

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

И.А.Щудро

ПРОЕКТИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ВЗАИМОДЕЙСТВУЮЩИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Рекомендовано к изданию Редакционно-издательским советом
федерального государственного бюджетного образовательного учреждения
высшего образования «Оренбургский государственный университет» в
качестве методических указаний для студентов, обучающихся по
программам высшего образования по направлению подготовки 09.03.01
Информатика и вычислительная техника

Оренбург
2016

УДК 004.451(076.5)
ББК 32.973-018.2я7
Щ 93

Рецензент – кандидат технических наук, доцент Д. В. Горбачев

Щ 93 **Щудро, И.А.**
Проектирование параллельных взаимодействующих
вычислительных процессов: методические указания к выполнению
курсовой работы по дисциплине «Операционные системы» /
И.А.Щудро; Оренбургский гос. ун-т. – Оренбург: ОГУ, 2016. – 26с.

Методические указания к выполнению курсовой работы по дисциплине «Операционные системы» предназначены для учебно-методического обеспечения подготовки бакалавров по образовательной программе уровня высшего образования: по направлению подготовки 09.03.01 Информатика и вычислительная техника.

Дисциплина «Операционные системы» входит в состав дисциплин базовой части профессионального цикла.

В методических указаниях изложен механизм синхронизации процессов, работающих с критическими секциями с использованием объектов синхронизации ядра: семафор, мьютекс, событие и критическая секция, на примере операционных систем класса Windows.

В методических указаниях содержатся практические задания на выполнение курсовой работы, требования по оформлению отчетов, варианты заданий и примеры выполнения основных этапов курсовой работы по дисциплине «Операционные системы».

УДК 004.451(076.5)
ББК 32.973-018.2я7

© Щудро И. А., 2016
© ОГУ, 2016

Содержание

Введение	4
1Порядок выполнения работ при проектировании параллельных взаимодействующих вычислительных процессов	5
2Основные теоретические положения	6
2.1Многозадачность и многопоточность	6
2.2Понятие о механизмах синхронизации в ОС Windows	6
2.3Объекты синхронизации ядра в ОС Windows	7
3Пример выполнения варианта задания	12
3.1 Постановка задачи.....	12
3.2 Алгоритм работы программы	13
3.3 Результаты работы консольного приложения.....	14
3.4 Листинг программы	15
Контрольные вопросы	18
Заключение	19
Список использованных источников	20
Приложение АТребования к выполнению структурных элементов курсовой работы.....	21
Приложение А.1Требования к оформлению отчетов курсовой работы.....	21
Приложение А.2 Задание на курсовую работу	23
Приложение Б Варианты заданий	24

Введение

Методические указания предназначены для выполнения курсовой работы по дисциплине «Операционные системы».

Целью курсовой работы является овладение методами и средствами проектирования, параллельных взаимодействующих вычислительных процессов с использованием объектов синхронизации ядра (семафор, мьютекс, событие, критическая секция) в операционных системах класса Windows.

Задачи, которые решаются при выполнении работы соответствуют рабочей программе дисциплины: мультипрограммирование; режим разделения времени; многопользовательский режим работы; понятие процесса и ядра; диспетчеризация и синхронизация процессов; понятия приоритета и очереди процессов; совместное использование памяти.

В соответствии с ФГОС ВО и ООП ВО по данному направлению подготовки, выполнение курсовой работы позволит сформировать следующие элементы компетенций:

ОПК-2 Способностью осваивать методики использования программных средств, для решения практических задач;

ПК-2 Способностью разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования.

1 Порядок выполнения работ при проектировании параллельных взаимодействующих вычислительных процессов

Для успешного выполнения работы требуется:

- ознакомление с теоретическими положениями синхронизации процессов в операционной системе Windows с помощью объектов ядра (семафор, мьютекс, событие, критическая секция);
- постановка задачи проектирования параллельных взаимодействующих вычислительных процессов в соответствии с вариантом задания;
- разработка алгоритма синхронизации процессов в контрольных точках схемы взаимодействия;
- разработка программного средства на языке высокого уровня, реализующего заданный объект синхронизации ядра операционной системы Windows в соответствии с вариантом задания;
- создание интерфейса программы или консольного приложения, для мониторинга синхронизации процессов в контрольных точках схемы взаимодействия;
- разработка руководства пользователя;
- формирование отчета о проделанной работе.

2 Основные теоретические положения

2.1 Многозадачность и многопоточность

Параллельные вычисления – способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно)[1].

С точки зрения операционной системы существуют два основных способа реализации параллельных вычислений: многозадачность и многопоточность.

Многозадачность означает, что каждый вычислительный процесс может быть реализован в виде процесса операционной системы.

Многопоточность – это свойство операционной системы или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся по независимому пути исполнения, одновременно с другими потоками.

Многопоточность не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Многопоточность применяется во многих программах, поскольку при выполнении некоторых задач разделение процесса на потоки помогает достичь более эффективного использования ресурсов вычислительной машины.

При создании многопоточных приложений необходимо контролировать взаимодействие отдельных потоков. Большинство ошибок при работе с потоками возникает из-за того, что во время работы приложения различные потоки пытаются обратиться к одним и тем же данным. Для предотвращения подобной ситуации в ОС Windows (и в других операционных системах) существуют средства синхронизации, которые позволяют контролировать доступ к разделяемым ресурсам [1, 2].

