ствует понятия иерархии процессов, и все процессы равноправны. Единственное, в чем проявляется что-то вроде иерархии процессов — создание процесса, в котором родительский процесс получает специальный маркер (так называемый **дескриптор**), позволяющий контролировать дочерний процесс. Но маркер можно передать другому процессу, нарушая иерархию. В UNIX это невозможно.

Потоки

В обычных операционных системах каждому процессу соответствует адресное пространство и одиночный управляющий поток. Фактически это и определяет процесс. Тем не менее часто встречаются ситуации, в которых предпочтительно

иметь несколько квазипараллельных управляющих потоков в одном адресном пространстве, как если бы они были различными процессами (однако разделяющим одно адресное пространство). В следующих разделах мы рассмотрим такие ситуации.

Модель потока

Модель процесса, которую мы рассматривали, базируется на двух независимых концепциях: группировании ресурсов и выполнении программы. Иногда полезно их разделять, и тут появляется понятие потока.

С одной стороны, процесс можно рассматривать как способ объединения родственных ресурсов в одну группу. У процесса есть адресное пространство, содержащее текст программы и данные, а также другие ресурсы. Ресурсами являются открытые файлы, дочерние процессы, необработанные аварийные сообщения, обработчики сигналов, учетная информация и многое другое. Гораздо проще управлять ресурсами, объединив их в форме процесса.

С другой стороны, процесс можно рассматривать как поток исполняемых команд или просто **поток**. У потока есть счетчик команд, отслеживающий порядок выполнения действий. У него есть регистры, в которых хранятся текущие переменные. У него есть стек, содержащий протокол выполнения процесса, где на каждую процедуру, вызванную, но еще не вернувшуюся, отведен отдельный фрейм. Хотя поток должен исполняться внутри процесса, следует различать концепции потока и процесса. Процессы используются для группирования ресурсов, а потоки являются объектами, поочередно исполняющимися на центральном процессоре.

Концепция потоков добавляет к модели процесса возможность одновременного выполнения в одной и той же среде процесса нескольких программ, в достаточной степени независимых. Несколько потоков, работающих параллельно в одном процессе, аналогичны нескольким процессам, идущим параллельно на одном компьютере. В первом случае потоки разделяют адресное пространство, открытые файлы и другие ресурсы. Во втором случае процессы совместно пользуются физической памятью, дисками, принтерами и другими ресурсами. Потоки обладают некоторыми свойствами процессов, поэтому их иногда называют **упрощенными процессами**. Термин **многопоточность** также используется для описания использования нескольких потоков в одном процессе.

На рис. 2.4, *а* представлены три обычных процесса, у каждого из которых есть собственное адресное пространство и одиночный поток управления. На рис. 2.4, *б* представлен один процесс с тремя потоками управления. В обоих случаях мы имеем три потока, но на рис. 2.4, *а* каждый из них имеет собственное адресное пространство, а на рис. 2.4, *б* потоки разделяют единое адресное пространство.

При запуске многопоточного процесса в системе с одним процессором потоки работают поочередно. Пример работы процессов в многозадачном режиме мы уже видели на рис. 2.1. Иллюзия параллельной работы нескольких различных последовательных процессов создается путем постоянного переключения системы между процессами. Многопоточность реализуется примерно так же. Процессор быстро переключается между потоками, создавая впечатление параллельной работы потоков, хотя и на не столь быстром процессоре. В случае трех ограниченных производительностью процессора потоков в одном процессе все потоки

будут работать параллельно, и каждому потоку будет соответствовать виртуальный процессор с быстродействием, равным одной трети быстродействия реального процессора.

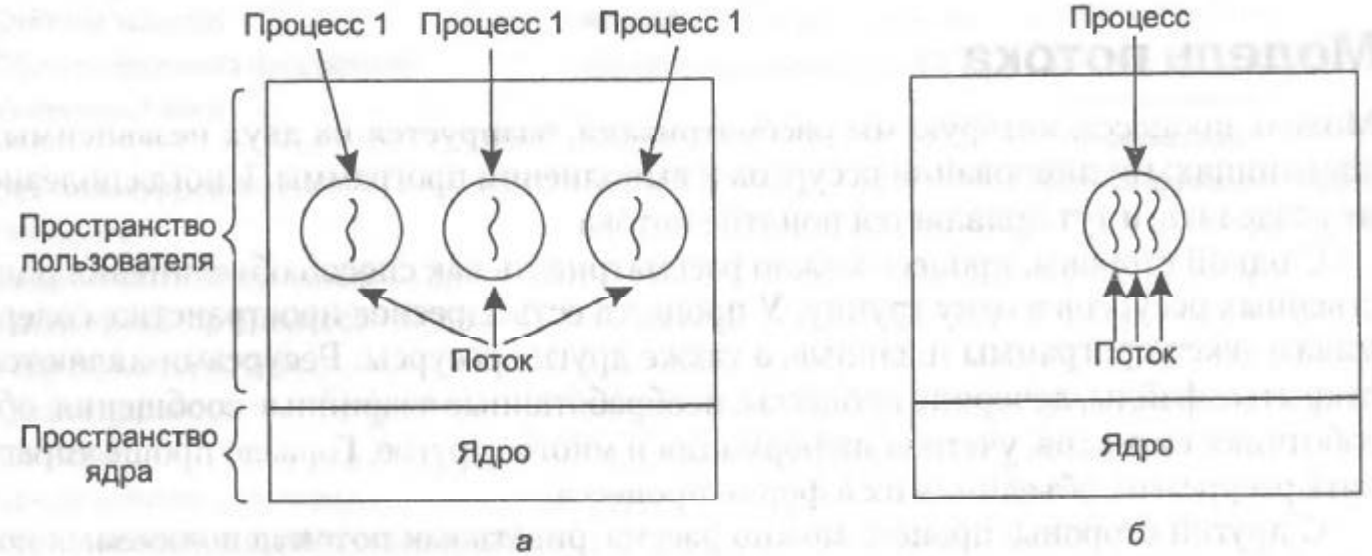

Рис. 2.4. Три процесса с одиночными потоками управления (*а*); один процесс с тремя потоками управления (*б*)

Различные потоки в одном процессе не так независимы, как различные процессы. У всех потоков одно и то же адресное пространство, что означает совместное использование глобальных переменных. Поскольку любой поток имеет доступ к любому адресу ячейки памяти в адресном пространстве процесса, один поток может считывать, записывать или даже стирать информацию из стека другого потока. Защиты не существует, поскольку (1) это невозможно и (2) это ненужно. В отличие от различных процессов, которые могут быть инициированы различными пользователями и преследовать несовместимые цели, один процесс всегда запущен одним пользователем, и потоки созданы таким образом, чтобы работать совместно, не мешая друг другу. Как показано в табл. 2.3, потоки разделяют не только адресное пространство, но и открытые файлы, дочерние процессы, сигналы и т. п. Таким образом, ситуацию на рис. 2.4, *а* следует использовать в случае абсолютно несвязанных процессов, тогда как схема на рис. 2.4, *б* будет уместна, когда потоки выполняют совместно одну работу.

Таблица 2.3. В первой колонке перечислены элементы, совместно используемые всеми потоками процесса, а во второй — элементы, индивидуальные для каждого потока

Элементы процесса	Элементы потока
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и их обработчики	
Информация об использовании ресурсов	

Первая колонка содержит элементы, являющиеся свойствами процесса, а не потока. Например, если один поток открывает файл, этот файл тут же становится видимым для остальных потоков, и они могут считывать информацию и записывать ее в файл. Это логично, поскольку процесс, а не поток является единицей управления ресурсами. Если бы у каждого потока было собственное адресное пространство, открытые файлы, аварийные сигналы, требующие обработки и т. д., это были бы отдельные процессы. Концепция потоков состоит в возможности совместного использования набора ресурсов несколькими потоками для выполнения некой задачи в тесном взаимодействии.

Как и любой обычный процесс (то есть процесс с одним потоком), поток может находиться в одном из нескольких состояний: рабочем, заблокированном, готовности или завершенном. Действующий поток взаимодействует с процессором. Блокированный поток ожидает некоторого события, которое его разблокирует. Например, при выполнении системного запроса чтения с клавиатуры поток блокируется, пока не поступит сигнал с клавиатуры. Поток может быть разблокирован каким-либо внешним событием или другим потоком. Поток в состоянии готовности будет запущен, как только до него дойдет очередь. Переходы между состояниями потоков такие же, как на рис. 2.2.

