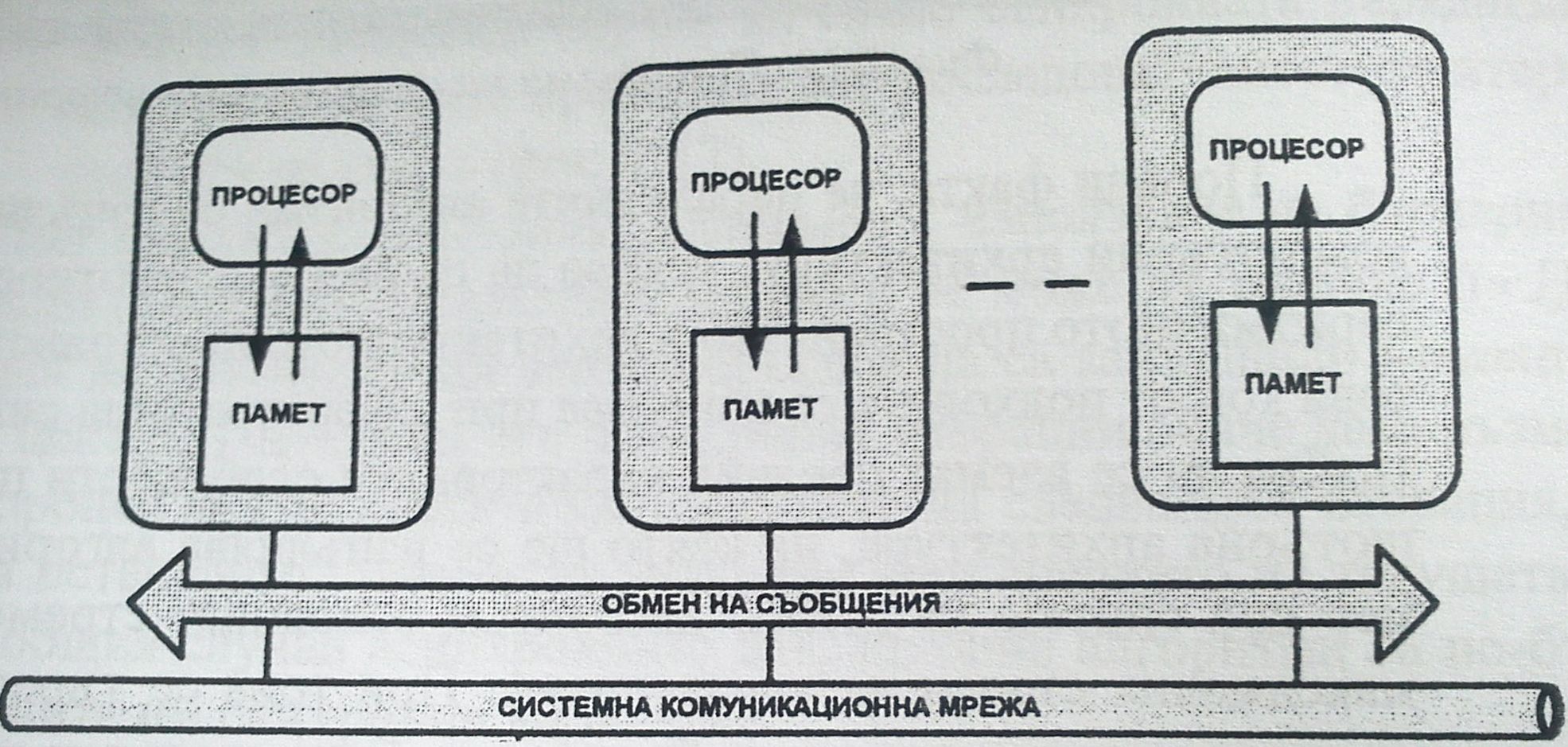
**Лабораторно упражнение 10**

**Въведение в MPI**

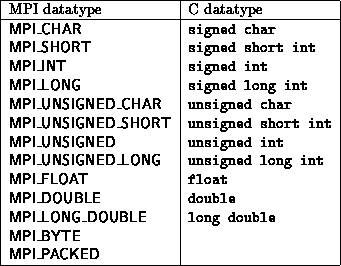
**Същност на МPI**

MPI(Message Passing Interface) е най-популярната спецификация за обмен на съобщения.. Библиотеките са свободно достъпни за ползване на произволен клъстер и осигуряват преносимост при различни платформи с различна производителност. MPI моделът предполага мултикомпютърна архитектура с множество процесори, всеки с локална памет. Системната комуникационна мрежа поддържа обмена на съобщения между всяка двойка процесори. Всеки процес може да комуникира с всеки друг процес. Паралелната програма обхваща множество процеси, всеки от които си има идентификатор наречен ранг и състояние. Изпълнението на паралелната програма прилкючва само тогава, когато са завършили изпълнението си всичките и процеси. При стартирането на програмата се определя броя на паралелните процеси. В общия случай той остава постоянен в хода на изпълнение на програмата. Всеки процес алтернативно извършва изчисления над локалните си променливи и комуникира с останалите процеси или В/И устройства. **Процесите използват съобщенията както за комуникация, така и за синхронизация**.  
Синтезът на паралелни алгоритми с конструиране на граф на задачите естествено води до парадигмата за програмиране с обмен на съобщения. Парадигмата с обмен на съобщения дава възможност на програмиста да управлява използването на паметта и да увеличи локалността. Запазването на на локалността на съобщенията към паметта е ключов фактор за оптимизиране на производителниостта, както при мултипроцесорите така и при мултикомпютрите. Програмите за обмен на съобщения могат да се изпълняват ефективно за широк спектър паралелни компютърни платформи.

Предимства на модела с обмен на съобщения:

* Ефективност за широк спектър MIMD архитектури.
* Осигурява се естествена среда за мултикомпютърни платформи, които не поддържат глобално адресно пространство.
* Възможно е да се изпълняват програми с обмен на съобщения на мултипроцесорни платформи, като общите променливи се използват като буфери на съобщенията.
* Стимулира синтеза на алoгритми, които максимално използват локални изчисления и минимизират комуникациите.
* Осигурява висока честота на попаденията в кеша, а следователно и ефективно управление на паметта.
* Настройката на програми с обмен на съобщения е по-лесна отколкото тази на програми с общи променливи.

**MPICH** е преносима, свободно разпространяема имплементация на пълната спецификация на MPI за широк спектър паралелни изчислтелни среди. Създадена е от националната лаборатория на Аргон и държавния университет на Мисисипи. Последната версия на имплементацията е MPICH2. Разпространява се като сорс код или в двоичен формат за UNIX и Windows , както и други ОС. Освен MPI библиотеката за обмен на съобщения MPICH съдържа програмна среда за за работа с MPI програми. Програмната среда включва преносим механизъм за стартиране, както и няколко профилиращи библиотеки за оценка на производителността. MPI предоставя собствени типове данни, съответстващи на основните типове данни в езика C(табл.). В програмата променливите се декларират като С типове, а MPI имената на съответните типове се използват като аргумент на MPI функции.



MPI предоставя наколко специални типа данни(структури) за имплементация на езика С, например: MPI\_Comm за манипулатор на главния комуникатор, MPI\_Status за информация за статус на комуникационна операция, MPI\_Datatype за потребителски дефинирани типове.

**Основни елементи на една MPI програма**Основните програмни конструктори в MPI са функциите, които започват с префикса MPI

