Pierwsze kroki w środowisku MPI

Zakres ćwiczenia

W tym ćwiczeniu dowiesz się, w jaki sposób napisać swój pierwszy program wykorzystujący środowisko MPI i jak taki program uruchomić.

Etapy tworzenia i uruchamiania aplikacji równoległej w środowisku MPI

W celu uruchomienia przetwarzania pod kontrolą środowiska MPI niezbędne jest podjęcie następujących kroków:

A. Przygotowanie kodów źródłowych programów w języku C lub Fortran:

W kodzie źródłowym programu można wykorzystać funkcje z biblioteki MPI, do których interfejs dla języka C znajduje się w pliku nagłówkowym `mpi.h`, a interfejs do języka Fortran znajduje się odpowiednio w pliku nagłówkowym `mpif.h`.

Przykładowy program w języku C, przystosowany do uruchomienia w środowisku MPI, pokazano poniżej. Program ten znajdziesz w pliku `hellow.c` w katalogu `\mpich2-1.0.3\examples`, po rozpakowaniu pliku `mpich2-1.0.3.tar.gz`. Plik `mpich2-1.0.3.tar.gz` możesz pobrać ze strony implementacji MPICH2 standardu MPI-2: http://www-unix.mcs.anl.gov/mpi/mpich/.

Implementację MPICH2 zrealizowano w Argonne National Laboratory, który jest wiodącym ośrodkiem prowadzącym badania nad przetwarzaniem rozproszonym. Ponieważ w ramach niniejszych ćwiczeń korzystamy z implementacji MPICH2, ze zrozumiałych względów często będziemy się odwoływać do tych badań.

```
/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *  (C) 2001 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int rank;
    int size;

    MPI_Init( 0, 0 );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Zwróć uwagę na postać tego programu. W początkowej części programu znajduje się dyrektywa preprocesora:

```
#include ″mpi.h″
```

Plik nagłówkowy `mpi.h` zawiera definicje, makra i prototypy funkcji konieczne dla kompilacji programu MPI.

W pętli głównej programu znajduje się funkcja `MPI_init`. Funkcja ta inicjuje obliczanie w środowisku MPI, przed tą funkcją w programie nie ma żadnej innej funkcji MPI. Następnie występują dwie kolejne funkcje MPI: `MPI_Comm_rank` i `MPI_Comm_size`. Pierwsza z nich, `MPI_Comm_rank`, podaje numer Twojego procesu, druga, `MPI_Comm_size` podaje liczbę procesów. Ostatnia z funkcji MPI w programie, `MPI_Finalize` kończy obliczanie w środowisku MPI. Po funkcji `MPI_Finalize` w programie nie może się znaleźć żadna inna funkcja MPI. Jak łatwo zauważyć, program ten powoduje wyświetlenie pozdrowienia od każdego procesu, z podaniem ile jest razem procesów.

B. Kompilacja kodów źródłowych i konsolidacja z odpowiednimi bibliotekami:

1. Utworzenie projektu w środowisku Microsoft Visual Studio. NET

Uruchom środowisko Microsoft Visual Studio .NET. Pojawia okno Microsoft Development Environment [design] Start Page. Wybierz File→New→Blank Solution. Pojawi się wpis Blank Solution w okienko Solution Explorera w prawej górnej ćwiartce okna Start Page. Nadaj temu rozwiązaniu nazwę, w tym wypadku będzie to nazwa **hello**. Otrzymasz w okienku Solution Explorera wpis: Solution 'hello' (0 projects). Kliknij prawym przyciskiem myszy na Solution 'hello' i dodaj projekt do tego rozwiązania, za pomocą Add→New Project. Pojawi się okienko Add New Project. W okienku tym, w jego lewym panelu wybierz z listy wpis Visual C++ Projects, a w prawym panelu wybierz Managed C++ Application. Poniżej podaj nazwę projektu, tym razem **hello_proj**, i zatwierdź lokalizację projektu w następującym katalogu:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello
```

gdzie w miejsce podkatalogu `michal` powinna się pojawić Twoja nazwa użytkownika w systemie Windows.