Важно понимать, что у каждого потока свой собственный стек, как показано на рис. 2.5. Стек каждого потока содержит по одному фрейму для каждой процедуры, вызванной, но еще не вернувшей управления. Во фрейме находятся локальные переменные процедуры и адрес возврата. Например, если процедура *X* вызывает процедуру *Y* и она, в свою очередь, вызывает процедуру *Z*, то во время работы процедуры *Z* в стеке будут находиться фреймы для всех трех процедур. Каждый поток может вызывать различные процедуры и, соответственно, иметь различный протокол выполнения процесса — именно поэтому каждому потоку необходим собственный стек.

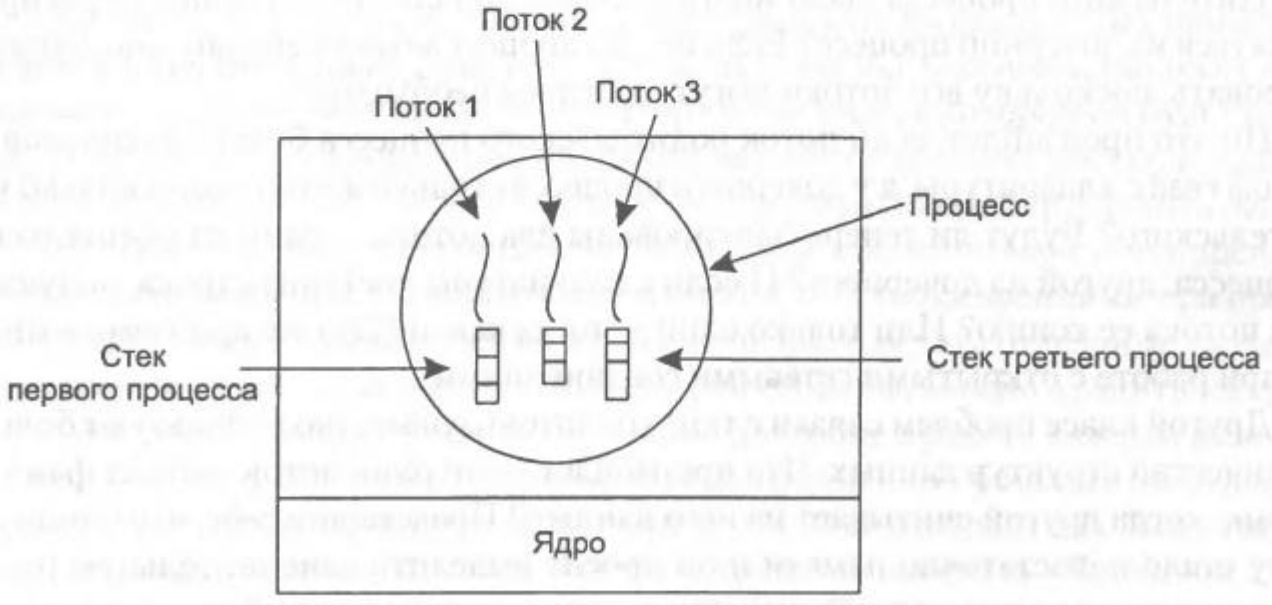

Рис. 2.5. У каждого потока свой собственный стек

В многопоточном режиме процессы, как правило, запускаются с одним потоком. Этот поток может создавать новые потоки, вызывая библиотечную процедуру, например *thread_create*. Параметром обычно является имя процедуры, кото-

рую необходимо запустить для создания нового потока. Указание какой-либо информации, касающейся адресного пространства нового потока, не является необходимым (или даже возможным), поскольку новый поток создается в адресном пространстве существующего потока. Иногда возникает иерархия потоков с отношениями типа «родительский—дочерний поток», но чаще всего иерархия отсутствует и все потоки считаются равнозначными. Независимо от иерархических отношений, создающему потоку чаще всего возвращается идентификатор потока, который дает имя новому потоку.

Выполнив задачу, поток может прекратить работу, вызвав библиотечную процедуру, скажем, *thread_exit*. После этого поток исчезает и уже не рассматривается планировщиком. В некоторых потоковых системах один поток может ждать прекращения работы другого (определенного) потока. Для этого вызывается процедура, например *thread_wait*. Процедура блокирует вызывающий процедуру поток, пока другой поток (определенный) не прекратит работу. В этом отношении создание и завершение потоков очень похожи на создание и завершение процессов с практически такими же параметрами.

