МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КУРСОВАЯ РАБОТА

По дисциплине «Операционные Системы»

Студент Дунаев Н.Ю. шифр 012758403 группа АВТ-610

Тема: «Управление процессами B403»

Срок представления работы (проекта) к защите «24» мая 2018 г.

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Коршикова Л.А. (подпись, дата) (инициалы, фамилия)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

г. Новосибирск

2018

Оглавление

[1. Введение 3](#_Toc514883209)

[1.1 Состояния процессов 3](#_Toc514883210)

[1.2 Планирование процессов 4](#_Toc514883211)

[1.3 Вытесняющие и невытесняющие алгоритмы планирования 6](#_Toc514883212)

[1.4 Функции планировщика-диспетчера 8](#_Toc514883213)

[1.5 Общие сведения о планировании задания 9](#_Toc514883214)

[1.6 Общие сведения о диспетчеризации 10](#_Toc514883215)

[1.7 Описание и работа симулятора «диспетчер» 11](#_Toc514883216)

[1.8 Порядок работы симулятора 12](#_Toc514883217)

[1.9 Дисциплины обслуживания (ДО) очередей 12](#_Toc514883218)

[1.9.1 Требования к ДО 13](#_Toc514883219)

[1.9.2 Классификация ДО 13](#_Toc514883220)

[2. Цель работы 29](#_Toc514883221)

[3. Раздел 1 30](#_Toc514883222)

[3.1 Задание 30](#_Toc514883223)

[3.2 Вычисление исходных данных 31](#_Toc514883224)

[3.2.1 Расчет и 32](#_Toc514883225)

[3.2.2 Формирование исходных данных 32](#_Toc514883226)

[3.3 Временная диаграмма при использовании ДО FIFO 33](#_Toc514883227)

[3.4 Временная диаграмма при использовании ДО SJF 36](#_Toc514883228)

[3.5 Сравнительный анализ двух диаграмм 37](#_Toc514883229)

[4. Раздел 2 38](#_Toc514883230)

[4.1 Задание 38](#_Toc514883231)

[4.2 Бесприоритетные ДО (FIFO) 38](#_Toc514883232)

[4.3 Приоритетные ДО (P = f( 39](#_Toc514883233)

[4.4 Программная реализация диспетчера 40](#_Toc514883234)

[5. Заключение 46](#_Toc514883235)

[6. Список литературы. 47](#_Toc514883236)

# Введение

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

## Состояния процессов

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:

* ВЫПОЛНЕНИЕ – активное состояние процесса, во время которого он обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
* ОЖИДАНИЕ – пассивное состояние процесса, он заблокирован, не может выполняться по своим внутренним причинам, 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 он заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако CPU занят выполнением другого процесса.

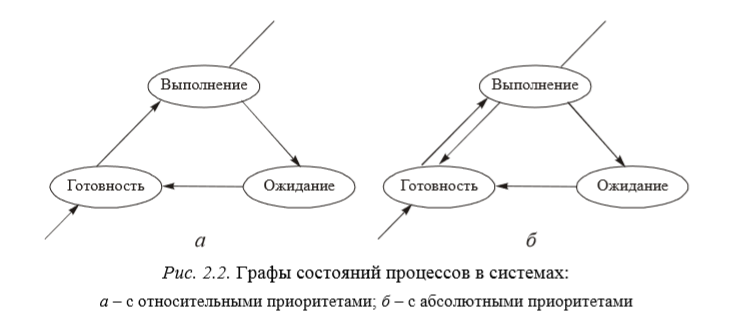
В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования, реализуемым в данной операционной системе. Типичный граф состояний процесса показан на рис. 2.1.

В состоянии ВЫПОЛНЕНИЕ в однопроцессорной системе может находиться только один процесс, а в каждом из состояний ОЖИДАНИЕ и ГОТОВНОСТЬ – несколько, которые образуют очереди соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния ГОТОВНОСТЬ, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние ВЫПОЛНЕНИЕ и находится в нем до тех пор, пока либо он сам не освободит процессор, перейдя в состояние ОЖИДАНИЯ какого-нибудь события, либо не будет насильно «вытеснен» из процессора, например, вследствие исчерпания отведенного данному процессу кванта процессорного времени. В последнем случае он возвращается в состояние ГОТОВНОСТЬ. В это же состояние процесс переходит из состояния ОЖИДАНИЕ после того как ожидаемое событие произойдет.

## 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 перечисленные выше задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы, наиболее часто встречающиеся: алгоритмы, основанные на квантовании, и алгоритмы, основанные на приоритетах.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если: процесс завершился и покинул систему; произошла ошибка; процесс перешел в состояние ОЖИДАНИЕ; исчерпан квант процессорного времени, отведенный данному процессу. Процесс, который исчерпал свой квант, переводится в состояние ГОТОВНОСТЬ и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени. Кванты, выделяемые процессам, могут быть одинаковыми для всех или различными. Кванты, выделяемые одному процессу, могут иметь фиксированную величину или изменяться в разные периоды его жизни. Процессы, которые не полностью использовали выделенный им квант (например, из-за ухода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при последующем обслуживании. По-разному может быть организована очередь готовых процессов: циклически, по правилу «первый пришел – первый обслужился» (FIFO) или по правилу «последний пришел – первый обслужился» (LIFO). Другая группа алгоритмов использует понятие «приоритет» процесса. Приоритет – это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности процессорного времени: чем выше приоритет, тем выше привилегии. Приоритет может выражаться целыми или дробными, положительными или отрицательными значениями. Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях. Приоритет может назначаться директивно администратором системы в зависимости от важности работы или внесенной платы либо вычисляться самой ОС по определенным правилам, он может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими. Существует две разновидности приоритетных алгоритмов: алгоритмы, использующие относительные приоритеты, и алгоритмы, использующие абсолютные приоритеты. В обоих случаях выбор процесса на выполнение из очереди готовых осуществляется одинаково: выбирается процесс, имеющий наивысший приоритет. По-разному решается проблема определения момента смены активного процесса. В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ОЖИДАНИЕ (или же произойдет ошибка, или процесс завершится). В системах с абсолютными приоритетами выполнение активного процесса прерывается еще при одном условии: если в очереди готовых процессов появился новый, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности. На рис. 2.2 показаны графы состояний процесса для алгоритмов с относительными (а) и абсолютными (б) приоритетами.



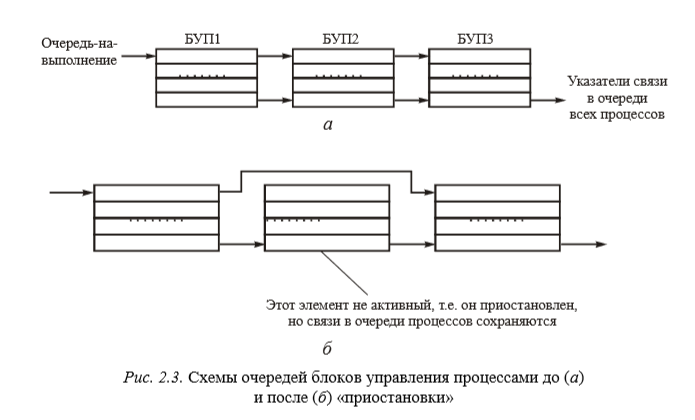
Графы состояний процессов в системах: а – с относительными приоритетами; б – с абсолютными приоритетами Во многих операционных системах алгоритмы планирования построены с использованием как квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора процесса из очереди готовых определяется приоритетами процессов.

## Вытесняющие и невытесняющие алгоритмы планирования

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 на выполнение процесс. Preemptive multitasking (вытесняющая многозадачность) – это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого принимается планировщиком операционной системы, а не самой активной задачей. Вытесняющая и невытесняющая многозадачность – это более широкие понятия, чем типы приоритетности. Приоритеты задач могут как использоваться, так и не использоваться и при вытесняющих, и при невытесняющих способах планирования. В случае относительных приоритетов эта дисциплина может быть отнесена к классу систем с невытесняющей многозадачностью, а дисциплина абсолютных приоритетов – к классу систем с вытесняющей многозадачностью. Бесприоритетная же дисциплина планирования, основанная на выделении равных квантов времени для всех задач, относится к вытесняющим алгоритмам. Основным различием между preemptive и non-preemptive вариантами многозадачности является степень централизации механизма планирования задач. При вытесняющей многозадачности механизм планирования задач целиком сосредоточен в операционной системе и программист пишет свое приложение, не заботясь о том, что оно будет выполняться параллельно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активной задачи, запоминает ее контекст, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст. При невытесняющей многозадачности механизм планирования распределен между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения своей очередной итерации и передает управление ОС с помощью какого-либо системного вызова, а ОС формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например, с учетом приоритетов) следующую задачу на выполнение. Такой механизм создает проблемы как для пользователей, так и для разработчиков. Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Если приложение тратит слишком много времени на выполнение какой-либо работы, например на форматирование диска, пользователь не может переключиться с этой задачи на другую, например на текстовый редактор, в то время как форматирование продолжалось и в фоновом режиме. Эта ситуация нежелательна, так как пользователи обычно не хотят долго ждать, когда машина завершит свою задачу. Поэтому разработчики приложений для non-preemptive операционной среды, возлагая на себя функции планировщика, должны создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Например, программа форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы та отформатировала следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста. Программист должен обеспечить «дружественное» отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая им управление. Крайним проявлением «недружественности» приложения является его зависание, которое приводит к общему краху системы. В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, поскольку центральный планирующий механизм снимет «зависшую» задачу с выполнения. Однако распределение функций планировщика между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, потому что дает возможность разработчику приложений самому проектировать алгоритм планирования, наиболее подходящий для данного фиксированного набора задач. Так как разработчик сам определяет в программе момент времени отдачи управления, то при этом исключаются нерациональные прерывания программ в «неудобные» для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждой итерации использует их монопольно и пользователь уверен, что на протяжении этого периода никто другой не изменит эти данные. Существенным преимуществом non-preemptive систем является более высокая скорость переключения с задачи на задачу. Примером эффективного использования невытесняющей многозадачности является файл-сервер NetWare, в котором в значительной степени достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование невытесняющей многозадачности в операционной среде Windows 3.х. Однако почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows NT, OS/2, VAX/VMS), реализована вытесняющая многозадачность. В последнее время дошла очередь и до ОС класса настольных систем, например, OS/2 Warp и Windows 95. Возможно, в связи с этим вытесняющую многозадачность часто называют истинной многозадачностью.

