

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Параллельные системы

- Системы с общей памятью
 - Многоядерные компьютеры
- Системы с распределенной памятью
 - Многопроцессорные кластеры

Многоядерный процессор

- Несколько ядер обращаются к общей оперативной памяти
 - Вычислительные ядра и оперативная память работают через системную шину
- Взаимодействие потоков – через глобальные переменные в памяти
 - Отсутствуют затраты времени на коммуникацию
 - Требуется корректно использовать общую память

Многопроцессорный кластер

- Каждый вычислительный узел – самостоятельный компьютер
 - Узлы соединены локальной сетью
 - Каждый узел работает со своей оперативной памятью
- Узлы обмениваются сообщениями
 - Затраты времени на коммуникацию
 - Не нужно следить trpf общей памятью

Процессы и потоки

- **Процесс** – программа в момент выполнения (запуск *.exe)
 - Программы могут обмениваться сообщениями
- **Поток (thread)** – вычисления «внутри» одного процесса
 - Потоки могут обращаться к глобальным переменным в оперативной памяти
 - Локальные переменные внутри потока недоступны другим потокам
- Процесс содержит хотя бы один поток

Параллельные потоки

- Программирование для многоядерных машин
- Функции работы с потоками
 - Windows Threads API
 - POSIX Threads API
 - Библиотеки распараллеливания потоков
 - OpenMP (Open Multi-Processing)
 - <http://openmp.org>

Параллельные процессы

- Организация обмена сообщениями между процессами
 - Функции операционной системы для обмена сообщениями
 - Библиотеки для обмена сообщениями
 - MPI (Message Passing Interface)
 - <http://www.mcs.anl.gov/research/projects/mpich2/>

Векторное распараллеливание

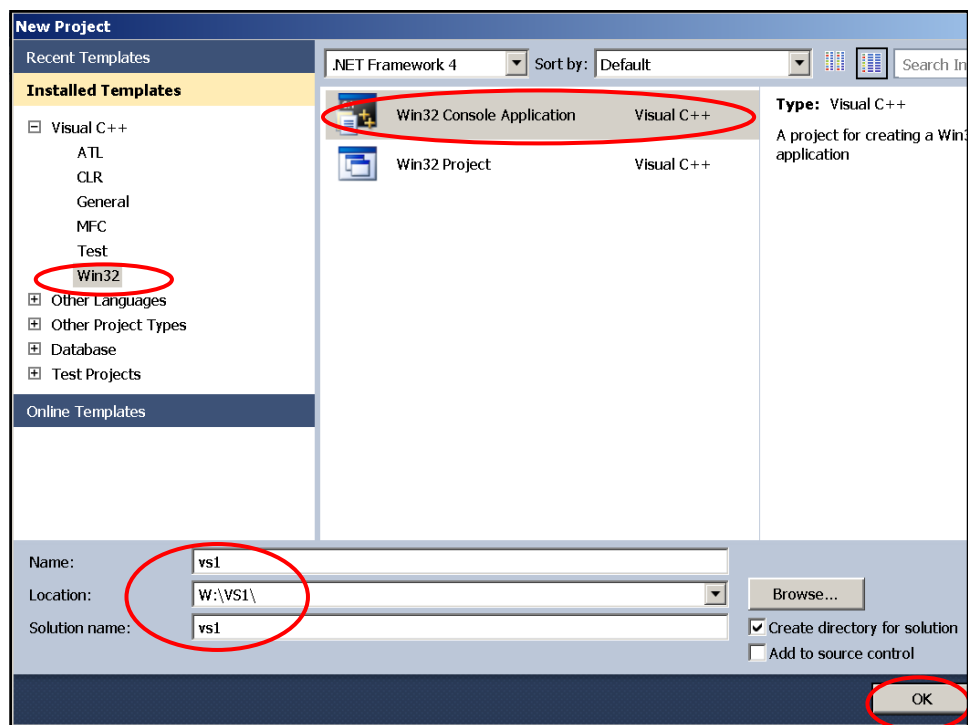
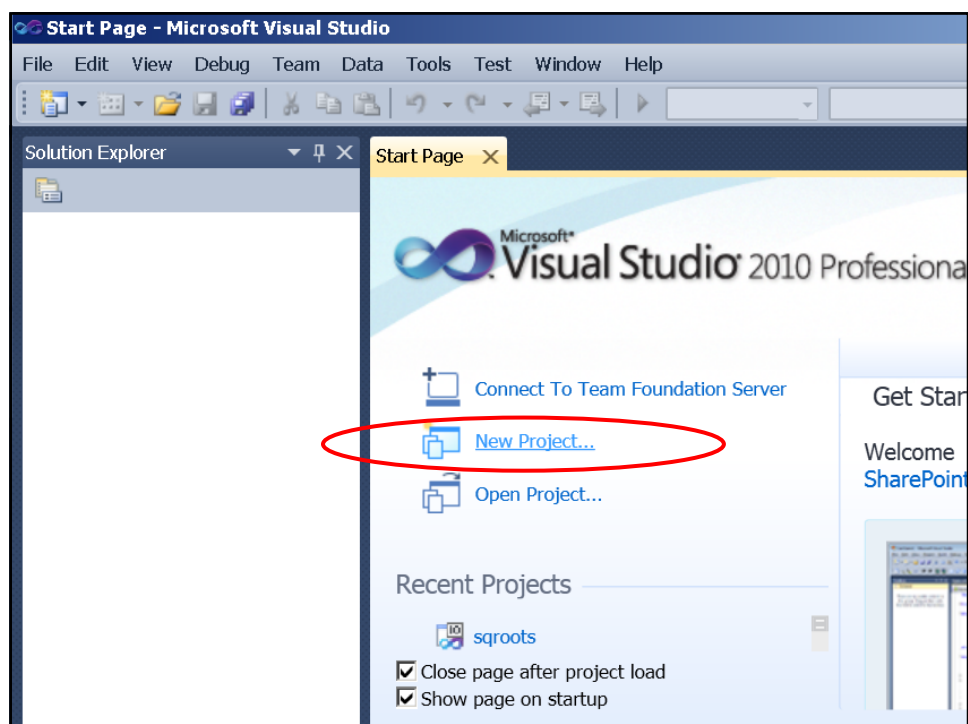
- Система команд процессора может включать команды обработки векторов
 - SSE (Streaming SIMD Extensions)
- Одна команда выполняет одну и ту же операцию над несколькими парами чисел одновременно
 - Сложение 4 пар чисел

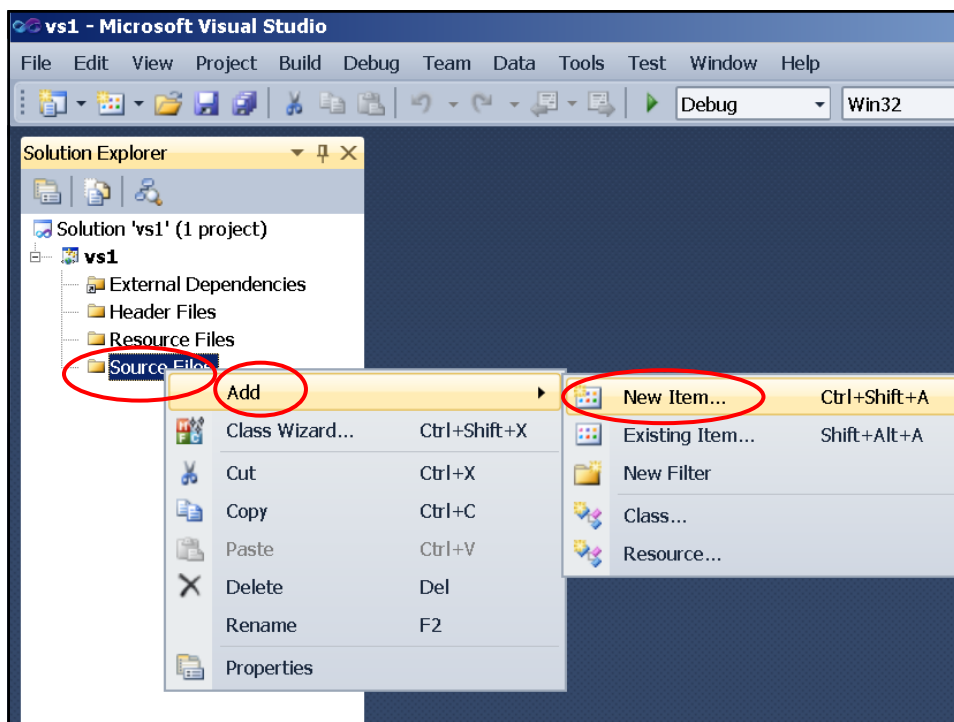
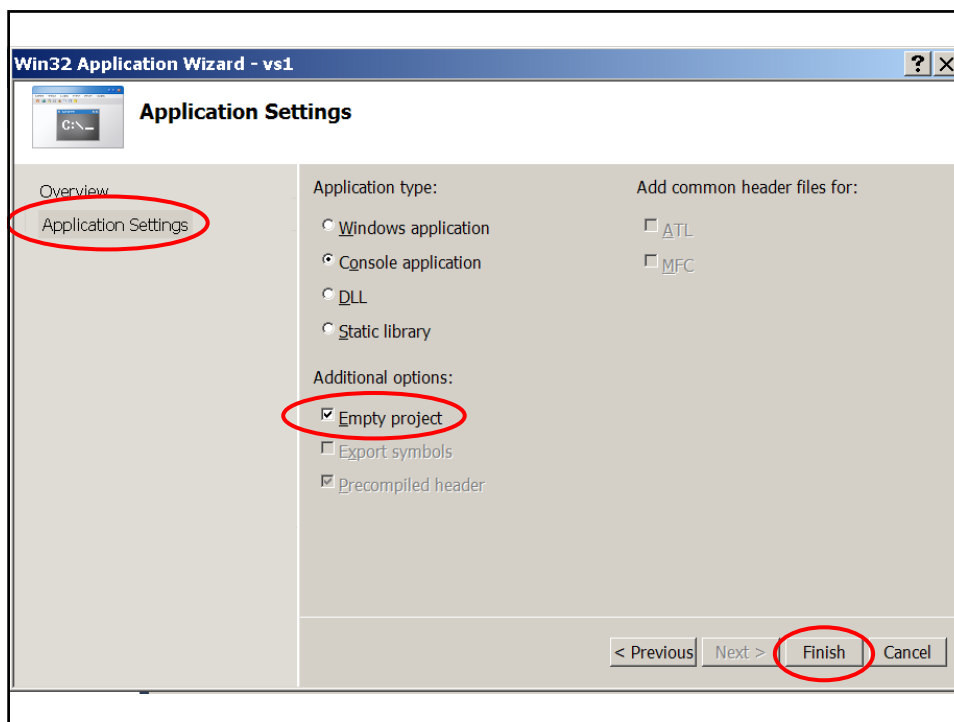
Visual Studio