2.2 Понятие о механизмах синхронизации в ОС Windows

Для программ, использующих несколько потоков или процессов, необходимо, чтобы все они выполняли возложенные на них функции в нужной последовательности. В среде Windows 9x для этой цели предлагается

использовать несколько механизмов, обеспечивающих слаженную работу потоков. Эти механизмы называют механизмами синхронизации [3].

Предположим, разрабатывается программа, в которой параллельно работают два потока. Каждый поток обращается к одной разделяемой глобальной переменной. Один поток при каждом обращении к этой переменной выполняет её инкремент, а второй – декремент. При одновременной асинхронной работе потоков неизбежно возникает такая ситуация:

- первый поток прочитал значение глобальной переменной в локальную;
- ОС прерывает его, так как закончился выделенный ему квант времени процессора, и передаёт управление второму потоку;
- второй поток также считал значение глобальной переменной в локальную, декрементировал её и записал новое значение обратно;
- ОС вновь передаёт управление первому потоку, тот, ничего не зная о действиях второго потока, инкрементирует свою локальную переменную и записывает её значение в глобальную.

Очевидно, что изменения, внесённые вторым потоком, будут утеряны.

Для исключения подобных ситуаций необходимо разделить во времени использование совместных данных. В таких случаях используются механизмы синхронизации, которые обеспечивают корректную работу нескольких потоков.

2.3 Объекты синхронизации ядра ОС Windows

Синхронизация может быть выполнена средствами прикладной программы и средствами ОС. Более универсальными и эффективными являются средства ОС. В ОС Windows предусмотрены, в частности, следующие основные объекты синхронизации:

- критические секции;
- мьютексы;
- семафоры;
- события.

Объектами синхронизации называют специальные системные объекты, которые видны всем потокам, в том числе и тем, которые

принадлежат разным процессам. Объекты могут находиться в двух состояниях: сигнальном (свободном, signaled) и занятом (несигнальном, unsignaled) [2].

Смысл «сигнальное состояние» (signaledstate) зависит от контекста. Например, поток переходит в сигнальное состояние, когда он заканчивается. Файл переходит в сигнальное состояние, когда операция ввода-вывода для этого файла завершается. Семафор находится в сигнальном состоянии, если его счетчик больше нуля, мьютекс – если он освобожден. Событие находится в сигнальном состоянии, если его флаг установлен (событие произошло).

Для остальных объектов сигнальное состояние устанавливается в результате выполнения специальных системных вызовов. Приостановка и активизация потоков осуществляются в зависимости от состояния синхронизирующих объектов ОС.

Критическая секция (CriticalSection) – это объект, который принадлежит процессу, а не ядру. А значит, не может синхронизировать потоки из разных процессов [2].

Существует так же функции инициализации (создания) и удаления, вхождения и выхода из критической секции:

- создание – InitializeCriticalSection(...);
- удаление – DeleteCriticalSection(...);
- вход – EnterCriticalSection(...);
- выход – LeaveCriticalSection(...).

Ограничения: поскольку это не объект ядра, то он не виден другим процессам, то есть можно защищать только потоки своего процесса.

Рассмотрим прием использования блокирующих переменных на примере критических секций (рисунок 1).

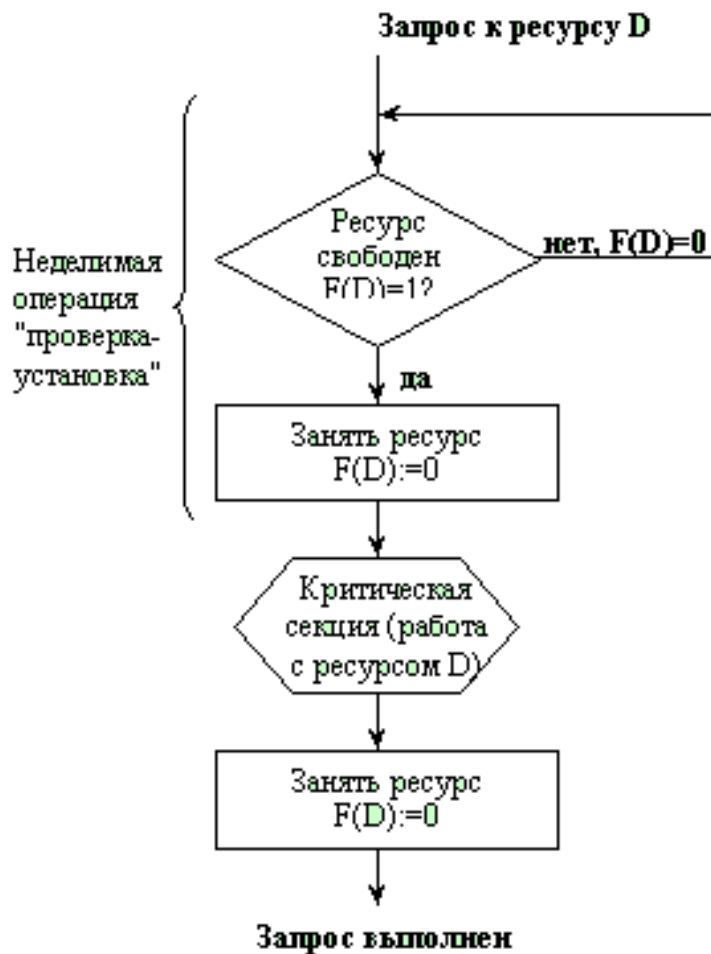


Рисунок 1 – Реализация критических секций с использованием блокирующих переменных