## Функции планировщика-диспетчера

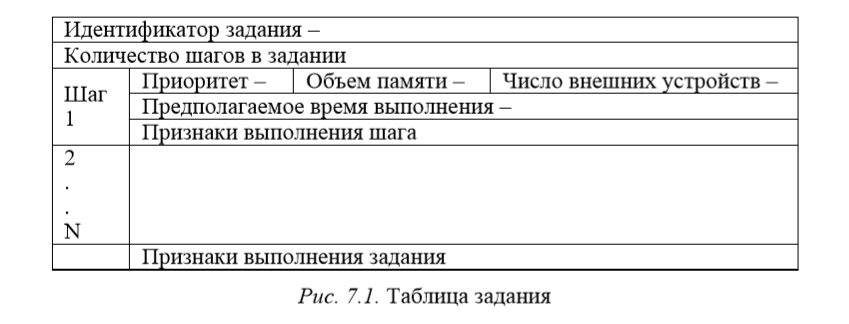
Во многих ОС эти функции могут быть представлены неразрывной последовательностью, поэтому введем термин планировщик-диспетчер. Планировщик – это программа, ответственная за постановку процессов в очередь-на-выполнение и управляющая этой очередью. Диспетчер – это программа, которая выбирает процессы из очереди-на-выполнение, переводит их в активное состояние и передает им контроль над CPU. Основная функция – возможность управлять действиями большого числа процессов. Приостановка процесса. В ходе своего выполнения системный процесс может установить, что требуемый ресурс занят или в данный момент ему не требуется производить каких-либо действий. Тогда процесс может «сознательно» приостановить свое выполнение до момента активизации его другим процессом (рис. 2.3). При этой процедуре адрес блока управления процессом (БУП) записывается в стек, затем для активизации другого процесса вызывается системный планировщик. Результатом этого действия является то, что «приостановленный» процесс не помещается в очередь-на-выполнение и не активизируется.



## Общие сведения о планировании задания

Функцией службы управления процессом является распределение аппаратных ресурсов центрального процессора. Можно выделить следующие компоненты этой службы: планировщик заданий; планировщик задач (планировщик процессов). Задание представляет собой описание комплекса работ, которые пользователь хочет выполнить на ЭВМ. Этот комплекс может быть представлен в виде последовательности некоторых частных работ, описываемых с помощью шагов задания. Из шагов задания формируются задачи. Для выполнения задач система создает процессы.

Объектами работы планировщика заданий (ПЛАНИРОВЩИКА) являются задания, а планировщик задач управляет процессами. ПЛАНИРОВЩИК решает, какие задания и в какой последовательности должны поступать на обработку (своего рода «макропланировщик»). Планировщик задач выступает в роли «микропланировщика», распределяющего процессор между процессами. В случае мультипрограммирования ПЛАНИРОВЩИК выбирает несколько заданий из множества всех представленных и вводит их в систему. Для каждого задания формируется таблица задания JCB (Job Control Block). Пример такой таблицы приведен на рис. 7.1.



Для программного выполнения заданий ПЛАНИРОВЩИК создает процессы и выделяет им необходимые ресурсы (память, внешние устройства). Планировщик процессов должен решить, какому из созданных процессов предоставить процессор, в какой момент и на какое время. ПЛАНИРОВЩИК обеспечивает определенную дисциплину выбора заданий на обработку. Для принятия решения можно учитывать такие характеристики заданий, как приоритет, необходимые ресурсы и т.п. ПЛАНИРОВЩИК не только выделяет необходимые ресурсы для поступающего на обработку задания, но и освобождает ресурсы после выполнения задания.

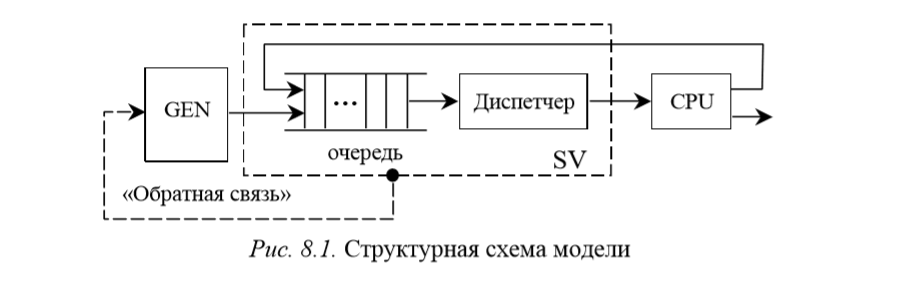
## Общие сведения о диспетчеризации

Планирование распределения процессора производится на нескольких уровнях. Один из них – диспетчеризация. На этом уровне диспетчер задач (планировщик процессов) выбирает одну задачу из числа готовых к выполнению и предоставляет ей процессор. Каждая задача занимает процессор относительно малое время (как правило, недостаточное для выполнения задачи), затем диспетчирование повторяется – процессор выделяется другой задаче. Диспетчер принимает текущие решения в динамике сложившейся конкретной обстановки. Таким образом, цели диспетчирования задач следующие: распределение центрального процессора в динамике в соответствии с критериями; эффективная отработка алгоритмов управления задачами; сбалансированное использование ресурсов; баланс между временем ответа и коэффициентом использования ресурсов. Итак, диспетчер – это программа, которая выбирает задачи (процессы) из очереди на выполнение, переводит их в активное состояние и передает на обработку центральному процессору.

## Описание и работа симулятора «диспетчер»

Для изучения некоторых наиболее распространенных правил обслуживания задач, стоящих в очереди на выполнение (тема 2), предлагается модель простейшей ЭВМ (симулятор диспетчеризации).

Структурная схема, поясняющая принцип работы этой ЭВМ, представлена на рис. 8.1.



В структуре выделено три основных блока ЭВМ: генератор (GEN); супервизор (SV); центральный процессор (CPU). Взаимодействие этих блоков и обеспечивает обработку задач (процессов). Замечание. В модели ведется отсчет времени относительно модельного времени главных часов (TM). Наряду с ТМ вводятся понятия: TSV – время работы SV; TGEN – время работы GEN; TCPU – время работы CPU над проблемной задачей (квант времени); TTGEN – квант времени, по истечении которого блок генератора должен вступить в работу; OST – остаток от кванта времени, отпущенного процессору для работы над задачей (время доработки). Блоком GEN формируются основные параметры задачи: номер, приоритет, трудоемкость, время поступления в систему следующей задачи. Эти параметры формируются, исходя из максимально заданных значений входных величин генератора случайных чисел. Блок генератора вступает в работу по истечении TTGEN, при этом текущая работа прерывается и возобновляется после постановки вновь поступившей задачи в очередь на обработку. Блок SV выполняет роль диспетчера задач.

## Порядок работы симулятора

1. Супервизор ожидает поступления в систему какой-либо задачи.

2. После поступления задачи в систему ей выделяется квант процессорного времени, по истечении которого вновь будет запущен супервизор.

3. При очередном запуске супервизора происходит анализ трудоемкости задачи.

4. Пункты 2 и 3 повторяются до тех пор, пока задача не отработает отведенное ей время целиком.

5. Через некоторые промежутки времени запускается генератор новой задачи, при этом происходит запоминание текущего состояния системы.

6. Пункты 1–5 повторяются, пока не будут выполнены все задачи.[1]

## Дисциплины обслуживания (ДО) очередей

1.  На каждый из планировщиков поступает поток работ (заявок).

2.  Формируются очереди работ.

3.  Имеется ресурс, способный выполнить соответствующую работу.

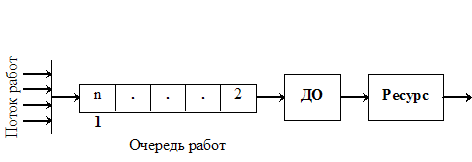
*4.  Обслуживание без отказа* – заявка не покидает систему пока не обслужится. (среднее время обслуживания заявок в системе (очереди) не зависит от дисциплины обслуживания).

