Слои программного обеспечения компьютерной системы.

Наиболее существенные изменения состояли в следующем:

Интерфейс между прикладной программой и ОС был организован при помощи набора системных вызовов.

Организация очереди из заданий в памяти и выделение процессора одному из заданий потребовали планирования заданий.

Для переключения процессора с одного задания на другое возникла потребность в сохранении содержимого регистров и структур данных, необходимых для выполнения задания, иначе говоря, контекста, для обеспечения правильного продолжения вычислений.

Поскольку память является ограниченным ресурсом, оказались нужны стратегии управления памятью, то есть потребовалось упорядочить процессы размещения, замещения и выборки информации из памяти.

Так как программы могут пожелать произвести санкционированный обмен данными, стало необходимо их обеспечить средствами коммуникации.

Для корректного обмена данными необходимо предусмотреть координацию программами своих действий, т.е. средства синхронизации.

Пять основных функций, которые выполняли классические операционные системы в процессе своей эволюции:

Планирование заданий и использования процессора.

Обеспечение программ средствами коммуникации и синхронизации.

Управление памятью.

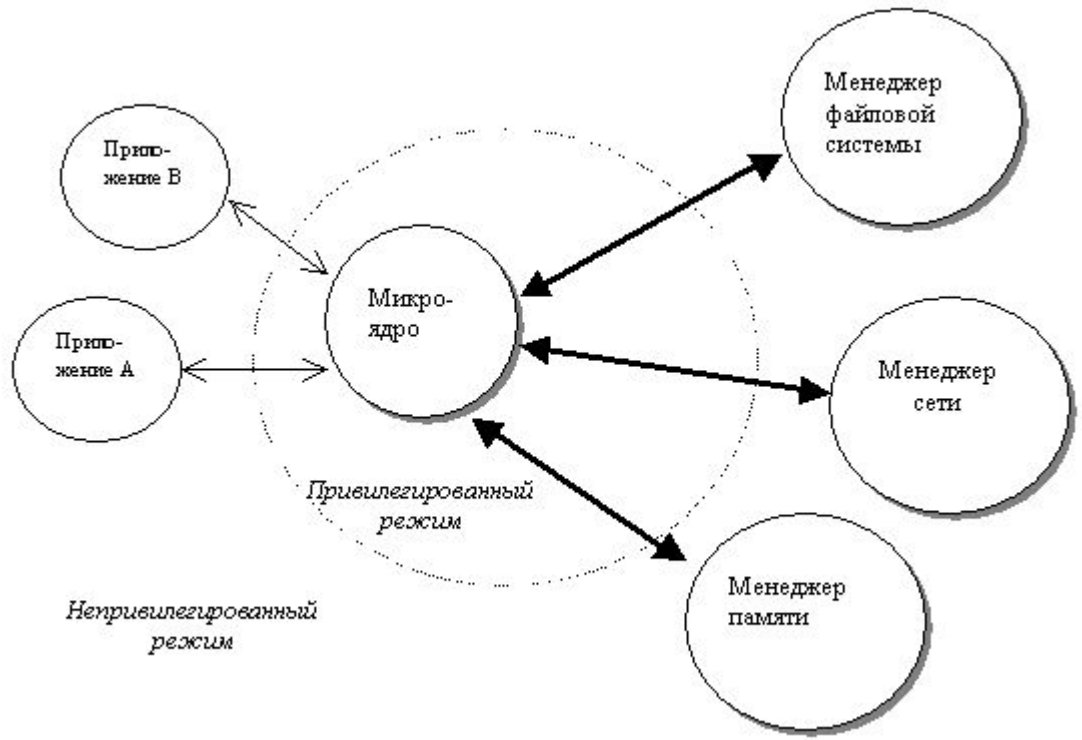
Управление файловой системой.

Управление вводом-выводом.

Обеспечение безопасности

5	Интерфейс пользователя
4	Управление вводом-выводом
3	Драйвер устройства связи оператора и консоли
2	Управление памятью
1	Планирование задач и процессов
0	Hardware

Программа пользователя	Программа пользователя	Программа пользователя
MS-DOS	Linux	Windows-NT
Виртуальное hardware	Виртуальное hardware	Виртуальное hardware
Реальная операционная система		
Реальное hardware		


Менеджер файловой системы
Прило-жение В
Микро-ядро
Прило-жение А
Менеджер сети
Привилегированный режим
Менеджер памяти
Непривилегированный режим

Процессы и их поддержка в операционной системе

Мы говорили: вычислительная система исполняет одну или несколько программ, операционная система планирует задания, программы могут обмениваться данными и т. д.

Когда мы говорили о системах пакетной обработки, мы ввели понятие “задание” как совокупности программы, набора команд языка управления заданиями, необходимых для ее выполнения, и входных данных, термины “программа” и “задание” предназначены для описания статических, неактивных объектов.