С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом), и значение 0, если ресурс занят. На рисунке 1 показан фрагмент алгоритма процесса, использующего для реализации взаимного исключения доступа к разделяемому ресурсу D блокирующую переменную $F(D)$. Перед входом в критическую секцию процесс проверяет, свободен ли ресурс D. Если он занят, то проверка циклически повторяется, если свободен, то значение переменной $F(D)$ устанавливается в 0, и процесс входит в критическую секцию. После того, как процесс выполнит все действия с разделяемым ресурсом D, значение переменной $F(D)$ снова устанавливается равным 1.

Критический раздел анализирует значение специальной переменной процесса, которая используется как флаг, предотвращающий исполнение

некоторого участка кода несколькими потоками одновременно. Среди синхронизирующих объектов критические разделы наиболее просты.

Семафор (semaphore). Объект ядра “семафор”, используются для учёта ресурсов и служат для ограничения одновременного доступа к ресурсу нескольких потоков [2]. Используя семафор, можно организовать работу программы таким образом, что к ресурсу одновременно смогут получить доступ несколько потоков, однако количество этих потоков будет ограничено. Создавая семафор, указывается максимальное количество потоков, которые одновременно смогут работать с ресурсом. Каждый раз, когда программа обращается к семафору, значение счетчика ресурсов семафора уменьшается на единицу. Когда значение счетчика ресурсов становится равным нулю, семафор недоступен.

Функции:

- создание CreateSemaphore;
- открытие OpenSemaphore;
- занять WaitForSingleObject;
- освобождение ReleaseSemaphore.

Мьютекс (mutex – mutableexclude) – это объект ядра, у него есть имя, а значит с их помощью можно синхронизировать доступ к общим данным со стороны нескольких процессов, точнее, со стороны потоков разных процессов [2]. Ни один другой поток не может завладеть мьютексом, который уже принадлежит одному из потоков. Если мьютекс защищает какие-то совместно используемые данные, он сможет выполнить свою функцию только в случае, если перед обращением к этим данным каждый из потоков будет проверять состояние этого мьютекса. Windows расценивает мьютекс как объект общего доступа, который можно перевести в сигнальное состояние или сбросить. Сигнальное состояние мьютекса говорит о том, что он занят. Потоки должны самостоятельно анализировать текущее состояние мьютексов. Если требуется, чтобы к мьютексу могли обратиться потоки других процессов, ему надо присвоить имя.

Функции:

- CreateMutex(имя) – создание;
- hnd=OpenMutex(имя) – открытие;
- WaitForSingleObject(hnd) – ожидание и занятие;
- ReleaseMutex(hnd) – освобождение;

- CloseHandle(hnd) – закрытие.

Его можно использовать в защите от повторного запуска программ.

Событие (event). События обычно просто оповещают об окончании какой-либо операции, они также являются объектами ядра. Можно не просто явным образом освободить, но также есть операция установки события. События могут быть мануальными (manual) и единичными (single) [2].

Единичное событие (singleevent) – это скорее общий флаг. Событие находится в сигнальном состоянии, если его установил какой-нибудь поток. Если для работы программы требуется, чтобы в случае возникновения события на него реагировал только один из потоков, в то время как все остальные потоки продолжали ждать, то используют единичное событие.

Мануальное событие (manualevent) — это не просто общий флаг для нескольких потоков. Оно выполняет несколько более сложные функции. Любой поток может установить это событие или сбросить (очистить) его. Если событие установлено, оно останется в этом состоянии сколь угодно долгое время, вне зависимости от того, сколько потоков ожидает установки этого события. Когда все потоки, ожидающие этого события, получат сообщение о том, что событие произошло, оно автоматически сбросится.

Функции: SetEvent, ClearEvent, WaitForEvent.

Типы событий:

- событие с автоматическим сбросом: WaitForSingleEvent.
- событие с ручным сбросом (manual), тогда событие необходимо сбрасывать: ReleaseEvent.

Некоторые теоретики выделяют ещё один объект синхронизации:

WaitableTimer – объект ядра ОС, который самостоятельно переходит в свободное состояние через заданный интервал времени (будильник).

3 Пример выполнения варианта задания

3.1 Постановка задачи

Реализовать синхронизацию процессов в трех контрольных точках (рисунок 2) с помощью объекта синхронизации ядра – «семафор» (по варианту задания). Все процессы в контрольных точках запускаются, процессом, пришедшим в контрольную точку первым. Разработать алгоритм синхронизации взаимодействующих процессов в контрольных точках и приложение на языке высокого уровня. Создать интерфейс программы или консольное приложение, для контроля синхронизации процессов в контрольных точках.

