Akademia Nauk Stosowanych w Nowym Sączu Programowanie Współbieżne i Rozproszone			
Temat: PWIR_11			**spr nr 11**
Nazwisko i imię: Ciapała Tadeusz		**Ocena sprawozdania**	**Zaliczenie:**
Data wykonania ćwiczenia:	**Grupa: P3**		

1) KODY:

KOD DO ZADAŃ:

```
#include "mpi.h"
#include <cstdio>
#include <iostream>
#include <time.h>

void mainProcess(int size) {
	srand(time(NULL));

	//alokujemy wektory o rozmiarze(5*(ilosc procesów-1))
	unsigned int* va = new unsigned int[5 * (size - 1)];
	unsigned int* vb = new unsigned int[5 * (size - 1)];
	unsigned int* vc = new unsigned int[5 * (size - 1)];

	//wypełniamy a i b losowymi danymi, a vc zerujemy
	for (unsigned int i = 0; i < 5 * (size - 1); i++) {
		va[i] = rand() % 10;
		vb[i] = rand() % 10;
		vc[i] = 0;
	}

	//broadcastujemy wektor a do pozostałych procesów
	MPI_Bcast(va, 5 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

	//broadcastujemy wektor b do pozostałych procesów
	MPI_Bcast(vb, 5 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

	//odpalamy nasłuch
	MPI_Request* requests = new MPI_Request[size - 1];
	MPI_Status* statuses = new MPI_Status[size - 1];
	for (unsigned int i = 0; i < size-1; i++) {
		MPI_Irecv(vc + i * 5, 5, MPI_UNSIGNED, i + 1, 0, MPI_COMM_WORLD,
&requests[i]);
	}
	MPI_Waitall(size - 1, requests, statuses);

	//wypisujemy wyniki
	for (unsigned int i = 0; i < (5 * (size - 1)); i++) printf("%d\t", va[i]);
	printf("\r\n");
	for (unsigned int i = 0; i < (5 * (size - 1)); i++) printf("%d\t", vb[i]);
	printf("\r\n");
	for (unsigned int i = 0; i < (5 * (size - 1)); i++) printf("%d\t", vc[i]);
	printf("\r\n");

	//zwalniamy pamięć
	delete[] va;
	delete[] vb;
	delete[] vc;
	delete[] requests;
	delete[] statuses;
}

void workerProcess(int id, int size) {
	//alokujemy buffor na moją część zadania
```

```
	unsigned int* v = new unsigned int[5];

	//alokujemny miejsce na wektor a oraz b
	unsigned int* va = new unsigned int[5 * (size - 1)];
	unsigned int* vb = new unsigned int[5 * (size - 1)];

	//nasłuchujemy bcasta wektora a
	MPI_Bcast(va, 5 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

	//nasłuchujemy bcasta wektora b
	MPI_Bcast(vb, 5 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

	//liczymy sumę
	for (unsigned int i = 0; i < 5; i++) {
		v[i] = va[(id - 1) * 5 + i] + vb[(id - 1) * 5 + i];
	}

	//odsyłamy wynik
	MPI_Send(v, 5, MPI_UNSIGNED, 0, 0, MPI_COMM_WORLD);

	//zwalniamy pamięc
	delete[] v;
	delete[] va;
	delete[] vb;
}

int main()
{
	int PID, PCOUNT;

	MPI_Init(NULL, NULL);

	MPI_Comm_rank(MPI_COMM_WORLD, &PID);
	MPI_Comm_size(MPI_COMM_WORLD, &PCOUNT);

	if (PID == 0) { //jestem procesem głównym
		mainProcess(PCOUNT);
	}
	else { //jestem procesem roboczym
		workerProcess(PID, PCOUNT);
	}

	MPI_Finalize();

	return 0;
}
```

2) **Zadania i rozwiązania:**

ZADANIE 1:

Naszym pierwszy zadaniem była analiza dostarczonego kodu. Ogólnie mówiąc en kod implementuje program równoległy w MPI, gdzie proces główny generuje losowe wektory, rozgłasza je do procesów roboczych, odbiera częściowe wyniki obliczeń od procesów roboczych i wyświetla wszystkie wektory na standardowym wyjściu. Procesy robocze otrzymują wektory, wykonują operacje dodawania na swoich częściach danych i przesyłają wyniki z powrotem do procesu głównego.

