Лабораторная работа №7

Тупиковые ситуации и подходы к их разрешению

Цель работы: Изучить причины возникновения тупиковых ситуаций и подходов к их разрешению.

1. Теоретические сведения

1.1 Тупики

Предположим, что несколько процессов конкурируют за обладание конечным числом ресурсов. Если запрашиваемый процессом ресурс недоступен, ОС переводит данный процесс в состояние ожидания. В случае когда требуемый ресурс удерживается другим ожидающим процессом, первый процесс не сможет сменить свое состояние. Такая ситуация называется тупиком (deadlock). Говорят, что в мультипрограммной системе процесс находится в состоянии тупика, если он ожидает события, которое никогда не произойдет. Системная тупиковая ситуация, или "зависание системы", является следствием того, что один или более процессов находятся в состоянии тупика. Иногда подобные ситуации называют взаимоблокировками. В общем случае проблема тупиков эффективного решения не имеет.

Рассмотрим пример. Предположим, что два процесса осуществляют вывод с ленты на принтер. Один из них успел монополизировать ленту и претендует на принтер, а другой наоборот. После этого оба процесса оказываются заблокированными в ожидании второго ресурса (см. рис. 1.1).

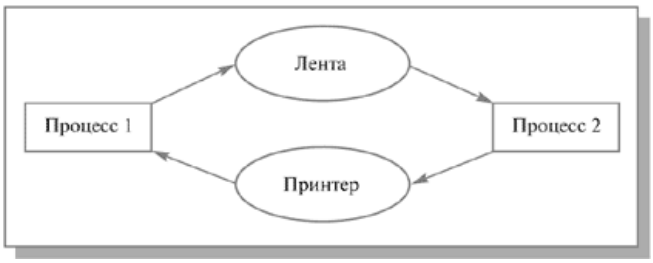

Рис. 1.1. Пример тупиковой ситуации

Определение. Множество процессов находится в тупиковой ситуации, если каждый процесс из множества ожидает события, которое может вызвать только другой процесс данного множества. Так как все процессы чего-то ожидают, то ни один из них не сможет инициировать событие, которое разбудило бы другого члена множества и, следовательно, все процессы будут спать вместе.

Выше приведен пример взаимоблокировки, возникающей при работе с так называемыми выделенными устройствами. Тупики, однако, могут иметь место и в других ситуациях. В этом случае может получиться так, что один из процессов заблокировал записи, необходимые другому процессу, и наоборот. Таким образом, тупики могут иметь место как на аппаратных, так и на программных ресурсах.

Тупики также могут быть вызваны ошибками программирования. Например, процесс может напрасно ждать открытия семафора, потому что в некорректно написанном приложении эту операцию забыли предусмотреть. Другой причиной бесконечного ожидания может быть дискриминационная политика по отношению к некоторым процессам. Однако чаще всего событие, которого ждет процесс в тупиковой ситуации, –

освобождение ресурса, поэтому в дальнейшем будут рассмотрены методы борьбы с тупиками ресурсного типа.
Ресурсами могут быть как устройства, так и данные. Некоторые ресурсы допускают разделение между процессами, то есть являются разделяемыми ресурсами. Например, память, процессор, диски коллективно используются процессами. Другие не допускают разделения, то есть являются выделенными, например лентопротяжное устройство. К взаимоблокировке может привести использование как выделенных, так и разделяемых ресурсов. Например, чтение с разделяемого диска может одновременно осуществляться несколькими процессами, тогда как запись предполагает исключительный доступ к данным на диске. Можно считать, что часть диска, куда происходит запись, выделена конкретному процессу. Поэтому в дальнейшем мы будем исходить из предположения, что тупики связаны с выделенными ресурсами , то есть тупики возникают, когда процессу предоставляется эксклюзивный доступ к устройствам, файлам и другим ресурсам.
Традиционная последовательность событий при работе с ресурсом состоит из запроса, использования и освобождения ресурса. Тип запроса зависит от природы ресурса и от ОС. Запрос может быть явным, например специальный вызов request, или неявным – open для открытия файла. Обычно, если ресурс занят и запрос отклонен, запрашивающий процесс переходит в состояние ожидания.
Далее в данной лекции будут рассматриваться вопросы обнаружения, предотвращения, обхода тупиков и восстановления после тупиков. Как правило, борьба с тупиками – очень дорогостоящее мероприятие. Тем не менее для ряда систем, например для систем реального времени, иного выхода нет.

1.2 Основные направления исследований по проблеме тупиков

В связи с проблемой тупиков были проведены одни из наиболее интересных исследований в области информатики и операционных систем. Результаты этих исследований оказались весьма эффективными в том смысле, что позволили разработать четкие методы решения многих распространенных проблем. В исследованиях по проблеме тупиков можно выделить следующие четыре основные направления:

1. предотвращение тупиков;
2. обход тупиков;
3. обнаружение тупиков;
4. восстановление после тупиков.

