программирование

ВВЕДЕНИЕ В ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ.

АСИНХРОННЫЕ ОПЕРАЦИИ КАК ЧАСТЬ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ. ПОТОКИ, ПРИМИТИВЫ СИНХРОНИЗАЦИИ В C++. ПРИВЯЗКА К ЯДРАМ.

Введение в параллельное программирование

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

-- E. Dijkstra, 1972 Turing Award Lecture

Знаменитый закон Мура

I Закон Мура (1965): каждые 2 года количество транзисторов в интегральной микросхеме удваивается.

Следствие Хауса: производительность центрального процессора компьютера удваивается каждые 18 месяцев.

Мур, 2007: закономерности перестанут работать вследствие атомарной природы вещества и ограничения скорости света.

Первый кризис ПО

60-70 годы 20 века

Проблема - язык программирования ассемблер.

Компьютеры были готовы обрабатывать гораздо более сложные программы, чем могли писать люди.

Для устранения была необходима абстракция над машинным кодом и переносимость программ между вычислителями.

Первый кризис ПО

Решение

Появление языков высокого уровня С и Фортран. Появление общих свойств у разных вычислителей. Модель ФонНеймана. Единая память. Единственный поток управления.

Второй кризис ПО

80-90 годы 20 века

Проблема: невозможность разработки сложных программ, состоящих из миллионов строк кода, которые разрабатываются сотнями людей. **Компьютеры все еще готовы обрабатывать гораздо более сложные программы, чем могли писать люди.**

Требовалась разработка стандартов по проектированию модульных, гибких и обслуживаемых программ. Скорость выполнения программ не была узким местом, вспоминаем закон Мура.

Второй кризис ПО

Решение

Появление ООП и развитие языков высокого уровня C#, Java, C++. Появление библиотек и инструментов для написания, отладки и тестирования программ. Разработка различных методологий построения программного обеспечения. Появление методологий автоматического тестирования, ревью кода, XP программирования.

Назревает третий кризис ПО

Особенности

- Четкая граница между программой и железом.
- Программисты больше ничего не должны знать о процессорах и их регистрах.
- Высокоуровневые языки полностью абстрагируют программу от платформы исполнения.
- Закон Мура все еще позволяет программистам получать хорошую скорость выполнения за счет мощных процессоров.
- Программы одинаково хорошо работают на разных процессорах.

Такой подход развязывает руки программистам..

Назревает третий кризис ПО

Проблема

Высокий уровень абстракции не дает программистам достаточной мотивации писать оптимальные программы, которые работают максимально быстро. Вместо этого скорость работы достигается путем появления новых процессоров.

Шутка

- Скажи мне, Microsofe Office, почему ресурс моего компьютера позволяет в онлайне управлять орбитальной группировкой из тысячи боевых спутников, отслеживающих пролет тысяч баллистических ракет врага, но не в состоянии без лагов прогружать страницу в Excel когда там более двух вкладок?)))

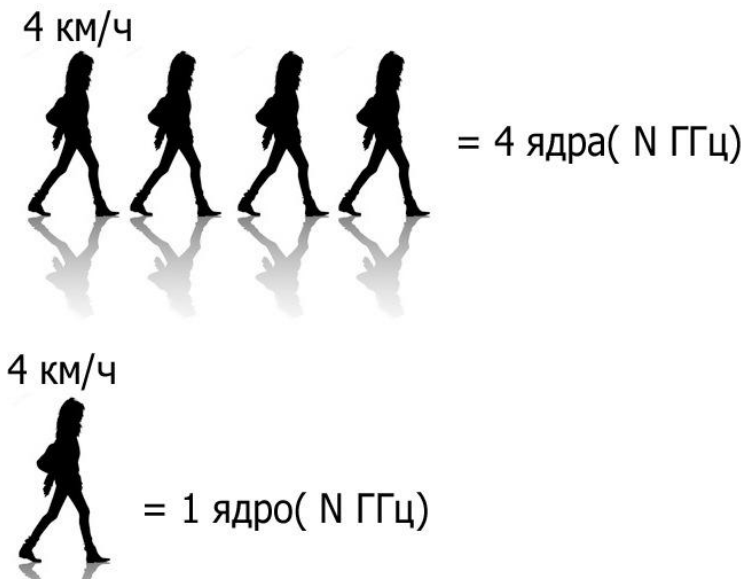
- Потому что ваш компьютер не удовлетворяет минимальным системным требованиям для офиса, так как все ресурсы уходят на слежение за вами. Чем я еще могу вам помочь?

http://pikabu.ru/story/kakim_glazom_ya_seychas_morgnul_4423571

Выводы из действия закона Мура

Раньше – производительность процессора росла сама по себе, медленная программа с ходом времени работала все быстрее.

- **Теперь** – последовательная производительность процессора практически не растет, программист должен использовать *параллельные вычисления*, чтобы ускорить программу



Алгоритм

Алгоритм – упорядоченная последовательность действий, приводящая к определенному результату.

Задача: $z = u \cdot x + v \cdot y$

Программа

Программа – способ выполнения алгоритма на определенном вычислителе. По сути набор инструкций, приводящий к определенному результату.