W tym momencie, w okienku Solution Explorera pojawia się projekt o nazwie **hello_proj** z „drzewkiem" dołączonych plików. W projekcie **hello_proj** otrzymujesz strukturę, w której występują katalogi: Source Files, Header Files, i Reasource Files.

Cała struktura Solution 'hello' wygląda następująco:

```
Solution 'hello' (1 project)
      hello_proj
            Source Files
                  hello_proj.cpp
                  AssemblyInfo.cpp
                  stdafx.cpp
            Header Files
                  stdafx.h
            Resource Files
            ReadMe.txt
```

Zobacz co jest w pliku `ReadMe.txt` (wystarczy, że na niego klikniesz, a tekst pojawi się w lewym górnym okienku). Jest to informacja o wyżej wymienionych plikach.

```
========================================================================
    APPLICATION : hello_proj Project Overview
========================================================================

AppWizard has created this hello_proj Application for you.

This file contains a summary of what you will find in each of the
files that
make up your hello_proj application.

hello_proj.vcproj
    This is the main project file for VC++ projects generated
using an Application Wizard.
    It contains information about the version of Visual C++ that
generated the file, and
    information about the platforms, configurations, and project
features selected with the
    Application Wizard.

hello_proj.cpp
    This is the main application source file.

AssemblyInfo.cpp
      Contains custom attributes for modifying assembly metadata.

/////////////////////////////////////////////////////////////////////
Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you
should add to or customize.

/////////////////////////////////////////////////////////////////////
```

Z wydruku tego możesz się zorientować, że Twój projekt znajduje się w pliku `hello_proj.vcproj`, a głównym plikiem źródłowym Twojej aplikacji jest plik `hello_proj.cpp`. Dowiadujesz się również, że środowisko Microsoft Visual Studio .NET za pomocą Application Wizard tworzy pewną ramę dla Twojej aplikacji, którą sam musisz wypełnić.

W podobny sposób możesz sprawdzić, co znajduje się w plikach `AssemblyInfo.cpp`, `stdafx.cpp` i `stdafx.h`. Nie musisz robić tego teraz. Zobacz natomiast co znajduje się w pliku `hello_proj.cpp`. Jest to właśnie ten plik źródłowy, wygenerowany przez Application Wizard, który powinieneś uzupełnić kodem Twojego programu.

Plik `hello_proj.cpp` początkowo wygląda następująco:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

// This is the entry point for this application
int _tmain(void)
{
    // TODO: Please replace the sample code below with your own.
    Console::WriteLine(S"Hello World");
    return 0;
}
```

Kod Twojego programu, o nazwie `hellow.c`, znajdziesz w katalogu:

`C:\Program Files\MPICH2\examples\moje przyklady\`

do którego wcześniej powinieneś go skopiować z katalogu `examples`.

Przenieś teraz kod programu `hellow.c` do Solution Explorera rozwiązania **'hellow'**, a dokładnie do projektu **hello_proj** w środowisku Microsoft Visual Studio .NET. W tym celu wejdź w okienko Solution Explorera, wybierz projekt **hello_proj,** kliknij prawym klawiszem myszy, wybierz Add →Add existing item i wskaż lokalizację:

`C:\Program Files\MPICH2\examples\moje przyklady\hellow.c`

Wtedy plik `hellow.c` pojawi się na liście Source Files w Solution Explorerze. Możesz go wyświetlić, klikając na jego nazwę. Zawartość pliku `hellow.c` pojawi się w lewym górnym okienku w oknie Microsoft Visual Studio .NET:

```
/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *  (C) 2001 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int rank;
    int size;

    MPI_Init( 0, 0 );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Oczywiście jest to ten sam program, który poznałeś na początku tego ćwiczenia.

Korzystając z Solution Explorera, podglądu obu programów, i operacji kopiowania *Ctrl-C* i wklejania *Ctrl-V*, wklej kod programu `hellow.c` do pliku `hello_proj.cpp`, za kodem pliku `hello_proj.cpp`. Otrzymasz następującą postać pliku `hello_proj.cpp`:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

// This is the entry point for this application
int _tmain(void)
{
    // TODO: Please replace the sample code below with your own.
    Console::WriteLine(S"Hello World");
    return 0;
}
/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *  (C) 2001 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int rank;
    int size;

    MPI_Init( 0, 0 );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Usuń w tym pliku wiersze od:

```
//This is the entry point ...
```

do końca automatycznie generowanego szablonu, czyli pierwszego wystąpienia nawiasu `}` po `return 0;`. Wiersze do usunięcia zacieniowano.