W funkcji mainProcess() zdefiniowane są operacje głównego procesu:

- mamy generowanie liczb losowych, inicjalizator srand()
- są trzy dynamicznie alokowane tablice typu unsigned int*
- są one wypełniane losowymi liczbami
- następnie mamy rozesłanie wartości tablic va i vb do innych procesów roboczych przy użyciu MPI_Beast
- tworzone są później tablice dynamiczne requests i statuses, przechowujące typy, które są używane przy komunikacji między poszczególnymi procesami MPI
- dalej mamy odbiory w funkcji MPI Irecv do tablicy vc
- mamy później waitwalla czy też barierę w postaci funkcji MPI_Waitwall()

W funkcji workerProcess:

- odbiera za pomocą MPI_Bcast (transmisja rozgloszeniowa) rozmiar wektora vectorSize i rozmiar części partSize od procesu głównego.
- tworzy dynamiczną tablicę “v” o rozmiarze partSize do przechowywania częściowego wyniku.
- tworzy dynamiczne tablice va i vb o rozmiarze partSize * (size - 1) do przechowywania częściowych wektorów va i vb.
- odbiera transmisję rozgłoszeniową wartości tablic va i vb od procesu głównego.
- oblicza sumę elementów odpowiadających swojemu indeksowi w tablicach va i vb i zapisuje wynik do tablicy v.
- wysyła tablicę v do procesu głównego przy użyciu funkcji MPI_Send.
- zwalnia dynamicznie zaalokowaną pamięć.

W main:

- Inicjalizuje MPI
- Pobiera identyfikator procesu PID i liczbę procesów w klastrze PCOUNT
- Jeśli PID jest równy 0, to wykonuje funkcję mainProcess() i zarządza innymi procesami. W przeciwnym razie, wykonuje funkcję workerProcess()
- Kończy pracę MPI oraz kończy program

ZADANIE 2:

Zmodyfikowany kod do zadania drugiego

```
void mainProcess(int size) {
    srand(time(NULL));

    const int PART_SIZE = 3; // liczba elementow przypisana do jednego procesu
    unsigned int* va = new unsigned int[PART_SIZE * (size - 1)];
    unsigned int* vb = new unsigned int[PART_SIZE * (size - 1)];
    unsigned int* vc = new unsigned int[PART_SIZE * (size - 1)];

    for (unsigned int i = 0; i < PART_SIZE * (size - 1); i++) {
        va[i] = rand() % 10;
        vb[i] = rand() % 10;
        vc[i] = 0;
    }

     MPI_Bcast(va, PART_SIZE * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);
    MPI_Bcast(vb, PART_SIZE * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    MPI_Request* requests = new MPI_Request[size - 1];
    MPI_Status* statuses = new MPI_Status[size - 1];

    for (unsigned int i = 0; i < size - 1; i++) {
        MPI_Irecv(vc + i * PART_SIZE, PART_SIZE, MPI_UNSIGNED, i + 1, 0, MPI_COMM_WORLD,
&requests[i]);
    }

    MPI_Waitall(size - 1, requests, statuses);

    for (unsigned int i = 0; i < (PART_SIZE * (size - 1)); i++) printf("%d\t", va[i]);
    printf("\r\n");

    for (unsigned int i = 0; i < (PART_SIZE * (size - 1)); i++) printf("%d\t", vb[i]);
    printf("\r\n");

    for (unsigned int i = 0; i < (PART_SIZE * (size - 1)); i++) printf("%d\t", vc[i]);
    printf("\r\n");

    delete[] va;
    delete[] vb;
    delete[] vc;
    delete[] requests;
    delete[] statuses;
}

void workerProcess(int id, int size) {

    const int PART_SIZE = 3;

    unsigned int* v = new unsigned int[PART_SIZE];
    unsigned int* va = new unsigned int[PART_SIZE * (size - 1)];
    unsigned int* vb = new unsigned int[PART_SIZE * (size - 1)];

    MPI_Bcast(va, PART_SIZE * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);
    MPI_Bcast(vb, PART_SIZE * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    for (unsigned int i = 0; i < PART_SIZE; i++) {
         v[i] = va[(id - 1) * PART_SIZE + i] + vb[(id - 1) * PART_SIZE + i];
    }

    MPI_Send(v, PART_SIZE, MPI_UNSIGNED, 0, 0, MPI_COMM_WORLD);
    delete[] v;
    delete[] va;
    delete[] vb;
}
```

ZADANIE 3:

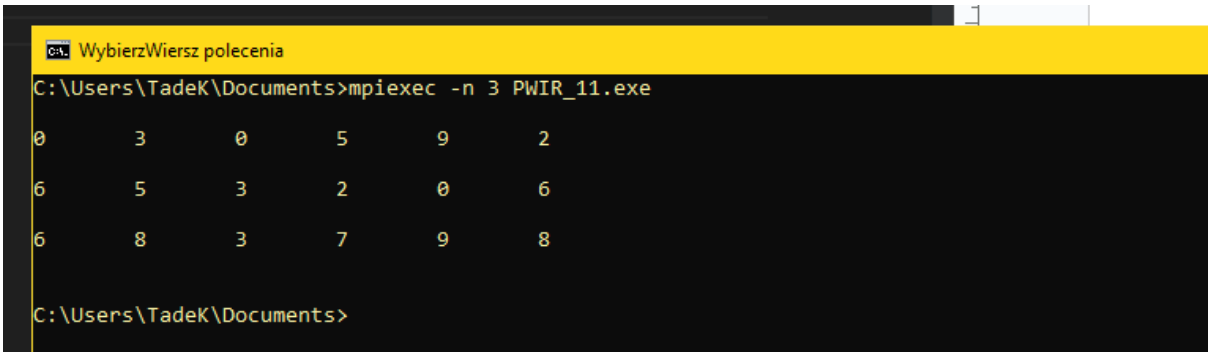
Zmodyfikowany kod do zadania trzeciego. Rozmiar wektorów pobierany jest od użytkowników:

```
#include "mpi.h"
#include <cstdio>
#include <iostream>
#include <time.h>

void mainProcess(int size, int vectorSize) {
    srand(time(NULL));

    // obliczamy ile elementów przypada na każdy proces
    int partSize = vectorSize / (size - 1);

    // alokujemy wektory o rozmiarze (partSize (ilosc procesów-1))
    unsigned int* va = new unsigned int[partSize * (size - 1)];
    unsigned int* vb = new unsigned int[partSize * (size - 1)];
    unsigned int* vc = new unsigned int[partSize * (size - 1)];

    // wypełniamy a i b losowymi danymi, a vc zerujemy
    for (unsigned int i = 0; i < partSize* (size - 1); i++) {
        va[i] = rand() % 10;
        vb[i] = rand() % 10;
        vc[i] = 0;
    }

    // broadcastujemy rozmiar wektora i partSize do pozostałych procesów
    MPI_Bcast(&vectorSize, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&partSize, 1, MPI_INT, 0, MPI_COMM_WORLD);

    // broadcastujemy wektor a do pozostałych procesów
    MPI_Bcast(va, partSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // broadcastujemy wektor b do pozostałych procesów
    MPI_Bcast(vb, partSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // odpalamy nasłuch
    MPI_Request* requests = new MPI_Request[size - 1];
    MPI_Status* statuses = new MPI_Status[size - 1];
    for (unsigned int i = 0; i < size - 1; i++) {
        MPI_Irecv(vc + i * partSize, partSize, MPI_UNSIGNED, i + 1, 0, MPI_COMM_WORLD,
&requests[i]);
    }
    MPI_Waitall(size - 1, requests, statuses);

    // wypisujemy wyniki
    for (unsigned int i = 0; i < (partSize * (size - 1)); i++) printf("%d\t", va[i]);
    printf("\r\n");
    for (unsigned int i = 0; i < (partSize * (size - 1)); i++) printf("%d\t", vb[i]);
    printf("\r\n");
    for (unsigned int i = 0; i < (partSize * (size - 1)); i++) printf("%d\t", vc[i]);
    printf("\r\n");

    // zwalniamy pamięć
    delete[] va;
    delete[] vb;
    delete[] vc;
    delete[] requests;
    delete[] statuses;
}

void workerProcess(int id, int size) {
    int vectorSize, partSize;

    // broadcastujemy rozmiar wektora i partSize
    MPI_Bcast(&vectorSize, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&partSize, 1, MPI_INT, 0, MPI_COMM_WORLD);

    // alokujemy buffor na moją część zadania
    unsigned int* v = new unsigned int[partSize];

    // alokujemy miejsce na wektor a oraz b
```

```
    unsigned int* va = new unsigned int[partSize * (size - 1)];
    unsigned int* vb = new unsigned int[partSize * (size - 1)];

    // nasłuchujemy bcasta wektora a
    MPI_Bcast(va, partSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // nasłuchujemy bcasta wektora b
    MPI_Bcast(vb, partSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // liczymyę
    for (unsigned int i = 0; i < partSize; i++) {
        v[i] = va[(id - 1) * partSize + i] + vb[(id - 1) * partSize + i];
    }

    // odsyłamy wynik
    MPI_Send(v, partSize, MPI_UNSIGNED, 0, 0, MPI_COMM_WORLD);

    // zwalniamy pamięć
    delete[] v;
    delete[] va;
    delete[] vb;
}

int main()
{
    int PID, PCOUNT;
    int vectorSize;

    MPI_Init(NULL, NULL);

    MPI_Comm_rank(MPI_COMM_WORLD, &PID);
    MPI_Comm_size(MPI_COMM_WORLD, &PCOUNT);

    if (PID == 0) { // jestem procesem głównym
        std::cout << "Podaj rozmiar wektora: ";
        std::cin >> vectorSize;
        mainProcess(PCOUNT, vectorSize);
    }
    else { // jestem procesem roboczym
        workerProcess(PID, PCOUNT);
    }

    MPI_Finalize();

    return 0;
}
```

Przykładowe uruchomienia programu:

ZADANIE 2:



ZADANIE 3: