№ занятия: 4. Лекция № 3

Тема: Процессы и потоки.

Введение.

Сегодня мы познакомимся с материалом 3-ей лекции. Тема лекции – “Процессы и потоки”.

Важнейшей функцией ОС является организация рационального использования всех ее аппаратных и информационных ресурсов.

К основным ресурсам могут быть отнесены процессоры, память, внешние устройства, данные и программы. Располагающая одними и теми же аппаратными ресурсами, но управляемая различными ОС, вычислительная система может работать с разной степенью эффективности. Поэтому знание внутренних механизмов операционной системы позволяет косвенно судить о ее эксплуатационных возможностях и характеристиках. Хотя и в однопрограммной ОС необходимо решать задачи управления ресурсами (например, задачу распределения памяти между приложением и ОС), главные сложности на этом пути возникают в мультипрограммных ОС, в которых за ресурсы конкурируют сразу несколько приложений. Именно поэтому большая часть всех проблем, рассматриваемых в этой лекции, относится к мультипрограммным системам.

Мультипрограммирование

Мультипрограммирование, или многозадачность (multitasking), — это такой вариант организации вычислительного процесса, при котором на одном процессоре попеременно выполняются сразу несколько программ.

Попеременно выполняемые программы совместно используют не только процессор, но и другие ресурсы компьютера: оперативную и внешнюю память, устройства ввода-вывода, данные. Мультипрограммирование призвано повысить эффективность использования вычислительной системы, однако эффективность может пониматься по-разному.

Наиболее характерными критериями эффективности вычислительных систем являются:

* *пропускная способность*, то есть количество задач, выполняемых вычислительной системой в единицу времени;
* *удобство работы пользователей*, заключающееся, в частности, в том, что они имеют возможность интерактивно работать одновременно с несколькими приложениями на одной машине;
* *реактивность системы*, то есть способность системы выдерживать заранее заданные (возможно очень короткие) интервалы времени между запуском программы и получением результата.

В зависимости от выбранного критерия эффективности мультипрограммные ОС делятся на

* системы пакетной обработки,
* системы разделения времени,
* системы реального времени.

Каждый тип ОС имеет специфические внутренние механизмы и особые области применения. Некоторые операционные системы могут поддерживать одновременно несколько режимов, например часть задач может выполняться в режиме пакетной обработки, а часть — в режиме реального времени или в режиме разделения времени.

Мультипрограммирование в системах пакетной обработки

При использовании мультипрограммирования для повышения пропускной способности компьютера главной целью является минимизация простоев всех устройств компьютера и, прежде всего, центрального процессора. Такие простои могут возникать из-за приостановки задачи по ее внутренним причинам, связанным, например, с ожиданием ввода данных для обработки. Данные могут храниться на диске или же поступать от пользователя, работающего за терминалом, а также от измерительной аппаратуры, установленной на внешних технических объектах. При такого рода блокировке выполняемой задачи естественным решением, ведущим к повышению эффективности использования процессора, является переключение процессора на выполнение другой задачи, у которой есть данные для обработки. Такая концепция мультипрограммирования положена в основу так называемых пакетных систем.

Системы пакетной обработки предназначены для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность.

Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования:

* в начале работы формируется пакет заданий, каждое задание содержит требование к системным ресурсам;
* из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач.

Для одновременного выполнения выбираются задачи, предъявляющие разные требования к ресурсам так, чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины. Например, в мультипрограммной смеси желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом.

Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание. Следовательно, в вычислительных системах, работающих под управлением пакетных ОС, невозможно гарантировать выполнение того или иного задания в течение определенного периода времени.

Рассмотрим более детально совмещение во времени операций ввода-вывода и вычислений.

Такое совмещение может достигаться разными способами. Один из них характерен для компьютеров, имеющих специализированный процессор вводавывода. В компьютерах класса мэйнфреймов такие процессоры называют каналами. Обычно канал имеет систему команд, отличающуюся от системы команд центрального процессора. Эти команды специально предназначены для управления внешними устройствами, например,

«проверить состояние устройства»,

«установить магнитную головку»,

«установить начало листа»,

«напечатать строку».