1.load r1, u

2.load r2, x

3.mul r3, r1,r2

4.load r1, v

5.load r2, y

6.mul r4, r1,r2

7.add r1, r3,r4

8.store z, r1

Программа

1.load r1, u

2.load r2, x

3.mul r3, r1,r2

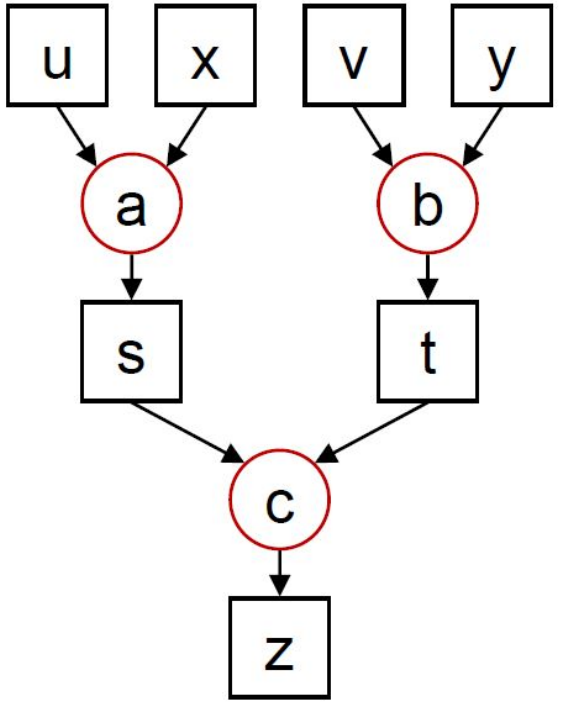
4.load r1, v

5.load r2, y

6.mul r4, r1,r2

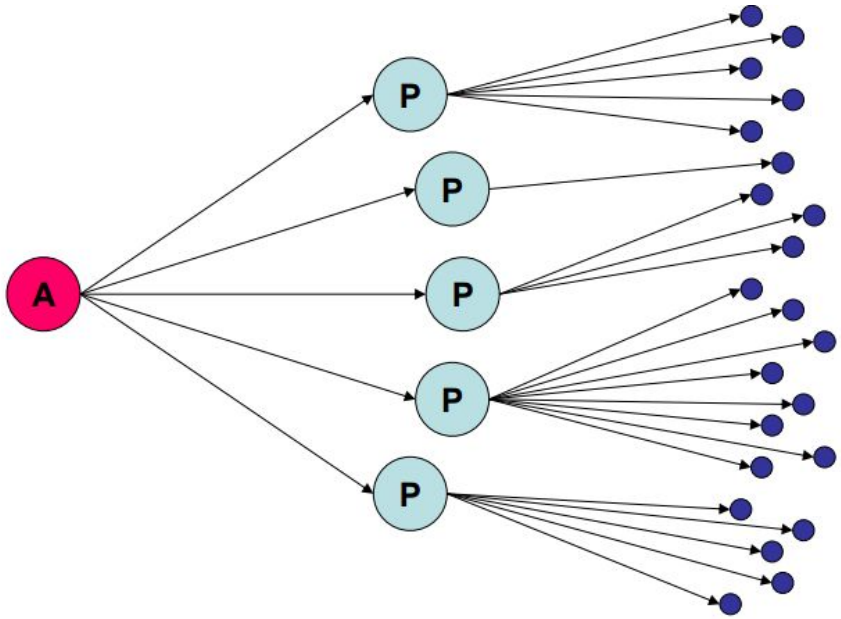
7.add r1, r3,r4

8.store z, r1



Реализация

Реализация – способ реализации алгоритма при помощи программы. Это множество путей выполнения алгоритма, часто неоптимальных.



Вы «царь и бог»

Компоненты вычислителя

• исполнительные устройства, ядра, процессоры, вычислительные узлы, кластера, …

• блоки памяти - Оперативная память, кэш-память, дисковая память …

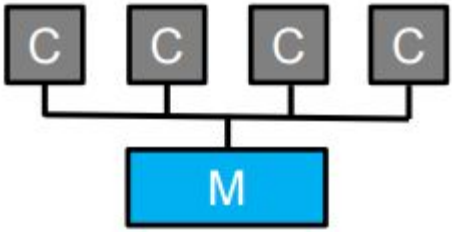
• связи - Процессор-память, межпроцессорные, межузловые, межкластерные, …