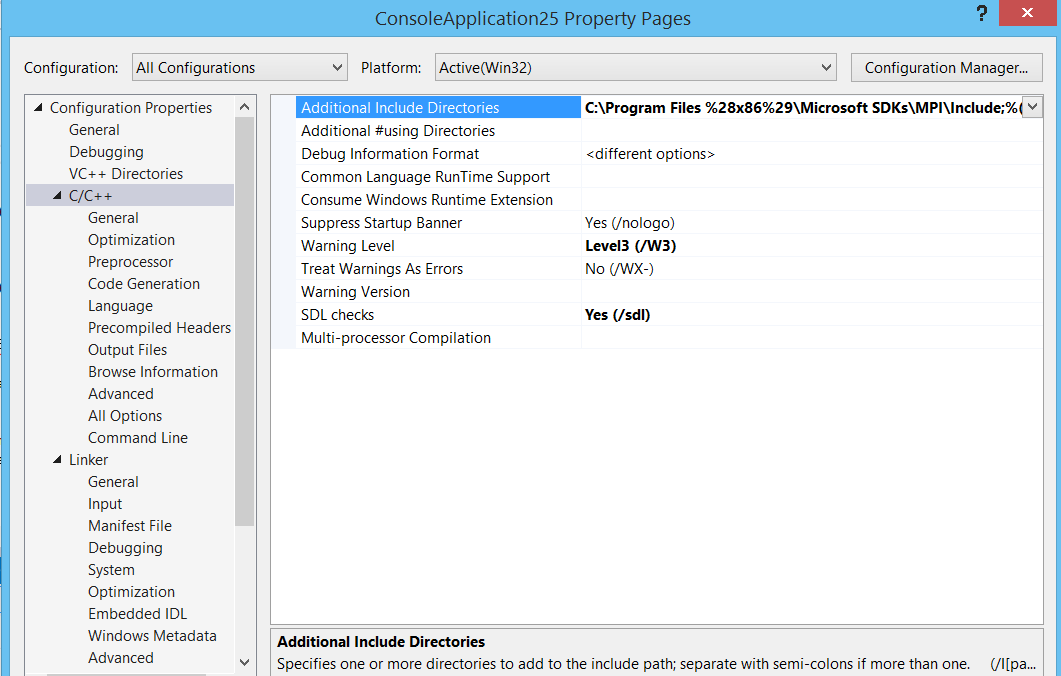
* **MPI\_Init(&argc, &argv).** MPI\_Init е функция за инициaлизиране на изпълнителната среда за обмен на съобщeния. Обръщението към нея е задължително за всяка MPI програма и винаги това е първата MPI функция в програмата. Допустимо е само едно обръщение към тази функция в програмата.  
  След инициализацията всеки активен процес става член на комуникатор, наречен **MPI\_COMM\_WORLD .** Комуникаторът представлява прозрачен обект, осигуряващ среда обмен на съобщения между процесите. В рамките на комуникатора процесите имат строго определен ред. Позицията на процеса в този ред се определя неговия идентификатор(ранг). В комуникатор с N процеса, всеки процес има уникален ранг(ID номер), вариращ между 0 и N-1. Комуникаторът специфицира наредена група комуникиращи процеси, формиращи областта на комуникацията. Съобщение, изпратено с един комуникатор може да бъде получено само със същия комуниктор.
* **MPI\_Comm\_rank(MPI\_Comm comm, int rank)** - даден процес може да използва ранга си, за да определи за коя част от изчисленията и данните е отговорен.
* **MPI\_Comm\_size (MPI\_Comm comm, int num\_of\_processes) –** определя броя на процесите.
* След завършване на всички MPI извиквания всеки процес трябва да направи обръщение към функцията **MPI\_Finalize**,за да освободят всички системни ресурси. С тази функция се затваря изпълнителната среда за обмен на съобщения и повече никакви обръщения към MPI функции не могат да бъдат осъществявани.

**Настройка на средата Visual Studio за разработка на паралелни програми**

* Сваляне и инсталиране на **Microsoft MPI**  от

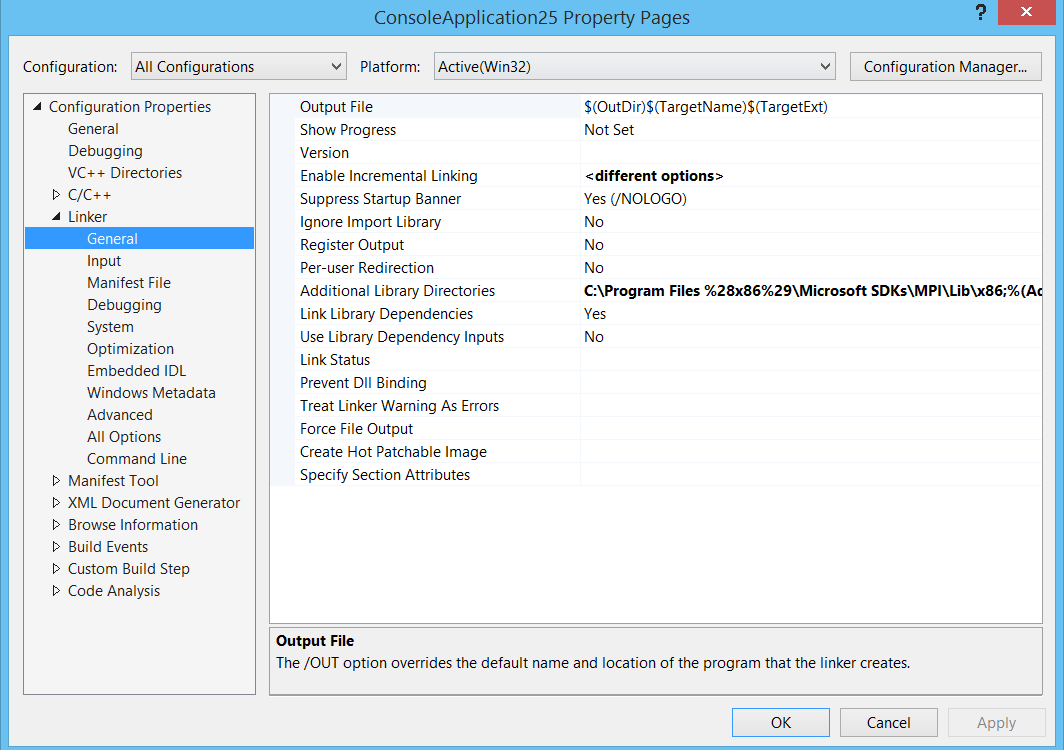
<https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi>