При *предотвращении тупиков* целью является обеспечение условий, исключающих возможность возникновения тупиковых ситуаций. Такой подход является вполне корректным решением в том, что касается самого тупика, однако он часто приводит к нерациональному использованию ресурсов. Тем не менее различные методы предотвращения тупиков широко применяются в практике разработчиков.

Цель средств обхода тупиков заключается в том, чтобы можно было предусматривать менее жесткие ограничения, чем в случае предотвращения тупиков, и тем самым обеспечить лучшее использование ресурсов. При наличии средств обхода тупиков не требуется такой реализации системы, при которой опасность тупиковых ситуаций даже не возникает. Напротив, методы обхода тупиков учитывают подобную возможность, однако в случае увеличения вероятности конкретной тупиковой ситуации здесь принимаются меры по аккуратному обходу тупика. (См. обсуждение алгоритма банкира, который предложил Дейкстра, ниже в настоящей главе.)

Методы *обнаружения* тупиков применяются в системах, которые допускают возможность возникновения тупиковых ситуаций как следствие либо умышленных, либо неумышленных действий программистов. Цель средств обнаружения тупиков — установить сам факт возникновения тупиковой ситуации, причем точно определить те процессы и ресурсы, которые оказались вовлеченными в данную тупиковую ситуацию.

После того как все это сделано, данную тупиковую ситуацию в системе можно будет устранить.

Методы *восстановления* после тупиков применяются для устранения тупиковых ситуаций, с тем чтобы система могла продолжать работать, а процессы, попавшие в тупиковую ситуацию, могли завершиться с освобождением занимаемых ими ресурсов. Восстановление — это, мягко говоря, весьма серьезная проблема, так что в большинстве систем оно осуществляется путем полного выведения из решения одного или более процессов, попавших в тупиковую ситуацию. Затем выведенные процессы обычно перезапускаются с самого начала, так что почти вся или даже вся работа, проделанная этими процессами, теряется.

1.3 Предотвращение тупиков

До сих пор разработчики систем при решении проблемы тупиков чаще всего шли по пути исключения самих возможностей тупиков. В настоящем разделе рассматриваются различные методы предотвращения тупиков, а также оцениваются различные последствия их реализации как для пользователя, так и для систем, особенно с точки зрения эксплуатационных характеристик и эффективности работы.

Хавендер показал, что возникновение тупика невозможно, если нарушено хотя бы одно из указанных выше четырех необходимых условий. Он предложил следующую стратегию.

1. Каждый процесс должен запрашивать все требуемые ему ресурсы сразу, причем не может начать выполняться до тех пор, пока все они не будут ему предоставлены.
2. Если процесс, удерживающий определенные ресурсы, получает отказ в удовлетворении запроса на дополнительные ресурсы, этот процесс должен освободить свои первоначальные ресурсы и при необходимости запросить их снова вместе с дополнительными.
3. Введение линейной упорядоченности по типам ресурсов для всех процессов; другими словами, если процессу выделены ресурсы данного типа, в дальнейшем он может запросить только ресурсы более далеких по порядку типов.

Отметим, что Хавендер предлагает три стратегических принципа, а не четыре. Каждый из указанных принципов, как мы увидим в следующих разделах, имеет целью нарушить какое-нибудь одно из необходимых условий существования тупика. Первое необходимое условие, а именно условие взаимоисключения, согласно которому процессы получают право на монопольное управление выделяемыми им ресурсами, мы не хотим нарушать, поскольку нам, в частности, нужно предусмотреть возможность работы с *закрепленными* ресурсами.

1.4 Нарушение условия «ожидания дополнительных ресурсов»

Первый стратегический принцип Хавендера требует, чтобы процесс сразу же запрашивал все ресурсы, которые ему понадобятся. Система должна предоставлять эти ресурсы по принципу «все или ничего». Если набор ресурсов, необходимый процессу, имеется, то система может предоставить все эти ресурсы данному процессу сразу же, так что он получит возможность продолжить работу. Если в текущий момент полного набора ресурсов, необходимых процессу, нет, этому процессу придется ждать, пока они не освободятся. Однако, когда процесс находится в состоянии ожидания, он не должен удерживать какие-либо ресурсы. Благодаря этому предотвращается возникновение условия «ожидания дополнительных ресурсов» и тупиковые ситуации просто невозможны.

На первый взгляд это звучит неплохо, однако подобный подход может привести к серьезному снижению эффективности использования ресурсов. Например, программа, которой в какой-то момент работы требуются 10 лентопротяжных устройств, должна запросить — и получить — все 10 устройств, прежде чем начать выполнение. Если все 10

устройств необходимы программе в течение всего времени выполнения, то это не приведет к серьезным потерям. Рассмотрим, однако, случай, когда программе для начала работы требуется только одно устройство (или, что еще хуже, ни одного), а оставшиеся устройства потребуются лишь через несколько часов. Таким образом, требование о том, что программа должна запросить и получить все эти устройства, чтобы начать выполнение, на практике означает, что значительная часть ресурсов компьютера будет использоваться вхолостую в течение нескольких часов.