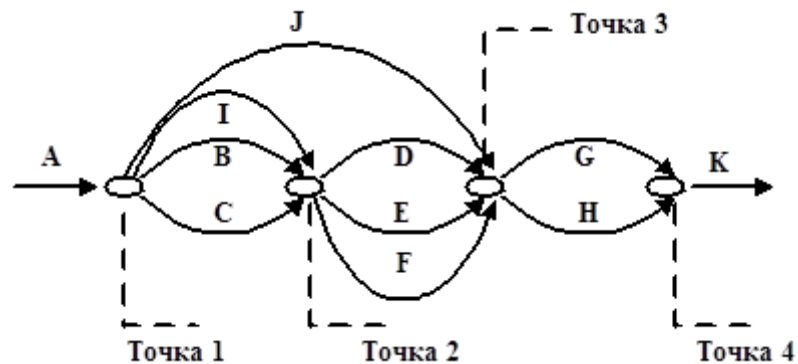


Рисунок 2 – Схема взаимодействия параллельно выполняющихся задач

3.2 Алгоритм работы программы

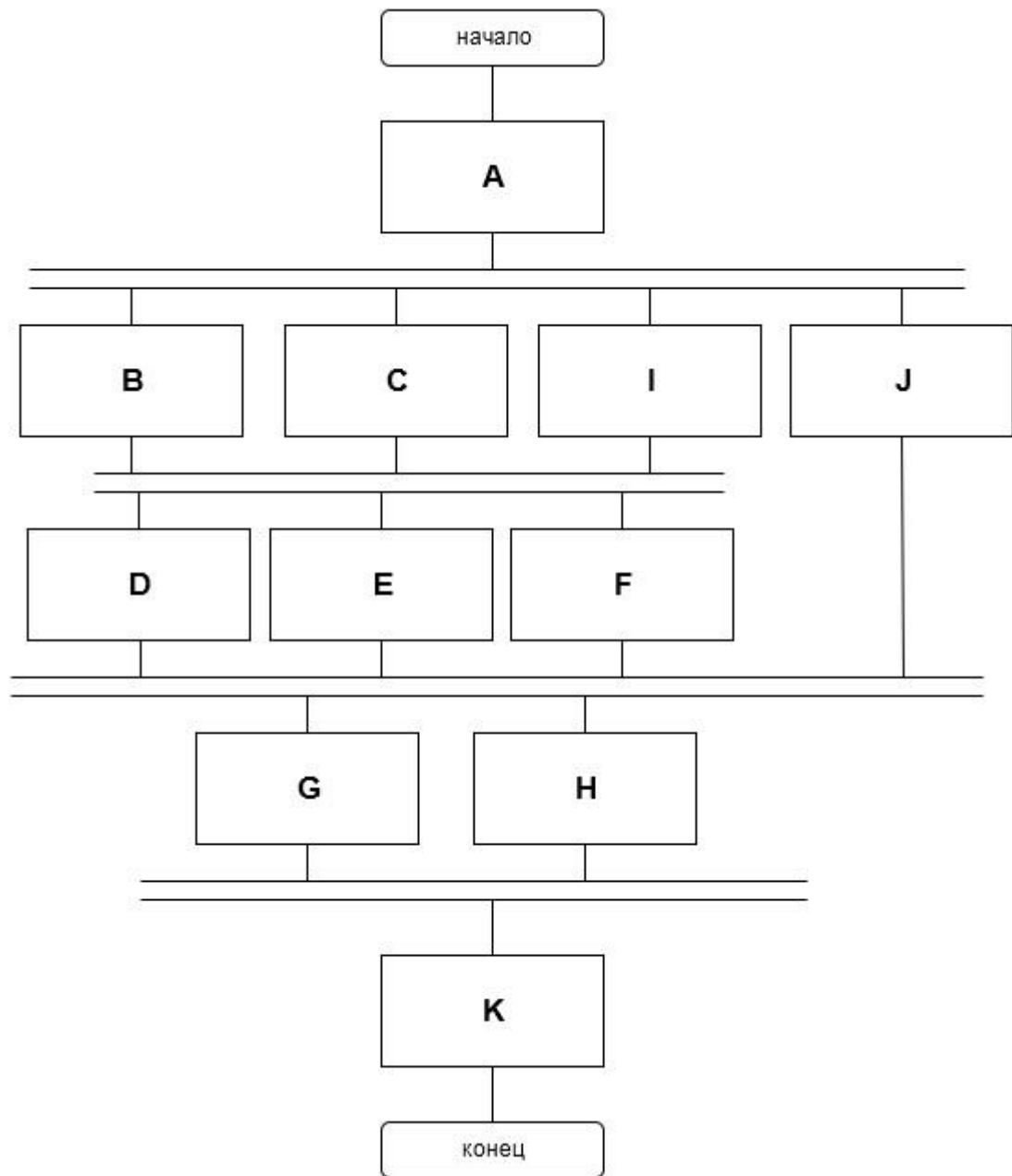


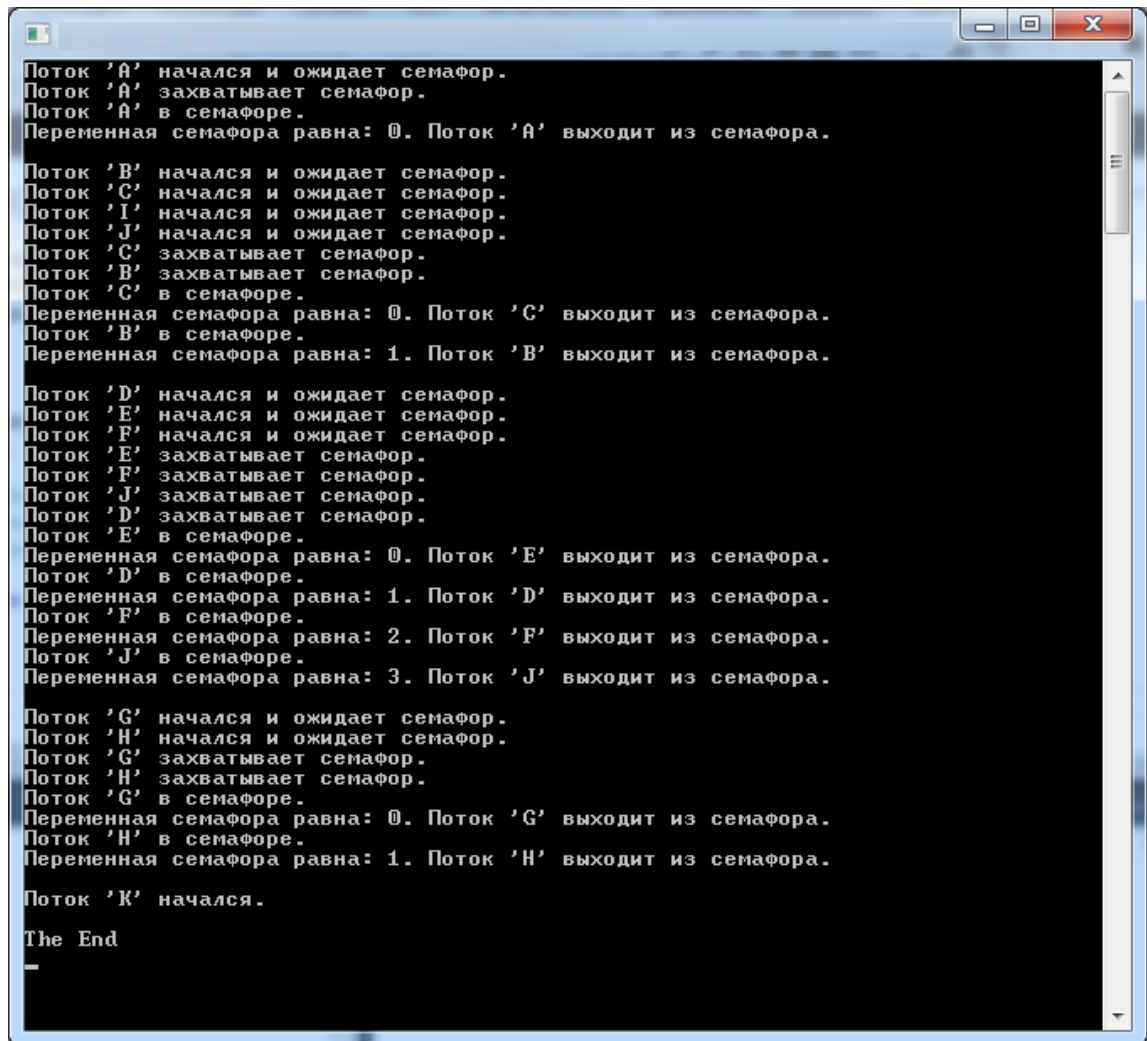
Рисунок 3 – Блок – схема алгоритма работы программы

В соответствии с алгоритмом: Процесс А запускает процессы В, С, I, J, процесс, выполняющийся первым из процессов В, С, I, запускает процессы D, E, F, а процесс J конкурирует в запуске процессов G, H с первым, выполнившимся из процессов D, E, F и, в заключении, первый, выполнившийся из процессов G, H, запускает процесс К, который завершает программу.

Синхронизация (блокировка) процессов происходит в точках 2, 3, 4.

3.3 Результат работы консольного приложения

На рисунке 4 представлен результат работы консольного приложения. Информация о запуске и выполнении потоков в контрольных точках выводится на экран в текстовом режиме.



```
Поток 'А' начался и ожидает семафор.
Поток 'А' захватывает семафор.
Поток 'А' в семафоре.
Переменная семафора равна: 0. Поток 'А' выходит из семафора.

Поток 'В' начался и ожидает семафор.
Поток 'С' начался и ожидает семафор.
Поток 'I' начался и ожидает семафор.
