Ministerstwo Nauki
i Szkolnictwa Wyższego

Regionalna Inicjatywa Doskonałości w Dyscyplinach
Informatyki, Elektrotechniki, Elektroniki, Automatyki i Robotyki
na Politechnice Częstochowskiej

Projekt w ramach programu Regionalna Inicjatywa Doskonałości, decyzja nr 020/RID/2018/19

Programowanie równoległe z wykorzystaniem współczesnych architektur komputerowych z pamięcią współdzieloną

Dr hab. inż. Łukasz Szustak
lszustak@icis.pcz.pl

Dr inż. Kamil Halbiniak
khalbniak@icis.pcz.pl

Politechnika Częstochowska

Programowanie równoległe z wykorzystaniem współczesnych architektur komputerowych z pamięcią współdzieloną

1. Wprowadzenie do współczesnych architektur komputerowych oraz programowania równoległego
2. Przegląd modeli i standardów programowania równoległego z pamięcią współdzieloną
3. Środowisko OpenMP programowania maszyn z pamięcią wspólna, w tym procesorów wielordzeniowych
4. Zrównoleglanie pętli oraz równoważenie obciążenia w systemach równoległych z pamięcią współdzieloną
5. Zrównoleglanie programów z wykorzystaniem modelu typu data parallelism
6. Zrównoleglanie programów z wykorzystaniem modelu typu task parallelism
7. Mapowanie obliczeń w hybrydowych systemach komputerowych z akceleratorami
8. Przykłady zrównoleglania obliczeń numerycznych

Programowanie równoległe z wykorzystaniem współczesnych architektur komputerowych z pamięcią współdzieloną

1. Wprowadzenie do współczesnych architektur komputerowych oraz programowania równoległego
2. **Przegląd modeli i standardów programowania równoległego z pamięcią współdzieloną**
3. Środowisko OpenMP programowania maszyn z pamięcią wspólna, w tym procesorów wielordzeniowych
4. Zrównoleglanie pętli oraz równoważenie obciążenia w systemach równoległych z pamięcią współdzieloną
5. Zrównoleglanie programów z wykorzystaniem modelu typu data parallelism
6. Zrównoleglanie programów z wykorzystaniem modelu typu task parallelism
7. Mapowanie obliczeń w hybrydowych systemach komputerowych z akceleratorami
8. Przykłady zrównoleglania obliczeń numerycznych

Ministerstwo Nauki
i Szkolnictwa Wyższego

Regionalna Inicjatywa Doskonałości w Dyscyplinach
Informatyki, Elektrotechniki, Elektroniki, Automatyki i Robotyki
na Politechnice Częstochowskiej

Projekt w ramach programu Regionalna Inicjatywa Doskonałości, decyzja nr 020/RID/2018/19

Temat nr 2

Przegląd modeli i standardów programowania równoległego z pamięcią współdzieloną

Dr inż. Kamil Halbiniak
khalbiniak@icis.pcz.pl

Politechnika Częstochowska

Agenda

- Przegląd modeli programowania równoległego:
  - Zrównoleglenie na poziomie danych
  - Zrównoleglenie na poziomie zadań
- Przegląd wybranych standardów programowania równoległego z pamięcią współdzieloną:
  - POSIX Threads (pThreads)
  - C++ Threads
  - OpenMP

Modele programowania równoległego

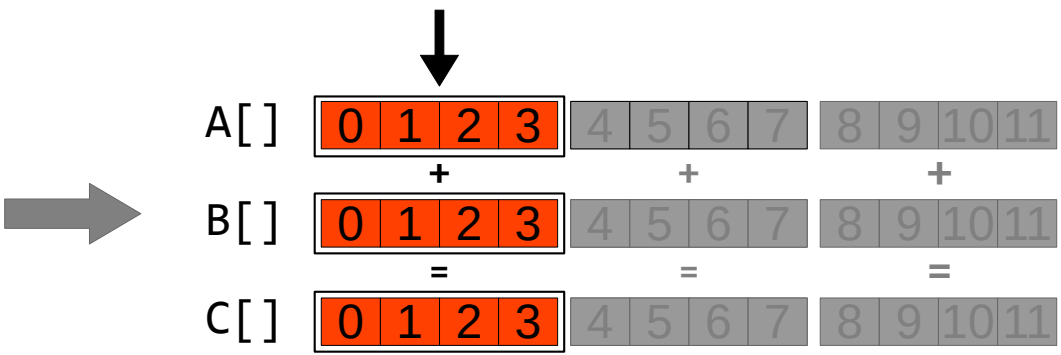
Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wektorowość
  - Wielordzeniowość

Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wektorowość
    - Zakłada wykorzystanie jednostek wektorowych
    - Pozwala na wykonanie w pojedynczym cyklu zegara tej samej operacji z wykorzystaniem określonej liczby danych
    - Ilość jednocześnie przetwarzanych danych zależy od rodzaju instrukcji wektorowych

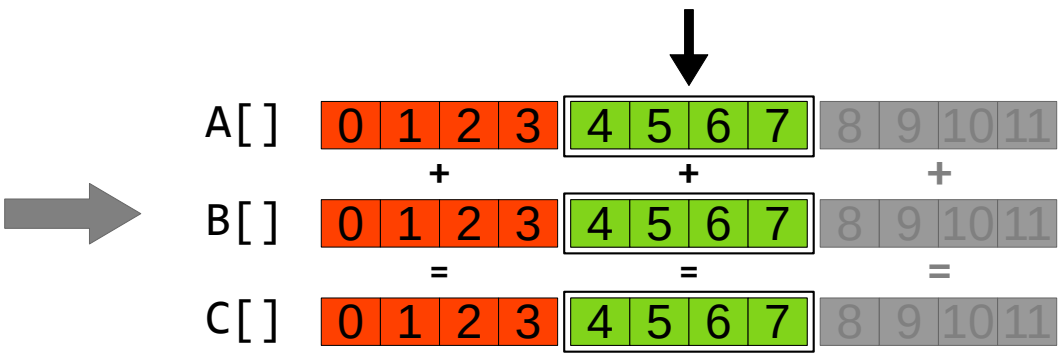
```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```



Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wektorowość
    - Zakłada wykorzystanie jednostek wektorowych
    - Pozwala na wykonanie w pojedynczym cyklu zegara tej samej operacji z wykorzystaniem określonej liczby danych
    - Ilość jednocześnie przetwarzanych danych zależy od rodzaju instrukcji wektorowych

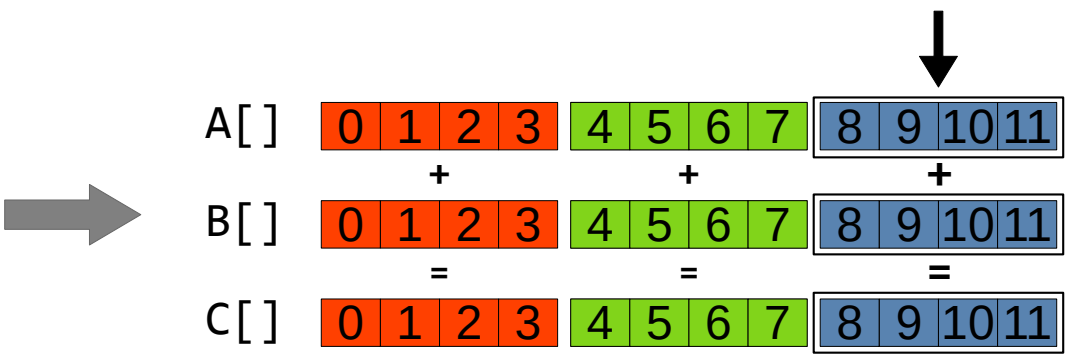
```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```



Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wektorowość
    - Zakłada wykorzystanie jednostek wektorowych
    - Pozwala na wykonanie w pojedynczym cyklu zegara tej samej operacji z wykorzystaniem określonej liczby danych
    - Ilość jednocześnie przetwarzanych danych zależy od rodzaju instrukcji wektorowych

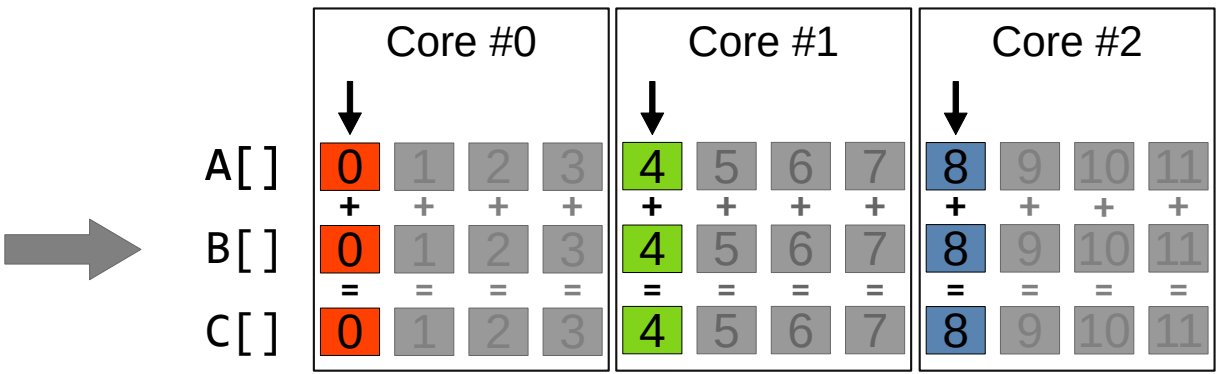
```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```



Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wielordzeniowość
    - Umożliwia wykonywanie tego samego zestawu operacji na różnych danych przez wiele rdzeni jednocześnie

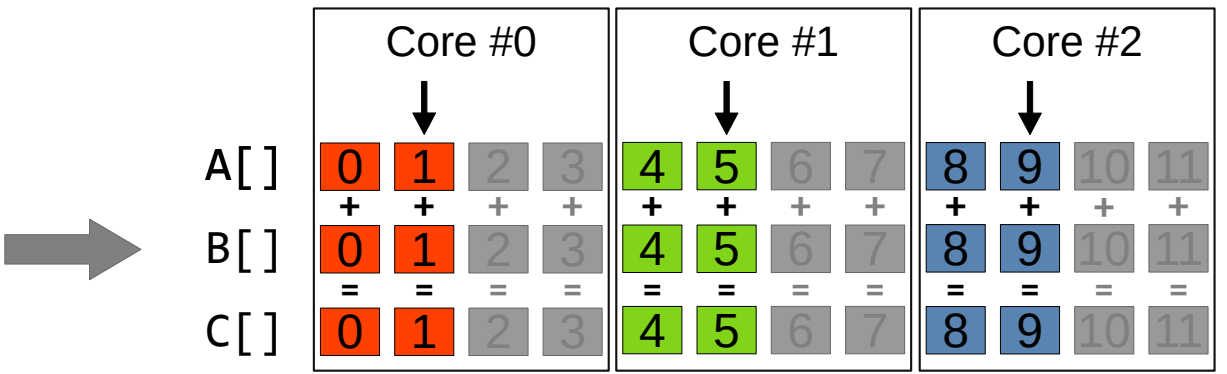
```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```



Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wielordzeniowość
    - Umożliwia wykonywanie tego samego zestawu operacji na różnych danych przez wiele rdzeni jednocześnie