Otrzymujesz nową postać pliku `hello_proj.cpp`:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *  (C) 2001 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int rank;
    int size;

    MPI_Init( 0, 0 );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Na koniec, usuń program `hellow.c` korzystając z listy Source Files w oknie Solution Explorera. Zaznacz program i naciśnij delete. Program `hellow.c` nie będzie już więcej potrzebny.

2. Kompilacja kodów źródłowych i konsolidacja z odpowiednimi bibliotekami

Proces kompilacji i konsolidacji rozpocznij od ustawienia odpowiednich cech w Twoim projekcie `hellow_proj`.

W tym celu kliknij na nazwę projektu w oknie Solution Explorera. W okienku Properties, w prawym dolnym rogu okna Microsoft Visual Studio .NET, pojawia się napis: **hellow_proj Project properties** oraz cztery ikony. Ikona z prawej nazywa się Property pages. Kliknij na nią, wtedy rozwinie się okienko z cechami. Wprowadź następujące ustawienia.

W wierszu Configuration Properties →C/C++ wybierz General. W panelu po prawej stronie, w wierszu Additional Include Directories wpisz, korzystając z listy podpowiedzi (po naciśnięciu przycisku **...** i ikony katalogu), następujący katalog:

```
C:\Program Files\MPICH2\include
```

W wierszu Configuration Properties →Linker wybierz General. W panelu po prawej stronie, w wierszu Additional Library Directories wpisz, korzystając z listy podpowiedzi (po naciśnięciu przycisku **...** i ikony katalogu), następujący katalog:

```
C:\Program Files\MPICH2\lib
```

W wierszu Configuration Properties →Linker wybierz Input. W panelu po prawej stronie, w wierszu Additional Dependencies wpisz, korzystając z listy podpowiedzi (po naciśnięciu przycisku **...** i ikony katalogu), następujące elementy na wejściu do konsolidatora (bez przecinków!):

```
mpi.lib cxx.lib
```

a gdy planujesz korzystanie z Fortranu:

```
mpi.lib cxx.lib fmpich2.lib
```

W tym momencie możesz dokonać kompilacji i konsolidacji programu. Na początku nie wszystko się udaje, gdyż w implementacji MPI-2 mogą się pojawić konflikty.

Wybierz z paska okna Twojego projektu w środowisku Microsoft Visual Studio .NET, które gdy wyświetlasz program `hello_proj.cpp`, przyjmuje nazwę:

hello_proj Microsoft Visual C++ [design] hello_proj.cpp

polecenie Build →Build Solution.

Otrzymujesz następujący zapis z procesu kompilacji i konsolidacji, w lewym dolnym okienku okna projektu:

```
------ Build started: Project: hello_proj, Configuration: Debug Win32 ------

Compiling...
stdafx.cpp
Compiling...
hellow.c
\Program Files\MPICH2\examples\moje przyklady\hellow.c(22) : fatal error C1010:
unexpected end of file while looking for precompiled header directive

Build log was saved at "file://c:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects\hello\hello_proj\Debug\BuildLog.htm"
hello_proj - 1 error(s), 0 warning(s)


---------------------- Done ----------------------

    Build: 0 succeeded, 1 failed, 0 skipped
```

Jeżeli pojawia się powyższy komunikat, oznacza to, że nie usunąłeś z Solution Explorera pliku `hellow.c`. Obecność tego pliku nie pozwala na zakończenie kompilacji, z powodu błędu o numerze C1010, oznaczającego niespodziewany koniec pliku.