Поток 'J' начался и ожидает семафор.
Поток 'С' захватывает семафор.
Поток 'В' захватывает семафор.
Поток 'С' в семафоре.
Переменная семафора равна: 0. Поток 'С' выходит из семафора.
Поток 'В' в семафоре.
Переменная семафора равна: 1. Поток 'В' выходит из семафора.

Поток 'D' начался и ожидает семафор.
Поток 'Е' начался и ожидает семафор.
Поток 'F' начался и ожидает семафор.
Поток 'Е' захватывает семафор.
Поток 'F' захватывает семафор.
Поток 'J' захватывает семафор.
Поток 'D' захватывает семафор.
Поток 'Е' в семафоре.
Переменная семафора равна: 0. Поток 'Е' выходит из семафора.
Поток 'D' в семафоре.
Переменная семафора равна: 1. Поток 'D' выходит из семафора.
Поток 'F' в семафоре.
Переменная семафора равна: 2. Поток 'F' выходит из семафора.
Поток 'J' в семафоре.
Переменная семафора равна: 3. Поток 'J' выходит из семафора.

Поток 'G' начался и ожидает семафор.
Поток 'H' начался и ожидает семафор.
Поток 'G' захватывает семафор.
Поток 'H' захватывает семафор.
Поток 'G' в семафоре.
Переменная семафора равна: 0. Поток 'G' выходит из семафора.
Поток 'H' в семафоре.
Переменная семафора равна: 1. Поток 'H' выходит из семафора.

Поток 'K' начался.

The End
-
```