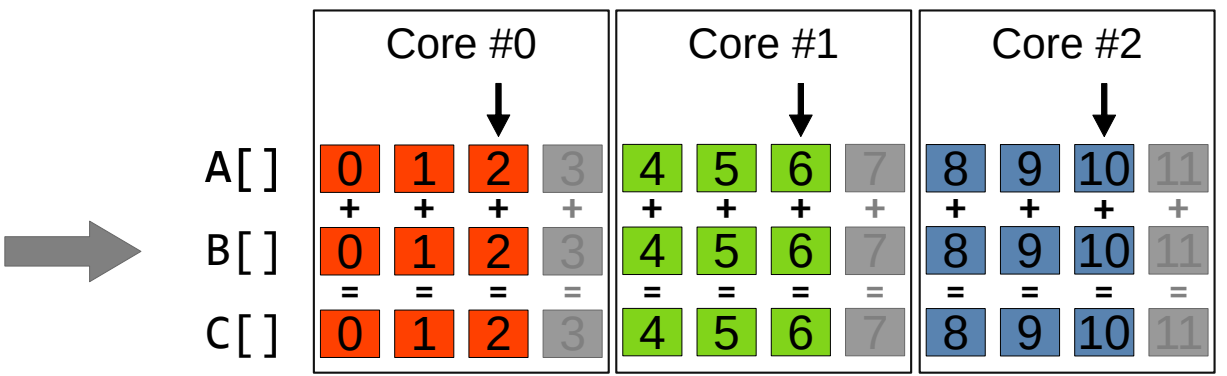
```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```



Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wielordzeniowość
    - Umożliwia wykonywanie tego samego zestawu operacji na różnych danych przez wiele rdzeni jednocześnie

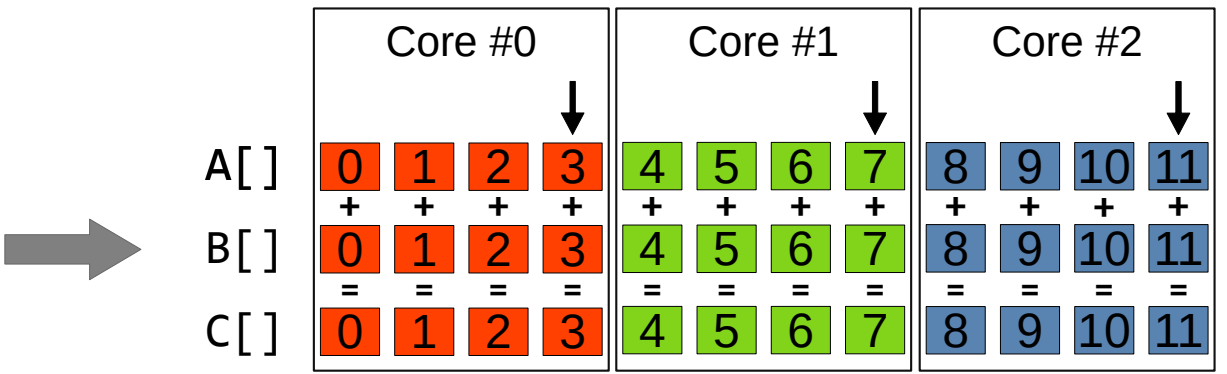
```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```



Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wielordzeniowość
    - Umożliwia wykonywanie tego samego zestawu operacji na różnych danych przez wiele rdzeni jednocześnie

```
for(int i=0; i<12; ++i)
{
   C[i] = A[i] + B[i]
}
```

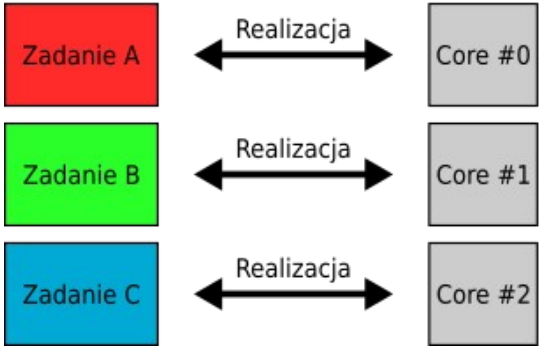


Zrównoleglenie na poziomie danych

- Zrównoleglenie na poziomie danych (ang. *data level parallelism*) zakłada, że ten sam zestaw instrukcji wykonywany jest jednocześnie dla wielu danych
- Równoległość danych wyrazić można poprzez:
  - Wektorowość
    - Zakłada wykorzystanie jednostek wektorowych
    - Pozwala na wykonanie w pojedynczym cyklu zegara tej samej operacji z wykorzystaniem określonej liczby danych
    - Ilość jednocześnie przetwarzanych danych zależy od rodzaju instrukcji wektorowych
  - Wielordzeniowość
    - Umożliwia wykonywanie tego samego zestawu operacji na różnych danych przez wiele rdzeni jednocześnie

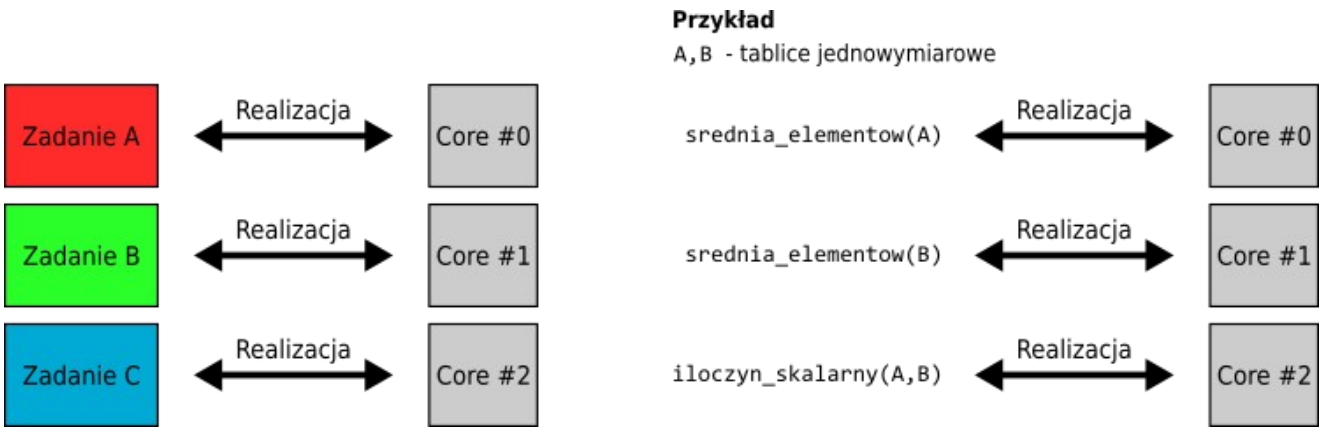
Zrównoleglenie na poziomie zadań

- Zrównoleglenie na poziomie zadań (ang. *task level parallelism)* opiera się na dystrybucji zadań, które wykonywane są równocześnie przez wiele rdzeni lub procesorów
- Równoległość zadań umożliwia wykonanie wielu różnych zadań na tych samych danych (bądź różnych)
- Wadą tutaj, jest fakt, iż rzadko pozwala ono osiągnąć przyspieszenie wykonania programu, jakie można osiągnąć dla zrównoleglenia na poziomie danych



Zrównoleglenie na poziomie zadań

- Zrównoleglenie na poziomie zadań (ang. *task level parallelism)* opiera się na dystrybucji zadań, które wykonywane są równocześnie przez wiele rdzeni lub procesorów
- Równoległość zadań umożliwia wykonanie wielu różnych zadań na tych samych danych (bądź różnych)
- Wadą tutaj, jest fakt, iż rzadko pozwala ono osiągnąć przyspieszenie wykonania programu, jakie można osiągnąć dla zrównoleglenia na poziomie danych



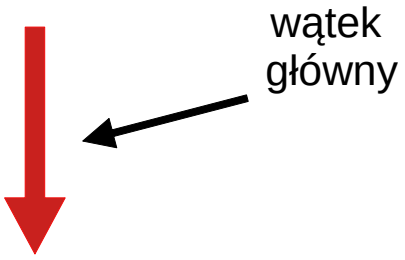
Standardy programowania równoległego

Model *fork-join*

- Model *fork-join* definiuje sposób wykonania programów równoległych

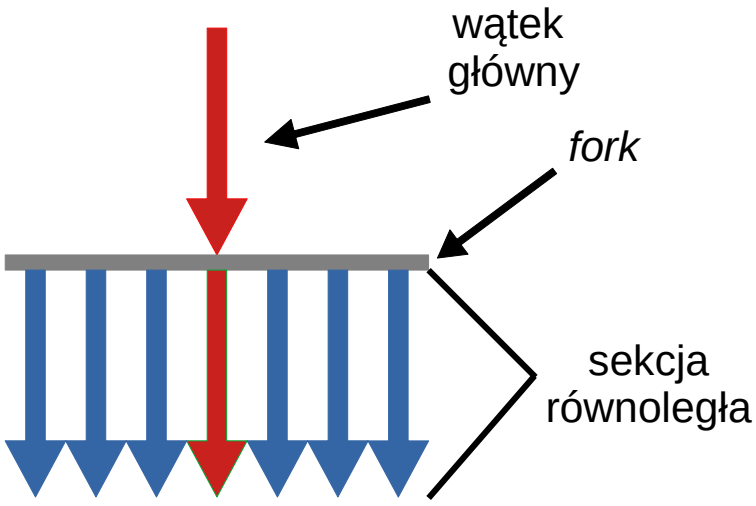
Model *fork-join*

- Model *fork-join* definiuje sposób wykonania programów równoległych
- Program początkowo wykonywany jest sekwencyjne przez wątek główny



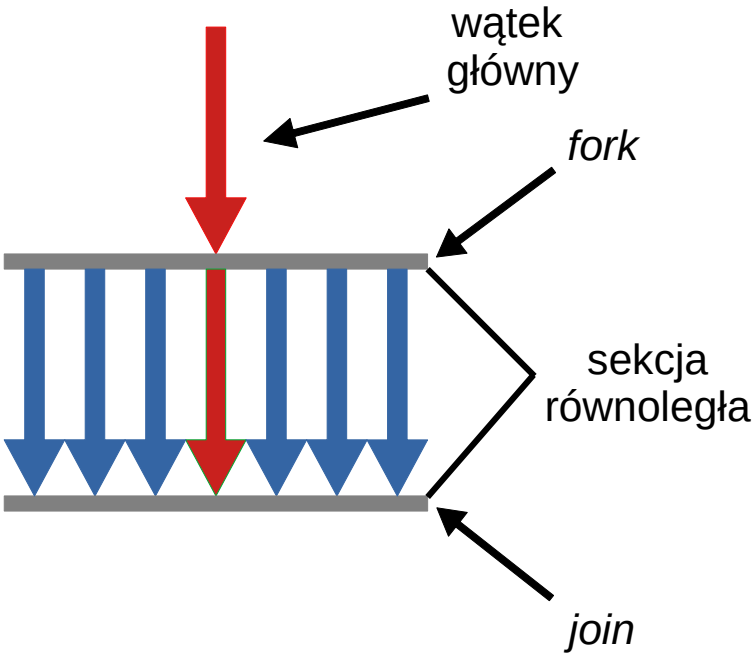
Model *fork-join*

- Model *fork-join* definiuje sposób wykonania programów równoległych
- Program początkowo wykonywany jest sekwencyjne przez wątek główny
- W momencie konieczności równoległej realizacji zadań tworzone są nowe wątki (*fork*)



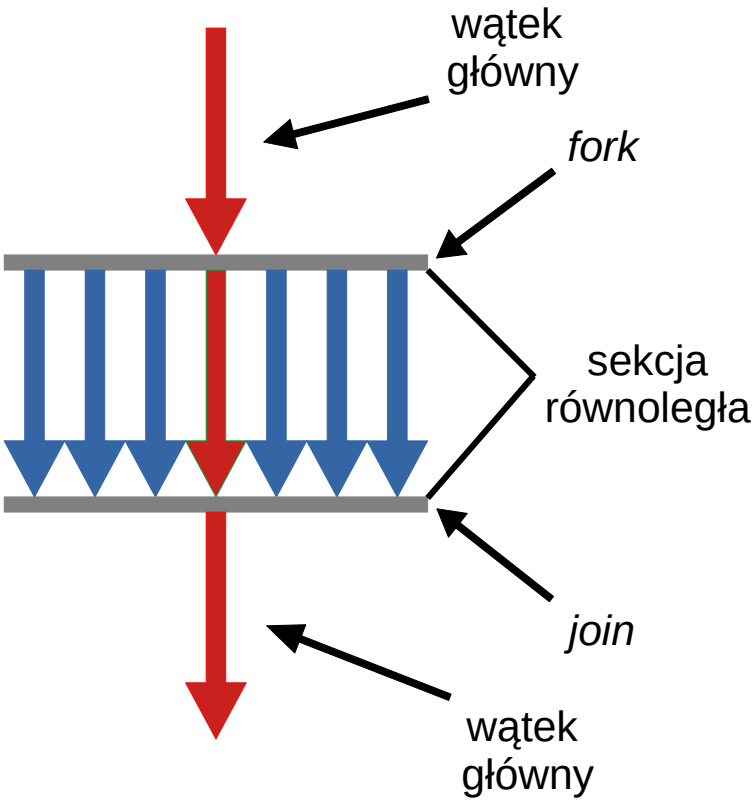
Model *fork-join*

- Model *fork-join* definiuje sposób wykonania programów równoległych
- Program początkowo wykonywany jest sekwencyjne przez wątek główny
- W momencie konieczności równoległej realizacji zadań tworzone są nowe wątki (*fork*)
- Po zakończeniu zadań realizowanych równolegle wszystkie wątki są dołączane do wątku głównego (*join*)



