Лабораторная работа № 3

Взаимодействие распределено выполняемых процессов с помощью сообщений. Образование каналов передачи данных.

1. Цель работы: изучение способов организации взаимодействия распределено выполняющихся процессов, их реализация средствами библиотеки PVM, также способов организации и методов реализации каналов передачи данных между хостами в вычислительном кластере.

2. Теоретическое введение.

Реализация распределенных вычислений осуществляется на вычислительных системах, имеющих архитектуру с распределенной памятью. При такой архитектуре процессоры имеют собственную локальную память (т.е. не разделяют общие переменные) и общаются с помощью сети связи, т.е. обмениваются сообщениями. При передаче сообщений процессы разделяют каналы связи. Каждый канал обеспечивает путь взаимодействия между процессами. Т.о. основной задачей при формировании распределенной программы является реализация межпроцессорного взаимодействия. Существуют различные виды взаимодействия: схема производителя и потребителя, схема «клиент-сервер» и схема «взаимодействующие равные».

Методы взаимодействия по схеме «производитель-потребитель» отличаются способами использования каналов связи и синхронизации процессов. Будем считать, что каналы, условно, обеспечивают однонаправленную передачу данных от передающего процесса получающему, а взаимодействие может быть как асинхронным, так и синхронным.

Особенностью передачи сообщений является то, что отправка сообщения может быть как асинхронной (не блокирующей выполнение процесса-генератора сообщения), так и синхронной (блокирующей процесс-генератор). Получение же сообщения из канала связи является блокирующим (в большинстве случаев, однако есть особенности в языковых реализациях).

При асинхронной передаче сообщений канал связи может считаться очередью сообщений, которые уже отправлены, но не получены. В общем виде процесс передачи сообщений через некоторый канал связи может быть представлен в виде следующего оператора:

SEND <имя_канала>(<параметр 1>, <параметр 2>…).

В этом случае, сообщение с полученными значениями присоединяется к концу очереди, связанной с каналом. Т.к. теоретически очередь не ограничена, выполнение операции SEND не вызывает задержку, т.е эта операция для источника данных (производителя) является неблокирующей.

Процесс получает сообщение из канала, выполняя операцию:

RECEIVE <имя_канала> (<параметр 1>, <параметр 2>…).

При выполнении операции RECEIVEпринимающий процесс приостанавливается до тех пор, пока в очереди канала не будет хотя бы одного сообщения. Т.о., выполнение операции RECEIVE может создать задержку, поэтому она является блокирующей. Однонаправленные каналы используются процессами совместно, поэтому они являются глобальными относительно процессов. Любой процесс может отправлять данные в любой канал и принимать их из любого канала (в этом случае каналы являются почтовыми ящиками). У канала может быть также только один получатель и несколько отправителей (такой канал называют входным портом).

При синхронной передаче отправка сообщения приводит к блокировке процесса-отправителя до тех пор, пока сообщение не будет получено. Аргументами примитива SEND с синхронизацией являются имя канала и само передаваемое сообщение (общий вид оператора SYNCH_SEND()).

Определенные выше примитивы асинхронной и синхронной передачи сообщений обеспечивают прежде всего односторонний обмен данными, когда процессы находятся в отношении «производитель-потребитель» (не равноправные отношения). Процессы могут также находится в отношении равноправного взаимодействия (схема «взаимодействующие равные»). В схеме «взаимодействующие равные» различают три способа взаимодействия:

1) централизованное решение – предполагает взаимный обмен сообщениями по разным КС – один КС для передачи данных, другой – для приема.
2) Симметричное решение – поочередно используется один симплексный канал сначала для приема, затем для отправки сообщений.
3) Кольцевое решение – процесс принимает сообщение от своего предшественника и передает своему преемнику. Т.о. сообщение может перемещаться по виртуальному кольцу хостов пока не достигнет требуемого хоста.

