie programu. Przetestowanie działania wariantu obliczeń sekwencyjnych dla różnych zadanych dokładności całkowania (różne wartości *dx* i wynikające z nich wartości *N* – uwaga: przy obliczaniu *N* dokonuje się zaokrąglenia (w efekcie *N*dx != (b-a)*), żeby otrzymać *dx* dokładnie odpowiadające *N*, należy je ponownie obliczyć – w kodzie obliczana jest nowa wartość *dx_adjust*, taka że *N*dx_adjust = (b-a)* – co odpowiada podziałowi przedziału [a,b] na N podprzedziałów). Sprawdzenie zbieżności wyniku do 2.0 w przypadku malejącej wysokości pojedynczego trapezu (np. *dx* = 0.1, 0.001, 0.00001 itd.) i rosnącej w związku z tym liczby trapezów (dla *dx*=0.000001 i *N*=3141593 wynik powinien mieć błąd mniejszy niż 10^{-12} – dwanaście cyfr znaczących) **(ocena)**
5. Uzupełnienie programu (plik *dekompozycja_petli.c* lub odpowiedni fragment pliku *pthreads_calka.c*) o wariant obliczania całki w sposób wielowątkowy (z wykorzystaniem środowiska *Pthreads*) wykorzystując **wzorzec zrównoleglenia pętli** (jest to wzorzec do wykonania automatycznego – nie jest potrzebna wiedza, że wykonuje się akurat całkowanie (zawartość wnętrza pętli z kodu sekwencyjnego można dosłownie przenieść do pętli w kodzie równoległym – dla zmiennych lokalnych))
   a) program główny wywołuje funkcję *calka_zrownoleglenie_petli* , która ma zwrócić wartość obliczonej całki. Funkcja *calka_zrownoleglenie_petli* ma utworzyć zadaną przez użytkownika liczbę wątków (liczba wątków ma być zadana w funkcji *main* - należy odkomentować odpowiedni fragment wczytywania liczby wątków lub ustawić ją na stałe, a następnie przesłać do funkcji *calka_zrownoleglenie_petli* jako argument). Funkcja *calka_zrownoleglenie_petli* ma zlecac wątkom wykonanie funkcji *calka_fragment_petli_w*, przesyłając jako argument identyfikator wątku i oczekiwać na zakończenie pracy wątków
   b) funkcja ma zrealizować wzorzec zrównoleglenia pętli, ostateczny wynik jest uzyskiwany przez redukcję wielowątkową, podobnie jak w *pthreads_suma.c* (uwaga na poprawne stosowanie zmiennych prywatnych i wspólnych – wariant w którym wątki otrzymują jako argument wykonywanej funkcji tylko swój identyfikator wymusza użycie wielu zmiennych wspólnych, np. globalnych, co można uznać za niekorzystne z punktu widzenia inżynierii oprogramowania)
   c) liczba iteracji pętli (*N*) w wariancie sekwencyjnym i równoległym (ze zrównolegleniem pętli) musi być identyczna, wysokość trapezów (*dx*) także jest identyczna – wynik wersji równoległej musi być identyczny z wersją sekwencyjną (z dokładnością bliską słowu maszynowemu – w praktyce ok. 10^{-12})
   d) dla celów dydaktycznych, w funkcji wątków, należy zastosować zmienne określające indeks początkowy iteracji, indeks końcowy iteracji oraz skok zmiennej sterującej pomiędzy iteracjami, obliczone na podstawie identyfikatora wątku. Należy zastosować wariant cykliczny dekompozycji pętli (dystrybucji iteracji), zapewniając równomierny podział iteracji między wątki (także w przypadku gdy liczba trapezów jest niepodzielna przez liczbę wątków) **(ocena)**
   e) po otrzymaniu poprawnego wyniku dla wariantu cyklicznego dekompozycji, należy zastosować dekompozycję blokową (najlepiej zakomentować podstawienie do zmiennych określających indeks początkowy iteracji, indeks końcowy iteracji oraz skok zmiennej sterującej pomiędzy iteracjami dla dekompozycji cyklicznej i utworzyć kod z podstawieniem do kolejnych zmiennych wartości odpowiednich dla dekompozycji blokowej. **(ocena)**
   - -> *Dlaczego wynik wersji równoległej według wzorca zrównoleglenia pętli jest (nieomal) identyczny z wynikiem wersji sekwencyjnej? Jakie można zauważyć wady wersji ze zrównolegleniem pętli?*