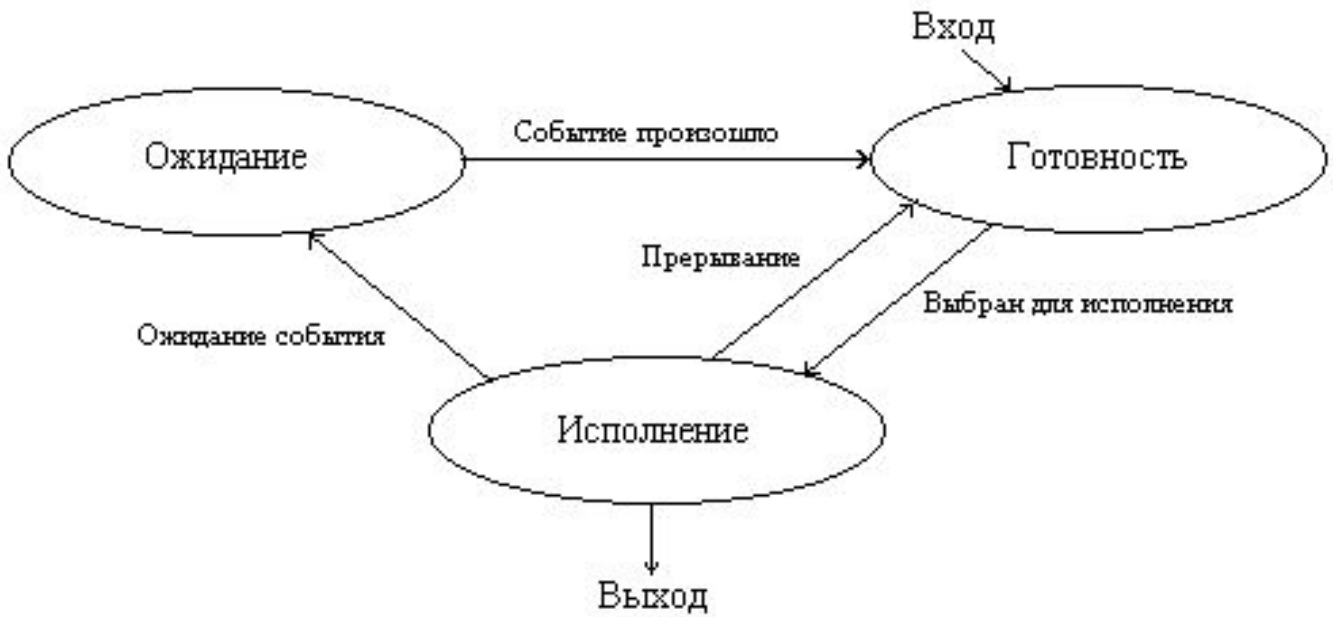
Программа же в процессе исполнения является динамическим, активным объектом. По ходу ее работы компьютер обрабатывает различные команды и преобразует значения переменных.

Для ее выполнения операционная система должна выделить определенное количество оперативной памяти, закрепить за ней определенные устройства ввода-вывода или файлы (откуда должны поступать входные данные и куда нужно доставить полученные результаты), то есть зарезервировать определенные ресурсы из общего числа ресурсов всей вычислительной системы.

Их количество и конфигурация могут изменяться с течением времени.

Для описания таких активных объектов внутри компьютерной системы вместо терминов “программа” и “задание” правильным будет использовать термин *“процесс”.*

Простейшая диаграмма состояний процесса



Более подробная диаграмма состояний процесса.

Всякий новый процесс, появляющийся в системе, попадает в состояние готовность.

Операционная система, пользуясь каким-либо алгоритмом планирования, выбирает один из готовых процессов и переводит его в состояние исполнение.

В состоянии исполнение происходит непосредственное выполнение программного кода процесса. Покинуть это состояние процесс может по трем причинам:

либо он заканчивает свою деятельность;

либо он не может продолжать свою работу, пока не произойдет некоторое событие, и операционная система переводит его в состояние ожидание;

либо в результате возникновения прерывания в вычислительной системе (например, прерывания от таймера по истечении дозволенного времени выполнения) его возвращают в состояние готовность.

Диаграмма состояний процесса, принятая в курсе

Набор операций

Процесс не может сам перейти из одного состояния в другое.

Изменением состояния процессов занимается операционная система, совершая операции над ними.

Удобно объединить их в три пары:

Создание процесса — завершение процесса;

Приостановка процесса — запуск процесса;

Блокирование процесса— разблокирование процесса;

Process Control Block и контекст процесса

Для того чтобы операционная система могла выполнять операции над процессами, каждый процесс представляется в ней некоторой структурой данных.

Эта структура содержит информацию, специфическую для данного процесса:

состояние, в котором находится процесс;

программный счетчик процесса или, другими словами, адрес команды, которая должна быть выполнена для него следующей;

содержимое регистров процессора;

данные, необходимые для планирования использования процессора и управления памятью;

учетные данные;

информацию об устройствах ввода-вывода, связанных с процессом;

Информацию, для хранения которой предназначен блок управления процессом, удобно для дальнейшего изложения разделить на две части.

Содержимое всех регистров процессора будем называть регистровым контекстом процесса, а все остальное – системным контекстом процесса.