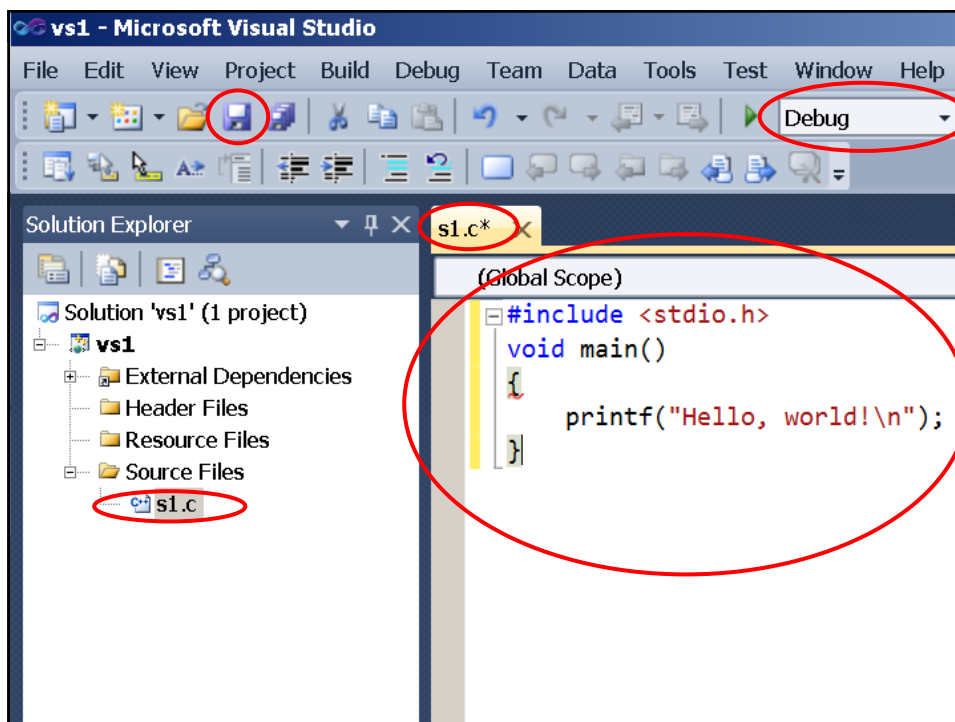
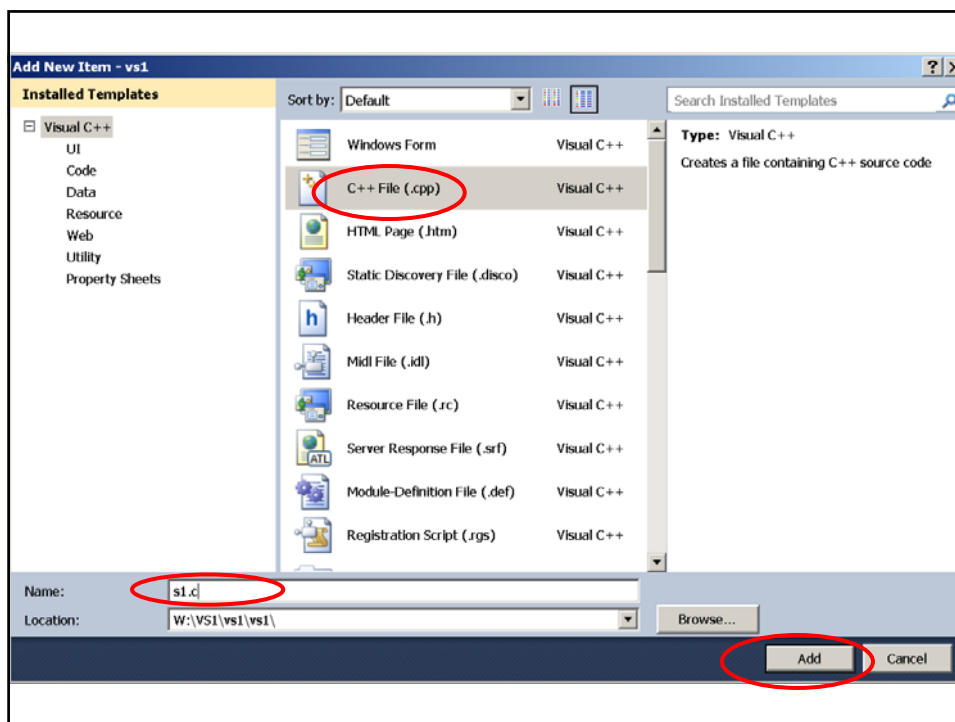
Среда программирования и
компиляторы

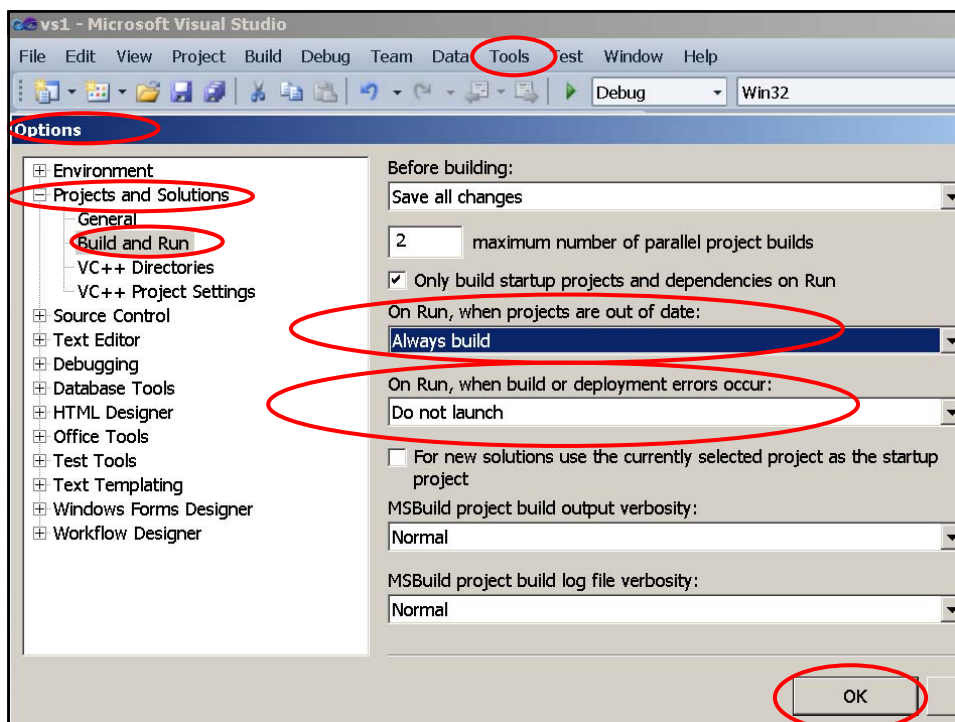
Компиляторы Visual C++ Visual Studio

- Visual C++ / Visual Studio 2008 / 2010 (EN / RU)
 - Express Edition
 - Professional
 - <http://www.microsoft.com/downloads/ru-ru/>
 - <http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express>
 - <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express>
 - <http://www.microsoft.com/visualstudio/ru-ru/products/2010-editions/visual-cpp-express>
- Студентам - бесплатно
 - <http://www.dreamspark.ru/>
- Варианты установки:
 - веб-установщик
 - ISO образ диска
 - виртуальная машина с пробной версией
 - Windows Virtual PC
 - MSt Virtual PC 2007 SP1
 - Windows Server 2008 Hyper-V



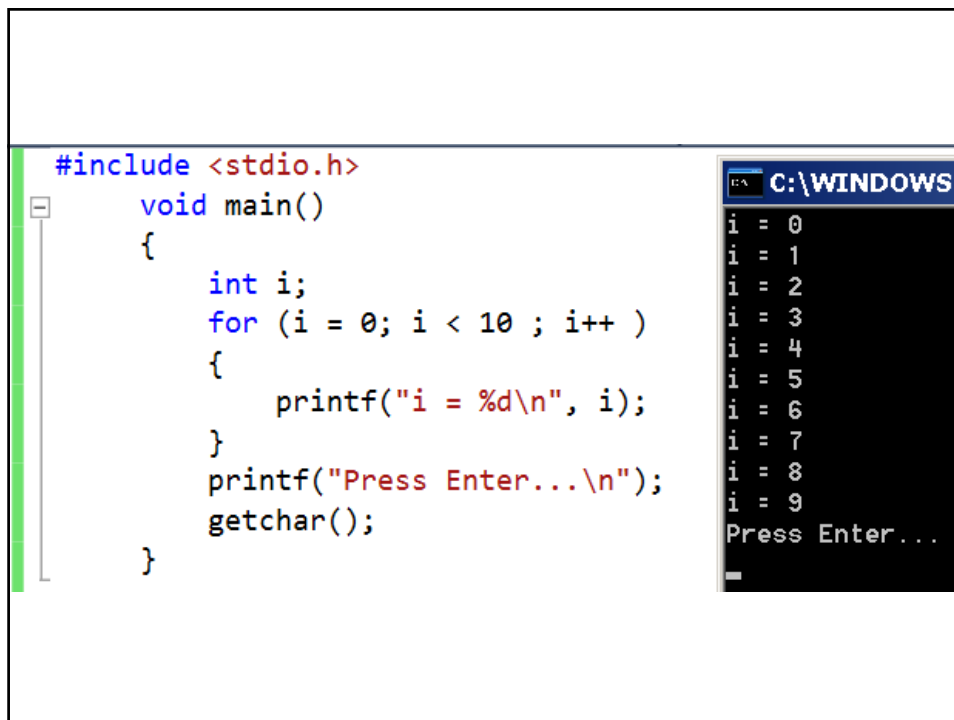
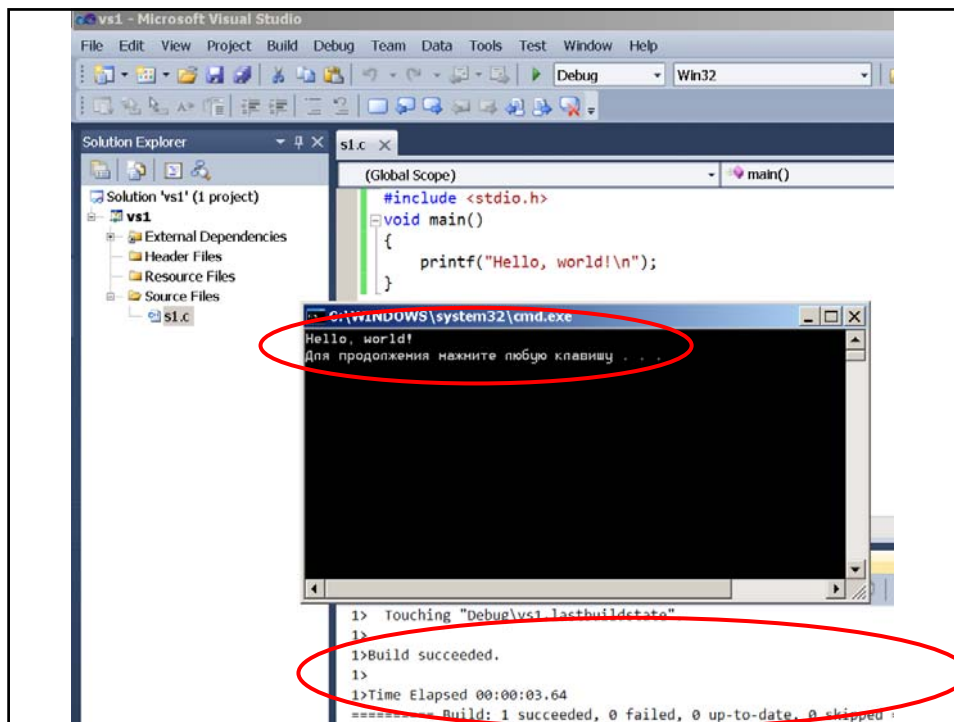


