Contents

[РОЗДІЛ 1. ПОРІВНЯННЯ МОВ ПРОГРАММУВАННЯ Ada ТА Java 4](#_Toc293878280)

[1.1 Понятие процесса и потока 4](#_Toc293878281)

[1.2 Задача взаимного исключения 6](#_Toc293878284)

[1.3 Критическая секция 8](#_Toc293878285)

[1.4 Монитор Хоара 10](#_Toc293878286)

[1.5 Задача синхронизации 16](#_Toc293878287)

[1.6 Семафоры 16](#_Toc293878288)

[1.7 Механизм рандеву в Ada 17](#_Toc293878289)

[1.8. Средства управления потоками 18](#_Toc293878290)

[1.9 Приоритеты 23](#_Toc293878293)

[1.10 Приостановка и зварешение потоков 26](#_Toc293878294)

[Висновки до Розділу 2 30](#_Toc293878295)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРАЛЕЛЬНИХ КОМП’ЮТЕРНИХ СИСТЕМ 31](#_Toc293878296)

[2.1 Розробка паралельного математичного алгоритму 31](#_Toc293878297)

[2.2 Аналіз завдання 31](#_Toc293878298)

[2.3 Розробка ПЗ для системи з спільною пам’яттю 34](#_Toc293878300)

[2.4 Розробка ПЗ для системи з локальною пам’яттю 35](#_Toc293878305)

[Висновки до Розділу 2 37](#_Toc293878310)

[РОЗДІЛ 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 39](#_Toc293878311)

[3.2 Тестування ПЗ для системи з спільною пам’яттю 39](#_Toc293878312)

[3.3 Тестування ПЗ для системи з локальною пам’яттю 41](#_Toc293878313)

[Висновки до Розділу 3 42](#_Toc293878314)

[Література 45](#_Toc293878315)

[Додатки 46](#_Toc293878316)

# РОЗДІЛ 1. ПОРІВНЯННЯ МОВ ПРОГРАММУВАННЯ Ada ТА Java

## Понятие процесса и потока

Точного определения термина «процесс» на данный момент не существует. В общих словах, процесс – это абстрактное понятие, описывающее действия, связанные с выполнением задачи в рамках вычислительной системы.

Разработчики операционных систем разработали концептуальную модель последовательных процессов. В этой модели всё функционирующее на компьютере программное обеспечение (далее – ПО), иногда включая собственно операционную систему (далее – ОС), организовано в виде набора последовательных процессов или, для краткости, просто процессов. Процессом является выполняемая программа, включающая значения счётчика команд, регистров и переменных. С позиции данной абстрактной модели, у каждого процесса есть собственный виртуальный центральный процессор. На самом деле, разумеется, реальный процессор переключается с процесса на процесс, а свойство ОС, обеспечивающее переключение между задачами, называется многозадачностью.

Потоки – это части одной программы, которые выполняются параллельно. Часто их называют легковесными процессами, в то время как процесс, в рамках которого существуют потоки, получил название тяжеловесного процесса.

Как уже было сказало ранее, во время работы компьютерной системы (далее – КС) происходит постоянное переключение с одного процесса на другой. Это же происходит и для программы, если она состоит из нескольких потоков. Однако время, необходимое для переключения легковесных процессов, меньше времени, необходимого для переключения тяжёлых процессов. Также потоки одного приложения значительно более тесно взаимодействуют между собой, т.к. они созданы для решения общей задачи.

Следует заметить, что программист не может напрямую управлять переключением потоков. У него нет возможности дать команду, обеспечивающую переключение на исполнение определённого потока. Механизмы синхронизации, о которых речь пойдет ниже, управляют потоками косвенным образом.

### Многопоточность

В ОС, поддерживающих принцип разделения процессов на потоки, каждый процесс состоит как минимум из одного – главного – потока.

Все потоки выполняются в адресном [пространстве](http://ru.wikipedia.org/wiki/%D0%90%D0%B4%D1%80%D0%B5%D1%81%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы [файлов](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB). Однако, с каждым из потоков связан свой набор значений регистров процессора, называемый контекстом потока.

Поскольку все потоки процесса работают в едином адресном пространстве, обмен данными между ними крайне прост, однако при этом требуется согласовывать их работу над совместными данными. Под термином «синхронизация», как правило, имеют в виду именно согласование работы потоков, принадлежащих одному процессу.

### Планирование потоков

Чтобы все потоки работали, операционная система отводит каждому из них определенное процессорное время. Тем самым создается иллюзия одновременного выполнения потоков (разумеется, для многопроцессорных компьютеров возможен истинный параллелизм). В Windows реализована система вытесняющего планирования на основе приоритетов, в которой всегда выполняется поток с наибольшим приоритетом, готовый к выполнению. Выбранный для выполнения поток работает в течение некоторого периода, называемого квантом. Квант определяет, сколько времени будет выполняться поток, пока операционная система не прервет его. По окончании кванта операционная система проверяет, готов ли к выполнению другой поток с таким же (или большим) уровнем приоритета. Если таких потоков не оказалось, текущему потоку выделяется еще один квант. Однако поток может не полностью использовать свой квант. Как только другой поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется, даже если его квант еще не истек.

Квант не измеряется в каких бы то ни было единицах времени, а выражается целым числом. Для каждого потока хранится текущее значение его кванта. Когда потоку выделяется квант процессорного времени, это значит, что его квант устанавливается в начальное значение. Оно зависит от операционной системы. Например, для Win2000 Professional начальное значение кванта равно 6, а для Win2000 Server - 36.

Всякий раз, когда возникает прерывание от таймера, из кванта потока вычитается 3, и так до тех пор, пока он не достигнет нуля. Частота срабатывания таймера зависит от аппаратной платформы. Например, для большинства однопроцессорных x86 систем он составляет 10мс, а на большинстве многопроцессорных x86 систем - 15мс.

В любом случае операционная система должна определить, какой поток выполнять следующим. Выбрав новый поток, операционная система переключает контекст. Эта операция заключается в сохранении параметров выполняемого потока (регистры процессора, указатели на стек ядра и пользовательский стек, указатель на адресное пространство, в котором выполняется поток и др.), и загрузке аналогичных параметров для другого потока, после чего начинается выполнение нового потока.

Планирование в Windows осуществляется на уровне потоков, а не процессов. Это кажется понятным, так как сами процессы не выполняются, а лишь предоставляют ресурсы и контекст для выполнения потоков. Поэтому при планировании потоков, система не обращает внимания на то, какому процессу они принадлежат. Например, если процесс А имеет 10 готовых к выполнению потоков, а процесс Б - два, и все 12 потоков имеют одинаковый приоритет, каждый из потоков получит 1/12 процессорного времени.

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме мультипрограммирования, может привести к их неправильной работе или даже к краху системы. Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций. Все определяется взаимными скоростями процессов и моментами их прерывания. Поэтому отладка взаимодействующих процессов является сложной задачей. Ситуации подобные той, когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей процессов, называются **гонками**.

## 1.2 Задача взаимного исключения

#### Общие сведенья

Задача взаимного исключения состоит в том, что во время выполнения двух или более параллельных процессов может возникнуть одновременное обращение процессов к одному и тому же общему ресурсу. Такое обращение обычно приводит к конфликту процессов, который состоит либо в резком замедлении работы программы, либо в её некорректной работе.

Общая схема решения задачи взаимного исключения базируется на том, что необходимо приостановить процесс, который обращается к общему ресурсу, уже занятому на момент обращения другим процессом. Разблокирование должно быть выполнено сразу после освобождения общего ресурса.

Существует 2 подхода к решению задачи взаимного исключения. Первый подход базируется на контроле процессов (семафоры, мютексы, критические секции), а второй – на контроле общего ресурса (мониторы).

Таким образом, для решения задачи взаимного исключения должны выполняться следующие условия:

* в каждый момент времени только один процесс может находиться внутри критического участка;
* ни один процесс не должен оставаться неограниченно долго внутри критического участка;
* никакой процесс, находящийся вне критического участка, не должен препятствовать другому процессу войти в критический участок.

#### Решение в языке Ada

Ада позволяет задачам совместно использовать разделяемые (общие) переменные (объекты, ресурсы), которые располагаются в общей памяти и могут быть необходимы как для организации взаимодействия задач, так и для хранения каких-либо общих данных.

В стандарте Ada95 для указания разделяемых переменных используются следующие директивы компилятора:

**pragma** Atomic ( Local\_Name );

**pragma** Atomic\_Components ( Local\_Array\_Name );

**pragma** Volatile ( Local\_Name );

**pragma** Volatile\_Components ( Local\_Array\_Name );

Здесь Local\_Name указывает локальное имя объекта или описание типа, а Local\_Array\_Name указывает локальное имя объекта-массива или описание типа-массива.

Директивы компилятора Atomic и Atomic\_Components обеспечивают непрерывность операций чтения/записи для всех указанных в них объектах. Такие объекты называют атомарными (atomic), а операции над ними выполняются только последовательно.

Директивы компилятора Volatile и Volatile\_Components обеспечивают выполнение операций чтения/записи для всех указанных в них объектах непосредственно в памяти.

Примеры применения этих директив компилятора могут иметь следующий вид:

Array\_Size : Positive;

**pragma** Atomic (Array\_Size);

**pragma** Volatile (Array\_Size);

Store\_Array **is array** (1..Array\_Size) **of** Integer;

**pragma** Atomic\_Components (Store\_Array);

**pragma** Volatile\_Components (Store\_Array);

#### Решение в языке Java

Java предоставляет практически идентичное, с точки зрения программиста, решение: используя модификатор переменной volatile можно легко решить задачу взаимного исключения.

**volatile** boolean initialized = false;

Данный модификатор гарантирует защиту от одновременного доступа к переменной несколькими процессами. Это достигается с помощью встроенного в виртуальную машину Java механизма. В частности, данная нотация запрещает кэширование значения переменной (т.к. её значение может поменяться не очевидным для данного потока способом) и указывается компилятору, что изменения переменной нужно немедленно заносить в память.

## 1.3 Критическая секция

#### Общие сведения

Важным понятием синхронизации процессов является понятие "критическая секция" программы. Критическая секция - это часть программы, в которой осуществляется доступ к разделяемым данным. Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Этот прием называют взаимным исключением.

Простейший способ обеспечить взаимное исключение - позволить процессу, находящемуся в критической секции, запрещать все прерывания. Однако этот способ непригоден, так как опасно доверять управление системой пользовательскому процессу; он может надолго занять процессор, а при крахе процесса в критической области крах потерпит вся система, потому что прерывания никогда не будут разрешены.

Другим способом является использование блокирующих переменных. С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом), и значение 0, если ресурс занят.

Если все процессы написаны с использованием вышеописанных соглашений, то взаимное исключение гарантируется. Следует заметить, что операция проверки и установки блокирующей переменной должна быть неделимой. Поясним это. Пусть в результате проверки переменной процесс определил, что ресурс свободен, но сразу после этого, не успев установить переменную в 0, был прерван. За время его приостановки другой процесс занял ресурс, вошел в свою критическую секцию, но также был прерван, не завершив работы с разделяемым ресурсом. Когда управление было возвращено первому процессу, он, считая ресурс свободным, установил признак занятости и начал выполнять свою критическую секцию. Таким образом был нарушен принцип взаимного исключения, что потенциально может привести к нежелаемым последствиям. Во избежание таких ситуаций в системе команд машины желательно иметь единую команду "проверка-установка", или же реализовывать системными средствами соответствующие программные примитивы, которые бы запрещали прерывания на протяжении всей операции проверки и установки.

#### Решение в языке Ada

Непосредственного решения нет.

