Державний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра «Комп'ютеризовані системи управління»

Сучасні технології програмування

Курсова робота

Варіанти завдань і рекомендації до виконання

для студентів спеціальності
«Автоматизація та комп'ютерно-інтегровані технології»
3 курс, 1 семестр

Одесса – 2021

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данная курсовая работа на тему: “Обслуживание процессором вычислительной системы очереди готовых заданий” является теоретической, ориентированной на изучение концепций построения операционных систем и методов управления ресурсами процессора. Изучению этой дисциплины может предшествовать изучение таких дисциплин, как “Программирование”, “Архитектура ЭВМ”, “Объектно-ориентированное программирование”.

Целью курсовой работы является изучение основных методов, используемых при управлении ресурсами в различных операционных системах и вычислительных системах.

Задачей курсовой работы является получение, как теоретических знаний, так и практических навыков, достаточных для проектирования и программирования системного программного обеспечения современных компьютеров, ознакомление с вопросами моделирования и анализа эффективности функционирования реальных вычислительных систем.

КРАТКАЯ ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Назначение и функции операционной системы

Операционная система (ОС) – это организованная совокупность программ и данных, которая выполняет функции посредника между пользователями и компьютером. ОС служит двум целям: во-первых, сделать компьютерную систему удобной для использования, и, во-вторых, эффективно использовать аппаратные средства компьютера.

ОС является *управляющей программой*. Управляющая программа контролирует выполнение программ пользователей для предотвращения ошибок и неправильного использования компьютера.

ОС реализует множество различных функций, в том числе:

- определяет так называемый интерфейс пользователя;
- обеспечивает разделение аппаратных ресурсов между пользователями;
- дает возможность работать с общими данными в режиме коллективного пользования;
- планирует доступ пользователя к общим ресурсам;
- обеспечивает эффективное выполнение операций ввода-вывода;
- осуществляет восстановление информации и вычислительного процесса в случае ошибок.

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

В зависимости от особенностей использованного алгоритма управления процессором, операционные системы делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многопоточную обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

Важное влияние на облик операционной системы в целом, на возможности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами — подсистем управления памятью, файлами, устройствами ввода-вывода.

Специфика ОС проявляется и в том, каким образом она реализует сетевые функции: распознавание и перенаправление в сеть запросов к удаленным ресурсам, передача сообщений по сети, выполнение удаленных запросов.

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки (например, ОС ЕС),
- системы разделения времени (UNIX, VMS),
- системы реального времени (QNX, RT/11).

Некоторые операционные системы могут совмещать в себе свойства систем разных типов, например, часть заданий может выполняться в режиме пакетной обработки, а часть — в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

Описать операционную систему можно только путем деления ее на меньшие компоненты. Не все ОС имеют одинаковую структуру. Однако во многих современных ОС ставится следующие компоненты:

- управление процессами;
- управление основной (оперативной) памятью;
- управление вторичной (внешней) памятью;
- управление вводом-выводом;
- управление файлами;
- защита системы;
- сетевое обслуживание;
- система интерпретации команд.

Управление процессами

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. ***Процесс*** (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

Жизненный цикл процесса может быть разбит на несколько состояний. Различают следующие состояния процесса:

- ***новый*** *(new)* — процесс только что создан, но еще не готов к выполнению;
- ***выполняемый*** *(running)* — команды процесса выполняются центральным процессором;
- ***ожидающий*** *(waiting)* — процесс ожидает наступления некоторого события (например, завершения ввода-вывода или поступления сигнала);
- ***готовый*** *(ready)* — процесс готов к выполнению и ожидает освобождения центрального процессора;
- ***завершенный*** (*terminated*) — процесс завершил выполнение, но еще не удален из системы.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе.

Контекст и дескриптор процесса

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок, выполняемых данным процессом системных вызовов и т.д. Эта информация называется ***контекстом процесса***.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют ***дескриптором процесса***.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит, по крайней мере, один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Процессы в системе могут выполняться параллельно и должны создаваться и удаляться динамически. Поэтому ОС должна предоставлять механизмы порождения и прекращения процессов.

