Параллельное программирование

- ✓~7 лекций;
- ✓~7 лабораторных занятий;
- ✓РГР.

Литература и источники

1. Параллельное и распределенное программирование с использованием C++. Камерон Хьюз, Трейси Хьюз, 2004
2. Корнеев В.Д. Параллельное программирование кластеров: учеб. пособие – Изд-во НГТУ, 2008.
3. **http://parallel.ru/tech**
4. Антонов А.С. Параллельное программирование с использованием технологии OpenMP. Изд-во МГУ, 2009.
5. Антонов А.С. Параллельное программирование с использованием технологии MPI . Изд-во МГУ, 2004.
6. Стандарт MPI **http://www.cluster.bsu.by/MPI_ALL.htm**
7. Стандарт OpenMP **http://openmp.org/wp/openmp-specifications/**
8. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA
9. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров

ПП – направление которое должно в будущем существенно изменить представление о программировании в целом?

?!

ПП – специфический раздел программирования, связанный с описанием и разработкой программ части которых могут выполняться ***одновременно***.

Классификаций архитектур ЭВМ по признакам наличия параллелизма

предложена М. Флинном в 1966 году.

- *SISD* – (*Single Instruction Single Data*)
- *SIMD* – (*Single Instruction Multiple Data*)
- *MISD* – (*Multiple Instruction Single Date*)
- *MIMD* – (*Multiple Instruction Multiple Date*)

Современные аппаратные возможности

- Векторизация
- Мультипроцессоры
- Кластеры, сети рабочих станций
- Графические устройства

Параллельное и распределенное программирование

Использование общей памяти

или

взаимодействие с помощью передачи сообщений.

Параллельное - многопоточное программирование. Инструменты:

- POSIX Threads
- Windows threads
- C++ std::thread
- C# System.Threading
- Java потоки
- Boost threads
- intel TBB
- OpenMP
- ….

Средства операционных систем.

Стандартные средства языков программирования.

Дополнительные библиотеки и расширенные возможности компиляторов.

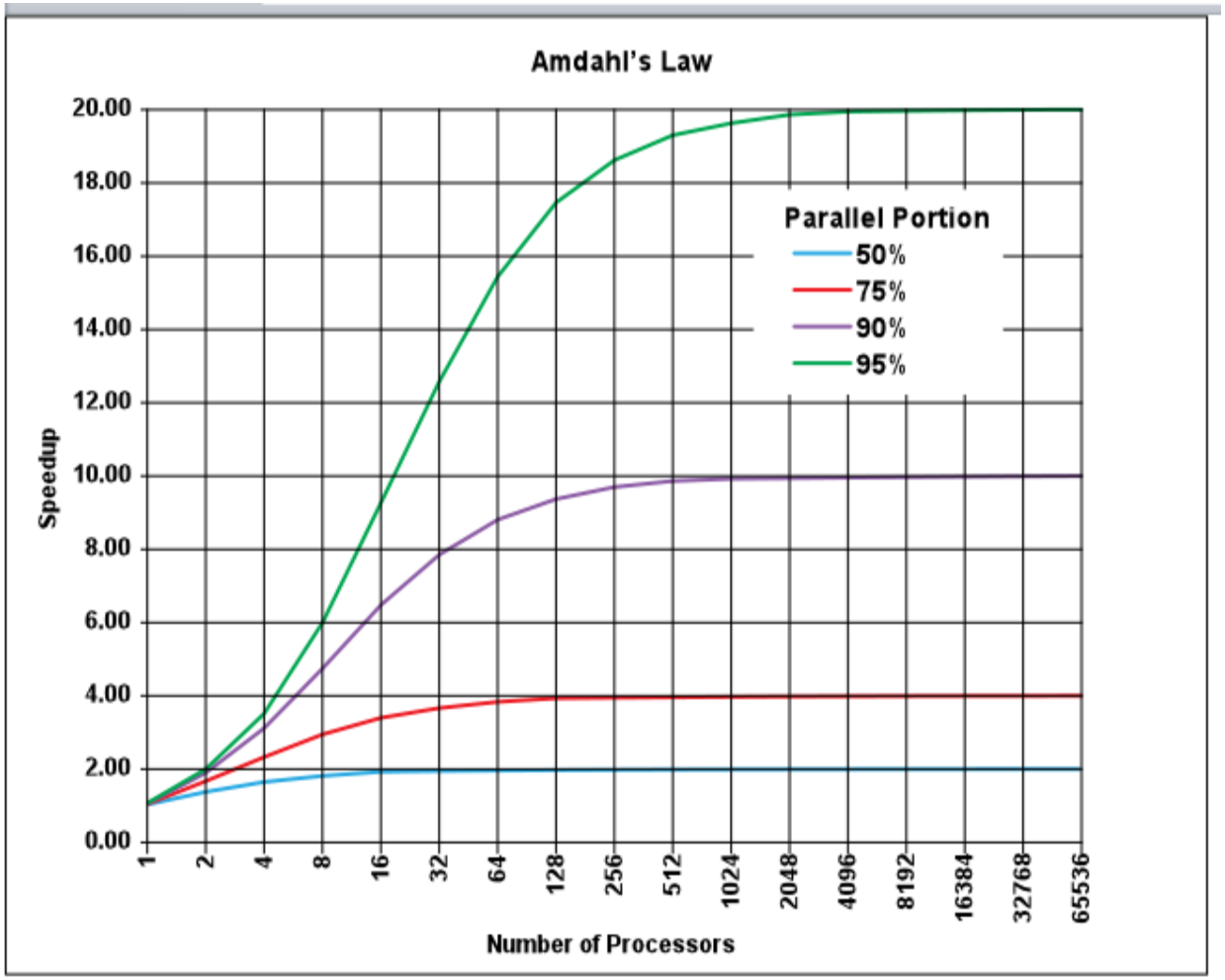
Распределенное программирование - передача сообщений