Знания регистрового и системного контекстов процесса достаточно для того, чтобы управлять его поведением в операционной системе, совершая над ним операции.

Однако этого недостаточно, чтобы полностью характеризовать процесс.

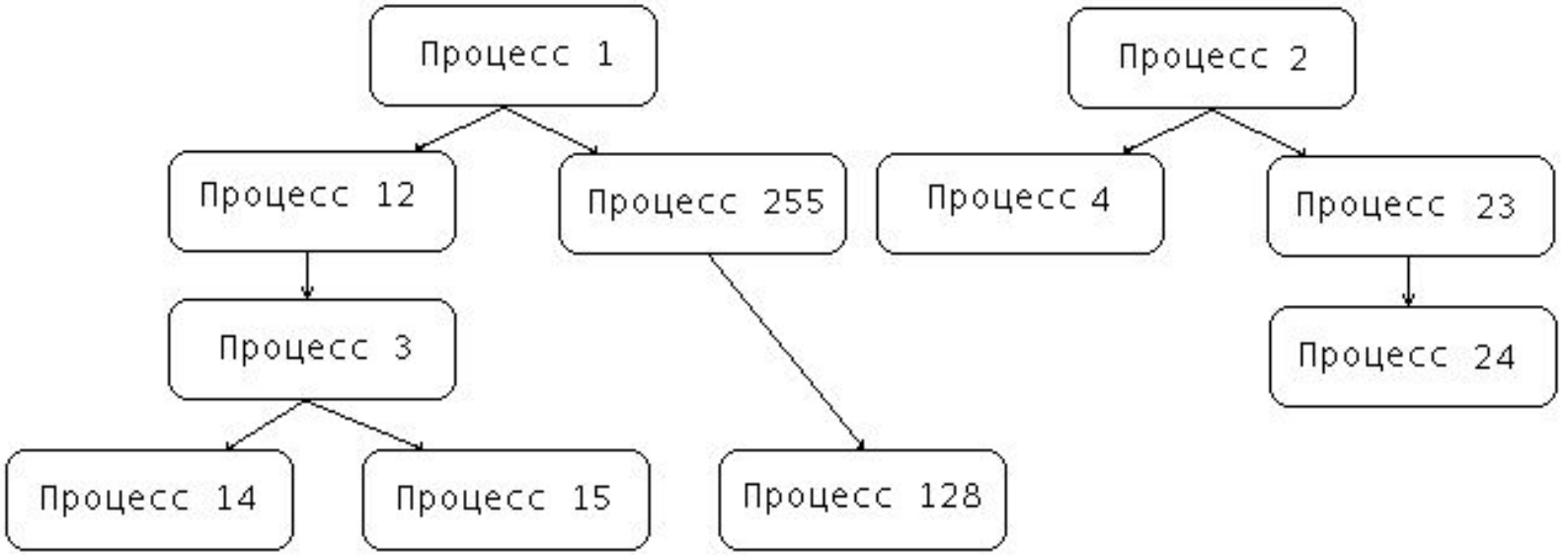
Операционную систему не интересует, какими именно вычислениями занимается процесс, т. е. какой код и какие данные находятся в его адресном пространстве.

Код и данные, находящиеся в адресном пространстве процесса, будем называть его пользовательским контекстом.

Совокупность регистрового, системного и пользовательского контекстов процесса для краткости принято называть просто контекстом процесса.

В любой момент времени процесс полностью характеризуется своим контекстом.

Процесс, инициировавший создание нового процесса, принято называть процессом-родителем (parent process), а вновь созданный процесс - процессом-ребенком (child process).



Упрощенный генеалогический лес процессов. Стрелочка означает отношение родитель-ребенок.

При рождении процесса система заводит новый PCB с состоянием процесса рождение и начинает его заполнение.

Обычно для выполнения своих функций процесс-ребенок требует определенных ресурсов: памяти, файлов, устройств ввода-вывода и т. д.

После наделения процесса-ребенка ресурсами необходимо занести в его адресное пространство программный код, значения данных, установить программный счетчик.

Порождение нового процесса как дубликата процесса-родителя приводит к возможности существования программ (т. е. исполняемых файлов), для работы которых организуется более одного процесса.

После того как процесс наделен содержанием, в PCB дописывается оставшаяся информация и состояние нового процесса изменяется на готовность.

После того, как процесс завершил свою работу, операционная система переводит его в состояние закончил исполнение и освобождает все ассоциированные с ним ресурсы, делая соответствующие записи в блоке управления процессом. При этом сам PCB не уничтожается, а остается в системе еще некоторое время. Это связано с тем, что процесс-родитель после завершения процесса-ребенка может запросить операционную систему о причине произошедшей смерти порожденного им процесса и/или статистическую информацию об его работе.

Многоразовые операции

Одноразовые операции приводят к изменению количества процессов, находящихся под управлением операционной системы, и всегда связаны с выделением или освобождением определенных ресурсов.

Многоразовые операции, напротив, не приводят к изменению количества процессов в операционной системе и не обязаны быть связанными с выделением или освобождением ресурсов.