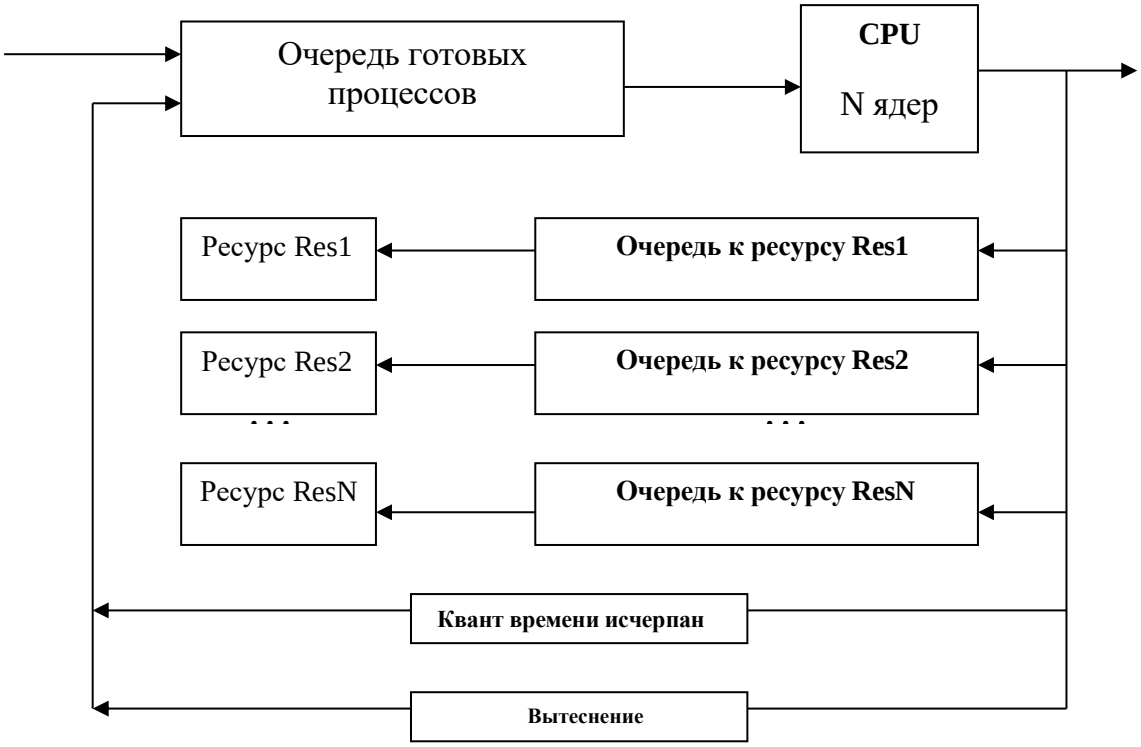

Рис 2 – Диаграмма движения процессов

Планирование процессов

Планирование процессов включает в себя решение следующих задач:

- определение момента времени для смены выполняемого процесса;
- выбор процесса на выполнение из очереди готовых процессов;
- переключение контекстов "старого" и "нового" процессов.

Первые две задачи решаются программными средствами, а последняя в значительной степени аппаратно

Существует множество различных алгоритмов планирования процессов, по-разному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы наиболее часто встречающихся алгоритмов: алгоритмы, основанные на *квантовании,* и алгоритмы, основанные на *приоритетах.*

Существует два основных типа процедур планирования процессов - вытесняющие (preemptive) и невытесняющие (non-preemptive).

Non-preemptive multitasking - ***невытесняющая многозадачность*** *— это* способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking - ***вытесняющая многозадачность*** *— это* такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Критерии планирования работы процессора

- Загрузка центрального процессора (КПД) измеряется в процентах.
- Пропускная способность процессора - количество процессов, выполненных в единицу времени.
- Время ожидания - интервал времени с момента появления процесса во входной очереди до момента его завершения.
- Время оборота - суммарное время нахождения процесса в очереди готовых процессов.
- Время отклика - время с момента попадания процесса во входную очередь до момента первого обращения к процессору.

Стратегии планирования процессора

FCFS (first come- first served)

Это дисциплина планирования, согласно которой задачи обслуживаются в порядке их появления. Дисциплина не требует внешнего вмешательства в ход вычислений. Алгоритм не является вытесняющим. К достоинствам дисциплины можно отнести простоту реализации и малые расходы системных ресурсов на реализацию очереди задач.

Планирование по принципу SJP («кратчайшее задание—первым»)

Принцип SJF («кратчайшее задание — первым») — это дисциплина планирования без переключения, согласно которой следующим для выполнения выбирается ожидающее задание (или процесс) с минимальным оценочным рабочим временем, остающимся до завершения. Принцип SJF обеспечивает уменьшение среднего времени ожидания по сравнению с дисциплиной FIFO. Однако времена ожидания при этом колеблются в более широких пределах (т. е. менее предсказуемы), чем в случае FIFO, особенно при больших заданиях.

Дисциплина SJF оказывает предпочтение коротким заданиям (или процессам) за счет более длинных. Многие разработчики считают, что чем короче задания, тем лучшие условия для его выполнения следует создавать. Далеко не все специалисты придерживаются подобного мнения, особенно применительно к случаям, когда необходимо учитывать приоритетность заданий.

Механизм SJP выбирает для обслуживания задания таким образом, чтобы очередное задание завершалось и покидало систему как можно быстрее. Благодаря этому обеспечивается уменьшение количества ожидающих заданий, а также количества заданий, стоящих в очереди за длинными заданиями. В результате дисциплина SJF позволяет свести к минимуму среднее время ожидания для заданий, проходящих через систему.

Очевидная проблема, связанная с реализацией принципа SJF, состоит в том, что он требует точно знать, сколько времени будет выполняться задание или процесс, а такой информации обычно не бывает. Самое большее, что может сделать механизм SJF — это полагаться на оценочные значения времен выполнения, указываемые пользователями. В условиях производственного счета, когда одни и те же задачи выполняются регулярно, довольно точные и обоснованные оценки, по-видимому, возможны. Однако в исследовательской работе пользователи редко знают, как долго будут выполняться их программы.