5.  Заявки имеют параметры:

* время обслуживания (трудоемкость);
* приоритет.

6.  Выбор заявок из очереди производится по правилу, которое называется дисциплиной обслуживания.

          На рисунке 4.1 представлена схема одноканальной системы без отказов

**

*Рис. 4.1.*Схема одноканальной системы без отказов

Замечание:Обслуживание заявок из очереди носит в общем случае стохастический характер, т.е. время обслуживания различно.

tнахождения\_в\_системе= tвыпол.+ tожид.

### Требования к ДО

* должны обеспечивать показатель эффективности обслуживания, т.е. tожид. должно быть равномерным;
* трудоемкость ДО должна быть минимальной.

          Приоритет– это преимущественное право на первоочередное обслуживание. Он устанавливается на основе статических и динамических характеристик заявок, на основе трудоемкости и на основе внешнего приоритета. Приоритет выступает как последовательность чисел, низшее число считается высшим приоритетом.

### Классификация ДО

* бесприоритетные ДО
  + линейные
    - в порядке поступления (FIFO)
    - в инверсном порядке (LIFO)
    - случайный выбор (RAND)
  + циклические ДО
    - циклический алгоритм (RR)
    - многоуровневый циклический алгоритм (FB)
    - смешанный алгоритм
* приоритетные ДО
  + с фиксированным приоритетом
    - ДО с относительным приоритетом (ОП)
    - с абсолютным приоритетом (АП)
    - адаптивное обслуживание
  + с динамическим приоритетом
    - в зависимости от tожид.
    - в зависимости от tобслуж.

**Среднее время пребывания заявок в состоянии ожидания (tожид.)**

зависит от:

* быстродействия ЦП;
* дисциплины обслуживания;
* трудоемкости задания;
* интенсивности решаемых задач.

**Среднее время ожидания** это есть: W=f (B, ДО, T, IZ, M), где

B – быстродействие;

ДО – дисциплина обслуживания;

T - трудоемкость;

IZ – интенсивность задач;

M – число задач (работ).

#### Бесприоритетные дисциплины обслуживания (БДО)

БДО**–** выбирают заявки без учета их важности, например, по признаку последовательности поступления*.*

**Линейные ДО**характеризуются одинаковым среднем временем ожидания не зависимо от длительности заявки (т.е. трудоемкости).

Примеры линейных ДО:

        FIFO – "первый пришел, первый вышел";

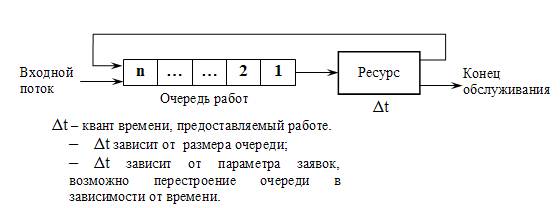
        LIFO – принцип стека;

        RAND –обслуживание в произвольном порядке.

**Циклические ДО**

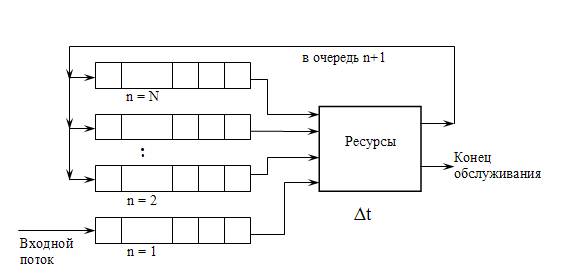
1.  RR – Robin Round (круговой циклический алгоритм)

На рисунке 4.2 представлена схема циклической ДО (RR)

*Рис.4.2*Схема циклической ДО (RR)

2.  FB – Feed Back (обратная связь) – многоуровневый циклический алгоритм

     На рисунке 4.3 представлена  схема циклической ДО (FB)

**

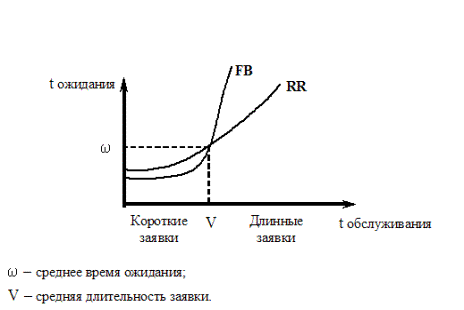
*Рис. 4.3.*Схема циклической ДО (FB)

n=1 – это первая очередь, в нее поступает входной поток заявок. Из нее заявка поступает на ресурс и/или полностью обслуживается или снова поступает в очередь, но с номером на 1 больше.

Поток заявок поступает в самую приоритетную очередь n=1. Заявка получает квант и переходит в очередь n+1. Заявка в i-ой очереди обслуживается,

если пусты все остальные очереди. В очереди N заявки обслуживаются до завершения (в очереди N принцип FIFO + RR).

          На рисунке 4.4 представлена временная диаграмма качества обслуживания для бесприоритетных дисциплин.



*Рис. 4.4*Временная диаграмма качества обслуживания для

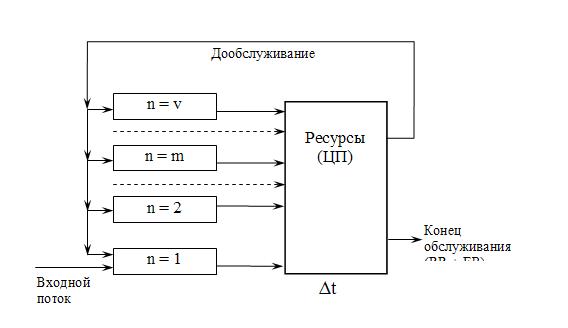
бесприоритетных дисциплин

**Замечания:**

* для RR - каждой вычислительный процесс или заявка получает процессор мощностью 1/l, где l-длина очереди (т.е. время ответа пропорциональной времени обслуживания);
* для RR - чем меньше величина кванта времени, выделяемого заявке, тем лучше пропорциональность по отношению ко всем заявкам, тем лучше обслуживаются короткие заявки.

3.  Смешанный алгоритм обслуживания (RR+FB)

На рисунке 4.5 представлена схема алгоритма обслуживания с учетом ДО (RR и FB)

**

*Рис.4.5.*Схема алгоритма обслуживания с учетом ДО (RR и FB)

     Каждая заявка проходит в i-ой очереди несколько кругов и только потом переходит в очередь i+1.

Достоинства смешанного алгоритма:

* сокращаются накладные расходы на перевод в другие очереди;
* возможно подобрать параметры алгоритма под традиционный поток заявок (те. можно варьировать http://ermak.cs.nstu.ru/~mos/index_7.files/image010.jpgt).

**Выводы к бесприоритетным ДО**

1.   Линейные ДО характеризуются одинаковым средним временем ожидания.

2.   Циклические ДО обслуживают короткие заявки с неявным приоритетом.

3.   Бесприоритетые ДО не требуют предварительной информации о длительности заявок.

4.   Уменьшение длительности ожидания коротких заявок происходит за счет увеличения tожид. длинных заявок.

#### Приоритетные дисциплины обслуживания (ПДО)

Принцип заложенный для приоритетов: HPF – highest priority first (наивысший приоритет - первым).

Приоритет – это право на первоочередное обслуживание.

ПДО – характеризуется более сложной работой с очередью, а именно:

* формирование очереди;
* обслуживание заявок из очереди.

Очередь:

* упорядоченная
  + формирование-перебор
  + обслуживание – первая заявка из очереди
* частично упорядоченная
  + формирование – первая
  + обслуживание - первая
* неупорядоченная
  + формирование – добавить в конец
  + обслуживание – поиск

Для упорядоченной очереди – заявки размещаются в соответствии с приоритетом.

Для неупорядоченной очереди - заявки размещаются в порядке поступления, но на выбор самой приоритетной требуется время.

Для частично упорядоченной очереди - очередь упорядочивается с периодом Т.

Назначение приоритета заявкам происходит на основе статистических характеристик:

* ожидаемый объем ввода/вывода;
* априорное tобслуж.;
* внешний приоритет;
* ожидаемый объем ОП,

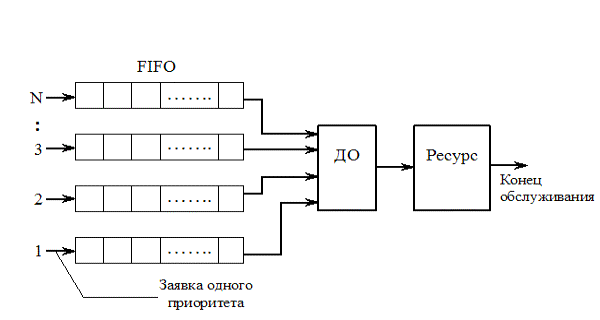
на основе динамических характеристик:

* время присутствия в системе;
* объем вводимых/выводимых операций в предыдущий момент времени.

##### **ДО с фиксированным приоритетом**

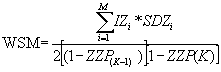
1. ДО с относительным приоритетом.