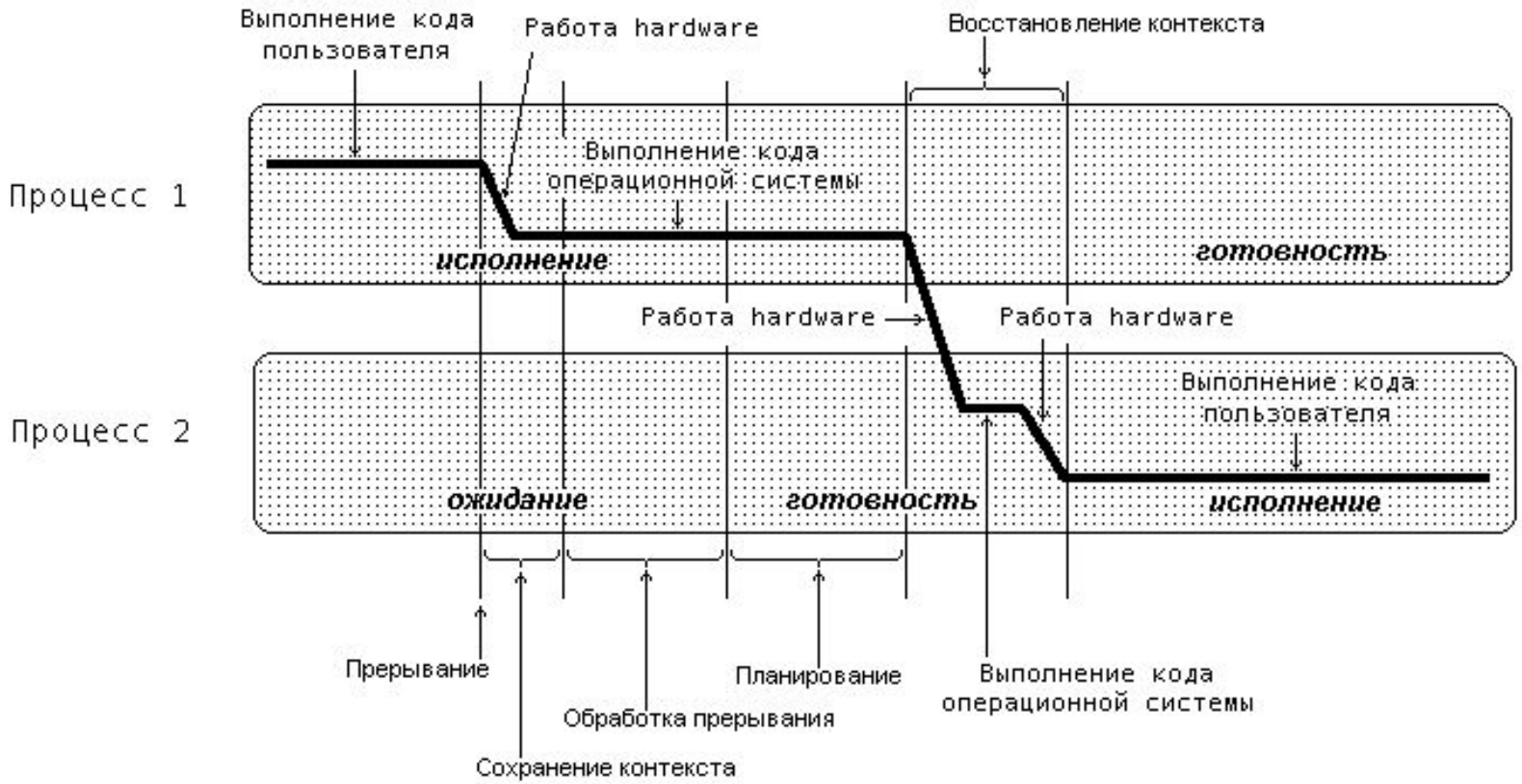
Операционная система проводит некоторые действия при выполнении многоразовых операций над процессами.

Запуск процесса.

Приостановка процесса.

Блокирование процесса.

Разблокирование процесса.



Выполнение операции разблокирования процесса. Использование термина "код пользователя" не ограничивает общности рисунка только пользовательскими процессами

Планирование процессов

Планирование заданий появилось в пакетных системах после того, как для хранения сформированных пакетов заданий начали использоваться магнитные диски.

Планирование заданий выступает в качестве долгосрочного планирования процессов. Оно отвечает за порождение новых процессов в системе, определяя ее степень мультипрограммирования, т. е. количество процессов, одновременно находящихся в ней.

Решение о выборе для запуска того или иного процесса оказывает влияние на функционирование вычислительной системы на протяжении достаточно длительного интервала времени. Отсюда и проистекает название этого уровня планирования — долгосрочное.

Планирование использования процессора выступает в качестве краткосрочного планирования процессов.

В некоторых вычислительных системах бывает выгодно для повышения их производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура получила название *swapping*, употребляется без перевода — свопинг.

Для каждого уровня планирования процессов можно предложить много различных алгоритмов.

