Łukasz Oprych Gr. Lab. 5 Informatyka Techniczna	Podstawy komunikacji MPI	Data 23.12.2023

Cel ćwiczenia:

Zapoznanie się z podstawowym przesyłaniem komunikatów, składnią, kompilacją MPI.

Przebieg ćwiczenia:

Po przygotowaniu środowiska, struktury katalogowej, instalacji paczki mpicc zgodnie z poleceniem prowadzącego, przystąpiono do wykonania programu MPI_Simple.c

Zdefiniowanie liczby procesów w Makefile, w tym przypadku 6:

```
run: MPI_simple
	$(MPI_run) -np 6 ./MPI_simple

# zaleznosci i komendy
```

Utworzenie tablicy hostname w celu przesłania do kolejnych procesów adresu nadawcy oraz receive_hostname w celu odebrania przez proces owego adresu. Użyto funkcji gethostname() do pobrania adresu nadawcy.

Użyto MPI_Send do przesłania ID procesu, oraz hostname'a który składa się z m.in. jego rozmiaru, typu zmiennej, docelowego procesu, oznaczenia wiadomości, komunikatora COMM_WORLD

Użyto MPI_Recv do odebrania ID procesu oraz hostname'a, który składa się z m.in. jego rozmiaru (tablica o wymiarze 100, ID o wymiarze 1), typu zmiennej, docelowego procesu, tagu wiadomości, aby rozróżnić procesy w przypadku przesłania więcej niż jednej wiadomości, komunikatora oraz statusu.

Na koniec wypisano ID procesu oraz hostname nadawcy.

```
16  if(size>1){
17
18    if( rank != 0 ){ dest=0; tag=0;
19       char hostname[100];
20       gethostname(hostname, 100);
21      MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
22      MPI_Send( &hostname, 100, MPI_CHAR, dest, tag+1, MPI_COMM_WORLD);
23    } else {
24      char receive_hostname[100];
25      for( i=1; i<size; i++ ) {
26        MPI_Recv( &ranksent, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status );
27        MPI_Recv( &receive_hostname, 100, MPI_CHAR, i, 1, MPI_COMM_WORLD, &status );
28        printf("Dane od procesu o randze (status.MPI_SOURCE ->) %d: %d (i=%d) otrzymany
  hostname: %s\n", status.MPI_SOURCE, ranksent, i, receive_hostname );
29      }
30
31    }
32
33  }
34  else{
35        printf("Pojedynczy proces o randze: %d (brak komunikatów)\n", rank);
36  }
```

Wynik:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_11/MPI_simple/simple$ make run
mpiexec -oversubscribe -np 6 ./MPI_simple
Dane od procesu o randze (status.MPI_SOURCE ->) 1: 1 (i=1) otrzymany hostname: loprych-VirtualBox
Dane od procesu o randze (status.MPI_SOURCE ->) 2: 2 (i=2) otrzymany hostname: loprych-VirtualBox
Dane od procesu o randze (status.MPI_SOURCE ->) 3: 3 (i=3) otrzymany hostname: loprych-VirtualBox
Dane od procesu o randze (status.MPI_SOURCE ->) 4: 4 (i=4) otrzymany hostname: loprych-VirtualBox
Dane od procesu o randze (status.MPI_SOURCE ->) 5: 5 (i=5) otrzymany hostname: loprych-VirtualBox
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_11/MPI_simple/simple$
```

Następnie utworzono katalog o nazwie sztafeta i utworzono plik sztafeta.c, gdzie skopiowano początkowo kod z pliku MPI_simple.c i dostosowano do polecenia:

```c
int rank, size;
MPI_Status status;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int send_value = 10;
int receive_value;

if (size > 1) {
//proces poczatkowy
  if (rank == 0) {
    MPI_Send(&send_value, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  //proces srodkowy
  else if (rank < size - 1) {
    MPI_Recv(&receive_value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, &status);
    printf("Wątek %d odebrał liczbę %d, od wątka %d\n", rank, receive_value, rank - 1);
    receive_value++;
    MPI_Send(&receive_value, 1, MPI_INT, (rank + 1) % size, 0, MPI_COMM_WORLD);
  }
  //proces koncowy
  else {
    MPI_Recv(&receive_value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, &status);
    printf("Wątek %d odebrał liczbę %d, od wątka %d\n", rank, receive_value, rank - 1);
  }
} else {
  printf("Pojedynczy wątek o ranku: %d (brak komunikatów)\n", rank);
}

MPI_Finalize();
```

Początkowo program zaczyna się od inicjowania MPI oraz pobrania liczby procesów za pomocą MPI_Comm_Size oraz własnego identyfikatora (rangi) dla poszczególnego procesu.

W celu identyfikacji procesów użyliśmy zmiennej rank odpowiadającej za rangę procesu. Dostępne procesy podzielono na 3 grupy, proces początkowy, procesy środkowe oraz proces końcowy. Proces początkowy nr 0 odpowiada w tym przypadku jedynie za przesłanie zmiennej send_value o nadanej wartości 10 do procesu nr 1 używając MPI_Send. W skład procesów środkowych wchodzą wszystkie wątki poza początkowym i ostatnim, te procesy kolejno odbierają wartość zmiennej receive_value od poprzedniego za pomocą MPI_Recv, podbijają jej wartość o 1 i przesyłają dalej do kolejnego procesu przy użyciu MPI_Send. Proces końcowy jedynie odbiera wartość receive_value za pomocą MPI_Recv, ostatni proces kończy sztafetę.

Wynik programu sztafeta:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_11/MPI_simple/sztafeta$ make run
mpicc -c -g -DDEBUG sztafeta.c
mpicc -g -DDEBUG sztafeta.o -o sztafeta -lm
mpiexec -oversubscribe -np 6 ./sztafeta
Wątek 1 odebrał liczbę 10, od wątka 0
Wątek 2 odebrał liczbę 11, od wątka 1
Wątek 3 odebrał liczbę 12, od wątka 2
Wątek 4 odebrał liczbę 13, od wątka 3
Wątek 5 odebrał liczbę 14, od wątka 4
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_11/MPI_simple/sztafeta$
```

Wnioski:

Na podstawie programu sztafeta.c możemy zauważyć standardową komunikację typu punkt-punkt, która odnosi się do bezpośredniej wymiany informacji między parą konkretnych procesów. MPI_Send oraz MPI_Recv są funkcjami o blokującej charakterystyce polecenia. Program wstrzymuje wykonanie aż do wysłania/odebrania wiadomości, pomaga to nam łatwo obsługiwać synchronizację i sterować przebiegiem programu. Nieblokującymi zastępnikami owych funkcji są MPI_Isend oraz MPI_Irecv, które natychmiast przekazują sterowanie dalszym instrukcjom programu, pozwala to na komunikację asynchroniczną i program może być wykonywany natychmiastowo. Można również stosować mieszaną kombinację blokujących i nieblokujących procedur. Zależnie od postawionego przed nami rodzaju problemu oba rodzaje funkcji będą mieć swoje zastosowanie.