Один из подходов, к которому часто прибегают разработчики систем с целью улучшения использования ресурсов в подобных обстоятельствах, заключается в том, чтобы разделять программу на несколько программных шагов, выполняемых относительно независимо друг от друга. Благодаря этому можно осуществлять выделение ресурсов для каждого шага программы, а не для целого процесса. Это позволяет уменьшить непроизводительное использование ресурсов, однако связано с увеличением накладных расходов на этапе проектирования прикладных программ, а также на этапе их выполнения.

Такая стратегия существенно повышает вероятность бесконечного откладывания, поскольку не все требуемые ресурсы могут оказаться свободными в нужный момент. Вычислительная система должна позволить завершиться достаточно большому числу заданий и освободить их ресурсы, чтобы ожидающее задание смогло продолжить выполнение. В течение периода, когда требуемые ресурсы накапливаются, их нельзя выделять другим заданиям, так что эти ресурсы простаивают. В настоящее время существуют противоречивые мнения о том, кто должен платить за эти неиспользуемые ресурсы. Некоторые разработчики считают, что, поскольку ресурсы накапливаются для конкретного пользователя, этот пользователь должен платить за них, даже в течение периода простоя ресурсов. Однако другие разработчики считают, что это привело бы к нарушению принципа *предсказуемости платы за ресурсы,* если пользователь попытается выполнить свою работу в период пиковой загрузки машины, ему придется платить гораздо больше, чем в случае, когда машина относительно свободна.

1.5 Нарушение условия неперераспределяемости

Второй стратегический принцип Хавендера, рассматриваемый здесь независимо от других, предотвращает возникновение условия неперераспределяемости. Предположим, что система позволяет процессам, запрашивающим дополнительные ресурсы, удерживать за собой ранее выделенные ресурсы. Если у системы достаточно свободных ресурсов, чтобы удовлетворять все запросы, тупиковая ситуация не возникнет. Посмотрим, однако, что произойдет, когда какой-то запрос удовлетворить не удастся. В этом случае один процесс удерживает ресурсы, которые могут потребоваться второму процессу для продолжения работы, а этот второй процесс может удерживать ресурсы, необходимые первому. Это и есть тупик.

Второй стратегический принцип Хавендера требует, чтобы процесс, имеющий в своем распоряжении некоторые ресурсы, если он получает отказ на запрос о выделении дополнительных ресурсов, должен освобождать свои ресурсы и при необходимости запрашивать их снова вместе с дополнительными. Такая стратегия действенно ведет к нарушению условия неперераспределяемости, подобным образом можно забирать ресурсы у удерживающих их процессов до завершения этих процессов.

Но этот способ предотвращения тупиковых ситуаций также не свободен от недостатков. Если процесс в течение некоторого времени использует определенные ресурсы, а затем освобождает эти ресурсы, он может потерять всю работу, проделанную до данного момента. На первый взгляд такая цена может показаться слишком высокой, однако необходим реальный ответ на вопрос «насколько часто приходится платить такую цену?» Если такая ситуация встречается редко, то можно считать, что в нашем распоряжении имеется относительно недорогой способ предотвращения тупиков. Если, однако, такая ситуация встречается часто, то подобный способ обходится дорого, причем приводит к

печальным результатам, особенно когда не удается вовремя завершить высокоприоритетные или срочные процессы.

Одним из серьезных недостатков такой стратегии является возможность бесконечного откладывания процессов. Выполнение процесса, который многократно запрашивает и освобождает одни и те же ресурсы, может откладываться на неопределенно долгий срок. В этом случае у системы может возникнуть необходимость вывести данный процесс из решения, с тем чтобы другие процессы получили возможность продолжения работы. Нельзя игнорировать и вероятный случай, когда процесс, откладываемый бесконечно, может оказаться не очень «заметным» для системы. В такой ситуации возможно поглощение процессом значительных вычислительных ресурсов и деградация производительности системы.

1.6 Нарушение условия «кругового ожидания»

Третий стратегический принцип Хавендера исключает круговое ожидание. Поскольку всем ресурсам присваиваются уникальные номера и поскольку процессы должны запрашивать ресурсы в порядке возрастания номеров, круговое ожидание возникнуть не может.

Такой стратегический принцип реализован в ряде операционных систем, однако это оказалось сопряжено с определенными трудностями.

Поскольку запрос ресурсов осуществляется в порядке возрастания их номеров и поскольку номера ресурсов назначаются при установке машины и должны «жить» в течение длительных периодов времени (месяцев или даже лет), то в случае введения в машину новых типов ресурсов может возникнуть необходимость переработки существующих программ и систем.