Части распределенной программы ***должны иметь возможность*** выполняться на различных компьютерах (возможно разной архитектуры).

- Самостоятельное решение;
- PVM (Parallel Virtual Machine) – виртуальная параллельная машина;
- MPI (Message Passing Interface) – интерфейс передачи сообщений;
- ….

Закон Амдала

Ускорение выполнения программы за счёт распараллеливания ограничено временем, необходимым для выполнения её последовательных частей.



α - доля последовательных вычислений;

p – количество исполнителей;

S_p – ускорение.

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

Выявление возможностей параллельного выполнения

- Декомпозиция по данным
- Декомпозиция по задачам

Модели параллельного программирования

- SPMD (single program, multiple data)
- MPMD (multiple program, multiple data)

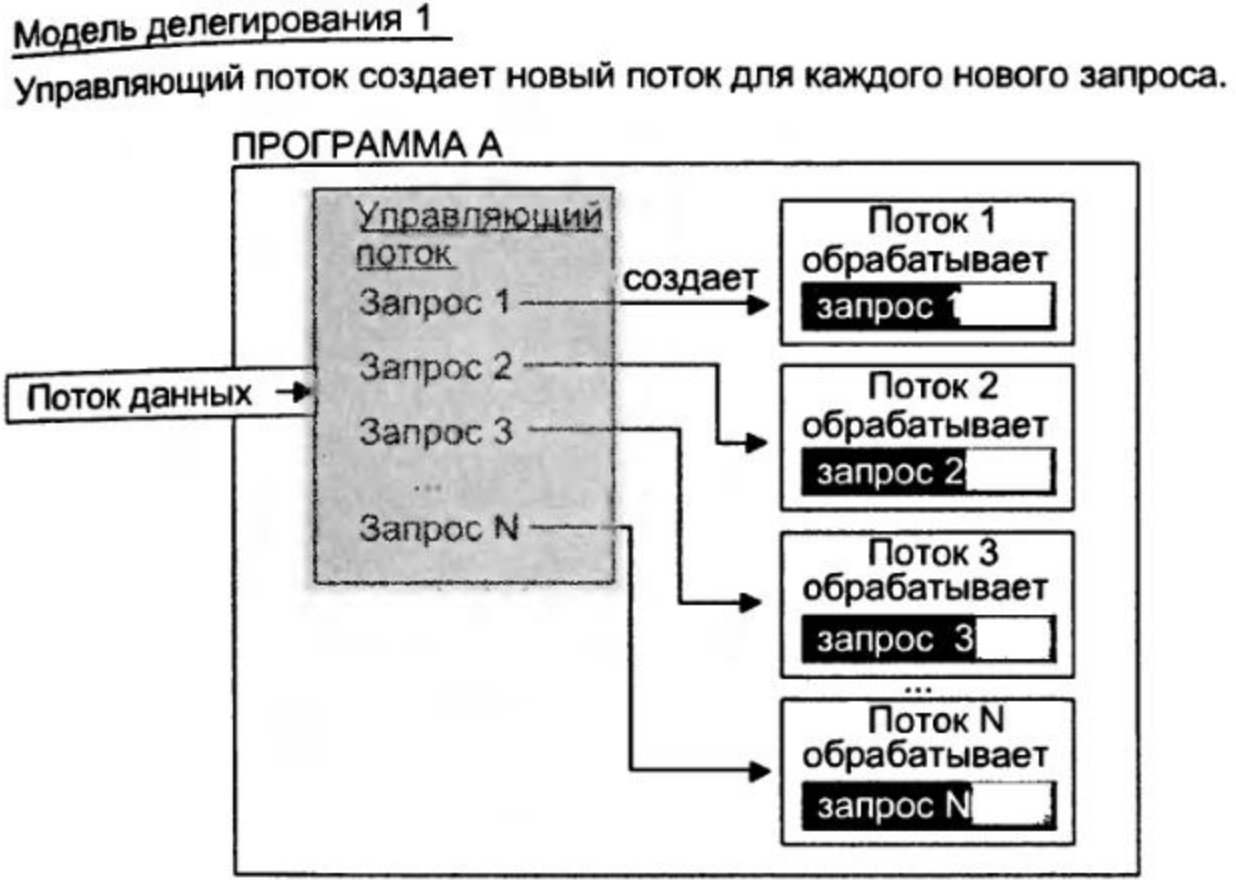
Модели создания и функционирования потоков:

- делегирование («управляющий-рабочий»);
- сеть с равноправными узлами;
- «изготовитель-потребитель».
- конвейер;

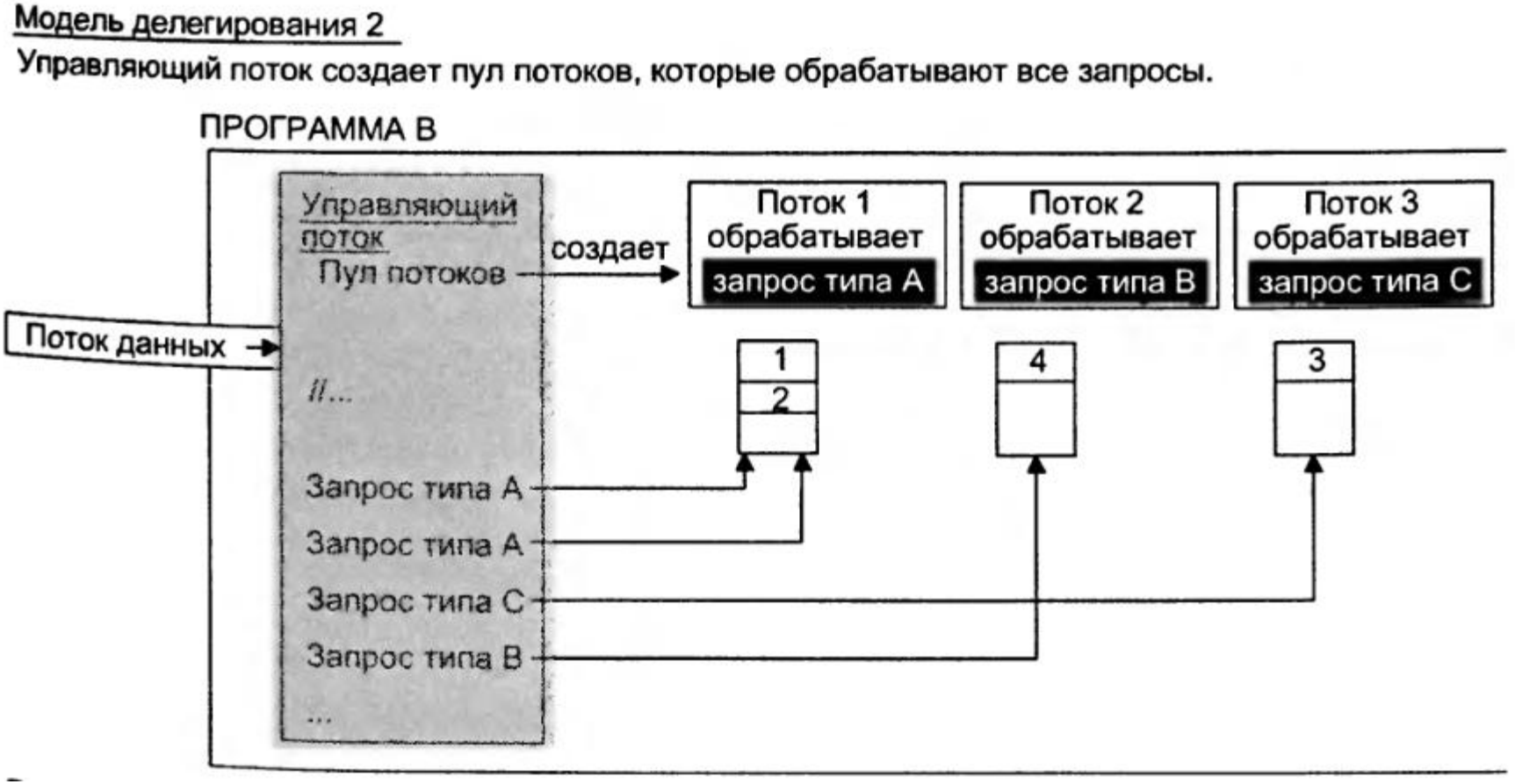
Модель делегирования

Центральный поток («управляющий») создает потоки («рабочие»), назначая каждому из них задачу. Управляющий поток может ожидать до тех пор, пока все потоки не завершат выполнение своих задач.