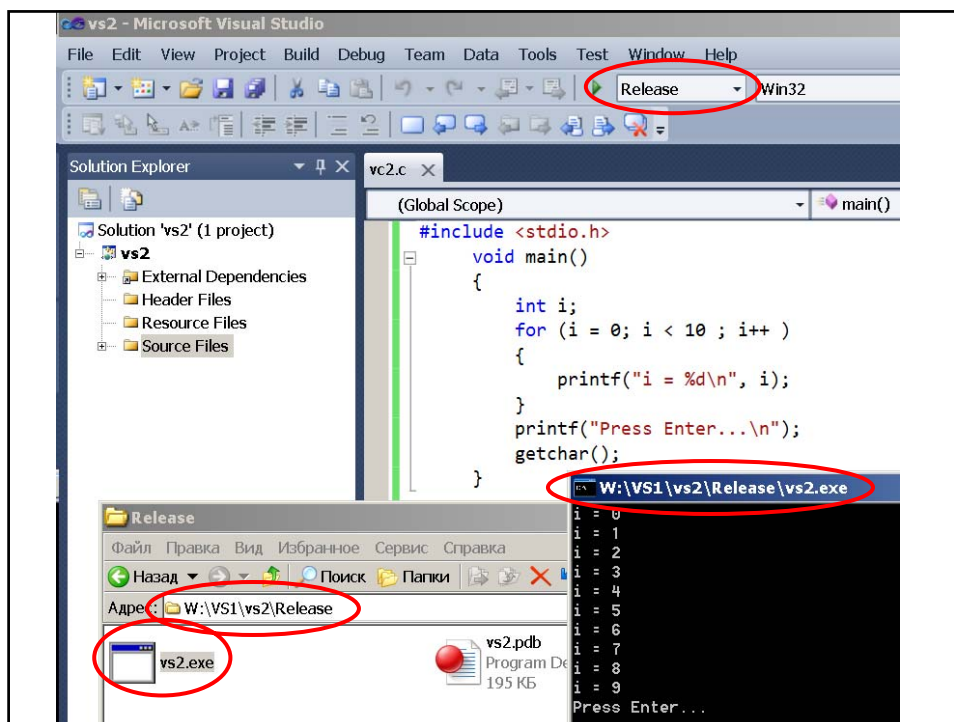




Функциональные клавиши Shortcuts

[F7] – создание исполняемого файла
[Ctrl+F5] – запуск программы на выполнение
[Ctrl + S] – сохранение файла





Параллельные потоки

Windows Threads

Потоки MS Windows

- Операционная система содержит готовые системные вызовы для работы с потоками
- Компилятор поддерживает вызов функций работы с потоками
- Организация взаимодействия потоков – ответственность программиста

Создание потока

- Win32
- CreateThread

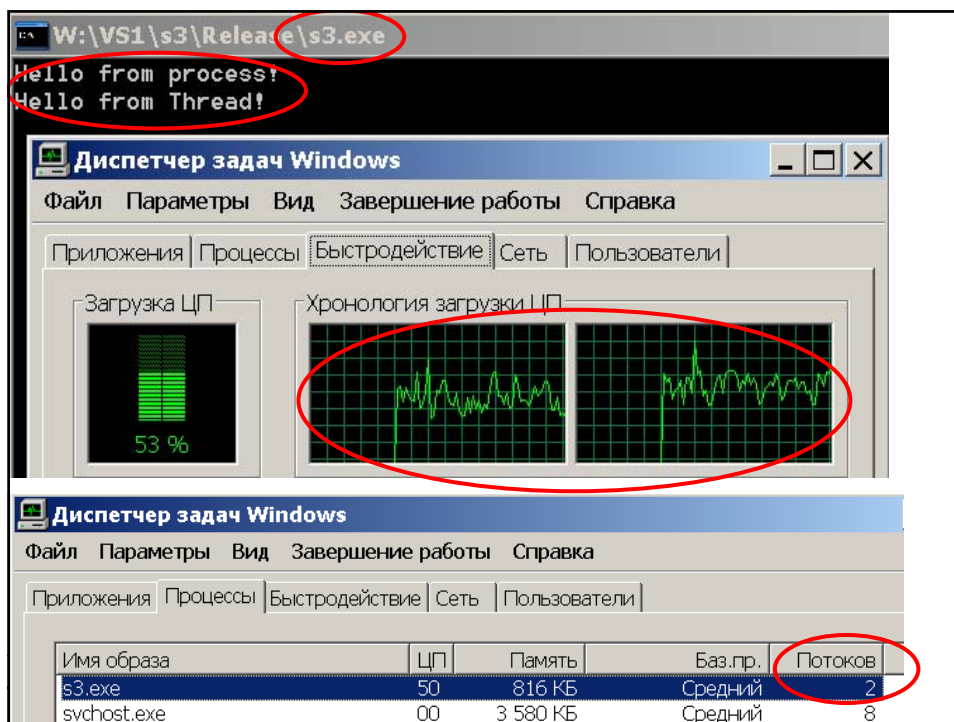
```
HANDLE Thread;  
Thread = CreateThread(NULL, 0,  
ThreadProc,  
(LPVOID)&GlobalVar, 0, NULL);
```

CreateThread

Creates a thread to execute within the virtual address space of the calling process.

```
HANDLE WINAPI CreateThread(  
    __in_opt LPSECURITY_ATTRIBUTES  
    lpThreadAttributes,  
    __in     SIZE_T dwStackSize,  
    __in     LPTHREAD_START_ROUTINE lpStartAddress,  
    __in_opt LPVOID lpParameter,  
    __in     DWORD dwCreationFlags,  
    __out_opt LPDWORD lpThreadId  
);
```

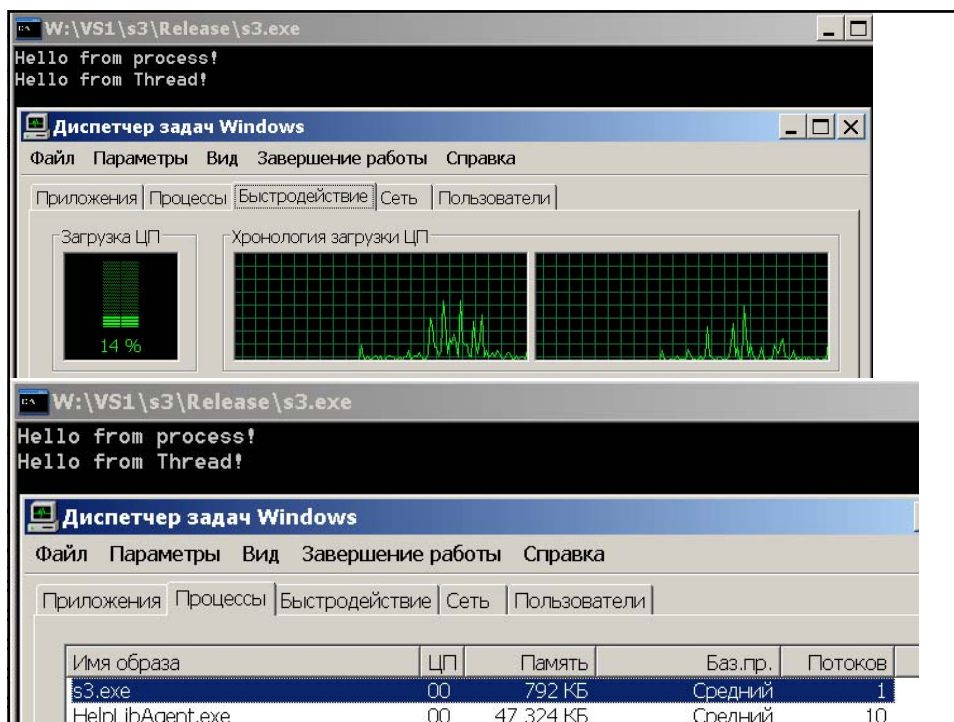
```
#include <windows.h>  
#include <stdio.h>  
  
void WINAPI ThreadProc()  
{  
    printf("Hello from Thread!\n");  
    while(1);  
}  
  
void main()  
{  
    HANDLE Thread;  
    Thread = CreateThread(0, 0,  
        (LPTHREAD_START_ROUTINE) ThreadProc, 0, 0, 0);  
    printf("Hello from process!\n");  
    getchar();  
}
```



```
#include <windows.h>
#include <stdio.h>

void WINAPI ThreadProc()
{
    printf("Hello from Thread!\n");
    //while(1);
}

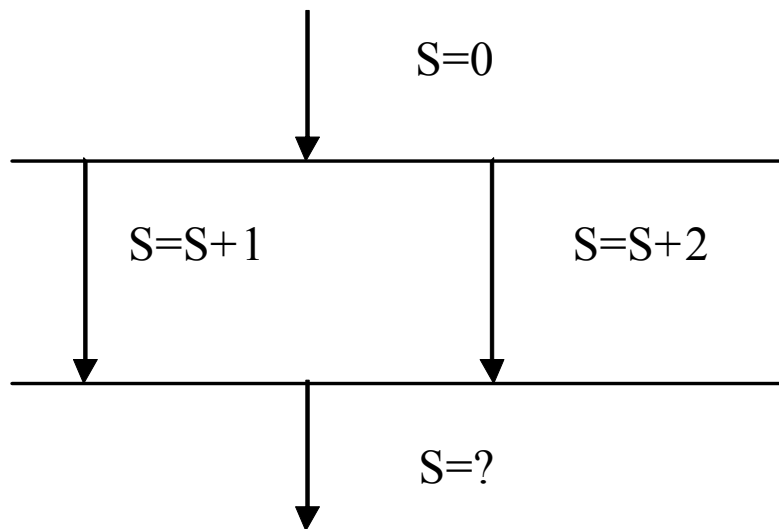
void main()
{
    HANDLE Thread;
    Thread = CreateThread(0, 0,
        (LPTHREAD_START_ROUTINE) ThreadProc, 0, 0, 0);
    printf("Hello from process!\n");
    getchar();
}
```