Канальные программы могут храниться в той же оперативной памяти, что и программы центрального процессора. В системе команд центрального процессора предусматривается специальная инструкция, с помощью которой канал у передаются параметры и указания на то, какую программу ввода-вывода он должен выполнить. Начиная с этого момента центральный процессор и канал могут работать параллельно (рис. 1, а).

*Рис. 1 . Параллельное выполнение вычислений и операций ввода-вывода*

Другой способ совмещения вычислений с операциями ввода-вывода реализуется в компьютерах, в которых внешние устройства управляются не процессором ввода-вывода, а контроллерами. Каждое внешнее устройство (или группа внешних устройств одного типа) имеет собственный контроллер, который автономно отрабатывает команды, поступающие от центрального процессора. При этом контроллер и центральный процессор работают асинхронно. Поскольку многие внешние устройства имеют электромеханические узлы, контроллер выполняет свои команды управления устройствами существенно медленнее, чем центральный процессор — свои. Это обстоятельство используется для организации параллельного выполнения вычислений и операций ввода-вывода: в промежутке между передачей команд контроллеру центральный процессор может выполнять вычисления (рис. 1, б). Контроллер может сообщить центральному процессору о том, что он готов принять следующую команду, либо сигналом прерывания, либо центральной процессор узнает об этом, периодически опрашивая состояние контроллера.

Максимальный эффект ускорения достигается при наиболее полном перекрытие вычислений и ввода-вывода. Рассмотрим случай, когда процессор выполняет только одну задачу. В этой ситуации степень ускорения зависит от природы данной задачи и от того, насколько тщательно был выявлен возможный параллелизм при ее программировании. В задачах, в которых преобладают либо вычисления, либо ввод-вывод, ускорение почти отсутствует. Параллелизм в рамках одной задачи невозможен также, когда для продолжения вычислений необходимо полное завершение операции ввода-вывода, например, когда дальнейшие вычисления зависят от вводимых данных. В таких случаях неизбежны простои центрального процессора или канала.

Если же в системе выполняется одновременно несколько задач, появляется возможность совмещения вычислений одной задачи с вводом-выводом другой.

Пока одна задача ожидает какого-либо события (заметим, что таким событием в мультипрограммной системе может быть не только завершение ввода-вывода, но и, например, наступление определенного момента времени, разблокирование файла или загрузка с диска недостающей страницы программы), процессор не простаивает, как это происходит при последовательном выполнении программ, а выполняет другую задачу.

Общее время выполнения смеси задач часто оказывается меньше, чем их суммарное время последовательного выполнения (рис. 2, а). Однако выполнение отдельной задачи в мультипрограммном режиме может занять больше времени, чем при монопольном выделении процессора этой задаче. Действительно, при совместном использовании процессора в системе могут возникать ситуации, когда задача не выполняется из-за того, что процессор занят выполнением другой задачи. В таких случаях задача, завершившая ввод-вывод, готова выполняться, но вынуждена ждать освобождения процессора, и это удлиняет срок ее выполнения. Так, из рис. 2 видно, что в однопрограммном режиме задача А выполняется за 6 единиц времени, а в мультипрограммном — за 7. Задача В также вместо 5 единиц времени выполняется за 6. Но зато время выполнения обоих задач в мультипрограммном режиме составляет всего 8 единиц, что на 3 единицы меньше, чем при последовательном выполнении. В системах пакетной обработки переключение процессора с выполнения одной задачи на выполнение другой происходит по инициативе самой активной задачи, например, когда она отказывается от процессора из-за необходимости выполнить операцию ввода-вывода. Поэтому существует высокая вероятность того, что одна задача может надолго занять процессор, и выполнение интерактивных задач станет невозможным. Взаимодействие пользователя с вычислительной машиной, на которой установлена система пакетной обработки, сводится к тому, что он приносит задание, отдает его диспетчеру-оператору, а в конце дня после выполнения всего пакета заданий получает результат. Очевидно, что такой порядок повышает эффективность функционирования аппаратуры, но снижает эффективность работы пользователя.