Системы с общей памятью

Многоядерные процессоры

Многопроцессорные узлы

…



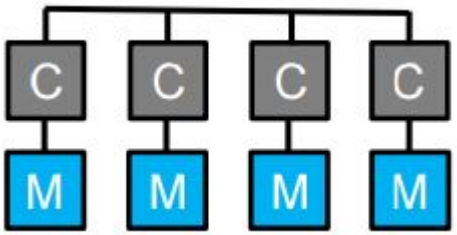
Системы с распределенной памятью

Сети рабочих станций

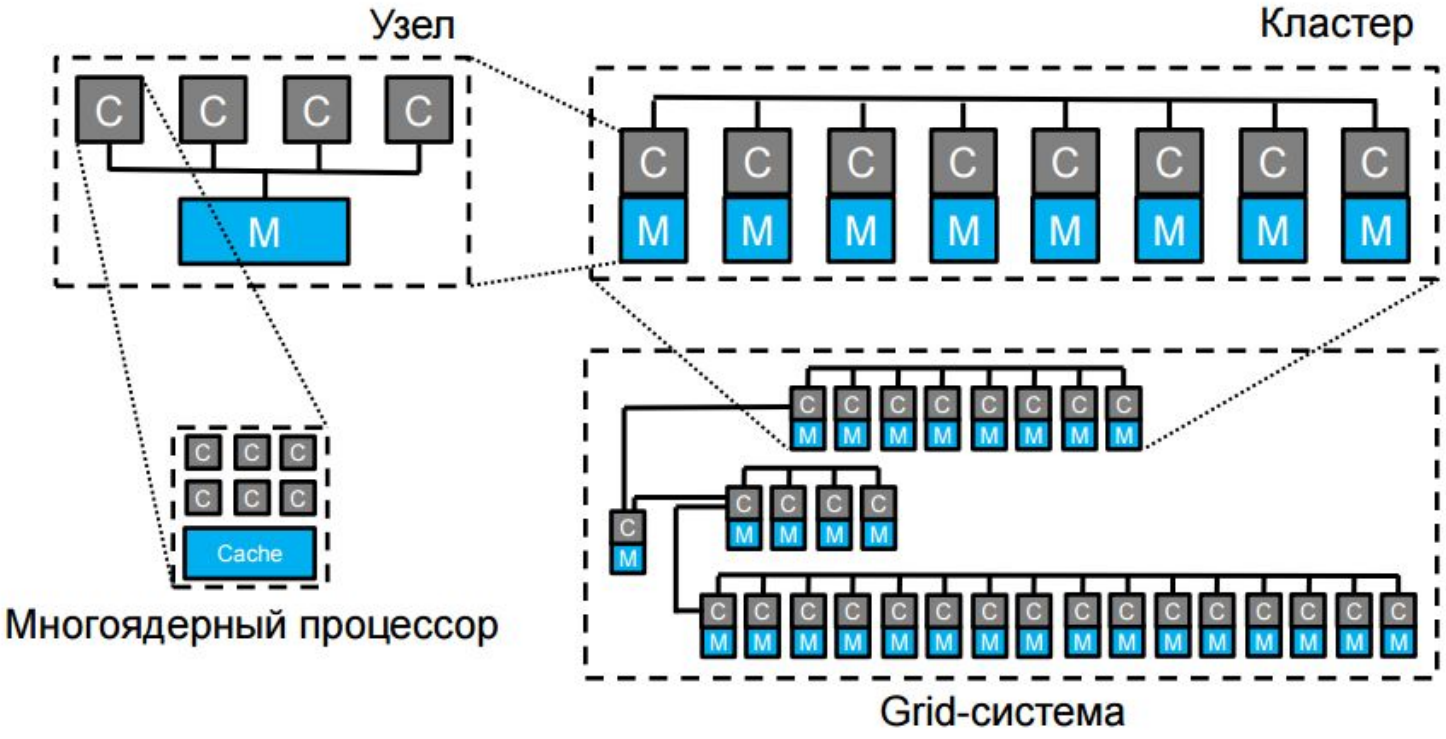
Кластера

Grid

…



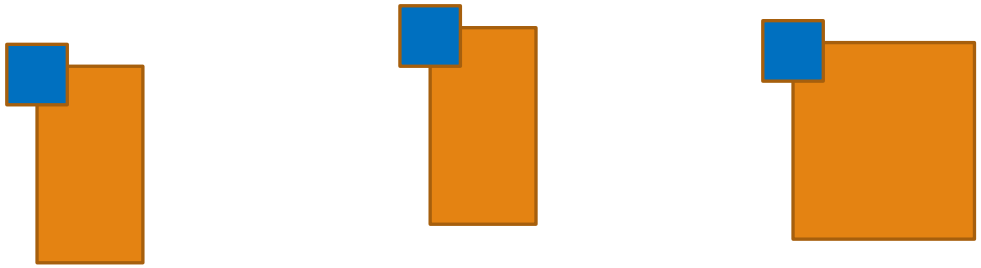
Общая иерархия



Меры качества параллельных программ

Производительность системы равняется производительности ее самого слабого звена..