К числу таких целей можно отнести:

Справедливость: гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе.

Эффективность: постараться занять процессор на все 100% рабочего времени.

Сокращение полного времени выполнения (turnaround time).

Сокращение времени ожидания (waiting time).

Сокращение времени отклика (response time).

Параметры планирования

Для осуществления поставленных целей разумные алгоритмы планирования должны опираться на какие-либо характеристики процессов в системе, заданий в очереди на загрузку, состояния самой вычислительной системы.

Все параметры планирования можно разбить на две большие группы: статические параметры и динамические параметры.

К статическим параметрам процессов относятся характеристики, как правило, присущие заданиям уже на этапе загрузки:

Каким пользователем запущен процесс или сформировано задание.

Насколько важной является поставленная задача, т. е. каков приоритет ее выполнения.

Сколько процессорного времени запрошено пользователем для решения задачи.

Каково соотношение процессорного времени и времени, необходимого для осуществления операций ввода-вывода.

Какие ресурсы вычислительной системы и в каком количестве необходимы заданию.

Алгоритмы долгосрочного планирования используют в своей работе статические и динамические параметры вычислительной системы и статические параметры процессов.

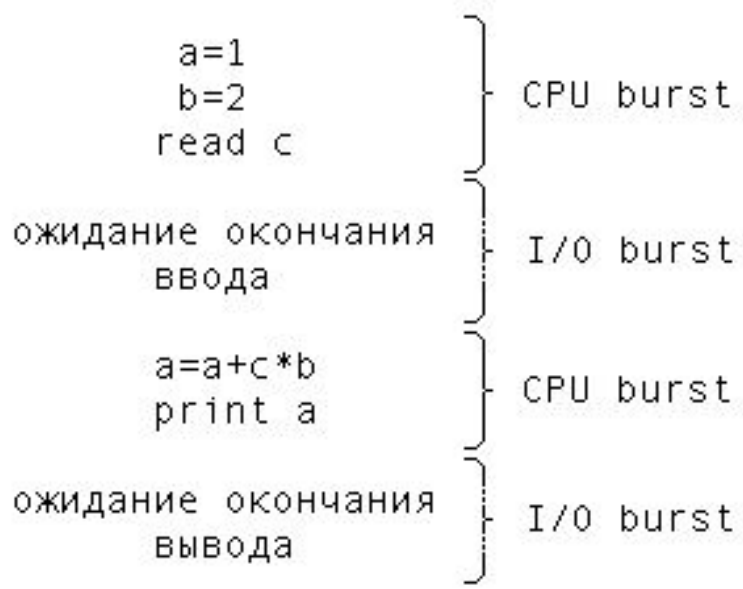
Алгоритмы краткосрочного и среднесрочного планирования дополнительно учитывают и динамические характеристики процессов.

Для среднесрочного планирования в качестве таких характеристик может выступать следующая информация:

Сколько времени прошло со времени выгрузки процесса на диск или его загрузки в оперативную память.

Сколько оперативной памяти занимает процесс.

Сколько процессорного времени было уже предоставлено процессу.



Фрагмент деятельности процесса с выделением промежутков непрерывного использования процессора и ожидания ввода-вывода.

Вытесняющее и невытесняющее планирование

Процесс планирования осуществляется частью операционной системы, называемой планировщиком.

Планировщик может принимать решения о выборе для исполнения нового процесса, из числа находящихся в состоянии готовность, в следующих четырех случаях:

1. Когда процесс переводится из состояния исполнение в состояние завершение.

2. Когда процесс переводится из состояния исполнение в состояние ожидание.

3. Когда процесс переводится из состояния исполнение в состояние готовность.

4. Когда процесс переводится из состояния ожидание в состояние готовность.

Алгоритмы планирования

First-Come, First-Served (FCFS)

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой FCFS по первым буквам его английского названия — First Come, First Served (первым пришел, первым обслужен).

Представим себе, что процессы, находящиеся в состоянии готовность, организованы в очередь. Когда процесс переходит в состояние готовность, он, а точнее ссылка на его PCB, помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование FIFO — сокращение от First In, First Out (первым вошел, первым вышел).

Такой алгоритм выбора процесса осуществляет невытесняющее планирование.