Model *fork-join*

- Model *fork-join* definiuje sposób wykonania programów równoległych
- Program początkowo wykonywany jest sekwencyjne przez wątek główny
- W momencie konieczności równoległej realizacji zadań tworzone są nowe wątki (*fork*)
- Po zakończeniu zadań realizowanych równolegle wszystkie wątki są dołączane do wątku głównego (*join*)
- Następnie program ponownie wykonywany jest w sposób sekwencyjny przez wątek główny



Standard POSIX Threads

- Zestaw funkcji dotyczący wątków zdefiniowany przez normę POSIX P1003.4a i nosi nazwę **pThreads** (skrót od **POSIX Threads**)
- Jest to zbiór typów i funkcji języków C oraz C++
- Standard pozwala na tworzenie aplikacji równoległych w systemach Linux
- W pThreads nazewnictwo oraz kolejność zmiennych na liście argumentów funkcji cechują się niezmiennym schematem
  - Nazwy funkcji oraz typów rozpoczynają się przedrostkiem **pthread_**
  - W przypadku stałych jest przedrostek ten pisany jest dużymi literami

Programowanie wątków POSIX

- Typy i funkcje POSIX zdefiniowane są w następującym pliku nagłówkowym:

```
#include <pthread.h>
```

- Kod programu należy skompilować z flagą `-pthread`

```
g++ -pthread -o prog prog.cpp
```

Tworzenie wątków

- Do uruchamiania wątków w standardzie pThreads możliwe jest przy użyciu dedykowanej do tego celu funkcji `pthread_create`:

```
int pthread_create(
    pthread_t *thread,
    pthread_attr_t *attr,
    void *(*thread_function)(void *),
    void *arg
);
```

Identyfikator utworzonego wątku

- Funkcja zwraca wartość **zero** jeśli operacja zakończyła się poprawnie

Tworzenie wątków

- Do uruchamiania wątków w standardzie pThreads możliwe jest przy użyciu dedykowanej do tego celu funkcji `pthread_create`:

```
int pthread_create(
    pthread_t *thread,
    pthread_attr_t *attr,
    void *(*thread_function)(void *),
    void *arg
);
```

Identyfikator utworzonego wątku

Atrybuty wątku

- Funkcja zwraca wartość **zero** jeśli operacja zakończyła się poprawnie

Tworzenie wątków

- Do uruchamiania wątków w standardzie pThreads możliwe jest przy użyciu dedykowanej do tego celu funkcji `pthread_create`:

```
int pthread_create(
    pthread_t *thread,
    pthread_attr_t *attr,
    void *(*thread_function)(void *),
    void *arg
);
```

Identyfikator utworzonego wątku

Atrybuty wątku

Wskaźnik do funkcji wykonywanej przez wątek

- Funkcja zwraca wartość **zero** jeśli operacja zakończyła się poprawnie

Tworzenie wątków

- Do uruchamiania wątków w standardzie pThreads możliwe jest przy użyciu dedykowanej do tego celu funkcji `pthread_create`:

```
int pthread_create(
    pthread_t *thread,
    pthread_attr_t *attr,
    void *(*thread_function)(void *),
    void *arg
);
```

Identyfikator utworzonego wątku

Atrybuty wątku

Wskaźnik do funkcji wykonywanej przez wątek

Argument przekazywany do funkcji

- Funkcja zwraca wartość **zero** jeśli operacja zakończyła się poprawnie

Oczekiwanie na zakończenie zadań

- Zakończenie pracy wątków wymaga jawnego wywołania funkcji `pthread_join`

```
int pthread_join(
   pthread_t thread,
   void **results
);
```

- Funkcja ta wywoływana jest na rzecz każdego utworzonego wątku
- Wywołanie funkcji `pthread_join` wstrzymuje realizację programu przez wątek główny do czasu zakończenia zadań w ramach wątku, na rzecz, którego została wywołana
- Funkcja zwraca wartość **zero** jeśli operacja zakończyła się poprawnie

Oczekiwanie na zakończenie zadań

- Zakończenie pracy wątków wymaga jawnego wywołania funkcji `pthread_join`

```
int pthread_join(
   pthread_t thread,
   void **results
);
```

Identyfikator wątku

Wartości zwracane po wykonaniu zadania

- Funkcja ta wywoływana jest na rzecz każdego utworzonego wątku
- Wywołanie funkcji `pthread_join` wstrzymuje realizację programu przez wątek główny do czasu zakończenia zadań w ramach wątku, na rzecz, którego została wywołana
- Funkcja zwraca wartość **zero** jeśli operacja zakończyła się poprawnie

Definiowanie zadań wykonywanych w ramach wątków

- Programista jawnie definiuje zadania, które będą wykonywane równolegle przez wątki
- Jest to realizowane poprzez definiowanie funkcji, które mają zostać wykonane przez poszczególne wątki
- Funkcja ta zawsze przyjmuje oraz zwraca wartości typu (`void *`)

```
void *thread_function(void *arg)
```

Przykład I – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *thread1_work(void *arg)
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void *thread2_work(void *arg)
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    if( pthread_create(&thread1, NULL, thread1_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, thread2_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przykład I – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *thread1_work(void *arg)
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void *thread2_work(void *arg)
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    if( pthread_create(&thread1, NULL, thread1_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, thread2_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Funkcje wykonywane przez wątki

Przykład I – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *thread1_work(void *arg)
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void *thread2_work(void *arg)
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    if( pthread_create(&thread1, NULL, thread1_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, thread2_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Funkcje wykonywane przez wątki

Tworzymy dwa wątki

Przykład I – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *thread1_work(void *arg)
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void *thread2_work(void *arg)
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    if( pthread_create(&thread1, NULL, thread1_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, thread2_work, NULL) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Funkcje wykonywane przez wątki

Tworzymy dwa wątki

Czekamy na zakończenie wykonywanych zadań

Przykład I – Tworzenie wątków

```cpp
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *thread1_work(void *arg)
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n
}

void *thread2_work(void *arg)
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    if( pthread_create(&thread1, NULL, thr
    {
        cout << "Wystapil blad w trakcie t
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, thre
    {
        cout << "Wystapil blad w trakcie t
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

```
rid@gpu2: ~/T2/posix
rid@gpu2:~/T2/posix$ g++ -pthread t2_posix1.cpp
rid@gpu2:~/T2/posix$ ./a.out
Watek 1: Hello world!
Watek 2: Hello world!
rid@gpu2:~/T2/posix$ ./a.out
Watek 2: Hello world!
Watek 1: Hello world!
rid@gpu2:~/T2/posix$
```

Kompilujemy oraz uruchamiamy program

Definiowanie zadań wykonywanych w ramach wątków

- Programista jawnie definiuje zadania, które będą wykonywane równolegle przez wątki
- Jest to realizowane poprzez definiowanie funkcji, które mają zostać wykonane przez poszczególne wątki
- Funkcja ta zawsze przyjmuje oraz zwraca wartości typu (`void *`)

```
void *thread_function(void *arg)
```

- **POSIX Threads dopuszcza możliwość wykorzystania tej samej funkcji do utworzenia wielu wątków**

Przykład II – Tworzenie wątków

```cpp
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *threads_work(void *arg)
{
    sleep(1);
    int threadID = *((int *)arg);
    string output = "Watek #" + to_string(threadID) + ": Hello world!";
    cout << output << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int thread1_id = 0;
    if( pthread_create(&thread1, NULL, threads_work, (void *) &thread1_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    int thread2_id = 1;
    if(pthread_create(&thread2, NULL, threads_work, (void *) &thread2_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przykład II – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *threads_work(void *arg)
{
    sleep(1);
    int threadID = *((int *)arg);
    string output = "Watek #" + to_string(threadID) + ": Hello world!";
    cout << output << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int thread1_id = 0;
    if( pthread_create(&thread1, NULL, threads_work, (void *) &thread1_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    int thread2_id = 1;
    if(pthread_create(&thread2, NULL, threads_work, (void *) &thread2_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Ta sama funkcja wykorzystana jest do utworzenia dwóch wątków

Przykład II – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *threads_work(void *arg)
{
    sleep(1);
    int threadID = *((int *)arg);
    string output = "Watek #" + to_string(threadID) + ": Hello world!";
    cout << output << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int thread1_id = 0;
    if( pthread_create(&thread1, NULL, threads_work, (void *) &thread1_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    int thread2_id = 1;
    if(pthread_create(&thread2, NULL, threads_work, (void *) &thread2_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Ta sama funkcja wykorzystana jest do utworzenia dwóch wątków

Zmienne przechowujące ID wątków

Przykład II – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *threads_work(void *arg)
{
    sleep(1);
    int threadID = *((int *)arg);
    string output = "Watek #" + to_string(threadID) + ": Hello world!";
    cout << output << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int thread1_id = 0;
    if( pthread_create(&thread1, NULL, threads_work, (void *) &thread1_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    int thread2_id = 1;
    if(pthread_create(&thread2, NULL, threads_work, (void *) &thread2_id) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Ta sama funkcja wykorzystana jest do utworzenia dwóch wątków

Przekazujemy argument do funkcji wykonywanej przez wątki

Zmienne przechowujące ID wątków

Przykład II – Tworzenie wątków

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

void *threads_work(void *arg)
{
    sleep(1);
    int threadID = *((int *)arg);
    string output = "Watek #" + to_string(
    cout << output << "\n";
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int thread1_id = 0;
    if( pthread_create(&thread1, NULL, thr
    {
        cout << "Wystapil blad w trakcie t
        exit(-1);
    }

    int thread2_id = 1;
    if(pthread_create(&threa  NULL, thre
    {
        cout << "Wystapil blad w
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Zmie

```
rid@gpu2: ~/T2/posix
rid@gpu2:~/T2/posix$ g++ -pthread t2_posix2.cpp
rid@gpu2:~/T2/posix$ ./a.out
Watek #0: Hello world!
Watek #1: Hello world!
rid@gpu2:~/T2/posix$ ./a.out
Watek #0: Hello world!
Watek #1: Hello world!
rid@gpu2:~/T2/posix$ ./a.out
Watek #1: Hello world!
Watek #0: Hello world!
rid@gpu2:~/T2/posix$
```

Kompilujemy oraz uruchamiamy program

ątki

Przekazywanie wielu argumentów do funkcji wykonywanej przez wątki

- Funkcja `pthread_create` pozwala programiście przekazać jeden argument do funkcji wykonywanej przez wątek
- W przypadku, kiedy chcemy przekazać większą liczbę argumentów należy:
  - Stworzyć strukturę, która będzie przechowywała wszystkie przekazywane argumenty
  - Następnie, przekazać wskaźnik do tej struktury do funkcji tworzącej wątek
- Instancję struktury należy rzutować na typ (`void *`)

Przykład III – Przekazywanie wielu argumentów

```cpp
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

```cpp
int main()
{
    pthread_t thread1, thread2;
    thread_data data1, data2;

    data1.threadID = 0;
    data1.message = "English: Hello World!";

    data2.threadID = 1;
    data2.message = "German: Guten Tag!";

    if( pthread_create(&thread1, NULL, threads_work, (void *) &data1) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, threads_work, (void *) &data2) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przykład III – Przekazywanie wielu argumentów

Struktura argumentów dla wątków

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

```
int main()
{
    pthread_t thread1, thread2;
    thread_data data1, data2;

    data1.threadID = 0;
    data1.message = "English: Hello World!";

    data2.threadID = 1;
    data2.message = "German: Guten Tag!";

    if( pthread_create(&thread1, NULL, threads_work, (void *) &data1) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, threads_work, (void *) &data2) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przykład III – Przekazywanie wielu argumentów

Struktura argumentów dla wątków

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Przygotowanie argumentów