Выполнение приоритетной заявки не прерываются при поступлении более приоритетной. На рисунке 4.6 представлена схема алгоритма обслуживания ДО(относительный приоритет ОП).



*Рис. 4.6.*Схема алгоритма обслуживания ДО (относительный приоритет ОП)

Если поступило несколько заявок одного приоритета, то в очереди они обслуживаются по принципу FIFO.

 , где

http://ermak.cs.nstu.ru/~mos/index_7.files/image016.gif;

http://ermak.cs.nstu.ru/~mos/index_7.files/image018.gif;

WSM – среднее время пребывания заявки в системе;

    М – число заданий;

IZ – интенсивность заданий;

    ZZ – загрузка ЦП;

SDZ – среднее время обслуживания;

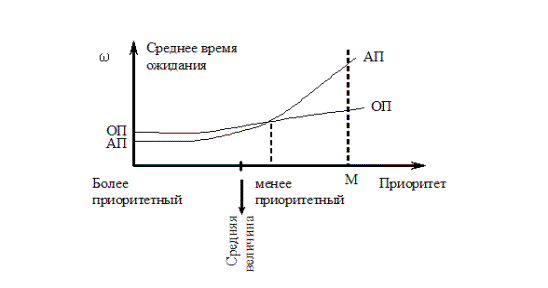
   К – какое-либо задание.

ZZP – коэффициент использования центрального

процессора (ЦП);

2. ДО с абсолютным приоритетом.

         На рисунке 4.7 представлена схема алгоритма обслуживания ДО (абсолютный приоритет - АП)



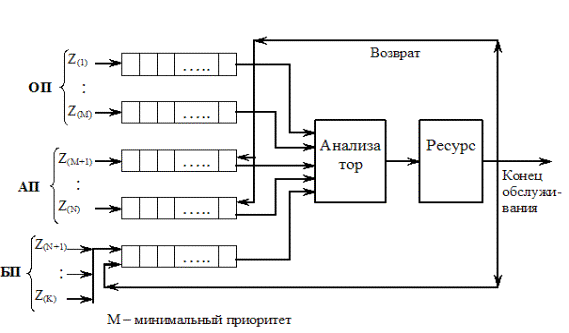
*Рис. 4.7.*Схема алгоритма обслуживания ДО (абсолютный приоритет - АП)

          Приоритетная заявка может прервать выполнение менее приоритетной. Приоритет складывается из статических и динамически составляющих:

DZ – трудоемкость

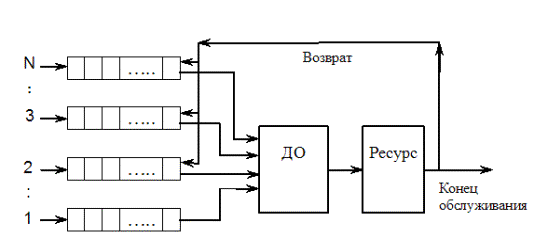
http://ermak.cs.nstu.ru/~mos/index_7.files/image022.gif

          На рисунке 4.8 представлена диаграмма качества обслуживания для приоритетных дисциплин.



*Рис.4.8.*Диаграмма качества обслуживания для приоритеных дисциплин

     На рисунке 4.9 представлена схема обслуживания заявок в смешанном режиме



*Рис.4.9.*Схема обслуживания заявок в смешанном режиме

Смешаный режим – когда различным группам работ присваиваются различные виды приоритетов. Используется этот метод тогда, когда ни одна из частных ДО не обеспечивает выполнения заданных ограничений.

ZZ(1)\*WSM(1)+**…..+** ZZ(K)\*WSM(K)=const

WSM – среднее время пребывания i работы в системе;

ZZ – загрузка ЦП(%);

БП - безприоритетные ДО.

При изменении ДО время ожидания работ одних типов сокращается за счет увеличения tожид.других типов.

3.Адаптивные ДО

В этом случае решения об абсолютном или относительном приоритетах принимаются в зависимости от обстановки. Для решения этого вопроса взвешивается - «что стоит» прерывание.

Пусть Сi – «стоимость» ожидания заявки для i-ой работы (т.е. высокоприоритетная), Сj – «стоимость» ожидания заявки для j-ой работы (низкоприоритетной), т.е. j>i (отсчет от меньшего к большему).

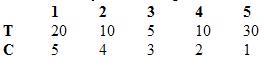
С – коэффициент, штраф за задержку заявки на единицу времени (целое число);

Тj – время обслуживания j-ой заявки;

Тi – время обслуживания i-ой заявки, тогда, если ТjСi <Тi Сj, то работа (заявка) не прерывается.

Если будет прервана, то будет «стоить» Тi Сj, если не будет прервана, то будет «стоить» ТjСi..

ПРИМЕР.Спроектировать обработку заявок с соответствующими требованиями



Т4С1<  Т1С4– не выполняется, т.е. 50 < 40 – прерываем, т.к. условие прерывания не выполняется

Штраф за задержку заявки- это отказ предоставления ресурса заявке на время t или понижение приоритета заявки и др.

**Недостатки дисциплин с фиксированным приоритетом**

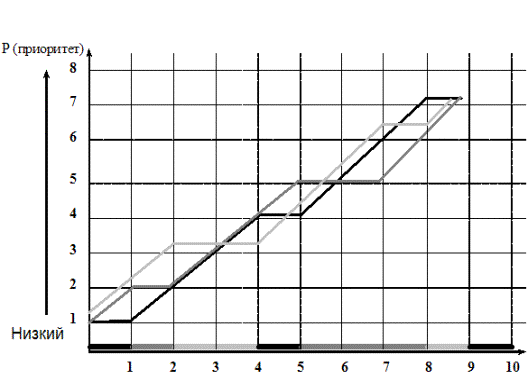
* неточность априорных характеристик приводит к приблизительности приоритета;
* возможность приспосабливаемости пользователей к системе приоритета. Если пользователю известно о системе приоритетов т.е. больше «уважают» короткие заявки в системе, то пользователь составит 2 короткие вместо 1-ой длинной, но тогда система становится на уровень FIFO;
* при хороших и средних характеристиках параметра tожидания, отдельные заявки в системе могут длительное время ждать. Для устранения этого недостатка надо бы повысить приоритет задачи, но т.к. он фиксирован, это невозможно;
* отсутствует настраиваемость системы на изменение характеристик входного потока заявок.

##### **ДО с динамическим приоритетом**

          Приоритет изменяется в период пребывания в системе (вычислительной), либо в процессе tожид. , либо в процессе tвыполн

На рисунке 4.10 представлена Временная диаграмма изменения приоритета от времени.

          Если низкоприоритетная заявка ожидает длительное время, необходимо повысить приоритет.Для ДО с линейно возрастающим приоритетом характерна зависимость:



Обозначения на диаграмме: Pi(t) = ai tожидан. + bi tвыполн., где Pi(t) – приоритет i- ой работы

*Рис. 4.10.*Временная диаграмма изменения приоритета от времени.

На временной диаграмме рассмотрено обслуживание трех заявок в системе.

**Достоинства системы с динамическим приоритетом:**

* возможность настраиваться при изменении характеристик входного потока на эффективное обслуживание;
* дисциплина основана на реальных динамических характеристиках заявок.

**Недостаток: сложность реализации.**

Вывод к приоритетным ДО - они сокращают tожид. высокоприоритетных заявок за счет увеличения tожид. низкоприоритетных.

**Критерии эффективности организации вычислительных процессов**

          Показатель эффективности – это производительность ЭВМ.

Производительность ЭВМ характеризуется параметрами:

* пропускная способность – объем работы в единицу времени;
* время ответа – промежуток от момента приема задания пользователя до получения ответа;
* коэффициент готовности – степень готовности обработать запрос на вычисление.

          Соотношение перечисленных факторов и назначения системы:

* система реального времени – МINtответа;
* пакетная обработка – MAX пропускная способность;
* система разделения времени (СРВ) – МАХ пропускной способности при соответствующих (разумных) ограничениях на tответа.

**Критерии эффективности вычислительного процесса**

1.  Среднее время ответа – время пребывания работ в системе (среднее время обращения).

http://ermak.cs.nstu.ru/~mos/index_7.files/image032.gif где n – число заданий

                               tзi – время завершения i-ой работы;

                               tni – время поступления i-ой работы.

2.  Среднее взвешенное время обращения

http://ermak.cs.nstu.ru/~mos/index_7.files/image034.gif где ti – трудоемкость i-ой работы (время нахождения на цикле)

W – определяет во сколько раз время ответа превышает требуемое время выполнения (t обслуживания).

Т- суммарное (общее) время выполнения работ. Чем меньше время выполнения, тем выше загрузка ресурсов.

Количество задач за единицу времени в системе.

Суммарная загрузка всех устройств.

Штраф за задержку задач (вперед про  пустить задачу за которую больше «платят», т.е. у которой больше приоритет).

http://ermak.cs.nstu.ru/~mos/index_7.files/image036.gif

**Планирование работ в вычислительной системе**

          Планирование заданий – это верхний уровень планирования использования ресурса - центрального процессора.

**Причины сложности планирования.**

1.  Каждому заданию необходимо несколько ресурсов (процессор, ОП, МД и др.).