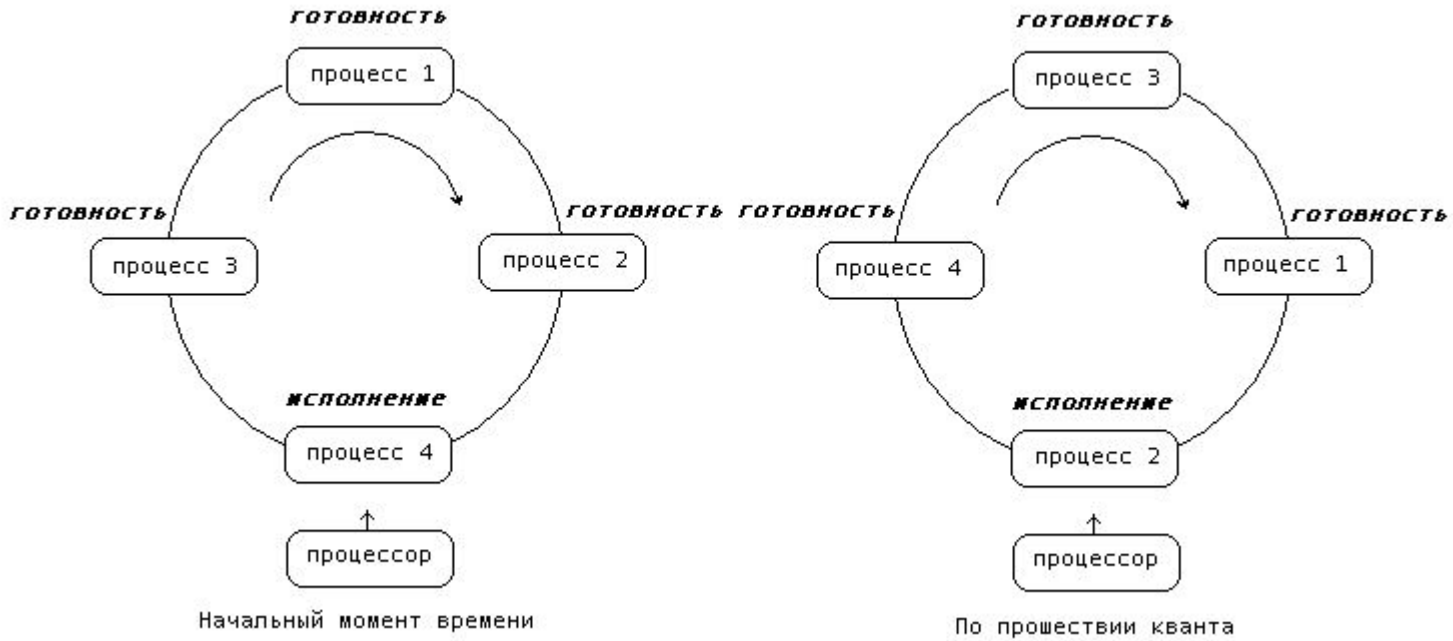
Процесс, получивший в свое распоряжение процессор, занимает его до истечения своего текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди.

Процесс	p_0	p_1	p_2
Продолжительность очередного CPU burst	13	4	1

Выполнение процессов при порядке p0,p1,p2

Выполнение процессов при порядке p2,p1,p0

Round Robin (RR)



Процессы на карусели.

время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
p_1	Г	Г	Г	Г	И	И	И	И										
p_2	Г	Г	Г	Г	Г	Г	Г	Г	И									

время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	И	Г	Г	И	Г	И	Г	И	Г	И	И	И	И	И	И	И	И	И
p_1	Г	И	Г	Г	И	Г	И	Г	И									
p_2	Г	Г	И															

Shortest-Job-First (SJF)

Процесс	p_0	p_1	p_2	p_3
Продолжительность очередного CPU burst	5	3	7	1

время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p_0	Г	Г	Г	Г	И	И	И	И	И							
p_1	Г	И	И	И												
p_2	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p_3	И															

Для рассмотрения примера вытесняющего SJF планирования мы возьмем ряд процессов p0, p1, p2 и p3с различными временами CPU burst и различными моментами их появления в очереди процессов готовых к исполнению

Процесс	Время появления в очереди	Продолжительность очередного CPU burst
p_0	0	6
p_1	2	2
p_2	6	7
p_3	0	5

время	1	2	3	4	5	6	7	8	9	10
p_0	Г	Г	Г	Г	Г	Г	Г	И	И	И
p_1			И	И						
p_2							Г	Г	Г	Г
p_3	И	И	Г	Г	И	И	И			

время	11	12	13	14	15	16	17	18	19	20
p_0	И	И	И							
p_1										
p_2										
p_3	Г	Г	Г	И	И	И	И	И	И	И

Основную сложность при реализации алгоритма SJF представляет невозможность точного знания времени очередного CPU burst для исполняющихся процессов.

В пакетных системах количество процессорного времени, требующееся заданию для выполнения, указывает пользователь при формировании задания.

Можно брать эту величину для осуществления долгосрочного SJF планирования.

Если пользователь укажет больше времени, чем ему нужно, он будет ждать получения результата дольше, чем мог бы, так как задание будет загружено в систему позже.

Если же он укажет меньшее количество времени, задача может не досчитаться до конца.

Таким образом, в пакетных системах решение задачи оценки времени использования процессора перекладывается на плечи пользователя.

При краткосрочном планировании мы можем делать только прогноз длительности следующего CPU burst, исходя из предыстории работы процесса. Пусть $t(n)$ – величина n-го CPU burst, $T(n + 1)$– предсказываемое значение для $n + 1$-го CPU burst, a – некоторая величина в диапазоне от 0 до 1.

Гарантированное планирование

При интерактивной работе N пользователей в вычислительной системе можно применить алгоритм планирования, который гарантирует, что каждый из пользователей будет иметь в своем распоряжении ~ 1/N часть процессорного времени.