Модель делегирования

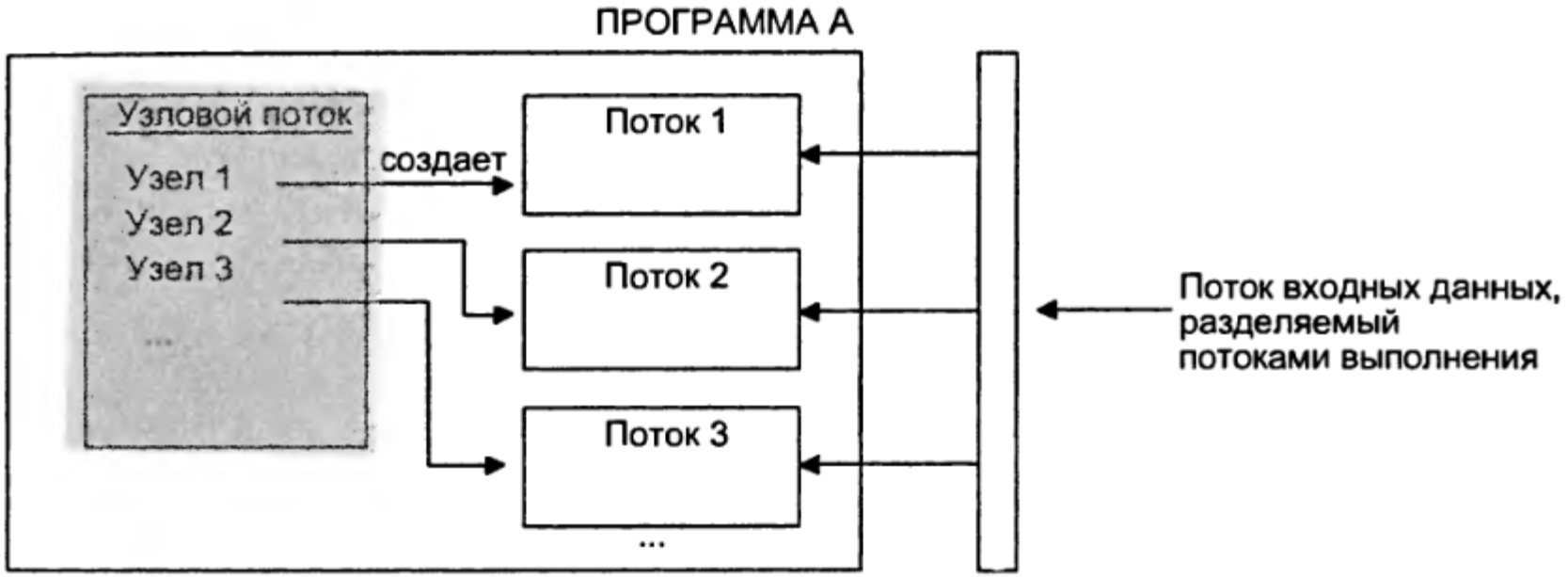


Модель делегирования

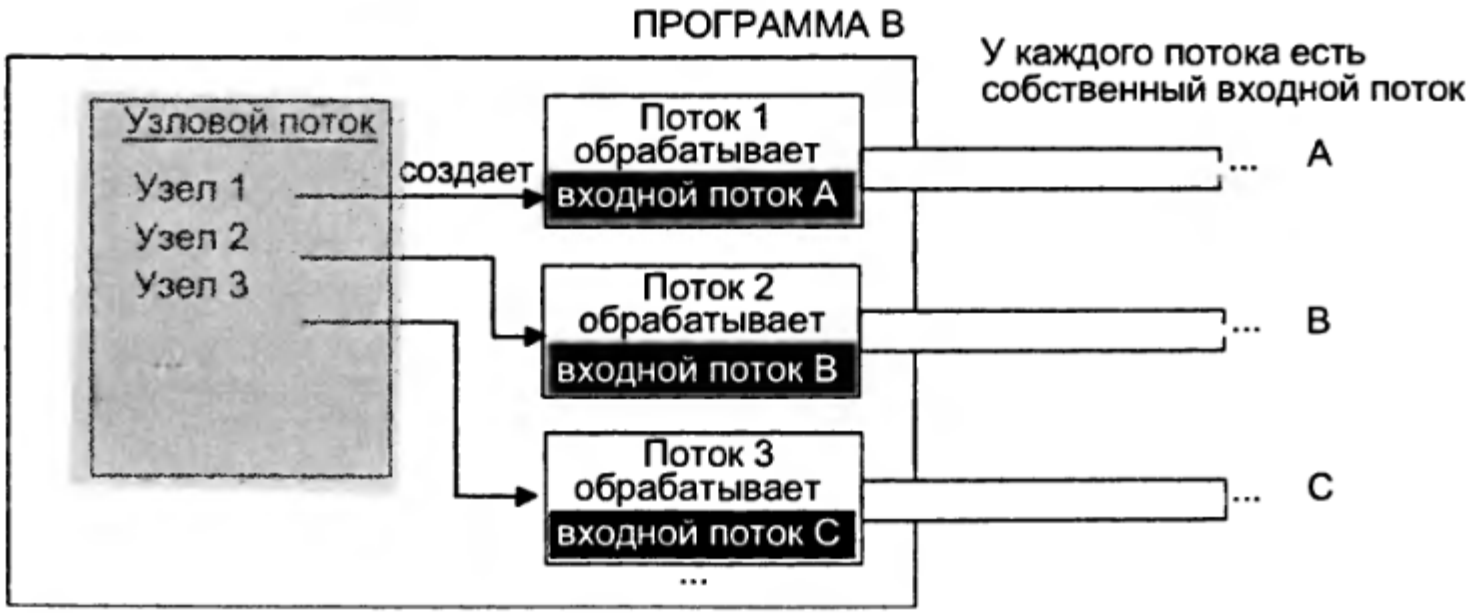


Модель с равноправными узлами

Все потоки имеют одинаковый рабочий статус. Несмотря на существование одного потока, который изначально создает остальные.

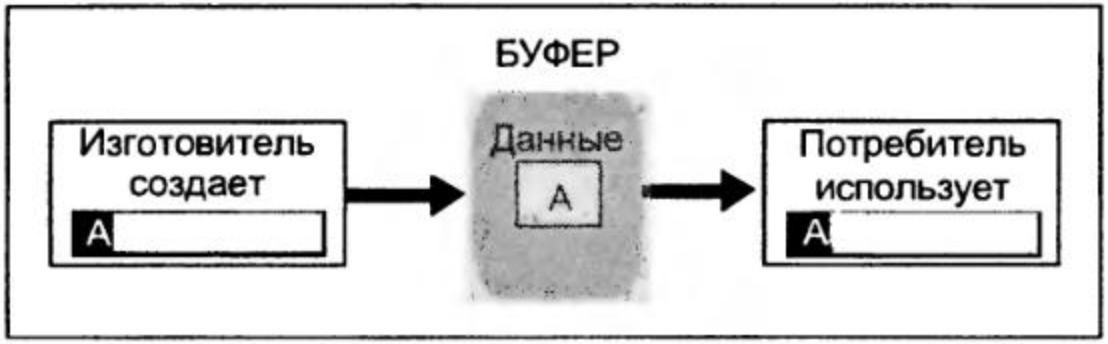


Модель с равноправными узлами



Модель «изготовитель-потребитель»

В модели «изготовитель-потребитель» существует поток-«изготовитель», который готовит данные, потребляемые потоком-«потребителем». Данные сохраняются в блоке памяти, разделяемом между ними.



Модель конвейера

Предполагает наличие потока элементов, которые обрабатываются поэтапно. На каждом этапе отдельный поток выполняет некоторые операции над элементом данных.

Наиболее распространенная проблема - «Гонка данных», «Состояние гонки» (race condition)

- это ситуация в которой несколько потоков конкурируют за обладание некоторым общим ресурсом.

Решение – грамотно реализованная синхронизация.

Одна из основных проблем синхронизации – «взаимная блокировка» (deadlock)

- это ситуация, в которой несколько потоков находятся в состоянии ожидания ресурсов, занятых друг другом.

Пример:

Поток$_1$ Хочет захватить A и B, начинает с A, захватывает его и ожидает освобождения ресурса B.

Поток$_2$ Хочет захватить A и B, начинает с B, захватывает его и ожидает освобождения ресурса A.

Другая проблема синхронизации и не только – «бесконечное ожидание»

- это ситуация, в которой поток ожидает некоторого события (поступления входных данных, сигнала из другого потока, освобождения общего ресурса) которое никогда не произойдет.

Средства описания и анализа работы параллельных алгоритмов в UML 2.0 (*Unified Modeling Language*).

Диаграмма последовательности (*sequence diagram*)

Иллюстрирует жизненный цикл нескольких объектов.

Основные элементы:

1. прямоугольники – объекты;
2. прямоугольники – деятельность объектов;
3. стрелки – взаимодействия.

Может использоваться для моделирования и анализа различных вариантов поведения потоков.