#### Решение в языке Java

Синхронизированные блоки в языке Java определяют в программе фрагмент кода потока, вход в который осуществляется в режиме взаимного исключения. Это означает, что при попытке входа в синхронизированный блок выполняется проверка, не занят ли синхронизированный блок другим потоком. Если занят, то вызывающий поток блокируется до тех пор, пока второй поток не покинет свой синхронизированный блок. Для указывания синхронизируемых блоков они должны связываться с помошью общих объектов:

**synchronized** (sync\_object) {

// Операции над общими ресурсами

}

## Монитор Хоара

#### Общие сведения

Монитор – это объект, который содержит общие ресурсы и процедуры доступа к ним. При чём доступ к общим переменным возможен только с помощью процедур монитора.

Монитор не является процессом, это пассивный объект, который приходит в активное состояние только тогда, когда какой-то объект обращается к нему за услугами. Часто монитор сравнивают с запертой комнатой, от которой имеется только один ключ. Услугами этой комнаты может воспользоваться только тот, у кого есть ключ. При этом процессу запрещается оставаться там сколь угодно долго. Другой процесс должен ждать до тех пор, пока первый не выйдет из нее и передаст ключ.

В качестве примера программы-монитора может выступать планировщик ресурсов. Действительно, каждому процессу когда-нибудь понадобятся ресурсы и он будет обращаться к планировщику. Но планировщик одновременно может обслуживать только один ресурс.

Иногда монитор задерживает обратившийся к нему процесс. Это происходит, например, в случае обращения за занятым ресурсом. Монитор блокирует процесс с помощью команды "ЖДАТЬ", а в случае освобождения ресурса выдает команду "СИГНАЛ". При этом освободившийся ресурс предоставляется одному из ожидавших его процессов вместе с разрешением на продолжение работы. Управление передается команде монитора, непосредственно следующей за операцией "ЖДАТЬ".

Мониторы более гибки, чем семафоры. В форме мониторов сравнительно легко можно реализовать различные синхронизирующие примитивы, в частности семафоры и почтовые ящики. Кроме того, мониторы позволяют нескольким процессам совместно использовать программу, представляющую собой критический участок.

#### Решение в языке Ada

Средством, которое позволяет организовать взаимно исключающий доступ к данным из разных, одновременно выполняющихся задач, являются защищенные модули, которые были введены стандартом Ada95. Характерной особенностью защищенных модулей является обеспечение ими синхронизированного доступа к приватным данным, однако в противоположность задачам, которые являются активными сущностями, защищенные модули - пассивны.

Защищенные модули (типы и объекты) Ады инкапсулируют данные и позволяют осуществлять доступ к ним с помощью защищенных подпрограмм или защищенных входов. Стандарт языка гарантирует, что в результате выполнения кода таких подпрограмм и входов изменение содержимого данных будет производиться в режиме взаимного исключения без необходимости создания дополнительной задачи.

Защищенный модуль может быть описан как защищенный тип или как одиночный защищенный объект, что аналогично одиночной задаче. В последнем случае предполагается, что защищенный объект имеет соответствующий анонимный тип. Следует учитывать, что защищенный тип является лимитированным и, тем самым, не обладает предопределенными операциями присваивания или сравнения.

Подобно задаче или пакету, защищенный модуль имеет спецификацию и тело. Спецификация описывает протокол доступа к защищенному модулю (интерфейс), и может содержать спецификации процедур, функций и входов защищенного модуля. Тело описывает детали реализации протокола доступа к данным защищенного модуля и, как правило, содержит тела защищенных подпрограмм и входов. Подобно задачам и записям, защищенный модуль может иметь дискриминанты дискретного или ссылочного типа.

В качестве простой иллюстрации рассмотрим пример следующего одиночного защищенного объекта:

-- спецификация защищенного объекта

**protected** Variable **is**

**function** Read **return** Item;

**procedure** Write(New\_Value : Item);

**private**

Data : Item;

**end** Variable;

-- тело защищенного объекта

**protected body** Variable **is**

**function** Read **return** Item **is**

**begin**

**return** Item;

**end**;

**procedure** Write (New\_Value : Item) **is**

**begin**

Data := New\_Value;

**end**;

**end** Variable;

Защищенный объект Variable предоставляет управляемый доступ к приватной переменной Data типа Item. Функция Read позволяет читать, а процедура Write - обновлять текущее значение переменной Data.

Защищаемые данные и данные о состоянии объекта должны быть помещены в приватную часть спецификации. Смысл этого заключается в том, что приватная часть не доступна клиенту непосредственно, а наличие всей информации в интерфейсе защищенного объекта необходимо компилятору для эффективного распределения памяти.

Защищенные процедуры предусматривают взаимно исключающий доступ к данным защищенного модуля по чтению и/или записи. Защищенные функции предоставляют одновременный доступ к данным защищенного модуля только по чтению, что подразумевает одновременное выполнение множества вызовов функций. Однако вызовы защищенных процедур и защищенных функций остаются взаимно исключающими. Порядок, в котором разные задачи ожидают выполнения защищенных процедур и защищенных функций, стандартом не определяется. Однако поддержка требований приложения D (Annex D) стандарта Ada95, в котором указаны требования для систем реального времени, позволяет сделать некоторые предположения о возможном порядке выполнения подпрограмм.

Для обращения к защищенным подпрограммам используется традиционная точечная нотация:

X := Variable.Read;

. . .

Variable.Write (New\_Value => Y);

Внутри тела защищенного объекта допускается несколько подпрограмм, при этом реализация Ада-системы будет гарантированно осуществлять вызовы подпрограмм по принципу взаимного исключения.

По аналогии со входами задач, защищенный модуль может иметь защищенные входы. Действия, выполняемые при вызове защищенного входа, предусматриваются в его теле. Защищенные входы подобны защищенным процедурам в том, что они гарантируют взаимно исключающий доступ к данным защищенного модуля по чтению и/или записи. Однако внутри тела защищенного модуля защищенные входы предохраняются логическим выражением, которое называют барьером, а результат вычисления этого логического выражения должен иметь предопределенный логический тип Standard.Boolean.

Если при вызове защищенного входа значение барьера есть False, то выполнение вызывающей задачи приостанавливается до тех пор, пока значение барьера не станет равным True и внутри защищенного модуля будут отсутствовать активные задачи (задачи, которые выполняют тело какого-либо защищенного входа или какой-либо защищенной подпрограммы). Следовательно, вызов защищенного входа может быть использован для реализации условной синхронизации.

#### Решение в языке Java

В Java-программах можно выделять критические интервалы, которые обозначаются ключевым словом synchronized. Если критическим интервалом является метод (а в случае с концепцией мониторов это именно так), спецификатор synchronized помещается в его (метода) заголовок.

Выполнение участка начинается только после получения потоком монопольного доступа к соответствующему объекту. До наступления этого момента поток блокируется.

Вызов wait() внутри критического интервала приводит к тому, что текущий поток уступает монопольное право на критический интервал и приостанавливается до тех пор, пока из какого-либо другого потока не будет сделан вызов notify() или notifyAll(). С помощью этих методов мониторы в Java можно легко использовать для решения задачи синхронизации.

Рассмотрим код, представленный ниже:

public synchronized void transfer(int from, int to, int amount) {

if (accounts[from] < amount) return;

accounts[from] —= amount;

accounts[to] += amount;

ntransacts++;

}

Таким образом, если в потоке вызывается синхронизированный метод, то он гарантированно завершится до того, как другой поток запустит синхронизированный метод для обработки того же самого объекта. Если метод transfer вызывается в одном, а затем в другом потоке, то работа второго потока будет приостановлена, пока первый поток на завершит выполнение метода transfer.

В общем случае отмечать ключевым словом synchronized необходимо те методы, для выполнения которых требуется несколько операций чтения и обновления данных. В этом случае все операции будут завершены до того, как обращение к тому же самому объекту будет происходить из другого потока.

Если поток для работы с каким-то объектом вызывает синхронизированный метод, то этот объект блокируется (locked), т.е. запирается так, как если бы у него была дверь замком. Можно провести аналогию с телефонной будкой, которая запирается изнутри на защелку. Когда в потоке вызывается синхронизированный метод, он запирает дверь, т.е. блокирует объект. Когда другой поток пытается вызвать синхронизированный метод для этого же объекта, он не может открыть дверь и прекращает выполнение. После завершения работы первого синхронизированного метода дверь отпирается, т.е. объект разблокируется. Планировщик потоков периодически активизирует потоки, ожидающие снятия блокировок, используя упомянутые выше обычные правила. Всякий раз, когда поток хочет вызвать синхронизированный метод для данного объекта, он проверяет наличие у него блокировки. Если блокировки нет, то поток становится следующим потоком, который может получить эксклюзивный доступ к объекту. Однако, другие потоки вполне могут вызывать не синхронизированные методы для этого же объекта.

Если поток прекращает выполнение синхронизированного метода в результате возникновения исключительной ситуации, то блокировка объекта снимается во избежание нежелательной бесконечной блокировки.

Если поток наложил блокировку на объект и вызывает для него другой синхронизированный метод, то последнему методу право доступа предоставляется автоматически. При этом блокировка будет снята с объекта лишь после завершения работы последнего синхронизированного метода.

Каждый объект имеет счетчик блокировок (lock count), в котором указано количество синхронизированных методов, вызванных потоком-владельцем блокировок. При каждом таком вызове счетчик увеличивается на единицу, а при завершении работы синхронизированного метода (нормальным образом или в результате не перехваченной исключительной ситуации) счетчик уменьшается. Когда значение счетчика становится равным нулю, поток разблокирует объект.

Следует обратить внимание, что могут существовать два различных объекта одного класса, каждый из которых может быть заблокирован своим потоком. Причем оба потока могут запустить один и тот же синхронизированный метод. Учтите, что блокируется объект, а не метод! По аналогии с телефонными будками два человека могут находиться в двух отдельных будках. Каждый из них может запустить один и тот же синхронизированный метод или разные методы.

Конечно, блокировкой объекта в заданный момент времени владеет только один поток. В то же время поток может наложить блокировку на несколько объектов, если для них вызываются синхронизированные методы. (Следует отметить, что здесь аналогия с телефонными будками уже не подходит.)

## Задача синхронизации

Синхронизация включает согласование поведения процессов и зависимость выполнения одного процесса от событий, которые могут возникать в другом процессе. Для решения этой задачи используются разные методы, такие как семафоры, события, мониторы.

Существует несколько схем синхронизации процессов:

* один процесс ожидает наступление события в другом процессе;
* несколько процессов ожидают наступления события в другом процессе;
* один процесс ожидает наступления нескольких событий в разных процессах.

## Семафоры

#### Общие сведения

Семафо́р — объект, позволяющий войти в заданный участок кода не более чем n [потокам](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA). Определение введено [Э. Дейкстрой](http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0).

[Семафоры](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) представляют собой доступные ресурсы, которые могут быть приобретены несколькими потоками в одно и то же время, пока пул ресурсов не опустеет. Тогда дополнительные потоки должны ждать, пока требуемое количество ресурсов не будет снова доступно. Семафоры очень эффективны, поскольку они позволяют одновременный доступ к ресурсам.

Семафор — это объект, с которым можно выполнить три операции.

* init(S, n) – создать семафор S и установить его начальное значение в n: S := n.
* P(S) – ждать, пока счётчик S будет больше 0; после этого уменьшить счётчик на единицу и продолжить выполнение.
* V(S) – увеличить счётчик S на единицу и продолжить выполнение.

Использование семафоров может решить как задачу взаимного исключения (с помощью бинарного семафора), так и задачу синхронизации.

#### Решение в языке Ada

Во втором стандарте языка Ada95 механизм семафоров представлен в виде пакета Synchronous\_Task\_Control. Пакет реализует механизм семафоров следующим образом. Семафорный тип обеспечивается приватным типом Suspension\_Object, операции P(S) и V(S) реализованы с помощью процедур Suspend\_Until\_True() и Set\_True(). Используется бинарный логический семафор, т.е. семафорные переменные принимают значения false и true.

Кроме описанных выше процедур, в пакете реализованы вспомогательные процедудры Set\_False() для установления значения семафора в false и Current\_State() для считывания текущего значения семафора.

#### Решение в языке Java

Непосредственного решения нет.

## Механизм рандеву в Ada

Как один из механизмов обеспечения надежного межзадачного обмена данными и взаимной синхронизации работы задач, Ада предоставляет механизм рандеву. Основополагающая идея механизма рандеву достаточно проста. В спецификации задачи публикуются различные входы (entry) в задачу, в которых она готова ожидать обращения к ней от других задач. Далее, в теле задачи указываются инструкции принятия обращений к соответствующим входам, указанным в спецификации этой задачи.

Необходимо обратить внимание на несимметричность такого механизма взаимодействия. Это означает, что в процессе взаимодействия одна из задач рассматривается как сервер, а вторая - как клиент, причем задача-сервер не может быть инициатором начала взаимодействия.

В простейшем случае, когда рассматривается взаимодействие только двух задач, задача-клиент, желающая обратиться к другой задаче (задаче-серверу), инициирует обращение к входу задачи-сервера. После этого задача-сервер откликается на обращение задачи-клиента, принимая обращение к этому входу. Таким образом, взаимодействие двух задач осуществляется в ситуации, когда задача-клиент обратилась к входу, а задача-сервер готова принять это обращение. Этот способ взаимодействия двух задач называется рандеву.

Поскольку задача-клиент и задача-сервер выполняются независимо друг от друга, то нет никакой гарантии, что обе задачи окажутся в точке осуществления рандеву одновременно. Поэтому, если задача-сервер оказалась в точке рандеву, но при этом нет ни одного обращения к входу (запроса на взаимодействие), то она должна ждать появления такого обращения. Аналогично, если задача-клиент обращается к входу, а задача-сервер не готова обслужить такое обращение, то задача-клиент должна ждать, пока задача-сервер обслужит это обращение. В процессе ожидания как задача-клиент, так и задача-сервер не занимают ресурсы процессора, находясь в состоянии, которое называют приостановленным или состоянием блокировки.

В случаях, когда вызовы к входу задачи-сервера осуществляют сразу несколько задач-клиентов, эти вызовы ставятся в очередь. Порядок обслуживания такой очереди зависит от соответствия конкретной реализации Ада-системы требованиям приложения D (Annex D) стандарта Ada95, в котором указываются требования для систем реального времени. Если реализация Ада-системы не обеспечивает соответствия этим требованиям, то очередь обслуживается в порядке поступления вызовов (FIFO - First-In-First-Out).

## Средства управления потоками

### Задачный тип Ada

В качестве стандартных средств поддержки многозадачности Ады используются задачи (tasks), которые хорошо известны со времен стандарта Ada83 и описывают последовательности действий, которые способны выполняться одновременно.

Программа на языке Ада состоит как минимум из одной, а возможно и множества задач, выполняющихся одновременно. Каждая задача выполняется независимо от остальных задач. Механизмы межзадачного обмена данными и синхронизации основаны на высокоуровневых концепциях рандеву и использовании защищенных объектов. Следует заметить, что рандеву и защищенные объекты обладают более высоким уровнем абстракции по сравнению с семафорами. Они предоставляют средства защитной блокировки и таймаутов, а также средства для выполнения выборочного перестроения очередей клиентов и аварийного завершения. Конструкция задачи обладает свойствами, которые характерны для пакетов, процедур и структур данных:

Подобно пакетам, задача имеет спецификацию и тело, однако она не может быть самостоятельно компилируемой единицей, помещенной в свой собственный файл с исходным текстом. Вместо этого, задача должна быть помещена в другую структуру (например, пакет или процедуру)

Подобно процедуре, задача содержит раздел описаний и исполнительную часть, однако она не вызывается как процедура. Вместо этого, она начинает выполняться автоматически, как часть структуры, в которой она описана.

Подобно структуре данных, задача имеет тип и может существовать как переменная этого типа. Кроме того, подобно записи, задача может иметь дискриминанты.

Спецификация задачи, начинающаяся зарезервированными словами task type, определяет тип задачи (или задачный тип). Значение объекта (переменная) типа задачи представляет собой задачу. Спецификация задачи, не содержащая зарезервированного слова type, определяет одиночную задачу, а описание задачи с такой спецификацией равносильно описанию анонимного типа задачи с одновременным описанием объекта этого типа.

Простым примером многозадачной программы может служить следующая программа:

**procedure** Multitasking\_Demo **is**

-- спецификация анонимной задачи

**task** Anonimous\_Task;

-- тело анонимной задачи

**task body** Anonimous\_Task **is**

**begin**

…

**end** Anonimous\_Task;

-- спецификация типа задачи

**task type** Simple\_Task (Message: Character);

**task body** Simple\_Task **is**

**begin** -- для Simple\_Task

Ada.Text\_IO.Put\_Line("Hello from Simple\_Task " & Message);

**end** Simple\_Task;

-- переменная задачного типа

Simple\_Task\_Variable: Simple\_Task(Message => 'A');

**begin**

-- в отличие от процедур, задачи не вызываются,

-- а активируются автоматически

**null**;

**end** Multitasking\_Demo;

В данном примере описана одиночная задача анонимного типа Anonimous\_Task, тип задачи Simple\_Task и переменная задачи Simple\_Task\_Variable, имеющая тип Simple\_Task. Примечательно, что описание типа задачи Simple\_Task содержит дискриминант, значение которого используется как параметр задачи и указывается при описании переменной задачи Simple\_Task\_Variable. Следует также обратить внимание на то, что любой тип задачи является лимитированным и, таким образом, для него не существует предопределенных операций присваивания или сравнения. Алгоритмы работы обоих задач просты и подобны - каждая задача выводит пять приветственных сообщений и завершает свою работу.

### Потоки в Java

Все классовые библиотеки Java написаны с учетом особенностей программирования в условиях параллельного выполнения потоков.

Поток выполнения в Java представляется экземпляром класса **Thread**. Для того, чтобы написать свой поток исполнения, необходимо наследоваться от этого класса и переопределить метод run(). Например:

public class MyThread extends Thread {

public void run() {

// некоторое долгое действие, вычисление

long sum=0;

for (int i=0; i<1000; i++) {

sum+=i;

}

System.out.println(sum);

}

}

Метод run() содержит действия, которые должны выполняться в новом потоке исполнения. Чтобы запустить его, необходимо создать экземпляр класса-наследника и вызвать унаследованный метод start(), который сообщает виртуальной машине, что требуется запустить новый поток исполнения и начать выполнять в нем метод run().

MyThread t = new MyThread();

t.start();

В результате чего на консоли появится результат:

499500

Когда метод run() завершен (в частности, встретилось выражение return), поток выполнения останавливается. Однако ничто не препятствует записи бесконечного цикла в этом методе. В результате поток не прервет своего исполнения и будет остановлен только при завершении работы всего приложения.

Описанный подход имеет один недостаток. Поскольку в Java множественное наследование отсутствует, требование наследоваться от Thread может привести к конфликту. Если еще раз посмотреть на приведенный выше пример, станет понятно, что наследование производилось только с целью переопределения метода run(). Поэтому предлагается более простой способ создать свой поток исполнения. Достаточно реализовать интерфейс **Runnable**, в котором объявлен только один метод – уже знакомый void run(). Запишем пример, приведенный выше, с помощью этого интерфейса:

public class MyRunnable implements Runnable {

public void run() {

// некоторое долгое действие, вычисление

long sum=0;

for (int i=0; i<1000; i++) {

sum+=i;

}

System.out.println(sum);

}

}

Также незначительно меняется процедура запуска потока:

Runnable r = new MyRunnable();

Thread t = new Thread(r);

t.start();

Если раньше объект, представляющий сам поток выполнения, и объект с методом run(), реализующим необходимую функциональность, были объединены в одном экземпляре класса MyThread, то теперь они разделены. Какой из двух подходов удобней, решается в каждом конкретном случае.

Подчеркнем, что Runnable не является полной заменой классу Thread, поскольку создание и запуск самого потока исполнения возможно только через метод Thread.start().

Поток можно превратить в потоковый демон (daemon thread) с помощью вызова метода setDaemon:

t.setDaemon(true);

Демон представляет собой поток, единственным предназначением которого является обслуживание других потоков. Примером может служить поток таймера, который через определенные интервалы времени посылает сигналы другим потокам. Если в данной программе запущенными остаются только потоковые демоны, то такая программа завершает работу (так как продолжать ее просто не имеет смысла).

### 1.9 Приоритеты

#### Решение в языке Ada

Каждая задача Ады может обладать своим собственным приоритетом выполнения, который задается с помощью директивы компилятора Priority:

**pragma** Priority ( expression );

Непосредственное использование этой директивы компилятора допускается:

* внутри спецификации задачи,
* внутри спецификации защищенного типа или объекта,
* в описательной части тела подпрограммы

Значение результата выражения expression, используемого для непосредственного указания приоритета, должно принадлежать целочисленному типу Integer, причем при указании директивы Priority в описательной части тела подпрограммы выражение expression должно быть статическим, а его значение должно принадлежать диапазону значений подтипа Priority, который описан в пакете System. Например:

task Some\_Task is

pragma Priority (5);

. . .

end Some\_Task;

Приоритет выполнения задачи определяет ее привилегии на обладание системными ресурсами (например, процессором). В простейшем случае, если две задачи с разными приоритетами готовы к выполнению, то к выполнению будет допущена та задача, приоритет которой выше. При равенстве приоритетов порядок выполнения задач не определен.

В Ada83 приоритет задачи строго фиксировался при ее описании. Согласно стандарту Ada95, приоритет задачи может быть изменен в процессе существования задачи (кроме приоритета, указанного для подпрограммы), то есть задачи могут иметь динамически изменяемые приоритеты.

Средства динамического определения и изменения текущего приоритета задачи предоставляются предопределенным стандартным пакетом Ada.Dynamic\_Priorities. Спецификация этого пакета проста и имеет следующий вид:

with System;

with Ada.Task\_Identification;

package Ada.Dynamic\_Priorities is

procedure Set\_Priority

(Priority : in System.Any\_Priority;

T : in Ada.Task\_Identification.Task\_ID :=

Ada.Task\_Identification.Current\_Task);

function Get\_Priority

(T : Ada.Task\_Identification.Task\_ID :=

Ada.Task\_Identification.Current\_Task)

return System.Any\_Priority;

end Ada.Dynamic\_Priorities;

Следует заметить, что правила планирования выполнения задач на основе статических и динамических приоритетов рассматриваются в приложении D (Annex D) стандарта Ada95, в котором указываются требования для систем реального времени.

#### Решение в языке Java

В языке программирования Java каждый поток имеет приоритет (priority), который по-умолчанию наследуется как приоритет родительского потока. Для повышения или понижения приоритета потока используется метод setPriority. Возможные значения приоритета находятся в диапазоне от минимального значения MIN\_PRIORITY (1 в классе Thread) до максимального значения MAX\_PRIORITY (10 в классе Thread). Обычный приоритет NORM\_PRIORITY, который используется по-умолчанию, имеет значение 5.