Usuń plik `hellow.c` i ponownie wydaj polecenie Build Solution. Pojawia się kolejna informacja o przebiegu procesu kompilacji:

```
------ Build started: Project: hello_proj, Configuration: Debug Win32 ------

Compiling...
stdafx.cpp
Compiling...
AssemblyInfo.cpp
hello_proj.cpp
c:\Program Files\MPICH2\include\mpicxx.h(26) : fatal error C1189: #error :
"SEEK_SET is #defined but must not be for the C++ binding of MPI"
Generating Code...

Build log was saved at "file://c:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects\hello\hello_proj\Debug\BuildLog.htm"
hello_proj - 1 error(s), 0 warning(s)


---------------------- Done ----------------------

    Build: 0 succeeded, 1 failed, 0 skipped
```

Pojawił się tu znany błąd, opisywany w *MPICH2 User's Guide,* polegający na tym, że zarówno `stdio.h`, jak i interfejs MPI C++, stosują `SEEK_SET`, `SEEK_CUR` i `SEEK_END`. Jest to błąd w standardzie MPI-2. Aby ten błąd usunąć, wpisz w pliku `hello_proj.cpp`, tuż przed `mpi.h`, następujące trzy wiersze:

```
#undef SEEK_SET
#undef SEEK_END
#undef SEEK_CUR
```

Otrzymasz następującą postać pliku `hello_proj.cpp`:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *  (C) 2001 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include <stdio.h>

#undef SEEK_SET
#undef SEEK_END
#undef SEEK_CUR

#include "mpi.h"
```

```
int main( int argc, char *argv[] )
{
    int rank;
    int size;

    MPI_Init( 0, 0 );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Ponownie wykonaj Build Solution. Otrzymasz następujący wynik kompilacji i konsolidacji:

```
------ Build started: Project: hello_proj, Configuration: Debug Win32 ------

Compiling...
hello_proj.cpp
Linking...
LINK : warning LNK4098: defaultlib 'LIBCMT' conflicts with use of other libs; use
/NODEFAULTLIB:library
cxx.lib(initcxx1.obj) : warning LNK4204: 'c:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects\hello\hello_proj\Debug\vc70.pdb' is missing
debugging information for referencing module; linking object as if no debug info

Build log was saved at "file://c:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects\hello\hello_proj\Debug\BuildLog.htm"
hello_proj - 0 error(s), 2 warning(s)


---------------------- Done ----------------------

    Build: 1 succeeded, 0 failed, 0 skipped
```

Kompilacja i konsolidacja zakończyły się poprawnie, niemniej podczas konsolidacji pojawiło się ostrzeżenie, które w tej chwili pominiemy. Zobacz teraz, co otrzymałeś jako wynik procesu kompilacji i konsolidacji. W tym celu przejdź do katalogu z Twoim projektem:

`C:\Documents and Settings\michal\Moje dokumenty\Visual Studio Projects\liczba_pi\hello_proj`

Wejdź następnie do katalogu `Debug`. Sprawdź, czy jest tam być plik `hellow_proj.exe`.

Jest to plik wynikowy z Twoim projektem.

3. Uruchomienie programu z wiersza poleceń systemu operacyjnego Windows

Aby sprawdzić czy program w ogóle działa, bez angażowania początkowo poleceń implementacji MPICH2 środowiska MPI, wystarczy uruchomić program `hellow_proj.exe` z wiersza poleceń systemu operacyjnego Windows.

W tym celu wejdź w polecenie Uruchom i wpisz `cmd`. Pojawi się okno poleceń systemu Windows. Przejdź do katalogu:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hello_proj\Debug
```

Pamiętaj, że zamiast podkatalogu `michal` jest Twoja nazwa użytkownika.

Sprawdź istnienie pliku `hellow_proj.exe` poleceniem `dir hellow_proj.exe`. Pojawi się okno z informacją o programie `hellow_proj.exe`:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hel
lo_proj\Debug>dir hello_proj.exe
 Wolumin w stacji C to IBM_PRELOAD
 Numer seryjny woluminu: 303C-7AE1

 Katalog: C:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects
\hello\hello_proj\Debug

2006-08-03  13:45           229 376 hello_proj.exe
               1 plik(ów)         229 376 bajtów
               0 katalog(ów)  40 552 341 504 bajtów wolnych

C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hel
lo_proj\Debug>
```