```
int main()
{
    pthread_t thread1, thread2;
    thread_data data1, data2;

    data1.threadID = 0;
    data1.message = "English: Hello World!";

    data2.threadID = 1;
    data2.message = "German: Guten Tag!";

    if( pthread_create(&thread1, NULL, threads_work, (void *) &data1) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, threads_work, (void *) &data2) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przykład III – Przekazywanie wielu argumentów

Struktura argumentów dla wątków

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Przygotowanie argumentów

Każdy wątek otrzymuje unikalną instancję struktury

```
int main()
{
    pthread_t thread1, thread2;
    thread_data data1, data2;

    data1.threadID = 0;
    data1.message = "English: Hello World!";

    data2.threadID = 1;
    data2.message = "German: Guten Tag!";

    if( pthread_create(&thread1, NULL, threads_work, (void *) &data1) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, threads_work, (void *) &data2) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przykład III – Przekazywanie wielu argumentów

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Struktura argumentów dla wątków

```
int main()
{
    pthread_t thread1, thread2;
    thread_data data1, data2;

    data1.threadID = 0;
    data1.message = "English: Hello World!";

    data2.threadID = 1;
    data2.message = "German: Guten Tag!";

    if( pthread_create(&thread1, NULL, threads_work, (void *) &data1) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, threads_work, (void *) &data2) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przygotowanie argumentów

Każdy wątek otrzymuje unikalną instancję struktury

Przekazywanie danych do wątków

Przykład III – Przekazywanie wielu argumentów

```cpp
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Struktura argumentów dla wątków

Rzutowanie argumentu na strukturę oraz dostęp do danych

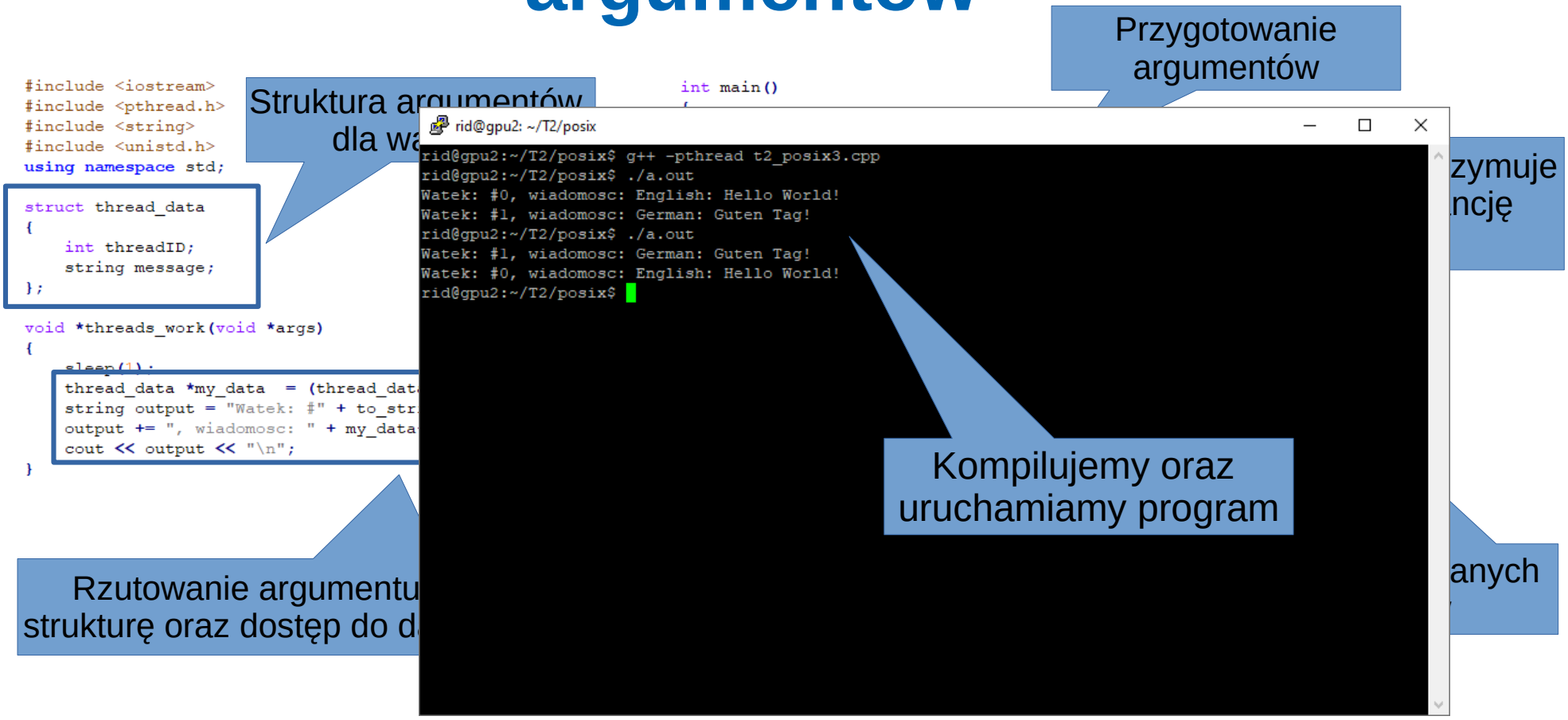
```cpp
int main()
{
    pthread_t thread1, thread2;
    thread_data data1, data2;

    data1.threadID = 0;
    data1.message = "English: Hello World!";

    data2.threadID = 1;
    data2.message = "German: Guten Tag!";

    if( pthread_create(&thread1, NULL, threads_work, (void *) &data1) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
        exit(-1);
    }

    if(pthread_create(&thread2, NULL, threads_work, (void *) &data2) != 0 )
    {
        cout << "Wystapil blad w trakcie tworzenia watku thread2!" << "\n";
        exit(-1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Przygotowanie argumentów

Każdy wątek otrzymuje unikalną instancję struktury

Przekazywanie danych do wątków

Przykład III – Przekazywanie wielu argumentów




```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_dat
    string output = "Watek: #" + to_str
    output += ", wiadomosc: " + my_data
    cout << output << "\n";
}

int main()
```

```
rid@gpu2: ~/T2/posix
rid@gpu2:~/T2/posix$ g++ -pthread t2_posix3.cpp
rid@gpu2:~/T2/posix$ ./a.out
Watek: #0, wiadomosc: English: Hello World!
Watek: #1, wiadomosc: German: Guten Tag!
rid@gpu2:~/T2/posix$ ./a.out
Watek: #1, wiadomosc: German: Guten Tag!
Watek: #0, wiadomosc: English: Hello World!
rid@gpu2:~/T2/posix$
```

Praca z duża liczbą wątków

- Efektywne oraz elastyczne zarządzanie dużą liczbą wątków POSIX możliwe jest poprzez zastosowanie tablic lub typów generycznych przechowujących identyfikatory wątków

```
pthread_t threads[10];
```

```
vector<pthread_t> threads;
```

- Tworzenie oraz dołączanie wątków może zostać realizowane z wykorzystaniem pętli

Przykład IV – Obsługa wielu wątków

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Tablica przechowująca Identyfikatory wątków

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Tablica przechowująca Identyfikatory wątków

Tablica przechowująca argumenty

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!"
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mi
    messages[6] = "Japan: Sekai e konnichi
    messages[7] = "Latin: Orbis, te sal

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Pętla, w której tworzone są wątki

Tablica przechowująca Identyfikatory wątków

Tablica przechowująca argumenty

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!"
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mi
    messages[6] = "Japan: Sekai e konnichi
    messages[7] = "Latin: Orbis, te sal

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Pętla, w której tworzone są wątki

Inicjalizacja argumentów dla *i*-tego wątku

Tablica przechowująca Identyfikatory wątków

Tablica przechowująca argumenty

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!"
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mi
    messages[6] = "Japan: Sekai e konnichi
    messages[7] = "Latin: Orbis, te sal

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Pętla, w której tworzone są wątki

Inicjalizacja argumentów dla *i*-tego wątku

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!"
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mi
    messages[6] = "Japan: Sekai e konnichi
    messages[7] = "Latin: Orbis, te sal
    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Tablica przechowująca Identyfikatory wątków

Tablica przechowująca argumenty

Tworzenie *i*-tego wątku

Przykład IV – Obsługa wielu wątków

Liczba wątków, które zostaną utworzone

```
#include <iostream>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

struct thread_data
{
    int threadID;
    string message;
};

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;
    string output = "Watek: #" + to_string(my_data->threadID);
    output += ", wiadomosc: " + my_data->message;
    cout << output << "\n";
}
```

Pętla, w której tworzone są wątki

Tablica przechowująca Identyfikatory wątków

Tablica przechowująca argumenty

Inicjalizacja argumentów dla *i*-tego wątku

Tworzenie *i*-tego wątku

Oczekiwanie na zakończenie zadań realizowanych przez wątki

```
int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!"
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mi
    messages[6] = "Japan: Sekai e konnichi
    messages[7] = "Latin: Orbis, te sal

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].message = messages[i];

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Przykład V – Podział zadań pomiędzy wątki

```cpp
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 3

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;

    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

```cpp
struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};

int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 3

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;

    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

```
int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Inicjalizacja tablic

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 3

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;

    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}

int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Inicjalizacja tablic

Przygotowanie argumentów oraz utworzenie wątków

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 3

void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;

    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}

int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }

    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>

struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Rzutowanie argumentów na strukturę

```
void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;

    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

Inicjalizacja tablic

```
int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }
```

Przygotowanie argumentów oraz utworzenie wątków

```
    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>

struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Rzutowanie argumentów na strukturę

```
void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;
```

Wątek *ID=0* oblicza sumę elementów tablicy A

```
    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

Inicjalizacja tablic

```
int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }
```

Przygotowanie argumentów oraz utworzenie wątków

```
    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>

struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Rzutowanie argumentów na strukturę

```
void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;
```

Wątek *ID=0* oblicza sumę elementów tablicy A

```
    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }
```

Wątek *ID=1* oblicza sumę elementów tablicy B

```
    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

Inicjalizacja tablic

```
int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }
```

Przygotowanie argumentów oraz utworzenie wątków

```
    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    delete[] B;
    delete[] A;

    return 0;
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>

struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Rzutowanie argumentów na strukturę

```
void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;
```

Wątek *ID=0* oblicza sumę elementów tablicy A

```
    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }
```

Wątek *ID=1* oblicza sumę elementów tablicy B

```
    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }
```

Wątek *ID=2* wyznacza iloczyn skalarny

```
    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

Inicjalizacja tablic

```
int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }
```

Przygotowanie argumentów oraz utworzenie wątków

```
    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>

struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
};
```

Inicjalizacja tablic

```
int main()
{
    srand(time(nullptr));

    const int ARRAY_SIZE = 10;
    int *A = new int[ARRAY_SIZE];
    int *B = new int[ARRAY_SIZE];

    for(int i=0; i<ARRAY_SIZE; ++i)
    {
        A[i] = rand() % 10;
        B[i] = rand() % 20;
    }
```

Przygotowanie argumentów oraz utworzenie wątków

```
    pthread_t threads[NUM_THREADS];
    thread_data data[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        data[i].threadID = i;
        data[i].size = ARRAY_SIZE;
        data[i].A = A;
        data[i].B = B;

        if( pthread_create(&threads[i], NULL, threads_work, (void *) &data[i]) != 0 )
        {
            cout << "Wystapil blad w trakcie tworzenia watku thread1!" << "\n";
            exit(-1);
        }
    }
```

Oczekiwanie na zakończenie zdań

```
    for(int i=0; i<NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }
```

Rzutowanie argumentów na strukturę

```
void *threads_work(void *args)
{
    sleep(1);
    thread_data *my_data  = (thread_data *) args;

    int threadID = my_data->threadID;
    const int size = my_data->size;
    const int *A = my_data->A;
    const int *B = my_data->B;
```

Wątek *ID=0* oblicza sumę elementów tablicy A

```
    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<size; ++i)
            sumA += A[i];

        string output = "Suma elementow tablicy A: " + to_string(sumA);
        cout << output << "\n";
    }
```

Wątek *ID=1* oblicza sumę elementów tablicy B

```
    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<size; ++i)
            sumB += B[i];

        string output = "Suma elementow tablicy B: " + to_string(sumB);
        cout << output << "\n";
    }
```

Wątek *ID=2* wyznacza iloczyn skalarny

