

Programowanie równoległe.

Przetwarzanie równoległe i rozproszone. Laboratorium 3

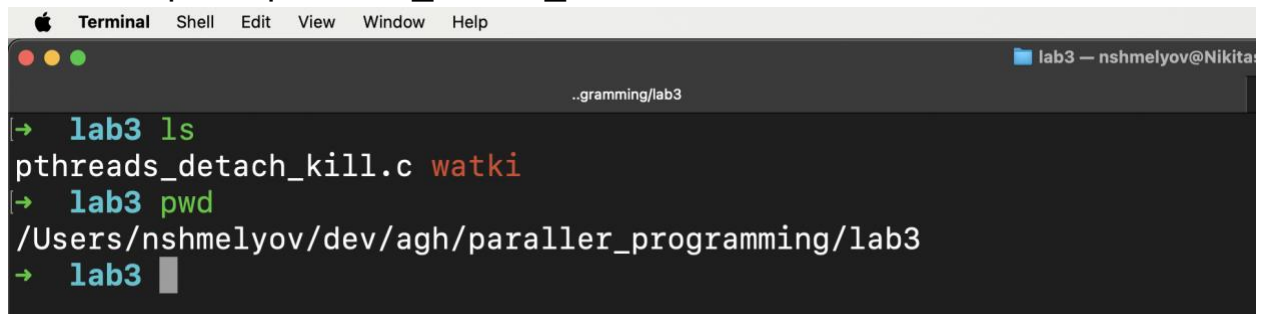
Wykonał: Mikita Shmialiou

Cel:

- nabycie praktycznej umiejętności manipulowania wątkami Pthreads – tworzenia, niszczenia, elementarnej synchronizacji
- przetestowanie mechanizmu przesyłania argumentów do wątku
- poznanie funkcjonowania obiektów określających atrybuty wątków.

Wykonanie ćwiczenia

1. Pobranie pliku „pthread_detach_kill.c”



```
Terminal  Shell  Edit  View  Window  Help
lab3 — nshmelyov@Nikita
..gramming/lab3
→ lab3 ls
pthread_detach_kill.c  watki
→ lab3 pwd
/Users/nshmelyov/dev/agh/paraller_programming/lab3
→ lab3
```