Очевидно, что назначаемые номера ресурсов должны отражать нормальный порядок, в котором большинство заданий фактически используют ресурсы. Для заданий, выполнение которых соответствует этому порядку, можно ожидать высокой эффективности. Если, однако, заданию требуются ресурсы не в том порядке, который предполагает вычислительная система, то оно должно будет захватывать и удерживать ресурсы, быть может, достаточно долго еще до того, как они будут фактически использоваться. А это означает потерю эффективности.

Одной из самых важных задач современных операционных систем является предоставление пользователю максимальных удобств для работы («дружественной» обстановки). Пользователи должны иметь возможность разрабатывать свои прикладные программы при минимальном «загрязнении своей окружающей среды» из-за ограничений, накладываемых недостаточно удачными аппаратными и программными средствами. Последовательный порядок заказа ресурсов, предложенный Хавендером, действительно исключает возможность возникновения кругового ожидания, однако безусловно отрицательно сказывается при этом на возможности пользователя свободно и легко писать прикладные программы.

1.7 Предотвращение тупиков и алгоритм банкира

Если даже необходимые условия возникновения тупиков не нарушены, то все же можно избежать тупиковой ситуации, если рационально распределять ресурсы, придерживаясь определенных правил. По-видимому, наиболее известным алгоритмом обхода тупиковых ситуаций является алгоритм банкира, который предложил Дейкстра; алгоритм получил такое любопытное наименование потому, что он как бы имитирует действия банкира, который, располагая определенным источником капитала, выдает ссуды и принимает платежи. Ниже мы изложим этот алгоритм применительно к проблеме распределения ресурсов в операционных системах.

1.7.1 Алгоритм банкира, предложенный Дейкстрой

Когда при описании алгоритма банкира мы будем говорить о ресурсах, мы будем подразумевать ресурсы одного и того же типа, однако этот алгоритм можно легко распространить на пулы ресурсов нескольких различных типов. Рассмотрим, например, проблему распределения некоторого количества *t* идентичных лентопротяжных устройств. Операционная система должна обеспечить распределение некоторого фиксированного числа *t* одинаковых лентопротяжных устройств между некоторым фиксированным числом пользователей *u.* Каждый пользователь заранее указывает максимальное число устройств, которые ему потребуются во время выполнения своей программы на машине. Операционная система примет запрос пользователя в случае, если максимальная потребность этого пользователя в лентопротяжных устройствах не превышает *t.*

Пользователь может занимать или освобождать устройства по одному. Возможно, что иногда пользователю придется ждать выделения дополнительного устройства, однако операционная система гарантирует, что ожидание будет конечным. Текущее число устройств, выделенных пользователю, никогда не превысит указанную максимальную потребность этого пользователя. Если операционная система в состоянии удовлетворить максимальную потребность пользователя в устройствах, то пользователь гарантирует, что эти устройства будут использованы и возвращены операционной системе в течение конечного периода времени.

Текущее состояние вычислительной машины называется *надежным,* если операционная система может обеспечить всем текущим пользователям завершение их заданий в течение конечного времени. (Здесь опять-таки предполагается, что лентопротяжные устройства являются единственными ресурсами, которые запрашивают пользователи.) В противном случае текущее состояние системы называется ненадежным.

Предположим теперь, что работают *n* пользователей.

Пусть *l(i)* представляет текущее количество лентопротяжных устройств, выделенных *i* пользователю. Если, например, пользователю 5 выделены четыре устройства, то *l*(5)=4. Пусть *m(i)* — максимальная потребность пользователя *i*, так что если пользователь 3 имеет максимальную потребность в двух устройствах, то *m*(3)=2. Пусть *c(t)* — текущая потребность пользователя, равная его максимальной потребности минус текущее число выделенных ему ресурсов. Если, например, пользователь 7 имеет максимальную потребность в шести лентопротяжных устройствах, а текущее количество выделенных ему устройств составляет четыре, то мы получаем

$$c(7)=m(7) - l(7)=6 - 4=2.$$

В распоряжении операционной системы имеются *t* лентопротяжных устройств. Число устройств, остающихся для распределения, обозначим через *a.* Тогда *a* равно *t* минус суммарное число устройств, выделенных различным пользователям.

Алгоритм банкира, который предложил Дейкстра, говорит о том, что выделять лентопротяжные устройства пользователям можно только в случае, когда после очередного выделения устройств состояние системы остается надежным. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние — это состояние, которое может со временем привести к тупику.

1.7.2 Пример надежного состояния

Предположим, что в системе имеются двенадцать одинаковых лентопротяжных устройств, причем эти накопители распределяются между тремя пользователями, как показывает состояние I.