```
    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<size; ++i)
            dot += (A[i] * B[i]);

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

Przykład V – Podział zadań pomiędzy wątki

Struktura dla argumentów funkcji

```
struct thread_data
{
    int threadID;
    int size;
    int *A;
    int *B;
```

Inicjalizacja tablic

Rzutowanie argumentów na strukturę

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <pthread.h>
#include <string>

void *threads_work(void
{
    sleep(1);
    thread_data *my_dat

    int threadID = my_d
    const int size = my
    const int *A = my_d
    const int *B = my_d

    if( threadID == 0 )
    {
        int sumA = 0;
        for(int i=0; i<
            sumA += A[i

        string output =
        cout << output
    }

    if( threadID == 1 )
    {
        int sumB = 0;
        for(int i=0; i<
            sumB += B[i

        string output =
        cout << output
    }

    if( threadID == 2 )
    {
        int dot = 0;
        for(int i=0; i<
            dot += (A[i

        string output = "Iloczyn skalarny: " + to_string(dot);
        cout << output << "\n";
    }
}
```

```
rid@gpu2: ~/T2/posix
rid@gpu2:~/T2/posix$ g++ -pthread t2_posix5.cpp
rid@gpu2:~/T2/posix$ ./a.out
Suma elementow tablicy B: 95
Iloczyn skalarny: 514
Suma elementow tablicy A: 49
rid@gpu2:~/T2/posix$
```

Kompilujemy oraz uruchamiamy program

Wątek *ID=2* wyznacza iloczyn skalarny

argumentów
ie wątków

]) != 0)

n";

anie na
enie zdań

Wielowątkowość w języku C++

- Wielowątkowość w języku C++ pojawiła się w 2010 roku wraz z opublikowaniem standardu C++11
- Standard ten oraz jego nowsze wersje (C++14 oraz C++17) definiują szeroką gammę bibliotek, które oferują różnego rodzaju mechanizmy umożliwiające zarządzanie wątkami
- Instalacji dodatkowego oprogramowania platformy obliczeniowej nie jest wymagana

Źródło: www.isocpp.org

Wielowątkowość w języku C++

- Wielowątkowość w języku C++ charakteryzuje się podejściem obiektowym
- Wątki reprezentowane są przez obiekty klasy `thread`
- Klasa ta zdefiniowana jest w następującym pliku nagłówkowym:

  ```
  #include <thread>
  ```

- Wykorzystanie wielowątkowości języka C++ wymaga kompilacji programu z flagami: `-std=c++11 -pthread`

Tworzenie wątków C++

- Nowy wątek uruchamiany jest natychmiast po utworzeniu obiektu klasy `thread`
- Po uruchomieniu wątki istnieją, aż do momentu wywołania funkcji `join`
- Funkcja ta wstrzymuje wykonanie programu przez wątek główny do czasu zakończenia zadań wykonywanych w ramach danego wątku, na rzecz, którego została wywołana
- Wywołanie funkcji na `join` na rzecz obiektu klasy `thread` automatycznie wywołuje jego destruktor

Wątki C++ przydział zadań

- Podobnie jak w przypadku standardu POSIX Threads, kod wykonywany przez wątki definiowany jest w osobnej funkcji
- Funkcja ta przypisywana jest do wątku w trakcie tworzenia (jest ona przekazywana jako argument konstruktora klasy `thread`)
- Najprostszym sposobem definiowania kodu, który ma zostać wykonany przez wątek jest implementacja funkcji zgodnie z językiem programowania C++
- Ta sama funkcja może zostać wykonana przez wiele wątków

Przykład I – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void thread1_work()
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void thread2_work()
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    thread t1(thread1_work);
    thread t2(thread2_work);

    t1.join();
    t2.join();

    return 0;
}
```

Przykład I – Tworzenie wątków C++

```
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void thread1_work()
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void thread2_work()
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    thread t1(thread1_work);
    thread t2(thread2_work);

    t1.join();
    t2.join();

    return 0;
}
```

Funkcje wykonywane przez wątki

Przykład I – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void thread1_work()
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void thread2_work()
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    thread t1(thread1_work);
    thread t2(thread2_work);

    t1.join();
    t2.join();

    return 0;
}
```

Funkcje wykonywane przez wątki

Kod wykonywany przez wątki zaimplementowany jest jako funkcja zgodna z językiem programowania C++

Przykład I – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void thread1_work()
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void thread2_work()
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    thread t1(thread1_work);
    thread t2(thread2_work);

    t1.join();
    t2.join();

    return 0;
}
```

Funkcje wykonywane przez wątki

Kod wykonywany przez wątki zaimplementowany jest jako funkcja zgodna z językiem programowania C++

Tworzymy dwa wątki wykorzystując zdefiniowane powyżej funkcje

Przykład I – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void thread1_work()
{
    sleep(1);
    cout << "Watek 1: Hello world!" << "\n";
}

void thread2_work()
{
    sleep(1);
    cout << "Watek 2: Hello world!" << "\n";
}

int main()
{
    thread t1(thread1_work);
    thread t2(thread2_work);

    t1.join();
    t2.join();

    return 0;
}
```

Funkcje wykonywane przez wątki

Kod wykonywany przez wątki zaimplementowany jest jako funkcja zgodna z językiem programowania C++

Tworzymy dwa wątki wykorzystując zdefiniowane powyżej funkcje

Czekamy na zakończenie zadań

Przykład I – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

void thread1_work()
{
    sleep(1);
    cout << "Watek 1:
}

void thread2_work()
{
    sleep(1);
    cout << "Watek 2:
}

int main()
{
    thread t1(thread1_
    thread t2(thread2_

    t1.join();
    t2.join();

    return 0;
}
```

```
rid@gpu2: ~/T2/cpp
rid@gpu2:~/T2/cpp$ g++ -std=c++11 -pthread ./t2_cpp1.cpp
rid@gpu2:~/T2/cpp$ ./a.out
Watek 1: Hello world!
Watek 2: Hello world!
rid@gpu2:~/T2/cpp$
```

Kompilujemy oraz uruchamiamy program

Wątki C++ przydział zadań

- Podobnie jak w przypadku standardu POSIX Threads, kod wykonywany przez wątki definiowany jest w osobnej funkcji
- Funkcja ta przypisywana jest do wątku w trakcie tworzenia (jest ona przekazywana jako argument konstruktora klasy `thread`)
- Najprostszym sposobem definiowania kodu, który ma zostać wykonany przez wątek jest implementacja funkcji zgodnie z językiem programowania C++
- Ta sama funkcja może zostać wykonana przez wiele wątków
- Klasa `thread` współpracuje również z obiektami funkcyjnymi (obiekt klasy z przeciążonym operatorem wywołania)

Przykład II – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator()()
        {
            cout << "Hello from a thread!" << "\n";
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask()));

    t1.join();
    t2.join();

    return 0;
}
```

Przykład II – Tworzenie wątków C++

Definiujemy obiekt funkcyjny

```
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator()()
        {
            cout << "Hello from a thread!" << "\n";
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask()));

    t1.join();
    t2.join();

    return 0;
}
```

Przykład II – Tworzenie wątków C++

Definiujemy obiekt funkcyjny

Przeciążamy operator wywołania

```
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator()()
        {
            cout << "Hello from a thread!" << "\n";
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask()));

    t1.join();
    t2.join();

    return 0;
}
```

Przykład II – Tworzenie wątków C++

```
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator()()
        {
            cout << "Hello from a thread!" << "\n";
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask()));

    t1.join();
    t2.join();

    return 0;
}
```

Definiujemy obiekt funkcyjny

Przeciążamy operator wywołania

Tworzymy dwa wątki wykorzystując obiekty funkcyjne

Przykład II – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator()()
        {
            cout << "Hello from a thread!" << "\n";
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask()));

    t1.join();
    t2.join();

    return 0;
}
```

Definiujemy obiekt funkcyjny

Przeciążamy operator wywołania

Tworzymy dwa wątki wykorzystując obiekty funkcyjne

Ten sam kod wykonywany jest przez dwa wątki

Przykład II – Tworzenie wątków C++

```cpp
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator()()
        {
            cout << "Hello from a thread!" << "\n";
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask()));

    t1.join();
    t2.join();

    return 0;
}
```

Definiujemy obiekt funkcyjny

Przeciążamy operator wywołania

Tworzymy dwa wątki wykorzystując obiekty funkcyjne

Ten sam kod wykonywany jest przez dwa wątki

Czekamy na zakończenie zadań

Przykład II – Tworzenie wątków C++

```
#include <iostream>
#include <thread>
#include <unistd.h>
using namespace std;

class MyTask
{
    public:
        void operator
        {
            cout << "H
        }
};

int main()
{
    MyTask mt;
    thread t1(mt);

    thread t2((MyTask

    t1.join();
    t2.join();

    return 0;
}
```

Definiujemy obiekt funkcyjny

```
rid@gpu2: ~/T2/cpp
rid@gpu2:~/T2/cpp$ g++ -std=c++11 -pthread ./t2_cpp2.cpp
rid@gpu2:~/T2/cpp$ ./a.out
Hello from a thread!
Hello from a thread!
rid@gpu2:~/T2/cpp$
```

Kompilujemy oraz uruchamiamy program

Przekazywanie argumentów do funkcji wykonywanej przez wątki

- Podobnie jak w przypadku POSIX Threads, w przypadku wielowątkowości języka C++ istnieje możliwość przekazywania danych do wątków poprzez argumenty funkcji
- Realizowane jest to poprzez podanie argumentów jako kolejne (po funkcji lub obiekcie funkcyjnym) parametry konstruktora klasy `thread`
  - Przekazane argumenty kopiowane lub przenoszone do uruchamianego wątku
  - Jeśli wymagane jest przekazanie referencji do funkcji wykonywanej przez wątek należy użyć standardowych wraperów referencji, przykładowo: `std::ref()`
- W przypadku obiektów funkcyjnych, przekazanie argumentów może być realizowane poprzez parametry konstruktora obiektu

Przykład III – Przekazywanie argumentów

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Przykład III – Przekazywanie argumentów

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Funkcja wykonywana przez wątki

Przykład III – Przekazywanie argumentów

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

Funkcja wykonywana przez wątki

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Przykład III – Przekazywanie argumentów

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

Funkcja wykonywana przez wątki

Inicjalizacja argumentów

Przykład III – Przekazywanie argumentów

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

Funkcja wykonywana przez wątki

Inicjalizacja argumentów

Uruchomienie dwóch wątków

Przykład III – Przekazywanie argumentów

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

Funkcja wykonywana przez wątki

Inicjalizacja argumentów

Uruchomienie dwóch wątków

Przekazanie argumentów do funkcji wykonywanej przez wątki

Przykład III – Przekazywanie argumentów

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    return 0;
}
```

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

Funkcja wykonywana przez wątki

Inicjalizacja argumentów

Uruchomienie dwóch wątków

Argumenty przekazywane są do konstruktora klasy `thread` po nazwie wykonywanej funkcji

Przekazanie argumentów do funkcji wykonywanej przez wątki

Przykład III – Przekazywanie argumentów

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

void thread_work(const int ID, const string message)
{
    sleep(1);
    string output = "Watek #" + to_string(ID) + ": " + message;
    cout << output << "\n";
}

int main()
{
    int thread1_id = 0;
    string thread1_msg = "English: Hello World!";

    int thread2_id = 1;
    string thread2_msg = "German: Guten Tag!";

    thread t1(thread_work, thread1_id, thread1_msg);
    thread t2(thread_work, thread2_id, thread2_msg);

    t1.join();
    t2.join();

    re    0;
}
```

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

Funkcja wykonywana przez wątki

Inicjalizacja argumentów

Uruchomienie dwóch wątków

Argumenty przekazywane są do konstruktora klasy `thread` po nazwie wykonywanej funkcji

Przekazanie argumentów do funkcji wykonywanej przez wątki

Oczekiwanie na zakończenie zadań

Przykład III – Przekazywanie argumentów

```
#include <iostream>
#include <strin
#include <threa
#include <unist
using namespace

void thread_wo
{
    sleep(1);
    string outp
    cout << out
}

int main()
{
    int thread1
    string thre

    int thread2
    string thre

    thread t1(t
    thread t2(t

    t1.join();
    t2.join();

    re      0;
}
```