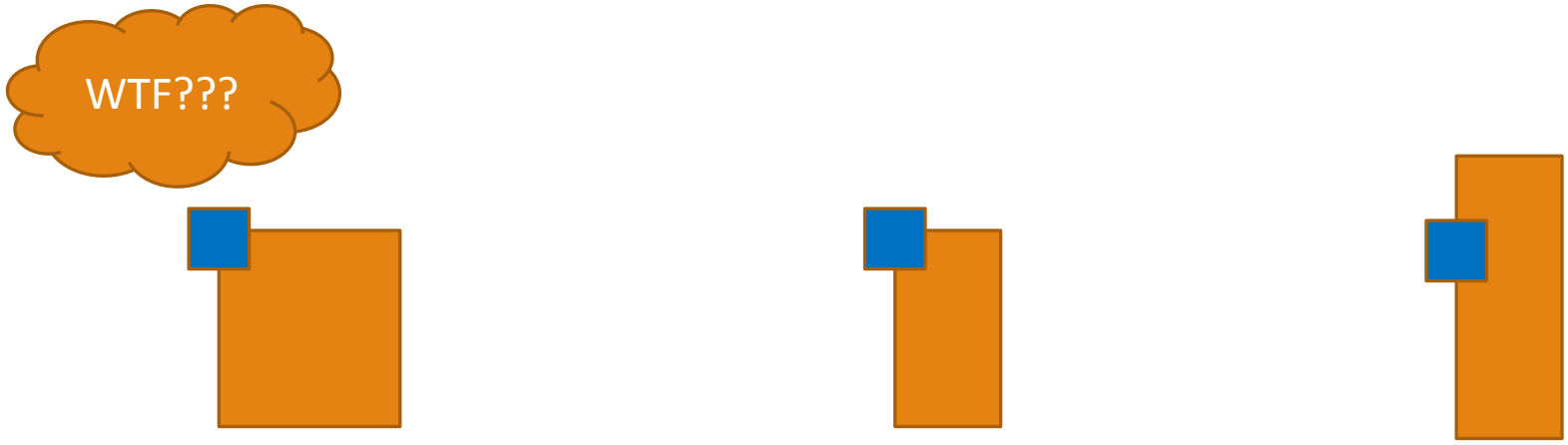
Поход бойскаутов



A B

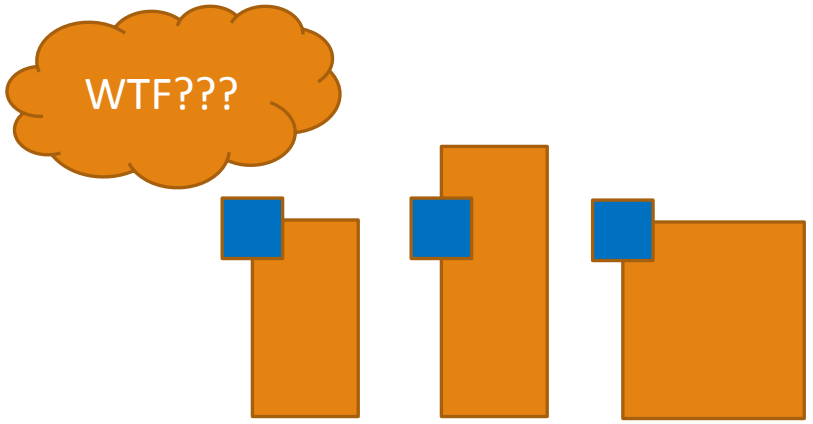
10 км

Поход бойскаутов



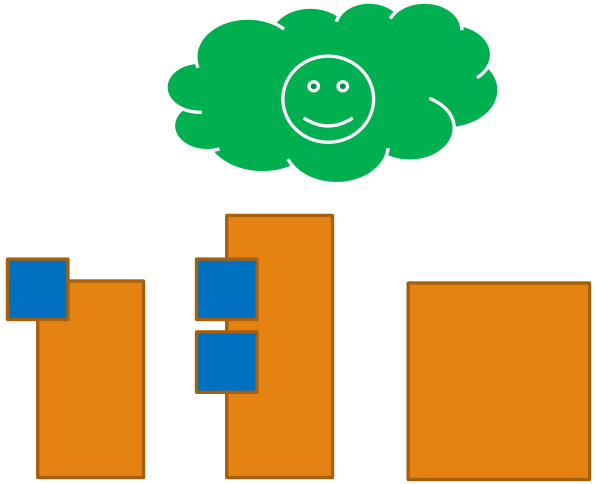
2 км/час

Поход бойскаутов



2 км/час

Поход бойскаутов



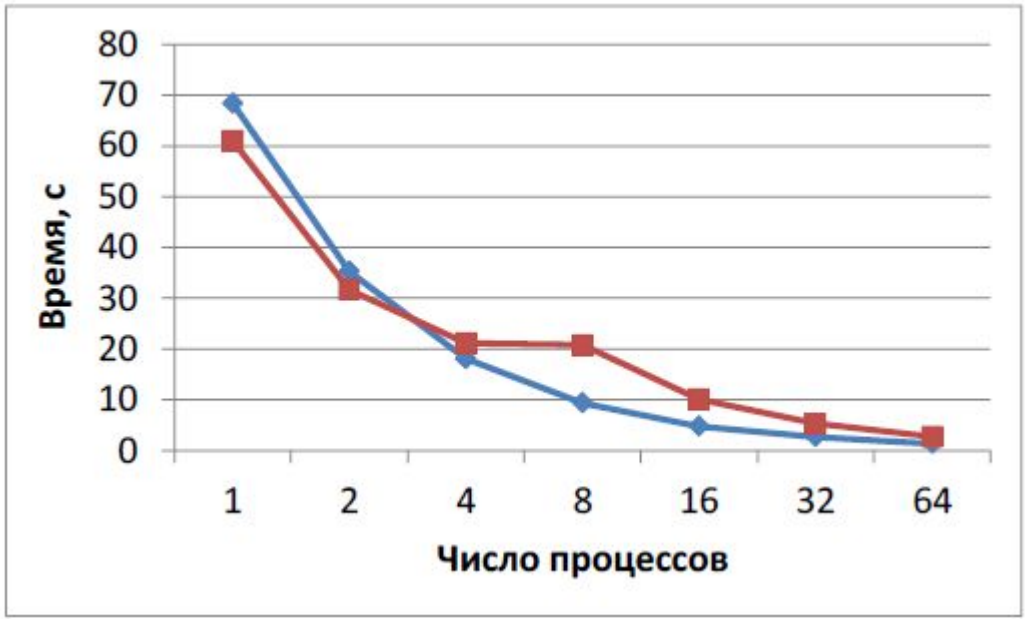
5 км/час

Меры качества параллельных программ

- Время работы
  - Сколько времени программа работает на N ядрах?
- Ускорение
  - Во сколько раз программа стала быстрее работать на N ядрах по сравнению с одним ядром / последовательной программой?
- Эффективность распараллеливания
  - Какой процент времени работы программы идёт на полезную работу?
- Масштабируемость
  - Как быстро эффективность падает с ростом числа ядер?
- …

Пример

- Хорошо масштабируется
- Плохо масштабируется



Меры качества параллельных программ

Ускорение

Ускорение параллельной программы при использовании N исполнительных устройств относительно…

- последовательной:

$$S^{seq}_N = T_{seq} / T_N$$

- параллельнои с использованием одного исполнительного устройства:

$$S^1_N = T_1 / T_N$$

где:

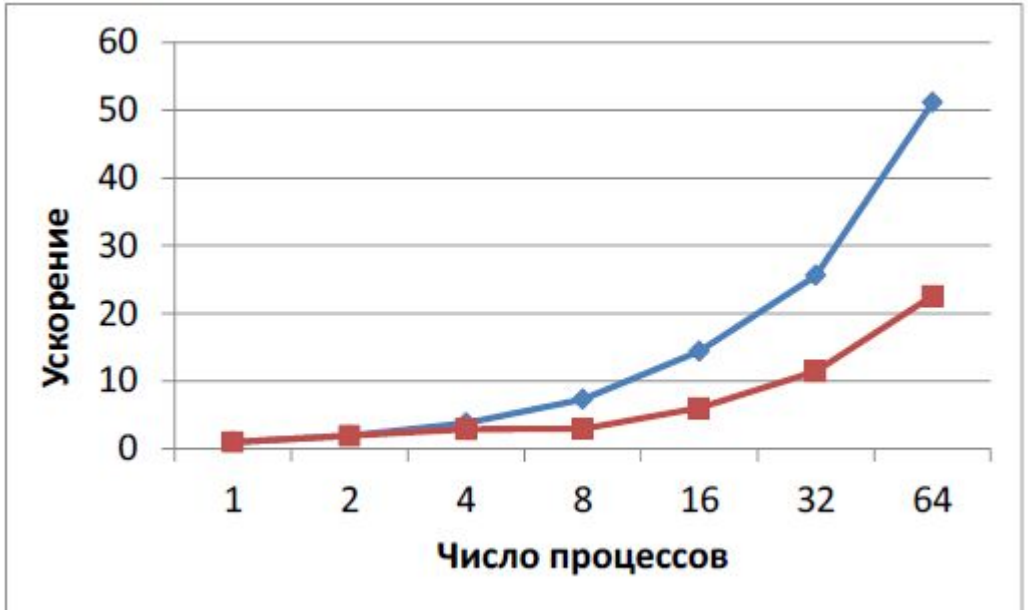
T_{seq} – время работы последовательной программы,

T_1 – время работы параллельной программы при использовании одного исполнительного устройства,

T_N – время работы параллельной программы при использовании N исполнительных устройств.

Пример

- Хорошо масштабируется
- Плохо масштабируется



Меры качества параллельных программ

Эффективность распараллеливания

Эффективность использования N исполнительных устройств относительно…

- последовательной программы:

$$E^{seq}_N = S^{seq}_N / N$$

- параллельной программы с использованием одного исполнительного устройства:

$$E^1_N = S^1_N / N$$

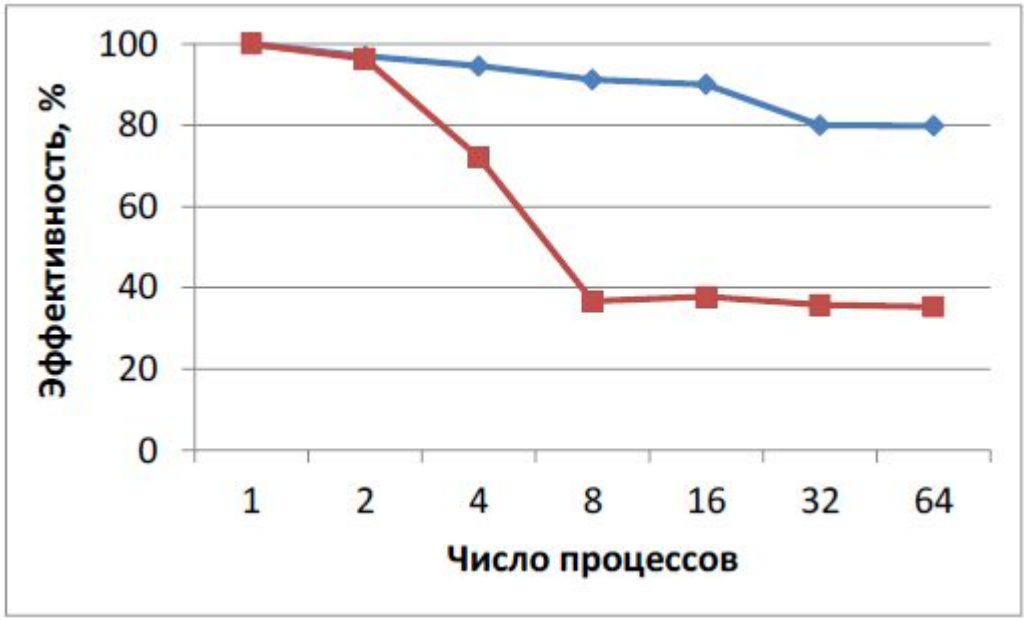
где:

S_{seqN} – ускорение параллельной программы при использовании N исполнительных устройств относительно последовательной,

$S1_N$ – ускорение параллельной программы при использовании N исполнительных устройств относительно параллельной при использовании одного исполнительного устройства.

Пример

- Хорошо масштабируется
- Плохо масштабируется



Предел ускорения: закон Амдала

α – доля последовательных вычислений [0;1]

Предел ускорения: закон Амдала

α – доля последовательных вычислений [0;1]

1 - α – доля параллельных вычислений

Предел ускорения: закон Амдала

α – доля последовательных вычислений [0;1]

1 - α – доля параллельных вычислений

1 – время выполнения последовательной программы

Предел ускорения: закон Амдала

α – доля последовательных вычислений [0;1]

1 - α – доля параллельных вычислений

1 – время выполнения последовательной программы

α+(1- α)/p - время выполнения параллельной программы на p вычислителях

Предел ускорения: закон Амдала

Ускорение, которое может быть получено на вычислительной системе из **p** процессоров, по сравнению с однопроцессорным решением не будет превышать величины:

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

Предел ускорения: закон Амдала

Ускорение, которое может быть получено на вычислительной системе из **p** процессоров, по сравнению с однопроцессорным решением не будет превышать величины:

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

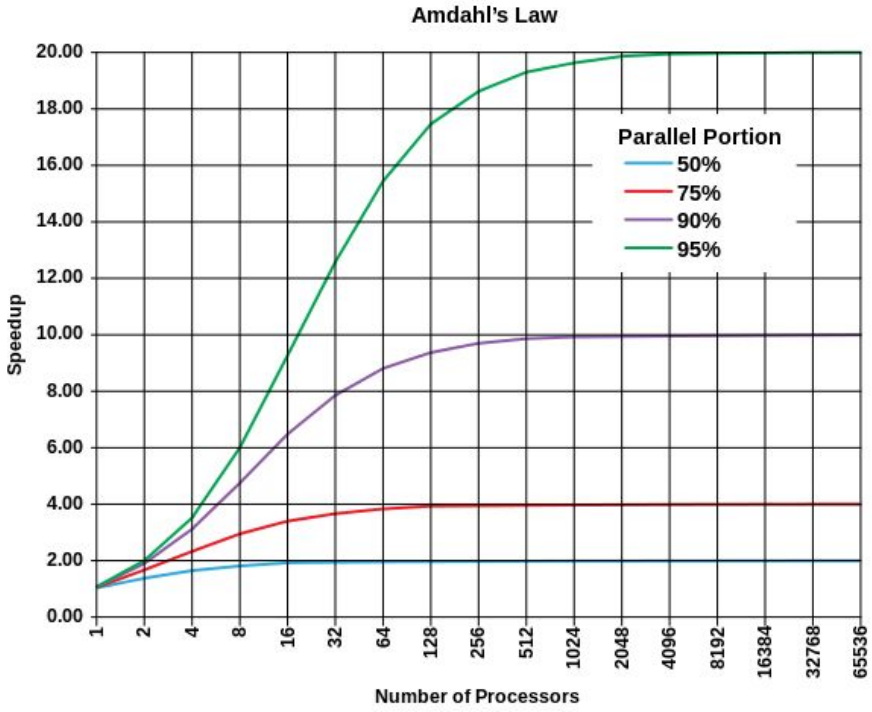
При каком значении α будет максимальное ускорение?

Предел ускорения: закон Амдала

α /p	10	100	1000
0	10	100	1000
10%	5.263	9.174	9.910
25%	3.077	3.883	3.988
40%	2.174	2.463	2.49

Если доля последовательных вычислений в алгоритме равна 25 %, то увеличение числа процессоров до 10 дает ускорение в 3,077 раза, а увеличение числа процессоров до **1000** даст ускорение в **3,988** раза.

Предел ускорения: закон Амдала



Способы реализация параллельных вычислений

Процесс (process) – работающий в текущий момент экземпляр программы

Способы реализация параллельных вычислений

Процесс (process) – работающий в текущий момент экземпляр программы

Многозадачность (multitasking) – несколько процессов выполняются условно одновременно за счет операционной системы

- Вытесняющая многозадачность
- Невытесняющая многозадачность

Способы реализация параллельных вычислений

Процесс (process) – работающий в текущий момент экземпляр программы

Многозадачность (multitasking) – несколько процессов выполняются условно одновременно за счет операционной системы

- Вытесняющая многозадачность
- Невытесняющая многозадачность

Поток – последовательно выполняемая ветвь кода, для которой выделен отдельный стек и обеспечивается независимость использования регистров процессора.