Рисунок 4 – Результат работы консольного приложения

3.4 Листинг программы

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
    static class Program
    {
        private static Semaphore _semaphore1;
        private static int _padding;
        private static int _semaphoreI;
        public static void Main()
        {
            Thread a = new Thread(Worker);
            _semaphore1 = new Semaphore(0, 1);
            a.Start("A");
            Thread.Sleep(500);
            _semaphore1.Release(1);
            Thread.Sleep(3000);
            _semaphore1.Close();
            a.Abort();
            Console.WriteLine();
            _semaphoreI = 0;
            Thread b = new Thread(Worker);
            Thread c = new Thread(Worker);
            Thread i = new Thread(Worker);
            Thread j = new Thread(Worker);
            _semaphore1 = new Semaphore(0, 3);
            b.Start("B");
            c.Start("C");
            i.Start("I");
            j.Start("J");
```

```

Thread.Sleep(500);
j.Suspend();
    _semaphore1.Release(3);
Thread.Sleep(3000);
    _semaphore1.Close();
b.Abort();
c.Abort();
i.Abort();
Console.WriteLine();
    _semaphoreI = 0;
    Thread d = new Thread(Worker);
    Thread e = new Thread(Worker);
    Thread f = new Thread(Worker);
    _semaphore1 = new Semaphore(0, 5);
d.Start("D");
e.Start("E");
    f.Start("F");
Thread.Sleep(500);
j.Resume();
    _semaphore1.Release(4);
Thread.Sleep(3000);
d.Abort();
e.Abort();
f.Abort();
j.Abort();
    _semaphore1.Close();
Console.WriteLine();
    _semaphoreI = 0;
    Thread g = new Thread(Worker);
    Thread h = new Thread(Worker);
    _semaphore1 = new Semaphore(0, 3);

```