Argumenty funkcji: liczba całkowita oraz łańcuch znaków

ątki

struktora
ej funkcji

Oczekiwanie na zakończenie zadań

rid@gpu2: ~/T2/cpp

```
rid@gpu2:~/T2/cpp$ g++ -std=c++11 -pthread ./t2_cpp3.cpp
rid@gpu2:~/T2/cpp$ ./a.out
Watek #1: German: Guten Tag!
Watek #0: English: Hello World!
rid@gpu2:~/T2/cpp$ ./a.out
Watek #0: English: Hello World!
Watek #1: German: Guten Tag!
rid@gpu2:~/T2/cpp$
```

Kompilujemy oraz uruchamiamy program

Praca z dużą liczbą wątków

- Podobnie jak w przypadku POSIX Threads, efektywne oraz elastyczne zarządzanie dużą liczbą wątków możliwe jest poprzez zastosowanie tablic lub typów generycznych

```
thread threads[10];
```

```
vector<thread> threads;
```

- Tworzenie oraz dołączanie wątków może zostać realizowane z wykorzystaniem pętli

Przykład IV – Obsługa wielu wątków

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

Liczba wątków

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Przykład IV – Obsługa wielu wątków

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja wykonywana przez wątki

Przykład IV – Obsługa wielu wątków

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę całkowitą oraz tablicę typu `string`

Funkcja wykonywana przez wątki

Przykład IV – Obsługa wielu wątków

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę całkowitą oraz tablicę typu `string`

Funkcja wykonywana przez wątki

Tablica przechowująca obiekty reprezentujące wątki

Przykład IV – Obsługa wielu wątków

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę całkowitą oraz tablicę typu `string`

Funkcja wykonywana przez wątki

Tablica przechowująca obiekty reprezentujące wątki

Pętla tworząca wątki

Przykład IV – Obsługa wielu wątków

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę całkowitą oraz tablicę typu `string`

Funkcja wykonywana przez wątki

Tablica przechowująca obiekty reprezentujące wątki

Pętla tworząca wątki

Tworzenie *i*-tego wątku oraz przekazywanie argumentów do funkcji

Przykład IV – Obsługa wielu wątków

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę całkowitą oraz tablicę typu `string`

Funkcja wykonywana przez wątki

Każdy z wątków wyświetla swoje ID oraz element tablicy od indeksie równym jego ID

Tablica przechowująca obiekty reprezentujące wątki

Pętla tworząca wątki

Tworzenie *i*-tego wątku oraz przekazywanie argumentów do funkcji

Przykład IV – Obsługa wielu wątków

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const int threadID, const string messages[])
{
    sleep(1);
    string output = "Watek: #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
    cout << output << "\n";
}

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    thread threads[NUM_THREADS];

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i] = thread(threads_work, i, messages);
    }

    for(int i=0; i<NUM_THREADS; ++i)
    {
        threads[i].join();
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę całkowitą oraz tablicę typu `string`

Funkcja wykonywana przez wątki

Każdy z wątków wyświetla swoje ID oraz element tablicy od indeksie równym jego ID

Tablica przechowująca obiekty reprezentujące wątki

Pętla tworząca wątki

Tworzenie *i*-tego wątku oraz przekazywanie argumentów do funkcji

Oczekiwanie na zakończenie zadań

Przykład IV – Obsługa wielu wątków

```
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

void threads_work(const
{
    sleep(1);
    string output = "Wat
    cout << output << "\
}

int main()
{
    string messages[NUM_
    messages[0] = "Engli
    messages[1] = "Frenc
    messages[2] = "Spani
    messages[3] = "Klingo
    messages[4] = "Germa
    messages[5] = "Russi
    messages[6] = "Japan
    messages[7] = "Latin

    thread threads[NUM_TH

    for(int i=0; i<NUM_TH
    {
        threads[i] = thre
    }

    for(int i=0; i<NUM_TH
    {
        threads[i].join()
    }

    return 0;
}
```

Liczba wątków

Funkcja przyjmuje dwa argumenty: liczbę

Oczekiwanie na zakończenie zadań

rid@gpu2: ~/T2/cpp

```
rid@gpu2:~/T2/cpp$ g++ -std=c++11 -pthread ./t2_cpp4.cpp
rid@gpu2:~/T2/cpp$ ./a.out
Watek: #1, wiadomosc: French: Bonjour, le monde!
Watek: #5, wiadomosc: Russian: Zdravstvytye, mir!
Watek: #3, wiadomosc: Klingon: Nuq neH!
Watek: #0, wiadomosc: English: Hello World!
Watek: #2, wiadomosc: Spanish: Hola al mundo
Watek: #6, wiadomosc: Japan: Sekai e konnichiwa!
Watek: #7, wiadomosc: Latin: Orbis, te saluto!
Watek: #4, wiadomosc: German: Guten Tag, Welt!
rid@gpu2:~/T2/cpp$
```

Kompilujemy oraz uruchamiamy program

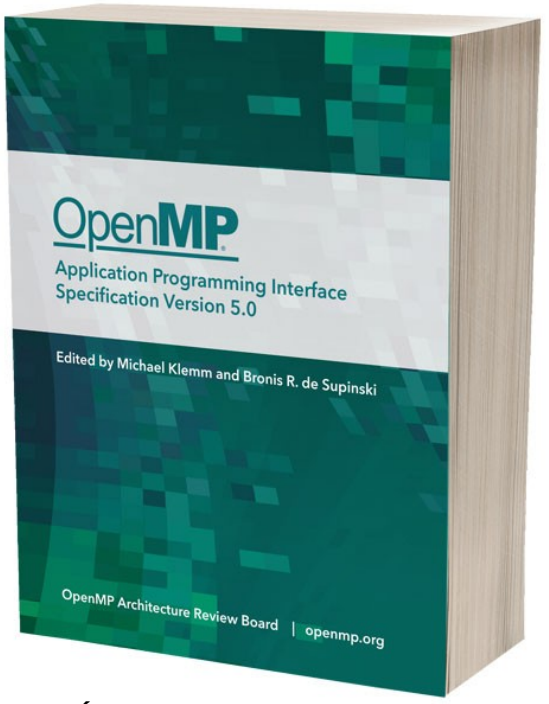
Standard OpenMP

- OpenMP (*Open Multi-Processing*) jest wieloplatformowym sandardem programowania równoległego
- Ze względu na swoje właściwości cieszy się on dużą popularnością
- OpenMP pozwala na tworzenie aplikacji równoległych dla systemów Linux oraz Windows
- Standard przeznaczony dla trzech języków programowania C, C++ oraz Fortran

Źródło: www.openmp.org

Standard OpenMP

- Standard OpenMP składa się z trzech komponentów:
  - Dyrektyw kompilatora
  - Funkcji bibliotecznych
  - Zmiennych środowiskowych
- Aktualnie dostępną wersją standardu jest wersja 5.0
- Strona domowa: https://www.openmp.org



Źródło: www.openmp.org

Zrównoleglenie aplikacji z wykorzystaniem standardu OpenMP

- Do zrównoleglenia programu w standardzie OpenMP wykorzystuje się dedykowaną dyrektywę `#pragma omp parallel`
- Dyrektywa ta informuje kompilator, że poprzedzony nią obszar kod źródłowego zostanie wykonany w sposób równoległy
- Kompilacja kodu źródłowego zawierającego dyrektywy OpenMP wymaga ustawienia odpowiedniej flagi
  - W przypadku kompilatora GCC jest to flaga `–fopenmp`
  - Brak flagi przy kompilacji powoduje, że dyrektywy zostaną zignorowane, a otrzymana aplikacja będzie aplikacją jednowątkową

```
g++ -fopenmp -o prog prog.cpp
```

Zrównoleglenie aplikacji z wykorzystaniem standardu OpenMP

- Zastosowanie dyrektywy `#pragma omp parallel` skutkuje utworzeniem wątków, których liczba jest równa liczbie logicznych rdzeni
- Sterowanie liczbą wątków w obszarze równoległym możliwe jest z wykorzystaniem między innymi klauzuli `num_threads`, przykładowo: `#pragma omp parallel num_threads(4)` tworzy cztery wątki
- Oprócz dyrektyw kompilatora, OpenMP dostarcza szereg funkcji bibliotecznych zdefiniowanych w nagłówku: `#include <omp.h>`. Przykładowe funkcje:
  - `omp_get_thread_num()` - zwraca identyfikator wątku w grupie
  - `omp_get_num_threads()` - zwraca informacje o liczbie utworzonych wątków

OpenMP – przykład I

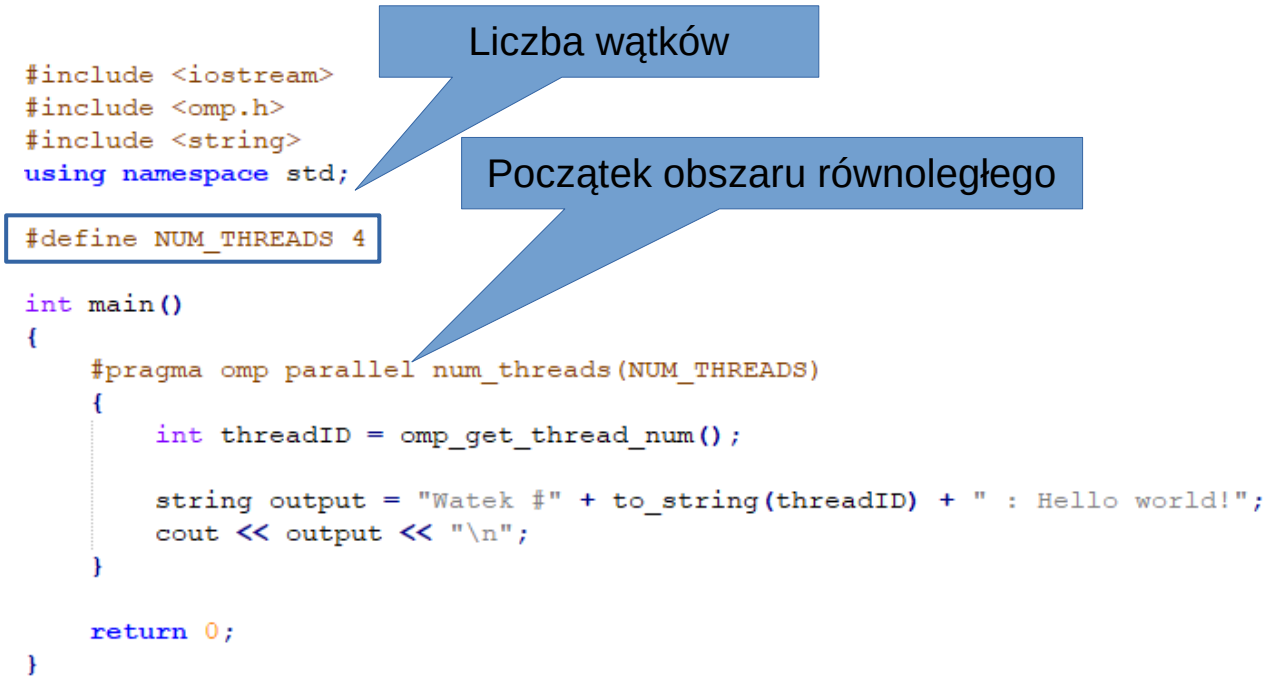
```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład I

Liczba wątków