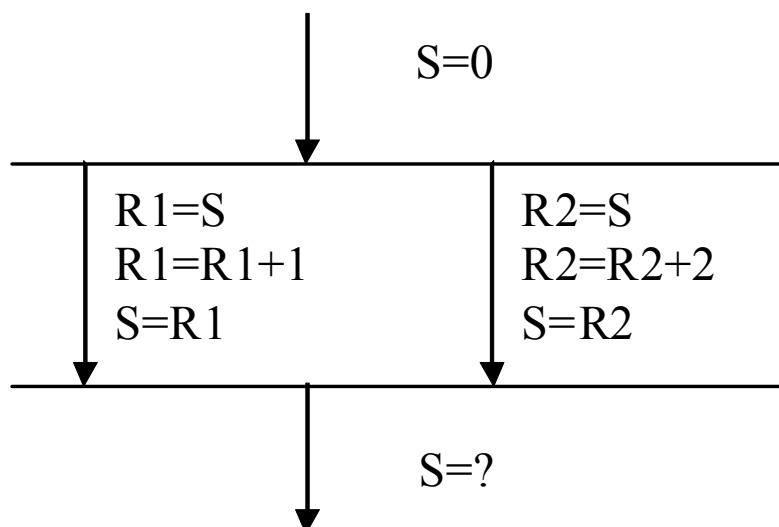
Data Race

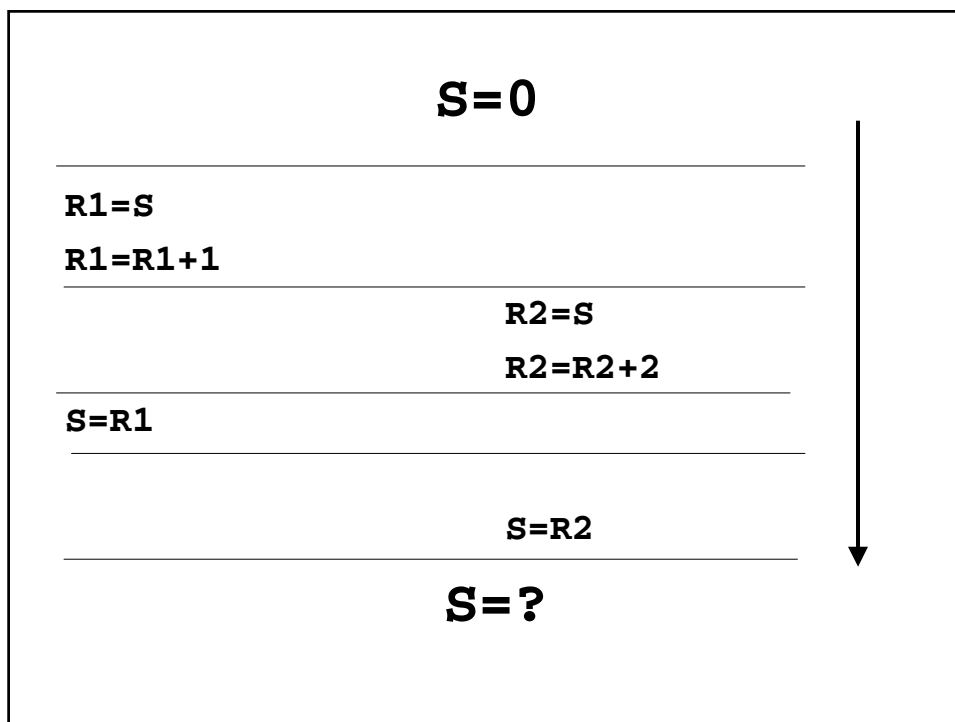
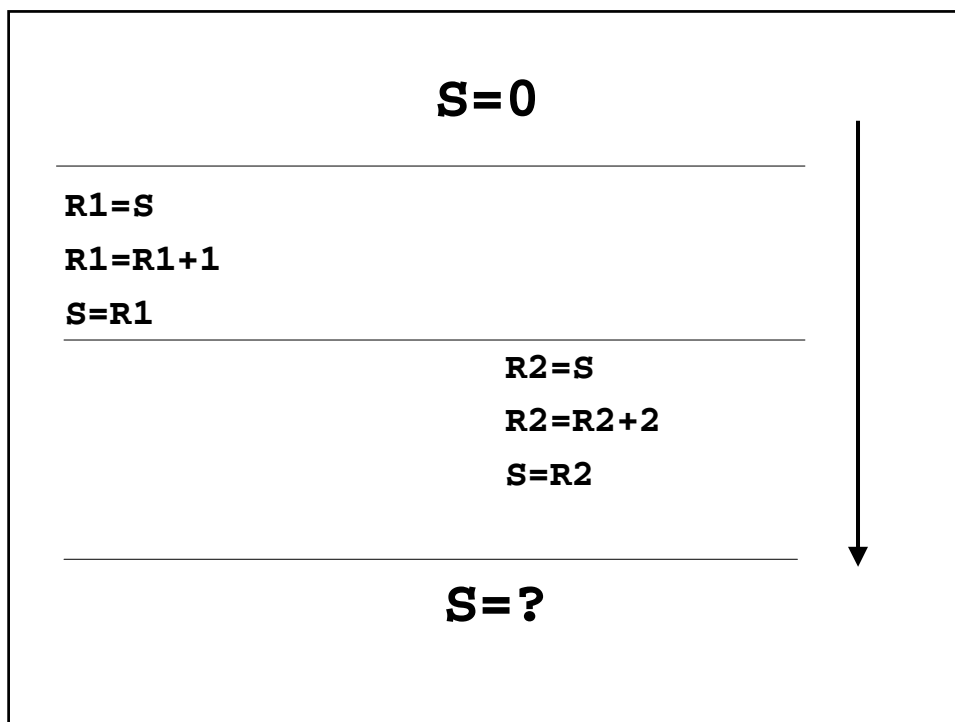
- Гонки, соревнование по доступу к данным
- Параллельный доступ вычислительных потоков к глобальным переменным
- Нарушение «целостности данных»
- Появление случайной ошибки в расчетах

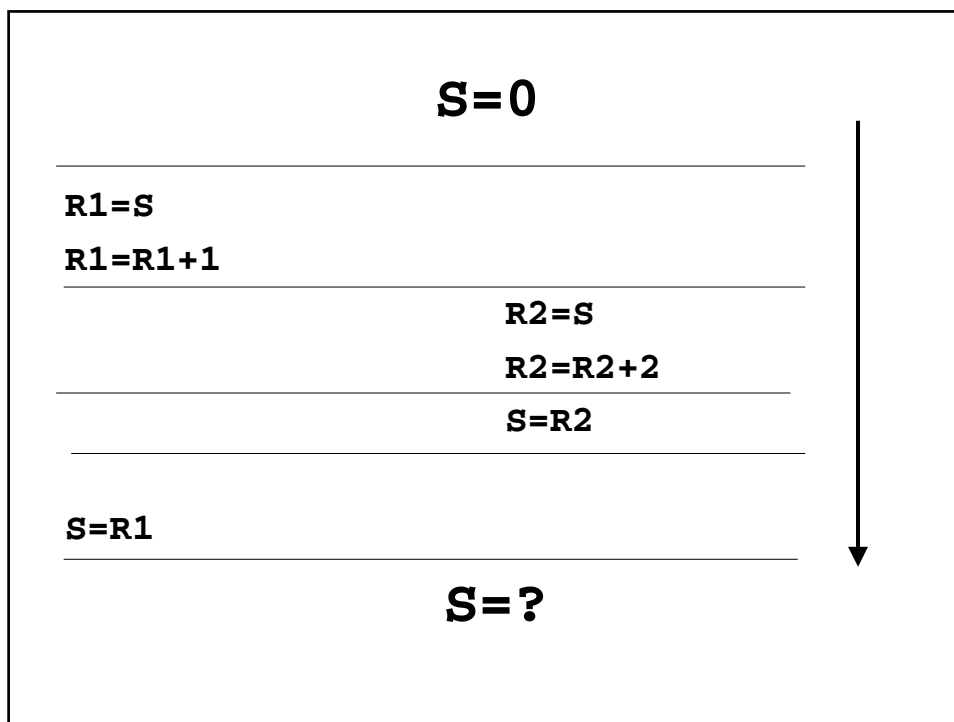
Параллельное суммирование



Чтение-запись регистры-ОЗУ

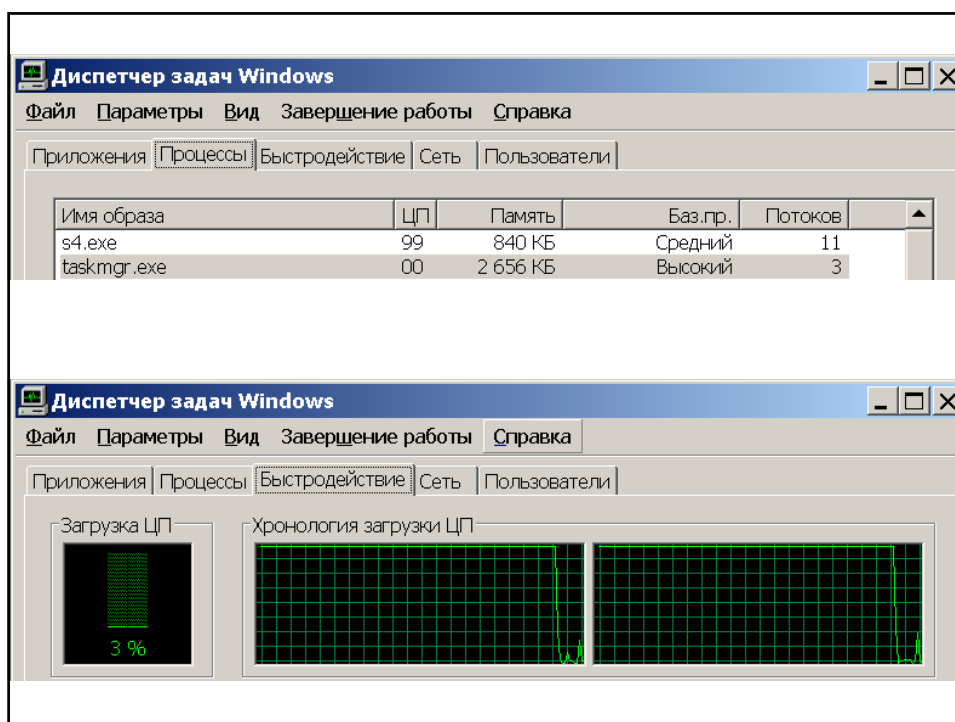






```
#include <windows.h>
#include <stdio.h>
#define NTh 200
#define Ns 10000000
int S = 0;
void WINAPI ThreadProc() {
    for (int i=0; i<Ns; i++)
        S = S + 1; }

void main() {
    HANDLE Thread [NTh];
    for (int i=0; i < NTh; i++)
        Thread [i] = CreateThread(0, 0,
            (LPTHREAD_START_ROUTINE) ThreadProc, 0, 0, 0);
    WaitForMultipleObjects(NTh, Thread, TRUE, INFINITE);
    printf("Ns*NTh = %d\n      S = %d\nError =
        %d\n", Ns*NTh, S, Ns*NTh-S);
    getchar(); }
```



```
#define NTh 200
#define Ns 10000000
int S = 0;
...
printf("Ns*NTh = %d\n      S = %d\nError
      = %d\n", Ns*NTh, S, Ns*NTh-S);
```

```
W:\VS1\s4\Release\s4.exe
Ns*NTh = 2000000000
S = 1980000000
Error = 20000000
```

```
W:\VS1\s4\Release\s4.exe
Ns*NTh = 2000000000
S = 2000000000
Error = 0
```

```
W:\VS1\s4\Release\s4.exe
Ns*NTh = 2000000000
S = 1930000000
Error = 70000000
```