Рис.2 . Время выполнения двух задач: в однопрограммной системе (а),

в мультипрограммной системе (б).

Мультипрограммирование в системах разделения времени

Система разделения времени — это такая форма организации вычислительного процесса, при которой сразу несколько пользователей одновременно работают на компьютере, причем каждому из них кажется, что он получил компьютер в полное свое распоряжение. Главной целью и критерием эффективности систем разделения времени является обеспечение удобства и эффективности работы пользователей.

Другими словами, в системах разделения времени пользователям (в частном случае — одному пользователю) предоставляется возможность интерактивной работы сразу с несколькими приложениями. Для этого каждое приложение должно регулярно получать возможность «общения» с пользователем. Понятно, что в пакетных системах возможности диалога пользователя с приложением весьма ограничены.

В системах разделения времени эта проблема решается за счет того, что ОС принудительно периодически приостанавливает приложения, не дожидаясь, когда они добровольно освободят процессор. Всем приложениям попеременно выделяется квант процессорного времени, таким образом, пользователи, запустившие программы на выполнение, получают возможность поддерживать с ними диалог.

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки — изоляцию пользователи-программиста от процесса выполнения его задач. Каждому пользователю в этом случае предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину.

Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая «выгодна» системе. Кроме того, производительность системы снижается из-за возросших накладных расходов вычислительной мощности на более частое переключение процессора с задачи на задачу. Это вполне соответствует тому, что критерием эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя. Тем не менее мультипрограммное выполнение интерактивных приложений повышает пропускную способность компьютера (пусть и не в такой степени, как пакетные системы) по сравнению с однопрограммной обработкой. Аппаратура загружается лучше, поскольку в то время, пока одно приложение ждет сообщения пользователя, процессор может заняться другими приложениями.

Мультипрограммирование в системах реального времени

Еще одна разновидность мультипрограммирования используется в системах реального времени, предназначенных для управления от компьютера различными техническими объектами (например, станком, спутником, научной экспериментальной установкой) или технологическими процессами (например, доменным процессом, гальванической линией).

Управляющая система должна собирать информацию о состоянии управляемого объекта, которое в общем случае изменяется непредсказуемым образом.

При наступлении того или иного события, например, когда температура раствора в гальванической ванне достигает заданного уровня, система должна выдавать соответствующее управляющее воздействие на исполнительный механизм, например нагревательный элемент. Очень важно, что во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная управляющая объектом программа.

В противном случае может произойти авария: спутник выйдет из зоны видимости, экспериментальные данные, поступающие с датчиков, будут потеряны, толщина гальванического покрытия не будет соответствовать норме. Таким образом, критерием эффективности здесь является способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы — реактивностью. Требования ко времени реакции зависят от специфики управляемого процесса. Контроллер робота может требовать от встроенного компьютера ответ в течение менее 1 мс, в то время как при моделировании полета ответ даже в 40 мс может быть вполне приемлемым. Таким образом, мы пришли к следующему определению.

Операционная система реального времени — это система, предназначенная для управления физическими объектами (процессами), которая способна обеспечить предсказуемое время реакции в ответ на изменение состояния управляемого объекта (процесса).

В системах реального времени мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ, а выбор программы на выполнение осуществляется по прерываниям (исходя из текущего состояния объекта) или в соответствии с расписанием плановых работ.

Способность аппаратуры компьютера и ОС к быстрому ответу зависит в основном от скорости переключения с одной задачи на другую и, в частности, от скорости обработки сигналов прерывания. Если при возникновении прерывания процессор должен опросить сотни потенциальных источников прерывания, то реакция системы будет слишком медленной. Время обработки прерывания в системах реального времени часто определяет требования к классу процессора даже при небольшой его загрузке.

В системах реального времени не стремятся максимально загружать все устройства, наоборот, при проектировании программного управляющего комплекса обычно закладывается некоторый - «запас» вычислительной мощности на случай пиковой нагрузки. Статистические аргументы о низкой вероятности возникновения пиковой нагрузки, основанные на том, что вероятность одновременного возникновения большого количества независимых событий очень мала, ко многим ситуациям в системах управления неприменимы.