```

g.Start("G");
h.Start("H");
Thread.Sleep(500);
    _semaphore1.Release(2);
Thread.Sleep(3000);
g.Abort();
h.Abort();
    _semaphore1.Close();
Console.WriteLine();
    _semaphoreI = 0;
    Thread k = new Thread(Worker);
Console.WriteLine("Поток 'К' начался.");
k.Abort();
Console.WriteLine("\nThe End");
Console.ReadLine();
    }
private static void Worker(object num)
    {
Console.WriteLine("Поток {0} начался и ожидает семафор.", num);
    _semaphore1.WaitOne();
int padding = Interlocked.Add(ref _padding, 100);
Console.WriteLine("Поток {0} захватывает семафор.", num);
Thread.Sleep(1000 + padding);
Console.WriteLine("Поток {0} в семафоре.", num);
    Console.WriteLine("Переменная семафора равна: {0}. Поток {1} выходит из
семафора.", _semaphoreI++, num);
    }
    }
}

```

Контрольные вопросы

- 1 Какие последовательные вычислительные процессы называются параллельными и почему?
- 2 Какие параллельные процессы называются независимыми, какие взаимодействующими?
- 3 Изложите алгоритм Деккера, позволяющий решить проблему взаимного исключения путем использования одной только блокировки памяти.
- 4 Изложите алгоритм «проверка и установка» для реализации синхронизации процессов.
- 5 Приведите временную диаграмму работы двоичного семафора Дейкстры.
- 6 Как реализовать семафорные примитивы для мультипроцессорной системы?
- 7 Изложите алгоритм решения задачи «поставщик-потребитель» на основе использования семафоров Дейкстры.
- 8 Изложите алгоритм решения задачи «обедающие философы» на основе мьютексов.
- 9 Сравните метод критической секции и использование семафоров.
- 10 Сравните метод критической секции и метод взаимных исключений mutex.

Заключение

Методические указания предназначены для выполнения курсовой работы по разработке приложения для проектирования параллельных взаимодействующих вычислительных процессов.

В данные методические указания был включен краткий обзор следующих объектов синхронизации, реализуемых в операционных системах класса Windows: критические секции; мьютексы; семафоры; события.

Целью данных методических указаний является привитие практических навыков студентам в разработке приложений на языках высокого уровня, реализующих синхронизацию процессов в контрольных точках схемы с использованием заданных объектов синхронизации.

В методические указания включен пример решения задачи с использованием семафоров на языке программирования C+.

В приложениях содержатся: варианты заданий и правила оформления отчета по курсовой работе.

Список использованных источников

1 Таненбаум, Э. Современные операционные системы / Э. Таненбаум. – 3-е изд. – СПб.: Питер, 2010. -1120 с.: ил.

2 Гордеев, А. В. Операционные системы: учебник для вузов. 2-е изд. [Электронный ресурс] / А.В. Гордеев. – СПб. : Питер, 2010.

3 Электронный ресурс: [https://msdn.microsoft.com/ru-ru/library/system.threading.mutex\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/system.threading.mutex(v=vs.110).aspx)

4 Шишкин, В. В. Проектирование многозадачных операционных систем реального времени для авиационных приборных комплексов / В. В. Шишкин, В. В. Улыбин // Датчики и системы, 2007. - N 11. - С. 11-14. - ил.- Библиогр.: с. 14 (9 назв.).

5 Дейтел, Х. М. Операционные системы Operating Systems / Х. М. Дейтел, П. Дж. Дейтел, Д. Р. Чофнес; пер. с англ. под ред. С. М. Молявко. – 3-е изд. – М. : Бином-Пресс, 2009. – 704 с.: ил. - ISBN 978-5-9518-0291-0.

6 Кручинин, А. Ю. Операционные системы [Электронный ресурс] / А. Ю. Кручинин. – Оренбург : ГОУ ОГУ, 2009.

7 Илюшечкин, В. М. Операционные системы [Электронный ресурс] / В.М. Илюшечкин. – М. : БИНОМ. ЛЗ, 2012.