В общем случае планировщик выполнения потоков выбирает поток с наивысшим приоритетом среди запущенных в данный момент. Прежде всего, следует отметить, что правила обработки приоритетов потоков сильно зависят от используемой системы. Если виртуальная машина Java полагается на реализацию потоков в используемой системе, то расписание выполнения потоков будет целиком и полностью зависеть от этой реализации. Виртуальная машина Java устанавливает соответствие между приоритетами потоков и приоритетами процессов в операционной системе (в которой может быть больше или меньше уровней приоритетов).

Планировщик выполнения потоков выбирает в качестве выполняемого потока, тот поток, который обладает наивысшим приоритетом среди оставшихся запущенных потоков.

В случае, если несколько потоков имеют одинаковый (высший) приоритет, по усмотрению планировщика выполнения потоков выбирается какой-то один из них. В языке программирования Java не гарантируется равноправный подход к обработке всех потоков с одинаковым приоритетом со стороны планировщика выполнения потоков.

Теоретически, на некоторых платформах возможна ситуация, когда планировщик выбирает поток случайно или же обрабатывает первый доступный из них. Такое свойство является недостатком языка Java и затрудняет создание многопоточных программ, которые могли бы выполнятся строго одинаково на всех виртуальных машинах.

На некоторых платформах (например, Windows NT) предусмотрено меньше уровней приоритета, чем в Java. В таких случаях независимо от того, какое соответствие уровней выбирается, некоторые из 10 уровней JVM будут отвечать одному и тому же уровню приоритета используемой платформы. В Sun JVM для Linux приоритеты потоков игнорируются.

Если в операционной системе предусмотрено меньше уровней приоритета, чем в Java, то поток с более высоким приоритетом может оказаться вытесненным потоком с меньшим приоритетом. Это означает, что в многопоточных программах не рекомендуется полагаться на уровни приоритета.

Рассмотрим, например, вызов метода yield . В некоторых реализациях он может не дать никакого эффекта, если уровни приоритета всех запущенных потоков соответствуют одному и тому же уровню приоритета в данной операционной системе.

Динамическое изменение приоритета потока в Java осуществляется следующим образом:

setPriority(Thread.MIN\_PRIORITY);

## Приостановка и зварешение потоков

#### Решение в языке Ada

Инструкции задержки выполнения могут быть использованы для приостановки выполнения тела задачи (или программы) на некоторое время. Рзличают два вида инструкций задержки выполнения: относительная задержка выполнения и абсолютная задержка выполнения.

Общий вид инструкции относительной задержки выполнения следующий:

**delay** время\_задержки;

Здесь результат выражения время\_задержки, имеющего предопределенный вещественный тип с фиксированной точкой Duration (описан в пакете Standard), указывает длительность задержки выполнения в секундах, на которую необходимо задержать выполнения задачи (или программы). При этом отсчет времени задержки выполняется относительно текущего момента времени. Наглядным примером использования инструкции относительной задержки, для задержки выполнения задачи на одну секунду, может служить следующее:

delay 1.0;

Общий вид инструкции абсолютной задержки выполнения следующий:

**delay until** время\_задержки;

В этом случае результат выражения время\_задержки, любого неограниченного типа, указывает момент времени, до наступления которого необходимо задержать выполнение задачи (или программы). Следующий пример демонстрирует использование инструкции абсолютной задержки выполнения задачи:

**delay until** Time\_Of(2010, 1, 1, 0.0); -- задержка выполнения до 1 января 2010 года

Следует также заметить, что описание предопределенного типа времени Time и ассоциируемых с ним операций предусматривается в стандартном пакете Ada.Calendar. Кроме того, пакет Ada.Calendar предоставляет функцию Clock, которая возвращает значение текущего момента времени (для получения более полной информации следует обратиться к спецификации этого пакета).

Отметим также, что при построении систем, которые должны работать в реальном масштабе времени, вместо типа времени Time, описанного в стандартном пакете Ada.Calendar, следует использовать тип времени Time, который описывается в пакете Ada.Real\_Time (для получения более полной информации следует обратиться к спецификации пакета Ada.Real\_Time).

Ада позволяет принудительно завершать выполнения объекта задачи. Это может быть выполнено с помощью инструкции прекращения, которая может иметь следующий вид:

**abort** Some\_Task\_Name;

Здесь Some\_Task\_Name - это имя какого-либо объекта задачи. Считается, что принудительно прекращенная задача находится в "ненормальном" (abnormal) состоянии и не может взаимодействовать с другими задачами. После того, как состояние задачи отмечено как "ненормальное", выполнение ее тела прекращается. Это подразумевает, что прекращается выполнение любых инструкций, расположенных в теле задачи, за исключением тех, которые вызывают операции, отложенные до принудительного прекращения (abort-deffered operations).

Следует заметить, что использование принудительного прекращения выполнения задачи является "аварийным" действием, и должно применяться только в тех случаях, когда принудительное прекращение выполнения задачи действительно необходимо (например, когда задача "зависла").

Следует понимать, что использование такого "сильнодействующего" средства остановки для задач, которые выполняют сохранение каких-либо дисковых данных, может повлечь за собой не только потерю самих данных, но и повреждение логической структуры дискового накопителя.

#### Решение в языке Java

Каждый объект в Java имеет не только блокировку для synchronized блоков и методов, но и так называемый wait-set, набор потоков исполнения. Любой поток может вызвать метод wait() любого объекта и таким образом попасть в его wait-set. При этом выполнение такого потока приостанавливается до тех пор, пока другой поток не вызовет у этого же объекта метод notifyAll(), который пробуждает все потоки из wait-set. Метод notify() пробуждает один случайно выбранный поток из данного набора.

Однако применение этих методов связано с одним важным ограничением. Любой из них может быть вызван потоком у объекта только после установления блокировки на этот объект. То есть либо внутри synchronized-блока с ссылкой на этот объект в качестве аргумента, либо обращения к методам должны быть в синхронизированных методах класса самого объекта. Рассмотрим пример:

public class WaitThread implements Runnable {

private Object shared;

public WaitThread(Object o) {

shared=o;

}

public void run() {

synchronized (shared) {

try {

shared.wait();

} catch (InterruptedException e) {}

System.out.println("after wait");

}

}

public static void main(String s[]) {

Object o = new Object();

WaitThread w = new WaitThread(o);

new Thread(w).start();

try {

Thread.sleep(100);

} catch (InterruptedException e) {}

System.out.println("before notify");

synchronized (o) {

o.notifyAll();

}

}

}

Результатом программы будет:

before notify

after wait

Следует обратить внимание, что метод wait(), как и sleep(), требует обработки InterruptedException, то есть его выполнение также можно прервать методом interrupt().

Поток становится **блокированным** (blocked) после выполнения одного из следующих действий.

1. Для данного потока вызван метод sleep().

2. Поток вызывает операцию блокирования на время ввода/вывода, т.е. управление не возвращается до тех пор, пока не будут завершены операции ввода/вывода.

3. Поток вызывает метод wait().

4. Поток пытается заблокировать объект, который уже блокирован другим потоком.

5. Для данного потока вызван метод suspend().Не рекомендуется использовать этот метод.

Для выхода из блокированного состояния для потока следует выполнить действия, противоположные тем, которые привели его в это состояние.

1. Если запущен метод sleep, то поток продолжит выполнение по истечении заданной паузы.

2. Если поток ожидает завершения операций ввода/вывода, то он продолжит выполнение по окончании ввода/вывода.

3. Если в потоке был вызван метод wait(), то в другом потоке должен быть вызван метод notifyAll() или notify().

4. Если поток ожидает разблокировки объекта, заблокированного другим потоком, то последний должен передать права владения блокировкой этого объекта.

5. Если поток был приостановлен методом suspend(), то должен быть вызван метод resume(). Но, так как метод suspend() не рекомендуется использовать, то точно также не рекомендуется использовать и метод resume().

Блокированный поток можно вернуть в выполняемое состояние только по тому же пути, по которому он был блокирован. Например, если поток был блокирован на время выполнения операции ввода, то вызов метода resume() не позволит разблокировать его.

Поток может находиться в **остановленном** (dead) состоянии по следующим двум причинам.

* метод run() нормально завершил работу;
* неперехваченная исключительная ситуация внезапно прекратила выполнение метода run().

Поток можно остановить, явно вызывая метод stop(), который генерирует исключительную ситуацию ThreadDeath и останавливает выполнение потока. Однако, метод stop() не рекомендуется использовать по причинам, которые рассматриваются ниже.

Для определения работоспособности потока (т.е. находится ли он в запущенном или блокированном состоянии) применяется метод isAlive(). Если поток запущен или заблокирован, то этот метод возвращает значение true, а если поток является созданным, но еще не запущенным или остановленным, то метод возвращает значение false.

К сожалению, для работоспособного потока нельзя определить, является ли он запущенным или все-таки блокированным, а также, является ли запущенный поток выполняемым. Кроме того, нельзя различить еще не запущенный поток и уже прекращенный поток.

## Висновки до Розділу 1

1. Были рассмотрены теоретические основы многопоточности, и параллельных вычислительных систем. Были произведёны обзоры средств и возможностей, которыми обладают языки программирования Ada и Java, призванных решить подобного рода проблемы.

2. Оба языка программирования – и Ada, и Java – обладают встроенной поддержкой многопочности. Это означает, что оба языка включают широкий и исчерпывающий набор инструментов для проектирования многопоточных приложений. Данный факт в значительной мере упрощает создание программ для параллельных вычислительных систем.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРАЛЕЛЬНИХ КОМП’ЮТЕРНИХ СИСТЕМ

## 2.1 Розробка паралельного математичного алгоритму

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратно P, що дозволяє розбити задачу на окремі шматочки Н, кожен з яких обробляється своїм завданням. .

Розрахунок даного матричного рівняння можливо провести у один етап:

A = sort (CH \* α + D \* (MO \* MTH))

## Аналіз завдання

### Оцінка розробленого паралельного алгоритму в рамках концепції необмеженого паралелізму

Оцінити паралельний алгоритм зручніше всього трьома етапами:

1. MA = MO \* MT.
2. A = C \* α + D \* MA
3. А = sort(A)

На рис. 2.1 зображена ​​ярусно-паралельна форма алгоритму обчислення одного елемента матричного рівняння MA = MO \* MT.



Рис. 2.1. Ярусно-паралельна форма обчислення матриці MA = MO \* MT.

Кількість ярусів для обчислення одного елемента матриці рівне log2N+1. Всю матрицю обчислювати паралельно неможливо. Для подальшого обчислення потрібно сформувати стовпець матриці. Кількість ярусів у даному випадку буде дорівнювати N(log2N + 1).

На рис. 2.2 зображена ярусно-паралельна форма алгоритму обчислення одного елемента матричного рівняння A = C \* α + D \* MA з урахуванням ярусів

обчислення стовпця МA.



Рис. 2.2. Ярусно-параллельная форма вычисления вектора A = С \* α + D \* MA

Як видно, для проходу по N елементів досить N(log2N+2)+N2(log2N+1) ярусів. Для закінчення визначення кількості ярусів потрібно відсортувати отриманий вектор. На рис.2.3 виконуєтся обчислення рівняння А = sort(A), де sort – звичайне бульбашкове сортування, а sort\* - сортування методом злиття.

 Рис. 2.3. Ярусно-параллельная форма вычисления вектора A = sort(A)

Вважаємо, що всі операції виконуються за одиницю часу, тоді має місце наступну рівність кількості ярусів і кількості операцій log2N + 1 = 2N - 1. Таким чином, використовуючи відому кількість ярусів можна дізнатися основні параметри та характеристики даного матричного рівняння.

