**Методические указания к лабораторным работам по дисциплине**

**"АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ РЕАЛЬНОГО ВРЕМЕНИ"**

**ОБЩАЯ ХАРАКТЕРИСТИКА ЗАДАНИЙ**

Все задания, представленные ниже, представляют собой задачи на организацию доступа к общему (разделяемому, критическому) ресурсу некоторого числа параллельных процессов. Для практической реализации заданий следует использовать библиотечный модуль MultiObj, являющийся ядром управления параллельными процессами в среде MS DOS.

Для управления доступом к общему ресурсу следует создать объект - монитор. Монитор состоит из множества переменных состояния и множества процедур, которые используют эти переменные. Некоторые из этих процедур, называемые внешними, доступны пользователю и называются точками входа в монитор. Процессы, которые используют монитор для синхронизации, не имеют прямого доступа к переменным состояния. Они могут воспользоваться монитором только путем вызова внешних процедур. Процедуры монитора включают операции, которые позволяют блокировать и активизировать процессы в соответствие со спецификацией задачи. Условия установки и снятия блокировки записываются в виде функций переменных состояния и механизм работы монитора гарантирует, что все преобразования этих переменных происходят при взаимном исключении.

Монитор имеет следующее общее описание (в терминах объектно-ориентированного языка Pascal):

**Type**  
 TMonitor = **Object**  
 *{данные - переменные состояния}*  
 **Constructor** Init(...);  
 **Destructor** Done; **Virtual**;  
 **Procedure** Enter;  
 **Procedure** Exit;  
 **End** *{TMonitor}*.

В качестве данных выступают параметры спецификации задачи и очереди ожидания процессов.

Метод TMonitor.Enter, анализируя данные, выполняет следующие действия:

- проверяет условия входа в критический участок; при выполнении этих условий процесс, вызвавший метод, продолжает выполняться, войдя в критический участок; при невыполнении - процесс блокируется в очереди монитора;

- устанавливает необходимые значения переменных состояния при блокировании процесса и при продолжении выполнения в критическом участке.

Метод TMonitor.Exit выполняет следующие действия:

- устанавливает необходимые значения переменных состояния при выходе из критического участка;

- проверяет условия возможной активизации ждущих процессов и активизирует их при выполнении этих условий.

Структура процедуры, реализующей процесс, осуществляющий доступ к критическому ресурсу с помощью монитора, будет выглядеть следующим образом:

**Procedure** Process;  
**Begin**  
 *{действия процесса до входа в критический участок}*  
 Monitor.Enter;  
 *{действия процесса в критическом участке}*  
 Monitor.Exit;  
 *{действия процесса после выхода из критического участка}*  
**End** *{Process}*;

здесь Monitor - это переменная типа TMonitor, являющаяся экземпляром указанного объекта.

Количество различающихся методов входа в критический участок и методов выхода из него определяется количеством разновидностей процессов в каждой прикладной задаче.

В каждой из перечисленных ниже задач требуется создать свой монитор и запрограммировать его, используя ядро MultiObj. Методика решения каждой задачи, приведенная в разделе "Методика

решения", ориентирована на использование ядра MultiObj. Основой решения каждой задачи является формирование и формализация условий входа в критический участок и условий активизации ждущих

процессов при выходе из критического участка.

Методика решения задачи 1 рассмотрена более подробно, чем остальных задач. Методика решения каждой задачи предлагает один выриант решения, что конечно же, не исключает возможности решения задачи с помощью других алгоритмов. Для тех, кто затрудняется создать свой монитор, рекомендуется запрограммировать условия, предлагаемые в методике решения каждого задания.

Предполагается, что для решения задач группа разбивается на бригады по 2 - 3 человека и программирует задачи по типу лабораторных работ. Практическое решение каждой задачи

оформляется в виде отчета (одного на бригаду). Отчет должен содержать:

- спецификацию задачи;

- обоснование условий входа в критический участок и активизации ждущих процессов при выходе из него для каждой разновидности процессов;

- текст программы с комментариями.

Требования к программам, реализующим задания:

- программа должна выглядеть как законченный продукт – иметь меню, иметь Помощь, описывающую программу и правила работы с ней, и функционирующую хотя бы вне мультизадачного режима, иметь средства, отображающие на экране действия и состояния всех

процессов.

К тексту задания прилагается текстовый файл мультизадачного объектноориентированного ядра MultiObj.Pas, с помощью которого необходимо реализовать программы заданий. Объекты ядра допустимо перекрывать при необходимости.

Прилагается также простейший вариант реализации задачи 1 в качестве примера.

**Задача 1. МОДЕЛЬ ЖЕЛЕЗНОДОРОЖНОГО ПЕРЕГОНА**

Железная дорога, соединяющая два города А и В, включает участок, на котором имеется только единственный путь, см. схему.

Движение поездов на единственном пути подчиняется следующим ограничениям:

- на свободный единственный путь может войти поезд любого направления;

- пока на единственном пути находится поезд некоторого направления, на него не может войти поезд другого направления, но может войти поезд того же направления.

--->--------¬ ------>----

v ^

А +----->------<------+ B

v ^

---<--------- L-----<----

Требуется запрограммировать задачу, написав монитор, для двух вариантов условий:

1) нет ограничений на количество поездов одного направления, находящихся на единственном пути;

2) ограничить количество поездов одного направления, находящихся на единственном направлении значением N и решить проблему "бесконечного ожидания", свойственную первому варианту, когда поезда одного направления, захватив единственный путь, не позволяют поездам другого направления пройти по этому пути. При этом необходимо реализовать рациональный порядок выделения единственного пути поездам обоих направлений.

**Методика решения**

В данной задаче "единственный путь" является общим ресурсом для поездов, которые должны моделироваться процессами. Регламентация доступа процессов к этому общему ресурсу осуществляется средством "монитор".