Например, в системе управления атомной электростанцией в случае возникновения крупной аварии атомного реактора многие аварийные датчики сработают одновременно и создадут коррелированную нагрузку. Если система реального времени не спроектирована для поддержки пиковой нагрузки, то может случиться так, что система не справится с работой именно тогда, когда она нужна в наибольшей степени.

Вопрос 1. Подсистема управления процессами и потоками

Одной из основных подсистем мультипрограммной ОС, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами и потоками, которая занимается их созданием и уничтожением, поддерживает взаимодействие между ними, а также распределяет процессорное время между несколькими одновременно существующими в системе процессами и потоками.

Подсистема управления процессами и потоками ответственна за обеспечение процессов необходимыми ресурсами. ОС поддерживает в памяти специальные информационные структуры, в которые записывает, какие ресурсы выделены каждому процессу. Она может назначить процессу ресурсы в единоличное или совместное с другими процессами пользование. Некоторые из ресурсов выделяются процессу при его создании, другие — динамически по запросам во время выполнения. Ресурсы могут быть приписаны процессу на все время его жизни или только на определенный период. При выполнении этих функций подсистема управления процессами взаимодействует с другими подсистемами ОС, ответственными за управление ресурсами, такими как подсистема управления памятью, подсистема ввода-вывода, файловая система.

Когда в системе одновременно выполняется несколько независимых задач, то возникают дополнительные проблемы. Хотя потоки появляются и выполняются асинхронно, у них может возникнуть необходимость во взаимодействии, например, при обмене данными. Согласование скоростей потоков также очень важно для предотвращения эффекта «гонок» (когда несколько потоков пытаются изменить один и тот же файл), взаимных блокировок или других коллизий, которые возникают при совместном использовании ресурсов. Синхронизация потоков является одной из важных функций подсистемы управления процессами и потоками.

Каждый раз, когда процесс завершается, ОС предпринимает шаги, чтобы «зачистить следы» его пребывания в системе. Подсистема управления процессами закрывает все файлы, с которыми работал процесс, освобождает области оперативной памяти, отведенные под коды, данные и системные информационные структуры процесса. Выполняется коррекция всевозможных очередей ОС и списков ресурсов, в которых имелись ссылки на завершаемый процесс.

Чтобы поддерживать мультипрограммирование, ОС должна определить и оформить для себя те внутренние единицы работы, между которыми будет разделяться процессор и другие ресурсы компьютера. В настоящее время в большинстве операционных систем определены два типа единиц работы. Более крупная единица работы, обычно носящая название процесс, или задача, требует для своего выполнения нескольких более мелких работ, для обозначения которых используют термины поток, или нить. Итак, в чем же состоят принципиальные отличия в понятиях «процесс» и «поток»?

В операционных системах, где существуют и процессы, и потоки, процесс рассматривается операционной системой как заявка на потребление всех видов ресурсов кроме одного – процессорного времени. Этот последний важнейший ресурс распределяется операционной системой между другими единицами работы – потоками, которые и получили свое название благодаря тому, что они представляют собой последовательности (потоки выполнения) команд.

В простейшем случае процесс состоит из одного потока, и именно таким образом трактовалось понятие «процесс» до середины 1980-х годов (например, в ранних версиях *Unix*) и в таком же виде оно сохранилось в некоторых современных ОС. В таких системах понятие «поток» полностью поглощается понятием «процесс», то есть остается только одна единица работы и потребления ресурсов – процесс. Мультипрограммирование осуществляется в таких ОС на уровне процессов.

Для того чтобы процессы не могли вмешаться в распределение ресурсов, а также не могли повредить коды и данные друг друга, важнейшей задачей ОС является изоляция одного процесса от другого. Для этого операционная система обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса.

Виртуальное адресное пространство процесса – это совокупность адресов, которыми может манипулировать программный модуль процесса. Операционная система отображает виртуальное адресное пространство процесса на отведенную процессу физическую память.