**T1=2N3+N2+2N-2** - час обчислення завдання на 1-процесорної машині.

**TN=log2N+N(log2N+2)+N2(log2N+1)** - час обчислення завдання на машині з кількістю процесорів N.

**КП =** **2N3+N2+2N-2/ log2N+N(log2N+2)+N2(log2N+1)** – коефіцієнт прискорення

**КЕ** = **2N3+N2+2N-2/ Nlog2N+N2(log2N+2)+N3(log2N+1)** – коефіцієнт еффективності.

Проаналізувавши ці вирази, робимо висновок, що при збільшенні кількості процесорів КП зростає, проте КЕ зменшується.

Оцінимо час виконання завдання.

Згідно з концепцією необмеженого паралелізму час на синхронізацію не враховується. Вважаємо, що всі операції виконуються за одиницю часу. Заданий математичне вираз на однопроцесорній обчислювальної системі буде вважатися за час **T1= 2N3+N2+2N-2.**

При P = р **Tp = log2p+( 2N3+N2+2N-2 +1-2]log2p[)/p**

При P => ∞ **Tp = log2(2N3+N2+2N-2 +1)**

Провівши дослідження аналітичним способом ми отримали результати досліджень, на підставі яких можна зробити висновок про те, що при використанні програми розробленої для паралельної обчислювальної системи з Р процесорами ми отримуємо значну перевагу в швидкості обчислення.

## Розробка ПЗ для системи з спільною пам’яттю

### Розробка алгоритмів процесів

**№ Т1 ТС/КУ**

1. Введення C, α
2. Сигнал задачам Т2, Т3, Т4 о закінченні введення S2.1, S3.1, S4.1
3. Чекати введення в T2, T3 W2.1, W3.1
4. Копія MO1 := MO, α1 := α, D1 := D КУ
5. Обчислення AH = sort(СH \* α1 – D1 \* (MO1 \* MTH)
6. Чекати закінчення обчислення AH в задачі Т2 W1.2
7. Злиття A2H = sort(AH, AH)
8. Чекати закінчення обчислення A2Н в задачі Т4 W4.3
9. Злиття A = sort(A2Н, A2Н)
10. Виведення А

**№ Т2 ТС/КУ**

1. Введення MT, D
2. Сигнал задачам Т1, Т3, Т4 о закінченні введення S1.1, S3.1, S4.1
3. Чекати введення в T1, T3 W1.1, W3.1
4. Копія MO2 := MO, α2 := α, D2 := D КУ
5. Обчислення AH = sort(СH \* α2 – D2 \* (MO2 \* MTH)
6. Сигнал задачі Т1 о закінченні обчислення A2H

**№ Т3 ТС/КУ**

1. Введення МО
2. Сигнал задачам Т1, Т2, Т4 о закінченні введення S1.1, S2.1, S4.1
3. Чекати введення в Т1, T2 W1.1, W2.1
4. Копія MO4 := MO, α4 := α, D4 := D КУ
5. Обчислення AH = sort(СH \* α2 – D2 \* (MO2 \* MTH)
6. Сигнал задачі Т4 о закінченні обчислення AH S4.2

**№ Т4 ТС/КУ**

1. Чекати введення в Т1, T2, T3 W1.1, W2.1, W3.1
2. Копія MO4 := MO, α4 := α, D4 := D КУ
3. Обчислення AH = sort(СH \* α4 – D4 \* (MO4 \* MTH)
4. Чекати закінчення обчислення AH в задачі Т3 W3.2
5. Злиття A2H = sort(AH, AH)
6. Сигнал задачі Т1 о закінченні обчислення A2H S1.1

## Розробка схеми взаємодії задач

## Схема взаємодії задач відображена на рис. 2.4.

### Розробка програми

Для реалізації алгоритму обчислення матричного рівняння на обчислювальній системі зі спільною пам’яттю було використано мову Ada та механізм захищених модулів

Лістинг програми наведено у Додатку Ж.

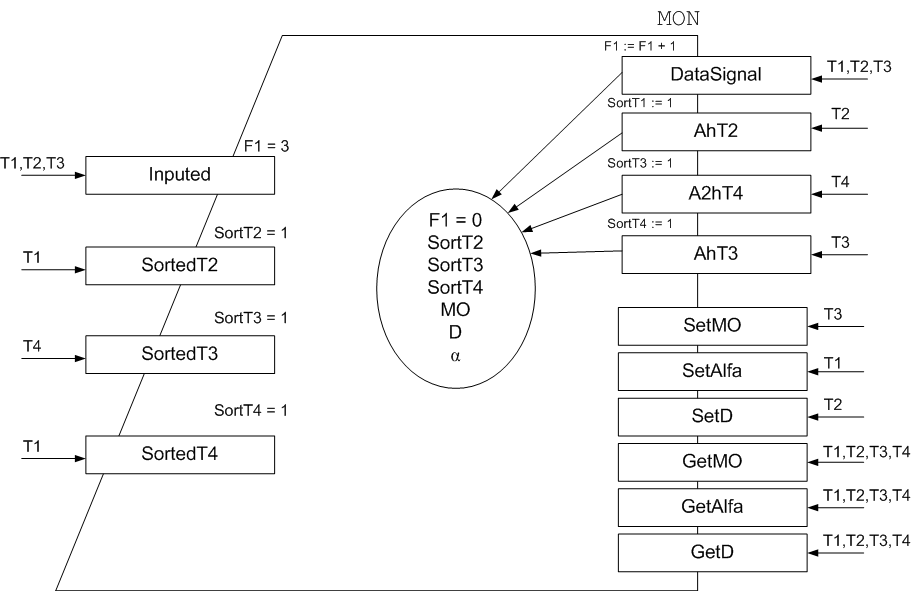


Рис. 2.4. Схема взаємодії задач для системи зі спільною пам’яттю

## Розробка ПЗ для системи з локальною пам’яттю

### Розробка алгоритмів процесів

**№ Алгоритм задачи Т1**

1. Введення C, α
2. Передати CH, α задачі Т4
3. Передати C2H, α задачі Т2
4. Прийняти D, MTH від задачі Т2
5. Прийняти МО від задачі Т2
6. Обчислення АН = sort(CH \* α + D \* (MO \* MTH))
7. Прийняти АН від задачі Т2
8. Злиття A2H = sort(AH, AH)
9. Прийняти А2H от задачі Т4
10. Злиття A = sort(А2H, А2H)
11. Виведення А

**№ Алгоритм задачи Т2**

1. Введення D, MT
2. Прийняти C2H, α від задачі Т1
3. Передати CH, α задачі Т3
4. Передати D, MTH задачі Т1
5. Передати D, MT2H задачі Т3
6. Прийняти МО от задачі Т3
7. Передати МО задачі Т1
8. Обчислення АН = sort(CH \* α + D \* (MO \* MTH))
9. Передати АН задачі Т1

**№ Алгоритм задачи Т3**

1. Введення MО
2. Прийняти CH, α від задачі Т2
3. Прийняти D, MT2H від задачі Т2
4. Передати МО задачі Т2
5. Передати МО задачі Т4
6. Передати D, MTH задачі Т4
7. Обчислення АН = sort(CH \* α + D \* (MO \* MTH))
8. Передать АН задачі Т4

**№ Алгоритм задачи Т4**

1. Прийняти CH, α від задачі Т1
2. Прийняти МО від задачі Т3
3. Прийняти D, MTH від задачі Т3
4. Обчислення АН = sort(CH \* α + D \* (MO \* MTH))
5. Прийняти АН від задачі Т3
6. Злиття A2H = sort(AH, AH)
7. Передати A2H задачі Т1

### Розробка програми

Для реалізації алгоритму обчислення векторно-матричного рівняння на обчислювальній системі з локальною пам’яттю було використано мову Ada та механізм рандеву. Рандеву – дуже потужний механізм, що використовує підхід пересилки повідомлень.

Лістинг програми наведено у Додатку З.

### Розробка схеми взаємодії задач

## Схема взаємодії задач відображена на рис. 2.5.

## Висновки до Розділу 2

1. Мова Ada має дуже зручні інструменти організації паралельних обчислень. Механізм захищених модулів рішає задачі взаємного виключення та синхронизації використовуючи захищені процедури, функції для повернення загального ресусру задачам та захищені входи.

2. При написанні програми для системи з локальною пам’яттю велика вірогідність зробити помилку у механізмі передачі повідомлень. Треба чітко уявляти собі весь алгоритм передачі повідомлень на кожній ітерації, та моделювати передачу перед розробкою програмного забезпечення для таких систем.

3. Порівнюючи процес розробки для системи зі спільною пам’яттю та системи з локальною пам’яттю можемо зробити висновок, що варіант зі спільною пам’яттю є більш простим у розробці та реалізації. Це зумовлено у першу чергу більш простим механізмом взаємодії потоків.



# РОЗДІЛ 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**3.1 Методика тестування.**

Для замірів часу виконання розробленного програмного забезпечення використовувались функції зі стандартних бібліотек мов, що були використані (Ada.Calendar(20) для Ada). Програмне забезпечення приймало різне значення обсягів даних та запускалось на різних кількостях ядер. Тестування на різних кількостях ядер виконувалось за допомогою виключення ядер з діспетчера задач Windows (Ctrl+Alt+Del).

Основні характеристики розробленного програмного забезпечення визначаются за допомогою формул.

Коефіцієнт прискорення (19):

Коефіцієнт ефективності (19):

## 3.2 Тестування ПЗ для системи з спільною пам’яттю

Таблиця 3.1. Час виконання програми обчислення на системі зі СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 20 c | 11 c | 7 c | 6 c |
| 1800 | 101 c | 51 c | 35 c | 26 c |
| 2700 | 434 c | 217 c | 148 c | 110 c |

Таблиця 3.2. Значення коефіцієнтів прискорення для системи с СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 1,8181 | 2,9324 | 3,3333 |
| 1500 | 1 | 1,9908 | 2,8857 | 3, 9454 |
| 2400 | 1 | 1,9803 | 2,8571 | 3, 8846 |

Таблиця 3.3 Значення коефіцієнтів ефективності для системи с СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 0,9774 | 0,9090 | 0,8333 |
| 1500 | 1 | 0,9954 | 0,9863 | 0,9619 |
| 2400 | 1 | 0,9901 | 0,9711 | 0,9524 |

**Рис. 3.1. Залежність коефіцієнту прискорення від кількості ЦП для систем з спільною пам’яттю

****

Рис. 3.2**.** Залежність коефіцієнту ефективності від кількості ЦП для систем з спільною пам’яттю

## Тестування ПЗ для системи з локальною пам’яттю

Таблиця 3.4. Час виконання програми обчислення на системі зі ЛП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 19 c | 10 c | 8 c | 5 c |
| 1500 | 92 c | 47 c | 32 c | 24 c |
| 2400 | 386 c | 221 c | 131 c | 97 c |

Таблиця 3.5. Значення коефіцієнтів прискорення для системи с ЛП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 1,9000 | 2,3750 | 3,8000 |
| 1500 | 1 | 1,9574 | 2,8750 | 3,8333 |
| 2400 | 1 | 1,7466 | 2,9465 | 3,9793 |

Таблиця 3.6. Значення коефіцієнтів ефективності для системи с СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 0,9500 | 0,7916 | 0,9500 |
| 1500 | 1 | 0,9787 | 0,9583 | 0,9583 |
| 2400 | 1 | 0,9948 | 0,9821 | 0,8733 |



Рис. 3.3. Залежність коефіцієнту прискорення від кількості ЦП для систем з локальною пам’яттю

****

Рис. 3.4. Залежність коефіцієнту ефективності від кількості ЦП для систем з локальною пам’яттю

## Висновки до Розділу 3

1. Аналізуючи графіки робіт програмного запезпечення для системи з локальною пам’яттю можна сказати, що коефіцієт присткорення зростає лінійно. Як видно на графіку, коефіцієнт прискорення при малих обсягах даних має менший коефіцієнт, ніж більші обсяги даних. Це пов’язано з тим, що використовуючи бібліотеку Calendar можна знаходити лише кількість секунд затрачених на використання, а для більш точного знаходженя часу використання краще використовувати мікросекунди. В цілому коефіцієнт прискорення веде себе доволі логічно, і обсяг оброблюваних данних не дуже впливає на зниження прискорення. Коефіцієнт ефективності також лінійно спадає, він оберненно пропорційний до графіку коефіцієнта прискорення

2. Коефіцієнт прискорення програмного забезпечення системи з локальною пам’яттю лінійно зростає зі збільшенням кількості процесорів так само як і в програмному забезпеченні для системи із спільною пам’яттю. При малих обсягах данних коефіцієнт ефективності веде себе не дуже логічно, це може бути викликано похибкою у вимірюванні часу (ми вимірювали секунди, а це таке вимірювання має більшу похибку ниж якби ми визначали мікросекунди роботи). В цілому коефіцієнт ефективності також лінійний і обернено-пропорційний до коефіцієнта прискорення.