* Създаване на нов проект C++ за конзолно приложение във Visual Studio
* Добавяне на бибиотеката mpi.h в програмата
* Настройване на проекта със задаване на include директория за MPICH –

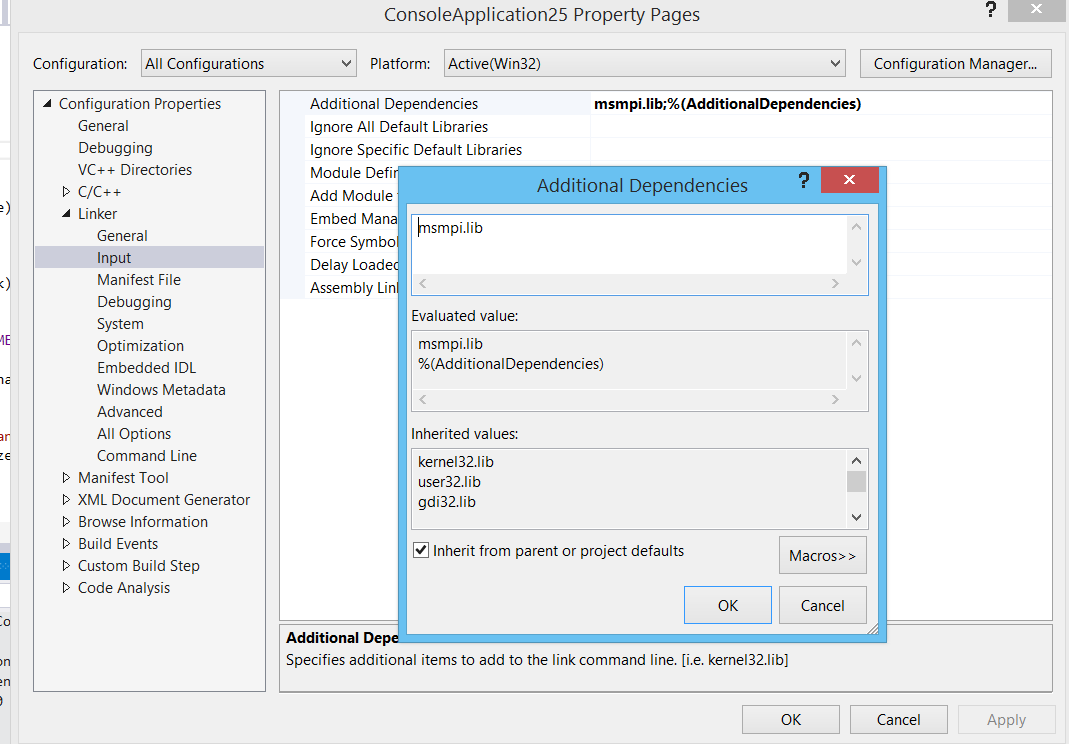
C:\Program Files x86\Microsoft SDKs\MPI\Include  


* Настройване на проекта със задаване на library директория за MPICH

C:\Program Files x86\Microsoft SDKs\MPI\Lib\x86



* Задават се допълнителни динамични библиотеки за MPICH

msmpi.lib  


**Задача:**Да се използват посочените по горе стъпки за настройка на средата и посочения по-долу код , за да се компилира и впоследствие стартира програмата “Hello World” използвайки приложението MPIexec. Броя на процесите се задава с –n.

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the number of processes

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

// Get the rank of the process

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

// Get the name of the processor

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

// Print off a hello world message

printf("Hello world from processor %s, rank %d out of %d processors\n",

processor\_name, world\_rank, world\_size);

// Finalize the MPI environment.

MPI\_Finalize();

int i;

}