Поскольку в задаче имеется два вида процессов - поезда, движущиеся слева направо, и поезда, движущиеся справа налево, то методов входа в критический участок и выходов из него будет в мониторе по два, на каждый вид процесса, то есть внутри объекта TMonitor должны быть объявлены следующие методы:

Procedure Enter\_L\_R; {вход поезда, движущегося слева направо, в критический участок}

Procedure Enter\_R\_L; {вход поезда, движущегося справа налево, в критический участок}

Procedure Exit\_L\_R; {выход поезда, движущегося слева направо, из критического участка}

Procedure Exit\_R\_L; {выход поезда, движущегося справа налево, из критического участка}.

Методы входа проверяют условия входа в критический участок. При выполнении этих условий процесс продолжает выполняться, входя в критический участок, а при невыполнении - процесс приостанавливается в очереди. При этом в обоих случаях устанавливаются необходимые данные, например, увеличивается количество движущихся в критическом участке поездов данного направления или увеличивается количество ждущих входа в критический участок поездов данного направления.

Методы выхода проверяют условия возможной активизации блокированных процессов, активизируют их при выполнении этих условий и устанавливаю необходимые данные, например, уменьшают количество движущихся в критическом участке поездов данного направления и уменьшают количество ждущих поездов.

Условия блокировки и активизации процессов-поездов позволяют определить те данные, которые необходимо анализировать в мониторе, а именно,

- количество поездов, движущихся по единственному пути слева направо, Nmlr;

- количество поездов, движущееся по единственному пути справа налево, Nmrl;

- количество поездов, ждущих входа на единственный путь слева направо, Nwlr;

- количество поездов, ждущих входа на единственный путь справа налево, Nwrl.

Кроме того, при ожидании выполнения условий входа на единственный путь процесс должен находиться в специально созданной для этого очереди. Поэтому в качестве данных в мониторе должны существовать следующие переменные:

R\_L\_List - очередь, в которой ждут освобождения единственного пути поезда, движущиеся справа налево;

L\_R\_List - очередь, в которой ждут освобождения единственного пути поезда, движущиеся слева направо.

C учетом приведенных выше описаний данных и методов объекта "Монитор" можно представить следующую структуру программы, реализующей задание:

Program Lab1;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

TMonitor = Object

Nmlr,

Nmrl,

Nwlr,

Nwrl : Word;

L\_R\_List,

R\_L\_List : TList;

Constructor Init;

Destructor Done; Virtual;

Procedure Enter\_L\_R;

Procedure Enter\_R\_L;

Procedure Exit\_L\_R;

Procedure Exit\_R\_L;

End {TMonitor};

{--Реализация методов монитора возлагается на учащегося--}

Var

Monitor : PMonitor;

Procedure L\_R\_Train;

Begin

{движение до входа в критический участок}

Monitor^.Enter\_L\_R;

{движение в критическом участке}

Monitor^.Exit\_L\_R;

{движение после выхода из критического участка}

{самоуничтожение}

End {L\_R\_Train};

Procedure R\_L\_Train;

Begin

{движение до входа в критический участок}

Monitor^.Enter\_R\_L;

{движение в критическом участке}

Monitor^.Exit\_R\_L;

{движение после выхода из критического участка}

{самоуничтожение}

End {R\_L\_Train};

Procedure KeyManager;

Begin

While True Do Begin

If Клавиша нажата Then Begin

Чтение клавиши;

Case Клавиша Of

'Esc' : Остановить работу ядра;

'L','l' : Создать процесс из процедуры L\_R\_Train;

'R','r' : Создать процесс из процедуры R\_L\_Train;

Else

End {Case};

End {If};

End {While};

- 9 -

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры KeyManager;

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab1}.

Наиболее важным этапом решения задачи является определение структуры методов TMonitor.Enter\_L\_R, TMonitor.Enter\_R\_L, TMonitor.Exit\_L\_R, а также условий входа в критический участок и выхода из него для процессов всех типов. Ниже предлагаются варианты данных методов для случая, когда нет ограничений на количество поездов одного направления, находящихся на

единственном пути.

Структура метода входа в критический участок имеет следующий вид:

Запрет прерываний;

If На единственном пути есть поезда противополжного направления

Then Begin

Увеличение числа процессов, ждущих в очереди, на единицу;

Включение процесса в очередь, ждущих входа на единственный

путь;

Передача управления процессу, первому в очереди готовых

процессов;

Уменьшение числа процессов, ждущих в очереди, на единицу;

End {If};

Увеличение числа процессов, находящихся на единственном пути, на единицу;

Разрешение прерываний;

Структура метода выхода из критического участка имеет

следующий вид:

Запрет прерываний;

Уменьшение числа процессов, находящихся на единственном пути, на

единицу;

If Единственный путь свободен Then Begin

While Очередь не пуста Do Begin

Исключить первый процесс из очереди процессов, ждущих

входа на единственный путь с противоположного

направления;

Включить этот процесс в очередь готовых процессов;

End {While};

End {If};

Разрешение прерываний;

**Замечания**

1) Методы монитора должны выполняться в режиме взаимного исключения, поэтому на входе любого метода выполняется запрет прерываний, а на выходе - разрешение прерываний.

2) Методы входа на критический участок и выхода из него для поездов обоих направлений идентичны по структуре и различаются лишь параметрами.

3) Представлена именно структура методов, их полная реализация возлагается на учащегося.

4) Предполагается, что учащийся освоил курс лабораторных работ, связанных с построением многозадачного ядра управления процессами, и знаком с реализацией механизмов исключения процессов из очередей, включения процессов в очереди, передачи управления и т.п.

Представленные алгоритмы входа в критический участок и выхода из него обладают тем недостатком, что приводят к бесконечному ожиданию выхода поездов одного направления на

единственный путь, если последний захвачен поездами другого направления.

Устранить данный недостаток можно различными способами, примеры которых перечислены ниже.

1) Заявки на прохождение единственного пути устанавливать в одну очередь и выпускать поезда на единственный путь согласно этой очереди. При этом, если следующим в очереди за выпускаемым находится поезд того же направления, то можно выпустить и его, вплоть до некоторого значения N. В последнем случае направление поезда следует писать в дескриптор процесса, его реализующего.