Содержимое назначенного процессу виртуального адресного пространства, то есть коды команд, исходные и промежуточные данные, а также результаты вычислений, называют образом процесса. При необходимости взаимодействия процессы обращаются к операционной системе, которая, выполняя функции посредника, предоставляет им средства межпроцессной связи – конвейеры, почтовые ящики, разделяемые секции памяти и некоторые другие.

Однако в системах, в которых отсутствует понятие потока, возникают проблемы при организации параллельных вычислений в рамках процесса. Потоки возникли в операционных системах как средство распараллеливания вычислений. Конечно, задача распараллеливания вычислений в рамках одного приложения может быть решена и традиционными способами.

*Во-первых*, прикладной программист может взять на себя сложную задачу организации параллелизма, выделив в приложении некоторую под программу диспетчер, которая периодически передает управление той или иной ветви вычислений. При этом программа получается логически весьма запутанной с многочисленными передачами управления, что существенно затрудняет ее отладку и модификацию.

*Во-вторых*, решением является создание в рамках одного приложения нескольких процессов для каждой из параллельных работ. Однако использование для создания процессов стандартных средств ОС не позволяет учесть тот факт, что эти процессы решают единую задачу, а значит, имеют много общего между собой – они могут работать с одними и теми же данными, использовать один и тот же кодовый сегмент, наделяться одними и теми же правами доступа к ресурсам вычислительной системы.

Многопоточная обработка (*multithreading*) представляет собой механизм распараллеливания вычислений, который учитывает тесные связи между отдельными ветвями вычислений одного и того же приложения.

При этом вводится новая единица работы – поток выполнения, а понятие «процесс» в значительной степени меняет смысл. Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой команде. ОС распределяет процессорное время между потоками. Процессу ОС назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем создание процессов. В отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в, гораздо меньшей степени, нежели процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти одно и тоже адресное пространство.

Итак, мультипрограммирование более эффективно на уровне потоков, а не процессов. Каждый поток имеет собственные счетчик команд и стек. Задача, оформленная в виде нескольких потоков в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного выполнения ее отдельных частей. Использование потоков связано не только со стремлением повысить производительность системы за счет параллельных вычислений, но и с целью создания более читабельных, логичных программ. Введение нескольких потоков выполнения упрощает программирование.

Поскольку они разделяют общий буфер, не стоит их делать отдельными процессами. Таким образом, применение потоков может сократить потребность в прерываниях пользовательского уровня. Наибольший эффект от введения многопоточной обработки достигается в мультипроцессорных системах, в которых потоки, в том числе принадлежащие одному процессу, могут выполняться на разных процессорах действительно параллельно, а не псевдопараллельно.

**Создание процессов и потоков**. Создать процесс – это, прежде всего, означает создать описатель процесса, в качестве которого выступает одна или несколько информационных структур, содержащих все сведения о процессе, необходимые операционной системе для управления им. В число таких сведений могут входить, например, идентификатор процесса, данные о расположении в памяти исполняемого модуля, степень привилегированности процесса (приоритет и права доступа) и т. п. Примерами описателей процесса являются блок управления задачей, управляющий блок процесса, дескриптор процесса, объект-процесс.

*Дескриптор процесса* – это информационная структура, содержащая информацию о процессе. Эта информация необходима ядру ОС в течение всего жизненного цикла процесса независимо от того, находится он в активном или пассивном состоянии, загружен образ процесса в оперативную память или вытеснен на диск. Дескрипторы отдельных процессов объединены в список, образующий таблицу процессов. Память для таблицы процессов отводится динамически в области ядра. На основании информации, содержащейся в таблице процессов, операционная система осуществляет планирование и синхронизацию процессов.

Контекст процесса содержит менее оперативную, но более объемную часть информации о процессе, необходимую операционной системе для возобновления выполнения процесса с прерванного места.

Контекст включает содержимое регистров процессора, коды ошибок выполняемых процессором системных вызовов, информацию обо всех открытых данным процессом файлах и незавершенных операциях ввода-вывода и другие данные, характеризующие состояние вычислительной среды в момент прерывания.