8 Назаров, С. В. Современные операционные системы: учебное пособие [Электронный ресурс] / С.В. Назаров, А.И. Широков. – Интернет-Университет Информационных Технологий, 2011.

9 Вирт, Н. Разработка операционной системы и компилятора. Проект Оберон = Project Oberon The Design of an Operating System and Compiler [Электронный ресурс] / Н. Вирт, Ю. Гуткнехт. – М. : ДМК Пресс, 2012.

Приложение А

Требования к выполнению структурных элементов курсовой работы

А.1 Требования к оформлению отчетов курсовой работы

Оформление отчета курсовой работы выполняется в соответствии с требованиями Стандарта организации СТО 02069024.101-2015.

Курсовая работа должна содержать текстовую и графическую часть.

Текстовая часть курсовой работы содержит следующие структурные элементы:

- титульный лист;
- задание;
- аннотация;
- содержание;
- введение;
- основная часть;
- список использованных источников;
- приложения.

Титульный лист является первым листом курсовой работы. Все надписи выполняют черной тушью, чернилами или пастой черного цвета. На титульном листе указывают классификационный код.

Бланк задания следует помещать после титульного листа. Задание должно содержать исходные данные, объем и срок выполнения курсовой работы с подписями руководителя и исполнителя.

Аннотация является третьим листом курсовой работы. Аннотация – это краткая характеристика курсовой работы с точки зрения содержания, назначения и результатов работы.

В элементе «Введение» указывают цель работы, область применения разрабатываемой проблемы, ее научное и практическое значение, решаемые инженерные задачи для достижения цели.

Во введении следует:

- обосновать актуальность темы для изучения операционных систем;
- произвести постановку цели исследования;
- сформулировать задачи работы;
- перечислить методы и средства, с помощью которых будут решаться поставленные инженерные задачи исследований;
- кратко изложить ожидаемые результаты.

Изложение текста основной части, оформление иллюстраций, построение таблиц, список использованных источников, приложения должны соответствовать требованиям, указанным в разделах 7 и 8 СТО 02069024.101-2015.

Заключение должно содержать краткие выводы по результатам выполненной работы, оценку полноты решения поставленных задач, рекомендации по конкретному использованию результатов работы.

Графическая часть включает в себя схему алгоритма программы многопоточного приложения, рисунки, отражающие этапы выполнения программы, результат работы консольного приложения (диалоговые окна программного продукта) и листинг программы.

А.2 Задание на курсовую работу

Исходные данные: Вариант № _____

Проектирование параллельных взаимодействующих вычислительных процессов
(наименование предметной области)

Перечень подлежащих разработке вопросов:

- 1) Разработка алгоритма реализации многопоточного приложения на основе заданного объекта синхронизации;
- 2) Разработка многопоточного приложения средства языка высокого уровня;
- 3) Создание интерфейса программы для моделирования потоков в приложении.

Перечень графического материала:

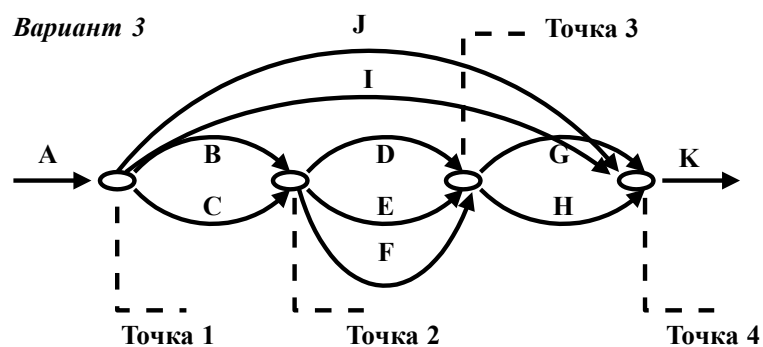
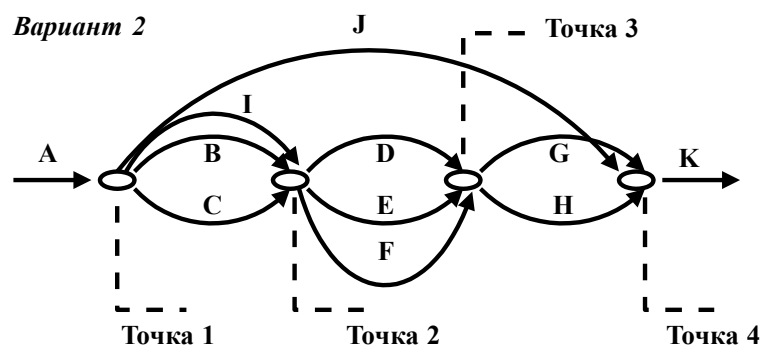
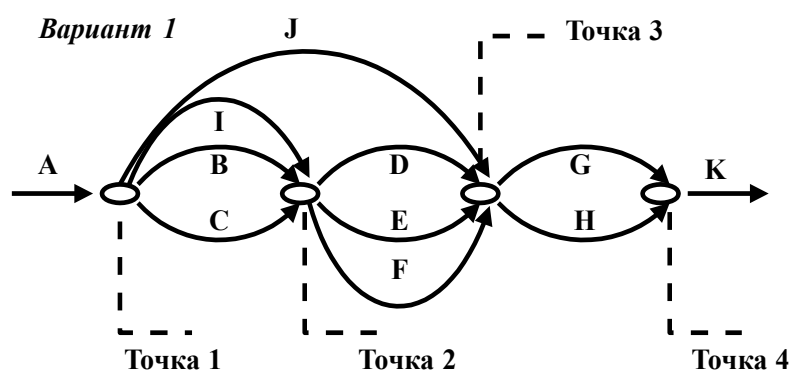
Схема алгоритма программы многопоточного приложения, рисунки, отражающие этапы выполнения программы и листинг программы.

Приложение Б

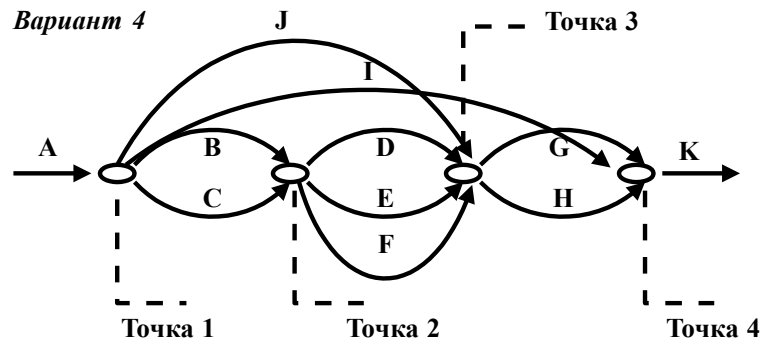
Варианты заданий

Таблица Б.1 Варианты заданий

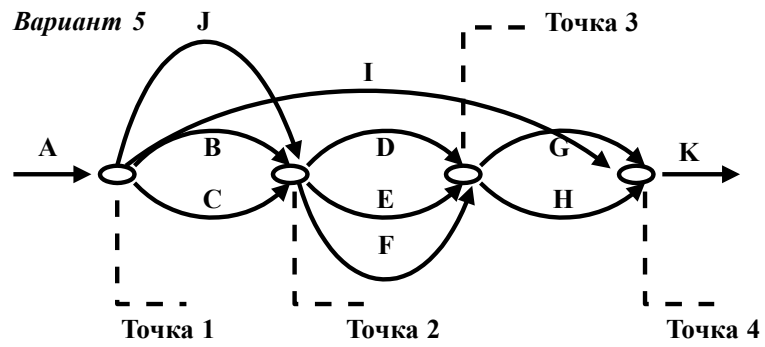
Объект синхронизации	Вариант
1. Мьютекс	1-9
2. Семафор	1-9
3. Событие	1-9
4. Критическая секция	1-9



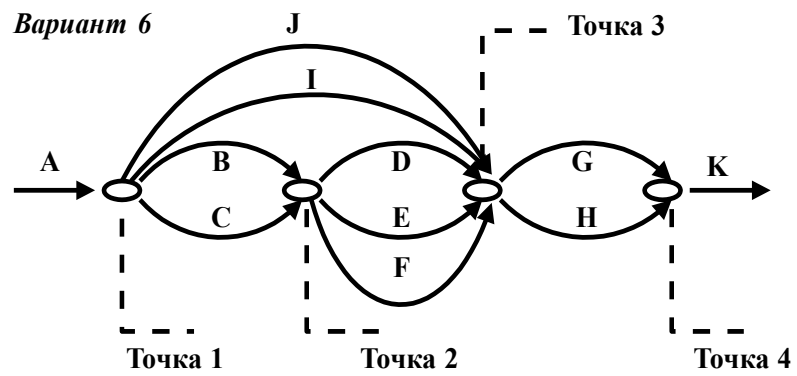
Вариант 4



Вариант 5



Вариант 6



Вариант 7