W oknie tym uruchom program, wpisując `hello_proj.exe`. Otrzymasz następujący obraz okienka poleceń:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hel
lo_proj\Debug>hello_proj.exe
Hello world from process 0 of 1

C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hel
lo_proj\Debug>
```

Widać, że program się wykonał, generując napis:

```
Hello world from process 0 of 1
```

Otrzymałeś wynik wykonania Twojego programu. Do jego wytworzenia wykorzystano jednak tylko jeden proces, gdyż zgłosił się jedynie proces o numerze 0. W kolejnym punkcie przekonasz się jak uruchomić Twój program z wykorzystaniem wielu procesów.

4. Uruchomienie programu w środowisku MPI

Twoim zadaniem jest uruchomienie programu `hello_proj.exe` z wykorzystaniem wielu procesów. Rozpocznij od uruchomienia środowiska MPI.

Uruchom aplikację MPICH2, w zwykły sposób: Start→Wszystkie programy → MPICH2. Pokazuje się lista opcji, wybierz program `wmpiexec` realizujący interfejs graficzny do uruchamianie Twojego programu w środowisku MPI. Pojawia się okno programu `wmpiexec`, w którym możesz wskazać plik wynikowy Twojego programu, oraz zaznaczyć, że chcesz go uruchomić na określonej liczbie procesorów. Okno programu `wmpiexec` nazywa się MPIEXEC wrapper. W oknie tym dokonaj następujących wpisów.

W okienku Application wybierz pełną ścieżkę do pliku wynikowego, w tym wypadku jest to plik `hello_proj.exe`. Pełna ścieżka do tego pliku to:

```
"C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hello_proj\Debug\hello_proj.exe"
```

Najprościej wybrać ją korzystając z przycisku przeglądania, oznaczonego trzema kropkami.

Następnie ustaw liczbę procesów na 4, zaznacz polecenie uruchomienia w oddzielnym oknie: run in separate window, oraz wybierz myszką przycisk show command, którego naciśnięcie wyzwala pokazanie wykorzystanego do uruchomienia programu polecenia środowiska MPI. Na pasku okna MPIEXEC wrapper pojawia się pełna ścieżka do polecenia mpiexec:

```
"C:\Program Files\MPICH2\bin\mpiexec.exe" -n 4  -noprompt
"C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hello_proj\Debug\hello_proj.exe"
```

Następnie wskaż myszką przycisk execute. Naciśnięcie tego przycisku wyzwala wydanie polecenia mpiexec w środowisku MPI-2, w postaci pokazanej powyżej. W wyniku jego wykonania otrzymujemy wynik programu w oknu poleceń systemu Windows:

```
Hello world from process 2 of 4
Hello world from process 1 of 4
Hello world from process 0 of 4
Hello world from process 3 of 4
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Widać, że każdy z 4 procesów, o numerach od 0 do 3, wysłał pozdrowienie. Przy kolejnym uruchomieniu, pozdrowienia przychodzą w innej kolejności:

```
Hello world from process 3 of 4
Hello world from process 2 of 4
Hello world from process 1 of 4
Hello world from process 0 of 4
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Na koniec, wykonaj uruchomienie pliku wynikowego, `hello_proj.exe`, na 4 procesorach, bezpośrednio w katalogu z poleceniem `mpiexec`, rezygnując z interfejsu `wmpiexec`.

W tym celu wejdź z wiersza poleceń systemu Windows do katalogu ze środowiskiem MPI-2:

```
C:\Program Files\MPICH2\bin\
```

I wykonaj polecenie:

```
mpiexec.exe -n 4  -noprompt C:\Documents and
Settings\michal\Moje dokumenty\Visual Studio
Projects\hello\hello_proj\Debug\hello_proj.exe
```

Oczywiście znowu pamiętaj, że katalog `michal` jest zastąpiony Twoją nazwą użytkownika. W oknie poleceń systemu Windows wygląda to tak:

```
C:\Program Files\MPICH2\bin>mpiexec.exe -n 4  -noprompt
C:\Documents and Setting
s\michal\Moje dokumenty\Visual Studio
Projects\hello\hello_proj\Debug\hello_proj
.exe
Hello world from process 3 of 4
Hello world from process 2 of 4
Hello world from process 1 of 4
Hello world from process 0 of 4