2) Сохранить две очереди в соответствие с направлениями движения поездов, но фиксировать момент времени постановки процесса в очередь и первым активизировать процесс, у которого момент постановки в очередь меньший не зависимо от очереди, в которой этот процесс находится. В этом случае время постановки в очередь следует писать в дескриптор процесса.

3) Не пропускать дополнительные поезда на единственный путь, а ставить их в очередь, если в очереди противоположного направления уже есть поезда, и не активизировать поезда текущего направления при выходе из критического участка, если есть поезда противоположного направления в очереди, а активизировать их, когда единственный путь окажется свободным.

В любом варианте не меняются структуры процедур монитора, реализующих вход в критический участок и выход из него, а меняется лишь вид условий If ... Then блокировки и активизации процессов.

Учащийся по согласованию с преподавателем выбирает для программирования вариант входа в критический участок и выхода из него или создает собственный алгоритм.

**Задача 2. МОДЕЛЬ ДОРОЖНОГО ПЕРЕКРЕСТКА**

Две автомобильные дороги В-Н и Л-П образуют перекресток, как показано схеме.

Автомобили могут двигаться через перекресток по дорогам В-Н и Л-П со следующими ограничениями:

- на свободный перекресток может выехать автомобиль с любого направления;

- если на территории перекрестка появился автомобиль направления В-Н или Н-В (Л-П или П-Л), то на ней не может появиться автомобиль направления Л-П или П-Л (В-Н или Н-В). Последний должен быть приостановлен и может возобновить свое движение только после освобождения перекрестка, автомобили же "своих" направлений могут въезжать в это время на перекресток.

В

¦ ^ ¦ ¦

¦ ¦ ¦ ¦

¦ ¦ ¦ ¦

===========- ¦ ¦ L==========

-------------+-+-----------> П

Л <------------+-+------------

===========¬ ¦ ¦ г==========

¦ ¦ ¦ ¦

¦ ¦ ¦ ¦

¦ ¦ v ¦

Н

Требуется запрограммировать задачу, написав монитор "Перекресток" для следующих условий:

- ограничить число автомобилей данного направления, допускаемых на перекресток, значением N;

- преодолеть проблему бесконечного ожидания, когда захватив перекресток, автомобили "своих" направлений не дают возможности выйти на перекресток автомобилям других направлений сколь угодно долго.

**Методика решения**

Методика решения включает в себя общую структуру программы и словесное описание условий входа в критический участок (перекресток) и условий выхода из него.

Общая структура программы выглядит следующим образом:

Program Lab2;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

TMonitor = Object

{данные - количество автомобилей, ждущих и

движущихся по перекрестку по каждому нарправлению}

{данные - очереди ожидания выхода на перекресток по

каждому направлению}

Сonstructor Init(...);

Destructor Done; Virtual;

Procedure Enter\_L\_R;

Procedure Enter\_R\_L;

Procedure Enter\_U\_D;

Procedure Enter\_D\_U;

Procedure Exit\_L\_R;

Procedure Exit\_R\_L;

Procedure Exit\_U\_D;

Procedure Exit\_D\_U;

End {TMonitor};

{--Реализация методов монитора возлагается на учащегося--}

Var

Monitor : PMonitor;

Procedure Move\_L\_R;

Begin

{движение до входа в критический участок}

Monitor^.Enter\_L\_R;

{движение в критическом участке}

Monitor^.Exit\_L\_R;

{движение после выхода из критического участка}

{самоуничтожение}

End {Move\_L\_R};

Procedure Move\_R\_L;

Begin

{движение до входа в критический участок}

Monitor^.Enter\_R\_L;

{движение в критическом участке}

Monitor^.Exit\_R\_L;

{движение после выхода из критического участка}

{самоуничтожение}

End {Move\_R\_L};

Procedure Move\_U\_D;

Begin

{движение до входа в критический участок}

Monitor^.Enter\_U\_D;

{движение в критическом участке}

Monitor^.Exit\_U\_D;

{движение после выхода из критического участка}

{самоуничтожение}

End {Move\_U\_D};

Procedure Move\_D\_U;

Begin

{движение до входа в критический участок}

Monitor^.Enter\_D\_U;

{движение в критическом участке}

Monitor^.Exit\_D\_U;

{движение после выхода из критического участка}

{самоуничтожение}

End {Move\_D\_U};

Procedure KeyManager;

Begin

While True Do Begin

If Клавиша нажата Then Begin

Чтение клавиши;

Case Клавиша Of

- 14 -

'Esc' : Остановить работу ядра;

'L','l' : Создать процесс из процедуры Move\_L\_R;

'R','r' : Создать процесс из процедуры Move\_R\_L;

'U','u' : Создать процесс из процедуры Move\_U\_D;

'D','d' : Создать процесс из процедуры Move\_D\_U;

Else

End {Case};

End {If};

End {While};

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры KeyManager;

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab2}.

Условие входа в критический участок на уровне словесного описания может выглядеть следующим образом:

If На критическом участке находятся автомобили пересекающих

направлений OR

Автомобилей своего направления больше, чем N, OR

Есть очередь из автомобилей своего направления OR

Есть очереди из автомобилей пересекающих направлений

Then Begin

Инкремент числа стоящих в очереди автомобилей;

Включение автомобиля в очередь;

Передача управления;

Декремент числа стоящих в очереди автомобилей;

End {If};

Инкремент числа находящихся на перекрестке автомобилей;

Условие выхода из критического участка на уровне словесного описания выглядит следующим образом:

Декремент числа находящихся на перекрестке автомобилей;

If Есть очереди из автомобилей пересекающих направлений

Then Begin

If Перекресток свободен от автомобилей непересекающих

направлений

Then Begin

Активизировать автомобили из очередей пересекающих

- 15 -

направлений, в количестве не более N для каждого;

End {If};

End Else Begin {нет очередй из автомобилей пересекающих

направлений}

Выпустить один автомобиль своего направления на перекресток,

если таковой есть в очереди;

End {If};

Учащемуся необходимо реализовать алгоритмы входа в критический участок и выхода из него, представленные выше в форме словесного описания, или реализовать собственные алгоритмы, обеспечивающие ограничение на количество автомобилей одного направления на перекрестке не более N и отсутствие бесконечного ожидания.