Способы реализация параллельных вычислений

Процесс (process) – работающий в текущий момент экземпляр программы

Многозадачность (multitasking) – несколько процессов выполняются условно одновременно за счет операционной системы

- Вытесняющая многозадачность
- Невытесняющая многозадачность

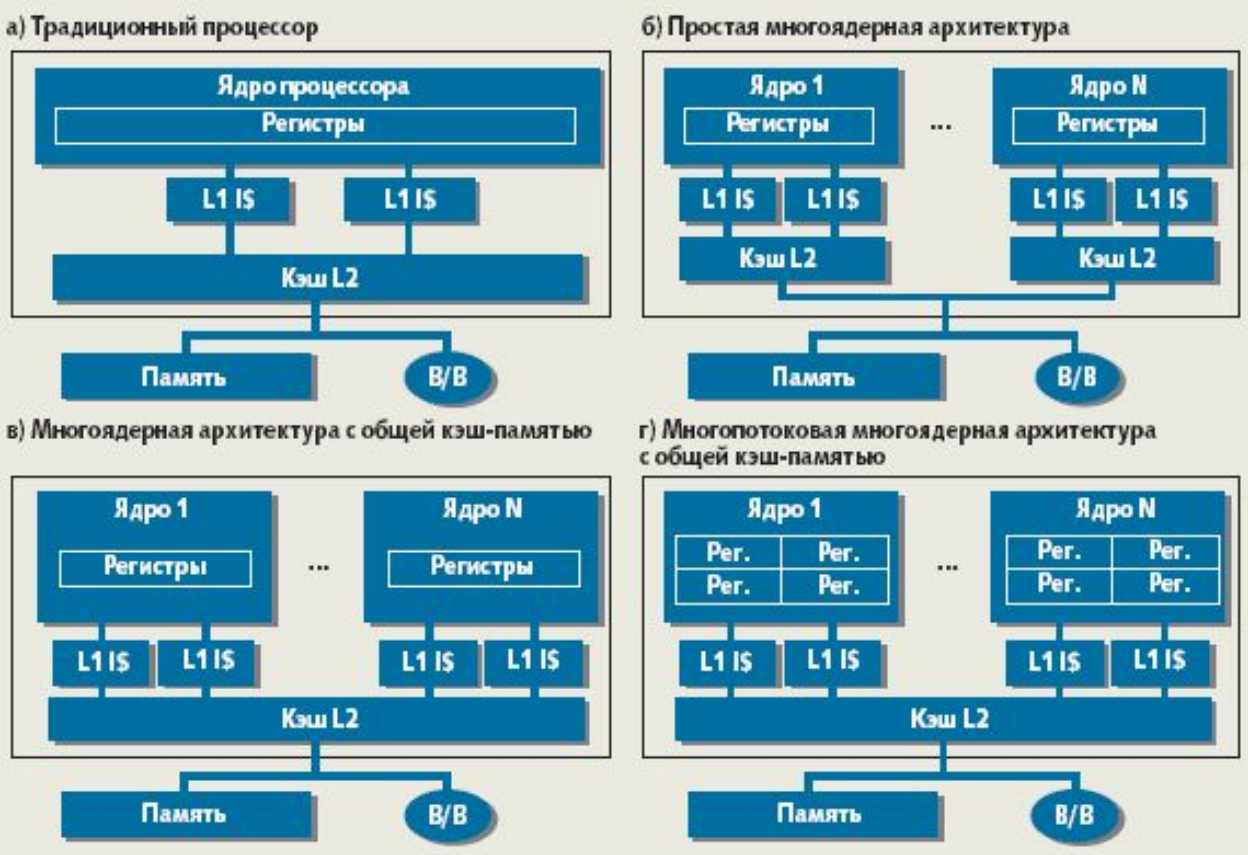
Поток – последовательно выполняемая ветвь кода, для которой выделен отдельный стек и обеспечивается независимость использования регистров процессора.

Многопоточность (multithreading) – несколько потоков выполнения кода работают внутри одной программы.

Способы реализация параллельных вычислений

Главный поток – поток, создаваемый для выполнения программы по умолчанию.

Способы реализация параллельных вычислений



Максимально нагруженная программа

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main( string[] args )
        {
            while ( true )
            {

            }
        }
    }
}
```

Максимально нагруженная программа

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main( string[] args )
        {
            while ( true )
            {

            }
        }
    }
}
```

На сколько % будет загружен четырех-ядерный процессор?

Максимально нагруженная программа

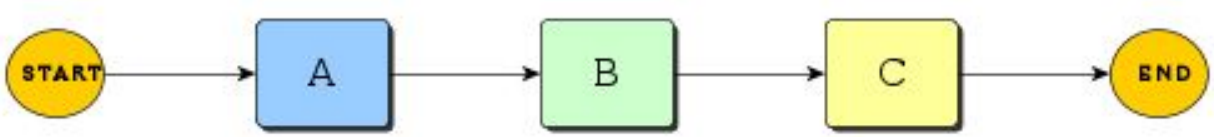
```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main( string[] args )
        {
            while ( true )
            {

            }
        }
    }
}
```

Image Name	PID	User Name	CPU	CPU
System Idle Process	0	SYSTEM	71	648:
ConsoleApplication1.vshost.exe *32	33936	alexey.gerasimov	25	0:
ScriptedSandbox64.exe	29508	alexey.gerasimov	01	0:

Асинхронные операции vs паралелльные вычисления

Проблемы UI

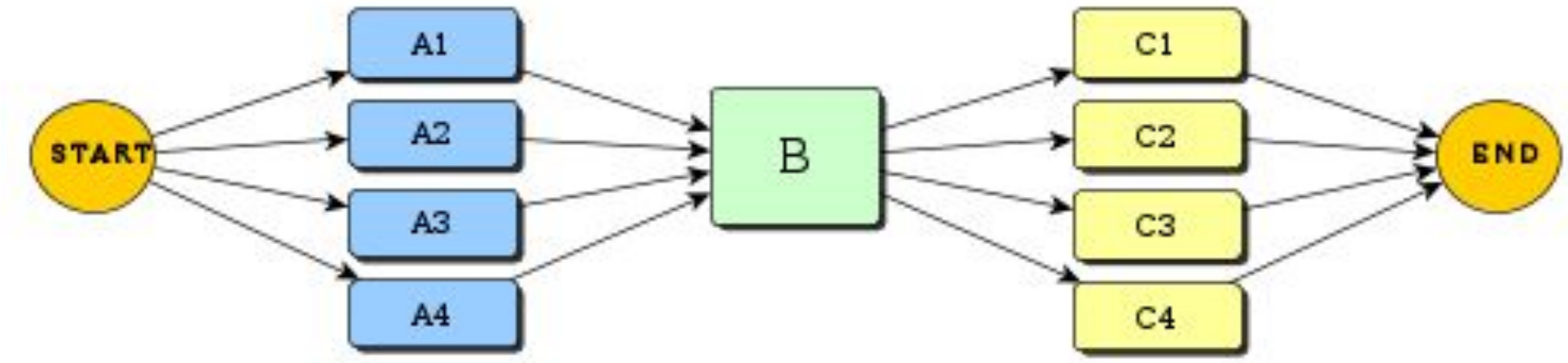


Обрабатываются запросы пользователя.