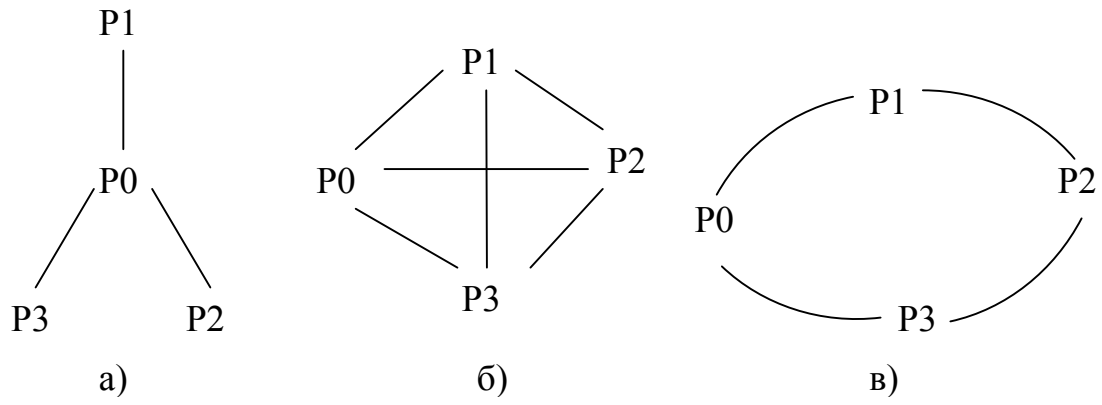

Рис. 2.1 Структуры «взаимодействия равных»

Естественным является сопоставление каналу передачи данных между процессами в вычислительном кластере некоторого буфера для хранения данных с дисциплиной обслуживания FIFO. Т.к. процесс, выполняющийся на хосте, может взаимодействовать с несколькими процессами, выполняющимися на других хостах кластера, посредством уникально определенных каналов, то для этого процесса возникает потребность организации нескольких буферов. Использование подпрограммы PVM-initsend () обеспечивает создание лишь одного активного буфера (в нашей интерпретации – канала передачи), посредством которого возможна организация обмена процесса с другими процессами вычислительного кластера. Однако такая реализация обмена требует наличия специальных средств контроля за приемом сообщений и не позволяет передавать их в произвольном темпе по мере формирования сообщений. Поэтому в PVM предусмотрена возможность создания и использования нескольких буферов, каждый из которых будет реализовывать отдельный канал передачи сообщений между процессами.

Подпрограмма pvm_mkbuf создает новый пустой буфер передачи. В качестве параметра указывается метод кодирования данных, а сама подпрограмма возвращает значение переменной, равное номеру созданного буфера. Формат подпрограммы:

Int bufid = pvm_mkbuf (encoding).

Отрицательное значение идентификатора буфера говорит об ошибке. После использования (передачи сообщения) буфер должен быть удален командой pvm_freebuf (bufid), указывающей номер удаляемого буфера.

В процессе передачи данных через буферы различным получателям необходимо осуществлять переключение между буферами (активизацию не активного буфера – активизацию канала передачи данных). Активизация буфера на передающей и принимающей сторонах осуществляется соответственно командами (в любой момент времени активным является 1 буфер):

Int oldbuf = pvm_setsbuf (bufid);
Int oldbuf = pvm_setrbuf (bufid);

где переменная bufid идентифицирует активизируемый буфер, в переменную oldbuf записывается идентификатор ранее активного буфера.

Контроль идентификатора текущего активного буфера осуществляется подпрограммами (на передающей и принимающей сторонах соответственно):

Int bufid = pvm_getsbuf ();
Int bufid = pvm_getrbuf ();

где bufid определяет значение идентификатора буфера.

Для отправки сообщений должна быть использована введенная ранее команда pvm-send (tid, msgtag), которая является неблокирующей. Введенная команда pvm-recv (tid, msqtag) является блокирующей работу процесса, ее вызвавшего. Библиотека PVM предоставляет дополнительные функции по контролю сообщений на стороне потребителя:

1) подпрограмма pvm_nrecv выполняет неблокирующий прием сообщений от процесса с идентификатором tid;

int buf = pvm_nrecv (tid, msgtag);

Если сообщение не пришло, эта подпрограмма возвращает 0, если сообщение пришло, то оно заместит текущее содержимое буфера, а подпрограмма вернет номер буфера, куда оно помещено.