Ориентация механизма планирования на оценки пользователей приводит к любопытным последствиям. Если пользователи знают, что система оказывает предпочтение заданиям с малыми оценочными временами выполнения, они могут задавать малые оценочные времена. Можно, однако, спроектировать

планировщик заданий таким образом, чтобы избавить их от подобного соблазна. Пользователя можно заранее предупредить о том, что в случае, если его задание будет выполняться дольше оценочного времени, то оно будет просто выводиться из системы, причем пользователю придется платить за всю работу. Второе возможное решение заключается в том, чтобы выполнять задание в течение указанного оценочного времени плюс небольшой дополнительный процент, а затем «консервировать» задание, т. е. запоминать его в текущем виде, с тем чтобы впоследствии можно было вновь продолжить его выполнение. При этом пользователю, естественно, придется оплачивать затраты на консервацию и последующую расконсервацию (т. е. перезапуск), и, кроме того, он пострадает от задержки в завершении его задания. Существует еще одно возможное решение — выполнять задание в течение оценочного времени по обычному тарифу, а затем за дополнительное время предъявлять счет по повышенному тарифу, значительно превышающему обычный. При таких условиях пользователь, указывающий нереально низкие оценочные времена, чтобы получить улучшенное обслуживание, будет в итоге платить значительно больше, чем обычно.

Принцип SJF подобно FIFO—это дисциплина обслуживания без переключения, поэтому ее не рекомендуется применять в системах разделения времени, где необходимо гарантировать приемлемые времена ответа.

Планирование по наивысшему приоритету

Один из простых методов планирования очереди готовых процессов состоит в том, что процессор предоставляется тому процессу, который имеет наивысший приоритет (правило HPF (По первым буквам «highest priority first», что означает «наивысший приоритет — первым»)). Если вытеснение не допускается, то процесс с наивысшим приоритетом выполняется до тех пор, пока не кончится или не заблокирует сам себя. Если в очередь поступает процесс с более высоким приоритетом, чем у текущего процесса, то он должен ждать, пока текущий процесс не освободит процессор. Если вытеснение разрешено, то при появлении процесса с более высоким приоритетом текущий процесс прерывается и управление переходит к вновь прибывшему процессу. Вытесненный процесс возвращается

в очередь готовых процессов. В правиле наивысшего приоритета компоненты приоритета, применение вытеснении и организация очереди являются свободными параметрами реализации.

Каждый раз, когда процессор освобождается, из очереди готовых процессов должно быть извлечено задание с наибольшим приоритетом. Если очередь упорядочена, то извлечь первый элемент просто. Но чтобы поставить процесс в такую очередь, требуется много времени: для каждой операции вставления нужно перебрать в среднем половину очереди, если считать, что элементы связаны в двусторонний список, т. е. с указателями из начала в конец и из конца в начало очереди. Если очередь не упорядочена, то ее надо просматривать каждый раз, когда нужен новый процесс, зато операция вставления тривиальна. Возможны также компромиссные стратегии. Например, очередь можно упорядочивать через каждые *T* единиц времени. Когда поступает новый процесс, он ставится в начало очереди. Но при этом сохраняется ссылка P, которая указывает на начало рассортированной части списка. Поиск процесса с наивысшим приоритетом начинается со ссылки P и продолжается в направлении начала очереди. Количество элементов, которые приходится перебирать, меньше, чем в случае неупорядоченной очереди, а время сортировки здесь сокращено по сравнению со случаем строго упорядоченной очереди. Правда, ради простоты в системах обычно применяют упорядоченную очередь готовых процессов.

Если пользоваться стратегией HPF, то прежде всего нужно решить, на каких характеристиках будет основан приоритет. Одна из наиболее часто упоминаемых в литературе стратегий HPF выбирает каждый раз самое короткое задание (SJF По первым буквам «shortest job first», что означает «кратчайшее задание первым)), т. е. приоритет обратно пропорционален времени обработки. Так как точное время выполнении процесса редко известно заранее, для определения приоритета обычно применяется ожидаемое время. Очевидно, что процессу выгодно иметь небольшое ожидаемое время выполнения. Пользователи могут сознательно указывать неверное ожидаемое время с целью повысить свой приоритет. С этим можно бороться, наказывая пользователей, которые занижают ожидаемое время выполнения. Например, для процессов, выходящих за пределы

своей оценки, может быть увеличена цена машинного времени.

Правило SJF можно реализовать с вытеснениями или без них. Если вытеснение запрещено, то процессы всегда работают до тех пор, пока не закончатся или не заблокируются. В случае, когда разрешены вытеснения, если поступает процесс с меньшим ожидаемым временем, чем у текущего процесса, то последний вытесняется и его место занимает новый процесс. Для коротких процессов стратегия с вытеснениями лучше, чем без вытеснения. Если вытеснения разрешены, то длинным процессам становится труднее удерживать процессор в своем распоряжении.

Хотя вытеснение улучшает обслуживание коротких процессов, с ним сопряжены определенные издержки. Вытеснение требует какого-то времени и тем самым задерживает всю очередь готовых процессов. Поэтому если выполняемый процесс близок к завершению своей работы, то, видимо, не стоит вытеснять его.