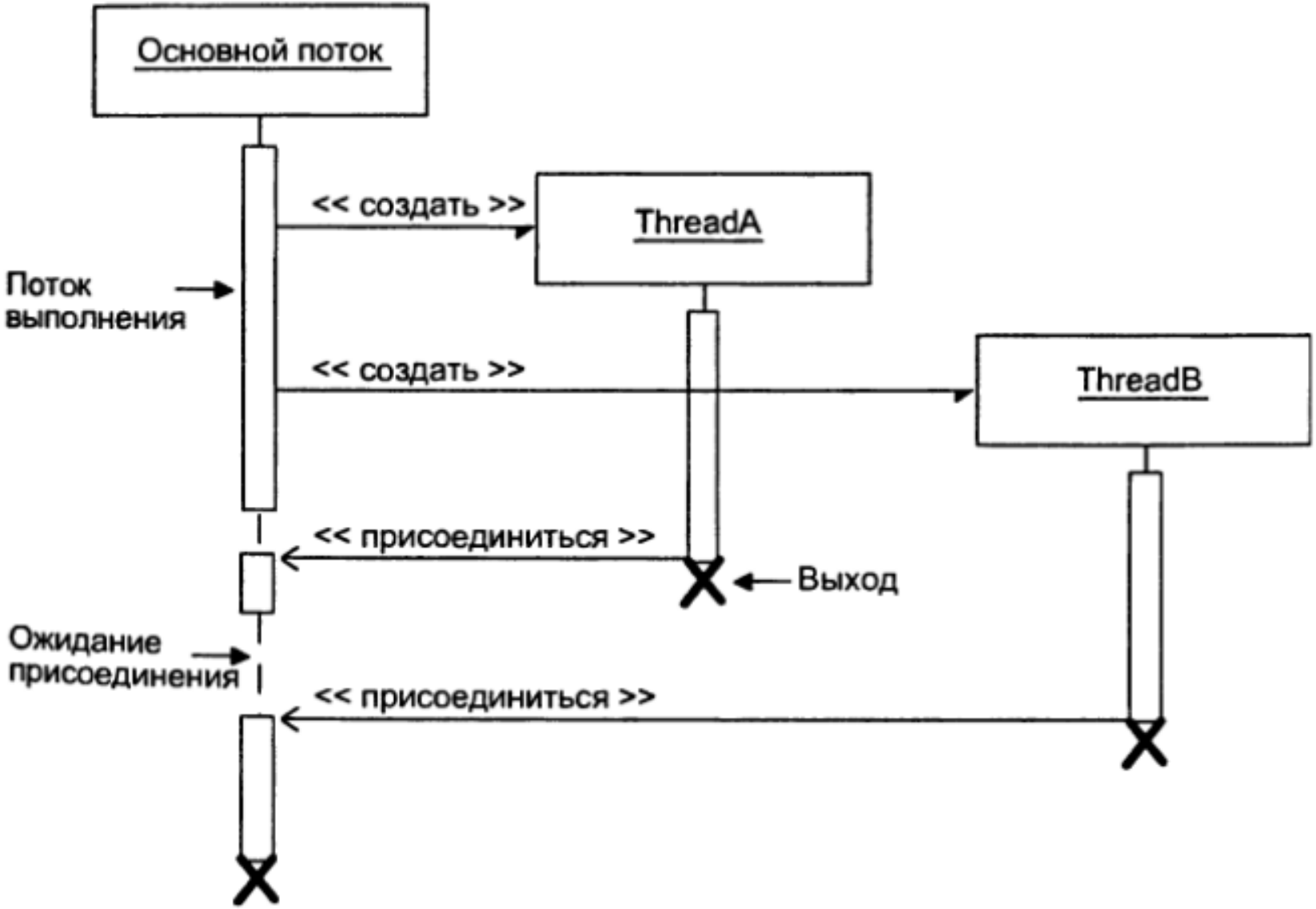

Основной поток
<< создать >>
ThreadA
Поток
выполнения
<< создать >>
ThreadB
<< присоединиться >>
Выход
Ожидание
присоединения
<< присоединиться >>

Диаграмма деятельности (*activity diagram*)

Имеет много общего с блок-схемой, содержит ряд средств подходящих ***для описания параллельного алгоритма***.

Точка разделения
и
Точка слияния

Прием события (receive event action)

ожидает некоторого события.

Передача сигнала (send signal action)

передает сигнал внешней системе (*другому потоку*).

POSIX

(Portable Operating System Interface for Unix)

- набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой.

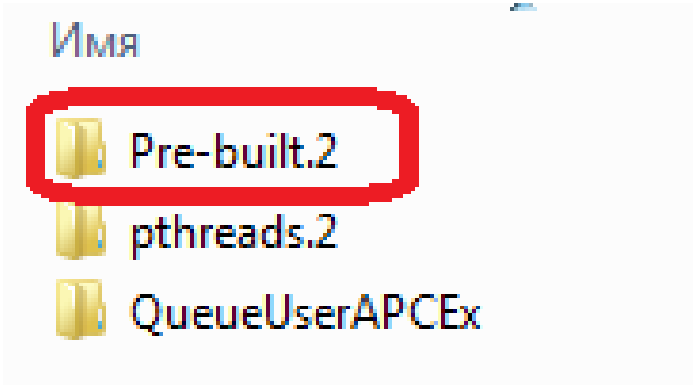
POSIX Threads – стандарт определяющий API для управления потоками.