Рисуется интерфейс.

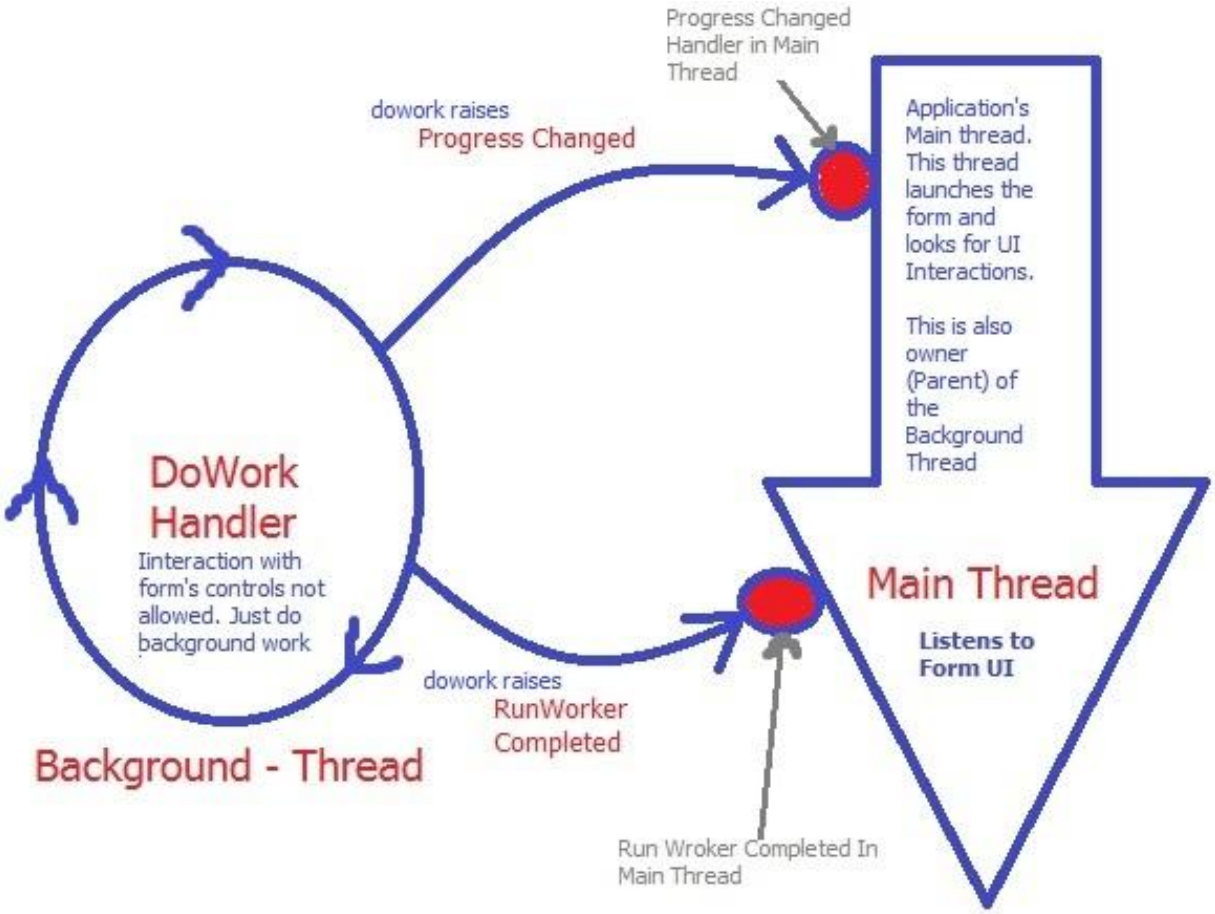
Выполняется полезная работа.

Многопоточность



Операции A1-A4 могут выполняться независимо, за их одновременное выполнение отвечает планировщик Windows. Если одна задача подвиснет, то после истечения определенного кванта времени ее сменит другая задача.

Отзывчивость интерфейса



Начальная ситуация

Повар готовит роллы == последовательная программа

Параллельность

Много поваров готовит роллы == параллельная реализация

Асинхронность

Повар готовит роллы, помощник варит рис == асинхронность

Асинхронные операции vs паралелльные вычисления

Операции, которые выполняются не прерывая основной поток выполнения программы, называются асинхронными. Эти операции служат не для ускорения работы программы, а для удобства использования программы.

Ускорением занимаются параллельные вычисления – это решение какой-то задачи с разбиением задачи на подзадачи, выполняющиеся в разных потоках одновременно.

Асинхронные операции

Примеры

- Скачивание интернет-ресурса
- Взаимодействие с сервером
- Фоновое копирование файлов в Total Commander
- И т.д.

Потоки

Потоки

Стандарт POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995) определяет API для управления потоками, их синхронизации и планирования.