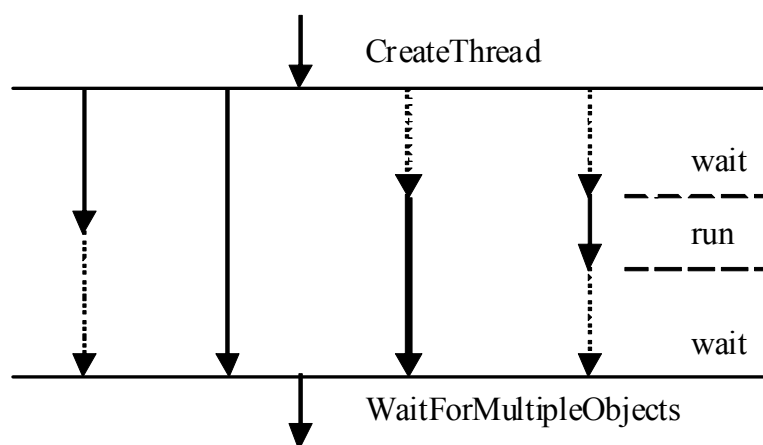
```
W:\VS1\s4\Release\s4.exe
Ns*NTh = 2000000000
S = 1990000000
Error = 10000000
```

WaitForMultipleObjects

Ожидание завершения работы всех потоков

`WaitForMultipleObjects(NTh,
Thread, TRUE, INFINITE)`

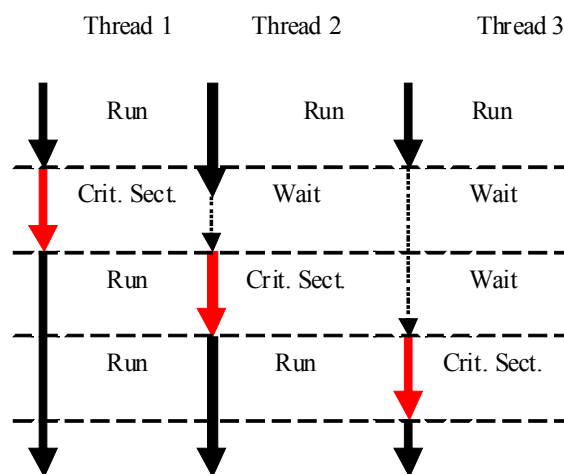
Барьерная синхронизация



Критическая секция

- Часть программного кода параллельной программы, выполняемая последовательно
- Когда один поток начинает выполнять критическую секцию, остальные потоки могут выполнять другие операции, но не могут параллельно войти в свою критическую секцию

Критическая секция



```

// Объявляем глобальную переменную CS
CRITICAL_SECTION CS;
void main()
{
    // Инициализируем критическую секцию
    InitializeCriticalSection(&CS);
    ...
    // Удаляем критическую секцию
    DeleteCriticalSection(&CS)
}
DWORD WINAPI ThreadProc( LPVOID lpParameter )
{
    // Входим в критическую секцию
    EnterCriticalSection(&CS);
    ...
    // Выходим из критической секции
    LeaveCriticalSection(&CriticalSection);
}

```

```

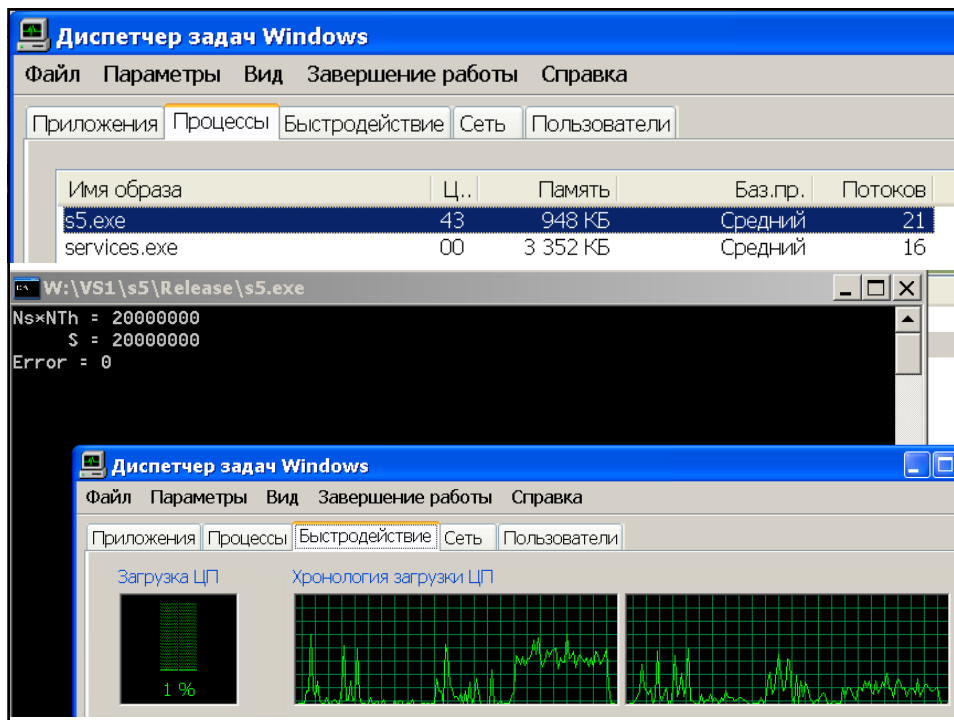
#include <windows.h>
#include <stdio.h>
#define NTh 20
#define Ns 1000000
int S = 0;
CRITICAL_SECTION CS;
void WINAPI ThreadProc()
{
    for (int i=0; i<Ns; i++)
    {
        EnterCriticalSection(&CS);
        S = S + 1;
        LeaveCriticalSection(&CS);
    }
}

```

```

void main()
{
HANDLE Thread [NTh];
InitializeCriticalSection(&CS);
for (int i=0; i < NTh; i++)
Thread [i] = CreateThread(0, 0,
(LPTHREAD_START_ROUTINE) ThreadProc, 0, 0, 0);
WaitForMultipleObjects(NTh,Thread,TRUE,INFINITE);
DeleteCriticalSection(&CS);
printf("Ns*NTh = %d\n      S = %d\nError =
%d\n",Ns*NTh,S,Ns*NTh-S);
getchar();
}

```



Параллельные процессы

MPI

MPI

- Message Passing Interface
- Компилятор поддерживает вызовы библиотечных функций
- Требуется установка дополнительной библиотеки MPI
 - MPICH – OpenSource библиотека MPI
 - <http://www.mcs.anl.gov/research/projects/mpich2/>
 - скачать MPICH2 Windows IA32 (binary)

Установка MPI

- SPMD process manager – привилегии администратора для установки
 - SPMD – служба запуска MPI процессов
- Passphrase (пароль по умолчанию)
- behappy
- Установка в папку (по умолчанию)
 - C:\Program Files\MPICH2\

Установка

- MPICH2 устанавливается на все машины, где будут запускаться процессы MPI
- При установке создаются каталоги
 - mpich2\bin
 - smpd.exe – менеджер MPI процессов
 - mpiexec.exe – программа для запуска заданий
 - mpich2\include
 - mpich2\lib
 - библиотеки для компиляции
- Библиотеки dll копируются в Windows\system32

Компиляция

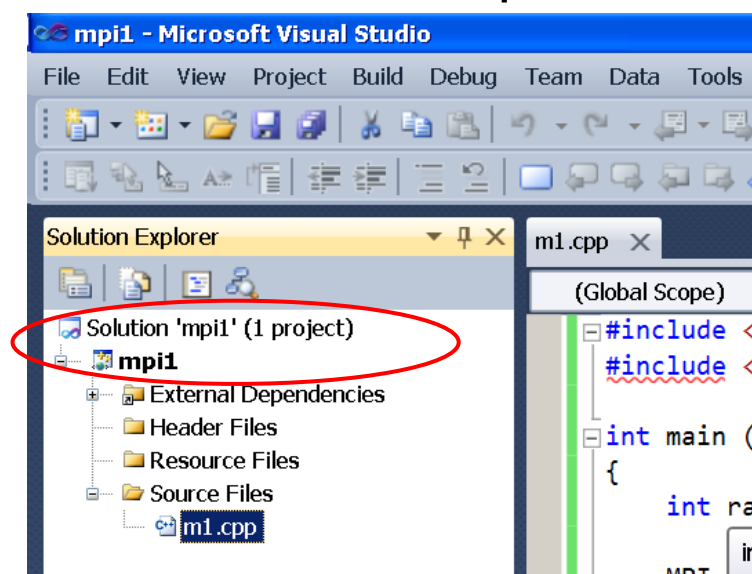
- 2) Add mpich2\include to the include path
- 3) Add mpich2\lib to the library path
- 4) For C applications add mpi.lib to your target link command.
- 6) Compile
- 7) Place your application and all the dlls it depends on in a shared location or copy them to all the nodes.
- 8) Run the application using mpiexec
- RUNNING MPI JOBS:
- mpiexec is a command line application used to launch MPI jobs. Bring up a command prompt to run it. Execute "mpiexec" to see the available options.
- The simplest mpiexec command is like this:
- mpiexec -n 3 myapp.exe

Минимальная установка

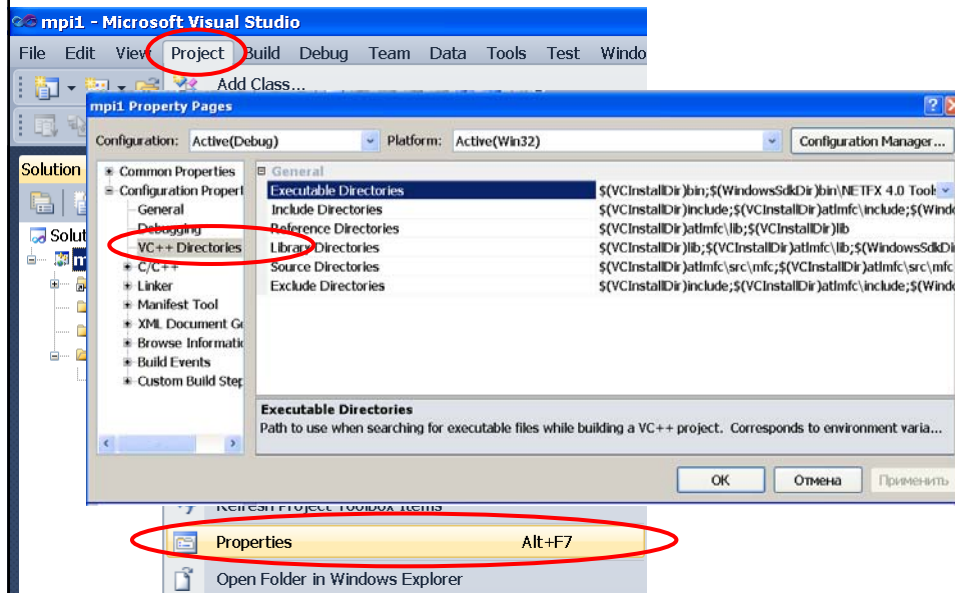
- Вычислительные узлы без компиляции, только для выполнения параллельной программы (worker nodes)
- Скопировать файл smpd.exe
- Выполнить команду
 - smpd.exe -install
- Скопировать dll библиотеки
 - в каталог windows\system32
 - в каталог, из которого запускается MPI программа
- For example, if you have a directory called \\myserver\mysharedfolder and you have myapp.exe and *mpich2*.dll in that directory then you can execute this command: "mpiexec -n 4 \\myserver\mysharedfolder\myapp.exe"
- Note: There are several mpich2 dlls and depending on your build target (Fortran, C, Debug or Release) you will need the corresponding dlls in the application directory.