**Планирование и диспетчеризация потоков**. На протяжении существования процесса выполнение его потоков может быть многократно прервано и продолжено. Переход от выполнения одного потока к выполнению другого осуществляется в результате планирования и диспетчеризации.

Работа по определению того, в какой момент необходимо прервать выполнение текущего активного потока и какому потоку предоставить возможность выполняться, называется планированием.

Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании могут приниматься во внимание приоритеты потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращений к вводу-выводу и другие факторы.

Планирование потоков, по существу, требует решения двух задач:

* + определения момента времени для смены текущего активного потока;
	+ выбора потока из очереди готовых потоков.

В большинстве операционных систем универсального назначения планирование осуществляется динамически (*on-line*), то есть решения принимаются во время работы системы на основе анализа текущей ситуации.

Другой тип планирования – статический. Он может быть использован в специализированных системах, в которых весь набор одновременно выполняемых задач определен заранее, например, в системах реального времени.

Диспетчеризация заключается в реализации найденного в результате планирования (динамического или статистического) решения, то есть в переключении процессора с одного потока на другой.

Диспетчеризация сводится к следующему:

* + сохранение контекста текущего потока, который требуется сменить;
	+ загрузка контекста нового потока, выбранного в результате планирования;
	+ запуск нового потока на выполнение.

Вопрос 2. Состояния процесса и переходы между ними

В многозадачной (многопроцессной) системе выделяют три основных состояния:

«ВЫПОЛНЕНИЕ» − активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

«ОЖИДАНИЕ» − пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;

«ГОТОВНОСТЬ» − также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе. Упрощенный граф состояний процесса показан на рис. 1.

В состоянии «ВЫПОЛНЕНИЕ» в однопроцессорной системе может находиться только один процесс, а в каждом из состояний «ОЖИДАНИЕ» и «ГОТОВНОСТЬ» − несколько процессов, эти процессы образуют очереди соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния «ГОТОВНОСТЬ», когда процесс готов к выполнению и ждет своей очереди.

Рис. 1. Граф состояний процесса в многозадачной среде

При активизации процесс переходит в состояние «ВЫПОЛНЕНИЕ» и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние «ОЖИДАНИЯ» какого-нибудь события, либо будет насильно «вытеснен» из процессора, например, вследствие исчерпания отведенного данному процессу кванта процессорного времени. В последнем случае процесс возвращается в состояние «ГОТОВНОСТЬ». В это же состояние процесс переходит из состояния «ОЖИДАНИЕ», после того, как ожидаемое событие произойдет.

В реальной системе каждое из этих состояний распадается на множество подсостояний и граф значительно усложняется.

В качестве примера приведем перечень состояний, соответствующий жизненному циклу процесса в ОС *Unix*:

* процесс выполняется в пользовательском режиме, или режиме задачи;
* процесс выполняется в привилегированном режиме, или режиме ядра;
* процесс не выполняется, но готов к запуску под управлением ядра;
* процесс приостановлен и находится в оперативной памяти;
* процесс готов к запуску, но программа подкачки (нулевой процесс) должна еще загрузить процесс в оперативную память, прежде чем он будет запущен под управлением ядра;
* процесс приостановлен, и программа подкачки выгрузила его во внешнюю память, чтобы в оперативной памяти освободить место для других процессов;
* процесс возвращен из привилегированного режима (режима ядра) в непривилегированный (режим задачи), ядро резервирует его и переключает контекст на другой процесс. Об отличии этого состояния от состояния 3 (готовность к запуску) пойдет речь ниже;
* процесс вновь создан и находится в переходном состоянии; процесс существует, но не готов к выполнению, хотя и не приостановлен. Это состояние является начальным состоянием всех процессов, кроме нулевого состояния;
* процесс вызывает системную функцию *exit* и прекращает существование. Однако после него осталась запись, содержащая код выхода, и некоторая хронометрическая статистика, собираемая родительским процессом. Это состояние является последним состоянием процесса.