**Задача 3. МОДЕЛЬ ЧИТАТЕЛЕЙ И РЕДАКТОРОВ**

Имеется файл, управляемый процессами двух типов:

- читателями, которые могут просматривать файл без изменения его содержания;

- редакторами, которые могут изменять файл.

Требуется реализовать программу, которая обеспечивала бы следующую дисциплину доступа к файлу:

- в произвольный момент времени работать с файлом может только один редактор и ни одного читателя;

- одновременно работать с файлом могут несколько читателей и ни одного редактора;

- отсутствует бесконечное ожидание, когда редактор не может получить доступ к файлу из-за постоянного появления читателей, обращающихся к файлу и наоборот, когда читатели не могут получить доступ к ресурсу из-за постоянного появления запросов редакторов.

Задачу требуется решить для фиксированного количества редакторов и читателей, например, количество редакторов - 2, количество читателей - 3, при этом реально с файлом можно не работать, а достаточно обозначить факты входа в критический участок и выхода из него для каждого процесса.

**Методика решения**

Методика решения включает в себя общую структуру программы и словесное описание условий входа в критический участок и выхода из него для процесса-редактора и процесса-читателя.

Общая структура программы выглядит следующим образом:

**Program** Lab3;  
**Uses** MultiObj;  
**Type**  
 PMonitor = ^TMonitor;  
 TMonitor = **Object**  
 Nrf, *{число читателей, работающих с файлом}*  
 Nwf, *{число редакторов, работающих с файлом}*  
 Nrw, *{число читателей, ждущих допуска к файлу}*  
 Nww : Integer;*{число редакторов, ждущих допуска к файлу}*  
 R\_List, *{очередь читателей}*  
 W\_List : List;*{очередь редакторов}*  
 Сonstructor Init;  
 **Destructor** Done; **Virtual**;  
 **Procedure** Enter\_R;  
 **Procedure** Enter\_W;  
 **Procedure** Exit\_R;  
 **Procedure** Exit\_W;  
 **End** *{TMonitor}*;  
*{--Реализация методов монитора возлагается на учащегося--}*  
**Var**  
 Monitor : PMonitor;  
  
**Procedure** Reader\_1;  
**Begin**  
 **While** True **Do** **Begin**  
 *{Случайное время работы без ресурса}*  
 Monitor^.Enter\_R;  
 *{Случайное время работы с ресурсом}*  
 Monitor^.Exit\_R;  
 **End** *{While}*;  
**End** *{Reader\_1}*;  
  
**Procedure** Reader\_2;  
**Begin**  
 **While** True **Do** **Begin**  
 *{Случайное время работы без ресурса}*  
 Monitor^.Enter\_R;  
 *{Случайное время работы с ресурсом}*  
 Monitor^.Exit\_R;  
 **End** *{While}*;  
**End** *{Reader\_2}*;  
  
**Procedure** Reader\_3;  
**Begin**  
 **While** True **Do** **Begin**  
 *{Случайное время работы без ресурса}*  
 Monitor^.Enter\_R;  
 *{Случайное время работы с ресурсом}*  
 Monitor^.Exit\_R;  
 **End** *{While}*;  
**End** *{Reader\_3}*;  
*{Процедуры Reader\_1, Reader\_2 и Reader\_3 структурно одинаковы, но*  
*различаются, например, координатами вывода на экран своего статуса -*  
*"работает с файлом", "ждет доступа к файлу", "не требует доступа к*  
*файлу".}*  
**Procedure** Writer\_1;  
**Begin**  
 **While** True **Do** **Begin**  
 *{Случайное время работы без ресурса}*  
 Monitor^.Enter\_W;  
 *{Случайное время работы с ресурсом}*  
 Monitor^.Exit\_W;  
 **End** *{While}*;  
**End** *{Writer\_1}*;  
  
**Procedure** Writer\_2;  
**Begin**  
 **While** True **Do** **Begin**  
 *{Случайное время работы без ресурса}*  
 Monitor^.Enter\_W;  
 *{Случайное время работы с ресурсом}*  
 Monitor^.Exit\_W;  
 **End** *{While}*;  
**End** *{Writer\_2}*;  
*{Процедуры Writer\_1 и Writer\_2 структурно одинаковы, но*  
*различаются, например, координатами вывода на экран своего статуса -*  
*"работает с файлом", "ждет доступа к файлу", "не требует доступа к*  
*файлу".}*  
**Procedure** KeyManager;  
**Begin**  
 **While** True **Do** **Begin**  
 **If** Клавиша нажата **Then** **Begin**  
 Чтение клавиши;  
 **Case** Клавиша **Of**  
 'Esc' : Остановить работу ядра;  
 **Else**  
 **End** *{Case}*;  
 **End** *{If}*;  
 **End** *{While}*;  
**End** *{KeyManager}*;  
  
**Begin**  
 Monitor := New(PMonitor, Init);  
 Создать процесс из процедуры KeyManager;  
 Создать процесс из процедуры Reader\_1;  
 Создать процесс из процедуры Reader\_2;  
 Создать процесс из процедуры Reader\_3;  
 Создать процесс из процедуры Writer\_1;  
 Создать процесс из процедуры Writer\_2;  
 Начать работу ядра;  
 Dispose(Monitor, Done);  
**End** *{Lab3}*.

Все разнообразие алгоритмов доступа читателей и редакторов к файлу реализуется в методах монитора. Ниже представлено словесное описание условий входа в критический участок и выхода из него для читателей и редакторов, которые могут быть реализованы учащимися.