- Типы данных:
  - pthread_t: дескриптор потока;
  - pthread_attr_t: перечень атрибутов потока;
  - pthread_barrier_t: барьер;
  - pthread_barrierattr_t: атрибуты барьера;
  - pthread_cond_t: условная переменная;
  - pthread_condattr_t: атрибуты условной переменной;
  - pthread_key_t: данные, специфичные для потока;
  - pthread_mutex_t: мьютекс;
  - pthread_mutexattr_t: атрибуты мьютекса;
  - pthread_rwlock_t: ;
  - pthread_rwlockattr_t: ;
  - pthread_spinlock_t: спинлок;
- Функции управления потоками:
  - pthread_create(): создание потока.
  - pthread_exit(): завершение потока (должна вызываться функцией потока при завершении).
  - pthread_cancel(): отмена потока.
  - pthread_join(): подключиться к другому потоку и ожидать его завершения; поток, к которому необходимо подключиться, должен быть создан с возможностью подключения (PTHREAD_CREATE_JOINABLE).
  - pthread_detach(): отключиться от потока, сделав его при этом отдельным (PTHREAD_CREATE_DETACHED).
  - pthread_attr_init(): инициализировать структуру атрибутов потока.
  - pthread_attr_setdetachstate(): указывает параметр "отделимости" потока (detach state), который говорит о возможности подключения к нему (при помощи pthread_join) других потоков (значение PTHREAD_CREATE_JOINABLE) для ожидания окончания или о запрете подключения (значение PTHREAD_CREATE_DETACHED); ресурсы отдельного потока (PTHREAD_CREATE_DETACHED) при завершении автоматически освобождаются и возвращаются системе.
  - pthread_attr_destroy(): освободить память от структуры атрибутов потока (уничтожить дескриптор).
- Функции синхронизации потоков:
  - pthread_mutex_init(), pthread_mutex_destroy(), pthread_mutex_lock(), pthread_mutex_trylock(), pthread_mutex_unlock(): с помощью мьютексов.
  - pthread_cond_init(), pthread_cond_signal(), pthread_cond_wait(): с помощью условных переменных.

Основные функции

- Создание потока
  - Передача параметров в поток
- Ожидание окончания потока
- Установка приоритета потока
- Привязка потока к ядру
- Окончание потока
- Исключения, возникающие в потоке

Пример

```
#include "stdafx.h"
#include <windows.h>
#include <string>
#include <iostream>

DWORD WINAPI ThreadProc(CONST LPVOID lpParam)
{
  char* ch = (char*)lpParam;
  for(int i = 0; i < 100; i++)
    std::cout << ch;
  ExitThread(0);
}

int _tmain(int argc, _TCHAR* argv[])
{
	char* ch = "A\0";
	char* ch2 = "B\0";

	HANDLE* handles = new HANDLE[2];
	handles[0] = CreateThread(NULL, 0, &ThreadProc, ch, CREATE_SUSPENDED, NULL);
	handles[1] = CreateThread(NULL, 0, &ThreadProc, ch2, CREATE_SUSPENDED, NULL);

	ResumeThread(handles[0]);
	ResumeThread(handles[1]);

	WaitForMultipleObjects(2, handles, true, INFINITE);
	return 0;
}
```

Пример 1. Результат

BBBBBBBBBBBBBBBBBBBABABABABAB
ABABABABABABABABABABABABABABA
BABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABA
BABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABA
BABABABABABABABABABABAAAAAAAAA
AAAAAAAAAA

Создание потока

CreateThread

- *lpThreadAttributes* – указатель на SECURITY_ATTRIBUTES (чаще всего NULL)
- *dwStackSize* - размер стека в байтах
- *lpStartAddress* - указатель на потоковую процедуру
- *lpParameter* - параметр потока (4 байта)
- *dwCreationFlags* - параметры создания (чаще всего 0 – поток запускается сразу после создания)

Удаление потока

Остановка выполнения

- TerminateThread
  - hThread – хендл потока
  - dwExitCode – код выхода потока

Удаление хендла

- CloseHandle
  - hThread – хендл потока

Изменение приоритета потока

Остановка выполнения

- TerminateThread
  - hThread – хендл потока
  - dwExitCode – код выхода потока

Удаление хендла

- CloseHandle
  - hThread – хендл потока

Другие операции

```
SetThreadPriority(handles[0], THREAD_PRIORITY_ABOVE_NORMAL);

SetThreadPriority(handles[1], THREAD_PRIORITY_BELOW_NORMAL);
```

Every thread has a base priority level determined by the thread's priority value and the priority class of its process. The system uses the base priority level of all executable threads to determine which thread gets the next slice of CPU time. Threads are scheduled in a round-robin fashion at each priority level, and only when there are no executable threads at a higher level does scheduling of threads at a lower level take place.

Результат выполнения?

ABABABABABABABABABABABABABABA
BABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABA
BABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABA
BABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABA
BABABABABAB

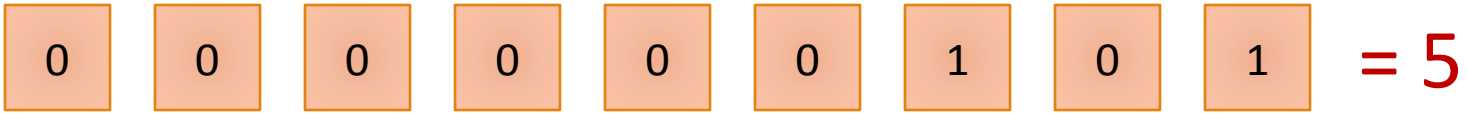
Привязка потоков к ядрам

Привязка потоков к ядрам

SetThreadAffinityMask(HANDLE hThread, DWORD_PTR dwThreadAffinityMask)

dwThreadAffinityMask – число, установленный i-ый бит (== 1), которого разрешают потоку исполняться на i-ом ядре

Например, чтобы разрешить исполнение на 0 и 2 ядре:



Привязка потоков к ядрам

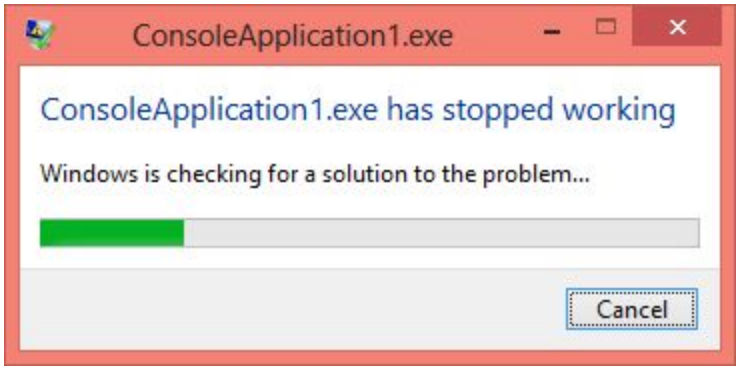
SetThreadAffinityMask(handles[0], 1);

SetThreadAffinityMask(handles[1], 1);

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

Исключения в потоках

```
if( i == 99)    i = i / 0; // exception thrown
```



Вопросы