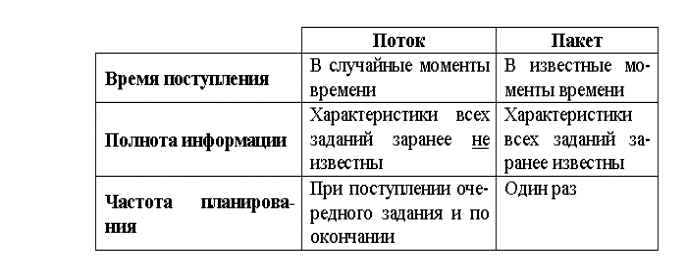
2.  Различные задания используют ресурсы многократно и в различных последовательностях.

3.  Длительность и порядок использования ресурсов могут быть заранее не известны.

4.  Планирование зависит от характера поступлений заданий в систему.

          В таблице 6.1 приведена частость планирования в вычислительной системе

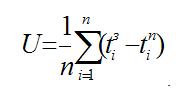
*Таблица  6.1.*Частость планирования в вычислительной системе



**Планирование заданий по критерию минимального среднего**

**времени ответа (обращения)**

          При планировании по данному критерию ориентируется на соотношение:

.

**Постановка задачи.**

1.  Существует поток из n-заданий.

2.  Каждому заданию требуется только процессор на i.

3.  Использовать ДО планировщика:

а. линейная ДО – FIFO;

б. циклическая ДО – RR;

в. приоритетное – SJF (Short Job Feast), т.е. короткое задание вперед.

4.  Построить временную диаграмму многопрограммного режима, используя ДО RR.

5.  Построить временные диаграммы однопрограммного режима, используя ДО FIFO, SJF.

6.  Оценить работу планировщика (по критериям U, W, T).

*ПРИМЕР*

По заданной матрице трудоемкости провести планирование работ в вычислительной системе, результаты планирования представить в табличном виде и в виде временных диаграмм.

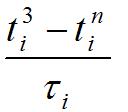
В таблице 6.2 приведены результаты планирования.

*Таблица 6.2.*Результаты планирования

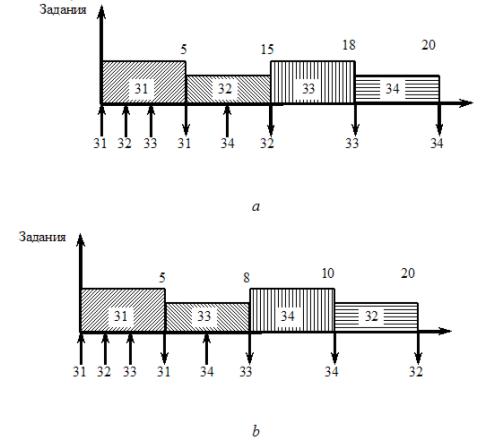


где    «1».  tзi – время завершения;

«2».  tзi - tni– время обращения;

«3». Взвешенное время обращения - 

          На рисунке 6.1 приведена временная диаграмма планирования работ : ДО FIFO (*а*), ДО SJF (*b*),  ДО RR(*c*).

******

******

*Рис.6.1.* Временная диаграмма планирования работ : ДО FIFO (*а*),

ДО SJF (*b*),  ДО RR(*c*).

*Таблица 6.3.*Таблица соответствия условиям работы вычислительной системы



          Чем больше неопределенность относительно времени вычисления, тем большую выгоду дает применение ДО RR по сравнению с ДО FIFO.

          Если задания поступают таким образом, что предыдущие успевают обслужиться, то дисциплина SJF не имеет смысла, т.к. не формируется очередь.[2]

# Цель работы

Работа делится на 2 раздела. Цель первого раздела - построить временную диаграмму мультипрограммной работы при использовании каждого из двух алгоритмов. На диаграмме выделить события (моменты поступления заданий, моменты назначения на выполнение, моменты начала счета, моменты завершения) и периоды между событиями. Для каждого периода указать процессорное время на задание, доступную память, доступную дисковую память, коэффицент мультипрограммирования. Провести сравнение двух случаев по средневзвешенному времени обращения.

Цель второго раздела - разработать структуру функционирования диспетчера работ в вычислительной системе, заданной в разделе 1. Квант времени, выделяемый каждой работе, выбирается исходя из конкретной ситуации: число работ, параллельно занимающих процессор; интервалы времени с коэффициентом многозадачности; дисциплина обслуживания

# Раздел 1

## Задание

Вычислительная система располагает оперативной памятью (ОП) - V и внешним объемом памяти - Н (НМД). Реали­зуется режим мультипрограммирования: если одновременно выполня­ется несколько задач, то процессорное время распределяется между ними равномерно. В систему поступает поток из М заданий, очеред­ное задание поступает через некоторое время, для простоты каждое задание состоит из одной задачи и требует объем ОП - vi, объем внешней памяти hi, процессорное время τi. Каждое задание использует свою внешнюю память только для ввода данных в течение времени q\*hi, после чего начинается счет. Однако закрепленные за каждым зада­нием носители освобождаются только после завершения задания. Предположим, возможно параллельное использование внешней памяти заданиями без задержки друг друга. Если бы задания выполнялись по одному, то на каждое задание было бы затрачено время Тi = q\*hi + τi. Вновь поступившие задания помещаются в очередь. Для выбора заданий из очереди а выполнение используются два алгоритма:

1. Среди заданий в очереди, для которых достаточно свободных ресурсов, выбирается задание, поступившее первым (правило FIFO).
2. Среди заданий в очереди, для которых достаточно свободных ресурсов, выбирается задание с наименьшим τi (дисциплина SJF).

Необходимо построить временную диаграмму мультипрограммной работы при использовании каждого из двух алгоритмов. На диаграмме выделить события (моменты поступления заданий, моменты назначения на выполнение, моменты начала счета, моменты завершения) и периоды между событиями. Для каждого периода указать процессорное время на задание, доступную память, доступную дисковую память, степень мультипрограммирования. Провести сравнение двух случаев по средневзвешенному времени обращения:

,

где - время завершения задания,

- время поступления задания в систему.

## Вычисление исходных данных

Для нормирования различных вариантов последовательностей заданий используется набор из 10 типов задач (см. таблицу 1). Каждое задание включает одну из этих 10 задач. В одном потоке заданий могут встретиться задания, содержащие одинаковые задачи. Номер задачи Кi для очередного задания определяется по формулам:

Xi = [7 \* Xi-1 + 417] mod 1000;

Ki = [Xi / 7] mod 10, i=1÷M, Xo = N,

где [c] - целая часть числа с,

y mod z - остаток от деления y на z,

Xo = 403 (последние три цифры из зачетной книжки; если четное число, то +1, чтобы получилось нечетное);

= + Кi - интервал поступления заданий ();

M=10 - число заданий.

### Расчет и

### Формирование исходных данных

Таблица 1. Исходные данные.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 403 | 238 | 83 | 998 | 403 | 238 | 83 | 998 | 403 | 238 |
| № Задачи | 7 | 4 | 1 | 2 | 7 | 4 | 1 | 2 | 7 | 4 |
| ОП, | 9 | 3 | 3 | 2 | 9 | 3 | 3 | 2 | 9 | 3 |
| Вп, | 1 | 2 | 4 | 3 | 1 | 2 | 4 | 3 | 1 | 2 |
|  | 30 | 60 | 90 | 20 | 30 | 60 | 90 | 20 | 30 | 60 |
|  | 7 | 11 | 12 | 14 | 21 | 25 | 26 | 28 | 35 | 39 |
|  | 5 | 10 | 20 | 15 | 5 | 10 | 20 | 15 | 5 | 10 |
|  | 35 | 70 | 110 | 35 | 35 | 70 | 110 | 35 | 35 | 70 |

Значение используемых параметров вычислительной системы:

V=16 - Оперативная память

H=12 - Внешняя память

q=5 - Коэффициент для вычисления времени ввода данных задания с внешней памяти ( q⋅hi ), после которого будет произведён счет