Состояние I

	Текущее количество выделенных устройств		Максимальная

			потребность
Пользователь (1)	1		4
Пользователь (2)	4		6
Пользователь (3)	5		8
Резерв		2	

Это состояние «надежное», поскольку оно дает возможность всем трем пользователям закончить работу. Отметим, что в текущий момент пользователь (2) имеет четыре выделенных ему устройства и со временем потребует максимум шесть, т. е. два дополнительных устройства. В распоряжении системы имеются двенадцать устройств, из которых десять в настоящий момент в работе, а два — в резерве. Если эти два резервных устройства, имеющихся в наличии, выделить пользователю (2), удовлетворяя тем самым максимальную потребность этого пользователя, то он сможет продолжать работу до завершения. После завершения пользователь (2) освободит все шесть своих устройств так что система сможет выделить их пользователю (1) и пользователю (3). Пользователь (1) имеет одно устройство и со временем ему потребуется еще три. У пользователя (3) — пять устройств и со временем ему потребуется еще три. Если пользователь (2) возвращает шесть накопителей, то три из них можно выделить пользователю (1), который получит таким образом возможность закончить работу и затем возвратить четыре накопителя системе. После этого система может выделить три накопителя пользователю (3), который тем самым также получает возможность закончить работу. Таким образом, основной критерий надежного состояния — это существование последовательности действий, позволяющей всем пользователям закончить работу.

1.7.3 Пример ненадежного состояния

Предположим, что двенадцать лентопротяжных устройств, имеющихся в системе, распределены согласно состоянию II.

Состояние II

	Текущее количество выделенных устройств		Максимальная потребность
Пользователь (1)	8		10
Пользователь (2)	2		5
Пользователь (3)	1		3
Резерв		1	

Здесь из двенадцати устройств системы одиннадцать в настоящий момент находятся в работе и только одно остается в резерве. В подобной ситуации независимо от того, какой пользователь запросит это резервное устройство, мы не можем гарантировать, что все три пользователя закончат работу. И действительно, предположим, что пользователь (1) запрашивает и получает последнее оставшееся устройство. При этом тупиковая ситуация может возникать в трех случаях, когда каждому из трех процессов потребуется запросить по крайней мере одно дополнительное устройство, не возвратив некоторое количество устройств в общий пул ресурсов.

Здесь важно отметить, что термин *«ненадежное состояние'!) не. предполагает, что в данный момент существует или в какое-то время обязательно возникнет тупиковая ситуация. Он просто говорит о том, что в случае некоторой неблагоприятной последовательности событий система может зайти в тупик.*

1.7.4 Пример перехода из надежного состояния в ненадежное

Если известно, что данное состояние надежно, это вовсе не означает, что все последующие состояния также будут надежными. Наш механизм распределения ресурсов должен тщательно анализировать все запросы на выделение ресурсов, прежде чем удовлетворять их. Рассмотрим, например, случай, когда система находится в текущем состоянии, показанном как состояние III.

Предположим теперь, что пользователь (3) запрашивает дополнительный ресурс. Если бы система удовлетворила этот запрос, она перешла бы в новое состояние, состояние IV.

Конечно, состояние IV не обязательно приведет к тупиковой ситуации. Если, однако, состояние III было надежным, то состояние IV стало ненадежным. Состояние IV характеризует систему,

Состояние III

	Текущее количество выделенных устройств		Максимальная потребность
Пользователь (1)	1		4
Пользователь (2)	4		6
Пользователь (3)	5		8
Резерв		2	

Состояние IV

	Текущее количество выделенных устройств		Максимальная потребность
Пользователь (1)	1		4
Пользователь (2)	4		6
Пользователь (3)	6		8
Резерв		1	

в которой нельзя гарантировать успешное завершение всех процессов пользователей. В резерве здесь остается только один ресурс, в то время как в наличии должно быть минимум два ресурса, чтобы либо пользователь (2), либо пользователь (3) могли завершить свою работу, возвратить занимаемые ими ресурсы системе и тем самым позволить остальным пользователям закончить выполнение.

1.7.5 Распределение ресурсов согласно алгоритму банкира

Сейчас уже должно быть ясно, каким образом осуществляется распределение ресурсов согласно алгоритму банкира, который предложил Дейкстра. Условия «взаимоисключения», «ожидания дополнительных ресурсов» и «неперераспределяемости» выполняются. Процессы могут претендовать на монопольное использование ресурсов, которые им требуются. Процессам реально разрешается удерживать за собой ресурсы, запрашивая и ожидая выделения дополнительных ресурсов, причем ресурсы нельзя отбирать у процесса, которому они выделены. Пользователи не ставят перед системой особенно сложных задач, запрашивая в каждый момент времени только один ресурс. Система может либо удовлетворить, либо отклонить каждый запрос. Если запрос отклоняется, пользователь удерживает за собой уже выделенные ему ресурсы и ждет определенный конечный период времени, пока этот запрос в конце концов не будет удовлетворен. *Система удовлетворяет только те запросы, при которых ее состояние остается надежным.* Запрос пользователя, приводящий к переходу системы в ненадежное состояние, откладывается до момента, когда его все же можно будет выполнить. Таким образом, поскольку система всегда поддерживается в надежном состоянии, рано или поздно (т. е. в течение конечного

времени) все запросы можно будет удовлетворить и все пользователи смогут завершить свою работу.