```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

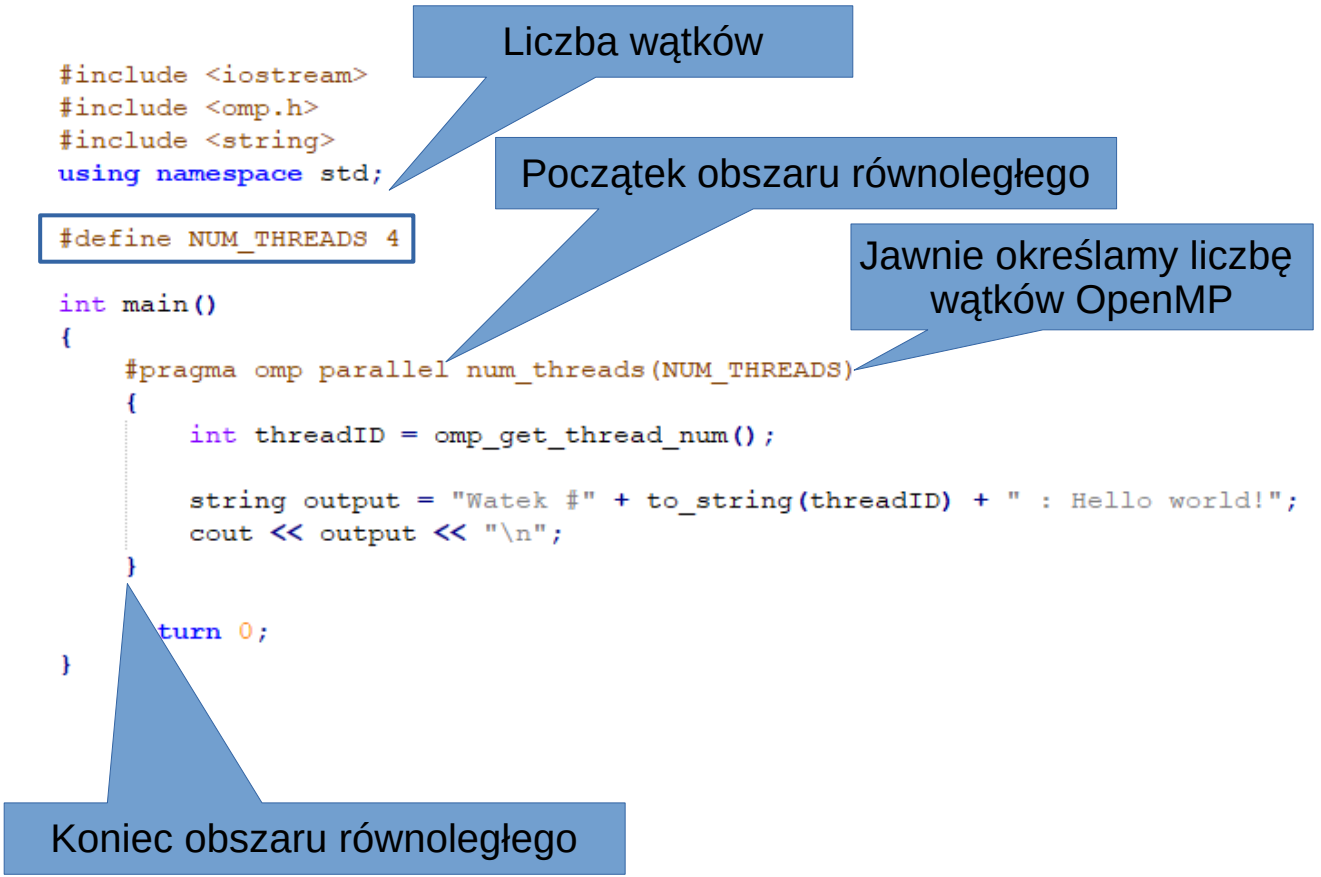
OpenMP – przykład I




```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

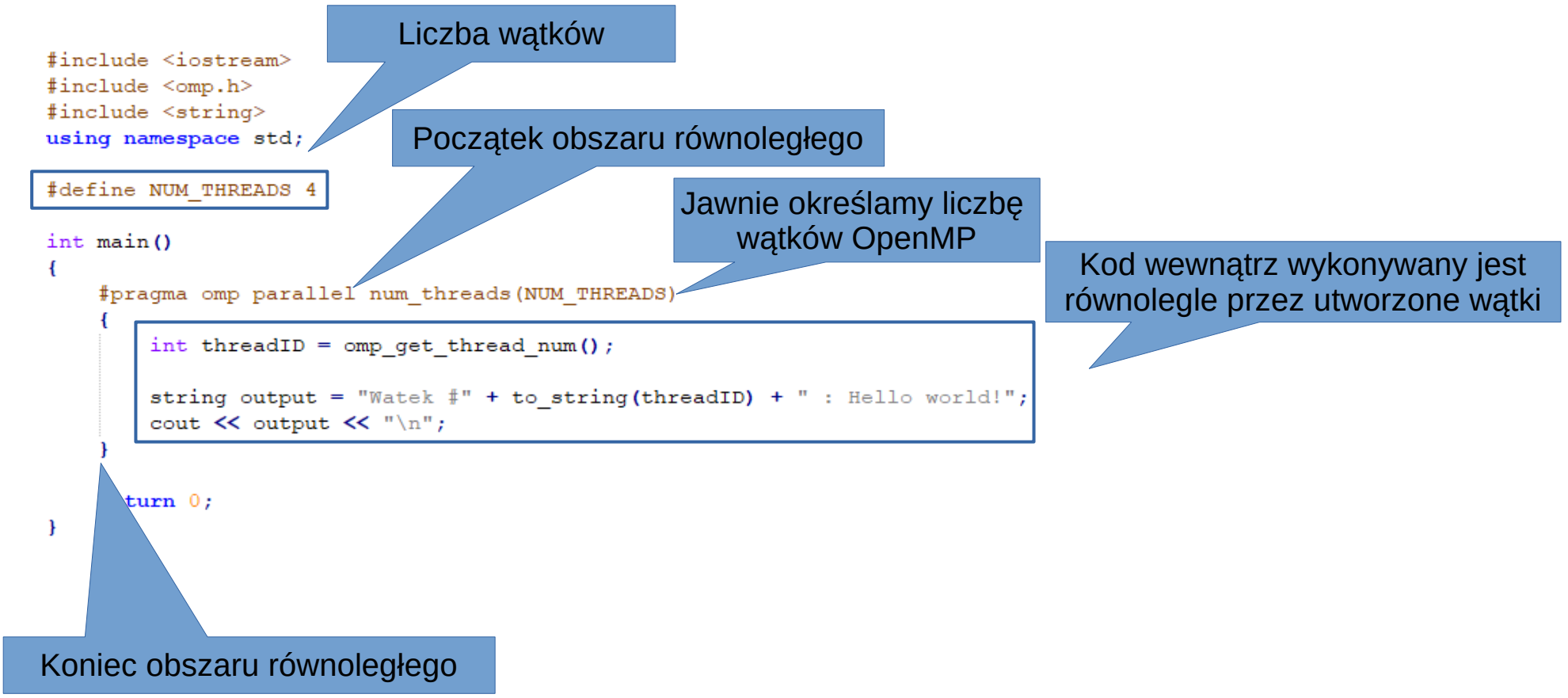
OpenMP – przykład I

OpenMP – przykład I



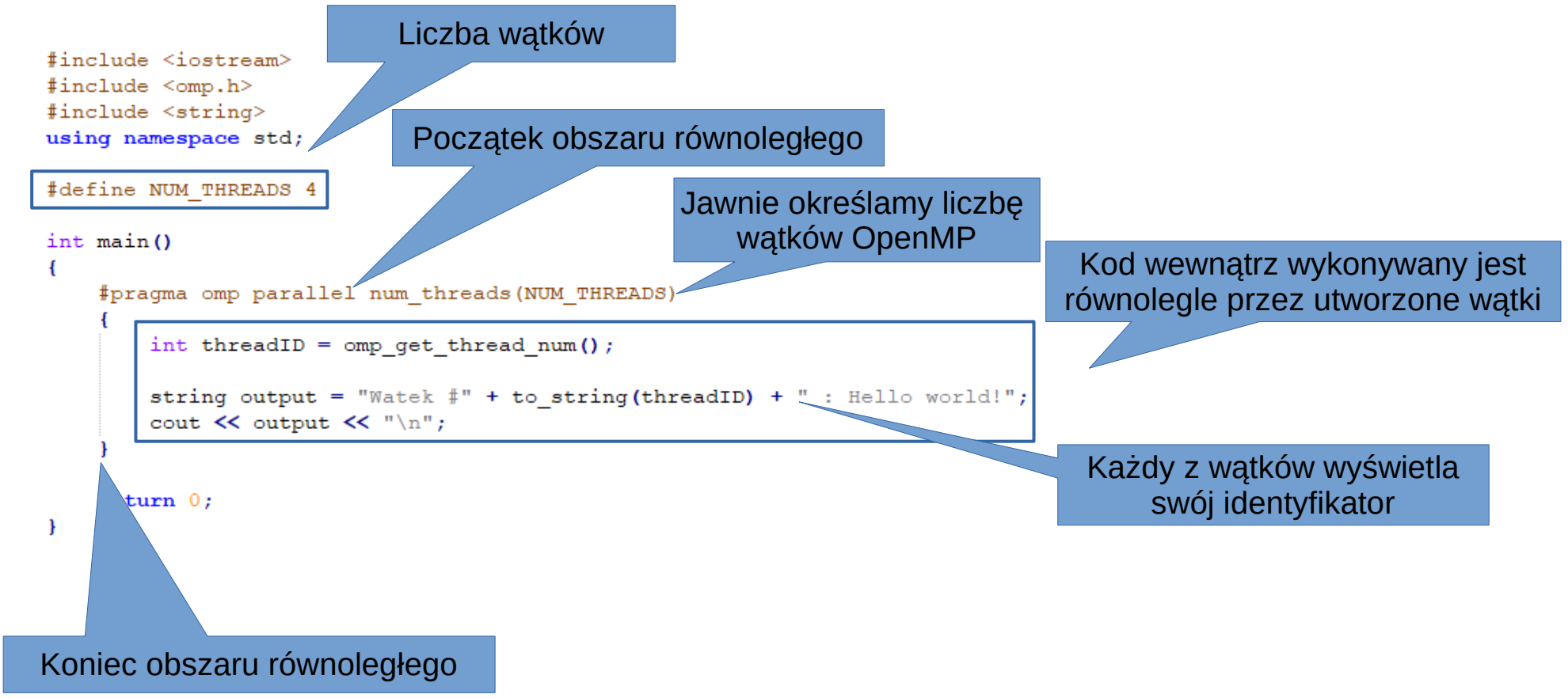