## Временная диаграмма при использовании ДО FIFO

Таблица 2. Временная диаграмма мультипрограммной работы ЭВМ (ДО FIFO).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | События | ОП  ( ΔОП) | ВУ  ( ΔВУ) | Коэф. Мульт. |
| 0 | Заданий нет. Простой системы | 16 | 12 | 0 |
| 7 | Поступило задание 1. Свободных ресурсов хватает. Начинается ввод задания 1 (5 квантов). | 16 (-9) | 12 (-1) | 0 |
| 11 | Поступило задание 2. Свободных ресурсов хватает. Начинается ввод задания 2 (10 квантов). Продолжается ввод задания 1. | 7 (-3) | 11 (-2) | 0 |
| 12 | Поступило задание 3. Свободных ресурсов хватает. Начинается ввод задания 3 (20 квантов). Завершен ввод задания 1. Продолжается ввод задания 2. Процессорное время выделяется заданию 1 | 4 (-3) | 9 (-4) | 1 |
| 14 | Поступило задание 4. Свободных ресурсов не хватает, задание 4 помещается в очередь. Процессорное время выделяется заданию 1. Продолжается ввод заданий 2,3. | 1 | 5 | 1 |
| 21 | Поступило задание 5. Свободных ресурсов не хватает, задание 5 помещается в очередь. Завершен ввод задания 2. Процессорное время выделяется заданиям 1,2. В очереди задания 4,5. Продолжается ввод задания 3. | 1 | 5 | 2 |
| 25 | Поступило задание 6. Свободных ресурсов не хватает, задание 6 помещается в очередь. Процессорное время выделяется заданиям 1, 2. В очереди задания 4,5,6. Продолжается ввод задания 3. | 1 | 5 | 2 |
| 26 | Поступило задание 7. Свободных ресурсов не хватает, задание 7 помещается в очередь. Продолжается ввод задания 3. Процессорное время выделяется заданиям 1, 2. В очереди задания 4,5,6,7. | 1 | 5 | 2 |
| 28 | Поступило задание 8. Свободных ресурсов не хватает, задание 8 помещается в очередь. Продолжается ввод задания 3. Процессорное время выделяется заданиям 1, 2. В очереди задания 4,5,6,7,8. | 1 | 5 | 2 |
| 32 | Завершен ввод задания 3. Процессорное время выделяется заданиям 1, 2, 3. В очереди задания 4,5,6,7,8. | 1 | 5 | 3 |
| 35 | Поступило задание 9. Свободных ресурсов не хватает, задание 9 помещается в очередь. Процессорное время выделяется заданиям 1, 2, 3. В очереди задания 4,5,6,7,8,9. | 1 | 5 | 3 |
| 39 | Поступило задание 10. Свободных ресурсов не хватает, задание 10 помещается в очередь. Процессорное время выделяется заданиям 1, 2, 3. В очереди задания 4,5,6,7,8,9,10. | 1 | 5 | 3 |
| 79 | Задание 1 завершено. Ресурсы освобождены. Начинается ввод заданий 4 (15 квантов),6 (10 квантов). Процессорное время выделяется заданиям 2, 3. В очереди задания 5,7,8,9,10. | 1 (+9-2-3) | 5(+1-3-2) | 2 |
| 89 | Завершился ввод задания 6. Процессорное время выделяется заданиям 2, 3, 6. | 5 | 1 | 3 |
| 94 | Завершился ввод задания 4. Процессорное время выделяется заданиям 2, 3, 4, 6. В очереди задания 5,7,8,9,10. | 5 | 1 | 4 |
| 174 | Задание 4 завершено. Начинается ввод задания 7 (20 квантов). Процессорное время выделяется заданиям 2, 3, 6. В очереди задания 5,8,9,10. | 5 | 1 | 3 |
| 194 | Ввод задания 7 завершен. Начинается ввод задания 10 (10 квантов). В очереди задания 5,8,9. | 4 (+3 -3) | 0 (+2 -2) | 4 |
| 216 | Задание 2 завершено. Начинается ввод задания 10. Процессорное время выделяется заданиям 3, 6,7. В очереди задания 5,8,9. | 4 | 0 | 3 |
| 226 | Завершен ввод задания 10. Процессорное время выделяется заданиям 3, 6, 7,10. В очереди задания 5,8,9. | 4 | 0 | 4 |
| 317 | Задание 6 завершено. Процессорное время выделяется заданиям 3, 7, 10. В очереди задания 5,8,9. | 4 (+3) | 0 (+2) | 3 |
| 345 | Задание 3 завершено. Начинается ввод задания 5 (5 квантов). Процессорное время выделяется заданиям 7, 10. В очереди задания 8,9. | 7 (+3 -9) | 2 (+4-1) | 2 |
| 350 | Завершен ввод задания 5. Процессорное время выделяется заданиям 5, 7, 10. В очереди задания 8, 9. | 1 | 5 | 3 |
| 426 | Задание 10 завершено. Начинается ввод задания 8 (15 квантов). Процессорное время выделяется заданиям 5, 7. В очереди задание 9. | 1 (+3-2) | 5 (+2-3) | 2 |
| 436 | Задание 5 завершено. Процессорное время выделяется заданиям 7. Начинается ввод задания 9 (5 квантов). | 2(+9-9) | 4(+1-1) | 1 |
| 441 | Ввод заданий 8,9 завершен. Процессорное время выделяется заданиям 7, 8, 9. | 2 | 4 | 3 |
| 473 | Задание 7 завершено. Процессорное время выделяется заданиям 8, 9. | 2(+3) | 4(+4) | 2 |
| 492 | Задание 8 завершено. Процессорное время выделяется заданию 9. | 5(+2) | 8(+3) | 1 |
| 502 | Задание 9 завершено. | 7(+9) | 11(+1) | 0 |
| … | Все задания завершены. | 16 | 12 | 0 |

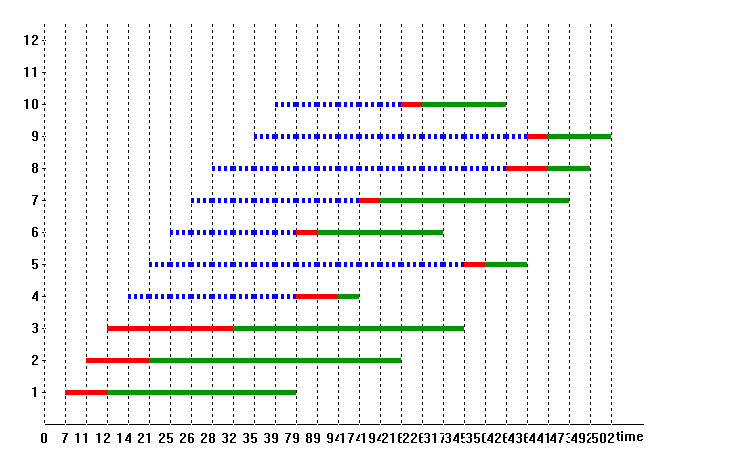
Временная диаграмма 1, дисциплина FIFO

Таблица 3. Характеристика выполнения заданий по ДО FIFO.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 7 | 11 | 12 | 14 | 21 | 25 | 26 | 28 | 35 | 39 |
|  | 7 | 11 | 12 | 79 | 345 | 79 | 174 | 426 | 436 | 216 |
|  | 12 | 21 | 32 | 94 | 350 | 89 | 194 | 441 | 441 | 226 |
|  | 79 | 216 | 345 | 174 | 436 | 317 | 473 | 492 | 502 | 426 |
|  | 72 | 205 | 333 | 95 | 91 | 238 | 299 | 66 | 66 | 210 |
|  | 1,00 | 1,00 | 1,00 | 1,52 | 4,79 | 1,25 | 1,62 | 5,05 | 5,65 | 1,89 |
| Вывод по ДО FIFO:  Максимальный коэффициент мультипрограммирования равен 4. Он достигается на интервалах 94-174, 216-226. Средневзвешенное время обращения Wср = 2,48. | | | | | | | | | | |
|
|
|
|
|  | | | | | | | | | | |