3. Як видно по графікам, програмні забезпечення для обох систем мають потужну властивість до росту коефіцієнта прискорення і в той же самий час дуже стабільний коефіцієнт ефективності який зі збільшенням кількості ядер залишиться майже незмінним.

**Висновки**

1. Было рассмотрено понятие процесса и потока. Показано их основные различия и характеристики поведения внутри системы

2. В качестве общих черт в вопросах параллельного программирования у этих двух технологий можно выделить следующее: возможность динамического создания потоков (задач), наличие задачного типа, что предоставляет значительную гибкость в управлении потоками, возможность задания и динамического изменения приоритета задач, возможность гарантии атомарного доступа к переменным, наличие критических секций.

3. Показано, что язык Ada обладает большим, нежели Java, количеством средств, предназначенных для обеспечения взаимодействия потоков: помимо прочего, в Ada реализованы концепция рандеву (обмен сообщениями) и бинарные семафоры. Также следует уточнить, что подходы к программированию у этих языков совершенно разные. Каждый набор возможностей наиболее эффективен в своей среде, поэтому вопрос выбора технологии зависит от конкретной задачи.

4. Результирующее программное обеспечение спроектированое под системы с локальной и с общей памятью имеют хорошую степень масштабируемости в плане увеличения ядер и объёма данных под задачу. В целом програмное обеспечение имеет почти одинаковые характеристики как по скорости вычисления, так и в еффективности использования ядер.

# Література

1. Шилдт Г. Полный справочник по Java, 7-е издание.:Пер. С англ. – М.:ООО «И. Д. Вильямс», 2007. – 1040 с.
2. Шилдт Г., Ноутон П. Java 2. С-П.: «БХВ-Петербург», 2006. – 1072 с.
3. Ортега Дж. Введение в параллельные и векторные методы решения линейніх систем. – М.: Радио и связь, 1989. – 280 с.
4. Таненбаум Э., Современные операционные системы. 2-е изд. – СПб, 2005 – 1038с.
5. Гавва А. «Адское» программирование. Ada-95. Компилятор GNAT. 2004. – 431 c.
6. Справочное руководство по ада 83. [З мережі] http://ada-rm.euro.ru/
7. Немнюгин С., Стесик О. Паралаллеьное программирование для .

многопроцессорных вычислительных систем. – Спб: БХВ - Петербург,

2002. – 400 с.

8. Эндрюс Г. Основы многопоточного, параллельного и распределенного

программирования.: Пер. с англ. – М. Изд.дом «Вильямс», 2003.

9. Персональные блоги програмистов [З мережі] http://habrahabr.com

10. Parallel.ru [З мережі] [www.parallel.ru](http://www.parallel.ru)

11. Фонд Вікімедіа [З мережі] ru.wikipedia.org

12. RSDN [З мережі] [www.rsdn.ru](http://www.rsdn.ru)

13. MSDN [З мережі] www.msdn.com

14. Niemeyer P., Knudsen J. *Learning Java* O'Reilly, 2000

15. Кауфман В.Ш. Языки программирования. Концепции и принципы – М.: Радио и связь, 1993

16. Foster I. *Designing and Building Parallel Programs.*

17. Java HOME [З мережі] http://java.com/javafx

18. Ada HOME [З мережі] http://ada-ru.org

19. Жуков І., Корочікн О. Паралельні та розподілені обчислення. – Київ, Корнійчук, 2005. – 226 с.

20. Корочкин А.В. Ада 95: Введение в программирование. - Киев; Свит, 1998. - 260 с.

# Додатки

## Структурна схема ПКС з ОП

## Структурна схема ПКС з ЛП

## Алгоритм головної программи для ПКС з ОП

## Алгоритми процессів головної программи для ПКС з ОП

## Алгоритм головної программи для ПКС з ЛП

## Алгоритми процессів головної программи для ПКС з ЛП

## Лістінг программи для ПКС з ОП

Source file: ..\lab4.adb Mon May 16 19:40:36 2011

1 ----------------------------------------------------------------

2 -- Paralel and distributed computing --

3 -- Course Work --

4 -- --

5 -- IO-83 --

6 -- 27.04.2011 --

7 ----------------------------------------------------------------

8

9 with Package1; use Package1;

10 with Ada.Text\_IO; use Ada.Text\_IO;

11 with Ada.Integer\_Text\_IO; use Ada.Integer\_Text\_IO;

12 with Ada.Calendar; use Ada.Calendar;

13 with ada.float\_text\_io; use ada.float\_text\_io;

14

15 procedure Lab4 is

16 MT: Matr;

17 A: vec;

18 C: Vec;

19 Time\_all: duration;

20 time1,time2: time;

21 T: Integer;

22

23 -----------------------------------------------------------

24 ---------------ОПИСАТЕЛЬНАЯ ЧАСТЬ ЗАЩИЩЕННЫХ МОДУЛЕЙ-------

25 -----------------------------------------------------------

26

27 ----ЗАЩИЩЕННЫЙ МОДУЛЬ MON

28 protected MON is

29 --Синхронизация

30 procedure DataSignal;

31 procedure AhT2;

32 procedure A2hT4;

33 procedure AhT3;

34 --Общие ресурсы

35 procedure SetMO(M: in Matr);

36 procedure SetAlfa(M: in Integer);

37 procedure SetD(M: in Vec);

38 function GetMO return Matr;

39 function GetAlfa return Integer;

40 function GetD return Vec;

41 --Защищённые входы (синхронизация)

42 entry Inputed;

43 entry SortedT2;

44 entry SortedT3;

45 entry SortedT4;

46 --приватная часть защищенного модуля

47 private

48 --Синхронизация

49 F1: integer := 0;

50 F2: integer := 0;

51 SortT2: integer := 0;

52 SortT3: integer := 0;

53 SortT4: integer := 0;

54 --Общие ресурсы

55 MO: Matr;

56 D: Vec;

57 alfa: Integer;

58 end MON;

59

60 -----------------------------------------------------------

61 ------------------ТЕЛА ЗАЩИЩЕННЫХ МОДУЛЕЙ------------------

62 -----------------------------------------------------------

63

64 --ЗАЩИЩЕННЫЙ МОДУЛЬ Synchro

65 protected body MON is

66 -- REALIZE Синхронизация

67 procedure DataSignal is

68 begin

69 F1 := F1 + 1;

70 end DataSignal;

71

72 procedure AhT2 is

73 begin

74 SortT2 := 1;

75 end AhT2;

76 procedure A2hT4 is

77 begin

78 SortT4 := 1;

79 end A2hT4;

80 procedure AhT3 is

81 begin

82 SortT3 := 1;

83 end AhT3;

84 -- REALIZE Общие ресурсы

85 procedure SetMO(M: in Matr) is

86 begin

87 for i in 1 .. N loop

88 for j in 1 .. N loop

89 MO(i)(j) := M(i)(j);

90 end loop;

91 end loop;

92 end SetMO;

93 procedure SetAlfa(M: in Integer) is

94 begin

95 alfa := M;

96 end SetAlfa;

97 procedure SetD(M: in Vec) is

98 begin

99 for i in 1 .. N loop

100 D(i) := M(i);

101 end loop;

102 end SetD;

103 function GetMO return Matr is

104 begin

105 return MO;

106 end;

107 function GetAlfa return Integer is

108 begin

109 return alfa;

110 end;

111 function GetD return Vec is

112 begin

113 return D;

114 end;

115 --Защищённые входы (синхронизация)

116 entry Inputed when F1 = 3 is --1

117 begin

118 null;

119 end Inputed;

120

121 entry SortedT2 when SortT2 = 1 is --2

122 begin

123 null;

124 end SortedT2;

125 entry SortedT3 when SortT3 = 1 is --2

126 begin

127 null;

128 end SortedT3;

129 entry SortedT4 when SortT4 = 1 is --2

130 begin

131 null;

132 end SortedT4;

133 end MON;

134

135 procedure START is

136 -----------------------------------------------------------

137 ------------------------З А Д А Ч И------------------------

138 -----------------------------------------------------------

139 task T1;

140 task T2;

141 task T3;

142 task T4;

143

144 -----------------------ЗАДАЧА Т1 --------------------------

145 task body T1 is

146 Sum1, Temp: Integer;

147 MO1: Matr; alfa1: Integer; D1: Vec;

148 MTF: Matr;

149 Uk1, Uk2: Integer;

150 Tv: Vec;

151 begin

152 Put\_Line("T1 start");

153

154 --1. Ввод C, ?

155 Alfa1 := 1;

156 MON.SetAlfa(Alfa1);

157 Vec\_Input(C);

158 --2. Сигнал задачам Т2, Т3, Т4 о завершении ввода C, ?

159 MON.DataSignal;

160 --3. Ждать введения в T2, T3

161 MON.Inputed;

162 --4. Копирование MO1 := MO, ?1 := ?, D1 := D

163 MO1 := MON.GetMO;

164 D1 := MON.GetD;

165 --5. Cчёт AH = sort(СH \* ?1 - D1 \* (MO1 \* MTH)

166 for i in 1.. N loop

167 for j in 1 .. H loop

168 sum1 := 0;

169 for z in 1 .. N loop

170 sum1 := sum1 + MT(Z)(J) \* MO1(I)(Z);

171 end loop;

172 MTF(I)(J) := Sum1;

173 end loop;

174 end loop;

175 Sum1 := 0;

176 for i in 1.. H loop

177 for j in 1 .. N loop

178 sum1 := sum1 + D1(J) \* MTF(j)(I);

179 end loop;

180 A(I) := C(I) \* Alfa1 + Sum1;

181 Sum1 := 0;

182 end loop;

183 for Count1 in 1 .. H loop

184 for Count2 in 1 .. H loop

185 if (A(Count1) > A(Count2)) then

186 begin

187 Temp := A(Count1);

188 A(Count1) := A(Count2);

189 A(Count2) := Temp;

190 end;

191 end if;

192 end loop;

193 end loop;

194 --6. Ждать завершение счёта AH в задаче Т2 W1.2

195 MON.SortedT2;

196 --7. Слияние A2H = sort(AH, AH)

197 uk1 := 1;

198 uk2 := H+1;

199

200 for Count1 in 1 .. 2 \* H loop

201 if ((uk2 >= 2 \* H) or ((A(uk1) <= A(uk2)) and (uk1 < H))) then

202 begin

203 Tv(Count1) := A(Uk1);

204 Uk1 := Uk1 + 1;

205 end;

206 else

207 begin

208 Tv(Count1) := A(Uk2);

209 Uk2 := Uk2 + 1;

210 end;

211 end if;

212 end loop;

213 for Count1 in 1 .. 2 \* H loop

214 A(Count1) := Tv(Count1);

215 end loop;

216 --8. Ждать завершение счёта A2Н в задаче Т4 W4.3

217 MON.SortedT4;

218 --9. Слияние A = sort(A2Н, A2Н)

219 uk1 := 1;

220 uk2 := 2 \* H + 1;

221

222 for Count1 in 1 .. N loop

223 if ((uk2 >= N) or ((A(uk1) <= A(uk2)) and (uk1 < 2 \* H))) then

224 begin

225 Tv(Count1) := A(Uk1);

226 Uk1 := Uk1 + 1;

227 end;

228 else

229 begin

230 Tv(Count1) := A(Uk2);

231 Uk2 := Uk2 + 1;

232 end;

233 end if;

234 end loop;

235

236 for Count1 in 1 .. N loop

237 A(Count1) := Tv(Count1);

238 end loop;

239 --10. Вывод А.