Условие входа в критический участок читателя может выглядеть следующим образом:

**If** Файл занят редактором **Or** Есть редакторы в очереди **Then** **Begin**  
 Инкремент числа читателей в очереди;  
 Постановка читателя в очередь;  
 Передача управления;  
 Декремент числа читателей в очереди;  
**End** *{If}*;  
Инкремент числа читателей, работающих с файлом;

Условие входа в критический участок редактора может выглядеть следующим образом:

**If** Файл занят редактором **Or** Файл занят читателями **Then** **Begin**  
 Инкремент числа редакторов в очереди;  
 Постановка редактора в очередь;  
 Передача управления;  
 Декремент числа редакторов в очереди;  
**End** *{If}*;  
Инкремент числа редакторов, работающих с файлом;

Условие выхода читателя из критического участка может выглядеть следующим образом:

Декремент числа читателей, работающих с файлом;  
**If** Нет читателей, работающих с файлом **And** Есть редактор в очереди **Then** **Begin**  
 Активизация редактора;  
**End** *{If}*;

Условие выхода редактора из критического участка может выглядеть следующим образом:

Декремент числа редакторов, работающих с файлом;  
**If** Есть читатели в очереди **Then** **Begin**  
 Активизация всех читателей из очереди;  
**End** **Else** **Begin** *{нет читателей, ждущих доступа к файлу}*  
 Активизация первого редактора из очереди,если таковая имеется;  
**End** *{If}*;

Устранение возможного бесконечного ожидания редактора, когда постоянно подходят читатели к критическому ресурсу, в данном случае устраняется следующим условием:

- читатель встает в очередь, если уже есть очередь из редакторов;

- последний из читателей, работающих в данный момент с ресурсом, активизирует редактора;

а устранение возможного бесконечного ожидания читателей устраняется тем, что выходящий из критического участка редактор в первую очередь активизирует читателей, ждущих ресурса, и если

таковых нет, то только тогда делается попытка активизировать редактора.

Таким образом обеспечивается достаточно справедливое выделение ресурса-файла процессам двух видов:

- после отработки редактора с файлом, последний предоставляется всем читателям, стоящим в очереди к моменту окончания работы с файлом редактора; а после отработки с файлом всех этих читателей, файл предоставляется одному редактору и т.д.

Вторым вариантом справедливого распределения ресурса является запись редакторов и читателей в одну очередь и предоставление ресурса в порядке очередности, но с учетом особенностей процессов:

- редактор активизируется один, а читатели активизируются все стоящие в очереди перед редактором.

Учащемуся рекомендуется реализовать на выбор один из предлагаемых алгоритмов или реализовать свой собственный.

**Задача 4. МОДЕЛЬ НАЗНАЧЕНИЯ ОДНОРОДНЫХ РЕСУРСОВ**

Имеется N единиц однородного ресурса и М процессов. Каждый процесс может запросить нужное ему количество ресурсов от 1 до N единиц. Если в свободных ресурсах нет запрашиваемого им количества, то процесс вынужден приостановиться и ждать их освобождения. Требуется запрограммировать задачу, написав монитор "Ресурс", реализующий выделение запрашиваемого количества ресурсов и свободный от бесконечного ожидания.

**Методика решения**

Если обозначить через n, количество единиц ресурса, запрашиваемое текущим процессом, и обозначить через Nсвоб количество свободных в данный момент ресурсов, то условие выделения ресурса процессу будет выглядеть следующим образом:

n <= Nсвоб.

Невыполнение данного условия должно приводить к блокированию процесса.

Общая структура программы выглядит следующим образом.

Program Lab4;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

TMonitor = Object

N,

Nсвоб : Integer;

RList : List;

Constructor Init(...);

- 21 -

Destructor Done;

Procedure Request;

Procedure Free;

End {TMonitor};

{--Реализация методов монитора возлагается на учащегося--}

Var

Monitor : PMonitor;

Procedure Process\_1;

Begin

While True Do Begin

{случайное времы работы без ресурса}

Monitor^.Request;

{случайное время работы с ресурсом}

Monitor^.Free;

End {While}:

End {Process\_1};

Procedure Process\_2;

Begin

While True Do Begin

{случайное времы работы без ресурса}

Monitor^.Request;

{случайное время работы с ресурсом}

Monitor^.Free;

End {While}:

End {Process\_2};

Procedure Process\_3;

Begin

While True Do Begin

{случайное времы работы без ресурса}

Monitor^.Request;

{случайное время работы с ресурсом}

Monitor^.Free;

End {While}:

End {Process\_3};

{Процедуры Process\_1, Process\_2 и Process\_3 структурно одинаковы,

но различаются, например, координатами вывода на экран своего

статуса - "не требует ресурса", "ждет данного количества

ресурса", "работает с данным количеством ресурса"}

Procedure KeyManager;

- 22 -

{Процедура KeyManager аналогична с задачей 3}

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры KeyManager;

Создать процесс из процедуры Process\_1;

Создать процесс из процедуры Process\_2;

Создать процесс из процедуры Process\_3;

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab4}.

Управление доступом к ресурсам осуществляется методами

Request и Free объекта TMonitor.

Исходя из условий задачи, структура метода Request может выглядеть следующим образом:

Определить случайное число n в диапазоне от 1 до N - количество

единиц ресурса, запрашиваемых текущим процессом;

Записать значение n в дескриптор процесса;

If n > Nсвоб ИЛИ Есть процессы, стоящие в очереди

Тhen Begin

Инкремент числа процессов, стоящих в очереди;

Включение процесса в очередь;

Передача управления;

Декремент числа процессов, стоящих в очереди;

End {If};

Ncвоб := Nсвоб - n;

Структура метода Free может выглядеть следующим образом:

Nсвоб := Nсвоб + n;

Temp := Nсвоб;

While Очередь не пуста Do Begin

If Для первого в очереди n <= Temp Then Begin

Процесс, первый в очереди, перевести в очередь готовых

процессов;

Temp := Temp - n;

End Else Begin

Break;

End {If};

End {While};

Возможность бесконечного ожидания процесса, ждущего большого количества ресурсов, когда все время появляются процессы, запрашивающие и получающие небольшое количество ресурсов, устраняется в данном решении следующим образом:

- процесс встает в очередь, если есть уже в очереди другие процессы;

- выделение ресурсов процессам при их активизации осуществляется исключительно, начиная с первого в очереди, если первому в очереди не хватает свободных ресурсов, то активизация

процессов не производится.