## Временная диаграмма при использовании ДО SJF

Таблица 4. Временная диаграмма мультипрограммной работы ЭВМ (ДО SJF)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | События | ОП  ( ΔОП) | ВУ  ( ΔВУ) | Коэф. Мульт. |
| 0 | Заданий нет. Простой системы | 16 | 12 | 0 |
| 7 | Поступило задание 1. Свободных ресурсов хватает. Начинается ввод задания 1 (5 квантов). | 16(-9) | 12(-1) | 0 |
| 11 | Поступило задание 2. Свободных ресурсов хватает. Начинается ввод задания 2 (10 квантов). Продолжается ввод задания 1. | 7(-3) | 11(-2) | 0 |
| 12 | Поступило задание 3. Свободных ресурсов хватает. Начинается ввод задания 3 (20 квантов). Завершен ввод задания 1. Продолжается ввод задания 2. Процессорное время выделяется заданию 1 | 4(-3) | 9(-4) | 1 |
| 14 | Поступило задание 4. Свободных ресурсов не хватает, задание 4 помещается в очередь. Процессорное время выделяется заданию 1. Продолжается ввод заданий 2,3. | 1 | 5 | 1 |
| 21 | Поступило задание 5. Свободных ресурсов не хватает, задание 5 помещается в очередь. Завершен ввод задания 2. Процессорное время выделяется заданиям 1,2. В очереди задания 4,5. Продолжается ввод задания 3. | 1 | 5 | 2 |
| 25 | Поступило задание 6. Свободных ресурсов не хватает, задание 6 помещается в очередь. Процессорное время выделяется заданиям 1, 2. В очереди задания 4,5,6. Продолжается ввод задания 3. | 1 | 5 | 2 |
| 26 | Поступило задание 7. Свободных ресурсов не хватает, задание 7 помещается в очередь. Продолжается ввод задания 3. Процессорное время выделяется заданиям 1, 2. В очереди задания 4,5,6,7. | 1 | 5 | 2 |
| 28 | Поступило задание 8. Свободных ресурсов не хватает, задание 8 помещается в очередь. Продолжается ввод задания 3. Процессорное время выделяется заданиям 1, 2. В очереди задания 4,5,6,7,8. | 1 | 5 | 2 |
| 32 | Завершен ввод задания 3. Процессорное время выделяется заданиям 1, 2, 3. В очереди задания 4,5,6,7,8. | 1 | 5 | 3 |
| 35 | Поступило задание 9. Свободных ресурсов не хватает, задание 9 помещается в очередь. Процессорное время выделяется заданиям 1, 2, 3. В очереди задания 4,5,6,7,8,9. | 1 | 5 | 3 |
| 39 | Поступило задание 10. Свободных ресурсов не хватает, задание 10 помещается в очередь. Процессорное время выделяется заданиям 1, 2, 3. В очереди задания 4,5,6,7,8,9,10. | 1 | 5 | 3 |
| 79 | Задание 1 завершено. Ресурсы освобождены. Начинается ввод заданий 4(15 квантов), 8 (15 квантов). Процессорное время выделяется заданиям 2, 3. В очереди задания 5,6,7,9,10. | 1(+9-2-2) | 5(+1-3-3) | 2 |
| 94 | Завершен ввод заданий 4,8. Процессорное время выделяется заданиям 2, 3, 4, 8. | 6 | 0 | 4 |
| 174 | Завершены задания 4,8. Начинается ввод задания 5. | 6(+2+2-9) | 0(+3+3-1) | 2 |
| 179 | Завершен ввод задания 5. | 1 | 5 | 3 |
| 205 | Задача 2 завершена. Начинается ввод задания 6(10 квантов). | 1(+3-3) | 5(+2-2) | 2 |
| 215 | Завершен ввод задания 6. | 1 | 5 | 3 |
| 264 | Задача 5 завершена. Начинается ввод задания 9. | 1(+9-9) | 5(+1-1) | 3 |
| 269 | Завершен ввод задания 9. | 1 | 5 | 4 |
| 304 | Задача 3 завершена. Начинается ввод задания 10. | 1(+3-3) | 5(+4-2) | 2 |
| 314 | Завершен ввод задания 10. | 1 | 7 | 3 |
| 354 | Задача 9 завершена. Начинается ввод задания 7. | 1(+9-3) | 7(+1-4) | 2 |
| 374 | Завершен ввод задания 7. | 7 | 4 | 3 |
| 377 | Задача 6 завершена. | 7(+3) | 4(+2) | 2 |
| 448 | Задача 10 завершена. | 10(+3) | 6(+2) | 1 |
| 502 | Задача 7 завершена. | 13(+3) | 8(+4) | 0 |
| … | Все задачи завершены. | 16 | 12 | 0 |

Временная диаграмма 2, дисциплина SJF см. приложение №1

Таблица 5. Характеристика выполнения заданий по ДО SJF.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 7 | 11 | 12 | 14 | 21 | 25 | 26 | 28 | 35 | 39 |
|  | 7 | 11 | 12 | 79 | 174 | 205 | 354 | 79 | 264 | 304 |
|  | 12 | 21 | 32 | 94 | 179 | 215 | 374 | 94 | 269 | 314 |
|  | 79 | 205 | 304 | 174 | 264 | 377 | 502 | 174 | 354 | 448 |
|  | 72 | 194 | 295 | 95 | 90 | 172 | 148 | 95 | 90 | 144 |
|  | 1,00 | 1,00 | 1,00 | 2,36 | 1,66 | 2,46 | 2,41 | 2,26 | 2,33 | 2,67 |

Вывод по SJF:

Максимальный коэффициент мультипрограммирования равен 4. Он был достигнут на участках 94-174, 269-304. Средневзвешенное время обращения Wср =1,92.

## Сравнительный анализ двух диаграмм

1. Максимальный коэффициент у обоих алгоритмов равен 4.
2. Дисциплина обслуживания **SJF** обладает меньшим значением средневзвешенного времени обращения (1,92) по сравнению с **FIFO**(2,48). Следовательно, для такой последовательности заданий выгоднее использовать ДО **SJF**. Это объясняется тем, что заданий с малой трудоемкостью в системе больше, чем продолжительных.
3. Обе дисциплины обеспечили одинаковое время работы ЭВМ

Таким образом, для данного набора задач предпочтительнее является ДО **SJF**, т.к. она имеет меньшее значение средневзвешенного времени обращения.

# Раздел 2

## Задание

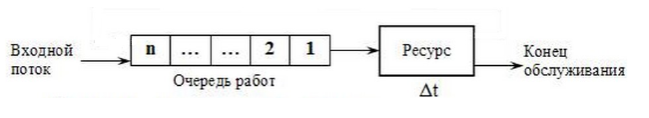
Разработать структуру функционирования диспетчера работ в вычислительной системе, заданной в разделе 1. Квант времени, выделяемый каждой работе, выбирается исходя из конкретной ситуации: число работ, параллельно занимающих процессор, интервалы времени с коэффициентом многозадачности, дисциплины обслуживания.

Диспетчер использует метод разделения времени в сочетании с приоритетами. ДО – следующие:

1) FIFO (БП)

2) Обслуживание с динамическим приоритетом (зависимость от времени обслуживания )

## Бесприоритетные ДО (FIFO)



FIFO - first in first out. Основная идея состоит в том, чтобы все задачи обслуживались в порядке их появления.

Данный алгоритм основан на квантовании, смена активного процесса происходит, если:

• Процесс завершился и покинул систему;

• Произошла ошибка;

• Процесс перешёл в состояние ОЖИДАНИЕ;

• Исчерпан квант процессорного времени, отведенный данному процессу.

После того как ранее заблокированная задача переходит в состояние готовности, она поступает в очередь вновь готовых задач, при этом из этих двух очередей сначала выбирается задачи ранее заблокированные. Тем самым поддерживается очередность исполнения задачи.

*Достоинства ДО FIFO:*

* простота диспетчеризации
* равноправность заявок

## Приоритетные ДО (P = f(

 Приоритет изменяется в период пребывания в системе в процессе обслуживания**.** Если высокоприоритетная заявка выполняется длительное время, необходимо понизить приоритет.

*Достоинства системы с динамическим приоритетом:*

* возможность настраиваться при изменении характеристик входного потока на эффективное обслуживание;
* дисциплина основана на реальных динамических характеристиках заявок.

Таблица 6. Матрица трудоемкостей.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Трудоемк. | 30 | 60 | 90 | 20 | 30 | 60 | 90 | 20 | 30 | 60 |
| Приоритет | 3 | 5 | 8 | 3 | 3 | 5 | 8 | 3 | 3 | 5 |
| t поступ. | 12 | 21 | 32 | 94 | 350 | 89 | 194 | 441 | 441 | 226 |

## Программная реализация диспетчера

public class Main {  
 static int *time* = 0; // Текущее время симуляции  
 private static int *KV* = 20; // Квант времени, который получают задачи

private static int *TSV* = 10; // Время работы диспетчераstatic Vector<Task> *vector* = new Vector<>(); // Массив для хранения задач  
 static int *maxPrtIndex*; // Переменная, указывающая на индекс задачи к выполнению в массиве vector (предназначена только для динамических приоритетов)  
 static boolean *sayAboutWait* = false;

Для моделирования задачи был создан класс Task. Каждая задача имеет следующие характеристики:

* Время поступления
* Трудоемкость
* Приоритет

public static class Task {  
 int tPrih;  
 int trudoemk;  
 boolean isCome;  
 int taskNum;  
 int priority;  
 boolean doEarly;  
 int waitTimes;

// Конструктор класса Task  
 Task(int tPrih, int trudoemk, int num, int priority) {  
 this.tPrih = tPrih;  
 this.trudoemk = trudoemk;  
 isCome = false;  
 taskNum = num;  
 this.priority = priority;  
 doEarly = false;  
 waitTimes =0;  
 }  
 }  
 // Функция, симулирующая приход задач. Если время прихода задачи соответствует времени симуляции, на выходе (в консоли) получаем сообщение о том, что задача пришла, а так же в массиве задач помечаем ее как пришедшую задачу

public static void checkTask()  
 {  
 for (int i = 0; i < *vector*.size(); i++) {  
 if (*time* > *vector*.get(i).tPrih && *vector*.get(i).isCome == false) {  
 System.*out*.println(*vector*.get(i).tPrih + " Постуила задача " + *vector*.get(i).taskNum);  
 *vector*.get(i).isCome = true;  
 }  
 }  
 }  
  
 //Подбор задачи к выполнению(для динамического приоритета)public static void getMaxPriority()  
 {  
 // Поиск задачи с наибольшим приоритетом: в массиве задач находим задачу с наибольшим приоритетом и сохраняем ее индекс в переменную maxPrtIndex.for (int i = 0; i < *vector*.size(); i++) {  
 if (*vector*.get(i).priority > *vector*.get(*maxPrtIndex*).priority && *vector*.get(i).isCome == true)  
 *maxPrtIndex* = i;  
 }  
 // Если среди задач есть более одной задачи с максимальным приритетом, сперва выполняется та задача, что выполнялась ранее. for (int i = 0; i < *vector*.size(); i++)  
 {  
 if(*vector*.get(i).priority == *vector*.get(*maxPrtIndex*).priority && *vector*.get(*maxPrtIndex*).doEarly == false && *vector*.get(i).doEarly == true) {  
 *maxPrtIndex* = i;  
 System.*out*.println("Задача " + *vector*.get(*maxPrtIndex*).taskNum + " уже выполнялась, поэтому выполняем ее ранее"); // В консоль выводится соответствующая информация  
 }  
 }  
 // Из двух задач с одинаковым приоритетом вперед будет выполнена та, что дольше ожидала выполненияfor (int i = 0; i < *vector*.size(); i++)  
 {  
 if(*vector*.get(i).priority == *vector*.get(*maxPrtIndex*).priority && *vector*.get(i).waitTimes > *vector*.get(*maxPrtIndex*).waitTimes && *vector*.get(i).doEarly == true)  
 {  
 *sayAboutWait* = true;  
 *maxPrtIndex* = i;  
 }  
 }