2. Uzupełnienie

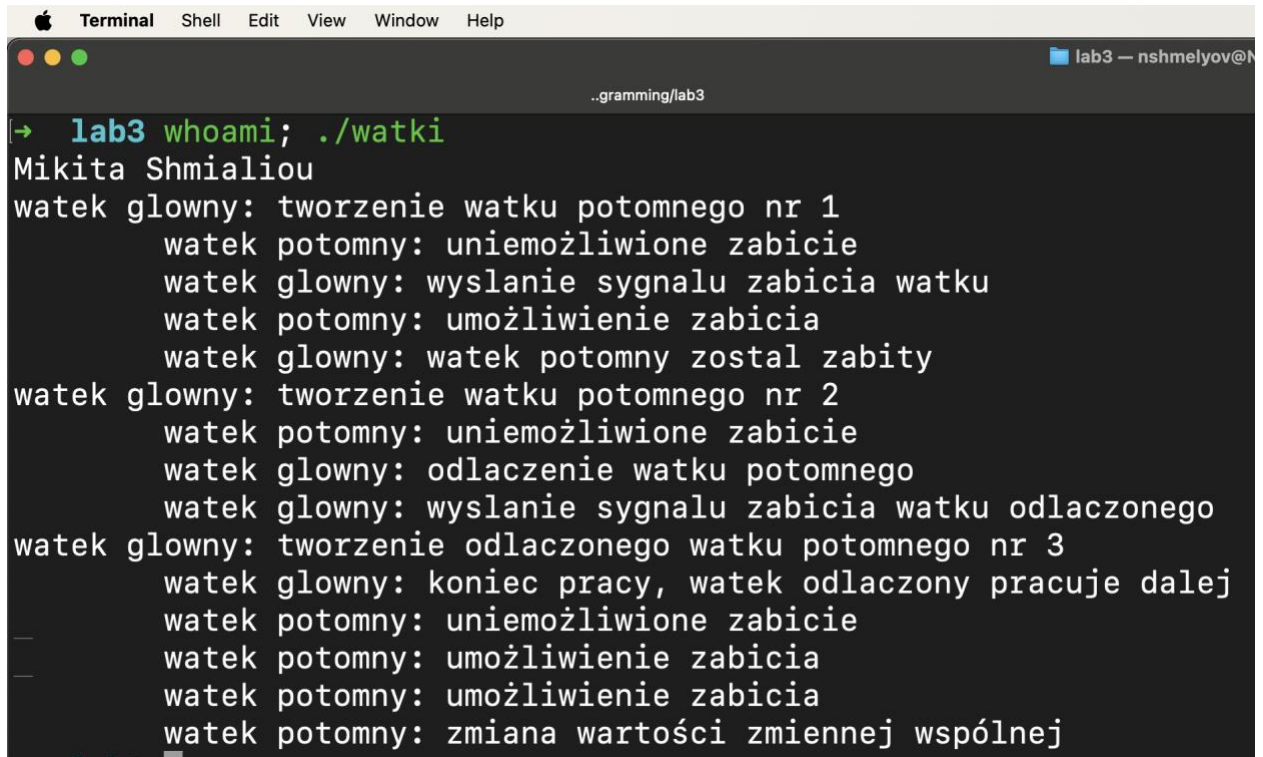
kodu

programu

```
53 // Tu wstaw kod tworzenia wątku z domyślnymi właściwościami
54 pthread_create(&tid, NULL, zadanie_watku, NULL);
55
56 sleep(2); // czas na uruchomienie wątku
57
58 printf("\twatek główny: wysłanie sygnału zabicia watku\n");
59 pthread_cancel(tid);
60
61 // Co należy zrobić przed sprawdzeniem czy wątki się skończyły?
62 pthread_join(tid, &wynik); // trzeba dołączyć wątek, żeby poznać jego status
63
64 if (wynik == PTHREAD_CANCELED)
65     printf("\twatek główny: watek potomny został zabity\n");
66 else
67     printf("\twatek główny: watek potomny NIE został zabity - błąd\n");
68
69 // Odłączanie wątku
70 zmienna_wspolna = 0;
71
72 printf("watek główny: tworzenie watku potomnego nr 2\n");
73
74 // Tu wstaw kod tworzenia wątku z domyślnymi właściwościami
75 pthread_create(&tid, NULL, zadanie_watku, NULL);
76
77 sleep(2); // czas na uruchomienie wątku
78
79 printf("\twatek główny: odłączenie watku potomnego\n");
80
81 // Instrukcja odłączenia?
82 pthread_detach(tid); // odłączenie wątku
83
84 printf("\twatek główny: wysłanie sygnału zabicia watku odłączonego\n");
85 pthread_cancel(tid);
86
87 // Czy wątek został zabity? Jak to sprawdzić?
88 // nie można sprawdzić statusu odłączonego wątku
89
90 // Wątek odłączony
91 // Inicjacja atrybutów?
92 pthread_attr_init(&attr); // inicjalizacja atrybutów
93
94 // Ustawianie typu wątku na odłączony
95 pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
96
97 printf("watek główny: tworzenie odłączonego watku potomnego nr 3\n");
98 pthread_create(&tid, &attr, zadanie_watku, NULL); // utworzenie odłączonego wątku
99
100 // Niszczenie atrybutów
101 pthread_attr_destroy(&attr);
102
103 printf("\twatek główny: koniec pracy, watek odłączony pracuje dalej\n");
104 pthread_exit(NULL); // co stanie się gdy użyjemy exit(0)?
105 // exit(0) zakończy cały proces natychmiast, nawet jeśli wątki jeszcze działają
106 }
107
```