В данной задаче учащимся необходимо создать объект-наследник объекта Process мультизадачного ядра MultiObj, добавив в дескриптор поле, где будет храниться количество запрошенных

процессом ресурсов.

**Задача 5. МОДЕЛЬ ОБЕДАЮЩИХ ФИЛОСОФОВ**

Пять философов сидят за круглым обеденным столом. Между каждыми двумя философами есть одна вилка, которая может быть общей для них.

Каждый философ может либо думать, не требуя вилок, либо есть, используя две соседние вилки, расположенные по одну и по другую стороны от него. Время обеих фаз "думать" и "есть" - произвольная конечная величина. При отсутствии двух свободных вилок, необходимых философу для еды, последний переходит в состояние ожидания.

Требуется решить задачу, написав монитор "Вилки", реализующий выделение вилок философам и обеспечивающий возможность их перехода от состояния "думать" к состоянию "есть" без бесконечного ожидания.

**Методика решения**

Примем следующую нумерацию философов и вилок:

- будем нумеровать философов от 0 до 4;

- будем нумеровать вилки от 0 до 4;

- будем считать, что у философа с номером i слева лежит вилка с номером i, а справа - вилка с номером (i+1) mod 5.

Если философов интерпретировать как процессы, а вилки – как общие ресурсы, то общую структуру программы можно представить следующим образом.

Program Lab5;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

TMonitor = Object

V : Array[0..4] Of Boolean; {данные, описывающие

состояние вилок - сободна/занята}

PHList : List;

Constructor Init(...);

Destructor Done; Virtual;

Procedure Enter\_Eat;

Procedure Exit\_Eat;

End {TMonitor};

{--Реализация методов монитора возлагается на учащегося--}

Var

Monitor : PMonitor;

Procedure Phil\_1;

Begin

While True Do Begin

{случайное время в фазе "думать"}

Monitor^.Enter\_Eat;

{случайное время в фазе "есть"}

Monitor^.Exit\_Eat;

End {While};

End {Phil\_1};

Procedure Phil\_2;

Begin

While True Do Begin

{случайное время в фазе "думать"}

Monitor^.Enter\_Eat;

{случайное время в фазе "есть"}

Monitor^.Exit\_Eat;

End {While};

End {Phil\_2};

Procedure Phil\_3;

Begin

While True Do Begin

{случайное время в фазе "думать"}

- 25 -

Monitor^.Enter\_Eat;

{случайное время в фазе "есть"}

Monitor^.Exit\_Eat;

End {While};

End {Phil\_3};

Procedure Phil\_4;

Begin

While True Do Begin

{случайное время в фазе "думать"}

Monitor^.Enter\_Eat;

{случайное время в фазе "есть"}

Monitor^.Exit\_Eat;

End {While};

End {Phil\_4};

Procedure Phil\_5;

Begin

While True Do Begin

{случайное время в фазе "думать"}

Monitor^.Enter\_Eat;

{случайное время в фазе "есть"}

Monitor^.Exit\_Eat;

End {While};

End {Phil\_5};

{Процедуры Phil\_1 .. Phil\_5 структурно одинаковы, но различаются,

например, координатами вывода состояния философа на экран}

Procedure KeyManager;

{Процедура KeyManager аналогична с задачами 3 и 4}

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры KeyManager;

Создать процесс из процедуры Phil\_1;

Создать процесс из процедуры Phil\_2;

Создать процесс из процедуры Phil\_3;

Создать процесс из процедуры Phil\_4;

Создать процесс из процедуры Phil\_5;

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab5}.

Бесконечное ожидание процессов может возникнуть, если организовать монитор таким образом, что философы могут брать вилки по одной. Тогда, если каждый философ захватит, например, правую от себя вилку и перейдет в состояние ожидания левой вилки, то вся система перейдет в состояние тупика. Поэтому введем ограничение, состоящее в том, что i-й философ может перейти в состояние "есть" только при наличии двух свободных вилок с номерами i и (i+1)mod5. Если философ захотел "есть" и только одна из необходимых ему вилок свободна (или ни одной), то он переходит в состояние ожидания.

Структура метода монитора, реализующего вход в критический участок может тогда выглядеть следующим образом:

Определение номера i философа, соответствующего текущему

процессу;

If Вилка[i] или Вилка[(i+1)mod5] заняты Then Begin

Поставить процесс в очередь;

Передать управление;

End {If};

Вилка[i] := занята;

Вилка[(i+1)mod5] := занята;

При освобождении вилок философом с номером i монитор делает проверку, есть ли философы, связанные с данными вилками в очереди ( это философы с номерами (i+1)mod5 и (i+4)mod5 ), и свободны ли дополнительные для этих философов вилки ( это вилки с номерами (i+2)mod5 и (i+4)mod5 ). При положительном ответе данные философы активизируются.

Структура метода монитора, реализующего выход из критического участка может тогда выглядеть следующим образом:

Определение номера i философа, соответствующего текущему

процессу;

Вилка[i] := свободна;

Вилка[(i+1)mod5] := свободна;

If Процесс с номером (i+1)mod5 в очереди

И

Вилка[(i+2)mod5] свободна

Then Begin

Вилка[(i+1)mod5] := занята;

Вилка[(i+2)mod5] := занята;

Активизировать процесс с номером (i+1)mod5;

- 27 -

End {If};

If Процесс с номером (i+4)mod5 в очереди

И

Вилка[(i+4)mod5] свободна

Then Begin

Вилка[(i+4)mod5] := занята;

Вилка[i] := занята;

Активизировать процесс с номером (i+4)mod5;

End {If};

В данной задаче учащимся необходимо создать объект-наследник

объекта Process мультизадачного ядра MultiObj, добавив в

дескриптор поле, где будет храниться номер процесса-философа.