- To specify a per-project directory list
- Project menu
- click Properties
- Configuration Properties
- VC++ Directories
- To edit one of the directory lists,
 - click its name
 - click the arrow that appears
 - click Edit
- You can add or remove values, and you can reorder any values that have been added. You can also select or clear Inherit from parent or project defaults.

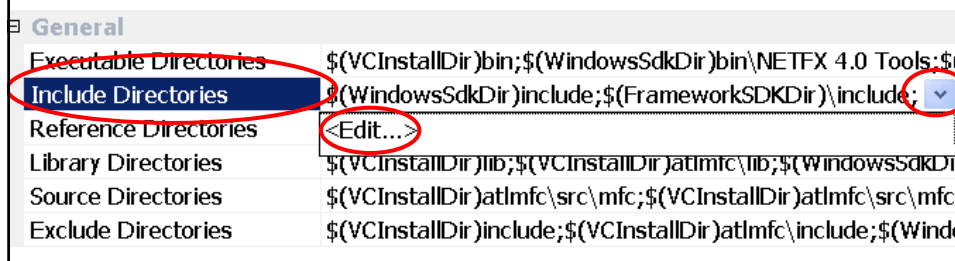
Создаем проект



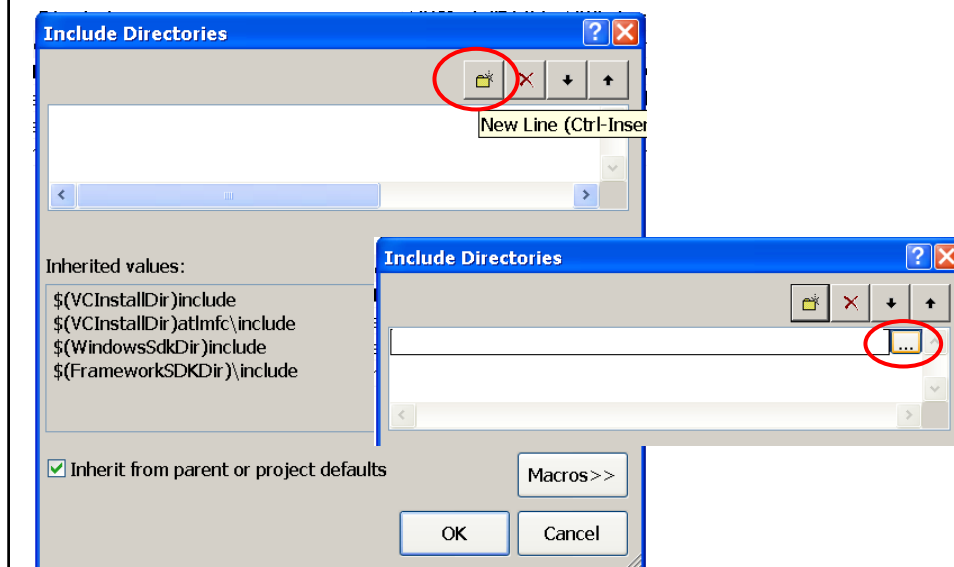
Project ► Properties ► VC++ Directories



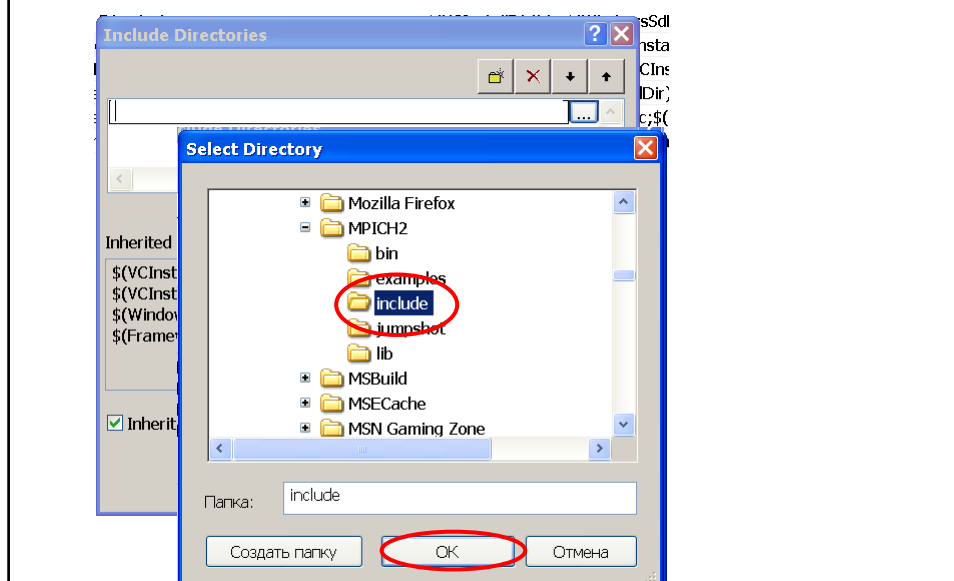
Include Directories ► Список ► Edit



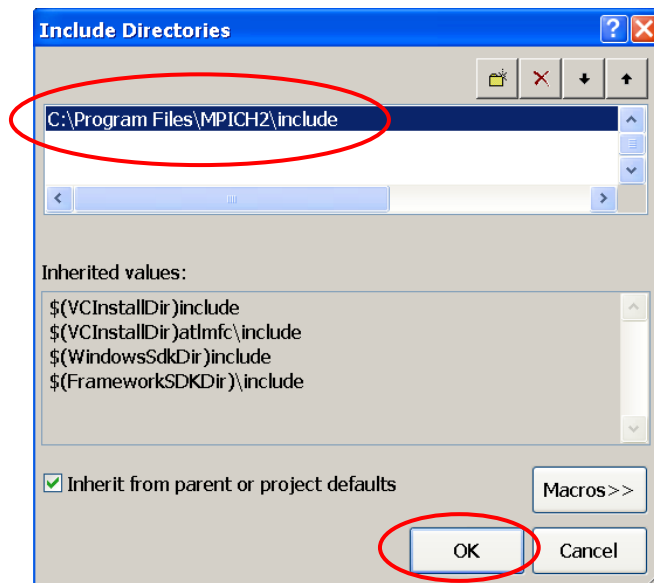
Include Directories ► New Line ► ...



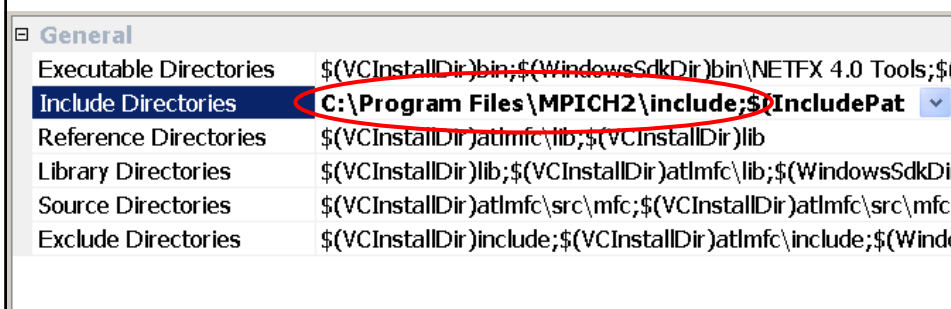
Select Directory ► C:\Program Files\MPICH2\include ► OK



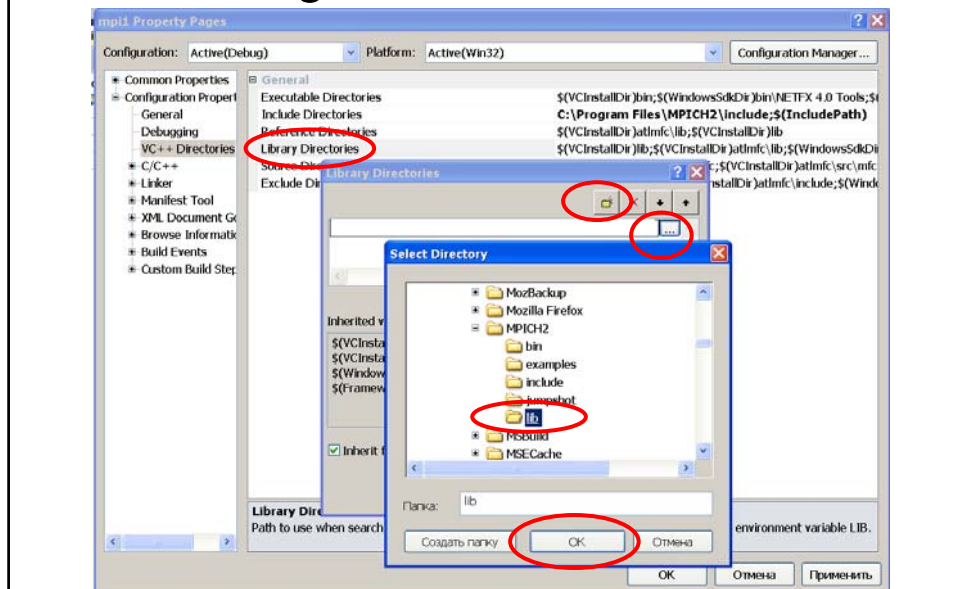
Include Directories ► OK



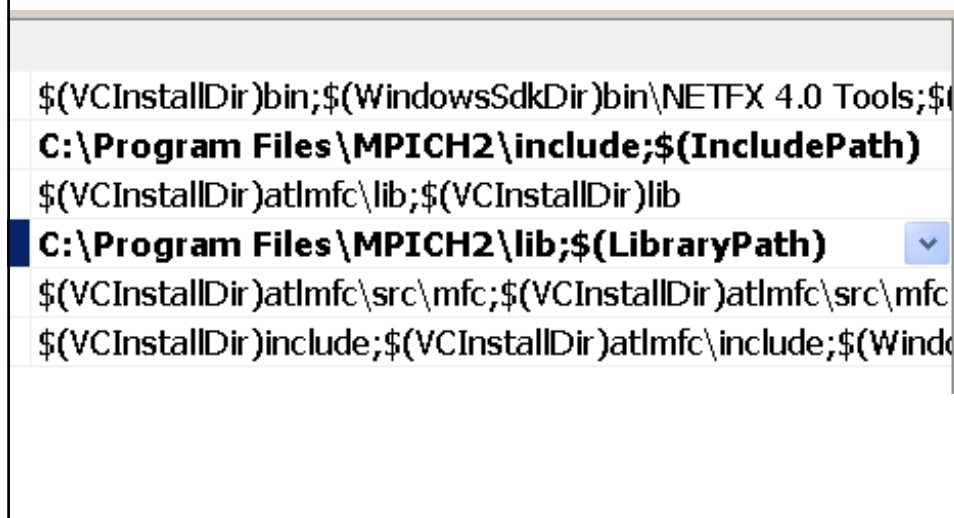
Каталог показан в списке



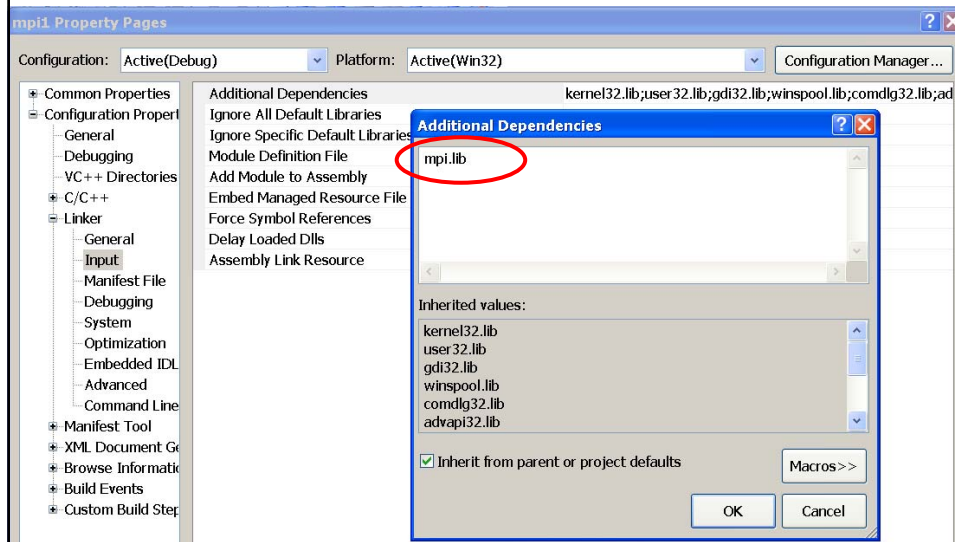
Library Directories ► C:\Program Files\MPICH2\lib