Для типичных распределений величины времени выполнения характерно преобладание коротких процессов. В таких условиях правило SJF работает вполне успешно, так как оно весьма благоприятствует коротким заданиям. Правда, длинные процессы обслуживаются плохо. Вели система сильно загружена, то с длинными процессами дело обстоит особенно плохо, и это положение еще усугубляется, если разрешено вытеснение. Впрочем, если система близка к пределу своих возможностей, то любое правило планирования работает не вполне хорошо.

Поскольку предварительная информация о времени выполнения обычно очень неточна, во многих стратегиях планирования предпринимаются попытки основывать выбор на информации о процессе, полученной за время его пребывания в системе. Один класс правил использует линейно возрастающий приоритет [Клейнрок, 1970]. Каждому процессу при входе в систему присваивается некий приоритет. Приоритет возрастает с коэффициентом *a* во время ожидания в очереди готовых процессов и с коэффициентом *Ь* во время выполнения. В зависимости от выбора *a* и *b* получаются разные правила планирования. Например, если 0<*a*<=*b,* то очередь обслуживается в порядке поступления (FCFS). Если 0>b>=a, то получается обслуживание в противоположном порядке (LCFS (По

первым буквам «last-come-first-served, что означает «последним пришел — первым обслужен»)). Все возможные правила этого класса, которые можно получить, варьируя параметры a и b, впервые описал Клейнрок. Один алгоритм из этого класса будет рассмотрен в следующем разделе.

Можно придумать много вариантов правила линейно возрастающего приоритета, хотя очень немногие из них изучены. Например, коэффициенты, a и b могут зависеть от внешнего приоритета, т. е. от затрат, которые пользователь считает приемлемыми. Можно выбирать и нелинейные функции. Можно устроить, чтобы приоритет убывал по линейному закону с течением времени. Когда достигается некое максимальное время, приоритет скачком возрастает до некоторой большой величины. Это благоприятствует коротким процессам, и при этом соблюдается условие, что ни одному процессу не придется ждать обслуживания слишком долго. Подобным же образом можно дополнить правило SJF, так чтобы длинные процессы, проведя некоторое время в системе, начинали получать добавочный приоритет. Избранный способ вычисления приоритета в большой степени определяется особенностями того типа процессов, с которыми имеет дело система.

Одна из важных задач планирования состоит в том, чтобы обеспечивать занятость внешних устройств. С этой целью процессам можно присваивать высокий приоритет в периоды, когда они интенсивно используют ввод-вывод. Такого рода интенсивность легко прослеживается, так как процесс обычно блокируется после обращения к операции ввода-вывода. В частности, приоритет процесса может быть обратно пропорционален длине промежутка времени с момента его последнего обращения к вводу-выводу. Имеются экспериментальные данные, свидетельствующие о том, что операции ввода-вывода обычно бывают сконцентрированы в отдельных частях программ. Применяя стратегию HPF. основанную на интенсивности ввода-вывода, можно обеспечить занятость периферийных устройств в течение большой доли времени, так как процессам, больше других использующим ввод-вывод, легче получить в свое распоряжение процессор.

Очевидно, что количество различных правил вычисления приоритета практически не ограничено. Несмотря на то что стратегия приносит желаемый

результат, обеспечивая большинству процессов хорошее обслуживание, она дает и побочный эффект: заставляет процессы с низким приоритетом долго ждать. В некоторых системах это неприемлемо. Например, в системах с разделением времени пользователю гораздо удобнее получать по одной секунде машинного времени каждые пять секунд, чем двенадцатиминутный интервал каждый час. Методы HPF не могут обеспечить такой вид обслуживания- Это приводит нас ко второму важному классу правил планирования, так называемым «круговоротам).

Круговорот

Простой *круговорот* (RR «round robin») не использует никакой статической или динамической информации о приоритетах. Первый процесс из очереди готовых процессов получает квант времени длиной в q секунд, а затем отправляется снова в конец очереди, если только он себя не заблокирует. Если в очереди имеется *К.* процессов, то каждый процесс получает q секунд из каждых Kq секунд процессорного времени. Следовательно, каждый процесс как бы выполняется процессором, скорость которого равна *[1/К* скорости физического процессора. Поэтому длина очереди—это важный параметр, определяющий быстроту продвижения процессов.

Величина кванта q определяет, насколько равномерно распределено время процессора по коротким периодам. Если время q бесконечно, то метод круговорота сводится к стратегии FCFS. Так как процессы, как правило, все время то входят в очередь, то покидают ее, большая (конечная) величина кванта может случайно благоприятствовать одним процессам в ущерб другим. Например, если четыре новых процесса поступают в очередь в тот самый момент, когда некоторый процесс заканчивает свой квант, то получение им следующего кванта отодвинется на 4q секунд. С другой стороны, может случиться, что какому-то процессу повезет и многие стоявшие перед ним в очереди процессы окажутся заблокированными, что сократит ему время ожидания в следующем круге. Такие случайные эффекты сглаживаются по мере уменьшения q, так как вероятность блокировки процесса или поступления нового в течение одного кванта становится малой.