**Задача 6. МОДЕЛЬ КЛИЕНТ-СЕРВЕР**

Имеется N процессов-клиентов и один процесс-сервер. Процессы-клиенты в случайные моменты времени запрашивают обслуживание у процесса-сервера. Если процесс-сервер свободен, то

он выполняет обслуживание клиента и переходит в состояние ожидания запроса.

Если процесс-клиент запрашивает обслуживание, когда процесс-сервер занят, то клиент приостанавливает свое выполнение и ждет обслуживания в очереди.

Требуется запрограммировать задачу, написав монитор "Сервер", реализующий обслуживание клиентов в порядке очередности их запросов.

**Методика решения**

Ведем дополнительные условия:

- процесс-сервер запускается первым и ждет запросов на обслуживание;

- процесс-клиент блокируется не только при ожидании обслуживания, но и на время самого обслуживания.

Структура программы для описанных условий может выглядеть следующим образом.

Program Lab6;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

- 28 -

TMonitor = Object

Cl\_List, {очередь клиентов, ждущих обслуживания}

Cl\_s\_List, {очередь из одного клиента, обслуживаемого

в данный момент}

Sr\_List : List; {очередь, в которой сервер ждет

запросов на обслуживание}

Constructor Init;

Destructor Done;

Procedure Request; {запрос клиента на обслуживание}

Procedure Waiting; {ожидание сервером запросов}

End {TMonitor};

{--Методы монитора разрабатываются учащимися--}

Var

Monitor : PMonitor;

Procedure Client\_1;

Begin

While True Do Begin

{случайное время до запроса на обслуживания}

Monitor^.Request;

End {While};

End {Client\_1};

Procedure Client\_2;

Begin

While True Do Begin

{случайное время до запроса на обслуживания}

Monitor^.Request;

End {While};

End {Client\_2};

Procedure Client\_3;

Begin

While True Do Begin

{случайное время до запроса на обслуживания}

Monitor^.Request;

End {While};

End {Client\_3};

{Процессы-клиенты отличаются друг от друга только координатами

вывода на экран своих состояний - "работа вне обслуживания",

"ожидание обслуживания", "обслуживание"}

Procedure Server;

- 29 -

Begin

While True Do Begin

Monitor^.Waiting;

{случайное время обслуживания клиента}

End {While};

End {Server};

Procedure KeyManager;

{Процедура KeyManager аналогична с задачами 3, 4 и 5}

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры Server;

Создать процесс из процедуры KeyManager;

Создать процесс из процедуры Client\_1;

Создать процесс из процедуры Client\_2;

Создать процесс из процедуры Client\_3;

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab6}.

Структура метода TMonitor.Request, который выполняется

клиентом, может быть представлена следующим образом.

If Сервер свободен Then Begin

Активизация сервера;

End {If};

Встать в очередь ожидания обслуживания;

Передача управления;

Структура метода TMonitor.Waiting, который выполняется

сервером, может выглядеть следующим образом:

If Обслужен процесс Then Begin

Активизировать обслуженный процесс;

End {If};

If Нет клиентов, запрашивающих обслуживание Then Begin

Блокирование процесса-сервера;

Передача управления;

End {If};

Перевод процесса, первого в очереди ожидания обслуживания, в

очередь обслуживаемых процессов;

Учащемуся предстоит реализовать программу, выводя на экран

- 30 -

состояния всех процессов и время, оставшееся до перехода в другое

состояние. А также решить задачу с учетом нескольких

обслуживающих процессов.

**Задача 7. МОДЕЛЬ ФУНКЦИОНИРОВАНИЯ ЛИФТА**

В двухэтажном доме имеется лифт. Пассажиры, появляющиеся на первом этаже, желают подняться на второй этаж, а пассажиры, появляющиеся на втором этаже, желают спуститься на первый этаж. Пассажиры, появляющиеся во время движения лифта, должны ожидать его освобождения в очереди. Лифт могут занимать N пассажиров.

Требуется запрограммировать задачу, написав монитор "Лифт" и обеспечив справедливое выделения лифта пассажирам в зависимости от их появления в системе. Лифт следует рассматривать как процесс-сервер.

**Методика решения**

Методика решения включает в себя общую структуру программы и словесное описание условий блокировки и активизации процессов-пассажиров и процесса-лифта.

Структура программы выглядит следующим образом.

Program Lab7;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

TMonitor = Object

D\_U\_List, {Очередь, ждущих внизу}

U\_D\_List, {Очередь, ждущих вверху}

MoveList, {Очередь перевозимых}

LiftList : TList; {Очередь лифта}

Constructor Init(...);

Destructor Done; Virtual;

Procedure Request\_D\_U;

Procedure Request\_U\_D;

Procedure Waiting;

End {TMonitor};

{--Методы монитора разрабатываются учащимися--}

Var

Monitor : PMonitor;

- 31 -

Procedure Client\_D\_U;

Begin

{Движение на нижнем этаже}

Monitor^.Request\_D\_U;

{Движение на верхнем этаже}

{Самоуничтожение}

End {Client\_D\_U};

Procedure Client\_U\_D;

Begin

{Движение на верхнем этаже}

Monitor^.Request\_U\_D;

{Движение на нижнем этаже}

{Самоуничтожение}

End {Client\_U\_D};

Procedure Lift;

Begin

While True Do Begin

Monitor^.Waiting;

{Движение}

End {While};

End {Lift};

Procedure KeyManager;

Begin

While True Do Begin

If Клавиша нажата Then Begin

Чтение клавиши;

Case Клавиша Of

'Esc' : Остановить работу ядра;

'U','u' : Создать процесс из процедуры Client\_U\_D;

'D','d' : Создать процесс из процедуры Client\_D\_U;

Else

End {Case};

End {If};

End {While};

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры KeyManager;

Создать процесс из процедуры Lift;

- 32 -

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab7}.