240 if N < 24 then

241 Vec\_Print(A);

242 end if;

243

244 time2:=clock;

245 time\_all:=time2-time1;

246 Put(Integer(time\_all));

247

248 Put\_Line("T1 stop");

249 end T1;

250

251 -----------------------ЗАДАЧА Т2 --------------------------

252 task body T2 is

253 Sum1, Temp, Uk1, Uk2: Integer;

254 MO2: Matr; alfa2: Integer; D2: Vec;

255 MTF: Matr;

256 Tv: Vec;

257 begin

258 Put\_Line("T2 start");

259

260 --1. Ввод MT, D

261 Matr\_Input(MT);

262 Vec\_Input(D2);

263 MON.SetD(D2);

264 --2. Сигнал задачам Т1, Т3, Т4 о завершении ввода МT, D

265 MON.DataSignal;

266 --3. Ждать введения в T1, T3

267 MON.Inputed;

268 --4. Копирование MO2 := MO, ?2 := ?, D2 := D

269 MO2 := MON.GetMO;

270 Alfa2 := MON.GetAlfa;

271 --5. Cчёт AH = sort(СH \* ?2 - D2 \* (MO2 \* MTH)

272 for i in 1.. N loop

273 for j in H + 1 .. 2\*H loop

274 sum1 := 0;

275 for z in 1 .. N loop

276 sum1 := sum1 + MT(Z)(J) \* MO2(I)(Z);

277 end loop;

278 MTF(I)(J) := Sum1;

279 end loop;

280 end loop;

281 Sum1 := 0;

282 for i in H + 1 .. 2\*H loop

283 for j in 1 .. N loop

284 sum1 := sum1 + D2(J) \* MTF(j)(I);

285 end loop;

286 A(I) := C(I) \* Alfa2 + Sum1;

287 Sum1 := 0;

288 end loop;

289 for Count1 in H + 1 .. 2\*H loop

290 for Count2 in H + 1 .. 2\*H loop

291 if (A(Count1) > A(Count2)) then

292 begin

293 Temp := A(Count1);

294 A(Count1) := A(Count2);

295 A(Count2) := Temp;

296 end;

297 end if;

298 end loop;

299 end loop;

300 --6. Сигнал задаче Т1 о завершении счёта A2H

301 MON.AhT2;

302

303 Put\_Line("T2 stop");

304 end T2;

305

306 -----------------------ЗАДАЧА Т3 ----------------------------

307 task body T3 is

308 Sum1, Temp: Integer;

309 MO3: Matr; alfa3, Uk1, Uk2, Count1: Integer; D3: Vec;

310 MTF: Matr; Tv:Vec;

311 begin

312 Put\_Line("T3 start");

313

314 --1. Ввод МО

315 Matr\_Input(MO3);

316 MON.SetMO(MO3);

317 --2. Сигнал задачам Т1, Т2, Т4 о завершении ввода МО

318 MON.DataSignal;

319 --3. Ждать введения в T2, T3

320 MON.Inputed;

321 --4. Копирование MO4 := MO, ?4 := ?, D4 := D

322 Alfa3 := MON.GetAlfa;

323 D3 := MON.GetD;

324 --5. Cчёт AH = sort(СH \* ?2 - D2 \* (MO2 \* MTH)

325 for i in 1.. N loop

326 for j in 2\*H + 1 .. 3\*H loop

327 sum1 := 0;

328 for z in 1 .. N loop

329 sum1 := sum1 + MT(Z)(J) \* MO3(I)(Z);

330 end loop;

331 MTF(I)(J) := Sum1;

332 end loop;

333 end loop;

334 Sum1 := 0;

335 for i in 2\*H + 1 .. 3\*H loop

336 for j in 1 .. N loop

337 sum1 := sum1 + D3(J) \* MTF(j)(I);

338 end loop;

339 A(I) := C(I) \* Alfa3 + Sum1;

340 Sum1 := 0;

341 end loop;

342 for Count1 in 2\*H + 1 .. 3\*H loop

343 for Count2 in 2\*H + 1 .. 3\*H loop

344 if (A(Count1) > A(Count2)) then

345 begin

346 Temp := A(Count1);

347 A(Count1) := A(Count2);

348 A(Count2) := Temp;

349 end;

350 end if;

351 end loop;

352 end loop;

353 --6. Сигнал задаче Т4 о завершении счёта AH S4.2

354 MON.AhT3;

355

356 Put\_Line("T3 stop");

357 end T3;

358

359 -----------------------ЗАДАЧА Т4 --------------------------

360 task body T4 is

361 Sum1, Temp: Integer;

362 MO4: Matr; alfa4: Integer; D4: Vec;

363 MTF: Matr;

364 Uk1, Uk2: Integer;

365 Tv: Vec;

366 begin

367 Put\_Line("T4 start");

368

369 --1. Ждать введения в Т1, T2, T3

370 MON.Inputed;

371 --2. Копирование MO4 := MO, ?4 := ?, D4 := D

372 Alfa4 := MON.GetAlfa;

373 D4 := MON.GetD;

374 MO4 := MON.GetMO;

375 --3. Cчёт AH = sort(СH \* ?4 - D4 \* (MO4 \* MTH)

376 for i in 1.. N loop

377 for j in 3\*H + 1 .. n loop

378 sum1 := 0;

379 for z in 1 .. N loop

380 sum1 := sum1 + MT(Z)(J) \* MO4(I)(Z);

381 end loop;

382 MTF(I)(J) := Sum1;

383 end loop;

384 end loop;

385 Sum1 := 0;

386 for i in 3\*H + 1 .. n loop

387 for j in 1 .. N loop

388 sum1 := sum1 + D4(J) \* MTF(j)(I);

389 end loop;

390 A(I) := C(I) \* Alfa4 + Sum1;

391 Sum1 := 0;

392 end loop;

393 for Count1 in 3\*H + 1 .. n loop

394 for Count2 in 3\*H + 1 .. n loop

395 if (A(Count1) > A(Count2)) then

396 begin

397 Temp := A(Count1);

398 A(Count1) := A(Count2);

399 A(Count2) := Temp;

400 end;

401 end if;

402 end loop;

403 end loop;

404 --4. Ждать завершение счёта AH в задаче Т3 W3.2

405 MON.SortedT3;

406 --5. Слияние A2H = sort(AH, AH)

407 uk1 := 2 \* H + 1;

408 uk2 := 3 \* H + 1;

409

410 for Count1 in 2 \* H + 1 .. n loop

411 if ((uk2 >= N) or ((A(uk1) <= A(uk2)) and (uk1 < 3 \* H))) then

412 begin

413 Tv(Count1) := A(Uk1);

414 Uk1 := Uk1 + 1;

415 end;

416 else

417 begin

418 Tv(Count1) := A(Uk2);

419 Uk2 := Uk2 + 1;

420 end;

421 end if;

422 end loop;

423

424 for Count1 in 2 \* H + 1 .. N loop

425 A(Count1) := Tv(Count1);

426 end loop;

427 --6. Сигнал задаче Т1 о завершении счёта A2H S1.1

428 MON.A2hT4;

429

430 Put\_Line("T4 stop");

431 end T4;

432

433 begin

434 null;

435 end START;

436

437

438 begin

439 --Put\_Line("N == ");

440 --Put(N);

441 --Get(T);

442 time1:=clock;

443 START;

444 end Lab4;

## Додаток З. Лістінг программи для ПКС з ЛП

Source file: ..\start.adb Mon May 16 19:22:42 2011

1 ----------------------------------------------------------------

2 -- Paralel and distributed computing --

3 -- --

4 -- Course Work --

5 -- IO-83 --

6 -- 9.02.2011 --

7 ----------------------------------------------------------------

8

9 with Package1; use Package1;

10 with Ada.Text\_IO; use Ada.Text\_IO;

11 with Ada.Integer\_Text\_IO; use Ada.Integer\_Text\_IO;

12 with Ada.Calendar; use Ada.Calendar;

13 with ada.float\_text\_io; use ada.float\_text\_io;

14

15 procedure CoursePRO is

16 N: integer := 1000; --size of structures

17 P: integer := 4; --count of processors

18 H: integer := N/P; --size of piece

19 time\_all: duration;

20 time1,time2: time;

21 A: Integer;

22

23 --типы оптимальные для рассылки данных между задачами.

24 type vector is array(integer range <>) of integer;

25 Subtype Vector\_N is Vector(1..N);

26 Subtype Vector\_2h is Vector(1..2\*H);

27 Subtype Vector\_H is Vector(1..H);

28

29 type Matrix is array(integer range <>) of Vector\_N;

30 Subtype Matrix\_N is Matrix(1..N);

31 Subtype Matrix\_2H is Matrix(1..2\*H);

32 Subtype Matrix\_H is Matrix(1..H);

33

34 procedure START is

35 --Спецификация задач

36 task T1 is

37 entry DMT(MT: in Matrix\_H; D: in Vector\_N); --кусочки данных

38 entry MMO(MO: in Matrix\_N);

39 entry AHT2(V: in Vector\_H);

40 entry A2HT4(V: in Vector\_2H);

41 end T1;

42

43 task T2 is

44 entry CAlfa(C: in Vector\_2H; Alfa: in Integer);

45 entry MMO(MO: in Matrix\_N);

46 end T2;

47

48 task T3 is

49 entry CAlfa(C: in Vector\_H; Alfa: in Integer);

50 entry DMT(MT: in Matrix\_2H; D: in Vector\_N); --кусочки данных

51 end T3;

52

53 task T4 is

54 entry CAlfa(C: in Vector\_H; Alfa: in Integer);

55 entry MMO(MO: in Matrix\_N);

56 entry DMT(MT: in Matrix\_H; D: in Vector\_N); --кусочки данных

57 entry AHT4(V: in Vector\_H);

58 end T4;

59

60 --Реализация задач

61 task body T1 is

62 Sum1: Integer := 0;

63 Temp, Count1, Count2, uk1, uk2: Integer;

64

65 Alfa1: Integer;

66 MO1, MTF: Matrix\_N;

67 D1: vector\_N;

68 Cx: Vector\_N;

69 MTx: Matrix\_H;

70 A: Vector\_N;

71 Tv:Vector\_N;

72 begin

73 put\_Line("T1 started!");

74

75 --1. Ввод C, ?