1.7.6 Алгоритм банкира для одного вида ресурса

Алгоритм банкира (рисунок 1.2):

1. Банкиру поступает запрос от клиента на получение кредита
2. Банкир проверяет, приводит ли этот запрос к небезопасному состоянию.
3. Банкир в зависимости от этого дает или отказывает в кредите.

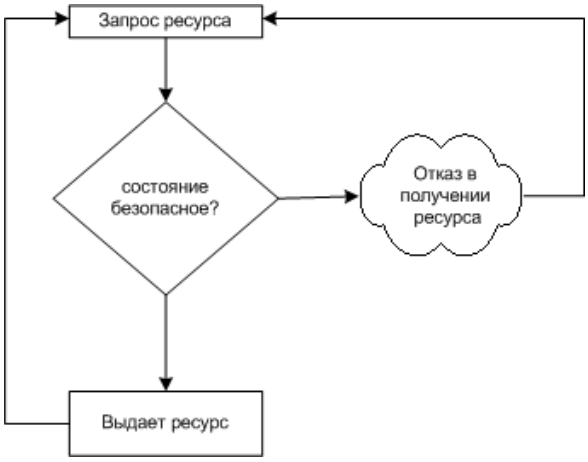

Рисунок 1.2 Алгоритм банкира для одного вида ресурса

1.7.7 Алгоритм банкира для нескольких видов ресурсов. Алгоритм поиска безопасного или небезопасного состояния.

Рассмотрим систему.

m - число классов ресурсов (например: принтеры это один класс);

n - количество процессов;

P(n) – процессы;

E - вектор существующих ресурсов;

E(i) - количество ресурсов класса I;

A - вектор доступных (свободных) ресурсов;

A(i) - количество доступных ресурсов класса I;

C - матрица текущего распределения (какому процессу, какие ресурсы принадлежат);

R - матрица запросов (какой процесс, какой ресурс запросил);

C(i, j) - количество экземпляров ресурса j, которое занимает процесс P(i);

R(i, j) - количество экземпляров ресурса j, которое максимум хочет получить процесс P(i).

Существующие ресурсы

$$(E_1 \quad E_2 \quad E_3 \quad \dots \quad E_m)$$

Доступные ресурсы

$$(A_1 \quad A_2 \quad A_3 \quad \dots \quad A_m)$$

Матрица текущего распределения

$$\begin{bmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{bmatrix}$$

Строка **n** предоставлена процессу **n**

Матрица запроса

$$\begin{bmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ R_{21} & R_{22} & \dots & R_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{n1} & R_{n2} & \dots & R_{nm} \end{bmatrix}$$

Строка **n** нужна процессу **n**

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Алгоритм обнаружения тупиков состоит из следующих шагов:

1. Ищем немаркированный процесс P_i для которого i-я строка матрицы R меньше вектора A или равна ему.
2. Если такой процесс найден, прибавляем f-ю строку матрицы C к вектору A, маркируем процесс и возвращаемся к шагу 1.
3. Если таких процессов не существует, работа алгоритма заканчивается.

Завершение алгоритма означает, что все немаркированные процессы, если такие есть, попали в тупик.

Рисунок 1.3 Алгоритм банкира для несколько видов ресурсов

На первом шаге алгоритм ищет процесс, который может доработать до конца. Такой процесс характеризуется тем, что все требуемые для него ресурсы должны находиться среди доступных в данный момент ресурсов. Тогда выбранный процесс проработает до своего завершения и после этого вернет ресурсы, которые он занимал, в общий фонд доступных ресурсов. Затем процесс маркируется как законченный. Если окажется, что все процессы могут работать, тогда ни один из них в данный момент не заблокирован. Если

некоторые из них никогда не смогут запуститься, значит, они попали в тупик. Несмотря на то что алгоритм не является детерминированным (поскольку он может просматривать процессы в любом допустимом порядке), результат всегда одинаков.
Для иллюстрации работы алгоритма обнаружения тупиков рассмотрим рис. 1.4. Здесь у нас есть три процесса и четыре класса ресурсов, которые мы произвольно назвали так: накопители на магнитной ленте, плоттеры, сканеры и устройство для чтения компакт-дисков. Процесс 1 использует один сканер. Процесс 2 занял два накопителя на магнитной ленте и устройство для чтения компакт-дисков. Процесс 3 занимает плоттер и два сканера. Каждый процесс нуждается в дополнительном устройстве, как показывает матрица *R*. Работая с алгоритмом обнаружения взаимоблокировок, мы ищем процесс, чей запрос ресурсов может быть удовлетворен в данной системе. Требования первого процесса нельзя выполнить, потому что в системе нет доступного устройства для чтения компакт-дисков. Второй запрос также нельзя удовлетворить, так как нет свободных сканеров. К счастью, третий процесс может получить все желаемое, следовательно, он работает, завершается и возвращает все свои ресурсы, давая:

$$A = (2\ 2\ 2\ 0).$$

С этого момента может выполняться процесс 2, по окончании возвращая свои ресурсы в систему. Мы получим:

$$A = (4\ 2\ 2\ 1).$$

Теперь может работать оставшийся процесс. В этой системе не возникает взаимоблокировки.