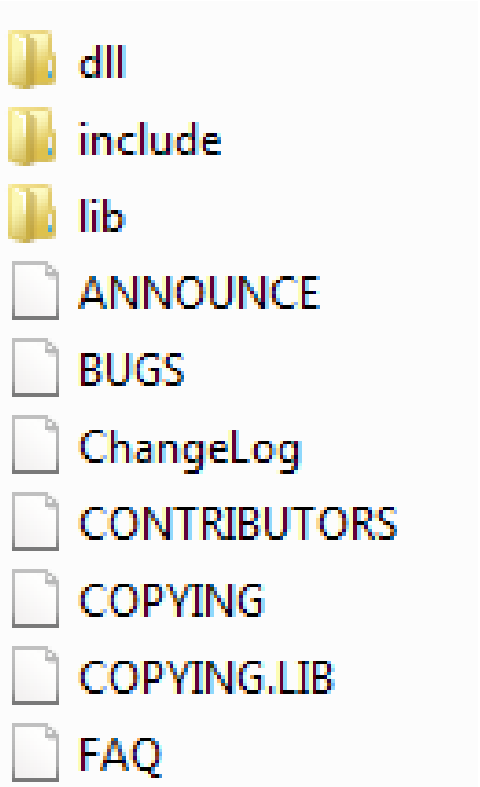
Эмуляция под Windows:

Open Source POSIX Threads for Win32

http://sourceware.org/pthreads-win32/
ftp://sourceware.org/pub/pthreads-win32

pthreads-w32-2-9-1-release.zip






```
#include <pthread.h>
#pragma comment(lib, "pthreadVCE2.lib")
```

Создание потоков

```
int pthread_create(     pthread_t *thread,
                        pthread_attr_t *attr,
                        void * (*start_routine)(void *),
                        void *arg);
```

thread – дескриптор потока (будет записан);

attr – атрибуты потока (NULL - по умолчанию);

start_routine – адрес функции которая будет выполняться в созданном потоке;

arg – аргумент функции по адресу start_routine.

Возвращаемое значение 0 или код ошибки.

Присоединяемые и откреплённые потоки

Откреплённым называется поток, который не синхронизирован с другими потоками, т.е. Не существует потока, который мог бы дожидаться его завершения.

По умолчанию потоки создаются как присоединяемые.

Для открепления можно использовать функцию

int pthread_detach(pthread_t tid);

Создание открепленных потоков с помощью объекта атрибутов

```
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

```
int pthread_attr_setdetachstate(
                            pthread_attr_t *attr,
                            int *detachstate);
int pthread_attr_getdetachstate(
                            const pthread_attr_t *attr,
                            int *detachstate);
```

Параметр detachstate может принимать одно из следующих значений:

PTHREAD_CREATE_DETACHED
PTHREAD_CREATE_JOINABLE

Присоединение потоков