Словестное описание запроса пассажира на пользование лифтом

может выглядеть следующим образом:

If Лифт свободен Then Begin

Активизировать лифт;

End {If};

Встать в очередь ожидания;

Передать управление;

Условия функционирования лифта как процесса-сервера могут

выглядеть следующим образом:

If Перевез пассажиров Then Begin

Перевод перевезенных пассажиров из очереди перевозимых в

очередь готовых процессов;

End {If};

If Нет заявок на обслуживание ни на текущем этаже ни на другом

Then Begin

Блокирование;

Передача управления;

End {If};

If Заявка с текущего этажа Then Begin

Перевод не более, чем N процессов, из очереди ждущих в

очередь перевозимых процессов;

End {If};

Учащемуся предстоит формализовать условия блокировки и

активизации лифта и пассажиров для реализации программы.

**Задача 8. МОДЕЛЬ ПАРИКМАХЕРСКОЙ**

План парикмахерской может быть представлен следующей схемой:

Зал ожидания Салон

--------------T--------¬

--> --> -->

L-------------+---------

Клиенты проходят через зал ожидания в салон, стригутся там и выходят. Если парикмахер занят обслуживанием клиента, то другие клиенты становятся в очередь в зале ожидания, но размер очереди

ожидания не должен превосходить N. Если в зале ожидания N клиентов стоят в очереди, то создается очередь на улице. Если размер очереди на улице равен M, то новый желающий постричься

обходит парикмахерскую и остается необслуженным.

Каждый выходящий из салона клиент позволяет войти к парикмахеру одному клиенту, ждущему в зале ожидания, а тот, в свою очередь, позволяет войти в зал ожидания одному клиенту, ждущему на улице.

Требуется запрограммировать задачу, написав монитор "Парикмахер", регламентирующий доступ клиентов в зал ожидания и в салон. Парикмахер представляет собой процесс-сервер.

**Методика решения**

Структура программы, реализующей задачу, может иметь

следующий вид.

Program Lab8;

Uses MultiObj;

Type

PMonitor = ^TMonitor;

TMonitor = Object

ServList, {Очередь, в которой парикмахер ждет

активизации}

HallList, {Очередь клиентов в зале ожидания}

StrList, {Очередь клиентов на улице}

ClList : TList; {Очередь, в которой клиент находится,

пока его стригут}

Nh, {Количество клиентов в очереди зала ожидания}

Ns : Integer; {Количество клиентов в очереди на улице}

Constructor Init(...);

Destructor Done; Virtual;

Function InStreet : Boolean;

Procedure InHall;

Procedure Waiting;

End {TMonitor};

{--Методы монитора разрабатываются учащимися--}

Var

Monitor : PMonitor;

Procedure Client;

Begin

- 34 -

{Движение к залу ожидания}

If Monitor^.InStreet Then Begin

{Движение к салону}

Monitor^.InHall;

{Движение из салона}

End Else Begin

{Движение по обходу парикмахерской}

End {If};

{Самоуничтожение}

End {Client};

Procedure Server;

Begin

While True Do Begin

Monitor^.Waiting;

{Случайное время обслуживания клиента}

End {While};

End {Server};

Procedure KeyManager;

Begin

While True Do Begin

If Клавиша нажата Then Begin

Чтение клавиши;

Case Клавиша Of

'Esc' : Остановить работу ядра;

'C','c' : Создать процесс из процедуры Client;

Else

End {Case};

End {If};

End {While};

End {KeyManager};

Begin

Monitor := New(PMonitor, Init);

Создать процесс из процедуры KeyManager;

Создать процесс из процедуры Server;

Начать работу ядра;

Dispose(Monitor, Done);

End {Lab8}.

Словестное описание метода монитора, моделирующего действия

парикмехера (процесса-сервера), может выглядеть следующим

- 35 -

образом:

If Обслужил клиента Then Begin

Активизация этого клиента;

End {If};

If Нет клиентов Then Begin

Блокирование в очереди парикмахера;

Передача управления;

End {If};

Перевод клиента из очереди ожидания в салоне в очередь

обслуживаемых;

Словестное описание метода TMonitor.InStreet выглядит

следующим образом:

If Количество клиентов, ждущих в очереде на улице, больше, чем М

Then Begin

InStreet := False;

End Else Begin

If Количество клиентов, ждущих на улице, больше, чем 0 ИЛИ

Количество клиентов, ждущих в зале ожидания, равно N

Then Begin

Блокирование в очереди клиентов на улице;

Передача управления;

End {If};

InStreet := True;

End {If};

Словестное описание метода TMonitor.InHall выглядит

следующим образом:

If Парикмахер свободен Then Begin

Активизация парикмахера;

End {If};

Блокирование в очереди клиентов в зале ожидания;

Активизация клиента, первого в очереди на улице;

Учащемуся необходимо реализовать представленные описательные условия в программе, а затем дополнить программу вторым процессом-парикмахером и убедиться в повышении скорости обслуживания клиентов.

**Содержание**

ОБЩАЯ ХАРАКТЕРИСТИКА ЗАДАНИЙ...............................3

Задача 1. МОДЕЛЬ ЖЕЛЕЗНОДОРОЖНОГО ПЕРЕГОНА.................5

Задача 2. МОДЕЛЬ ДОРОЖНОГО ПЕРЕКРЕСТКА ...................11

Задача 3. МОДЕЛЬ ЧИТАТЕЛЕЙ И РЕДАКТОРОВ...................15

Задача 4. МОДЕЛЬ НАЗНАЧЕНИЯ ОДНОРОДНЫХ РЕСУРСОВ...........20

Задача 5. МОДЕЛЬ ОБЕДАЮЩИХ ФИЛОСОФОВ......................23

Задача 6. МОДЕЛЬ КЛИЕНТ-СЕРВЕР............................27

Задача 7. МОДЕЛЬ ФУНКЦИОНИРОВАНИЯ ЛИФТА...................30

Задача 8. МОДЕЛЬ ПАРИКМАХЕРСКОЙ...........................32