Еще одно распространенное обращение потока — *thread_yield* — позволяет потоку добровольно «уступать свою очередь» другому потоку. Это важный момент, поскольку в случае потоков не существует прерывания по таймеру, позволяющего установить режим разделения времени, как это было в случае процессов. Потокам необходимо быть вежливыми и время от времени самим уступать процессор другим потокам. Существуют и процедуры, позволяющие одному потоку подождать, пока другой завершит какое-либо действие, оповестить о том, что он закончил какое-либо действие и т. п.

Несмотря на то что потоки часто бывают полезными, они существенно усложняют программную модель. Представьте себе системный вызов `fork` в UNIX. Если у родительского процесса было много потоков, должно ли это свойство распространяться на дочерний процесс? Если нет, то процесс может неправильно функционировать, поскольку все потоки могут оказаться необходимы.

Но что произойдет, если поток родительского процесса будет блокирован вызовом `read` с клавиатуры, а у дочернего процесса столько же потоков, сколько у родительского? Будут ли теперь блокированы два потока — один из родительского процесса, другой из дочернего? И если с клавиатуры поступит строка, получат ли оба потока ее копию? Или только один — тогда какой? Эта же проблема возникает при работе с открытыми сетевыми соединениями.

Другой класс проблем связан с тем, что потоки совместно используют большое количество структур данных. Что произойдет, если один поток закроет файл в то время, когда другой считывает из него данные? Представьте себе, что одному потоку стало недостаточно памяти и он просит выделить дополнительную память. На полпути происходит переключение потоков, и теперь новый поток также замечает, что ему не хватает памяти, и просит выделить дополнительную память. В этой ситуации память может быть выделена дважды. Все эти проблемы можно решить, но для создания корректно работающих многопоточных программ необходима тщательная и всесторонне обдуманная разработка.

Потоки выполнения. Распределенные и параллельные системы, их отличия. Масштабируемость, закон Амдала.

Параллельные системы

Параллельные системы — это физические компьютерные, а также программные системы, реализующие тем или иным способом параллельную обработку данных на многих вычислительных узлах.

Идея распараллеливания вычислений базируется на том, что большинство задач может быть разделено на набор меньших задач, которые могут быть решены одновременно. Обычно параллельные вычисления требуют координации действий. Параллельные вычисления существуют в нескольких формах: параллелизм на уровне битов, параллелизм на уровне инструкций, параллелизм данных, параллелизм задач. Параллельные вычисления использовались много лет в основном в высокопроизводительных вычислениях, но в последнее время к ним возрос интерес вследствие существования физических ограничений на рост тактовой частоты процессоров. Параллельные вычисления стали доминирующей парадигмой в архитектуре компьютеров, в основном в форме многоядерных процессоров.

Писать программы для параллельных систем сложнее, чем для последовательных, так как конкуренция за ресурсы представляет новый класс потенциальных ошибок в программном обеспечении, среди которых состояние гонки является самой распространенной. Взаимодействие и синхронизация между процессами представляют большой барьер для получения высокой производительности параллельных систем. В последние годы также стали рассматривать вопрос о потреблении электроэнергии параллельными компьютерами. Характер увеличение скорости программы в результате распараллеливания объясняется законом Амдала.

Если при вычислении не применяются циклические (повторяющиеся) действия, то N вычислительных модулей никогда не выполнят работу в N раз быстрее, чем один единственный вычислительный модуль.

Например, для быстрой сортировки массива на двухпроцессорной машине можно разделить массив пополам и сортировать каждую половину на отдельном процессоре. Сортировка каждой половины может занять разное время, поэтому необходима синхронизация.

Распределенные системы

Распределённые системы — способ решения трудоёмких вычислительных задач с использованием двух и более компьютеров объединённых в сеть.

Распределённые вычисления являются частным случаем параллельных вычислений, то есть одновременного решения различных частей одной вычислительной задачи несколькими процессорами одного или нескольких компьютеров. Поэтому необходимо, чтобы решаемая задача была сегментирована - разделена на подзадачи, которые могут вычисляться параллельно. При этом для распределенных вычислений приходится также

учитывать возможное различие в вычислительных ресурсах, которые будут доступны для расчёта различных подзадач. Однако, не всякую задачу можно