С уменьшением *q* улучшается обслуживание более коротких процессов. Когда *q* очень мало, все готовые процессы обслуживаются одинаково и время ожидания прямо пропорционально объему требуемых услуг. Однако можно выбратьq слишком малым. Для квантов «разумной» величины можно пренебречь временем, необходимым для переключения процессора с одного процесса на другой. Если же *q*— величина того же порядка, что и время переключения, то задержка при переключении становится заметной. В самом деле, если время *q* слишком мало, то система может потратить на переключение процессов больше времени, чем на их выполнение. Вот почему при круговороте надо брать квант времени средней величины, чтобы накладные расходы, связанные с переключением, не оказались чересчур высокими.

Метод круговорота часто применяется в системах с разделением времени, рассчитанных на большое число пользователей. Такие системы должны, как правило, гарантировать каждому процессу приемлемое время ответа. Иначе говоря, пользователи ожидают, что независимо от величины кванта их будут обслуживать по меньшей мере один раз каждые несколько секунд. Если *q* постоянно и в очереди *К.* процессов, то время ответа приблизительно равно *Kq.* Для больших *К* это может быть слишком долго. Определенное время ответа можно гарантировать, если применить круговорот, ориентированный на цикл. В этом методе выбирается время цикла для очереди готовых процессов, равное максимальному достижимому времени ответа. Величина кванта вычисляется вначале каждого круга прохождения очереди путем деления времени цикла на количество процессов в очереди. Процессы, вновь поступающие во время цикла, задерживаются до окончания цикла, прежде чем будут поставлены в очередь. Если очередь слишком длинная, то потери на переключение могут быть заметными, так как *q* будет совсем маленьким. Эту проблему можно урегулировать, задав минимальную величину кванта. При сильной загрузке системы пострадает время ответа, но по крайней мере издержки на переключение останутся на низком уровне.

Существует множество других разновидностей круговорота. По одному из правил, называемому *круговоротом со смещением,* каждому процессу предоставляется квант, величина которого зависит от внешнего приоритета процесса.

Каждому процессу соответствует своя длина кванта, так что процесс продвигается со скоростью, пропорциональной его приоритету. Можно придумать аналогичные правила, основанные на других составляющих приоритета, таких, как распределение ресурсов, интенсивность ввода-вывода и ожидаемое время обслуживания. Другой алгоритм круговорота получается из схемы линейно возрастающего приоритета Клейнрока. Если параметры *a* и *b* таковы, что 0<=b<a, то в результате получается правило планирования, которое называется *«эгоистическим» круговоротом.* Войдя в систему, процесс ждет, пока его приоритет не достигнет величины приоритета работающих процессов. Затем он выполняется в круговороте с другими работающими процессами. Поскольку приоритет выполняемых процессов увеличивается с коэффициентом b<*a,* ожидающие процессы должны их в конце концов догнать. И «эгоистический», и простой круговороты более благоприятны для коротких процессов, чем стратегия FCFS просто потому, что короткие процессы получают с каждым квантом большую долю своего общего времени. При b=0 «эгоистический» круговорот фактически сводится к простому.

Имея несколько очередей готовых процессов, система может извлечь выгоду из характера распределения времен работы. Пусть, например, известно, что если процесс не закончился в течение трех квантов времени, то с вероятностью 85% он не закончится, пока не получит 12 квантов. Этим свойством времен работы процессов можно воспользоваться, применяя две очереди готовых процессов: ведущую и фоновую. Каждый процесс проходит через три кванта в ведущей очереди. После третьего кванта он сбрасывается в фоновую очередь. Процессор тратит большую часть своего времени на выполнение процессов из ведущей очереди, обеспечивая таким образом отличное обслуживание коротких процессов. В одном из алгоритмов, ориентированных на цикл, процессам из ведущей очереди предоставляется фиксированный квант времени. Если после того, как поработали все процессы ведущей очереди, до конца цикла еще остается время, то оно используется для выполнения фоновых процессов. Такую систему с двумя очередями можно обобщить до многоуровневого метода очередей с обратной связью.

Очереди с обратной связью

В основном алгоритме очередей с *обратной связью* (FB («feedback»), что означает «обратная связь») используется *n* очередей, каждая из которых обслуживается в порядке поступления. Новый процесс, входя в систему, попадает в первую очередь. После получения одного кванта он поступает во вторую очередь. После получения каждого последующего кванта он переходит в очередь со следующим номером. Время процессора планируется так, что он обслуживает непустую очередь с наименьшим номером. В методе FB вновь поступивший процесс неявно получает высокий приоритет и выполняется подряд в течение стольких квантов времени, сколько успеет пройти до прихода следующего процесса, но не больше, чем успел проработать предыдущий. Правило FB обеспечивает лучшее, чем при круговороте, обслуживание коротких процессов, не требуя предварительной информации о времени выполнения. Однако работа с несколькими очередями может повлечь большие издержки времени в начальный период, особенно если величина кванта мала.

Один из способов уменьшить требуемое количество очередей представляет собой обобщение метода ведущей и фоновой очередей, ориентированного на цикл. Вместо того чтобы менять очередь, после каждого кванта, позволим процессу фиксированное число раз проходить каждую очередь, прежде чем опускаться на следующий, более низким уровень приоритета. При этом очереди, упорядоченные по правилу FB, обслуживаются по правилу круговорота внутри каждой очереди, и по тому же правилу распределяется время процессора между очередями. Хотя при этом различие между процессами разной длины проводится уже не так четко, накладные расходы, связанные с содержанием очередей, уменьшаются.