Пронумеруем всех пользователей от 1 до N.

Для каждого пользователя с номером i введем две величины: Ti - время нахождения пользователя в системе, или, другими словами длительность сеанса его общения с машиной, и ti - суммарное процессорное время уже выделенное всем его процессам в течение сеанса.

Справедливым для пользователя было бы получение Ti/N процессорного времени. Если

$$\tau_i << \frac{T_i}{N},$$

$$\tau_i >> \frac{T_i}{N},$$

$$\frac{\tau_i N}{T_i}$$

Приоритетное планирование

При приоритетном планировании каждому процессу присваивается определенное числовое значение — приоритет, в соответствии с которым ему выделяется процессор.

Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst.

Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс. Для алгоритма гарантированного планирования приоритетом служит вычисленный коэффициент справедливости. Чем он меньше, тем больше приоритет у процесса.

Принципы назначения приоритетов могут опираться как на внутренние критерии вычислительной системы, так и на внешние по отношению к ней.

Внутренние используют различные количественные и качественные характеристики процесса для вычисления его приоритета. Это могут быть, например, определенные ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и используемых устройств ввода-вывода, отношение средних продолжительностей I/O burst к CPU burst и т. д.

Внешние критерии исходят из таких параметров, как важность процесса для достижения каких-либо целей, стоимость оплаченного процессорного времени и других политических факторов.

Процесс	Время появления в очереди	Продолжительность очередного CPU burst	Приоритет
p_0	0	6	4
p_1	2	2	3
p_2	6	7	2
p_3	0	5	

Многоуровневые очереди (Multilevel Queue)

Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

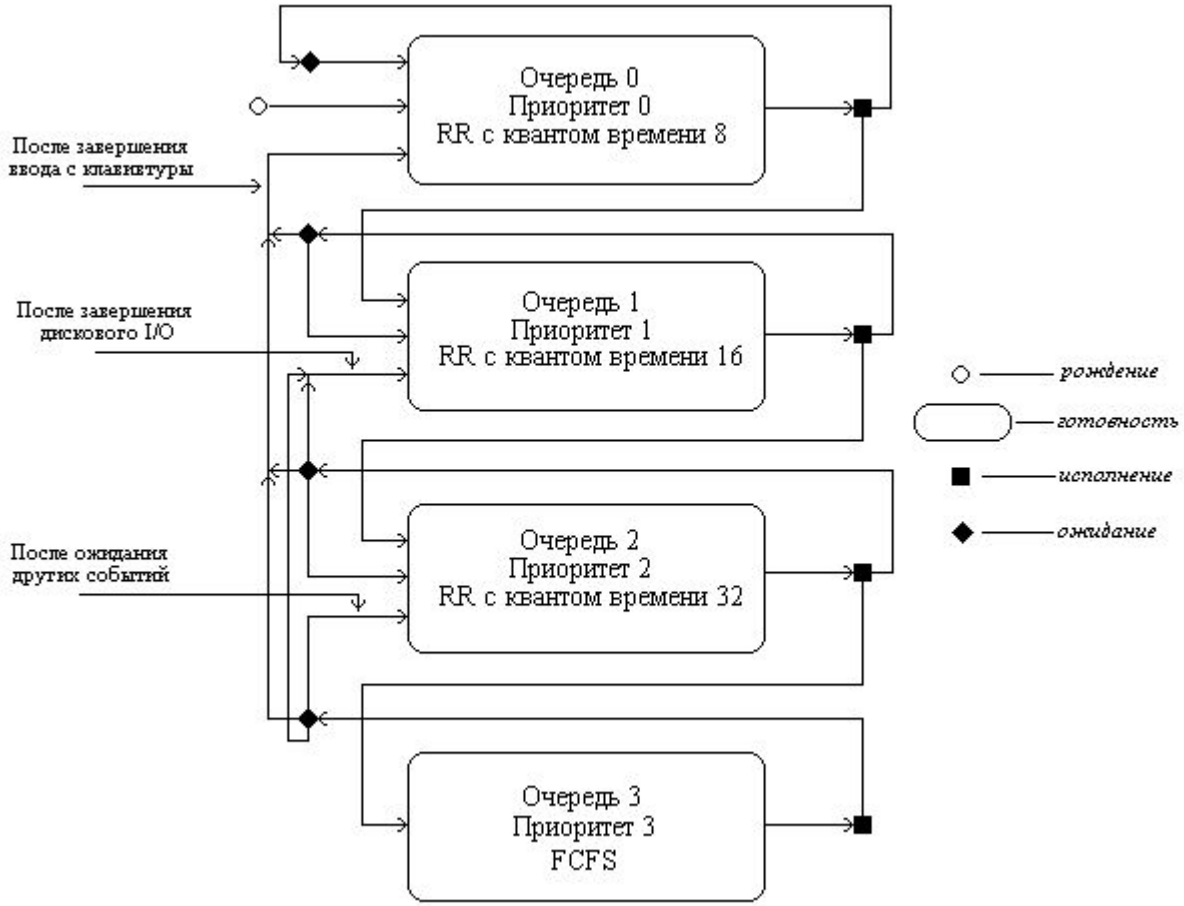


Схема миграции процессов в многоуровневых очередях планирования с обратной связью. Вытеснение процессов более приоритетными процессами и завершение процессов на схеме не показано.