76 for i in 1 .. N loop

77 cx(i) := 1;

78 end loop;

79 Alfa1 := 1;

80 --2. Передать CH, ? задаче Т4

81 T4.CAlfa(Cx(3\*H+1..N), Alfa1);

82 --3. Передать C2H, ? задаче Т2

83 T2.CAlfa(Cx(H+1..3\*H), Alfa1);

84 --4. Принять D, MTH от задачи Т2

85 accept DMT(MT: in Matrix\_H; D: in Vector\_N) do

86 D1 := D;

87 MTx := MT;

88 end DMT;

89 --5. Принять МО от задачи Т2

90 accept MMO(MO: in Matrix\_N) do

91 MO1 := MO;

92 end MMO;

93 --6. Счёт АН = sort(CH \* ? + D \* (MO \* MTH))

94 for i in 1.. N loop

95 for j in 1 .. H loop

96 sum1 := 0;

97 for z in 1 .. N loop

98 sum1 := sum1 + MTx(J)(Z) \* MO1(Z)(I);

99 end loop;

100 MTF(I)(J) := Sum1;

101 end loop;

102 end loop;

103 Sum1 := 0;

104 for i in 1.. H loop

105 for j in 1 .. N loop

106 sum1 := sum1 + D1(J) \* MTF(j)(I);

107 end loop;

108 A(I) := Cx(I) \* Alfa1 + Sum1;

109 Sum1 := 0;

110 end loop;

111 for Count1 in 1 .. H loop

112 for Count2 in 1 .. H loop

113 if (A(Count1) > A(Count2)) then

114 begin

115 Temp := A(Count1);

116 A(Count1) := A(Count2);

117 A(Count2) := Temp;

118 end;

119 end if;

120 end loop;

121 end loop;

122 put\_Line("LOL!");

123 --7. Принять АН от задачи Т2

124 accept AHT2(V: in Vector\_H) do

125 A(H+1..2\*H) := V;

126 end AHT2;

127 --8. Слияние A2H = sort(AH, AH)

128 uk1 := 1;

129 uk2 := H+1;

130 for Count1 in 1 .. 2 \* H loop

131 if ((uk2 >= 2 \* H) or ((A(uk1) <= A(uk2)) and (uk1 < H))) then

132 begin

133 Tv(Count1) := A(Uk1);

134 Uk1 := Uk1 + 1;

135 end;

136 else

137 begin

138 Tv(Count1) := A(Uk2);

139 Uk2 := Uk2 + 1;

140 end;

141 end if;

142 end loop;

143 for Count1 in 1 .. 2 \* H loop

144 A(Count1) := Tv(Count1);

145 end loop;

146 --9. Принять А2H от задачи Т4

147 accept A2HT4(V: in Vector\_2H) do

148 A(2\*H+1..N) := V;

149 end A2HT4;

150 --10. Слияние A = sort(А2H, А2H)

151 uk1 := 1;

152 uk2 := 2\*H+1;

153 for Count1 in 1 .. N loop

154 if ((uk2 >= N) or ((A(uk1) <= A(uk2)) and (uk1 < 2 \* H))) then

155 begin

156 Tv(Count1) := A(Uk1);

157 Uk1 := Uk1 + 1;

158 end;

159 else

160 begin

161 Tv(Count1) := A(Uk2);

162 Uk2 := Uk2 + 1;

163 end;

164 end if;

165 end loop;

166 for Count1 in 1 .. N loop

167 A(Count1) := Tv(Count1);

168 end loop;

169 --11. Вывод А

170 if N < 24 then

171 for i in 1 .. N loop

172 Put(A(i), 4);

173 end loop;

174 end if;

175

176 time2:=clock;

177 time\_all:=time2-time1;

178 Put(Integer(time\_all));

179 end T1;

180

181 task body T2 is

182 Sum1: Integer := 0;

183 Temp, Count1, Count2, uk1, uk2: Integer;

184

185 Alfa1: Integer;

186 MO1, MTF: Matrix\_N;

187 D1: vector\_N;

188 Cx: Vector\_2H;

189 MTx: Matrix\_N;

190 A: Vector\_H;

191 Tv:Vector\_N;

192 begin

193 put\_Line("T2 started!");

194

195 --1. Ввод D, MT

196 for i in 1 .. N loop

197 for J in 1 .. N loop

198 MTx(i)(J) := 1;

199 end loop;

200 end loop;

201 for i in 1 .. N loop

202 D1(i) := 1;

203 end loop;

204 --2. Принять C2H, ? от задачи Т1

205 accept CAlfa(C: in Vector\_2H; Alfa: in Integer) do

206 Alfa1 := Alfa;

207 Cx := C;

208 end CAlfa;

209 --3. Передать CH, ? задаче Т3

210 T3.CAlfa(Cx(H+1..2\*H), Alfa1);

211 --4. Передать D, MTH задаче Т1

212 T1.DMT(MTx(1..H), D1);

213 --5. Передать D, MT2H задаче Т3

214 T3.DMT(MTx(2\*H+1..N), D1);

215 --6. Принять МО от задачи Т3

216 accept MMO(MO: in Matrix\_N) do

217 MO1 := MO;

218 end MMO;

219 --7. Передать МО задаче Т1

220 T1.MMO(MO1);

221 --8. Счёт АН = sort(CH \* ? + D \* (MO \* MTH))

222 for i in 1.. N loop

223 for j in 1 .. H loop

224 sum1 := 0;

225 for z in 1 .. N loop

226 sum1 := sum1 + MTx(J+H)(Z) \* MO1(Z)(I);

227 end loop;

228 MTF(I)(J) := Sum1;

229 end loop;

230 end loop;

231 Sum1 := 0;

232 for i in 1.. H loop

233 for j in 1 .. N loop

234 sum1 := sum1 + D1(J) \* MTF(j)(I);

235 end loop;

236 A(I) := Cx(I) \* Alfa1 + Sum1;

237 Sum1 := 0;

238 end loop;

239 for Count1 in 1 .. H loop

240 for Count2 in 1 .. H loop

241 if (A(Count1) > A(Count2)) then

242 begin

243 Temp := A(Count1);

244 A(Count1) := A(Count2);

245 A(Count2) := Temp;

246 end;

247 end if;

248 end loop;

249 end loop;

250 --9. Передать АН задаче Т1

251 T1.AHT2(A);

252

253 Put\_Line("T2 ended");

254 end T2;

255

256 task body T3 is

257 Sum1: Integer := 0;

258 Temp, Count1, Count2, uk1, uk2: Integer;

259

260 Alfa1: Integer;

261 MO1, MTF: Matrix\_N;

262 D1: vector\_N;

263 Cx: Vector\_H;

264 MTx: Matrix\_2H;

265 A: Vector\_H;

266 Tv:Vector\_N;

267 begin

268 put\_Line("T3 started!");

269

270 --1. Ввод MО

271 for i in 1 .. N loop

272 for J in 1 .. N loop

273 MO1(i)(J) := 1;

274 end loop;

275 end loop;

276 --2. Принять CH, ? от задачи Т2

277 accept CAlfa(C: in Vector\_H; Alfa: in Integer) do

278 Alfa1 := Alfa;

279 Cx := C;

280 end CAlfa;

281 --3. Принять D, MT2H от задачи Т2

282 accept DMT(MT: in Matrix\_2H; D: in Vector\_N) do

283 D1 := D;

284 MTx := MT;

285 end DMT;

286 --4. Передать МО задаче Т2

287 T2.MMO(MO1);

288 --5. Передать МО задаче Т4

289 T4.MMO(MO1);

290 --6. Передать D, MTH задаче Т4

291 T4.DMT(MTx(H+1..2\*H), D1);

292 --7. Счёт АН = sort(CH \* ? + D \* (MO \* MTH))

293 for i in 1.. N loop

294 for j in 1 .. H loop

295 sum1 := 0;

296 for z in 1 .. N loop

297 sum1 := sum1 + MTx(J)(Z) \* MO1(Z)(I);

298 end loop;

299 MTF(I)(J) := Sum1;

300 end loop;

301 end loop;

302 Sum1 := 0;

303 for i in 1.. H loop

304 for j in 1 .. N loop

305 sum1 := sum1 + D1(J) \* MTF(j)(I);

306 end loop;

307 A(I) := Cx(I) \* Alfa1 + Sum1;

308 Sum1 := 0;

309 end loop;

310 for Count1 in 1 .. H loop

311 for Count2 in 1 .. H loop

312 if (A(Count1) > A(Count2)) then

313 begin

314 Temp := A(Count1);

315 A(Count1) := A(Count2);

316 A(Count2) := Temp;

317 end;

318 end if;

319 end loop;

320 end loop;

321 --8. Передать АН задаче Т4

322 T4.AHT4(A);

323

324 Put\_Line("T3 ended");

325 end T3;

326

327 task body T4 is

328 Sum1: Integer := 0;

329 Temp, Count1, Count2, uk1, uk2: Integer;

330

331 Alfa1: Integer;

332 MO1, MTF: Matrix\_N;

333 D1: vector\_N;

334 Cx: Vector\_H;

335 MTx: Matrix\_H;

336 A: Vector\_2H;

337 Tv:Vector\_N;

338 begin

339 put\_Line("T4 started!");

340

341 --1. Принять CH, ? от задачи Т1

342 accept CAlfa(C: in Vector\_H; Alfa: in Integer) do

343 Alfa1 := Alfa;

344 Cx := C;

345 end CAlfa;

346 --2. Принять МО от задачи Т3

347 accept MMO(MO: in Matrix\_N) do

348 MO1 := MO;

349 end MMO;

350 --3. Принять D, MTH от задачи Т3

351 accept DMT(MT: in Matrix\_H; D: in Vector\_N) do

352 D1 := D;

353 MTx := MT;

354 end DMT;

355 --4. Счёт АН = sort(CH \* ? + D \* (MO \* MTH))

356 for i in 1.. N loop

357 for j in 1 .. H loop

358 sum1 := 0;

359 for z in 1 .. N loop

360 sum1 := sum1 + MTx(J)(Z) \* MO1(Z)(I);

361 end loop;

362 MTF(I)(J) := Sum1;

363 end loop;

364 end loop;

365 Sum1 := 0;

366 for i in 1.. H loop

367 for j in 1 .. N loop

368 sum1 := sum1 + D1(J) \* MTF(j)(I);

369 end loop;

370 A(I+H) := Cx(I) \* Alfa1 + Sum1;

371 Sum1 := 0;

372 end loop;

373 for Count1 in 1 .. H loop

374 for Count2 in 1 .. H loop

375 if (A(Count1) > A(Count2)) then

376 begin

377 Temp := A(Count1);

378 A(Count1) := A(Count2);

379 A(Count2) := Temp;

380 end;

381 end if;

382 end loop;

383 end loop;

384 --5. Принять АН от задачи Т3

385 accept AHT4(V: in Vector\_H) do

386 A(1..H) := V;

387 end AHT4;

388 --6. Слияние A2H = sort(AH, AH)

389 uk1 := 1;

390 uk2 := H+1;

391 for Count1 in 1 .. 2 \* H loop

392 if ((uk2 >= 2 \* H) or ((A(uk1) <= A(uk2)) and (uk1 < H))) then

393 begin

394 Tv(Count1) := A(Uk1);

395 Uk1 := Uk1 + 1;

396 end;

397 else

398 begin

399 Tv(Count1) := A(Uk2);

400 Uk2 := Uk2 + 1;

401 end;

402 end if;

403 end loop;

404 --7. Передать A2H задаче Т1

405 T1.A2HT4(A);

406

407 Put\_Line("T4 ended");

408 end T4;

409

410 begin

411 null;

412 end START;

413

414 begin

415 Put\_Line("N == ");

416 Put(N);

417 Get(A);

418 time1:=clock;

419 START;

420 end CoursePRO;

421