При стратегии FB пользователи могут извлечь для себя некоторую выгоду, разбивая свои задания на короткие процессы. Время обслуживания всех частей будет значительно лучше, чем если представить задание в виде одного длинного процесса.

Основное преимущество очередей с обратной связью и круговорота по

сравнению с обслуживанием по наивысшему приоритету состоит в том, что первые два метода позволяют хорошо обслуживать короткие задания, не требуя предварительной информации о времени выполнения процесса. С другой стороны, в тех случаях, когда приоритет вычисляется как функция от ресурсов, выделяемых процессу, часто желательно отдать предпочтение процессам с высоким приоритетом, чтобы увеличить загруженность ресурсов. Если желательно, чтобы учитывалась информация о приоритетах, то можно использовать смешанные правила, которые сочетают использование динамических и статических данных, например, круговорот со смещением.

Методы распределения оперативной памяти

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса. Функциями ОС по управлению памятью являются:

- отслеживание свободной и занятой памяти,
- выделение памяти процессам и освобождение памяти при завершении процессов, вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место,
- настройка адресов программы на конкретную область физической памяти.

Методы размещения процессов в основной памяти по отношению к расположению участков памяти, выделенных для процесса, делят на два класса:

- методы смежного размещения;
- методы несмежного размещения.

Смежное размещение является простейшим и предполагает, что в памяти, начиная с некоторого начального адреса, выделяется один непрерывный участок адресного пространства.

При несмежном размещении программа разбивается на несколько частей, которые располагаются в различных, не обязательно смежных участках адресного пространства.

Рассмотрим сначала некоторые методы смежного размещения.

Распределение памяти фиксированными разделами

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее

генерации. Очередная задача, поступившая на выполнение, помещается либо в общую очередь, либо в очередь к некоторому разделу.

Подсистема управления памятью в этом случае выполняет следующие задачи:

- сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел,
- осуществляет загрузку программы и настройку адресов.

При очевидном преимуществе - простоте реализации - данный метод имеет существенный недостаток - жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов независимо от того, какой размер имеют программы. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к неэффективному использованию памяти. С другой стороны, даже если объем оперативной памяти машины позволяет выполнить некоторую программу, разбиение памяти на разделы не позволяет сделать этого.

Распределение памяти разделами переменной величины

В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

Задачами операционной системы при реализации данного метода управления памятью является:

- ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти,
- при поступлении новой задачи - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи,

- загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей,
- после завершения задачи корректировка таблиц свободных и занятых областей.

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам, таким, например, как "первый попавшийся раздел достаточного размера", или "раздел, имеющий наименьший достаточный размер", или "раздел, имеющий наибольший достаточный размер". Все эти правила имеют свои преимущества и недостатки.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток - *фрагментация памяти*. Фрагментация - это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

К методам несмежного размещения относятся сегментная, страничная и сегментно-страничная организация памяти.

ЗАДАНИЕ К КУРСОВОЙ РАБОТЕ

Для модели вычислительной системы (ВС) с N-ядерным процессором и мультипрограммным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в вычислительных системах.

При построении модели функционирования вычислительной системы должны учитываться следующие основные моменты обслуживания заданий:

- генерация нового задания;

- постановка задания в очередь для ожидания момента освобождения процессора;

- выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

Генерация задания:

Считается, что в распоряжении вычислительной системы имеется N ГБ оперативной памяти для размещения рабочей области процесса и M (3<=m<=9) ресурсов Res1, Res2,…, Resm, обращение к которым переводит процесс в состояние ожидания.

Генерация нового задания (процесса) может происходить:

- в интерактивном режиме по запросу пользователя
- автоматически системой как случайное событие

Каждый процесс характеризуется:

- именем;
- длиной рабочей области;
- интервалом непрерывного выполнения;
- причиной прекращения непрерывной работы (обращение к ресурсу или завершение работы);
- приоритетом, если он требуется используемым методом планирования процессора.

Перед постановкой задания в очередь имитируется размещения рабочей области процесса в оперативной памяти. В случае невозможности размещения процесс отвергается, в противном случае ему выделяется память и процесс помещается в очередь готовых заданий.

Размещение в ОП происходит одним из трёх методов:

1. первого подходящего;
2. наиболее подходящего;
3. наименее подходящего;

Выборка задания из очереди готовых процессов происходит в момент, когда текущий процесс исчерпал интервал непрерывной работы и освободил CPU.

В случае обращения к ресурсу процесс помещается в очередь к нему, причем время использования ресурса генерируется случайным образом.

В случае завершения процесс удаляется из очереди готовых процессов.

Все очереди к ресурсам обслуживаются алгоритмом FCFS (в порядке поступления). Считается, что в каждый момент времени процесс может обратиться только к одному ресурсу. По окончании работы с ресурсом процесс вновь помещается в очередь готовых заданий, причем генерируется новые интервал непрерывной работы и причина ее прекращения.