3. Testowanie

programu



```
Terminal Shell Edit View Window Help
lab3 — nshmelyov@N
..gramming/lab3
[→ lab3 whoami; ./watki
Mikita Shmialiou
watek glowny: tworzenie watku potomnego nr 1
    watek potomny: uniemożliwienie zabicie
    watek glowny: wysłanie sygnału zabicia watku
    watek potomny: umożliwienie zabicia
    watek glowny: watek potomny został zabity
watek glowny: tworzenie watku potomnego nr 2
    watek potomny: uniemożliwienie zabicie
    watek glowny: odłączenie watku potomnego
    watek glowny: wysłanie sygnału zabicia watku odłączonego
watek glowny: tworzenie odłączonego watku potomnego nr 3
    watek glowny: koniec pracy, watek odłączony pracuje dalej
    watek potomny: uniemożliwienie zabicie
    watek potomny: umożliwienie zabicia
    watek potomny: umożliwienie zabicia
    watek potomny: zmiana wartości zmiennej wspólnej
```

4. Pytania:

1. W jakich dwóch trybach mogą funkcjonować wątki Pthreads?
 - Joinable (łąchalne / dołączalne)
 - Detached (odłączone)
2. Jaka jest różnica między tymi trybami?

Odłączone wątki działają niezależnie od innych wątków i nie mogą czekać na inne wątki. Czekanie na inne wątki jest realizowane za pomocą funkcji `pthread_join()` (więcej później). Wątki dołączalne mogą wywołać `pthread_join()`, aby czekać na inny wątek.

3. Kiedy wątek standardowo kończy swoje działanie?

Wątek standardowo kończy swoje działanie po zakończeniu procedury (funkcji) punktu wejścia, czyli po wykonaniu wszystkich instrukcji, które zostały dla niego zdefiniowane.

4. W jaki sposób można wymusić zakończenie działania wątku? (czym różnią się w tym przypadku wątki odłączone i standardowe?)

Wątek może zakończyć działanie w dowolnym momencie, wywołując podprogram `pthread_exit`. Wywołanie podprogramu `exit` kończy cały proces, włączając wszystkie wątki.

Joinable: Można użyć `pthread_join` po anulowaniu, żeby potwierdzić zakończenie i odzyskać kod.

Detached: Nie da się sprawdzić rezultatu anulowania — po zakończeniu wątek sam czyści zasoby.

5. Jak wątek może chronić się przed próbą "zabicia"? Jak można sprawdzić czy próba "zabicia" wątku powiodła się? (czym różnią się w tym przypadku wątki odłączone i standardowe?)

Najprostszy i główny sposób, w jaki wątek może chronić się przed anulowaniem (próbą "zabicia"), to tymczasowe wyłączenie możliwości anulowania za pomocą funkcji:

```
pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL)
```

Wątek wywołuje tę funkcję przed wejściem do krytycznego fragmentu kodu, a po jego zakończeniu przywraca możliwość anulowania przez:

```
pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL)
```

Gdy anulowanie jest wyłączone, żądania `pthread_cancel` są wstrzymywane i nie przerywają działania wątku.

5. Zaprojektowanie i utworzenie nowej procedury tworzenia wątków

Kod:

```
zad_2 > C main.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #define MAX_WATKI 100
6
7  void* zadanie_watku(void* arg) {
8      int identyfikator = *(int*)arg;
9      pthread_t system_id = pthread_self();
10     printf("Watek ID: %d, pthread_self: %lu\n", identyfikator, system_id);
11     return NULL;
12 }
13
14 int main(int argc, char* argv[]) {
15     if (argc != 2) {
16         fprintf(stderr, "Użycie: %s <liczba_watkow>\n", argv[0]);
17         exit(EXIT_FAILURE);
18     }
19
20     int liczba_watkow = atoi(argv[1]);
21     if (liczba_watkow <= 0 || liczba_watkow > MAX_WATKI) {
22         fprintf(stderr, "Błąd: liczba wątków powinna być z zakresu 1..%d\n", MAX_WATKI);
23         exit(EXIT_FAILURE);
24     }
25
26     pthread_t watki[MAX_WATKI];
27     int identyfikatory[MAX_WATKI];
28
29     for (int i = 0; i < liczba_watkow; ++i) {
30         identyfikatory[i] = i;
31         if (pthread_create(&watki[i], NULL, zadanie_watku, &identyfikatory[i]) != 0) {
32             perror("pthread_create");
33             exit(EXIT_FAILURE);
34         }
35     }
36
37     for (int i = 0; i < liczba_watkow; ++i) {
38         pthread_join(watki[i], NULL);
39     }
40
41     printf("Wszystkie wątki zakończyły działanie.\n");
42
43     return 0;
44 }
```