---------- 3.0 ----------

6. Uzupełnienie programu o wariant obliczania całki w sposób wielowątkowy (z wykorzystaniem środowiska *Pthreads*) wykorzystując **wzorzec dekompozycji w dziedzinie problemu**:
   a) program główny wywołuje funkcję *calka_dekompozycja_obszaru*, która ma zwrócić wartość obliczonej całki. Funkcja *calka_dekompozycja_obszaru* ma utworzyć zadaną przez użytkownika

liczbę wątków, zlecić im wykonanie funkcji *calka_podobszar_w* i czekać na zakończenie pracy wątków

b) idea działania funkcji *calka_dekompozycja_obszaru* polega na podziale obszaru obliczeniowego na podobszary i potraktowaniu całkowania w każdym podobszarze jako odrębnego zadania do wykonania przez wątek (zadania są wykonywane niezależnie – nie ma komunikacji, synchronizacji, muteksów, itp.)

c) każdy wątek otrzymuje zadanie wycałkowania funkcji w podobszarze: nie musi znać swojego identyfikatora (choć może), musi tylko znać dokładne parametry całkowania w podobszarze: współrzędnego lewego i prawego krańca podobszaru oraz zadaną dokładność w postaci wysokości pojedynczego trapezu *dx*
   - zadanie do wykonania przez wątek odpowiada dokładnie funkcji *calka_sekw* z odpowiednio dobranymi parametrami
   - wątek może wywołać funkcję calka_sekw, ewentualnie samodzielnie przeprowadzać obliczenia
   - w każdym z przypadków na podstawie *dx* obliczana jest liczba trapezów *N* (tym razem dla podprzedziału) i poprawiane *dx* – tak jak w programie *main* z *dx_adjust*, uwaga: czy poprawione wartości *dx* są takie same we wszystkich wątkach?)

d) w praktyce każdy wątek otrzymuje jako argument wykonywanej funkcji wskaźnik do odpowiedniej struktury z zadanymi parametrami wejściowymi, przeprowadza obliczenia i przekazuje wynik – np. może zapisać go w specjalnie zaprojektowanym polu struktury stanowiącej argument

e) jako zadanie do samodzielnego wykonania pozostaje poprawne zaimplementowanie przesłania wyniku całkowania z *calka_podobszar_w* do wiadomości *calka_dekompozycja_obszaru*, tak żeby funkcja *calka_dekompozycja_obszaru* mogła zwrócić ostateczny wynik (można wykorzystać sugerowaną postać struktury zawierającej definicję zadania dla konkretnego wątku) **(ocena)**

- *-> Jak wygląda kod w wersji dekompozycji w dziedzinie problemu (dekompozycji obszaru całkowania)? (podaj odpowiedni fragment funkcji wątku i omów go). Dlaczego wynik w wersji dekompozycji obszaru może różnić się od wyniku sekwencyjnego? (podpowiedź: jaki warunek spełniają parametry dokładności dx_adjust? - co z tego wynika?) Który ze sposobów przekazywania argumentów (w zmiennych globalnych dla dekompozycji pętli czy w strukturze będącej argumentem funkcji wątków) wydaje się być bardziej zgodny z zasadami bezpiecznego programowania?*

---------- 4.0 ----------

Dalsze kroki dla podniesienia oceny:

1. Wykorzystanie do zwracania wyniku z funkcji realizowanych przez wątki argumentu funkcji *pthread_exit*, który następnie odczytywany jest przez *pthread_join* (uwaga na rzutowanie typów).
   - sensowne wykorzystanie tej opcji zakłada dynamiczną alokację zmiennej, przesłanie jej adresu (uwaga funkcja main powinna zwolnić pamięć)
   - alternatywą jest posługiwanie się adresami zmiennych globalnych lub statycznych - wtedy jednak korzystanie ze zwracanej wartości nie jest uzasadnione (wątek główny i tak zna adres tych zmiennych)
2. Optymalizacja obliczania całki w pętli, np. tak żeby liczyć każdą wartość funkcji tylko raz, a dodatkowo zmniejszyć liczbę operacji wyciągając mnożenie przed sumowanie (modyfikacja np. dla wersji równoległej z dekompozycją obszaru)
   - optymalizacji można dokonać w funkcji *całka_sekw*, a następnie zmodyfikować kod, tak żeby korzystała z niego i wersja sekwencyjna, i wersja z dekompozycją obszaru
3. Dowolne zadania polecane przez prowadzących

---------- 5.0 ----------

Warunki zaliczenia:

- Obecność na zajęciach i wykonanie kroków 1-5
- Oddanie sprawozdania o treści i formie zgodnej z regulaminem ćwiczeń laboratoryjnych – z opisem zadania, kodem źródłowym programów i analizą wyników dla wszystkich wariantów (wydruki wyników wklejone jako obrazy z identyfikacją osoby przeprowadzającej obliczenia – zgodnie z regulaminem laboratoriów) oraz odpowiednimi wnioskami.
- Symbol -> oznacza pytania, na które odpowiedzi ma dać laboratorium (odpowiedzi powinny znaleźć się w sprawozdaniu – najlepiej w punktach odpowiadających pytaniom)