```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład I



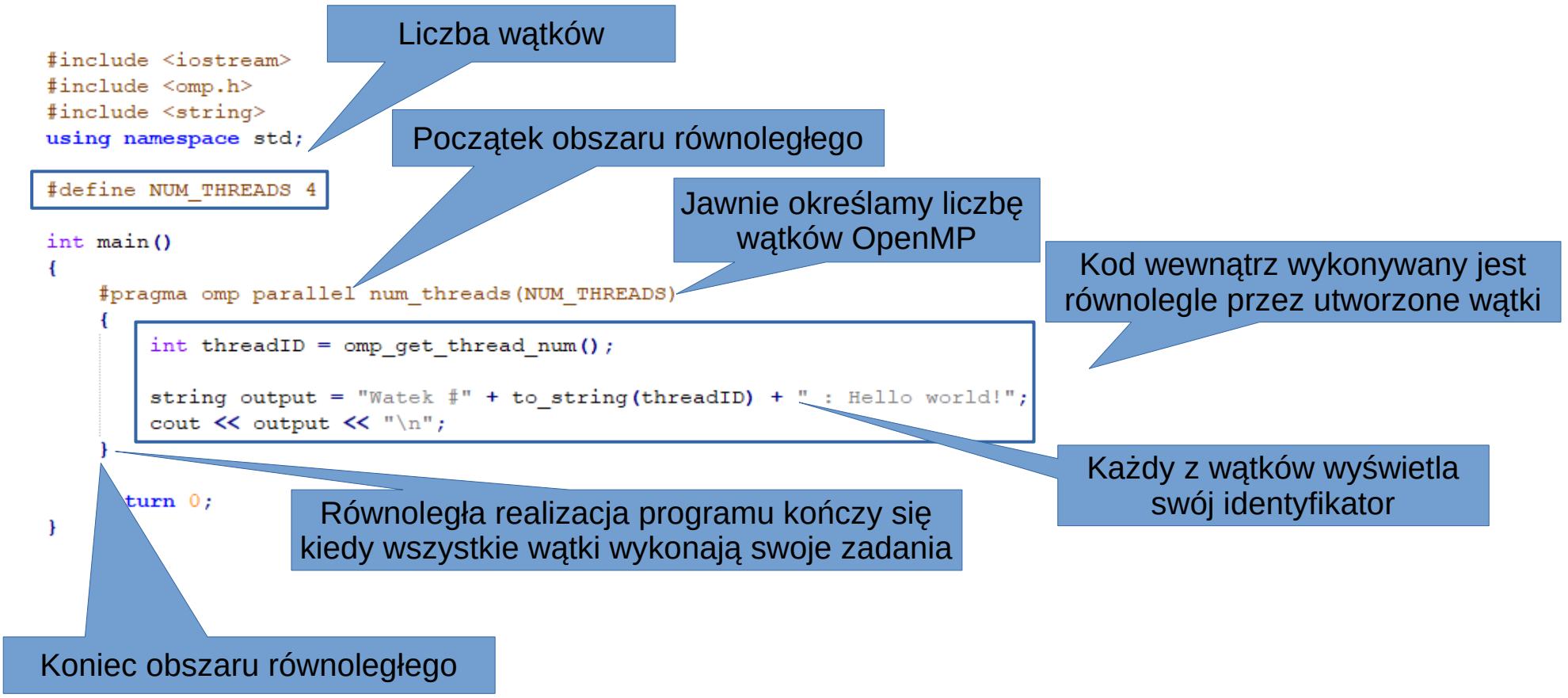

```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład I



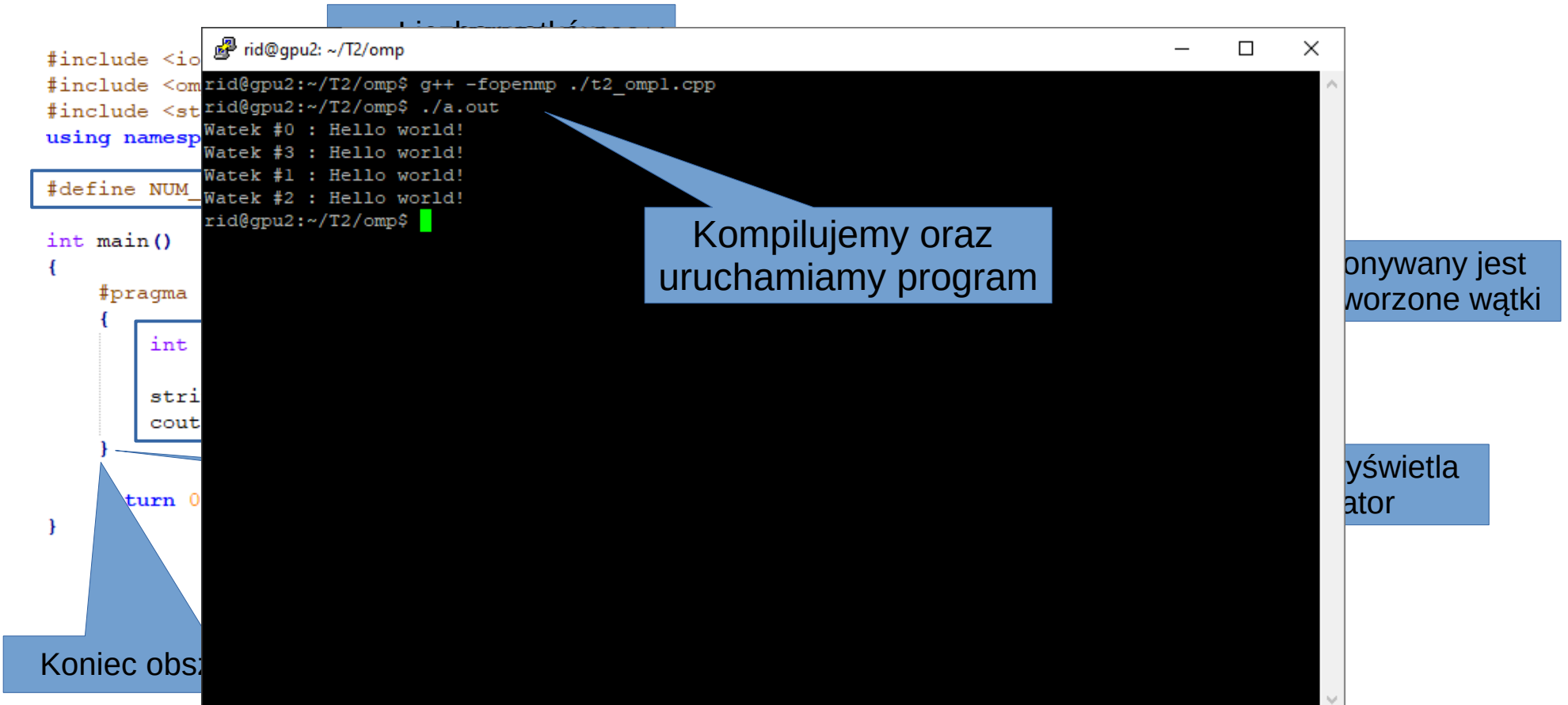

```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład I




```
#include <iostream>
#include <omp.h>
#include <string>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład I



OpenMP – przykład II

```cpp
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

OpenMP – przykład II

Liczba wątków

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

Liczba wątków

Początek obszaru równoległego

Koniec obszaru równoległego

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

Liczba wątków

Początek obszaru równoległego

Jawnie określamy liczbę wątków OpenMP

Koniec obszaru równoległego

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

Liczba wątków

Początek obszaru równoległego

Jawnie określamy liczbę wątków OpenMP

Fragment kodu wykonywany równolegle przez wszystkie uruchomione wątki

Koniec obszaru równoległego

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

Liczba wątków

Początek obszaru równoległego

Jawnie określamy liczbę wątków OpenMP

Fragment kodu wykonywany równolegle przez wszystkie uruchomione wątki

Kod wykonywany przez wątek o Identyfikatorze *threadID*=0

Koniec obszaru równoległego

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

Liczba wątków

Początek obszaru równoległego

Jawnie określamy liczbę wątków OpenMP

Fragment kodu wykonywany równolegle przez wszystkie uruchomione wątki

Kod wykonywany przez wątek o Identyfikatorze *threadID*=0

Wątek *ID*=0 wyświetla informacje o liczbie uruchomionych wątków

Koniec obszaru równoległego

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 4

int main()
{
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + " : Hello world!";
        cout << output << "\n";

        if(threadID == 0)
        {
            sleep(1);
            int totalThreads = omp_get_num_threads();

            output = "Liczba uruchomionych watkow: " + to_string(totalThreads);
            cout << output << "\n";
        }
    }

    return 0;
}
```

Liczba wątków

Początek obszaru równoległego

Jawnie określamy liczbę wątków OpenMP

Fragment kodu wykonywany równolegle przez wszystkie uruchomione wątki

Kod wykonywany przez wątek o Identyfikatorze *threadID*=0

Wątek *ID*=0 wyświetla informacje o liczbie uruchomionych wątków

Koniec obszaru równoległego

Wątek o *ID*=0 jest wątkiem głównym programu

OpenMP – przykład II

```
#include <iostream>
#include <omp.h>
#include <string
#include <unistd
using namespace

#define NUM_THRE

int main()
{
    #pragma omp
    {
        int thre

        string o
        cout <<

        if(threa
        {
            slee
            int

            outp
            cout
        }
    }

    return 0;
}
```

Liczba watków

```
rid@gpu2: ~/T2/omp
rid@gpu2:~/T2/omp$ g++ -fopenmp ./t2_omp2.cpp
rid@gpu2:~/T2/omp$ ./a.out
Watek #0 : Hello world!
Watek #2 : Hello world!
Watek #1 : Hello world!
Watek #3 : Hello world!
Liczba uruchomionych watkow: 4
rid@gpu2:~/T2/omp$
```

Kompilujemy oraz uruchamiamy program

konywany
wszystkie
wątki

z wątek o
adID=0

informacje
ch wątków

głównym programu

OpenMP – przykład III

```cpp
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład III

Liczba wątków

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład III

Liczba wątków

Obszar równoległy zawierający NUM_THREADS wątków

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

OpenMP – przykład III

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

Liczba wątków

Obszar równoległy zawierający NUM_THREADS wątków

Kod wykonywany przez wszystkie uruchomione wątki

OpenMP – przykład III

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

Liczba wątków

Obszar równoległy zawierający NUM_THREADS wątków

Każdy z wątków wyświetla swoje *ID* oraz element tablicy od indeksie równym jego *ID*

Kod wykonywany przez wszystkie uruchomione wątki

OpenMP – przykład III

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

Liczba wątków

Obszar równoległy zawierający NUM_THREADS wątków

Każdy z wątków wyświetla swoje *ID* oraz element tablicy od indeksie równym jego *ID*

Kod wykonywany przez wszystkie uruchomione wątki

Tablica `messages` jest współdzielona przez uruchomione wątki

OpenMP – przykład III

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

Liczba wątków

Wszystkie zmienne utworzone przed obszarem równoległym domyślnie są współdzielone przez wszystkie uruchomione wątki OpenMP

Obszar równoległy zawierający NUM_THREADS wątków

Każdy z wątków wyświetla swoje *ID* oraz element tablicy od indeksie równym jego *ID*

Kod wykonywany przez wszystkie uruchomione wątki

Tablica `messages` jest współdzielona przez uruchomione wątki

OpenMP – przykład III

```
#include <iostream>
#include <omp.h>
#include <string>
#include <unistd.h>
using namespace std;

#define NUM_THREADS 8

int main()
{
    string messages[NUM_THREADS];
    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    #pragma omp parallel num_threads(NUM_THREADS)
    {
        int threadID = omp_get_thread_num();

        string output = "Watek #" + to_string(threadID) + ", wiadomosc: " + messages[threadID];
        cout << output << "\n";
    }

    return 0;
}
```

Liczba wątków

Wszystkie zmienne utworzone przed obszarem równoległym domyślnie są współdzielone przez wszystkie uruchomione wątki OpenMP

Obszar równoległy zawierający NUM_THREADS wątków

Każdy z wątków wyświetla swoje *ID* oraz element tablicy od indeksie równym jego *ID*

Kod wykonywany przez wszystkie uruchomione wątki

Zmienne utworzone wewnątrz obszaru równoległego są zmiennymi prywatnymi danego wątku

Tablica `messages` jest współdzielona przez uruchomione wątki

OpenMP – przykład III

```
#include <iostream>
#include <omp.h>
#include <strin
#include <unist
using namespace

#define NUM_THR

int main()
{
    string mess
    messages[0]
    messages[1]
    messages[2]
    messages[3]
    messages[4]
    messages[5]
    messages[6]
    messages[7]

    #pragma omp
    {
        int thr

        string
        cout <<
    }

    return 0;
}
```

Liczba wątków

```
rid@gpu2: ~/T2/omp
rid@gpu2:~/T2/omp$ g++ -fopenmp ./t2_omp3.cpp
rid@gpu2:~/T2/omp$ ./a.out
Watek #0, wiadomosc: English: Hello World!
Watek #6, wiadomosc: Japan: Sekai e konnichiwa!
Watek #3, wiadomosc: Klingon: Nuq neH!
Watek #2, wiadomosc: Spanish: Hola al mundo
Watek #1, wiadomosc: French: Bonjour, le monde!
Watek #7, wiadomosc: Latin: Orbis, te saluto!
Watek #4, wiadomosc: German: Guten Tag, Welt!
Watek #5, wiadomosc: Russian: Zdravstvytye, mir!
rid@gpu2:~/T2/omp$
```

Kompilujemy oraz uruchamiamy program

oje *ID*
nym jego *ID*

ywany przez
uchomione wątki

t współdzielona
ione wątki

zmiennymi prywatnymi danego wątku

Zadanie 1

Przykład V opracowany dla standardu POSIX Threads, wyznaczający sumy elementów wektorów A i B oraz iloczyn skalarny tych wektorów, zaimplementować z wykorzystaniem standardu OpenMP. Program rozszerzyć o wyznaczanie wartości minimalnej oraz maksymalnej dla każdego z wektorów.

Zadanie 2

Przykład V opracowany dla standardu POSIX Threads, wyznaczający sumy elementów wektorów A i B oraz iloczyn skalarny tych wektorów, zaimplementować z wykorzystaniem wielowątkowości języka C++. Program rozszerzyć o wyznaczanie wartości minimalne oraz maksymalnej dla każdego z wektorów.

Dziękuję za uwagę!