Рисунок 1.4 Пример использования алгоритма обнаружения тупиков

1.7.8 Недостатки алгоритма банкира

Алгоритм банкира представляет для нас интерес потому, что он дает возможность распределять ресурсы таким образом, чтобы обходить тупиковые ситуации. Он позволяет продолжать выполнение таких процессов, которым в случае системы с предотвращением тупиков пришлось бы ждать. Однако у этого алгоритма имеется ряд серьезных недостатков, из-за которых разработчик системы может оказаться вынужденным выбрать другой подход к решению проблемы тупиков.

1. Алгоритм банкира исходит из фиксированного количества распределяемых ресурсов. Поскольку устройства, представляющие ресурсы, зачастую требуют технического обслуживания либо из-за возникновения неисправностей, либо в целях профилактики, мы не можем считать, что количество ресурсов всегда остается фиксированным.
2. Алгоритм требует, чтобы число работающих пользователей оставалось постоянным. Подобное требование также является нереалистичным. В современных мультипрограммных системах количество работающих пользователей все время меняется. Например, большая система с разделением времени вполне может обслуживать 100 или более пользователей одновременно. Однако текущее число

обслуживаемых пользователей непрерывно меняется, быть может, очень часто, каждые несколько секунд.

3. Алгоритм требует, чтобы банкир—распределитель ресурсов гарантированно удовлетворял все запросы за некоторый конечный период времени. Очевидно, что для реальных систем необходимы гораздо более конкретные гарантии.
4. Аналогично, алгоритм требует, чтобы клиенты (т. е. задания или процессы) гарантированно «платили долги» (т. е. возвращали выделенные им ресурсы) в течение некоторого конечного времени. И опять-таки для реальных систем требуются гораздо более конкретные гарантии.
5. Алгоритм требует, чтобы пользователи заранее указывали свои максимальные потребности в ресурсах. По мере того как распределение ресурсов становится все более динамичным, все труднее оценивать максимальные потребности пользователя. Вообще говоря, поскольку компьютеры становятся более «дружественными» по отношению к пользователю, все чаще встречаются пользователи, которые не имеют ни малейшего представления о том, какие ресурсы им потребуются.

2. Индивидуальные задания

В соответствии с вариантом, выполнить построение последовательности **надежных** состояний системы при удовлетворении запросов на ресурсы в соответствии с алгоритмом «банкира». Исходные данные представлены в таблицах 2.1 и 2.2.

2.1 Задание 1 – Один ресурс

Таблица 2.1 Варианты заданий для одного ресурса

№	Ресурсы	Процесс 1	Процесс 2	Процесс 3	Макс. ресурсов
1	Выдано	1	3	2	9
	Потребность	4	8	6	
2	Выдано	1	2	2	7
	Потребность	3	5	7	
3	Выдано	3	0	5	12
	Потребность	7	4	10	
4	Выдано	1	2	1	7
	Потребность	3	7	5	
5	Выдано	1	2	0	6
	Потребность	5	5	5	
6	Выдано	2	2	1	8
	Потребность	8	3	4	
7	Выдано	3	2	1	9
	Потребность	7	9	3	
8	Выдано	0	3	2	8
	Потребность	1	8	4	
9	Выдано	5	1	2	12
	Потребность	10	5	8	
10	Выдано	3	1	2	10
	Потребность	9	3	6	
11	Выдано	2	1	1	8
	Потребность	7	7	3	
12	Выдано	0	2	1	8
	Потребность	3	8	4	
13	Выдано	5	0	3	12
	Потребность	11	2	7	
14	Выдано	1	3	3	10
	Потребность	5	6	7	
15	Выдано	4	3	2	13
	Потребность	10	10	5	

Пример 1.

№	Ресурсы	Процесс 1	Процесс 2	Процесс 3	Макс. ресурсов
1	Выдано	6	2	1	11
	Потребность	9	14	3	

Решение

Шаг 0

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	6	<	9
Процесс2	2	<	7
Процесс3	1	<	3
Итого	9	2	

Шаг 1

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	6	<	9
Процесс2	2	<	7
Процесс3	1+2	=	3
Итого	11	0	

Шаг 2

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	6	<	9
Процесс2	2	<	7
Процесс3	-		-
Итого	8	3	

Шаг 3

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	6+3	=	9
Процесс2	2	<	7
Процесс3	-		-
Итого	11	0	

Шаг 4

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	-		-
Процесс2	2	<	7
Процесс3	-		-
Итого	2	9	

Шаг 5

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	-		-
Процесс2	2+5	=	7

Процесс3	–		–
Итого	7	4	

Шаг 6

Процессы	Текущее количество выделенного ресурса	Резерв	Максимальная потребность
Процесс1	–		–
Процесс2	–		–
Процесс3	–		–
Итого	0	11	

2.2 Задание 2 – Несколько ресурсов

Максимальное количество ресурсов P1 – 7, P2 – 6. Ресурсы выделяются последовательно (в соответствии со значениями приведенными в скобках).