Варианты заданий

Вариант	Метод планирования памяти	Стратегия планирования	Наличие вытеснения	Способ организации очереди				Динамическое повышение приоритета	Критерий вытеснения для SJF	
				упорядоченный список	не упорядоченный список	список частично упорядочивается через t тактов	каждому приоритету своя очередь		По интервалу непрерывного выполнения	По оставшемуся времени
1	1	HPF	-	+				-		
2	2	HPF	-		+			-		
3	3	HPF	-			+		-		
4	1	HPF	-				+	-		
5	2	HPF	-	+				+		
6	3	HPF	-		+			+		
7	1	HPF	-			+		+		
8	2	HPF	-				+	+		
9	3	HPF	+	+				-		
10	1	HPF	+		+			-		
11	2	HPF	+			+		-		
12	3	HPF	+				+	-		
13	1	HPF	+	+				+		
14	2	HPF	+		+			+		
15	3	HPF	+			+		+		
16	1	HPF	+				+	+		
17	2	SJF	-	+						
18	3	SJF	-		+					
19	1	SJF	-			+				
20	2	SJF	-				+		+	
21	3	SJF	+	+					+	
22	1	SJF	+		+				+	
23	2	SJF	+			+			+	
24	3	SJF	+				+			+
25	1	SJF	+	+						+
26	2	SJF	+		+					+
27	3	SJF	+			+				+
28	1	RR (простой) с приоритетами	-					-		
29	1	RR со смещением	+					-		
30	2	RR (простой, без приоритетов)	-					-		
31	2	RR со смещением	+					-		
32	3	RR	-					-		
33	3	RR со смещением	+					-		
34	1	RR (простой) с приоритетами	-					+		
35	2	RR со смещением	+					+		
36	3	RR (простой)	-					+		
37	3	RR (простой)	+					+		

Требования к оформлению курсовой работы

Результаты выполнения курсовой работы предоставляются на носителе, который должен содержать все исходные тексты программы, а также все остальные файлы, необходимые для компиляции и запуска выполненной работы в используемой среде программирования.

Пояснительная записка к курсовой работе должна содержать:

- вариант задания;
- постановку задачи;
- подробное описание алгоритмов планирования процессора и памяти;
- диаграмму движения процессов;
- объектную модель;
- описание всех модулей, содержащих исходный текст программы;
- инструкцию по пользованию программой.

Требования к интерфейсу

Курсовую работу требуется выполнить с использованием шаблона MVC в среде визуального программирования (например, Microsoft Visual Studio, IntelliJIdea, Eclipse, NetBeans и т.п.).

Экранная форма интерфейса приложения должна соответствовать рекомендациям по эргономике оконных приложений.

Работа может состоять из одной или нескольких форм. В формах могут быть использованы любые компоненты по желанию студентов, необходимые для того, чтобы отразить полностью работу модели вашей системы.

Система должна работать пошагово, медленно и при необходимости увеличивать скорость. Это может быть реализовано как при помощи компонентов, так и при помощи разработанного вами объекта таймер. Действия должны происходить по нажатию кнопки в главной форме.

Обязательно должны полностью просматриваться очереди к процессору и отдельно ко всем ресурсам. Должно быть видно, какой процесс занимает в

данный момент ресурс. При необходимости нужно иметь возможность показать полные сведения о любом находящемся в системе процессе.

В интерактивном режиме по нажатию кнопки система имитации должна выводить информацию об условиях проведения эксперимента (интенсивность потока задания, размер очередей заданий и очередей к ресурсам, производительность процессора, число тактов имитации) и полученные в результате имитации показатели функционирования вычислительной системы, в т.ч.:

- количество заданий, поступивших в вычислительную систему;
- количество отказов в обслуживании заданий из-за переполнения очереди;
- количество заданий, выполненных процессором;
- количество заданий, выполненных ресурсами;
- количество заданий, покинувших систему;
- среднее количество тактов ожидания обслуживания;
- количество тактов простоя процессора из-за отсутствия в очереди заданий для обслуживания;
- среднее время ожидания процесса в очереди готовых процессов;
- среднее время выполнения процесса;
- среднее время отклика системы.

Объектное моделирование

Для построения программной модели имитации процесса обслуживания заданий в вычислительной системе необходимо построить объектную модель этой системы. Для этого необходимо продумать, какие классы будут использоваться в процессе проектирования, какие у них свойства и какие методы. Необходимо также продумать, сколько будет необходимо объектов каждого класса. Нужно учитывать, что в каждом варианте свойства и методы могут существенно меняться. Кроме того, каждый студент в праве изменять по своему усмотрению и набор классов.

Ниже приводится ориентировочный вариант объектной модели.

Классы:

1) процесс (Process)
2) ресурс (Resource) (CPU, Res1, Res2, Res3)
3) очередь к ресурсу (QueueRes) (Res1, Res2, Res3, …)
4) очередь готовых процессов (к процессору) (QueueCPU)
5) очередь завершивших работу процессов (QueueFinished)
6) планировщик (Scheduler)
7) тактовый генератор (Timer)

1. Процесс:

Поля:

- имя;
- идентификационный номер процесса;
- интервал работы в тактах;
- время поступления процесса в систему;
- время работы процесса на центральном процессоре;
- состояние процесса;
- приоритет.