Вопрос 3. Типы и атрибуты процессов

**Типы процессов. Системные процессы**. Системные процессы являются частью ядра и всегда расположены в оперативной памяти. Системные процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы. Выполняемые инструкции и данные этих процессов находятся в ядре системы, таким образом, они могут вызывать функции и обращаться к данным, недоступным для остальных процессов. Системными процессами являются: *shed* (диспетчер свопинга), *vhand* (диспетчер страничного замещения), *bdfflush* (диспетчер буферного кэша) и *kmadaemon* (диспетчер памяти ядра). К системным процессам следует отнести *init*, являющийся прародителем всех остальных процессов в *Unix*. Хотя *init* не является частью ядра, и его запуск происходит из исполняемого файла (/*etc/init*), его работа жизненно важна для функционирования всей системы в целом.

**Демоны**. Демоны – это не интерактивные процессы, которые запускаются обычным образом – путем загрузки в память соответствующих им программ, и выполняются в фоновом режиме. Демоны не связаны ни с одним пользовательским сеансом работы и не могут непосредственно управляться пользователем. Большую часть времени демоны ожидают пока тот или иной процесс запросит определенную услугу, например, доступ к файловому архиву или печать документа.

**Прикладные процессы**. К прикладным процессам относятся все остальные процессы, выполняющиеся в системе. Как правило, это процессы, порожденные в рамках пользовательского сеанса работы. Важнейшим пользовательским процессом является основной командный интерпретатор (*login shell*), который обеспечивает вашу работу в *Unix*. При выходе из системы все пользовательские процессы будут уничтожены.

**Зомби процессы**. До сих пор предполагалось, что вызовы *exit* и *wait* используются правильно, и родительский процесс ожидает завершения каждого подпроцесса. Вместе с тем иногда могут возникать две другие ситуации, которые стоит обсудить:

* в момент завершения дочернего процесса родительский процесс не выполняет вызов *wait*;
* родительский процесс завершается, в то время как один или несколько дочерних процессов продолжают выполняться.

В первом случае завершающийся процесс как бы «теряется» и становится зомби-процессом. Зомби-процесс занимает ячейку в таблице, поддерживаемой ядром для управления процессами, но не использует других ресурсов ядра.

В конце концов, он будет освобожден, если его родительский процесс вспомнит о нем и вызовет *wait*. Тогда родительский процесс сможет прочитать статус завершения процесса, и ячейка освободится для повторного использования.

Во-втором случае родительский процесс завершается нормально. Дочерние процессы (включая зомби-процессы) приминаются процессом *init*.

**Структуры данных процесса**. С практической точки зрения процесс в системе *Unix* является объектом, создаваемым в результате выполнения системной операции *fork*. Каждый процесс, за исключением нулевого, порождается в результате запуска другим процессом операции *fork*. Процесс, запустивший операцию *fork*, называется родительским, а вновь созданный процесс − порожденным. Каждый процесс имеет одного родителя, но может породить много процессов. Ядро системы идентифицирует каждый процесс по его номеру, который называется идентификатором процесса (*PID*). Нулевой процесс является особенным процессом, который создается «вручную» в результате загрузки системы; после порождения нового процесса (процесс 1) нулевой процесс становится процессом подкачки. Процесс 1, известный под именем init, является предком любого другого процесса в системе и связан с каждым процессом особым образом.

Ядро загружает исполняемый файл в память при выполнении системной операции *exec*, при этом загруженный процесс состоит, по меньшей мере, из трех частей, так называемых областей: текста, данных и стека (рис. 2).

Области текста и данных корреспондируют с секциями текста и *bss*-данных исполняемого файла, а область стека создается автоматически и ее размер динамически устанавливается ядром системы во время выполнения. Стек состоит из логических записей активации, помещаемых в стек при вызове функции и выталкиваемых из стека при возврате управления в вызвавшую процедуру; специальный регистр, именуемый указателем вершины стека, показывает текущую глубину стека.