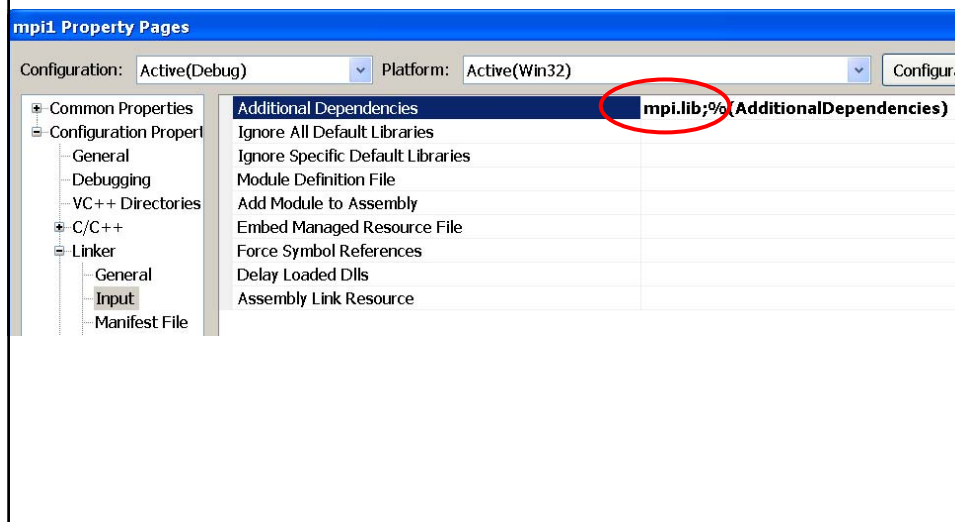
Проверяем каталоги MPICH2\include MPICH2\lib



Linker ► Input ► Additional Dependencies ► mpi.lib



mpi.lib



Debug / Release

- Повторить установку для конфигурации Release
 - Include Dir - MPICH2\include
 - Lib Dir - MPICH2\lib
 - Linker - mpi.lib

- Для работы smrd на всех компьютерах должны быть одинаковые логин и пароль пользователей
- Пароль не должен быть пустым

Запуск MPI приложения

- Запускаем CMD
- Переходим в рабочий каталог
- `mpiexec -n 3 myapp.exe`
- Для выполнения команды скопировать `mpiexec.exe` в рабочий каталог либо установить путь Path

Hello, World!

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

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

Запуск MPI

- Запуск параллельного приложения MPI
 - Одна и та же программа запускается несколько раз
- Запускается заданное число идентичных процессов
 - на локальной машине
 - на любой машине в сети
 - на машинах с заданными адресами

Коммуникатор

- Название группы процессов, которые могут обмениваться сообщениями
- По умолчанию, все процессы входят в коммуникатор
- **MPI_COMM_WORLD**

MPI_Init

```
MPI_Init (&argc, &argv);
```

- Инициализация
- Запуск системы обмена сообщениями MPI

MPI_Finalize

```
MPI_Finalize();
```

- Завершение
- Окончание работы системы обмена сообщениями MPI

MPI_Comm_size

- **MPI_Comm_size**
(MPI_COMM_WORLD, &size);
- Количество процессов в коммутаторе

MPI_Comm_rank

MPI_Comm_rank
(MPI_COMM_WORLD, &rank)

Номер процесса в коммутаторе

Можно поручить нулевому процессу собирать результаты вычислений и выводить на экран

Вывод на экран

- **printf**
- вызывается из любого потока
- производится на экран машины, с которой запущено задание

MPI_Reduce

```
MPI_Reduce(&PartSum, &MyPi, 1,  
MPI_DOUBLE, MPI_SUM, 0,  
MPI_COMM_WORLD);
```

- Операция редукции
- Выполняется операция суммирования MPI_SUM над локальными переменными PartSum всех параллельных процессов
- Результат записывается в переменную MyPi процесса номер 0 в коммуникаторе MPI_COMM_WORLD

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\mpiuser>w:

W:\>cd 1

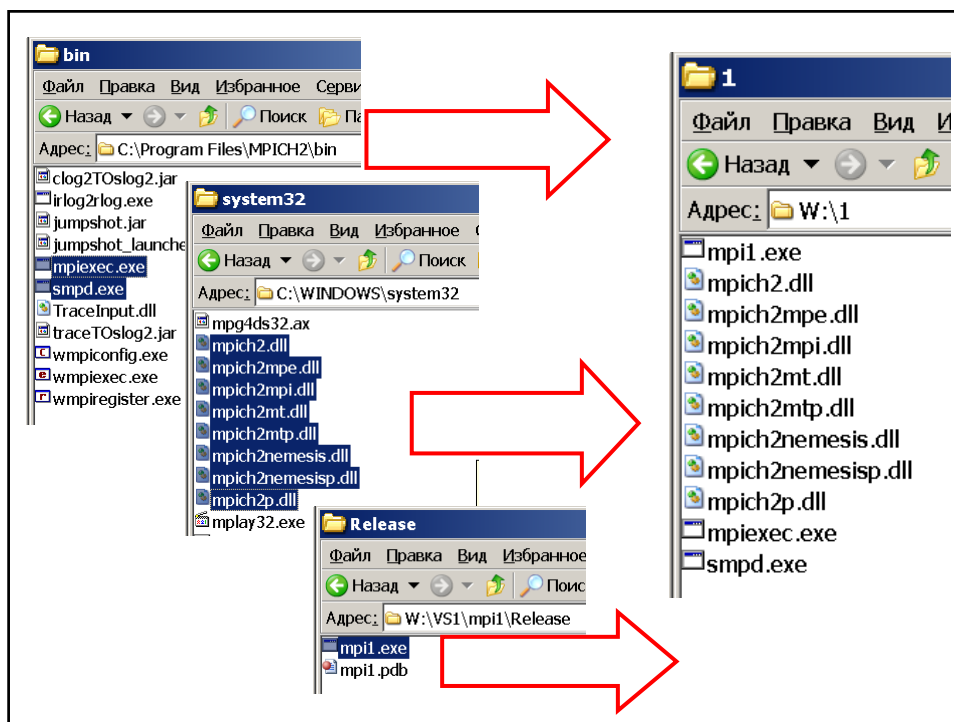
W:\1>dir
Том в устройстве W имеет метку Disk-1-W 400 GB
Серийный номер тома: 2032-3546

Содержимое папки W:\1

22.01.2012  15:09    <DIR>          .
22.01.2012  15:09    <DIR>          ..
22.01.2012  15:00                7a168 mpi1.exe
01.09.2011  15:25                491a520 mpiexec.exe
                2 файлов                498a688 байт
                2 папок                4a947a578a880 байт свободно

W:\1>mpiexec -n 3 mpi1.exe
User credentials needed to launch processes:
account (domain\user) [WALL\mpiuser]:
password:
Hello world from process 1 of 3
Hello world from process 2 of 3
Hello world from process 0 of 3

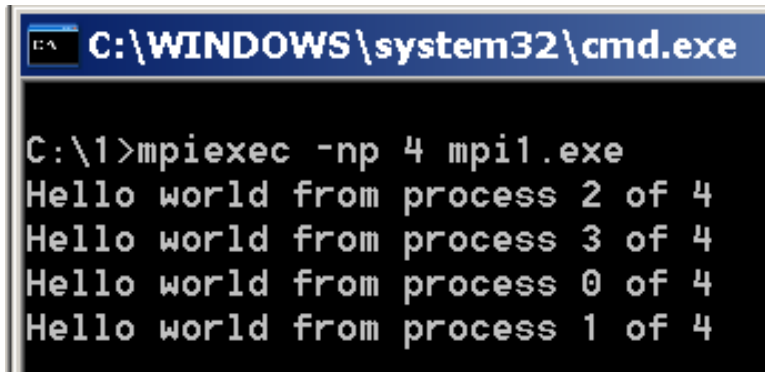
W:\1>_
```



mpiexec.exe

Запуск 4 процессов на локальной машине

mpiexec -np 4 mpi1.exe



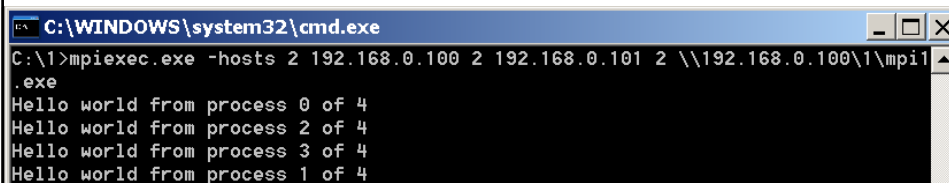
```
C:\WINDOWS\system32\cmd.exe