Многоуровневые очереди с обратной связью представляют собой наиболее общий подход к планированию процессов из числа подходов, рассмотренных нами.

Они наиболее трудоемки в реализации, но в то же время они обладают наибольшей гибкостью.

Для полного описания их конкретного воплощения необходимо указать:

Количество очередей для процессов, находящихся в состоянии готовность.

Алгоритм планирования, действующий между очередями.

Алгоритмы планирования, действующие внутри очередей.

Правила помещения родившегося процесса в одну из очередей.

Правила перевода процессов из одной очереди в другую.

Кооперация процессов и основные аспекты ее логической организации

Взаимодействующие процессы

Для достижения поставленной цели различные процессы могут исполняться псевдопараллельно на одной вычислительной системе или параллельно на разных вычислительных системах, взаимодействуя между собой.

Для чего процессам нужно заниматься совместной деятельностью?

Одной из причин является повышение скорости работы.

Второй причиной является совместное использование данных.

Третьей причиной является модульная конструкция какой-либо системы.

Наконец, это может быть необходимо просто для удобства работы пользователя, желающего, например, редактировать и отлаживать программу одновременно.

Категории средств обмена информацией

Процессы могут взаимодействовать друг с другом только обмениваясь информацией.

По объему передаваемой информации и степени возможного воздействия на поведение другого процесса все средства такого обмена можно разделить на три категории:

Сигнальные. Вспомним профессора Плейшнера из кинофильма “Семнадцать мгновений весны”. Сигнал тревоги — цветочный горшок на подоконнике.

Канальные. Общение процессов происходит через линии связи, предоставленные операционной системой, и напоминает общение людей по телефону, с помощью записок, писем или объявлений.

Разделяемая память. Два или более процессов могут совместно использовать некоторую область адресного пространства. Разделяемая память представляет собой наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

Логическая организация механизма передачи информации

Давайте кратко охарактеризуем основные вопросы, требующие освещения при изучении того или иного способа обмена информацией.

Как устанавливается связь?

Буферизация

Здесь можно выделить три принципиальных варианта:

Буфер нулевой емкости или отсутствует.

Буфер ограниченной емкости. Размер буфера равен n, то есть линия связи не может хранить до момента получения более чем n единиц информации.

Буфер неограниченной емкости. Теоретически это возможно, но практически вряд ли реализуемо.

При использовании канального средства связи с непрямой адресацией под емкостью буфера обычно понимается количество информации, которое может быть помещено в промежуточный объект для хранения данных.

Поток ввода/вывода и сообщения

Существует две модели передачи данных по каналам связи — поток ввода-вывода и сообщения.

При передаче данных с помощью потоковой модели, операции передачи/приема информации вообще не интересуются содержимым данных.

Процесс, прочитавший 100 байт из линии связи, не знает и не может знать, были ли они переданы одновременно, т. е. одним куском, или порциями по 20 байт, пришли они от одного процесса или от разных процессов. Данные представляют собой простой поток байт, без какой-либо их интерпретации со стороны системы. Примерами потоковых каналов связи могут служить pipe и FIFO.

Будем называть способ коммуникации надежным, если при обмене данными выполняются следующие четыре условия:

Не происходит потери информации.

Не происходит повреждения информации.

Не появляется лишней информации.

Не нарушается порядок данных в процессе обмена.

Подобные действия могут быть возложены:

на операционную систему;

на процессы, обменивающиеся данными;

совместно на систему и процессы, разделяя их ответственность.

Ввести массив a
Ввести массив b
Ввести массив c
a = a + b
c = a + c
Вывести массив c

Ввести массив a
Ожидание окончания операции ввода
Ввести массив b
Ожидание окончания операции ввода
Ввести массив c
Ожидание окончания операции ввода
a = a + b
c = a + c
Вывести массив c

Процесс 1	Процесс 2
Ввести массив a	*Ожидание ввода*
Ожидание окончания операции ввода	*массивов a и b*
Ввести массив b	
Ожидание окончания операции ввода	
Ввести массив c	
Ожидание окончания операции ввода	*a = a + b*
c = a + c	
Вывести массив c	
Ожидание окончания операции вывода	

Процесс 1		Процесс 2
Создать процесс 2		
	Переключение контекста	
		Выделение общей памяти
		Ожидание ввода a и b
	Переключение контекста	
Выделение общей памяти *Ввести массив a* *Ожидание окончания операции ввода*		
Ввести массив b *Ожидание окончания операции ввода*		
Ввести массив c *Ожидание окончания операции ввода*		
	Переключение контекста	
		a = a + b
	Переключение контекста	
c = a + c *Вывести массив c* *Ожидание окончания операции вывода*		