Запись активации включает параметры, передаваемые функции, ее локальные переменные, а также данные, необходимые для восстановления предыдущей записи активации, в том числе значения счетчика команд и указателя вершины стека в момент вызова функции. Текст программы включает последовательности команд, управляющие увеличением стека, а ядро системы выделяет, если нужно, место под стек.

Рис. 2. Информационные структуры для процессов

Каждой системной операции соответствует точка входа в библиотеке системных операций; библиотека системных операций написана на языке ассемблера и включает специальные команды прерывания, которые, выполняясь, порождают «прерывание», вызывающее переключение аппаратуры в режим ядра. Процесс ищет в библиотеке точку входа, соответствующую отдельной системной операции, подобно тому, как он вызывает любую из функций, создавая при этом для библиотечной функции запись активации. Когда процесс выполняет специальную инструкцию, он переключается в режим ядра, выполняет операции ядра и использует стек ядра.

Стек ядра содержит записи активации для функций, выполняющихся в режиме ядра. Стек ядра для процесса пуст, если процесс выполняется в режиме задачи.

Таблица процессов включает в себя указатели на промежуточную таблицу областей процессов, точки входа в которую служат в качестве указателей на собственно таблицу областей.

Областью называется непрерывная зона адресного пространства, выделяемая процессу для размещения текста, данных и стека.

Когда процесс запускает системную операцию *exec*, ядро системы выделяет области под ее текст, данные и стек, освобождая старые области, которые использовались процессом.

Если процесс запускает операцию *fork*, ядро удваивает размер адресного пространства старого процесса, позволяя процессам совместно использовать области, когда это возможно, и с другой стороны, производя физическое копирование.

Если процесс запускает операцию *exit*, ядро освобождает области, которые использовались процессом.

Таблица процессов ссылается на промежуточную таблицу областей, используемых процессом, в которой содержатся указатели на записи в собственно таблице областей, соответствующие областям для текста, данных и стека процесса.

Запись в таблице процессов и часть адресного пространства задачи, выделенная процессу, содержат управляющую информацию и данные о состоянии процесса, такие как:

* атрибуты процесса;
* идентификатор процесса *Process ID* (*PID*). Каждый процесс имеет уникальный идентификатор *PID*, позволяющий ядру системы различать процессы. Когда создается новый процесс, ядро присваивает ему следующий свободный (т. е. не ассоциированный ни с каким процессом) идентификатор. Присвоение идентификаторов происходит по возрастающей, т. е. идентификатор нового процесса больше, чем идентификатор процесса, созданного перед ним. Если идентификатор достиг максимального значения, следующий процесс получит минимальный свободный *PID* и цикл повторяется. (При завершении процесса, ядро освобождает занятый им идентификатор);
* идентификатор родительского процесса *Parent Process ID* (*PPID*);
* идентификатор процесса, породившего данный процесс;
* приоритет процесса (*Nice Number*);
* относительный приоритет процесса, учитываемый планировщиком при определении очередности запуска;
* терминальная линия (*TTY*);
* терминал или псевдотерминал, ассоциированный с процессом, если такой существует. Процессы-демоны не имеют ассоциированного терминала.
* реальный (*RID*) и эффективный (*EUID*) идентификаторы пользователя. Реальным идентификатором пользователя данного процесса является идентификатор пользователя, запустившего процесс. Эффективный идентификатор служит для определения прав доступа процесса к системным ресурсам (в первую очередь, к ресурсам файловой системы). Обычно реальный и эффективный идентификаторы эквивалентны, т. е. процесс имеет в системе те же права, что и пользователь, запустивший его. Однако существует возможность задать процессу более широкие права, чем права пользователя путем установки флага SUID, когда эффективному идентификатору присваивается значение идентификатора владельца исполняемого файла (например, администратора);
* реальный (*RGID*) и эффективный (*EGID*) идентификаторы группы. Реальный идентификатор группы равен идентификатору первичной или текущей группы пользователя, запустившего процесс. Эффективный идентификатор служит для определения прав доступа к системным ресурсам по классу доступа группы. Так же, как и для эффективного идентификатора пользователя, возможна его установка равным идентификатору группы владельца исполняемого файла (флаг *SGID*).