C:\Program Files\MPICH2\bin>
```

Oczywiście otrzymujesz wynik podobny do rezultatów osiąganych z wykorzystaniem interfejsu graficznego `wmpiexec`.

Komunikacja dwupunktowa w środowisku MPI

A. Idea komunikacji dwupunktowej:

Komunikacja dwupunktowa jest podstawowym mechanizmem komunikacji w środowisku MPI. Polega ona na tym, że komunikaty są przesyłane między parą procesów, z których jeden jest stroną nadającą, a drugi stroną odbierającą. Komunikację tę realizują dwie funkcje MPI: `MPI_Send` i `MPI_Recv`. Funkcja `MPI_Send` nadaje komunikat, a funkcja `MPI_Recv` go odbiera. Obie te funkcje, łącznie z czterema funkcjami poznanymi na początku ćwiczeń, uznane są za funkcje o charakterze podstawowym.

Należy tu wyraźnie zaznaczyć, że w ogólności, nadawanie i odbiór może mieć charakter blokujący i nieblokujący.

Nadawanie blokujące oznacza, że funkcja nadawania blokuje się do czasu odzyskania bufora nadawczego. Odbiór blokujący oznacza, że funkcja odbioru blokuje się do czasu, gdy w buforze odbiorczym pojawi się cały komunikat.

Funkcje nieblokujące nadawania i odbioru pozwalają na nakładanie się przesyłania komunikatów na ich przetwarzanie, albo na nakładanie się na siebie przesyłania komunikatów. Funkcje nieblokujące zazwyczaj dzielą się na dwa rodzaje: funkcje zgłaszające i funkcje testujące zakończenie realizacji. Zadaniem funkcji nieblokujących jest zwiększenie efektywności przesyłania i obliczeń.

Funkcje `MPI_Send` i `MPI_Recv`, zastosowane w przykładzie poniżej, mają charakter blokujący.

B. Przygotowanie kodów źródłowych programów w języku C:

Dokonamy obecnie modyfikacji znanego nam kodu źródłowego `hellow.c`, korzystając z dokumentacji MPI. Przypomnijmy, że gdy uruchomiłeś ten program w wersji pierwotnej, każdy proces wysyłał pozdrowienia. Obecnie, w wyniku wprowadzenia modyfikacji, którą w kodzie źródłowym zaznaczono za pomocą cieniowania, program po uruchomieniu działa następująco. Proces 0 odbiera komunikaty od pozostałych procesów, ale nic nie nadaje. Każdy z pozostałych procesów nadaje jeden komunikat do procesu 0. To co otrzymuje proces 0 jest wyświetlane. Kod źródłowy ze zmianami nazwij `hellow_except0.c`.

```
#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int rank;
    int size;
    int source;
    int dest;
    int tag=50;
    char message[100];
    MPI_Status status;
```

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

    if(rank != 0)
    {
      sprintf(message, ”Hello world from process %d of %d\n”,
rank, size);
      dest = 0;
      MPI_Send(message, strlen(message)+1, MPI_CHAR, dest, tag,
MPI_COMM_WORLD);
    }
    else
      for (source = 1; source<size; source++)
      {
        MPI_Recv(message, 100, MPI_CHAR, source, tag,
MPI_COMM_WORLD, &status);
        printf(”%s\n”, message);
      }
    MPI_Finalize();
    return 0;
}
```

Zadanie do samodzielnego wykonania

Dokonaj kompilacji, konsolidacji oraz uruchomienia programu z poprzedniego punktu, zgodnie z poznanymi w tym ćwiczeniu zasadami. Dokonaj analizy otrzymanych wyników.

Podsumowanie

Podczas tego ćwiczenia miałeś okazję przygotować program w języku C, skompilować go i skonsolidować w środowisku Microsoft Visual Studio .NET, a następnie uruchomić w środowisku implementacji MPICH standardu MPI-2.

Co powinieneś wiedzieć:

- Jak kompilować i konsolidować programy w środowisku Microsoft Visual Studio .NET.
- Jak uruchamiać zadania w środowisku MPI.
- Co to jest komunikacja dwupunktowa.