Методы:

- конструктор (инициализирует значения параметров процесса);
- финиширование (осуществляет проверку, завершил ли процесс работу на центральном процессоре);
- увеличение времени работы на центральном процессоре.

2. Ресурс:

Поля:

- имя;
- состояние ресурса;
- "указатель" на процесс.

Методы:

- конструктор (устанавливает начальное значение для ресурса);
- присвоение статуса ресурсу;
- загрузка процесса в центральный процессор.

3. Очередь к ресурсу:

Поля:

- указатель на "вход";
- указатель на "выход"

Методы:

- конструктор;
- деструктор;
- проверка на пустоту;
- очистка очереди;
- добавление элемента в очередь;
- удаление элемента очереди;

4. Очередь готовых процессов:

Поля:

- указатель на "вход";
- указатель на "выход".

Методы:

- конструктор;
- деструктор;
- проверка на пустоту;
- очистка очереди;
- добавление элемента в очередь;
- удаление элемента очереди;
- динамическое повышение приоритета.

5. Планировщик:

Поля:

- данные о ресурсе (CPU, Resl, Res2, Res3,…);
- данные об очереди (q);
- данные об очередях к ресурсам (QRes1, QRes2, QRes3, …)

Методы:

- конструктор;
- новый такт (производит изменения в системе при увеличении такта).

6. Тактовый генератор:

Поля:

- номер такта.

Методы:

- конструктор (получение номера текущего такта);
- переход на новый такт;

Очереди

Наша система имитирует работу реальной операционной системы по организации работы процессора и ресурсов в мультипрограммном режиме работы. Количество процессов, поступающих в систему, не ограничено. Вследствие чего, очередь готовых процессов и очереди к ресурсам не должны быть

ограничены определенным количеством элементов. А, следовательно, они могут быть реализованы программно в виде связанных списков либо динамических массивов. В зависимости от заданного вариантом способа организации очереди по вашему усмотрению выбирается однонаправленный либо двунаправленный список, либо динамический массив.

В курсовой работе применяется два типа очередей. Очереди к ресурсам всегда организованы по принципу FIFO, а очередь готовых процессов в соответствии с вариантом и отличаются по способу организации очереди в соответствии с стратегией планирования.

Очередь (***queue***) – это линейный список элементов, допускающий добавление элементов на одном конце списка, а удаление элементов на другом его конце. То есть элементы удаляются в порядке поступления, или, другими словами, очередь обслуживается в порядке FIFO (firs-in / first-out) (первый пришёл / первый ушёл) или FCFS (first-come / first-served) (первым пришёл / первым обслужен).

Классические примеры очередей – обслуживание клиентов в очереди (такими клиентами могут быть, например, процессы операционной системы) и буферизация задач принтера в системе входных и выходных потоков принтера.

Как и для стека, условие полноты очереди формулируется только, если для хранения элементов очереди используется массив.

Очереди приоритетов

Очередь приоритетов отличается от классической FIFO очереди тем, что из нее выбирается не самый старый, а самый важный элемент (с высшим приоритетом), оцениваемый по некоторому критерию, который различает элементы в списке.

Очереди приоритетов часто используются в операционных системах при планировании процессов.

Существуют два подхода к организации очереди приоритетов:

а) Очередь не упорядочена. Новый элемент всегда помещается в конец очереди. При удалении элементов из очереди ищется первый, поступивший из

элементов с наивысшим приоритетом, упрощается вставка нового элемента и затрудняется поиск минимального элемента.

б) Очередь упорядочена. Новый элемент помещается сразу на своё место. Из очереди всегда удаляется первый элемент. Вставка нового элемента в отсортированный список требует просмотра в среднем половины элементов списка.

Рекомендации по программированию

Приступая к реализации объектной модели необходимо продумать, где будут находиться созданные классы. Хорошим стилем программирования считается размещение каждого класса в своём отдельном модуле. Это улучшает читаемость программы, уменьшая длину файлов исходных текстов. Однако, следует постоянно следить за появлением перекрёстных ссылок на используемые модули). Полезно составлять схемы взаимоподключения модулей.

Критерии оценивания

Соответствие заданию	Реализация				Пояснительная записка		Всего
	Эффективность программного кода	Стабильность функционирования	Функциональ-ность	Эргономика	Качество описания	Соответствие ГОСТу	
20	20	15	15	10	10	10	**100**

СПИСОК ЛИТЕРАТУРЫ

1. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. –СПб.: Питер, 2002. – 544 с.
2. Д. Цикритзис, Ф. Бернстайн. Операционные системы / пер. с англ. –М.: Мир, 1977. –336с.
3. П. Кейлингерт. Элементы операционных систем. Введение для пользователей / пер. с англ. –М.: Мир, 1985. -295с.
4. А. Шоу. Логическое проектирование операционных систем / пер. с англ. –М.: Мир, 1981. –360 с.
5. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация (+CD). Классика CS. 3-е изд. — СПб.: Питер, 2007. — 704 с: ил.
6. Ахо А., Хопкрофт Д., Ульман Д. – Структуры данных и алгоритмы.: Пер. с англ.: Уч. пос.- М., Издательский дом «Вильямс», 2016. – 400 с.