№	Максимальная потребность (и последовательность запрашиваемых ресурсов)											
	Процесс 1		Процесс 2		Процесс 3		Процесс 4		Процесс 5		Процесс 6	
	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2
1	4(4+0+0)	3(1+1+1)	6(5+0+1)	3(0+0+3)	2(1+0+1)	2(0+0+2)	5(4+0+1)	4(1+1+2)	4(4+0+0)	2(0+1+1)	2(1+0+1)	2(2+0+0)
2	2(2+0+0)	2(0+2+0)	4(4+0+0)	6(5+1+0)	3(0+3+0)	3(0+2+1)	4(0+3+1)	2(1+0+1)	5(1+4+0)	4(3+1+0)	2(0+0+2)	4(0+0+4)
3	3(1+1+1)	2(2+0+0)	6(6+0+0)	3(3+0+0)	4(1+1+2)	4(0+4+0)	2(2+0+0)	3(1+2+0)	2(2+0+0)	2(0+2+0)	5(4+1+0)	3(0+2+1)
4	6(1+5+0)	6(0+5+1)	2(2+0+0)	4(3+1+0)	5(4+1+0)	3(0+3+0)	3(1+0+2)	2(0+2+0)	5(4+1+0)	4(1+3+0)	3(0+3+0)	3(0+0+3)
5	3(1+1+1)	3(1+0+2)	2(0+0+2)	2(0+2+0)	3(3+0+0)	6(4+0+2)	4(1+3+0)	2(0+2+0)	4(4+0+0)	5(5+0+0)	2(0+0+2)	3(2+0+1)
6	2(1+0+1)	3(3+0+0)	4(0+0+4)	5(1+1+3)	5(3+1+1)	2(0+0+2)	2(2+0+0)	2(0+0+2)	3(2+0+1)	2(0+0+2)	3(3+0+0)	4(4+0+0)
7	3(2+1+0)	3(0+1+2)	4(4+0+0)	5(0+4+1)	6(5+0+1)	2(0+2+0)	3(2+0+1)	4(3+0+1)	5(4+0+1)	5(1+0+4)	3(3+0+0)	2(1+0+1)
8	3(2+1+0)	2(1+0+1)	2(0+2+0)	2(2+0+0)	6(3+0+3)	5(2+0+3)	2(1+0+1)	3(3+0+0)	4(1+3+0)	4(3+1+0)	3(2+0+1)	3(1+2+0)
9	4(3+1+0)	2(1+0+1)	4(4+0+0)	4(0+4+0)	5(4+0+1)	6(3+0+3)	5(4+1+0)	3(2+1+0)	2(1+1+0)	5(1+1+3)	2(0+2+0)	2(1+1+0)
10	6(5+1+0)	2(1+0+1)	4(4+0+0)	3(3+0+0)	3(1+1+1)	2(1+0+1)	4(3+1+0)	5(4+1+0)	5(5+0+0)	2(0+1+1)	2(0+0+2)	3(1+1+1)
11	5(4+1+0)	2(2+0+0)	3(1+1+1)	3(0+0+3)	2(1+0+1)	2(0+0+2)	6(3+2+1)	4(1+1+2)	3(3+0+0)	2(0+2+0)	5(3+0+2)	4(4+0+0)
12	6(4+1+1)	3(3+0+0)	2(1+1+0)	2(0+1+1)	5(4+0+1)	6(4+0+2)	4(2+1+1)	2(0+1+1)	2(1+1+0)	2(1+0+1)	4(3+1+0)	5(4+0+1)
13	5(3+2+0)	3(1+1+1)	4(3+1+0)	4(3+0+1)	4(4+0+0)	3(3+0+0)	5(4+0+1)	4(4+0+0)	2(0+2+0)	4(1+1+2)	3(1+1+1)	3(2+1+0)
14	2(1+1+0)	4(1+3+0)	4(1+3+0)	2(1+0+1)	6(3+2+1)	4(3+1+0)	2(1+1+0)	3(1+1+1)	3(1+1+1)	4(1+1+2)	5(4+1+0)	2(2+0+0)
15	5(4+0+1)	2(0+2+0)	2(1+1+0)	2(2+0+0)	5(5+0+0)	2(1+1+0)	4(3+1+0)	2(0+1+1)	6(5+0+1)	2(1+0+1)	5(3+1+1)	4(0+4+0)

Вопросы.

1. Дайте определение понятия тупика. Приведите пример.
2. Сформулируйте условия возникновения тупика.
3. Перечислите подходы к разрешению проблемы тупиков.
4. Алгоритм банкира.
5. Понятие надежных и не надежных состояний.
6. Алгоритм банкира для одного и нескольких видов ресурсов.
7. Недостатки алгоритма банкира