Testowanie:

```
Terminal  Shell  Edit  View  Window  Help
..gramming/lab3
[→ lab3 whoami; ./zad_2/watki
Mikita Shmialiou
Użycie: ./zad_2/watki <liczba_watkow>
[→ lab3 whoami; ./zad_2/watki 10
Mikita Shmialiou
Watek ID: 0, pthread_self: 6133411840
Watek ID: 3, pthread_self: 6135132160
Watek ID: 1, pthread_self: 6133985280
Watek ID: 4, pthread_self: 6135705600
Watek ID: 2, pthread_self: 6134558720
Watek ID: 5, pthread_self: 6136279040
Watek ID: 6, pthread_self: 6136852480
Watek ID: 7, pthread_self: 6137425920
Watek ID: 8, pthread_self: 6137999360
Watek ID: 9, pthread_self: 6138572800
Wszystkie wątki zakończyły działanie.
[→ lab3
```

Jak poprawnie przekazać identyfikator do wątku?

Poprawne przekazywanie odbywa się przez przekazanie wskaźnika do oddzielnej zmiennej dla każdego wątku – najczęściej tablicy `int identyfikatory[N]`, gdzie `&identyfikatory[i]` jest unikalne dla każdego wątku.

Sprawozdanie: Równoległe obliczanie całki metodą trapezów

Wykonał: Mikita Shmialiou

1. Cel ćwiczenia

Celem laboratorium było zaimplementowanie programu obliczającego całkę oznaczoną funkcji sinus w przedziale $[0, \pi]$ metodą trapezów, wykorzystując trzy różne podejścia:

- Wersja sekwencyjna - obliczenia na jednym wątku
- Zrównoleglenie pętli - podział iteracji między wątki
- Dekompozycja obszaru - podział przedziału całkowania między wątki

2. Metodyka

2.1. Algorytm całkowania

Wykorzystano metodę trapezów:

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

2.2. Implementacja równoległa

Zrównoleglenie pętli:

- Każdy wątek oblicza sumę cząstkową dla przypisanych iteracji
- Wyniki sumowane są z użyciem mutexa

Dekompozycja obszaru:

- Każdy wątek oblicza całkę w swoim podprzedziale
- Wyniki sumowane są bez konieczności synchronizacji

3. Wyniki

3.1. Testy dla $dx = 0.1$

Metoda	Czas wykonania [s]	Wynik
Sekwencyjna	0.000033	1.998393360970144
Zrównoleglenie pętli (4w)	0.000292	1.998393360970145
Dekompozycja obszaru (4w)	0.000157	1.998393360970145

3.2. Testy dla $dx = 0.000001$

Metoda	Czas wykonania [s]	Wynik
Sekwencyjna	0.029032	1.999999999999758
Zrównoleglenie pętli (4w)	0.008087	1.999999999999837
Dekompozycja obszaru (4w)	0.007473	1.999999999999847

4. Analiza

4.1. Poprawność wyników

Wszystkie metody zwracają identyczne wyniki, co potwierdza poprawność implementacji. Dla $dx = 0.000001$ wynik jest bardzo bliski teoretycznej wartości 2.0.

4.2. Wydajność

Przyspieszenie: Dla 4 wątków osiągnięto przyspieszenie $\sim 3.5x$

Różnice między metodami:

- Dekompozycja obszaru jest nieco szybsza od zrównoleglenia pętli

- Różnica wynika z braku konieczności synchronizacji w metodzie dekompozycji

4.3. Wnioski

- Dla dużych N (małe dx) równoległość daje znaczące przyspieszenie
- Dekompozycja obszaru jest bardziej efektywna dla całkowania numerycznego
- Im większa liczba trapezów, tym większy zysk z równoległości