// Задачи, ожидающие, но не получившие свой квант времени накапливают время ожидания  
 for (int i = 0; i < *vector*.size(); i++)  
 if(*vector*.get(i).isCome == true) {  
 *vector*.get(i).waitTimes++;  
 *vector*.get(i).doEarly = false;  
 }  
 }  
  
 public static void main(String[] args) {  
 *// Задачи сформированы согласно таблице 6.*

*// При инициализации программ в конструктор класса Task передаем время прихода задачи, трудоемксть, номер и приоритет.*

Task task1 = new Task(12, 30, 1, 3);  
 Task task2 = new Task(21, 60, 2, 5);  
 Task task3 = new Task(32, 90, 3, 8);  
 Task task4 = new Task(94, 20, 4, 3);  
 Task task5 = new Task(350, 30, 5, 3);  
 Task task6 = new Task(89, 60, 6, 5);  
 Task task7 = new Task(194, 90, 7, 8);  
 Task task8 = new Task(441, 20, 8, 3);  
 Task task9 = new Task(441, 30, 9, 3);  
 Task task10 = new Task(226, 60, 10, 5);

// Все задачи добавляем в массив задач

*vector*.add(task1);  
 *vector*.add(task2);  
 *vector*.add(task3);  
 *vector*.add(task4);  
 *vector*.add(task5);  
 *vector*.add(task6);  
 *vector*.add(task7);  
 *vector*.add(task8);  
 *vector*.add(task9);  
 *vector*.add(task10);

**Диспетчеризация ДО FIFO**

// Имеется цикл, который будет выполняться, пока массив задач не станет пустым, т.е. все задачи не выполнятся.while (!*vector*.isEmpty()) {  
 *checkTask*(); // Функция, описанная выше  
 int next = 0;

// Из всех пришедших задач выбирается та, что пришла раньше всех.  
 for(int i = 0; i < *vector*.size(); i++)  
 {  
 if(*vector*.get(i).isCome == true && *vector*.get(i).tPrih < *vector*.get(next).tPrih)  
 next = i;  
 }  
 if (*vector*.get(next).isCome == true)  
 {

// Выбранной задаче выделяется квант времени, равный KV, а также KV прибавляем к времени симуляции  
 System.*out*.println((*time*) + " Задача " + *vector*.get(next).taskNum + " запущена");  
 if (*vector*.get(next).trudoemk > *KV*) {  
 *vector*.get(next).trudoemk -= *KV*;  
 *time* += *KV*;  
 *checkTask*();

// В консоль выводится соответствующая информация  
 System.*out*.println(*time* + " Истек квант времени задачи " + *vector*.get(next).taskNum);   
 } else

// Если задача закончена, удаляем ее из массива задач, причем если задача закончилась быстрее, чем закончится квант времени, который ей выделен, к времени симуляции прибавляем не квант времени, выделяемый задаче, а время, которое оставалось задаче до конца выполнения.   
 {  
 *time* += *vector*.get(next).trudoemk;

// В консоль выводится соответствующая информация  
 System.*out*.println(*time* + " задача " + *vector*.get(next).taskNum + " завершена");  
 *vector*.remove(next);  
 }  
 System.*out*.println(*time* + " Супервизор запущен");  
 }

// К времени симуляции прибавляем время работы диспетчера, равное TSV (симуляция работы диспетчера)  
 *time* += *TSV*;  
 }

**Диспетчеризация ПДО (зависимость от времени обслуживания)**

// Аналогично предыдущей ДО, имеется цикл, который будет выполняться, пока массив задач не станет пустым, т.е. все задачи не выполнятся.  
 while(!*vector*.isEmpty()){  
 *maxPrtIndex* = 0;

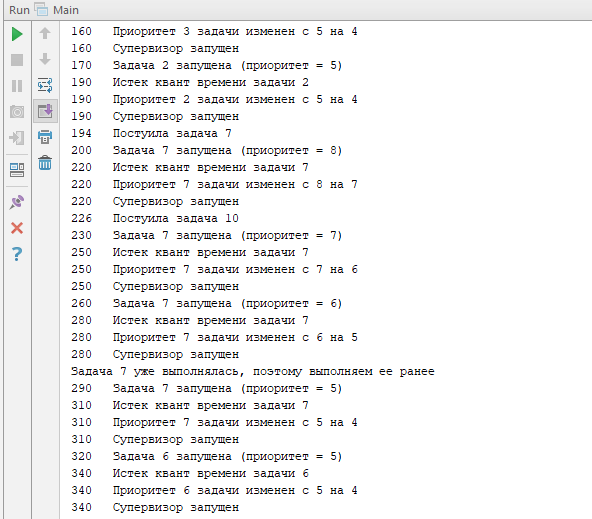
// Оповещаем о приходе задач  
 *checkTask*();

// По критерию дисциплины обслуживания подбираем задачу, которой будет выделен квант времени  
 *getMaxPriority*();  
 if (*vector*.get(*maxPrtIndex*).isCome == true)  
 {  
 if(*sayAboutWait*)  
 {  
 System.*out*.println(*time* + " Задача " + *vector*.get(*maxPrtIndex*).taskNum + " будет запущена, т.к. имеет большой приоритет и ее время ожидания наибольшее ");  
 *sayAboutWait* =false;  
 }  
  
 System.*out*.println((*time*) + " Задача " + *vector*.get(*maxPrtIndex*).taskNum + " запущена (приоритет = " + *vector*.get(*maxPrtIndex*).priority + ")");  
 if (*vector*.get(*maxPrtIndex*).trudoemk > *KV*)  
 {

//Аналогично предыдущей ДО, выбранной задаче выделяется квант времени, равный KV, а также KV прибавляем к времени симуляции  
 *vector*.get(*maxPrtIndex*).doEarly = true;  
 *vector*.get(*maxPrtIndex*).trudoemk -= *KV*;  
 *vector*.get(*maxPrtIndex*).waitTimes = 0;  
 *time* += *KV*;  
 *checkTask*();  
 System.*out*.println(*time* + " Истек квант времени задачи " + *vector*.get(*maxPrtIndex*).taskNum);

// Каждый раз, когда задаче выделяется квант времени, уменьшаем ее приоритет  
 *vector*.get(*maxPrtIndex*).priority--;  
 System.*out*.println(*time* + " Приоритет " + *vector*.get(*maxPrtIndex*).taskNum + " задачи изменен с " + (*vector*.get(*maxPrtIndex*).priority + 1) + " на " + *vector*.get(*maxPrtIndex*).priority);  
 *checkTask*();  
 } else  
 {  
 *time* += *vector*.get(*maxPrtIndex*).trudoemk;  
 System.*out*.println(*time* + " задача " + *vector*.get(*maxPrtIndex*).taskNum + " завершена");  
 *vector*.remove(*maxPrtIndex*);  
 }  
 System.*out*.println(*time* + " Супервизор запущен");  
 }  
 *time* += *TSV*;  
 }  
 }  
 }

В качестве выходных данных в консоль записывается описание прерываний в формате *время + событие.*



Временные диаграммы диспетчеризации ДО FIFO и ПДО (P = f(, а также временная диаграмма измененя приоритетов представлены в приложениях 2,3,4.

# Заключение

Выполнение работы было разделено на два раздела.

В ходе выполнения первого раздела были получены диаграммы мультипрограммной работы для двух заданных алгоритмов (FIFO и SJF). Одна построена вручную, вторая с использованием программы MultiVis. Так же для каждого алгоритма построены таблицы трассировки. На диаграммах и таблицах выделены события (моменты поступления заданий, моменты назначения на выполнение, моменты начала счета, моменты завершения) и периоды между событиями. Для каждого периода указано процессорное время на задание, доступную память, доступную дисковую память, коэффицент мультипрограммирования. По результатам этого раздела было проведено сравнение эффективности двух дисциплин для заданных начальных данных.

Во втором разделе данной разработана программа, имитирующая работу диспетчера задач для двух дисциплин обслуживания FIFO и P=f(tобс.). По результатам работы программы построены диаграммы диспетчеризации для двух указанных дисциплин на заданных данных. Так же построена диаграмма изменения приоритетов для дисциплины P=f(tобс)

# Список литературы.

1)​ Л.А. Коршикова. Операционные системы как системы управления вычислительными ресурсами: Учеб. пособие. – Новосибирск: Изд-во НГТУ, 2001. – 63 с.

2) http://ermak.cs.nstu.ru/~mos/index\_7.htm