C:\>mpiexec -np 4 mpi1.exe
Hello world from process 2 of 4
Hello world from process 3 of 4
Hello world from process 0 of 4
Hello world from process 1 of 4
```

mpiexec.exe

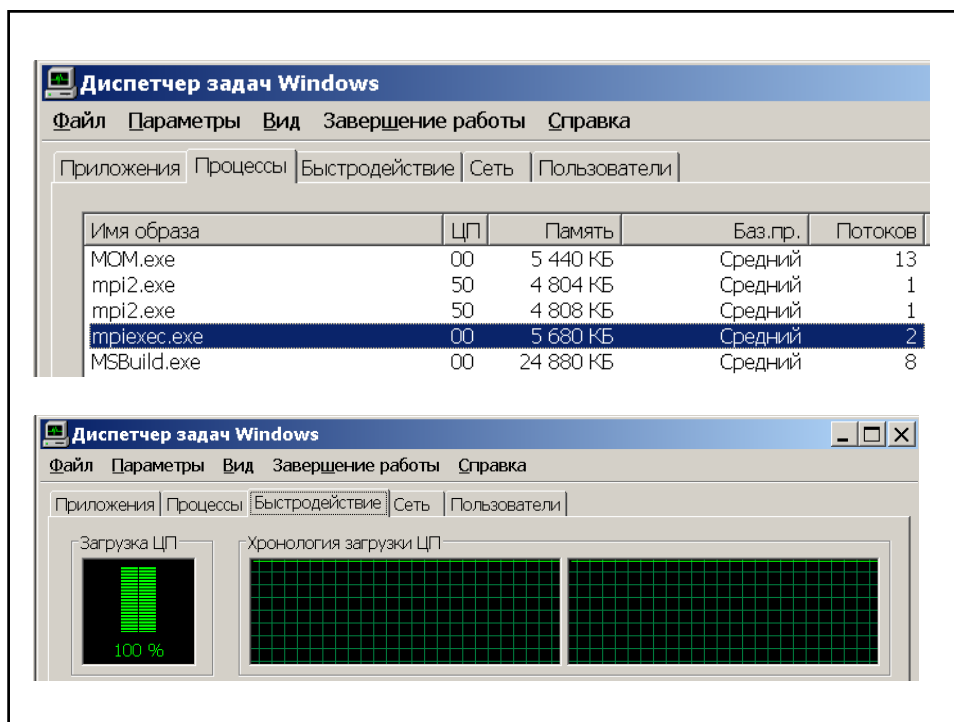
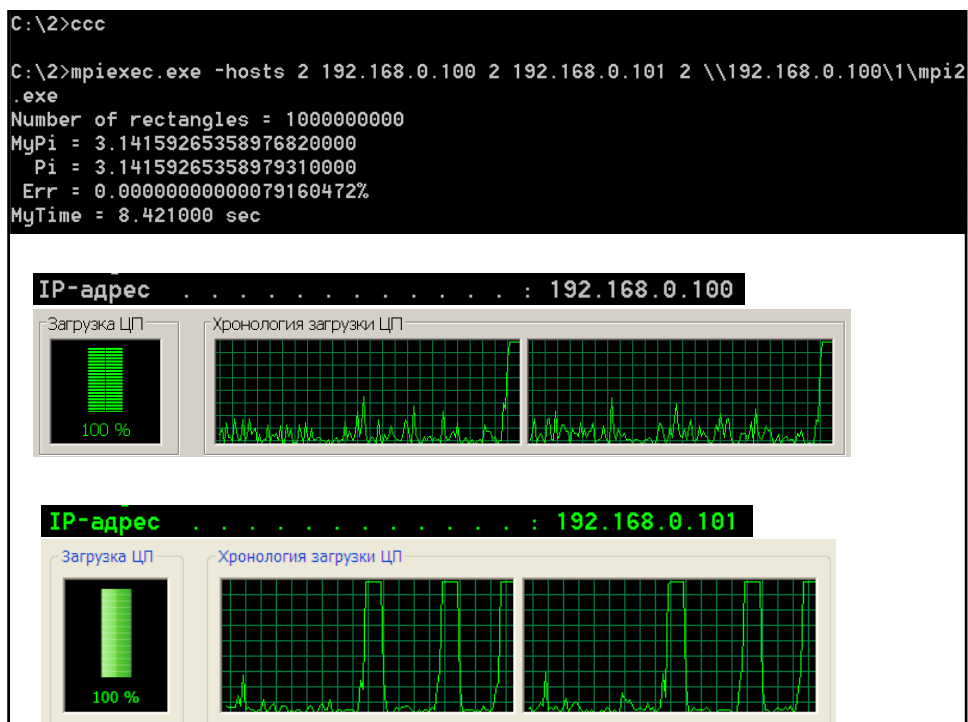
Запуск 2 процессов на 2 машинах с сетевого диска

**mpiexec.exe -hosts 2 192.168.0.100 2 192.168.0.101 2
\\192.168.0.100\1\mpi1.exe**



```
C:\WINDOWS\system32\cmd.exe

C:\>mpiexec.exe -hosts 2 192.168.0.100 2 192.168.0.101 2 \\192.168.0.100\1\mpi1.exe
Hello world from process 0 of 4
Hello world from process 2 of 4
Hello world from process 3 of 4
Hello world from process 1 of 4
```



Оценка числа Пи

- Интегрирование методом прямоугольников

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

```
// Вычисление числа Пи: интеграл методом
// прямоугольников
#include <stdio.h>
#include "mpi.h"
#include <time.h>
#include <math.h>
#define NUMRECT 100000000
int main () {
    int ProcNum, ProcRank, i;
    double x, MyPi = 0;
    double PartSum = 0;
    double TotalTime;
    clock_t StartClock, EndClock;
    StartClock = clock ();
    MPI_Init( NULL, NULL );
    MPI_Comm_size (MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank (MPI_COMM_WORLD, &ProcRank);
    double h = 1.0 / NUMRECT;
    for (i = ProcRank; i < NUMRECT; i+= ProcNum) {
        x = ( i + 0.5 ) * h;
        PartSum += 4.0 / ( 1 + x*x ) ; }
    EndClock = clock ();
    TotalTime = (EndClock - StartClock) / CLOCKS_PER_SEC;
    printf("ProcRank: %d, MyPi: %f, TotalTime: %f\n", ProcRank, MyPi, TotalTime);
}
```

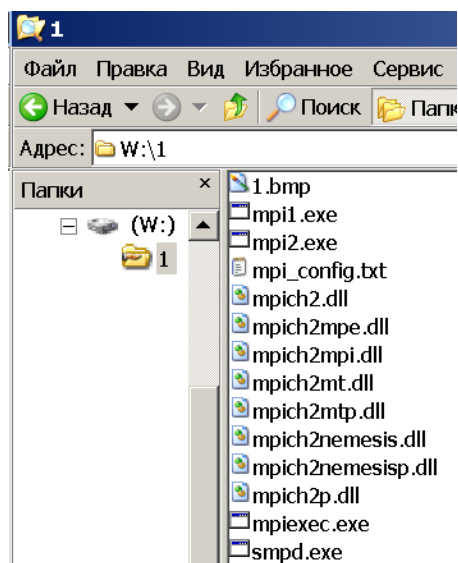
```

MPI_Reduce(&PartSum, &MyPi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
MPI_Finalize ();
if (ProcRank == 0)
{
MyPi *= h;
EndClock = clock ();
printf("Number of rectangles = %i\n", NUMRECT);
printf("MyPi = %3.20f\n", MyPi);
double pi = 3.1415926535897932384626433832795;
printf("  Pi = %3.20f\n", pi);
printf(" Err = %3.20f\n", abs(MyPi-pi));
TotalTime = (double) ( EndClock - StartClock ) /
CLOCKS_PER_SEC;
printf("MyTime = %f sec\n", TotalTime);
}
}

```

Компиляция – Запуск

- Файлы скопировать в разделяемую сетевую папку



```

n=4;
h=1/n;
x=(0:n-1)*h+h/2;
f=4.0./(1+x.^2);
MyPi=sum(f)*h;
printf("MyPi=%3.20f\n",MyPi)
pi = 3.1415926535897932384626433832795;
printf("  Pi=%3.20f\n",pi)
printf(" Err=%3.20f\n",abs(MyPi-pi))

```

```

MyPi=3.14680051839394270000
  Pi=3.14159265358979310000
 Err=0.00520786480414958670

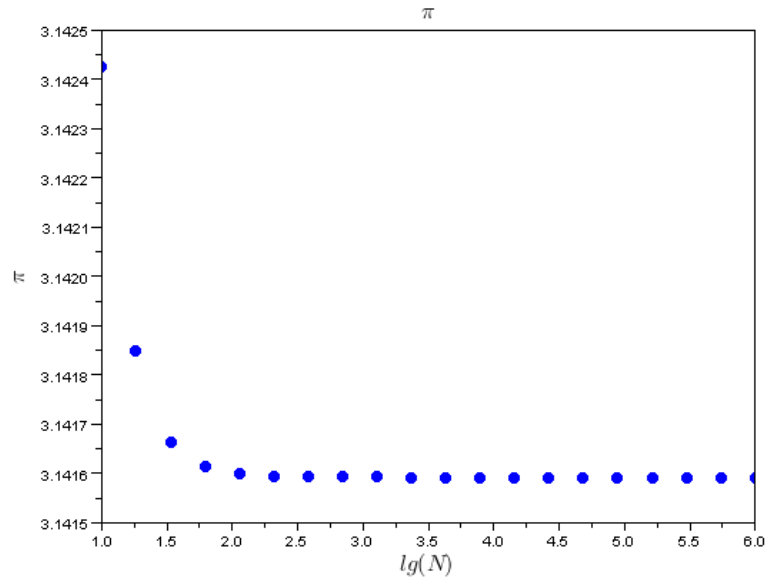
```

```

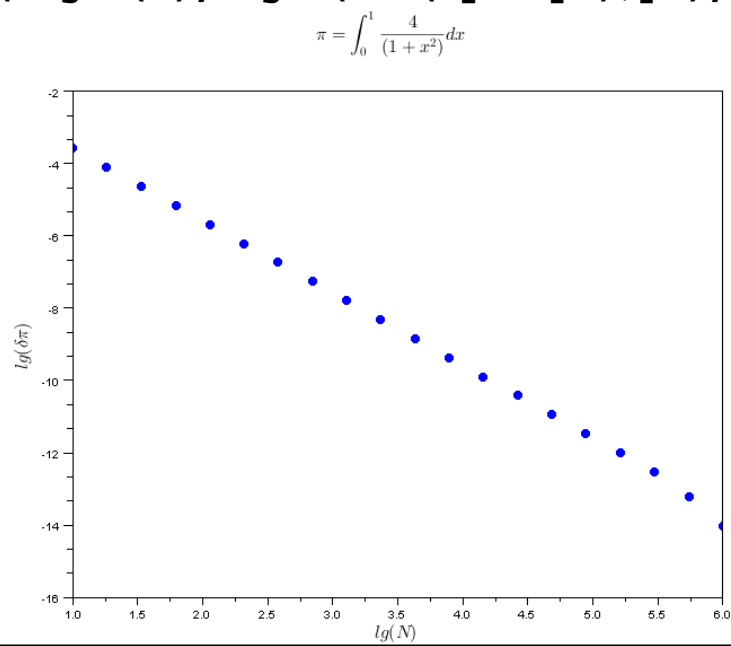
n=round(logspace(1,6,20));
for i=1:length(n)
h=1/n(i);
x=(0:n(i)-1)*h+h/2;
f=4.0./(1+x.^2);
MyPi(i)=sum(f)*h;
end
plot(n,MyPi, '.')
plot(log10(n),MyPi, '.')
xlabel('$\pi$', '$lg(N)$', '$\pi$')
plot(log10(n),log10(abs(MyPi-
  pi)/pi), '.')
xlabel('$\pi=\int_0^1 \frac{4}{1+x^2}
  dx$', '$lg(N)$', '$lg(\delta\pi)$')

```

plot(log10(n), MyPi, '.')



plot(log10(n), log10(abs(MyPi-pi)/pi), '.')



Параллельные пакеты прикладных программ

Параллельные приложения

- Готовые параллельные версии прикладных пакетов программ
- Для конкретной предметной области
- Вычисления без программирования
 - Коммерческие пакеты
 - OpenSource