2) Подпрограмма pvm_probe проверяет прибыло ли ожидаемое сообщение, но не принимает его. Формат вызова следующий:

Int buf = pvm_probe (tid, msgtag).

Подпрограмма возвращает нулевое значение переменной buf, если сообщение не пришло. В случае приема сообщения переменная buf содержит номер активного буфера, куда оно помещено.

Указание в качестве значений параметров tid и msgtag значения -1 позволяет разрешить прием сообщения с любым номером от любого процесса, функционирующего в составе кластера.

3. Задание на работу.

Задание на работу выбирается в соответствии с вариантом, назначаемым преподавателем.

1 вариант

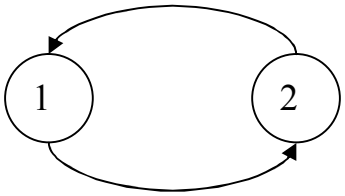

Рис.3.1
Схема взаимодействия выполняющихся процессов

Реализовать взаимодействие двух распределено выполняющихся процессов по централизованной схеме «взаимодействие равных» с организацией двух однонаправленных каналов обмена данными (создание двух буферов обмена, рис.3.1). При этом первой программой выполняется ввод исходных данных и вывод полученных результатов. Вторая программа осуществляет вычисление определителя матрицы 3x3. Ни один из выполняемых процессов не осуществляет блокирующего ожидания сообщения, при этом второй процесс первоначально определяет принятие сообщения, а затем его принимает.

2 вариант

Реализовать взаимодействие процессов, выполняемых на трех хостах вычислительного кластера по кольцевой схеме «взаимодействие равных» с организацией трех каналов передачи данных между хостами (рис.3.2). Первая программа при этом осуществляет ввод исходных данных и вывод результата. Вторая программа выполняет вычисление определителя матрицы размером 3x3, а третья программа определяет матрицу,

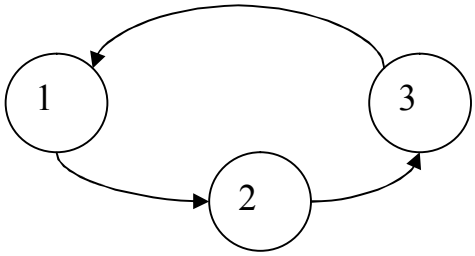

Рис. 3.2
Кольцевая схема взаимодействия выполняющихся процессов

обратную введенной матрице размером 3x3. Первый процесс ожидает сообщения в блокирующем режиме, второй процесс не блокируется в ожидании сообщения, а третий процесс первоначально проверяет принятие сообщения, а уже затем его принимает.

3 вариант

Реализовать конвейер вычислительных операций на вычислительном кластере, состоящем из трех хостов. Первый сегмент конвейера (хост 1) осуществляет ввод данных, второй – их обработку (умножение матриц размером 3x3), третий – ввод данных (рис.3.3). Для передачи данных между сегментами выделены отдельные каналы. Второй сегмент выполняет неблокирующий прием сообщений, третий перед приемом сообщения проверяет его наличие.

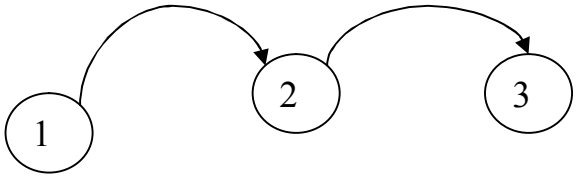

Рис. 3.3
Конвейер вычислительных операций

Во всех вариантах в соответствии с условиями в каждом из КС предполагается наличие сообщений. Таким образом, при реализации одного активного буфера для обмена данными между хостами кластера последующее размещенное в буфере сообщение может уничтожить находящиеся там еще не прочитанные сообщения. Поэтому для обмена необходимо выделять отдельные каналы передачи данных между хостами кластера.

4. Контрольные вопросы.

Понятие канала передачи между процессами в вычислительном кластере.

Понятие блокирующего и неблокирующего сообщения, блокирующего и неблокирующего ожидания сообщения.

Схемы взаимодействия между «равными» процессами.

Средства библиотеки PVM по созданию и управлению буферами передачи данных.

Средства библиотеки PVM по реализации неблокирующего приема сообщений.