```
int pthread_join(pthread_t threadid, void **value_ptr);
```

Переводит поток, из которого она была вызвана в состояние ожидания до тех пор, пока не завершится поток, определяемый идентификатором threadid.

Необходимо вызывать для всех присоединяемых потоков с целью освобождения ресурсов

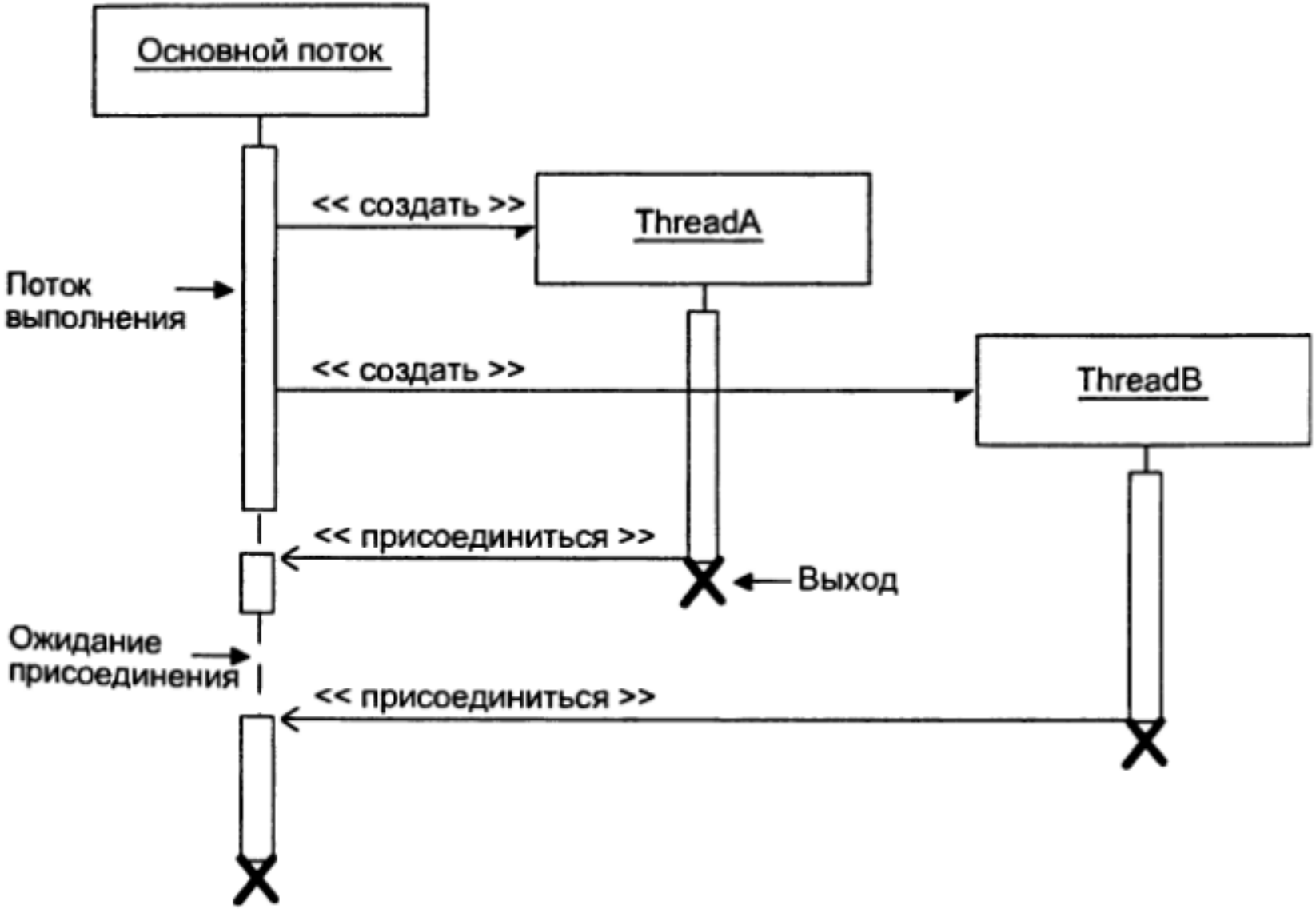
```
#include <iostream>
#include <pthread.h>

void *task1(void *X) // Определяем задачу для выполнения
                     // потоком ThreadA.
{
   //...
   cout << "Поток A завершен." << endl;
}

void *task2(void *X) // Определяем задачу для выполнения
                     // потоком ThreadB.
{
   //...
   cout << "Поток B завершен." << endl;
}

int main(int argc, char *argv[])
{
   pthread_t ThreadA,ThreadB; // Объявляем потоки.

   pthread_create(&ThreadA,NULL,task1,NULL); // Создаем
                                             // потоки.
   pthread_create(&ThreadB,NULL,task2,NULL);
   // Дополнительная обработка.
   pthread_join(ThreadA,NULL); // Ожидание завершения
   pthread_join(ThreadB,NULL); // потоков.
   return (0);
}
```


Основной поток
<< создать >>
ThreadA
Поток
выполнения
<< создать >>
ThreadB
<< присоединиться >>
Выход
Ожидание
присоединения
<< присоединиться >>

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

   pthread_t ThreadA,ThreadB;
   pthread_attr_t DetachedAttr;
   int N;

   if(argc != 2){
      cout << "Ошибка" << endl;
      exit (1);
   }
   N = atoi(argv[1]);
   pthread_attr_init(&DetachedAttr);


    pthread_attr_setdetachstate(&DetachedAttr,
                                PTHREAD_CREATE_DETACHED);
    pthread_create(&ThreadA,NULL,task1,&N);
    pthread_create(&ThreadB,&DetachedAttr,task2,&N);
    cout << "Ожидание присоединения потока A." << endl;
    pthread_join(ThreadA,NULL);
    return (0);
}
```

Значение возвращаемое функцией выполняемой потоком

int pthread_join (pthread_t thread, void **status_addr);

Функция pthread_join блокирует работу вызвавшего ее потока до завершения потока с идентификатором thread.

После разблокирования в указатель, расположенный по адресу status_addr, заносится адрес, который вернул завершившийся thread.

Пример

```
int w;
void *task1(void *X)
{
//w=2;
//return &w;
int *st=new int;
*st=3;
return st;
}
```

```
int main(int argc, _TCHAR* argv[])
{
pthread_t thread_a;
pthread_create(&thread_a, NULL, task1, NULL);
int s;
void *ss;
pthread_join(thread_a, &ss);
s=*((int*)ss);
cout << endl << "s=" << s << endl;
system("pause");
return 0;
}
```